General Framework for Metric Optimization Problems with Delay or with Deadlines

Yossi Azar

azar@tau.ac.il

Tel Aviv University

Noam Touitou

noamtouitou@mail.tau.ac.il

Tel Aviv University

Abstract

In this paper, we present a framework used to construct and analyze algorithms for online optimization problems with deadlines or with delay over a metric space. Using this framework, we present algorithms for several different problems. We present an $O(D^2)$-competitive deterministic algorithm for online multilevel aggregation with delay on a tree of depth D, an exponential improvement over the $O(D^4 2^D)$-competitive algorithm of Bienkowski et al. (ESA '16). We also present an $O(\log^2 n)$-competitive randomized algorithm for online service with delay over any general metric space of n points, improving upon the $O(\log^4 n)$-competitive algorithm by Azar et al. (STOC '17).

In addition, we present the problem of online facility location with deadlines. In this problem, requests arrive over time in a metric space, and need to be served until their deadlines by facilities that are opened momentarily for some cost. We also consider the problem of facility location with delay, in which the deadlines are replaced with arbitrary delay functions. For those problems, we present $O(\log^2 n)$-competitive algorithms, with n the number of points in the metric space.

The algorithmic framework we present includes techniques for the design of algorithms as well as techniques for their analysis.

1 Introduction

Recently in the field of online algorithms, there has been an increasing interest in online problems involving deadlines or delay. In such problems, requests of some form arrive over time, requiring service. In problems with deadlines, each request is equipped with a deadline, by which the request must be served. In problems with delay, this hard constraint is replaced with a more general constraint. In those problems, each request is equipped with a delay function, such that an algorithm accumulates delay cost while the request remains pending. This provides an incentive for the algorithm to serve the request as soon as possible. Deadlines are a special case of delay, as deadlines can be approximated arbitrarily well by delay functions.

The mechanism of adding delay or deadlines can be used to convert a problem over a sequence into a problem over time. For example, a problem in which an arriving request must immediately be served by the algorithm can be converted into a problem with deadlines, providing more flexibility to a possible solution. This conversion often creates interesting problems over time from problems that are trivial over a sequence, as well as enables much better solutions (i.e. lower cost).

A case of special interest is the case of such problems over a metric space. A notable example, which we consider in this paper, is the **online multilevel aggregation problem**. In this problem, the requests arrive on the leaves of a tree. At any time, the algorithm may choose to transmit any subtree that includes the root of the tree, at a cost which is the sum of the weights of the subtree's edges. Pending requests on any leaves contained in the transmitted subtree are served by the transmission. The general delay case of this problem was first considered by Bienkowski et al. [7], who gave a $O(D^4 2^D)$-competitive algorithm for the problem, with D the depth of the tree. Buchbinder et al. [13] then showed a $O(D)$-competitive deterministic algorithm for the deadline case. In this paper, we improve the result of [7] for general delay exponentially.

Another notable example is the **online service with delay** problem, presented in [5]. In this problem, requests arrive on points in a metric space, accumulating delay while pending. There is a single server in the metric space, which can be moved from one point to another at a cost which is the distance between the two points. Moving a server to a point at which there exists a pending request serves that request. In [5], an $O(\log^4 n)$-competitive randomized algorithm is given for the problem, where n is the number of points in the metric space. This algorithm encompasses a random embedding to an hierarchical well-separated tree (HST) of depth $h = O(\log n)$, and an $O(h^3)$-competitive deterministic algorithm for online service with delay on HSTs. In this paper, we also improve this result to $O(\log^2 n)$ competitiveness.

In addition, we also present the problem of **online facility location with deadlines**. In this problem, requests arrive over time on points of a metric space, each equipped with a deadline. The algorithm can open a facility at any point of the metric space, at some fixed cost. Immediately upon opening a facility, the algorithm may connect any number of pending requests to that facility, serving these requests. Connecting a request to a facility incurs a connection cost which is the distance between the location of the request and the location of the facility. In contrast to previous considerations of online facility location, in our problem the facility is only opened momentarily, disappearing immediately after connecting the requests. We also consider the problem of **online facility location with delay**, in which the deadlines are replaced with arbitrary delay functions. For those problems we present $O(\log^2 n)$-competitive algorithms, with n the number of points in the metric space.

The problem of facility location is a widely researched classic problem. The modification of ephemeral facilities is highly motivated, as it describes an option of renting facilities instead of buying them. As renting shared resources is a growing trend (e.g. in cloud computing), this problem captures many practical scenarios.

Our paper presents algorithms for online facility location with deadlines, online facility location with

delay, online multilevel aggregation with delay and online service with delay. These algorithms all share a common framework that we develop. The framework includes techniques for both the design of the algorithms and their analysis. We believe the flexibility and generality of this framework would enable designing and analyzing algorithms for additional problems with deadlines or with delay.

Our Results

In this paper, we present a framework used to construct and analyze online optimization problems with deadlines or with delay over a metric space. Using this framework, we present the following algorithms.

1. An $O(D^2)$-competitive deterministic algorithm for online multilevel aggregation with delay on a tree of depth D. This is an exponential improvement over the $O(D^4 2^D)$-competitive algorithm in [7].

2. An $O(\log^2 n)$-competitive randomized algorithm for online service with delay over a metric space with n points. This improves upon the $O(\log^4 n)$-competitive randomized algorithm in [5].

3. An $O(\log^2 n)$-competitive randomized algorithm for online facility location with deadlines over a metric space with n points.

4. An $O(\log^2 n)$-competitive randomized algorithm for online facility location with delay over a metric space with n points.

Our algorithms all share a common framework, which we present. The framework provides general structure to both the algorithm and its analysis.

Such an improvement for the online multilevel aggregation problem is only known for the special case of deadlines, as given in [13].

The algorithms for online facility location with deadlines and with delay can be easily extended to the case in which the cost of opening a facility is different for each point in the metric space. This changes the competitiveness of the algorithms to $O(\log^2 \Delta + \log \Delta \log n)$, where Δ is the aspect ratio of the metric space.

Our Techniques

All of our algorithms are based on corresponding competitive algorithms for HSTs. The randomized algorithms for general metric spaces are obtained through randomized HST embedding. The $O(D^2)$-competitive deterministic algorithm for online multilevel aggregation with delay on a tree is based on decomposing the tree into a forest of HSTs. This decomposition is similar to that used in [13] for the case of deadlines.

The framework – algorithm design. In designing algorithms for the problems over HSTs, we use a certain framework. In an algorithm designed using the framework, there is a counter for every node (in the case of facility location) or every edge (in the case of online multilevel aggregation and service with delay). The sizes of the counters vary between the problems considered. When the counter for a tree element (either node or edge) is full, the algorithm resets the counter and explores the subtree rooted at that element.

The process of exploration serves some of the pending requests at that subtree, while simultaneously charging counters of descendant tree elements. The exploration takes place in a DFS fashion – if at any time during the exploration of an element the counter of a descendant element is full, the algorithm immediately

suspends the exploration of the current element in favor of its descendant. The exploration of the original element resumes only when the exploration of the descendant is complete.

The exploration of specific element has a certain budget, used to charge counters of descendants. This budget is equal to the size of the counter of the element being explored. The algorithm adheres to the budget very strictly, spending exactly the amount specified. This is a crucial part of the framework, as exceeding the budget (or falling below budget) by even a constant factor would yield a competitiveness which is exponential in the depth of the tree.

This DFS exploration method is very different from previous algorithms, and enables us to get our improved results. The counter-based structure of our algorithms enables this DFS exploration while controlling the budget. Using the counter-based structure is, in turn, enabled by the techniques that we present in our framework's analysis.

The framework – analysis. The analysis of the algorithms of this framework require constructing a *preflow* - a weighted directed graph which is similar to a flow network, but in which we allow nonnegative excesses at nodes (i.e. more incoming than outgoing). We refer to nodes of the preflow as *charging nodes*. We construct a source charging node, from which the output is proportional to the cost of the optimum, and then use the preflow to propagate this output throughout the preflow graph. Since the excesses are nonnegative, the sum of the excesses of any subset of charging nodes is a lower bound of the total output from the source charging node, and thus also some lower bound on the cost of the optimum. We construct the preflow in a manner that allows us to locate such a subset of high-excess charging nodes, thus providing the required lower bound.

In the preflows we construct, each tree element (node or edge) is converted to multiple charging nodes, each corresponding to an exploration of that tree element. The possible edges between charging nodes in the preflow depend on the structure of the tree and the operation of our algorithm. Of those possible edges, we describe a procedure that chooses the actual edges of the preflow. This procedure depends on the optimal solution. Though the original metric space is a tree, the multiple copies of each tree element cause the resulting preflow to be a general directed graph.

The goal of the preflow creation procedure is to propagate the optimum's costs to some "top layer" of charging nodes. This top layer usually consists of nodes corresponding to explorations of the root tree element, though in the case of online service with delay the definition is different. The charging nodes of that top layer are then chosen to lower bound the optimum, as described.

The preflow creation procedure involves creating colors at the "top" layer of the charging nodes. These colors are then propagated, through some set of propagation rules, to nodes in lower layers. Each color corresponds to the charging node in which it originated, with the exception of two colors – the empty color, and an additional "special" color. As nodes are colored, the possible edges that contain them become actual edges of the preflow.

We now discuss the techniques used in each of the problems in this paper.

Online facility location with deadlines. We use our framework in constructing an algorithm for this problem over an HST. The algorithm maintains a counter on each node (other than the root node), such that each counter is of size f, where f is the cost of opening a facility. Whenever a counter is full, it resets and triggers an exploration of that node. Whenever the deadline of a pending request expires, the algorithm starts an exploration of the root node.

In the exploration of a node u, the algorithm opens a facility at u, and considers pending requests in the node's subtree according to increasing deadline. For each request considered, it raises the counter of the child node on the path to the request by the cost of connecting that request to u. If the counter of the child is full, an exploration of that child is called recursively, which would surely serve the considered request. Otherwise, the algorithm connects that request to u. As per the framework, the budget of u's exploration

for raising these counters is exactly f.

Online facility location with delay. The algorithm for this problem is an extension of the deadline case. An exploration of the root node is now triggered upon a set of requests which is *critical*, i.e. has accumulated large delay.

The significant difference between the delay case and the deadline case is in the exploration itself. In the deadline case, the exploration of a node u spends its budget attempting to "push back" the next occurrence of a single event (i.e. the earliest deadline of a pending request in the subtree rooted at u). In the delay case, there are two events to consider. The first event is a single request with a delay large enough to justify connection to u. The second event is a "coalition" of many tightly-grouped requests with small individual delay, but large overall delay. This coalition does not justify connection to u, but does merit opening a facility near the coalition.

Online multilevel aggregation with delay. In our algorithm for this problem over HSTs, each edge has a counter. The size of the counter is the weight of edge. This is in contrast to our algorithms for the facility location problems, in which all counters were of the same size. We assume, without loss of generality, that there exists a single edge exiting the root node, called the root edge. As in the facility location case, an exploration of the root edge is triggered when the delay of a set of requests becomes high.

In our algorithm, exploring an edge means adding descendant edges to the transmitted subtree. The explored edge again has a budget equal to its weight. The exploration repeatedly chooses the earliest point in time in which the delay of a set of requests exceeds the cost of expanding the transmission to include these requests. It then raises the counter of the descendant edge in the direction of that request set. Note the contrast with the algorithms for facility location – the explored edge is allowed to raise the counters of its descendant edges, and not just of its immediate children.

While the analysis for our facility location problems required constructing a single preflow to get a lower bound on the cost of the optimum, the analysis for online multilevel aggregation with delay requires constructing an additional preflow to get an upper bound on the cost of the algorithm.

Online service with delay. Our algorithm for this problem uses the exploration method of the algorithm for online multilevel aggregation with delay. However, the tree to be explored is not the entire tree, but rather some subtree according to the location of the server. The concepts of relative trees and major edges are defined in a similar way to [5]. We also use a potential function based on the distance of the algorithm's server from the optimum's server. As the algorithm consists (mainly) of making calls to the multilevel aggregation exploration, the analysis divides these explorations to those for which the optimum can be charged (using similar arguments to the analysis of the multilevel aggregation algorithm), and explorations for which the costs are covered by the potential function.

Related Work

The online multilevel aggregation problem generalizes a range of studied problems, such as the TCP acknowledgment problem [14, 17, 22] and the joint replenishment problem [8, 12, 15]. For both the deadline and delay variants of online multilevel aggregation, the best known lower bounds are only constant [8]. Bienkowski et al. [7] were the first to present an algorithm for the online multilevel aggregation problem with arbitrary delay functions, which is $O(D^4 2^D)$-competitive. Buchbinder et al. [13] presented an $O(D)$-competitive algorithm for the special case of deadlines.

The problem of online service with delay was presented in [5], along with the $O(\log^4 n)$-competitive randomized algorithm for a general metric space of n points. The problem has also been studied over specific metric spaces, such as uniform metric and line metric, in which improved results can be achieved [5, 11].

Another metric optimization problem with delay is the problem of matching with delay [2, 19, 18, 4, 9, 10]. For this problem, arbitrary delay functions are intractable, and thus the main line of work focuses on linear delay functions.

Additional problems with delay exist other than those over a metric space. The set aggregation problem, presented in [16], is a variant of set cover with delay. The problem of bin packing with delay is presented in [3].

The classic online facility location problem, suggested by Meyerson [23], has also been studied [21, 1]. In this problem, requests arrive one after the other, and the algorithm must either connect a request to an existing facility immediately upon the request's arrival, or open a facility at the request's location. This problem is different from the problems of facility location with deadlines and facility location with delay presented in this paper. The main difference is that in our problems, a facility is only opened momentarily, which only allows immediate connection of pending requests. In contrast, an opened facility in the online facility location of [23] is permanent, allowing the connection of any future request to that facility.

Paper Organization   Section 2 presents the problem of online facility location with deadlines, and an $O(\log^2 n)$-competitive randomized algorithm for the problem, as well as its analysis. Section 3 discusses the more general problem of online facility location with delay, and extending the algorithm for the deadline case in section 2 to an $O(\log^2 n)$-competitive algorithm for the case of delay.

Section 4 presents the $O(D^2)$-competitive deterministic algorithm for online multilevel aggregation with delay. Section 5 presents the $O(\log^2 n)$-competitive randomized algorithm for online service with delay, which relies on the algorithm for online multilevel aggregation with delay given in Section 4.

2   Online Facility Location with Deadlines

2.1   Problem and Notation

In the online facility location with deadlines problem, requests arrive on points of a metric space over time. Each request is associated with a deadline, by which it must be served. An algorithm for the problem can choose, in any point in time, to open a facility at any point in the metric space momentarily, at a cost of f. Immediately upon opening the facility, the algorithm must choose the subset of pending requests (i.e. requests that have arrived but have not been served) to connect to the facility. The cost of connecting each request to the facility is the distance between the request's location and the facility's location. Connecting a request to a facility serves that request. Immediately after connecting the requests, the facility disappears. We allow opening a facility at the same point more than once, at different times.

Formally, we are given a metric space $\mathcal{A} = (A, \delta_{\mathcal{A}})$ such that $|A| = n$. A request is a tuple $q = (v_q, r_q, d_q)$ such that v_q is a point in $\mathcal{A}$, the arrival time of the request is r_q and the deadline of the request is d_q. We assume, without loss of generality, that all deadlines are distinct. For any instance of the problem, the algorithm's solution has two costs. The first is the *buying cost* (or *opening cost*) $\mathrm{ALG}^B = mf$, where m is the number of facilities opened by the algorithm. Denoting by Q the set of requests in the instance, and denoting by β_q the location of the facility to which the algorithm connects request q, the second cost of the algorithm is the *connection cost* $\mathrm{ALG}^C = \sum_{q \in Q} \delta_{\mathcal{A}}(v_q, \beta_q)$. Wherever a single metric space $\mathcal{A}$ is considered, we write $\delta = \delta_{\mathcal{A}}$.

The goal of the algorithm is to minimize the total cost, which is

$$\mathrm{ALG} = \mathrm{ALG}^B + \mathrm{ALG}^C$$

For the special case in which A is a tree T, and δ is the distance between nodes in T, we denote the root of T by r and the weight function on the edges of the tree by w. We assume, without loss of generality, that the requests only arrive on leaves of the tree.

The following definitions regarding trees are used throughout the paper.

Definition 2.1. For every tree node $u \in T$, we use the following notations:

- For $u \neq r$, we denote by $p(u)$ the parent of u in the tree.

- We denote by T_u the subtree rooted at u.

- For a set of requests $Q \subseteq T_u$, we denote by $T_u^Q \subseteq T_u$ the subtree spanned by u and the leaves of Q.

- We define the *height* of u to be the depth of T_u.

The following definition is similar to the usual definition of a β-HST, except that we allow a child edge to be strictly smaller than $\frac{1}{\beta}$ times its parent edge.

Definition 2.2 (($\geq \beta$)-HST)**.** A rooted tree T is a $(\geq \beta)$-HST if for any two edges $e, e' \in T$ such that e is a parent edge of e', we have that $w(e) \geq \beta w(e')$.

When considering the problem over a tree T, we assume, without loss of generality, that $w(e) \leq f$ for any edge $e \in T$. Indeed, if this is not the case, no request would be connected over e, effectively yielding two disjoint instances of the problem.

In this section, we prove the following theorem.

Theorem 2.3. *There exists an $O(\log^2 n)$-competitive randomized algorithm for online facility location with deadlines for any metric space of n points.*

2.2 Algorithm for HSTs

We present an algorithm for facility location with deadlines on a (≥ 2)-HST T of depth D. We denote the root of the tree by r.

We make the assumption that the total weight of any path from the root to a leaf is at most f. In a (≥ 2)-HST, the total weight of such a path is at most twice the weight of the top edge, which is at most f. Thus, this assumption only costs us a constant factor of 2 in competitiveness.

Without loss of generality, we allow the algorithm to open facilities on internal nodes of the tree. Indeed, any algorithm that opens facilities on internal nodes can be converted to an algorithm that only opens facilities on leaves in the following manner. Consider a facility opened by the original algorithm on the internal node u, and denote by Q the set of requests connected to that facility. The modified algorithm would open the facility at v_{q^*} instead, where $q^* = \arg\min_{q \in Q} \delta(u, v_q)$, and connect the original requests. Through triangle inequality, the connection cost of the modified algorithm is at most twice larger.

Algorithm's description. The algorithm for facility location with deadlines on a (≥ 2)-HST is given in Algorithm 1. The algorithm waits until the deadline of a pending request. It then begins exploring the root node. An exploration of a node u consists of considering the pending requests in T_u by order of increasing deadline. The exploration has a budget of exactly f to spend on raising counters of child nodes – it maintains that budget in the variable b_u. When considering a request q, the algorithm raises the counter of the child node v, denoted c_v in the algorithm, for the child node v in the request's direction. The counter is raised by the smallest of $\delta(v_q, u)$, the amount required to fill c_v, and the remaining budget

---
Algorithm 1: Facility Location with Deadlines
---
1 Initialization.

2 Initialize $c_u \leftarrow 0$ for any node $u \in T \backslash \{r\}$.

3 Declare b_u for any node $u \in T$.

4

5 Event Function `UponDeadline()` *// Upon expired deadline of pending request at time t*

6   `Explore(r)`

7

8 Function `Explore(u)`

9   `Open(u)`
       // Spend a budget of f on charging child node counters

10   set $b_u \leftarrow f$

11   **while** $b_u \neq 0$ **and** there remain pending requests in T_u **do**
         // Consider pending requests by increasing deadline

12   let q be the pending request with earliest deadline in T_u

13   let v be the child of u on the path to v_q

14   call `Invest(`u`,`v`,`$\delta(u, v_q)$`)`

15   **if** $c_v = f$ **then** set $c_v \leftarrow 0$ **;** call `Explore(v)`.

16   **if** q *is still pending* **then** connect q to facility at u
---

b_u. If this fills the counter of v, the exploration of u is paused, and a new exploration of v is started, in a DFS manner. We claim, in the analysis, that this exploration of v connects q. Otherwise, the request q is connected to u.

The operation of the algorithm is visualized in Figure 6 of Appendix A.

2.3 Analysis

Fix any instance of online facility location with deadlines on a (≥ 2)-HST. Let OPT be any solution to the instance. We denote by OPT^B the total buying cost of OPT, and by OPT^C the total connection cost of OPT. Denote by ALG the total cost of the solution of Algorithm 1 for this problem. In this subsection, we prove the following theorem.

Algorithm 1: Facility Location with Deadlines (cont.)
---
1 Function `Open(u)`

2   open facility at u.

3   **if** u *is a leaf node* **then** connect to facility all pending requests on u

4

5 Function `Invest(`u`,`v`,`x`)` *// Invests in v's counter either x, or until v's counter*
                              is full, or until u is out of budget.

6   let $y \leftarrow \min(x, b_u, f - c_v)$

7   increase c_v by y

8   decrease b_u by y

9   **return** y

Theorem 2.4. $\text{ALG} \leq O(D^2) \cdot \text{OPT}^B + O(D) \cdot \text{OPT}^C$.

To prove Theorem 2.4, we show validity of the algorithm, an upper bound for ALG and a lower bound for OPT.

Throughout the analysis, we denote by k the number of calls to `UponDeadline` made by the algorithm. We also denote by $t_1, ..., t_k$ the times of these k calls, by increasing order.

2.3.1 Validity of the Algorithm

The following proposition and its corollary show that the algorithm is valid.

Proposition 2.5. *Let q be a request considered in a call to* `Explore`(u)*. Then q is served when* `Explore`(u) *returns.*

Proof. This is guaranteed by the condition check at the end of the main loop in `Explore`. $\square$

Corollary 2.6. *Every request is served by its deadline. That is, the algorithm is valid.*

Proof. Observe that upon the deadline of a request q, `Explore`(r) is called, and immediately considers q. Proposition 2.5 concludes the proof. $\square$

2.3.2 Upper Bounding ALG

We now proceed to bound ALG by proving the following lemma.

Lemma 2.7. $\text{ALG} \leq 3 \cdot (D+1) \cdot kf$.

The proof of Lemma 2.7 is through providing an upper bound for the cumulative amount by which counters are raised in the algorithm, then bounding the cost of the algorithm by that cumulative amount.

Observation 2.8. *Observe any node u, and consider a call to* `Explore`(u)*. Denote by x the total amount by which* `Explore`(u) *increases the counters of its children nodes through calls to* `Invest`*. Then we have that $x \leq f$. Moreover, if there exists a pending request in T_u after the return of* `Explore`(u)*, then $x = f$.*

From the previous observation, the following observation follows.

Observation 2.9. *For any u,* `Explore`(u) *is called at most once at any time t.*

Using the last observation, we refer to a call to `Explore`(u) at time t by `Explore`$_t(u)$.

Observe the state of each counter in the algorithm over time. The counter undergoes phases, such that in the start of each phase its value is 0. The counter increases in value during the phase until it reaches f, and is then reset to 0, triggering a service and the end of the phase.

We define a virtual counter $\bar{c}_u$ which contains the cumulative value of c_u. That is, whenever c_u increases, $\bar{c}_u$ increases by the same amount, but $\bar{c}_u$ is never reset when c_u is reset. For the sake of analysis, we also consider a virtual counter $\bar{c}_r$, which is raised by f whenever `Explore`(r) is called.

We define $\bar{C}_j = \sum_{\text{node } u \text{ at depth } j} \bar{c}_u$. Observe that $\bar{C}_0 = \bar{c}_r$.

Proposition 2.10. *For every $j \in [D]$, $\bar{C}_j \leq \bar{C}_{j-1}$.*

9

Proof. Observe that the counters at depth j are raised only upon a call to $\texttt{Explore}(u)$ for a node u at depth $j-1$. $\texttt{Explore}(u)$ is only called after $\bar{c}_u$ is raised by f, and every such call raises counters at depth j by at most f (using Observation 2.8). $\square$

Corollary 2.11. $\sum_{u \in T} \bar{c}_u \leq (D+1)kf$.

Proof. Observe that $\bar{C}_0 = \bar{c}_r = kf$. Using Proposition 2.10, we have that

$$\sum_{u \in T} \bar{c}_u = \sum_{j=0}^{D} \bar{C}_j \leq \sum_{j=0}^{D} \bar{C}_0 = (D+1)kf$$

$\square$

Proposition 2.12. *Suppose the function* $\texttt{Explore}(u)$ *calls* $\texttt{Invest}(u,v,x)$ *when considering request* q. *Then at least one of the following holds:*

1. $\texttt{Invest}(u,v,x)$ *returns* x.

2. $b_u = 0$ *after the return of* $\texttt{Invest}$.

3. *The condition check in* $\texttt{Explore}$ *of whether* q *is still pending fails.*

Proof. If $\texttt{Invest}(u,v,x)$ does not return x, and $b_u \neq 0$ after its return, then it must be that $c_v = f$. In this case, $\texttt{Explore}(u)$ calls $\texttt{Explore}(v)$ when checking the condition after the return of $\texttt{Invest}$. Request q is the pending request with earliest deadline under T_u, and thus also under T_v. Hence, q is immediately considered by $\texttt{Explore}(v)$, and is thus served by the end of $\texttt{Explore}(v)$ by Proposition 2.5. $\square$

Proposition 2.13. $\text{ALG} \leq 3 \cdot \sum_{u \in T} \bar{c}_u$

Proof. The costs of the algorithm (both opening and connection) are contained in calls to the function $\texttt{Explore}$ (where we associate the opening costs in $\texttt{Open}$ to the $\texttt{Explore}$ call that invoked it). In each call to $\texttt{Explore}(u)$, the algorithm has a cost of f in opening a facility at u.

In addition, the algorithm incurs connection costs, as $\texttt{Explore}(u)$ connects any considered request if it is still pending at the end of the loop's iteration. From Proposition 2.12, if $\texttt{Explore}(u)$ connects a request q, then either the preceding $\texttt{Invest}(u,v,\delta(u,v_q))$ returned $\delta(u,v_q)$, or $b_u = 0$ after the return of that call to $\texttt{Invest}$.

Observe the calls to $\texttt{Invest}(u,v,\delta(u,v_q))$ that return $\delta(u,v_q)$. For those requests, the connection cost of q is exactly the return value of $\texttt{Invest}$. But the return values of $\texttt{Invest}$ sum to at most the initial value of b_u, which is f. Thus, connection costs for those requests sum to at most f.

As for calls to $\texttt{Invest}$ after which we have that $b_u = 0$, observe that there is at most one such call, after which the loop in $\texttt{Explore}$ ends. The connection cost for the request considered in this iteration is $\delta(u,v_q) \leq f$.

Overall, the connection costs in $\texttt{Explore}(u)$ sum to at most $2f$.

Thus, in each call to $\texttt{Explore}(u)$, the total cost of the algorithm (buying and connection) is at most $3f$. Observing that $\texttt{Explore}(u)$ is called only upon raising $\bar{c}_u$ by f concludes the proof. $\square$

Proof of Lemma 2.7. The lemma results directly from Proposition 2.13 and Corollary 2.11. $\square$

(a) Charging Nodes for Single Tree Node   (b) Charging Nodes for a Branch

Sub-figure 1a is a visualization of the charging nodes corresponding to a single tree node, displayed over a timeline. Each rectangle is a charging node. Note the charging node from $-\infty$ and the charging node to ∞.

Sub-figure 1b shows the charging nodes corresponding to a branch in the tree. Observe the containment of charging node intervals for a certain tree node in the intervals of descendant tree nodes.

Figure 1: Visualization of Charging Nodes

2.3.3 Lower Bounding OPT

We now lower bound the cost of OPT.

Charging nodes and incurred costs. We define a charging node to be a tuple $(u, [\tau_1, \tau_2])$ such that $u \in T$, and τ_1, τ_2 are two subsequent times in which $\texttt{Explore}(u)$ is called. We allow the charging nodes of the form $(u, [\tau_1, \tau_2])$ in which $\tau_1 = -\infty$ and τ_2 is the first time in which $\texttt{Explore}(u)$ is called. Similarly, we allow the charging nodes $(u, [\tau_1, \tau_2])$ in which τ_1 is the last time $\texttt{Explore}(u)$ is called, and $\tau_2 = \infty$. We denote by M the set of charging nodes.

For a charging node $\mu = (u, [\tau_1, \tau_2])$, we define the following.

1. Let $c_b(\mu)$ be the *buying cost incurred by* OPT *in* μ, defined to be the total cost at which OPT opened facilities in T_u during $[\tau_1, \tau_2]$.

2. Let $c_c(\mu)$ be the *connection cost incurred by* OPT *in* μ, defined to be $\sum_{q \in Q} \delta(p(u), v_q)$, where Q is the set of requests q such that $v_q \in T_u$, $r_q \in [\tau_1, \tau_2]$ and OPT connected q to a facility outside T_u.

Let $c(\mu) = c_b(\mu) + c_c(\mu)$ be the *total cost* OPT *incurred in* μ.

Lemma 2.14. $\sum_\mu c(\mu) \leq 2(D+1) \cdot \text{OPT}^B + 4 \cdot \text{OPT}^C$.

Proof. We consider each action of OPT and how it affects the incurred cost at various charging nodes.

For a facility that is opened by OPT at node u at time t to participate in $c_b((u', [\tau_1, \tau_2]))$, we must have that $u \in T_{u'}$. Hence, u' is on the branch from the root to u, and thus u' is one of at most $D+1$ possible nodes. We also have that $t \in [\tau_1, \tau_2]$. Using Observation 2.9, we have that $\tau_2 > \tau_1$, and therefore t can belong to at most two such intervals, for every choice of u'. This yields that the cost of each facility opened by OPT is counted in $\sum_\mu c_b(\mu)$ at most $2(D+1)$ times.

As for connection costs, consider a request q that OPT connects to a facility at node v. Denote by u the least common ancestor of v and v_q. If OPT incurs connection cost due to q in charging node $(u', [\tau_1, \tau_2])$, then $v_q \in T_{u'}$ and $v \notin T_{u'}$. Therefore, u' must be on the path from v_q to u (including v_q, and not including u). Let $u = u^{(0)}, u^{(1)}, u^{(2)}, ..., u^{(m)} = v_q$ be the path from u to v_q. As with the buying cost, for every $l \in [m]$, OPT may incur connection cost due to q in at most 2 charging nodes of the form $(u^{(l)}, [\tau_1, \tau_2])$ for some τ_1, τ_2. Therefore, denoting the total connection cost of OPT due to connecting q by X, we have that

$$X \leq 2 \cdot \sum_{l=1}^{m} \delta(u^{(l-1)}, v_q)$$

Now observe that since the tree is a (≥ 2)-HST, we have that the total weight of any path from a node to a descendant leaf is at most the weight of the node's ancestor edge. Therefore, for every $l \in [m]$:

$$\delta(u^{(l-1)}, v_q) = w((u^{(l-1)}, u^{(l)})) + \delta(u^{(l)}, v_q)$$
$$\geq 2\delta(u^{(l)}, v_q)$$

Therefore, we have that $\delta(u^{(l)}, v_q) \leq \frac{1}{2^l} \cdot \delta(u, v_q)$. Hence:

$$X \leq 2 \cdot \sum_{l=1}^{m} \frac{1}{2^{l-1}} \cdot \delta(u, v_q) \leq 4\delta(u, v_q)$$

Since u is on the path from v to v_q, we have that $\delta(u, v_q) \leq \delta(v, v_q)$. Hence X is at most 4 times the connection cost of OPT for q. This concludes the lemma. $\qquad\square$

Definition 2.15 (excess). Let $G = (V', E)$ be a directed multigraph, with a non-negative weight function $\alpha : E \to R^+$ defined on its edges. We denote by $E_v^+ \subseteq E$ the set of edges entering node v, and by $E_v^- \subseteq E$ the set of edges leaving v. We define the *excess at a node* $v \in V'$ to be $\chi_v = \sum_{\sigma \in E_v^+} \alpha(\sigma) - \sum_{\sigma \in E_v^-} \alpha(\sigma)$.

Note that every edge $\sigma \in E$ from u to v is counted in χ_u and χ_v with opposite signs. The following observation follows.

Observation 2.16. *For any $G = (V', E)$ and weights $\alpha : E \to \mathbb{R}^+$, we have $\sum_{v \in V'} \chi_v = 0$.*

Definition 2.17. For a graph $G = (V' = V \cup \{s\}, E)$ and non-negative weights $\alpha : E \to \mathbb{R}^+$, We say that $Z = (G, s, \alpha)$ is a *preflow* if for every node $v \neq s$ we have that $\chi_v \geq 0$. We call s the *source node* of the preflow.

Observation 2.16 yields that $\chi_s \leq 0$ for every preflow $Z = (G, s, \alpha)$. We write $\omega_Z = -\chi_s$.

Proposition 2.18. *For $G = (V \cup \{s\}, E)$ a directed graph, for weights $\alpha : E \to \mathbb{R}^+$ such that $Z = (G, s, \alpha)$ is a preflow, and for every $S \subseteq V$, we have $\sum_{v \in S} \chi_v \leq \omega_Z$.*

Proof. Observation 2.16 and the definition of a preflow, we get $\sum_{v \in S} \chi_v \leq \sum_{v \in V} \chi_v = -\chi_s = \omega_Z$. $\qquad\square$

We now construct a preflow to lower bound OPT. The graph G underlying the preflow has the set of nodes $M \cup \{s\}$, where M is the set of charging nodes and s is a source node.

Consider a charging node $\mu = (u, [\tau_1, \tau_2])$. We have that $[\tau_1, \tau_2]$ corresponds to a phase of the counter c_u, since c_u was empty at τ_1 and was filled and emptied again until time τ_2.

Definition 2.19 (Investing). Observe two charging nodes $\mu = (u, [\tau_1, \tau_2])$ and $\mu' = (u' = p(u), [\tau_1', \tau_2'])$. We say that μ' *invested* x in μ if the function call $\texttt{Explore}_{\tau_1'}(u')$ increased c_u by x, through calls to $\texttt{Invest}$, during the phase of c_u between τ_1 and τ_2.

Procedure 2: PreflowBuilder - Facility Location with Deadlines

---

1 Initialization.

2 Let the set of vertices of G be $M \cup \{s\}$, and initialize G's edge set to be $E = \emptyset$.

3 Initialize dictionary $\texttt{Color}[\mu] = \texttt{None}$ for every $\mu \in M$.

4 foreach $\mu = (u, [\tau_1, \tau_2]) \in M$ such that OPT opened a facility in T_u during $[\tau_1, \tau_2]$ **do**

5   set $\texttt{Color}[\mu] \leftarrow \texttt{Special}$

6 foreach $\mu \in M$ such that $c(\mu) > 0$ **do**

7   add a new edge $\sigma = (s, \mu)$ to E, and set $\alpha(\sigma) = c(\mu)$

8

9 Function $\texttt{PreflowBuilder}()$

    // Create new colors for root charging nodes

10   **for** i from 1 to k **do**

11   let $\mu \leftarrow (r, [t_{i-1}^r, t_i^r])$

12 $\texttt{SetColor}(\mu, \mu)$

    // Propagate colors to other charging nodes

13   **for** j from 1 to D **do**

14   **foreach** $\mu = (u, [\tau_1, \tau_2]) \in M$ such that u is of depth j **do**

15   **foreach** edge $\sigma \in E_\mu^-$ incoming to a node μ' **do**

16   **if** $\texttt{SetColor}(\mu, \texttt{Color}[\mu']) \neq None$ **then break**

Procedure 2: PreflowBuilder - Facility Location with Deadlines (cont.)

---

1 Function $\texttt{SetColor}(\mu, \mu^\star)$ *// If $\texttt{Color}[\mu] = None$, tries to set it to $\mu^\star$*

2   let $[\tau_1, \tau_2]$ be the interval of μ and let $[\tau_1^\star, \tau_2^\star]$ be the interval of $\mu^\star$.

3   **if** ($\texttt{Color}[\mu] = \texttt{None}$ **and** $\tau_1 \neq -\infty$ **and** $\lambda_\mu \leq \tau_2^\star$) **then**

4   add the edges in $\bar{E}$ incoming to μ to E.

5   set $\texttt{Color}[\mu] \leftarrow \mu^\star$

6   **return** $\texttt{Color}[\mu]$

Definition 2.20 (λ_u^t and λ_μ). For every function call $\texttt{Explore}_t(u)$ for some $u \in T$ and time t, we denote by λ_u^t the earliest deadline of a pending request in T_u immediately after the return of $\texttt{Explore}_t(u)$ (if there are no pending requests in T_u, we write $\lambda_u^t = \infty$).

In addition, for a charging node $\mu = (u, [\tau_1, \tau_2])$ with $\tau_1 \neq -\infty$, we write $\lambda_\mu = \lambda_u^{\tau_1}$.

Possible edges. We describe the set of possible edges in G from nodes in M to other nodes in M, denoted by $\bar{E}$, and the weight function $\alpha : \bar{E} \to \mathbb{R}^+$. The final set of edges added to G by Procedure 2 from the nodes of M to themselves is a subset of $\bar{E}$. The set $\bar{E}$ contains an edge σ from any charging node $\mu_1 = (u_1, [\tau_1^1, \tau_2^1])$ to any charging node $\mu_2 = (u_2, [\tau_1^2, \tau_2^2])$ if μ_1 invested in μ_2. We set the weight $\alpha(\sigma)$ to be the amount that μ_1 invested in μ_2.

In the analysis of the preflow $Z = (G, s, \alpha)$ resulting from this procedure, we refer to the values of the variables used in the procedure in their final state.

Proposition 2.21. *For every charging node $\mu = (u, [\tau_1, \tau_2]) \in M$, we have that $\sum_{\sigma \in E_\mu^-} \alpha(\sigma) \leq f$.*

(a) Initial state before coloring

(b) After assigning colors to root charging nodes

(c) After propagating color to depth 1

(d) Final state after preflow construction

In this figure, we see the stages of the preflow construction in Procedure 2, visualized on a set of charging nodes corresponding to a branch in the tree. Sub-Figure 2a shows the state after the initialization, where the `Special` color has been assigned. The node with the `Special` color appears as striped. The gray edges are the edges of $\bar{E}$, not yet added to the edge set E.

Sub-Figure 2b shows the state after the creation of the colors at the root charging nodes. Note that when a node is colored, the edges of $\bar{E}$ incoming to that node are added to E. In the figure, nodes with the `None` color for which the procedure will not call `SetColor` again (i.e. `None` is their final color) are colored gray. Note that the procedure does *not* call `SetColor` for the final root node (the interval of which ends at ∞), and thus its color remains `None`.

Sub-Figure 2c shows the state after the propagation of colors to the nodes at depth 1. Sub-Figure 2d shows the final state of the preflow, after coloring the nodes at the maximum depth. Note that the color of a node μ at the maximum depth is either `Special` or `None`, as $\lambda_\mu = \infty$.

Figure 2: Visualization of preflow construction

Proof. Observe that E_μ^- is a subset of $\bar{E}$. In $\bar{E}$, the sum of $\alpha(\sigma)$ over edges σ outgoing from μ is exactly the amount μ invested in other charging nodes, which is at most f. ☐

Corollary 2.22. *For a charging node $\mu \in M$ in which $\sum_{\sigma \in E_\mu^+} \alpha(\sigma) \geq f$ we have $\chi_\mu \geq 0$.*

Proposition 2.23. *Let $\mu = (u, [\tau_1, \tau_2])$ be such that $Color[\mu] = \mu^\star$ for some charging node $\mu^\star = (r, [\tau_1^\star, \tau_2^\star])$. Then OPT did not open a facility in T_u during $[\tau_1, \tau_2^\star]$.*

Proof. Since $Color[\mu] \neq$ `Special`, we have that OPT did not open a facility in T_u during $[\tau_1, \tau_2]$.

The proof is by induction on the depth of u. If $u = r$, then it must be that $\mu = \mu^\star$, completing the proof. Otherwise, observe the node $\mu' = (p(u), [\tau_1', \tau_2'])$ from which μ inherited its color. By the induction hypothesis, OPT did not open a facility in $T_{p(u)}$ during $[\tau_1', \tau_2^\star]$. Since there exists an edge from μ to μ', we must have that $\tau_1' \in [\tau_1, \tau_2]$, which completes the proof. ☐

Lemma 2.24. *$Z = (G, s, \alpha)$ is a preflow. That is, for every charging node $\mu = (u, [\tau_1, \tau_2]) \in M$ we have $\chi_\mu \geq 0$.*

Proof. We observe the following cases according to the final values of the variables in the graph construction procedure.

Case 1: $\texttt{Color}[\mu] = \texttt{Special}$. In this case, OPT opened a facility in T_u during $[\tau_1, \tau_2]$, implying that $c(\mu) \geq f$. In the initialization of Procedure 2, an edge σ from s to μ with $\alpha(\sigma) = c(\mu) \geq f$ is created, and thus Corollary 2.22 implies that $\chi_\mu \geq 0$.

Case 2: $\texttt{Color}[\mu] = \mu^\star$ for some charging node $\mu^\star$. In this case, incoming edges to μ were added, with a total weight which is the total amount invested by μ. Since $\texttt{Color}[\mu] = \mu^\star$ was set by $\texttt{SetColor}$, we must have that $\tau_1 \neq -\infty$ and that $\lambda_\mu < \infty$. Using Observation 2.8, we have that $\texttt{Explore}_{\tau_1}(u)$ raised counters by a total of exactly f. Corollary 2.22 therefore proves the lemma for this case.

Case 3: $\texttt{Color}[\mu] = \texttt{None}$. Observe any edge $\sigma \in E_\mu^-$, incoming to some node $\mu' = (u', [\tau_1', \tau_2'])$. It must be that $\texttt{Color}[\mu'] = \mu^\star$, for some charging node $\mu^\star = (r, [\tau_1^\star, \tau_2^\star])$. Note that μ' invested in μ, and thus $\tau_1' \in [\tau_1, \tau_2]$. Combining Proposition 2.23 for μ' and the fact that $\texttt{Color}[\mu] \neq \texttt{Special}$, we have that OPT did not open a facility in T_u during $[\tau_1, \tau_2^\star]$.

We therefore have that for every request q such that $v_q \in T_u$ and $[r_q, d_q] \subseteq [\tau_1, \tau_2^\star]$, OPT must connect q to some facility at a node $v \notin T_u$. If it also holds that $r_q \leq \tau_2$, then OPT incurs a connection cost of $\delta(v_q, u')$ in μ on connecting q. We proceed to find such requests q.

Now observe that μ' invested $\alpha(\sigma)$ in μ. Thus, there exists a set of requests L_σ that are considered in $\texttt{Explore}_{\tau_1'}(u')$ such that $\alpha(\sigma) \leq \sum_{q \in L_\sigma} \delta(a_q, u')$ and $a_q \in T_u$ for every $q \in L_\sigma$. Since the requests of L_σ are considered in $\texttt{Explore}_{\tau_1'}(u')$, we have that $\lambda_{\mu'} \geq d_q$ for every $q \in L_\sigma$. Since $\texttt{Color}[\mu'] = \mu^\star$, we have that $\lambda_{\mu'} \leq \tau_2^\star$, and thus $d_q \leq \tau_2^\star$.

Observe any $q \in L_\sigma$. It holds that $r_q \leq \tau_1' \leq \tau_2$, since q is considered in $\texttt{Explore}_{\tau_1'}(u')$. Now, observe that $\texttt{Color}[\mu] = \texttt{None}$ even though $\texttt{Color}[\mu'] = \mu^\star$. Hence, either $\tau_1 = -\infty$ or $\lambda_\mu > \tau_2^\star$. If $\tau_1 = -\infty$, then $r_q \geq \tau_1$. Otherwise, $\tau_1 \neq -\infty$ and $\lambda_\mu > \tau_2^\star$. Since $d_q \leq \tau_2^\star$, it must be that q was not pending immediately after the return of $\texttt{Explore}_{\tau_1}(u)$. However, $\texttt{Explore}_{\tau_1'}(u')$ considered q when raising c_u toward $\texttt{Explore}_{\tau_2}(u)$. Thus, q was released after τ_1.

Overall, for every $q \in L_\sigma$ we have that $r_q \in [\tau_1, \tau_2]$ and $d_q \leq \tau_2^\star$. Thus, OPT incurs a connection cost of at least $\sum_{q \in L_\sigma} \delta(v_q, u^{(1)})$ in the charging node μ due to L_σ, which is at least $\alpha(\sigma)$.

Now, if for every distinct $\sigma_1, \sigma_2 \in E_\mu^-$ we have that $L_{\sigma_1} \cap L_{\sigma_2} = \emptyset$, then the connection cost OPT incurs in μ is at least $\sum_{\sigma \in E_\mu^-} \alpha(\sigma)$. Indeed, observe that σ_1 and σ_2 enter two distinct charging nodes $\mu^{(1)} = (p(u), [\tau_1^{(1)}, \tau_2^{(1)}])$ and $\mu^{(2)} = (p(u), [\tau_1^{(2)}, \tau_2^{(2)}])$. Lemma 2.9 implies that $\tau_1^{(1)} \neq \tau_1^{(2)}$. It is enough to observe that for $b \in \{1, 2\}$, each request $q \in L_{\sigma_b}$ is pending before $\tau_1^{(b)}$ and is served after $\tau_1^{(b)}$.

Overall, we have that $c(\mu) \geq \sum_{\sigma \in E_\mu^-} \alpha(\sigma)$. In the initialization of Procedure 2, an edge σ from s to μ with $\alpha(\sigma) = c(\mu)$ is created, and thus $\chi_\mu \geq 0$ as required. This concludes the proof of the current case, and the lemma. □

Lemma 2.25. *For each $i \in [k]$ and charging node $\mu = (r, [t_{i-1}, t_i])$, we have $\chi_\mu \geq f$.*

Proof. Observe that $E_\mu^- = \emptyset$. It remains to see that $\sum_{\sigma \in E_\mu^+} \alpha(\sigma) \geq f$.

If $\texttt{Color}[\mu] \neq \texttt{None}$, it holds that $\sum_{\sigma \in E_\mu^+} \alpha(\sigma) \geq f$ identically to Cases 1 and 2 in Lemma 2.24.

Otherwise, we must have that $\texttt{SetColor}(\mu, \mu)$ returned $\texttt{None}$. Thus, it must be that either $t_{i-1} = -\infty$ or $\lambda_\mu > t_i$. We claim that either of these cases contradicts $\texttt{Color}[\mu] \neq \texttt{Special}$. Indeed, observe that $\texttt{Explore}_{t_i}(r)$ is called upon a deadline of a request q. If $t_{i-1} = -\infty$, it holds that $r_q \geq t_{i-1}$. If $\lambda_\mu > t_i$, it must be that $r_q \geq t_{i-1}$ as well. Overall, $[r_q, d_q] \in [t_{i-1}, t_i]$, and thus OPT must open a facility in that interval, in contradiction. □

Figure 3: Cases of Lemma 2.24

Lemma 2.26. $kf \leq 2(D+1) \cdot \mathrm{OPT}^B + 4 \cdot \mathrm{OPT}^C$

Proof. Lemma 2.24 yields that Z is a valid preflow. For $i \in [k]$, let $\mu_i = (r, [t_{i-1}, t_i])$. Using Lemma 2.25 and Proposition 2.18, we have that

$$kf \leq \sum_{i=1}^{k} \chi_{\mu_i} \leq \omega_Z$$

Now observe that $E_s^+ = \emptyset$, and that $\sum_{\sigma \in E_s^-} \alpha(\sigma) = \sum_{\mu \in M} c(\mu)$. Using Lemma 2.14, we obtain

$$kf \leq \omega_Z = \sum_{\sigma \in E_s^-} \alpha(\sigma) = \sum_{\mu \in M} c(\mu) \leq 2(D+1) \cdot \mathrm{OPT}^B + 4 \cdot \mathrm{OPT}^C$$

as required. $\qquad\square$

Proof of Theorem 2.4. Combining Lemmas 2.7 and 2.26, we have that

$$\mathrm{ALG} \leq Dkf \leq O(D^2) \cdot \mathrm{OPT}^B + O(D) \cdot \mathrm{OPT}^C$$

$\qquad\square$

Remark 2.27. Our algorithm and its analysis also work in the case that the cost of opening a facility is different between nodes in the tree, as long as the cost of opening a facility at a node is at least the cost of opening a facility at its parent node. If this is not the case, observe that the analysis of Case 1 of Lemma 2.24 would no longer hold.

2.4 From HST to General Metric Space

In this subsection, we show how the deterministic Algorithm 1 for (≥ 2)-HSTs yields a randomized $O(\log^2 n)$-competitive algorithm for facility location with deadlines on a general metric space with n points, thus proving Theorem 2.3. To do this, we consider a standard probabilistic embedding of a metric space to an HST.

Theorem 2.28. *For any metric space $\mathcal{X} = (X, \delta_{\mathcal{X}})$ such that $|\mathcal{X}| = n$, there exists a distribution $\mathcal{D}$ over (≥ 2)-HSTs of depth $O(\log n)$ such that X are the leaves of the HST, such that the expected distortion is $O(\log n)$. That is, for every $x_1, x_2 \in X$ we have that*

$$\delta_{\mathcal{X}}(x_1, x_2) \leq \mathbb{E}_{\mathcal{T} \sim \mathcal{D}}\left[\delta_{\mathcal{T}}(x_1, x_2)\right] \leq O(\log n)$$

where $\delta_{\mathcal{T}}$ is the distance in the tree $\mathcal{T}$.

Theorem 2.28 is a direct result of composing the embeddings of Fakcharoenphol et al. [20] and Bansal et al. [6].

We observe the following randomized algorithm for facility location with delay on a general metric space:

1. Embed the metric space to a (≥ 2)-HST according to the distribution in Theorem 2.28.

2. Run Algorithm 1 on the resulting (≥ 2)-HST.

Proof of Theorem 2.3. We show that the randomized algorithm described above is indeed $O(\log^2 n)$-competitive. Fix any instance of facility location with deadlines. We denote by $\mathrm{ALG}^{\mathcal{T}}$ the cost of the algorithm on the instance with regard to distances on the chosen (≥ 2)-HST $\mathcal{T}$. Since $\delta_{\mathcal{T}}(x_1, x_2) \geq \delta_{\mathcal{X}}(x_1, x_2)$, we have that $\mathrm{ALG}^{\mathcal{X}} \leq \mathrm{ALG}^{\mathcal{T}}$, where $\mathrm{ALG}^{\mathcal{X}}$ is the actual cost incurred by the algorithm on this instance.

From Theorem 2.4, we know that for any solution $\mathrm{OPT}^{\mathcal{T}}$ for the instance on $\mathcal{T}$, it holds that $\mathrm{ALG}^{\mathcal{T}} \leq O(D^2) \cdot \mathrm{OPT}^{\mathcal{T},B} + O(D) \cdot \mathrm{OPT}^{\mathcal{T},C}$, where D is the depth of $\mathcal{T}$ (and thus $D = O(\log n)$).

Now, denote by $\mathrm{OPT}^{\mathcal{X}}$ the optimal solution for the instance over $\mathcal{X}$. Observe that for every $\mathcal{T}$ in the support of $\mathcal{D}$, OPT yields a solution $\mathrm{OPT}^{\mathcal{T}}$ by opening facilities at the same locations, at the same times, and connecting the same requests. It holds that $\mathrm{OPT}^{\mathcal{T},B} = \mathrm{OPT}^{\mathcal{X},B}$, and that $\mathbb{E}_{\mathcal{T} \sim \mathcal{D}}\left[\mathrm{OPT}^{\mathcal{T},C}\right] \leq O(\log n) \cdot \mathrm{OPT}^{\mathcal{X},C}$.

Combining the above facts, we have that

$$\mathbb{E}\left[\mathrm{ALG}^{\mathcal{X}}\right] \leq \mathbb{E}_{\mathcal{T} \sim \mathcal{D}}\left[\mathrm{ALG}^{\mathcal{T}}\right] \leq \mathbb{E}_{\mathcal{T} \sim \mathcal{D}}\left[O(\log^2 n) \cdot \mathrm{OPT}^{\mathcal{T},B} + O(\log n) \cdot \mathrm{OPT}^{\mathcal{T},C}\right]$$
$$\leq O(\log^2 n) \cdot \mathrm{OPT}^{\mathcal{X},B} + O(\log^2 n) \cdot \mathrm{OPT}^{\mathcal{X},C} = O(\log^2 n) \cdot \mathrm{OPT}^{\mathcal{X}}$$

proving the theorem. $\qquad\square$

The reasoning behind the main theorem of this subsection is that the connection cost is distorted upon HST embedding, while the buying cost is not. Thus, the HST algorithm is allowed to lose a larger factor over OPT^B ($\Theta(\log^2 n)$) compared to the factor it loses over OPT^C ($\Theta(\log n)$). This property is used to analyze the other problems considered in this paper in a similar manner.

Remark 2.29. For the case of different costs for opening facilities at different points in the metric space, we obtain a $O(\log^2 \Delta + \log \Delta \log n)$-competitive randomized algorithm, with Δ the aspect ratio of the metric space, in the following manner. We use the embedding of Fakcharoenphol et al. [20] without composing it with the embedidng of [6]. This yields a 2-HST (rather than a (≥ 2)-HST), which has a depth of $O(\log \Delta)$, with Δ the aspect ratio of the original metric space. This tree has an distortion of $O(\log n)$.

In this 2-HST, for each node u, the distances between u and the leaves in T_u are equal. Thus, we can allow the algorithm to open a facility at u, at a cost which is the minimal cost of opening a facility at a leaf in T_u, at a loss of a factor of 2 in connection cost. The resulting tree has non-decreasing opening costs from the root to any leaf, and is (in particular) a (≥ 2)-HST of depth $D = O(\log \Delta)$. Thus, using the algorithm for (≥ 2)-HSTs and Remark 2.27, and applying the distortion of $O(\log n)$ to the connection cost as in the proof of Theorem 3.1, we obtain the $O(\log^2 \Delta + \log \Delta \log n)$-competitive algorithm.

3 Facility Location with Delay

3.1 Problem and Notation

We now describe the facility location with delay problem. The problem is an extension of the facility location with deadlines problem, in which the deadline for each request q is replaced with an arbitrary delay function $d_q(t)$ associated with that request. Each delay function is required to be continuous and monotonically non-decreasing. This is indeed an extension of the deadline problem, as a deadline can be described as a step function, which goes from 0 to infinity at the time of the deadline. Such a step function can be approximated arbitrarily well by a continuous delay function.

A feasible solution for a facility location with delay instance consists of opening facilities and connecting each request to some facility, as in the deadline case. In addition to the opening costs and connection costs incurred, the solution also pays $d_q(t)$ for each request q connected at time t. Overall, for an instance of the problem with requests Q, the algorithm incurs the delay cost $\text{ALG}^D = \sum_{q \in Q} d_q(t_q)$, where t_q is the time in which q is served by the algorithm. Thus, the algorithm's goal is to minimize the total cost

$$\text{ALG} = \text{ALG}^B + \text{ALG}^C + \text{ALG}^D$$

Without loss of generality, we assume that $d_q(r_q) = 0$. Indeed, if this is not the case, observe that any solution (including the optimal one) must pay this initial amount of $d_q(r_q)$ in delay for that request, which only reduces the competitive ratio of any online algorithm.

In this section, we prove the following theorem.

Theorem 3.1. *There exists an $O(\log^2 n)$-competitive randomized algorithm for facility location with delay for a general metric space of size n.*

3.2 Algorithm for HSTs

In this subsection, we present a deterministic algorithm for facility location with delay on a (≥ 2)-HST. This algorithm yields a randomized $O(\log^2 n)$-competitive algorithm for general metric spaces, in a similar way to the deadline case.

We require the following definitions.

Definition 3.2 (Solution). Let Q be a set of requests. For $S \subseteq X$, and a function $\phi : Q \to S$ we say that (S, ϕ) is a *solution for Q*, with a cost $|S| \cdot f + \sum_{q \in Q} \delta(v_q, \phi(q))$. If $S \subseteq T_u$ for some node u, we write that (S, ϕ) is a *solution for Q under u*.

Definition 3.3 (Ancestor-closed solution). Let Q be a set of requests, and let (S, ϕ) be a solution for Q under a node u. We say that (S, ϕ) is an *ancestor-closed solution for Q under u* if for every $s \in S$ such that $s \neq u$ we have that $p(s) \in S$.

If $u = r$, we simply write that (S, ϕ) is an ancestor-closed solution for Q.

Definition 3.4 ($\psi(Q)$ and $\psi_u(Q)$)**.** We define $\psi(Q)$ to be the cost of the minimal-cost ancestor-closed solution for Q. Similarly, we define $\psi_u(Q)$ to be the cost of the minimal-cost ancestor-closed solution for Q under u.

Definition 3.5 (Critical request set)**.** We say that a set of pending requests Q at time t is *critical* if $d_Q(t) \geq \psi(Q)$.

Algorithm's description. Our algorithm is given in Algorithm 3. The algorithm calls `UponCritical` whenever a set of pending requests Q becomes critical. It uses the `Open` and `Invest` functions from Algorithm 1, but redefines the `Explore` function. The function call `Explore`(u) now forwards time until the first occurrence of one of two events.

The first event is a pending request q in T_u, the delay of which exceeds the cost of connecting it to u. Handling this event is similar to handling the event considered in Algorithm 1 – we attempt to raise the counter of the child node v in q's direction by $\delta(u, v_q)$. If this fills the counter of v, this triggers an immediate call to `Explore`(v). However, in contrast to the deadline case, calling `Explore`(v) is not guaranteed to connect the request q. For this reason, `Explore`(u) must invest the remainder of $\delta(u, v_q)$ (or until $b_u = 0$) in v's counter before connecting q to u.

The second event is not analogous to the deadline case. In this event, for a child v of u and a "coalition" of pending requests Q in T_v, we have that the delay of Q exceeds $\psi_v(Q)$. In this case, the algorithm invests in v until either it is out of budget ($b_u = 0$) or v's counter is full ($c_v = f$). It is important to note that in contrast to the first event, the fact that `Explore`(u) considered Q does not provide any guarantees for connecting the requests of Q.

Algorithm 3 changes `Explore`(u) so that time is forwarded until one of two events happens, rather than the single event in Algorithm 1 (i.e. expired deadline). These two events are shown in Figure 4.

3.3 Analysis

Fix any instance of facility location with delay on a (≥ 2)-HST.

Theorem 3.6. $\mathrm{ALG} \leq O(D^2) \cdot \mathrm{OPT}^B + O(D) \cdot \mathrm{OPT}^C + O(D^2) \cdot \mathrm{OPT}^D$.

Observe that the connection cost is distorted by the embedding, while the buying and delay costs are not. Thus, using an identical argument to the proof of Theorem 2.3 of the deadline problem, Theorem 3.6 implies Theorem 3.1.

We devote this subsection to prove Theorem 3.6.

3.3.1 Upper Bounding ALG

To upper bound the cost of the algorithm, we show the following Lemma.

Lemma 3.7. $\mathrm{ALG} \leq 6 \cdot (D + 1) \cdot kf$

Proposition 3.8. $\mathrm{ALG}^D \leq \mathrm{ALG}^B + \mathrm{ALG}^C$

Proof. The algorithm explicitly maintains that for every set of pending requests Q at any time t we have that $\psi(Q) \geq d_Q(t)$. Now, consider that since the delay of a pending request goes to infinity, the algorithm ultimately serves every request. Consider a specific service made by the algorithm, described by a solution (S, ϕ) to some set of requests Q, and note that (S, ϕ) is an ancestor-closed solution to Q. Thus, its total cost is at least $\psi(Q)$, completing the proof. $\qquad \square$

Algorithm 3: Facility Location with Delay

---

1 Initialization.
2 Initialize $c_u \leftarrow 0$ for any node $u \in T \backslash \{r\}$.
3 Declare b_u for any node $u \in T$.
4
5 Event Function `UponCritical()` // *Upon request set becoming critical as per Definition 3.5*
6 $\lfloor$ `Explore`(r)
7
8 Function `Explore`(u)
9   `Open`(u)
10   set $b_u \leftarrow f$
11   **while** $b_u \neq 0$ **and** there remain pending requests in T_u **do**
12   let Q be the set of pending requests in T_u.
13   let $t'_1 \geq t$ be the earliest time in which a request $q \in Q$ satisfies $d_q(t'_1) \geq \delta(v_q, u)$.
14   let $t'_2 \geq t$ be the earliest time in which there exists v such that $p(v) = u$, and a set of
         requests $Q' \subseteq Q \cap T_v$ such that $d_{Q'}(t'_2) \geq \psi_v(Q')$.
15   **if** $t'_1 \leq t'_2$ **then**
16   let $q \in Q$ be the request in the definition of t'_1. let v be the child of u on the path to v_q.
17   call `Invest`$(u,v,\delta(u,v_q))$, and let y be the return value.
18   **if** $c_v = f$ **then** set $c_v \leftarrow 0$ **;** call `Explore`(v).
19   **if** *q is still pending* **then**
20   call `Invest`$(u,v,\delta(u,v_q)-y)$
21   connect q to facility at u
22   **else**
23   let v be as in the definition of t'_2.
24   call `Invest`(u,v,∞).
25   **if** $c_v = f$ **then** set $c_v \leftarrow 0$ **;** call `Explore`(v).
26
27 Function `Open`(u) // *As in Algorithm 1*
28 Function `Invest`(u,v,x) // *As in Algorithm 1*

Lemma 3.9. $\mathrm{ALG}^B + \mathrm{ALG}^C \leq 3 \cdot (D+1) \cdot kf$.

Proving Lemma 3.9 is very similar to proving Lemma 2.7 of the deadline case. Defining cumulative counters as in the deadline case, we can prove Corollary 2.11 holds in the delay case using an identical proof. It remains to show and prove analogues to Propositions 2.12 and 2.13.

Note that the connection costs in `Explore`(u) only occur during iterations of the main loop in which the main **if** condition is entered.

Proposition 3.10 (analogue of Proposition 2.12)**.** *Suppose the function* `Explore`(u) *enters the main **if** condition in an iteration, and let q be the pending request under consideration. Then at least one of the following holds:*

    1. The sum of return values of calls to `Invest` *in that iteration is $\delta(u,v_q)$.*

    2. $b_u = 0$ at the end of the iteration.

(a) Event 1: single request's delay exceeds connection

(b) Event 2: coalition of low-delay requests exceeds

This figure shows the two events considered in the time-forwarding stage of $\texttt{Explore}(u)$. The first event is a request whose delay exceeds the cost of connecting it to u. This event is very similar to the single event in the deadline case.

The second event is a "coalition" Q of low-delay requests under some child node of u, denoted u_1. Though the delay of no single request in the coalition exceeds the cost of its connection to u, the total delay of the coalition exceeds $\psi_{u_1}(Q)$ – the cost of an optimal ancestor-closed solution under u_1 for that coalition (the optimal solution is shown in the dash-dotted rectangle).

Figure 4: Time forwarding conditions of Algorithm 3

3. q is no longer pending after the first call to $\texttt{Invest}$.

Proof. Consider the state after the return of the first call to $\texttt{Invest}$. Either $\texttt{Invest}$ returned $\delta(u, v_q)$, or $b_u = 0$, or $c_v = f$. In the first two cases, we are done. In the third case, $\texttt{Explore}(u)$ then calls $\texttt{Explore}(v)$. If q is connected during $\texttt{Explore}(v)$, we are done. Otherwise, $\texttt{Explore}(u)$ enters the nested **if** condition upon observing that q is still pending.

Denote by y the return value of the first call to $\texttt{Invest}$, and consider the return value of the second call to $\texttt{Invest}$ made in the nested **if**. If the return value is $\delta(u, v_q) - y$, we are done. Otherwise, consider that $c_v = 0$ before the call to $\texttt{Invest}$, and since $\delta(u, v_q) - y \leq f$ it must thus be that $b_u = 0$ after the return of the second call to $\texttt{Invest}$. This concludes the proof. $\square$

Proposition 3.11 (analogue of Proposition 2.13). $\mathrm{ALG}^B + \mathrm{ALG}^C \leq 3 \cdot \sum_{u \in T} \bar{c}_u$

Proof. The costs of the algorithm (both opening and connection) are again contained in calls to $\texttt{Explore}$, as in the proof of Proposition 2.13. Each call to $\texttt{Explore}(u)$ has an opening cost of f.

As for connection costs, note that they only occur in iterations of the main loop in $\texttt{Explore}(u)$ in which the main **if** condition is entered, and not in the **else** condition. In each iteration in which the main **if** condition is entered, a request q is considered, which may be connected to u at cost $\delta(u, v_q)$. Through Proposition 3.10, in each such iteration either the return values of calls to $\texttt{Invest}$ sum to $\delta(u, v_q)$ (and thus b_u decreases by $\delta(u, v_q)$), $b_u = 0$ at the end of the iteration (in which case this is the last iteration), or $\texttt{Explore}(u)$ does not connect q.

21

Since b_u can decrease by at most f, the connection cost of the algorithm is bounded by $f + \delta(u, v_q)$ for q the last request considered, which is at most $2f$.

Noting that the total cost of $\texttt{Explore}(u)$ is at most $3f$, and that $\bar{c}_u$ is raised by f before calling $\texttt{Explore}(u)$ yields the proposition. $\qquad\square$

Proof of Lemma 3.9. Results directly from Proposition 3.11 and Corollary 2.11 (which holds for the delay case as well). $\qquad\square$

Proof of Lemma 3.7. The lemma results directly from Proposition 3.8 and Lemma 3.9. $\qquad\square$

3.3.2 Lower Bounding OPT

To lower bound the cost of the optimum, we prove the following lemma, which is analogous to Lemma 2.26 of the deadline case.

Lemma 3.12. $kf \leq (D+1) \cdot \text{OPT}^B + 2 \cdot \text{OPT}^C + (D+1) \cdot \text{OPT}^D$.

Charging nodes and incurred costs. We again use charging nodes, defined as in the deadline case. However, the charging nodes for the delay case use half-closed intervals instead of the closed intervals of the deadline case. The reason for this is that we do not have the guarantee that only one call to $\texttt{UponCritical}$ is made at a given time, so that using closed intervals would break the analogue of Lemma 2.14 for our case.

Let M be the set of charging nodes. The definitions of $c_b(\mu)$ (buying costs) and $c_c(\mu)$ (connection costs) are identical to the definition in the deadline case. For the delay case, we also define incurring *delay* costs,

Definition 3.13 ($c_d(\mu)$). Let $\mu = (u, [\tau_1, \tau_2))$ be a charging node. Let $c_d(\mu)$ be the *delay cost incurred by* OPT *on* μ, defined to be the total delay cost incurred by OPT on requests in T_u released in $[\tau_1, \tau_2)$.

We write $c(\mu) = c_b(\mu) + c_c(\mu) + c_d(\mu)$.

We use Procedure 2 to create the preflow. However, we give a different definition to λ than in the deadline case. The definition follows.

Definition 3.14 (λ_u^t and λ_μ). For every function call $\texttt{Explore}_t(u)$ for some $u \in T$ and time t, let Q be the set of requests pending in T_u immediately after the return of $\texttt{Explore}(u)$. We define λ_u^t to be the first time $t' \geq t$ in which one of the following conditions occurs:

1. There is a request $q \in Q$ such that $d_q(t') \geq \delta(v_q, u)$.

2. There exists a set of requests $Q' \subseteq Q$ such that $Q' \subseteq T_{u'}$, for some u' a child of u, and also $d_{Q'}(t') \geq \psi_{u'}(Q')$.

Like in the deadline case, we write $\lambda_\mu = \lambda_u^{\tau_1}$ where $\mu = (u, [\tau_1, \tau_2))$.

Lemma 3.15. $\sum_\mu c(\mu) \leq (D+1) \cdot \text{OPT}^B + 2 \cdot \text{OPT}^C + (D+1) \cdot \text{OPT}^D$.

Proof. $\sum_\mu c_b(\mu)$ can be charged to $(D+1) \cdot \text{OPT}^B$ and $\sum_\mu c_c(\mu)$ can be charged to $2 \cdot \text{OPT}^C$ as in Lemma 2.14 (since the intervals are not closed, this improves by a factor of 2). It remains to charge $\sum_\mu c_d(\mu)$ to $(D+1) \cdot \text{OPT}^D$. To do so, observe that the delay incurred by OPT on a request q can only be counted in charging nodes with intervals containing r_q, and defined by a node which is an ancestor of v_q. There are at most $D+1$ such nodes. $\qquad\square$

Observation 3.16. *Let (S, ϕ) be a minimal-cost ancestor-closed solution for Q under u. Then it holds for every $q \in Q$ that $\phi(q)$ the least ancestor of v_q in S.*

Observation 3.17. *Let (S, ϕ) be a minimal-cost ancestor-closed solution for Q under u. Let $u' \in S$ be a descendant of u. Observing the set $Q' = Q \cap T_{u'}$, we have that $(S \cap T_{u'}, \phi \restriction_{Q'})$ is a minimal-cost ancestor-closed solution for Q' under u'.*

Proposition 3.18 (Decomposition of minimum-cost ancestor-closed solutions)**.** *Let (S, ϕ) be a minimum-cost ancestor-closed solution for $Q \subseteq T_u$ under u, and let $\bar{S} \subseteq S$ be the children of u in S. Define $Q_1^{u'} = Q \cap T_{u'}$, and define $Q_2 = Q \setminus \left(\bigcup_{u' \in \bar{S}} Q_1^{u'} \right)$. Then*

$$\psi_u(Q) = \left(\sum_{u' \in \bar{S}} \psi_{u'}(Q_1^{u'}) \right) + f + \sum_{q \in Q_2} \delta(u, v_q)$$

Proof. For every $u' \in \bar{S}$, Observation 3.16 implies that the requests of $Q_1^{u'}$ only connect to facilities in $S \cap T_{u'}$. The opening costs of $S \cap T_{u'}$, plus the connection costs of $Q_1^{u'}$, are exactly $\psi_{u'}(Q_1^{u'})$ according to Observation 3.17, for a total of $\sum_{u' \in \bar{S}} \psi_{u'}(Q_1^{u'})$.

In addition, opening the facility at u costs f. Observation 3.16 implies that the requests of Q_2 are connected to the facility at u, at a total cost of $\sum_{q \in Q_2} \delta(u, v_q)$. This finishes the proof of the proposition. $\square$

Lemma 3.19. $\chi_\mu \geq 0$ *for every* $\mu = (u, [\tau_1, \tau_2]) \in M$. *That is, the preflow* $Z = (G, s, \alpha)$ *defined in Procedure 2 is valid.*

Proof. We observe the following cases for μ.

Case 1: $\texttt{Color}[\mu] = \texttt{Special}$. This case is identical to Case 1 in Lemma 2.24.

Case 2: $\texttt{Color}[\mu] = \mu^\star$ for a charging node $\mu^\star$. Again, this case is similar to Case 2 in Lemma 2.24.

From now on, assume we are not in the previous two cases, and thus $\texttt{Color}[\mu] = \texttt{None}$. Every outgoing edge from μ to some charging node $\mu' = (u', [\tau_1', \tau_2'])$ is created from μ' investing in μ, which means that $\texttt{Explore}_{\tau_1'}(u')$ raised the counter c_u towards $\texttt{Explore}_{\tau_2}(u)$.

Case 3: For every such μ', we have that $\texttt{Explore}_{\tau_1'}(u')$ raised c_u towards $\texttt{Explore}_{\tau_2}(u)$ only through calls to $\texttt{Invest}$ inside the main **if** condition of $\texttt{Explore}$, and *not* through the main **else** condition. In this case, we show that $c_c(\mu) + c_d(\mu) \geq \sum_{\sigma \in E_\mu^-} \alpha(\sigma)$, proving the lemma for this case. The proof is almost identical to the proof of Case 3 of Lemma 2.24, in which we showed for the deadline case that $c_c(\mu) \geq \sum_{\sigma \in E_\mu^-} \alpha(\sigma)$. The argument for the deadline case consisted of finding a set of requests which the optimum had to connect, all released in $[\tau_1, \tau_2)$. The difference between our delay case and the deadline case is that OPT might choose not to connect some of those requests, in which case it must incur a delay cost which is at least its connection cost.

Case 4: There exists an outgoing edge from μ to a charging node $\mu' = (u', [\tau_1', \tau_2'])$, such that $\texttt{Explore}_{\tau_1'}(u')$ raised c_u towards $\texttt{Explore}_{\tau_2}(u)$ through calls to $\texttt{Invest}$ inside the main **else** condition of $\texttt{Explore}$. Let $\texttt{Color}[\mu'] = \mu^\star = (r, [\tau_1^\star, \tau_2^\star])$. Observing that Proposition 2.23 holds for the delay problem as well, and using $\texttt{Color}[\mu] \neq \texttt{Special}$, we have that OPT did not open a facility in T_u during $[\tau_1, \tau_2^\star)$.

Since $\texttt{Explore}_{\tau_1'}(u')$ raised c_u towards $\texttt{Explore}_{\tau_2}(u)$ inside the main **else** condition, there was a set Q of requests pending at τ_1' such that there exists a time $\hat{t} \leq \lambda_{\mu'} \leq t_i$ in which $d_Q(\hat{t}) \geq \psi_u(Q)$. In addition, the main **else** condition is only reached if $t_1' > t_2' = \hat{t}$. Thus, for every request $q \in Q$ we have that $d_q(\hat{t}) < \delta(u', v_q)$.

Observe that every $q \in Q$ is pending at $\tau_1' \leq \tau_2$, and thus released prior to τ_2. Showing that $r_q \geq \tau_1$, together with the fact that OPT did not open a server in T_u during $[\tau_1, \tau_2^\star)$, would yield that OPT either:

- connected q to a facility outside T_u at a cost of at least $\delta(v_q, p(u) = u')$, which is at least $d_q(\hat{t})$, or

- did not connect q until time $\tau_2^\star$, in which case it paid a delay cost of $d_q(\tau_2^\star) \geq d_q(\hat{t})$.

In either case, OPT paid at least $d_q(\hat{t})$ in delay and connection costs on q. Since we have that $r_q \in [\tau_1, \tau_2)$, we have that OPT incurred a cost of $d_q(\hat{t})$ in u due to q. It remains to find a set of such requests $Q' \subseteq Q$ such that $r_q \geq \tau_1$ for every $q \in Q'$, and such that $d_{Q'}(\hat{t}) \geq f$.

Claim: there exists a set of requests $Q' \subseteq Q$ such that $r_q \geq \tau_1$ for every $q \in Q'$, and such that $d_{Q'}(\hat{t}) \geq f$.

Now, since $\texttt{Color}[u] = \texttt{None}$, we have that either $\tau_1 = -\infty$ or $\lambda_\mu > \tau_2^\star$. If $\tau_1 = -\infty$, then $r_q \geq \tau_1$ for every $q \in Q$. Since $d_Q(\hat{t}) \geq \psi_u(Q) \geq f$, choosing $Q' = Q$ completes the proof of the claim.

Otherwise, $\tau_1 \neq -\infty$ and $\lambda_\mu > t_i$. Let (S, ϕ) be the minimal-cost ancestor-closed solution for Q under u. Defining $\bar{S}, Q_2$, and $Q_1^{u'}$ for every $u' \in \bar{S}$ as in Proposition 3.18, we have that

$$
d_Q(\hat{t}) \geq \psi_u(Q) = \left(\sum_{u' \in \bar{S}} \psi_{u'}(Q_1^{u'}) \right) + f + \sum_{q \in Q_2} \delta(u, v_q)
$$

Now, denote by $\hat{Q} \subseteq Q$ the subset of Q that was pending immediately after $\texttt{Explore}_{\tau_1}(u)$. We make the following observations.

1. For every $q \in Q_2$, we have that $d_q(\hat{t}) \geq \delta(v_q, u)$. Otherwise, $Q \backslash \{q\}$ would become critical before $\hat{t}$. But since $\lambda_\mu > \tau_2^\star \geq \hat{t}$, we must have that $q \notin \hat{Q}$. Thus, $Q_2 \cap \hat{Q} = \emptyset$.

2. Writing $\hat{Q}_1^{u'} = \hat{Q} \cap Q_1^{u'}$, we observe that since $\lambda_\mu > \hat{t}$, we have that $d_{\hat{Q}_1^{u'}}(\hat{t}) \leq \psi_{u'}(\hat{Q}_1^{u'}) \leq \psi_{u'}(Q_1^{u'})$

Overall, we get that

$$
d_{\hat{Q}}(\hat{t}) = \sum_{u' \in \bar{S}} d_{\hat{Q}_1^{u'}}(\hat{t}) \leq \sum_{u' \in \bar{S}} \psi_{u'}(Q_1^{u'})
$$

Thus, we have that

$$
d_{Q \backslash \hat{Q}}(\hat{t}) = d_Q(\hat{t}) - d_{\hat{Q}}(\hat{t}) \geq f + \sum_{q \in Q_2} \delta(u, v_q) \geq f
$$

Observing that $r_q \geq \tau_1$ for each $q \in Q \backslash \hat{Q}$ yields the claim, and thus the lemma. $\square$

Lemma 3.20. *For every $i \in [k]$, the charging node $\mu = (r, [t_{i-1}, t_i))$ has $\chi_\mu \geq f$.*

Proof. Observe that $E_\mu^- = \emptyset$. It remains to see that $\sum_{\sigma \in E_\mu^+} \alpha(\sigma) = f$.

If $\texttt{Color}[\mu] \neq \texttt{None}$, this holds similarly to Lemma 2.25.

Otherwise, assume that $\texttt{Color}[\mu] = \texttt{None}$. Since $\texttt{Color}[\mu] \neq \texttt{Special}$, OPT did not open a facility in $[t_{i-1}, t_i)$. We find a set of requests Q' released in $[t_{i-1}, t_i)$ on which OPT incurs at least f delay. The argument that follows is similar to that of Case 4 of Lemma 3.19, the structure of which we repeat for clarity.

We must have that either $t_{i-1} = -\infty$ or $\lambda_\mu > t_i$. Denote by Q the set of requests that triggered the service at t_i. We have that $d_Q(t_i) \geq \psi(Q)$. Observe that $r_q < t_i$ for every $q \in Q$. If $t_{i-1} = -\infty$, then $r_q \geq t_{i-1}$ for every $q \in Q$, and since $\psi(Q) \geq f$ the proof is complete.

Otherwise, $\lambda_\mu > t_i$. Denoting by (S, ϕ) the minimum-cost ancestor-closed solution for Q, we define $\bar{S}$, Q_2 and $Q_1^{u'}$ for every $u' \in S$ as in Proposition 3.18. Proposition 3.18 yields

$$d_Q(t_i) \geq \left(\sum_{u' \in \bar{S}} \psi_{u'}(Q_1^{u'}) \right) + f + \sum_{q \in Q_2} \delta(u, v_q)$$

Define $\hat{Q} \subseteq Q$ to be the subset of Q alive immediately after the return of $\texttt{Explore}_{t_{i-1}}(r)$. Using $\lambda_\mu > t_i$, and choosing $\hat{t} = t_i$, we use an identical argument to Case 4 of Lemma 3.19 to show that

$$d_{\hat{Q}}(t_i) \leq \left(\sum_{u' \in \bar{S}} \psi_{u'}(Q_1^{u'}) \right)$$

Choosing $Q' = Q \backslash \hat{Q}$ completes the proof of lemma, identically to Case 4 of Lemma 3.19. □

We can now prove Lemma 3.12.

of Lemma 3.12. Lemma 3.19 yields that Z is a valid preflow. For $i \in [k]$, let $\mu_i = (r, [t_{i-1}, t_i))$. Using Lemma 3.20 and Proposition 2.18, we have that

$$kf \leq \sum_{i=1}^{k} \chi_{\mu_i} \leq \omega_Z$$

Now observe that $E_s^+ = \emptyset$, and that $\sum_{\sigma \in E_s^-} \alpha(\sigma) = \sum_{\mu \in M} c(\mu)$. Using Lemma 3.15, we obtain

$$kf \leq \omega_Z = \sum_{\sigma \in E_s^-} \alpha(\sigma) = \sum_{\mu \in M} c(\mu) \leq (D+1) \cdot \text{OPT}^B + 2 \cdot \text{OPT}^C + (D+1) \cdot \text{OPT}^D$$

as required. □

Proof of Theorem 3.6. Using Lemmas 3.7 and 3.12 completes the proof. □

4 Online Multilevel Aggregation with Delay

4.1 Problem and Notation

In the online multilevel aggregation with delay problem, requests arrive on the leaves of a rooted tree over time. Each such request accumulates delay until served. At any point in time, an algorithm for this problem may choose to transmit a subtree which contains the root, at a cost which is the weight of that subtree. Any pending requests on a leaf in the transmitted subtree are served by the transmission.

Formally, as in the facility location with delay problem, a request is a tuple $(v_q, r_q, d_q(t))$ where the leaf of the request is v_q, the arrival time of the request is r_q and $d_q(t)$ is the request's delay function. The function $d_q(t)$ is again required to be non-decreasing and continuous.

We observe online multilevel aggregation with delay on a (≥ 2)-HST. We assume, without loss of generality, that only a single edge exits the root node, called the root edge. Otherwise, we operate on each edge that exits the root node separately, as there is no interaction between the subtrees rooted at those edges. We denote the tree by T, and its root edge by r.

For a request q, and a set of edges E we write that $q \in E$ if the leaf edge on which q is released is in E. In accordance, we write $Q \subseteq E$ if $q \in E$ for every $q \in Q$. For a set of pending requests Q at time t, we denote by $d_Q(t)$ the total delay incurred by the requests of Q until time t. We denote by $w(e)$ the weight of an edge, and by $w(E) = \sum_{e \in E} w(e)$ the total weight of a set of edges.

We assume that each request would gather infinite delay if it remains pending forever.

The following notations are similar to those for facility location, but refer to edges instead of nodes.

Definition 4.1 (Similar to Definition 2.1). For every tree edge $e \in T$, we use the following notations:

- For $e \neq r$, we denote by $p(e)$ the parent edge of e in the tree.

- We denote by T_e the subtree rooted at e.

- For a set of requests $Q \subseteq T_e$, we denote by $T_e^Q \subseteq T_e$ the subtree spanned by e and the leaves of Q. We denote $T^Q = T_r^Q$.

- We define the *height* of e, denoted h_e, to be the depth of T_e.

In this section, we prove the following theorem.

Theorem 4.2. *There exists a $O(D^2)$-competitive deterministic algorithm for online multilevel aggregation with delay on any tree of depth D.*

4.2 Algorithm for HSTs

We now present an algorithm for the online multilevel aggregation with delay problem over a (≥ 2)-HST of depth D.

Definition 4.3 (saturation and critical sets). For any edge e, we say that a set of pending requests $Q \subseteq T_e$ *saturates* T_e if $d_Q(t) \geq w(T_e^Q)$. We say that a set of pending requests Q is *critical* at time t if Q saturates the root edge r.

Upon a set of critical requests, the algorithm starts a service. In every service, the algorithm maintains a tree $\mathcal{T}$, which it expands and ultimately transmits.

Definition 4.4 (live cut). At any time during the construction of $\mathcal{T}$, we define the *live cut under* $e \in \mathcal{T}$ to be the set of edges $E = \{e' | e' \in T_e \backslash \mathcal{T} \wedge p(e') \in \mathcal{T}\}$.

Algorithm's description. The algorithm is given in Algorithm 4. When a set of requests is critical, a call is made to `UponCritical`, which resets the tree to transmit $\mathcal{T}$, calls `Explore`(r) to expand $\mathcal{T}$, then transmits $\mathcal{T}$.

The exploration of an edge e adds e to $\mathcal{T}$. It then considers the live cut underneath e, which is the set of potential candidates for expanding $\mathcal{T}$. The exploration forwards time until a set of pending requests saturates $T_{e'}$ for an edge e' in the the live cut. It then invests in raising the counter of e', until either the counter is full (which triggers `Explore`(e') immediately) or `Explore`(e) is out of budget. The counter of e, as well as the budget of `Explore`(e), is equal to $w(e)$.

Note that the live cut under e can change significantly after every iteration of the loop in `Explore`(e), as making a recursive call to `Explore`(e') can add many additional edges to $\mathcal{T}$.

Algorithm 4: Online Multilevel Aggregation with Delay

---

1 Initialization.
2 Initialize $c_e \leftarrow 0$ for any edge $e \in T \backslash \{r\}$
3 Declare b_e for every edge $e \in T$.
4 Declare $\mathcal{T}$.

5

6 Event Function `UponCritical()` // *Upon request set becoming critical as per Definition 4.3*
7   set $\mathcal{T} \leftarrow \emptyset$
8   `Explore`(r)
9   transmit $\mathcal{T}$

10

11 Function `Explore`(e)
12   `Add`(e)
13   set $b_e \leftarrow w(e)$
14   **while** $b_e \neq 0$ **and** there remain pending requests in T_e **do**
15   let H be the live cut under e.
16   let Q be the set of pending requests in T_e.
17   let t' be the earliest time such that there exists a set of requests $Q' \subseteq Q$ that saturates $T_{e'}$ for
            some $e' \in H$.
18   call `Invest`(e,e')
19   **if** $c_{e'} = w(e')$ **then** set $c_{e'} = 0$; call `Explore`(e').

4.3 Analysis

Fix any instance of online multilevel aggregation with delay, and observe the behavior of Algorithm 4 for that instance. We denote by ALG the algorithm's total cost. We also define ALG^B to be the algorithm's buying cost, and ALG^D to be the algorithm's delay cost, such that $\text{ALG} = \text{ALG}^B + \text{ALG}^D$. We similarly define OPT, OPT^B and OPT^D for the optimal solution for the instance.

In this subsection, we prove the following theorem.

Theorem 4.5. $\text{ALG} \leq O(D) \cdot \text{OPT}^B + O(D^2) \cdot \text{OPT}^D$

In the following analysis, we denote by k the number of times that the algorithm transmits a tree. We also denote the times of the k transmissions by $t_1, ..., t_k$ in increasing order.

4.3.1 Upper Bounding ALG

We upper bound the cost of the algorithm by proving the following lemma.

Lemma 4.6. $\text{ALG} \leq 2kDw(r)$

The main technique used in proving Lemma 4.6 is constructing a preflow to provide an upper bound for ALG^B. Bounding ALG^D by ALG^B then yields the lemma.

Observation 4.7. *Every call to* `Explore`(r) *serves at least one pending request.*

Proposition 4.8. *Every request is eventually served.*

Algorithm 4: Online Multilevel Aggregation with Delay (cont.)

---

1 Function Add(e)
2 $\quad \mathcal{T} \leftarrow \mathcal{T} \cup \{e\}$
3 $\quad$ **if** e *is a leaf edge* **then** mark all pending requests on e as served

4

5 Function Invest(e,e')
6 $\quad$ let $y \leftarrow \min(b_e, w(e') - c_{e'})$
7 $\quad$ increase $c_{e'}$ by y
8 $\quad$ decrease b_e by y
9 $\quad$ **return** y.

Proof. Consider a request q. As assumed in the model, the delay of q goes to infinity as q remains pending. But at some point, the delay of q would exceed $T_r^{\{q\}}$, making $\{q\}$ critical, and triggering calls to Explore(r) until q is served. Each such call serves at least one pending request due to Observation 4.7, and thus q will eventually be served. $\qquad\square$

The following observation follows from the fact that a tree is transmitted whenever a set of requests becomes critical.

Observation 4.9. *At any time t during the algorithm, and for any set of requests Q pending at t, it holds that $d_Q(t) \leq w(T^Q)$.*

Lemma 4.10. $\mathrm{ALG}^D \leq \mathrm{ALG}^B$.

Proof. Denote by $\mathcal{Q}$ the set of all requests released in the instance. Through Proposition 4.8, we can partition $\mathcal{Q}$ into the sets of requests Q_i, for $i \in k$, such that Q_i is served in the i'th service. Denote by T_i the tree bought by the algorithm in the i'th service, and denote by $d(Q_i)$ the total delay incurred by the algorithm on the requests of Q_i. To prove the lemma, it is enough to show that $d(Q_i) \leq w(T_i)$ for every $i \in [k]$.

Now, observe that since all of Q_i are served in t_i. Therefore, $d(Q_i) = d_{Q_i}(t_i)$. Since transmitting T_i serves Q_i, we have that $T^{Q_i} \subseteq T_i$. Using Observation 4.9, we have that $d(Q_i) \leq w(T_i)$ as required. $\qquad\square$

It remains to bound ALG^B.

Let V be the set of calls to Explore made by the algorithm. Observe that in the algorithm, whenever an edge e is bought, a call to Explore(e) is made immediately afterwards. Therefore, we have that $\mathrm{ALG}^B = \sum_{\mathrm{Explore}_\tau(e) \in V} w(e)$.

In addition, immediately prior to calling Explore(e) the counter c_e is zeroed. We say that $\mathrm{Explore}_{\tau_1}(e_1)$ *invested* x in $\mathrm{Explore}_{\tau_2}(e_2)$ if $\mathrm{Explore}_{\tau_1}(e_1)$ raised c_{e_2} by x, such that the next zeroing of c_{e_2} triggers $\mathrm{Explore}_{\tau_2}(e_2)$.

We now construct a graph $G = (V \cup \{s\}, E)$ and a weight function $\alpha : E \to \mathbb{R}^+$, such that $Z = (G, s, \alpha)$ is a preflow. We construct E and α in the following way:

1. For every $j \in [k]$, and for every root function call $\mathrm{Explore}_\tau(r)$, add to E an edge σ from s to $\mathrm{Explore}_\tau(r)$, and set $\alpha(\sigma) = D \cdot w(r)$.

2. For every function call $\mathtt{Explore}_\tau(e) \in V$, and for each function call $\mathtt{Explore}_{\tau'}(e') \in V$ that invested some amount x in $\mathtt{Explore}_\tau(e)$, we add to E an edge σ from $\mathtt{Explore}_{\tau'}(e')$ to $\mathtt{Explore}_\tau(e)$, and set $\alpha(\sigma) = h_e \cdot x$.

Lemma 4.11. *For every $v = \mathtt{Explore}_\tau(e) \in V$ we have that $\chi_v \geq w(e)$, implying that Z is a valid preflow.*

Proof. We first claim that $\sum_{\sigma \in E_v^+} \alpha(\sigma) \geq h_e \cdot w(e)$. If $e = r$, this is true since there exists an edge σ from s to $\mathtt{Explore}_\tau(e)$ such that $\alpha(\sigma) = Dw(r) \geq h_e \cdot w(e)$.

Otherwise, observe that the total amount invested in $\mathtt{Explore}_\tau(e)$ is exactly $w(e)$, and thus $\sum_{\sigma \in E_v^+} \alpha(\sigma) \geq h_e \cdot w(e)$.

Now, observe that $\mathtt{Explore}_\tau(e)$ invests at most $w(e)$ in counters for edges of height at most $h_e - 1$, and thus $\sum_{\sigma \in E_v^-} \alpha(\sigma) \leq (h_e - 1) \cdot w(e)$. Combining this with the previous claim, we get $\chi_v \geq w(e)$ as required. $\qquad\square$

We can now prove Lemma 4.6.

of Lemma 4.6. Observe the preflow Z. Note that

$$\omega_Z = \sum_{\sigma \in E_s^-} \alpha(\sigma) = kDw(r)$$

Using Lemmas 4.11 and 2.18, we have

$$\mathrm{ALG}^B = \sum_{\mathtt{Explore}_\tau(e) \in V} w(e) \leq \sum_{v \in V} \chi_v \leq \omega_Z = kDw(r)$$

Using Lemma 4.10, we get that $\mathrm{ALG} \leq 2kDw(r)$ as required. $\qquad\square$

4.3.2 Lower Bounding OPT

The following lemma provides a lower bound on the cost of the optimum.

Lemma 4.12. $kw(r) \leq \mathrm{OPT}^B + D \cdot \mathrm{OPT}^D$.

Charging nodes and incurred costs. We now define charging nodes for the analysis of our algorithm. The charging nodes are tuples of the form $(e, [\tau_1, \tau_2))$, such that τ_1 and τ_2 are two subsequent times in which the edge e is bought. As in the facility location case, we allow $\tau_1 = -\infty$ and $\tau_2 = \infty$.

For a charging node $\mu = (e, [\tau_1, \tau_2))$ we say that:

- OPT incurs a *buying cost* of $w(e)$ in μ if OPT bought the edge e during $[\tau_1, \tau_2)$. We denote the buying cost that OPT incurs in μ by $c_b(\mu)$.

- OPT incurs a *delay cost* in μ equal to the delay incurred by OPT on the set of requests $Q = \{q \in T_e | r_q \in [\tau_1, \tau_2)\}$. We denote the delay cost that OPT incurs in μ by $c_d(\mu)$.

We denote the total cost that OPT incurs in μ by $c(\mu) = c_b(\mu) + c_d(\mu)$.

Denote by M the set of all charging nodes. To prove Lemma 4.12, we show a preflow on the set of vertices $M \cup \{s\}$, where s is the source node.

The following definition of charging node investment is very similar to the definition for the facility location case.

Procedure 5: PreflowBuilder - Online Multilevel Aggregation with Delay

1 Initialization.

2 Let the set of vertices be $M \cup \{s\}$, and initialize the edge set to be $E = \emptyset$.

3 Initialize dictionary $\texttt{Color}[w] = None$ for every $\mu \in M$.

4 foreach $\mu = (e, [\tau_1, \tau_2)) \in M$ *such that* OPT *transmitted edge* e *during* $[\tau_1, \tau_2)$ **do**

5 $\quad$ set $\texttt{Color}[\mu] \leftarrow \texttt{Special}$

6 foreach $\mu \in M$ *such that* $c(\mu) > 0$ **do**

7 $\quad$ add a new edge $\sigma = (s, \mu)$ to E, and set $\alpha(\sigma) = c(\mu)$

8

9 Function $\texttt{PreflowBuilder}()$

10 $\quad$ **for** i *from* 1 *to* k **do**

11 $\quad\quad$ let $\mu \leftarrow (r, [t_{i-1}, t_i))$

12 $\quad\quad$ $\texttt{SetColor}(\mu, \mu)$

13 $\quad$ **for** j *from* 2 *to* D **do**

14 $\quad\quad$ **foreach** $\mu = (e, [\tau_1, \tau_2)) \in M$ *such that* e *is of depth* j **do**

15 $\quad\quad\quad$ **foreach** *edge* $\sigma \in E_\mu^-$ *incoming to a node* μ' **do**

16 $\quad\quad\quad\quad$ **if** $\texttt{SetColor}(\mu, Color[\mu']) \neq None$ **then break**

17 Function $\texttt{SetColor}(\mu, \mu^\star)$ *// As in Procedure 2*

Definition 4.13 (Investing). For two charging nodes $\mu_1 = (e_1, [\tau_1^1, \tau_2^1))$ and $\mu_2 = (e_2, [\tau_1^2, \tau_2^2))$, such that e_1 is an ancestor of e_2, we say that μ_1 *invested* x *in* μ_2 if $\texttt{Explore}_{\tau_1^1}(e_1)$ raised the counter c_{e_2} by x, through calls to $\texttt{Invest}$, during the counter phase of c_{e_2} between τ_1^2 and τ_2^2.

Definition 4.14 (λ_e^t and λ_μ). For every function call $\texttt{Explore}_t(e)$ for some edge $e \in T$ and time t, let Q be the set of requests pending in T_e immediately after the return of $\texttt{Explore}_t(e)$. We define λ_e^t to be the first time $t' \geq t$ such that there exists $Q' \subseteq Q$ such that $d_t(Q') \geq w(T_e^{Q'}) - w(e)$.

For a charging node $\mu = (e, [\tau_1, \tau_2))$ such that $\tau_1 \neq -\infty$, we write $\lambda_\mu = \lambda_e^{\tau_1}$.

Possible edges. We describe the set of possible edges in G from nodes in M to other nodes in M, denoted by $\bar{E}$, and the weight function $\alpha : \bar{E} \to \mathbb{R}^+$. The final set of edges added to G by Procedure 5 from the nodes of M to themselves is a subset of $\bar{E}$. The set $\bar{E}$ contains an edge σ from any charging node $\mu_1 \in M$ to any charging node $\mu_2 \in M$ if μ_1 invested in μ_2. We set the weight $\alpha(\sigma)$ to be the amount that μ_1 invested in μ_2.

We can now construct the preflow required for the analysis using Procedure 5. This procedure is very similar to Procedure 2, used for analysis of our algorithms for facility location. It uses the function $\texttt{SetColor}$ as defined in Procedure 2.

The procedure for the construction is given in Procedure 5 very similar to that given in Procedure 2.

Definition 4.15 (Cut). We say that a set of edges $H \subseteq T$ is a *cut* if no edge in H is an ancestor of another edge in H.

It is easy to verify that any live cut is a cut.

Proposition 4.16. *Let e be an edge, and let H be a cut in T_e that does not include e. Let $Q \subseteq \bigcup_{h \in H} T_h$ be a set of pending requests and t be a time such that $d_Q(t) \geq w(T_e^Q) - w(e)$. Then there exists an $h \in H$ and a subset $Q_h \subseteq Q$ such that $Q_h \subseteq T_h$ and $d_{Q_h}(t) \geq w(T_h^{Q_h})$.*

Proof. Partition Q into $|H|$ disjoint sets Q_h for every $h \in H$, according to the subtree T_h in which the requests are. Now, observe that $w(T_e^Q) \geq w(e) + \sum_{h \in H} w(T_h^{Q_h})$. We thus have that

$$\sum_{h \in H} d_{Q_h}(t) = d_Q(t) \geq w(T_e^Q) - w(e) \geq \sum_{h \in H} w(T_h^{Q_h})$$

and thus there exists $h \in H$ such that $d_{Q_h}(t) \geq w(T_h^{Q_h})$, as required. $\qquad\square$

Proposition 4.17. *Observe the function call $\texttt{Explore}_t(e)$, and let P be the set of times chosen as t' in $\texttt{Explore}_t(e)$. Then for every $t' \in P$ we have that $t' \leq \lambda_e^t$.*

Proof. Fix some point during the execution of $\texttt{Explore}_t(e)$. Denote by Q the set of pending requests in T_e, and let $\lambda \geq t$ be the first time such that there exists $Q' \subseteq Q$ for which $d_{Q'}(\lambda) \geq w(T_e^{Q'}) - w(e)$. Observe the next time chosen as t' in $\texttt{Explore}_t(e)$. Observe that the subtrees rooted in edges of the current live cut contain all of Q. Thus, using Proposition 4.16, we obtain that $t' \leq \lambda$.

Since during $\texttt{Explore}_t(e)$ requests are being served but do not arrive, we have that λ only increases during $\texttt{Explore}_t(e)$. Since the final value of λ is λ_e^t, for every $t' \in P$ we have $t' \leq \lambda_e^t$ as required. $\qquad\square$

Proposition 4.18. *For every charging node $\mu = (e, [\tau_1, \tau_2)) \in M$, it holds that $\sum_{\sigma \in E_\mu^-} \alpha(\sigma) \leq w(e)$.*

Proof. Observe that an outgoing edge σ from μ only goes to a node μ' that invested in μ, and is labeled $\alpha(\sigma) = x$ where x is the amount that μ' invested in μ. These amount sum to at most $w(e)$, since the counter c_e can only reach $w(e)$ before it is zeroed and e is bought (thus ending the counter phase $[\tau_1, \tau_2)$). $\qquad\square$

Corollary 4.19. *Every charging node $\mu \in M$ such that $\sum_{\sigma \in E_\mu^+} \alpha(\sigma) \geq w(e)$ has that $\chi_\mu \geq 0$.*

The following observation results from the condition checks in $\texttt{SetColor}$.

Observation 4.20. *For any node $\mu = (e, [\tau_1, \tau_2))$ such that $\texttt{Color}[\mu] = \mu^\star$ for some charging node $\mu^\star$, we have that $\tau_1 \neq -\infty$, and also $\lambda_\mu < \infty$.*

The following Proposition is analogous to Proposition 2.23, and its proof is identical.

Proposition 4.21. *Let $\mu = (e, [\tau_1, \tau_2))$ such that $\texttt{Color}[\mu] = \mu^\star$ for some charging node $\mu^\star = (r, [\tau_1^\star, \tau_2^\star))$. Then OPT did not transmit e during $[\tau_1, \tau_2^\star)$.*

Lemma 4.22. *The preflow defined by Procedure 5 is valid.*

Proof. We need to show that $\chi_\mu \geq 0$ for every $\mu = (e, [\tau_1, \tau_2)) \in M$.
We consider the following cases:

Case 1: $\texttt{Color}[\mu] = \texttt{Special}$. In this case, we have that $c(\mu) \geq c_b(\mu) = w(e)$. Observe that an edge σ from s to μ is created with $\alpha(\sigma) = c(\mu)$, and thus from Corollary 4.19 we have that $\chi_\mu \geq 0$.

Case 2: $\texttt{Color}[\mu] = \mu^\star$, for some charging node $\mu^\star$. Using Observation 4.20, observe that $\tau_1 \neq -\infty$ and $\lambda_\mu < \infty$, and thus the call $\texttt{Explore}_{\tau_1}(e)$ has raised counters by exactly $w(e)$, and thus μ has invested a total of $w(e)$ in other charging nodes. Thus, in $\sum_{\sigma \in \bar{E}_\mu^+} \alpha(\sigma) = w(e)$. Observe that $\texttt{SetColor}$ added the edges of $\bar{E}_\mu^+$ to E, and thus $\sum_{\sigma \in E_\mu^+} \alpha(\sigma) \geq w(e)$. Using Corollary 4.19, we have that $\chi_\mu \geq 0$.

Case 3: $\texttt{Color}[\mu] = \texttt{None}$. If there are no outgoing edges from μ, then clearly $\chi_\mu \geq 0$ and we are done. Otherwise, there exists an outgoing edge σ to some node $\mu' = (e', [\tau_1', \tau_2'])$ with $\texttt{Color}[\mu'] = \mu^\star$, for some charging node $\mu^\star$. Denote $\mu^\star = (r, [\tau_1^\star, \tau_2^\star])$, and observe that since μ' invested in μ, we must have that $\tau_1' \leq \tau_2$. Using Proposition 4.21, and the fact that $\texttt{Color}[\mu] \neq \texttt{Special}$, we have that OPT did not transmit e during $[\tau_1, \tau_2^\star)$.

Claim – There exists a set of requests $Q' \subseteq T_e$ such that $r_q \in [\tau_1, \tau_2)$ such that $d_{Q'}(\tau_2^\star) \geq w(e)$.

Proof of Claim. Since $\texttt{Color}[\mu'] = \mu^\star$, it must be that $\tau_1' \neq -\infty$ and $\lambda_{\mu'} \leq \tau_2^\star$. Since μ' invested in μ, we have that at some point during $\texttt{Explore}_{\tau_1'}(e')$, e was in the live cut under e', and the algorithm detected a set of pending requests $Q \subseteq T_e$ such that $d_Q(\hat{t}) \geq w(T_e^Q)$ for some time $\hat{t} \geq \tau_1'$. From Proposition 4.17, we have that $\hat{t} \leq \lambda_\mu \leq \tau_2^\star$. Note also that since Q is pending at τ_1', we have that $r_q < \tau_1' \leq \tau_2$. for every $q \in Q$.

Now observe that since $\texttt{Color}[\mu'] = \mu^\star$, $\texttt{Color}[\mu] = \texttt{None}$, and there exists an edge from μ to μ', we must have that $\texttt{SetColor}(\mu, \mu^\star)$ was called. Since $\texttt{Color}[\mu] = \texttt{None}$, it must be that either $\tau_1 = -\infty$ or $\lambda_\mu > \tau_2^\star$.

If $\tau_1 = -\infty$, then $r_q \geq \tau_1$ for every $q \in Q$. Combining this with $r_q < \tau_2$, we have that $r_q \in [\tau_1, \tau_2)$ for every $q \in Q$. We also have that $d_Q(\tau_2^\star) \geq d_Q(\hat{t}) \geq w(T_e^Q) \geq w(e)$, by Q's definition. Thus choosing $Q = Q'$ proves the claim for this case.

Otherwise, we have that $\tau_1 \neq -\infty$ and $\lambda_\mu > \tau_2^\star$. We denote by $\hat{Q} \subseteq Q$ the subset of requests pending immediately after the return of $\texttt{Explore}_{\tau_1}(e)$. By the definition of λ_μ, and since $\hat{t} < \lambda_\mu$, we have that $d_{\hat{Q}}(\hat{t}) < w(T_e^{\hat{Q}}) - w(e)$. Thus,

$$\begin{aligned} d_{Q \setminus \hat{Q}}(\tau_2^\star) &\geq d_{Q \setminus \hat{Q}}(\hat{t}) = d_Q(\hat{t}) - d_{\hat{Q}}(\hat{t}) \\ &\geq w(T_e^{\hat{Q}}) - \left(w(T_e^{\hat{Q}}) - w(e) \right) = w(e) \end{aligned}$$

Denote $Q' = Q \setminus \hat{Q}$. The requests of Q' were not pending immediately during $\texttt{Explore}_{\tau_1}(e)$, and therefore $r_q \geq \tau_1$ for any $q \in Q'$. As seen before, for every $q \in Q$ we have that $r_q < \tau_2$, and thus for any $q \in Q'$ we have $r_q \in [\tau_1, \tau_2)$. Q' therefore proves the claim. $\square$

We now use the claim. As shown before, OPT did not buy e during $[\tau_1, \tau_2^\star)$, and has therefore did not serve any request from Q' until time $\tau_2^\star$. Therefore, OPT incurs a delay cost of $w(e)$ at μ on the requests of Q', and thus $c(\mu) \geq w(e)$. Observe that an edge σ from s to μ is created with $\alpha(\sigma) = c(\mu)$, and thus Corollary 4.19 implies that $\chi_\mu \geq 0$. This concludes the proof of Lemma 4.22. $\square$

Lemma 4.23. *For every* $j \in [k]$, *the charging node* $\mu = (r, [t_{j-1}, t_j])$ *has that* $\chi_\mu \geq w(r)$.

Proof. We denote $\tau_1 = t_{j-1}, \tau_2 = t_j$. Observe that no other nodes invest in μ, and thus $E_\mu^- = \emptyset$. It remains to show that $\sum_{e \in E_\mu^+} \alpha(e) \geq w(e)$.

If $\texttt{Color}[\mu] \neq \texttt{None}$, then identically to Cases 1 and 2 of Lemma 4.22, we have that $\sum_{e \in E_\mu^+} \alpha(e) \geq w(e)$. This completes the proof for these cases.

If $\texttt{Color}[\mu] = \texttt{None}$, then we have a very similar proof to case 3 of Lemma 4.22. Observe the pending requests Q that became critical at τ_2, triggering the service. Clearly, $r_q < \tau_2$ for every $q \in Q$. Observe that $\texttt{SetColor}(\mu, \mu)$ was called, yet $\texttt{Color}[\mu] = \texttt{None}$. Thus, it must be that either $\tau_1 = -\infty$ or $\lambda_\mu > \tau_2$. To complete the proof, we need the following claim.

Claim – there exists a set of requests Q' such that $r_q \in [\tau_1, \tau_2)$ for every $q \in Q'$, and $d_{Q'}(\tau_2) \geq w(r)$.

Proof of Claim. Observe the two cases of the claim. If $\tau_1 = -\infty$, then $r_q \in [\tau_1, \tau_2)$ for every $q \in Q$. Together with the fact that $d_Q(\tau_2) \geq w(T^Q) \geq w(r)$, choosing $Q' = Q$ proves the claim.

Otherwise, $\tau_1 \neq -\infty$ and $\lambda_\mu > \tau_2$. In this case, denote by $\hat{Q} \subseteq Q$ the requests of Q pending immediately after $\texttt{Explore}_{\tau_1}\{r\}$ and observe, as in Case 3 of Lemma 4.22, that $d_{\hat{Q}}(\tau_2) \leq w(T^{\hat{Q}}) - w(r) \leq w(T^Q) - w(r)$. Thus, we have that $d_{Q \setminus \hat{Q}}(\tau_2) \geq w(r)$. Observe that $r_q \geq \tau_1$ for every $q \in Q \setminus \hat{Q}$, and thus $r_q \in [\tau_1, \tau_2)$ for every $q \in Q \setminus \hat{Q}$. Thus choosing $Q' = Q \setminus \hat{Q}$ yields the claim. $\qquad\square$

Now, observe that $\texttt{Color}[\mu] \neq \texttt{Special}$ and thus OPT did not transmit e during $[\tau_1, \tau_2)$. Using the claim, OPT incurred delay cost of at least $w(r)$ on μ due to Q'. Thus $c(\mu) \geq w(r)$, and thus $\sum_{e \in E_\mu^+} \alpha(e) \geq w(r)$, completing the proof of the lemma. $\qquad\square$

Proposition 4.24. $\omega_Z \leq \mathrm{OPT}^B + D \cdot \mathrm{OPT}^D$

Proof. Observe that $E_s^+ = \emptyset$, and that for every $\sigma \in E_s^-$ to a node $\mu \in M$ we have that $\alpha(\sigma) = c(\mu)$. Therefore, $\omega_Z = \sum_{\mu \in M} c(\mu)$.

Observe that for buying an edge e at time t, OPT incurs buying cost only at the unique charging node $(e, [\tau_1, \tau_2))$ such that $t \in [\tau_1, \tau_2)$.

In addition, when OPT incurs delay for a request q released on leaf edge e, it incurs delay cost in at most D charging nodes, of the form $(e', [\tau_1, \tau_2))$ such that $r_q \in [\tau_1, \tau_2)$ and e' is an ancestor of e.

Thus, $\sum_{\mu \in M} c(\mu) \leq \mathrm{OPT}^B + D \cdot \mathrm{OPT}^D$, proving the proposition. $\qquad\square$

We can now prove Lemma 4.12.

Proof (of Lemma 4.12). Observe the set of charging nodes $N = \{(r, [t_{j-1}, t_j) | j \in [k]\}$. Using Lemma 4.23, we have that $\sum_{\mu \in N} \chi_\mu \geq kw(r)$.

We now use Propositions 2.18 and 4.24 to obtain

$$kw(r) \leq \omega_Z \leq \mathrm{OPT}^B + D \cdot \mathrm{OPT}^D$$

proving the lemma. $\qquad\square$

We now prove the main theorem for this subsection.

Proof of Theorem 4.5. The theorem results immediately from Lemmas 4.6 and 4.12. $\qquad\square$

4.4 From HSTs to General Trees

In this subsection, we show how to extend our result for multilevel aggregation on (≥ 2)-HSTs to general trees, thus proving Theorem 4.2. To do so, we use a similar method to that used in [13] to form a virtual forest of (≥ 2)-HSTs, based on the edges of the original tree.

The decomposition. Let T be the tree, with general weights, rooted at root edge r. We create a forest, the edges of which are the edges of T.

Definition 4.25 (parenthood in virtual (≥ 2)-HST). For every edge e, we define $p'(e)$, the *virtual parent* of e, to be the least ancestor e' of e in T such that $w(e) \leq 2w(e')$. If there is no such e', then e is *the root edge* of a virtual tree in the forest.

This figure shows an example of a forest decomposition. The original tree is on the left, and the virtual forest of (≥ 2)-HSTs resulting from the decomposition in given in the dash-dotted box on the right.

Figure 5: Forest Decomposition

We define the forest according to the function p'. Observe that each connected component is indeed a tree, and specifically a (≥ 2)-HST. Denote by $T^1, ..., T^m$ the virtual trees formed from T, and denote by r^i the root edge of T^i

Let I be an instance of online multilevel aggregation with delay. We partition the requests of I to $I^1, ..., I^m$, such that a request belongs to I^i if the leaf edge $v_q \in I^i$.

We denote by OPT_i the optimal solution for the multilevel aggregation instance I_i in the virtual tree T_i. Using an identical argument to Observation 4.2 in [13], we have the following observation.

Observation 4.26. $\text{OPT} \geq \sum_{i=1}^{m} \text{OPT}_i$

Definition 4.27. Let $e \in T_i$. We define B_e to be the set of edges in T on the path from e to $p'(e)$ (including e, not including $p'(e)$). If $e = r^i$, then let B_e be all the edges from e to r, including r.

Definition 4.28. Let $\mathcal{T}_i$ be some transmittable subtree in T_i for any i. We define $\bar{T}_i = \bigcup_{e \in \mathcal{T}_i} B_e$ to be the *concretization* of $\mathcal{T}_i$.

The algorithm. We now describe the algorithm for online multilevel aggregation with delay on a general tree. The algorithm is:

1. Run Algorithm 4 for each of $T_1, ..., T_m$ separately.

2. Whenever the instance of Algorithm 4 for T_i transmits the virtual subtree $\mathcal{T}_i$, transmit its concretization $\bar{\mathcal{T}}_i$.

Observe that any transmission made by the main algorithm indeed serves the same requests as the original, virtual transmission. We denote by ALG_i the virtual cost of the (≥ 2)-HST algorithm for T_i – that is, the delay of the requests of I_i plus the sum of the costs of virtual transmissions triggered by the (≥ 2)-HST algorithm for T_i.

We denote by k_i for $i \in [m]$ the number of transmissions caused by the algorithm for T_i. The following lemma is a restatement of Lemma 4.12.

Lemma 4.29. $k_i w(r^i) \leq \mathrm{OPT}_i^B + D \cdot \mathrm{OPT}_i^D \leq D \cdot \mathrm{OPT}_i$

It remains to bound the cost of the algorithm.

Proposition 4.30. $\mathrm{ALG}^D \leq \mathrm{ALG}^B$

Proof. Observe that $\mathrm{ALG}^D = \sum_{i=1}^m \mathrm{ALG}_i^D$ and that $\mathrm{ALG}^B \geq \sum_{i=1}^m \mathrm{ALG}_i^B$. Thus, we have that

$$\mathrm{ALG}^D = \sum_{i=1}^m \mathrm{ALG}_i^D \leq \sum_{i=1}^m \mathrm{ALG}_i^B \leq \mathrm{ALG}^B$$

where the second inequality is from Lemma 4.10. $\qquad\square$

We denote by $\overline{\mathrm{ALG}}_i^B = \sum_{j=1}^{k_i} w(\overline{\mathcal{T}}_i^j)$ where $\mathcal{T}_i^j$ is the j'th transmission made by the T_i algorithm. Observe that $\mathrm{ALG}^B = \sum_{i=1}^m \overline{\mathrm{ALG}}_i^B$.

The following lemma bounds the cost of the algorithm, and provides the final component for Theorem 4.2.

Lemma 4.31. *For every i, we have that $\overline{\mathrm{ALG}}_i^B \leq 2Dk_i \cdot w(r^i)$.*

Fix $i \in [m]$. We denote by $\mathcal{T}_j$ for $j \in [k_i]$ the j'th virtual transmission made by the T_i-algorithm. For $j \in [k_i]$, we denote by t_j the time of $\mathcal{T}_j$'s transmission.

To prove Lemma 4.31, we construct a preflow, in a similar manner to the proof of Lemma 4.6. However, in this case we also have nodes that correspond to edges that for which `Explore` is not called.

We now describe the construction of the graph $G = (V \cup \{s\}, E)$, and the weight function α, such that $Z = (G, s, \alpha)$ is a preflow. Each vertex in V is of the form (e, j) where $e \in \overline{\mathcal{T}}_j$. To describe the edge set E, we require the following definition.

Definition 4.32 (x-route). Let (e, j), (e', j') be two edges such that e is an ancestor of e', and $j' \geq j$. Denote by $e = e_0, e_1, ..., e_l = e'$ the path from e to e' in T. We define an x-route from (e_1, j_1) to (e_2, j_2) to be the set of the following charging node edges.

1. An edge σ from (e, j) to (e_1, j') with $\alpha(\sigma) = x \cdot h_{e_1}$.

2. For each $\beta \in [l-1]$, an edge σ from (e_β, j') to $(e_{\beta+1}, j')$ with $\alpha(\sigma) = x \cdot h_{e_{\beta+1}}$.

We also define an x-route from s to (e', j') in a similar manner. Let $r = e_1, e_2, ..., e_l = e'$ the path from the root of T to e'. The edges of this x-route are:

1. An edge σ from s to (r, j') with $\alpha(\sigma) = x \cdot D$.

2. For each $\beta \in [l-1]$, and edge σ from (e_β, j') to $(e_{\beta+1}, j')$ with $\alpha(\sigma) = x \cdot h_{e_{\beta+1}}$.

We can now describe E. The edges of E are constructed in the following way:

1. For each $j \in [k_i]$, add to E the edges of a $w(r_i)$-route from s to (r_i, j).

2. For two charging nodes $(e_1, j_1), (e_2, j_2)$ such that $e_1, e_2 \in T_i$, e_1 is an ancestor of e_2 and $\mathtt{Explore}_{t_{j_1}}(e_1)$ invested x in $\mathtt{Explore}_{t_{j_2}}(e_2)$, add to E the edges of an x-route from (e_1, j_1) to (e_2, j_2).

Observation 4.33. *For every two edges $e \in T$, $e' \in T_i$ such that $e \in B_{e'}$ it holds that $w(e) \leq 2w(e')$.*

Lemma 4.34. *For every charging node $\mu = (e, j)$ it holds that $\chi_\mu \geq \frac{w(e)}{2}$.*

Proof. Observe that x-routes do not

It must be that $e \in \bar{\mathcal{T}}_j$. Hence, there exists an edge $e' \in \mathcal{T}_j$ such that $e \in B_{e'}$. Since $e' \in \mathcal{T}_j$, then we are in one of the following cases.

Case 1: $e' = e$, and thus $e' \in \mathcal{T}_j$. It can be shown that $\sum_{\sigma \in E_\mu^+} \alpha(\sigma) \geq w(e) \cdot h_e$, and that $\sum_{\sigma \in E_\mu^-} \alpha(\sigma) \leq w(e) \cdot (h_e - 1)$, similarly to the proof of Lemma 4.11.

Case 2: $e \neq e'$ and $e' \neq r_i$. Thus, $e \notin T_i$. Observe that since $e \notin T_i$, adding any x-route cannot decrease χ_μ. Indeed, adding an x-route can only create an outgoing edge from μ when creating an incoming edge with greater α. Thus, we locate a set of x-routes that increases χ_μ to at least $w(e')$. From Observation 4.33, we get that $w(e') \geq \frac{w(e)}{2}$, proving the lemma.

If $e' = r_i$, then a $w(r_i)$-route is created from s to (r_i, j). Since e is on the path from r to r_i, it must be that the route adds:

- An incoming edge σ to (e, j) with $\alpha(\sigma) \geq h_e \cdot w(r_i)$.

- An outgoing edge σ^- from (e, j) with $\alpha(\sigma) \leq (h_e - 1) \cdot w(r_i)$.

showing that $\chi_\mu \geq w(r_i) \geq \frac{w(e)}{2}$.

Otherwise, $e' \neq r_i$. Observe that any x-route to (e', j) contains μ, and increases χ_μ by at least x (using the same argument as the case for $e' = r_i$). In this case, observe that a total of $w(e')$ has been invested has been invested in (e', j) to trigger $\texttt{Explore}_{t_j}(e')$. This completes the proof. $\qquad\square$

Proof of Lemma 4.31. We have that $\overline{\mathrm{ALG}}_i^B \leq 2Dk_i \cdot w(r_i)$.

Observe the preflow Z as constructed. We have that $\omega_Z = Dk_i w(r_i)$. From Lemma 4.34, and using Proposition 2.18, we have

$$\overline{\mathrm{ALG}}_i^B \leq \sum_{j=1}^{k_i} w(\bar{\mathcal{T}}_j) = \sum_{(e,j) \in V} w(e) = 2 \cdot \sum_{\mu = (e,j) \in V} \chi_\mu \leq 2 \cdot \omega_Z = 2Dk_i w(r_i)$$

$\qquad\square$

Proof of Theorem 4.2. From Lemmas 4.29 and 4.31, we have that for every $i \in [m]$

$$\overline{\mathrm{ALG}}_i^B \leq 2D^2 \mathrm{OPT}_i$$

From Observation 4.26, we have that

$$\mathrm{ALG}^B = \sum_{i=1}^m \overline{\mathrm{ALG}}_i^B \leq 2D^2 \cdot \sum_{i=1}^m \mathrm{OPT}_i \leq 2D^2 \cdot \mathrm{OPT}$$

Using Lemma 4.30, we have that

$$\mathrm{ALG} \leq 2\mathrm{ALG}^B \leq 4D^2 \cdot \mathrm{OPT}$$

as required. $\qquad\square$

5 Online Service with Delay

5.1 Problem and Notation

In the online service with delay (OSD) problem, a single server exists on a point in a metric space. Requests arrive on points of the metric space over time, and accumulate delay until served, where serving a request requires moving the server to that request. The cost of moving the server from one point to another is the distance between those two points in the metric space. The goal is to minimize the sum of the moving cost and the delay cost.

Formally, a request is a tuple $q = (v_q, r_q, d_q(t))$ such that v_q is the point on which q arrives, the request arrives at time r_q, and $d_q(t)$ is an arbitrary non-decreasing continuous delay function. We also assume that $d_q(t)$ tends infinity as time progresses. For any instance of OSD I, denote by ALG^B the total cost of moving the algorithm's server. We also denote by $\mathrm{ALG}^D = \sum_{q \in Q} d_q(t_q)$, where t_q is the time in which the request q is served. Then the algorithm's goal is to minimize the total cost

$$\mathrm{ALG} = \mathrm{ALG}^B + \mathrm{ALG}^D$$

As in the previous problems in this paper, we also consider the special case in which the metric space is the leaves of a (≥ 2)-HST. Without loss of generality, we allow an algorithm to move its server to the internal nodes of the tree, even though they are not a part of the original metric space. This is implemented by lazy moving of the server – that is, the server never really moves to those internal nodes, but its virtual location in an internal node is kept in the algorithm's memory for the sake of calculations.

In this section, we prove the following theorem.

Theorem 5.1. *There exists a randomized $O(\log^2 n)$-competitive algorithm for online service with delay on a general metric space of n points.*

5.2 Algorithm for HSTs

In this subsection, we present an algorithm for online service with delay on (≥ 2)-HSTs. We assume that the weight of each edge is a power of 2 – this can be enforced, at a loss factor of 2 to competitiveness. This algorithm encapsulates our algorithm for online multilevel aggregation with delay, while using similar mechanisms to those in [5].

For an edge e, denote $\mathcal{C}(e) = \{e' | p(e') = p(e) \wedge w(e') < w(e)\}$, the set of sibling edges of e with smaller weight. Note that for every $e' \in \mathcal{C}(e)$ we have $w(e') \leq \frac{1}{2}w(e)$, since edge weights are powers of 2. We define the following.

Definition 5.2 (Top and bottom nodes). For an edge e, we define $v_e^\top$ to be the top node of e, and $v_e^\perp$ to be the bottom node of e.

Definition 5.3 (Relative subtree R_e). For an edge e, we define the *relative subtree of e* to be $\{e\} \cup \bigcup_{e' \in \mathcal{C}(e)} T_{e'}$.

The following definition is required for defining exactly what we mean when referring to locations of servers and requests.

Definition 5.4 (Locations of servers and requests). Consider the location of a server (either the algorithm's or the optimum's).

- For T_e, we say that *the server is internal to T_e* if the server is in one of the nodes of T_e *other than $v_e^\top$*.

- For $R_e = \{e\} \cup \left(\bigcup_{e' \in \mathcal{C}(e)} T_{e'}\right)$, we say that *the server is internal to* R_e if the server in one of the nodes of R_e *other than* $v_e^\perp$.

The same applies for saying that a *request q is internal to T_e (or R_e)*, and writing $q \in T_e$ (or $q \in R_e$).

Let $Q \subseteq R_e$ be a set of requests, and denote by $Q \restriction_{T_{e'}} = \{q \in T_{e'} | q \in Q\}$. Then we define R_e^Q to be

$$\left(\bigcup_{e' \in \mathcal{C}(e)} T_{e'}^{Q \restriction_{T_{e'}}}\right) \cup \{e\}$$

We sometimes write Y_e to make claims that refer to either R_e or T_e.

Definition 5.5 (Saturation). We say that a set of requests $Q \subseteq Y_e$ *saturates* Y_e *at time* t if $d_Q(t) \geq w(Y_e^Q)$.

Definition 5.6 (Major edges). We say that an edge e is *major* at a time t if every edge e' on the path from the algorithm's server to e has that $w(e') \leq w(e)$.

Definition 5.7 (Critical set). We say that a set of requests Q is *critical* at time t if it saturates Y_e at time t for an edge e which is major at time t.

Definition 5.8. Let e be an edge, and Y_e be either T_e or R_e. We say that the algorithm's server is *on the other side of e than Y_e* if:

- The server is internal to T_e and $Y_e = R_e$.

- The server is not internal to T_e and $Y_e = T_e$.

The following proposition allows us to assume that whenever a set of requests is critical by saturating Y_e for a major edge e, we have that the algorithm's server is on the other side of e than Y_e.

Proposition 5.9. *Suppose there exists a critical set of requests Q, saturating Y_e for e a major edge, at some point in time. Then there exists another critical set of requests Q', saturating $Y_{e'}$ for another major edge e', such that the algorithm's server is on the other side of e' than $Y_{e'}$.*

Proof. If the server is on the other side of e than Y_e, we are done. Suppose otherwise, and let Q be the minimal set saturating Y_e.

Consider the case that $Y_e = T_e$, and the algorithm's server is internal to T_e. Note that e cannot be a leaf edge – otherwise, the server and all requests in Q must be on $v_e^\perp$, in contradiction to the requests of Q being pending.

1. If the server is in $v_e^\perp$, we can thus choose e' to be any child edge of e saturated by $Q \restriction_{T_{e'}}$ (such an edge must exist, otherwise Q would not have saturated T_e).

2. If the server is internal to $T_{\hat{e}}$ for some $\hat{e}$ child edge of e, then:

   (a) If there exists a sibling $\tilde{e}$ of $\hat{e}$ such that $w(\tilde{e}) \geq w(\hat{e})$ such that $Q \restriction_{T_{\tilde{e}}}$ is saturated, then $\tilde{e}$ is major, and thus $Q \restriction_{T_{\tilde{e}}}$ is critical. The server is on the other side of $\tilde{e}$ than $T_{\tilde{e}}$, completing the proof.

   (b) If there is no such $\tilde{e}$, by the minimality of Q we have for any $\tilde{e}$ sibling of $\hat{e}$ such that $w(\tilde{e}) \geq w(e)$ that $Q \restriction_{T_{\tilde{e}}} = \emptyset$.

Algorithm 6: Online Service with Delay

---
1 Initialization.
2 Initialize $c_e \leftarrow 0$ for any edge $e \in T \backslash \{r\}$
3

4 Event Function `UponCritical()` // *Upon request set becoming critical as per Definition 5.7*
5   Let Y_e be the tree saturated by the critical requests, such that e is a major edge.
      // *The server is on the other side of e than Y_e, by Proposition 5.9.*
6   let $e = (u_1, u_2)$, such that the server is on u_1's side.
7   move server to u_1.
8   let $\mathcal{T} \leftarrow$ `MultilevelAggregationExplore`(Y_e)
9   traverse $\mathcal{T}$ in DFS order, finishing at u_1.
10   traverse e to reach u_2.

---

     i. If $Q \restriction_{T_{\hat{e}}}$ does not saturate $T_{\hat{e}}$, then again from minimality of Q we have $Q \restriction_{T_{\hat{e}}} = \emptyset$. Thus $Q \subseteq R_{\hat{e}}$. Since $w(R_{\hat{e}}^Q) = w(T_e^Q) - w(e) + w(\hat{e}) \leq w(T_e^Q)$, it holds that Q saturates $\hat{e}$, and is thus critical.

     ii. Otherwise, $Q \restriction_{T_{\hat{e}}}$ saturates $T_{\hat{e}}$, and is thus critical. Since the server is internal to $T_{\hat{e}}$, induction on the height of e yields the proof.

The case that $Y_e = R_e$ and the server is not internal to T_e is very similar. $\qquad\square$

Algorithm's description. The algorithm for service with delay on a (≥ 2)-HST is given in Algorithm 6. The algorithm triggers a service whenever a set of requests becomes critical. We assume that the set of requests considered by the algorithm is always on the other side of the major edge than the server. This assumption uses Proposition 5.9.

Whenever a set of requests becomes critical, saturating Y_e for a major edge e, the algorithm moves the server to the closer node touching e (denoted by u_1). It then calls the exploration function of the multilevel aggregation algorithm for (≥ 2)-HSTs, given in Algorithm 4. To make this well defined, a call to `MultilevelAggregationExplore`(Y_e) observes the (≥ 2)-HST Y_e, in which e is the root edge. If $Y_e = R_e$, then e is "promoted" to be the parent edge of its siblings in R_e for the sake of the multilevel aggregation exploration (note that the resulting tree is indeed a (≥ 2)-HST). The counters used by the exploration are the same counters c_e of the service with delay algorithm.

The exploration of the multilevel aggregation algorithm yields a tree to transmit $\mathcal{T}$. In the case of service with delay, instead of transmitting $\mathcal{T}$, we traverse it with the server, in DFS order, returning to the node u_1. Note that the cost of this is exactly twice the weight of $\mathcal{T}$. To conclude, the server crosses e, ending the service on the other side of e than before the service. Observe that while this concludes the call to `UponCritical`, it may immediately trigger new calls to `UponCritical` due to new edges becoming major in the server's new location.

5.3 Analysis

Fix any instance of online service with delay on the tree T. Define ALG^B and ALG^D to be the total moving cost and the total delay cost of the algorithm on the instance, respectively. Define $\text{ALG} = \text{ALG}^B + \text{ALG}^D$. Define $\text{OPT}^B, \text{OPT}^D$ and OPT similarly for the optimum.

In this subsection, we prove the following theorem.

Theorem 5.10. $\mathrm{ALG} \leq O(D) \cdot \mathrm{OPT}^B + O(D^2) \cdot \mathrm{OPT}^D$.

Observe that upon embedding from a general metric space of n points to a (≥ 2)-HST, the moving cost is distorted but the delay cost is not. Thus, using similar arguments to the proof of Theorem 2.3, we have that Theorem 5.10 implies Theorem 5.1 for general metric spaces.

5.3.1 Upper Bounding ALG

We again denote by k the number of services made by the algorithm. That is, k is the number of calls to `UponCritical`. We denote by e_i for $i \in [k]$ the major edge considered in the i'th service. We also denote by t_i the time of the i'th service.

We devote this part of the analysis to proving the following lemma.

Lemma 5.11. $\mathrm{ALG} \leq O(D) \cdot \sum_{i=1}^{k} w(e_i)$

Observe the operation of the algorithm. Upon a critical set of requests the algorithm calls `UponCritical` a few times consecutively, until there is no critical set of requests with regard to the server's current location. The algorithm then enters the waiting state. We call each such instantaneous set of services a *service phase*. We denote by k' the number of these phases. We also assume that no two sets of requests become critical at the same time, which can easily be enforced by the algorithm by breaking ties arbitrarily.

Proposition 5.12. *Consider the service phase which starts from a set of requests Q becoming critical by saturating Y_e, for a major edge e. Then during the entire phase, the server only serves requests internal to Y_e.*

Proof. The first service in the phase only serves requests internal to Y_e, and the server finishes the service in a point internal to Y_e. We claim that during the rest of the phase, the server remains internal to Y_e, which proves the proposition.

Assume otherwise. Then we must have that at some point during the phase, a set of pending requests Q' is critical (with regards to the server's location at that point in the phase) by saturating $Y_{e'}$ for an edge $e' \notin Y_e$. Consider the first such point during the phase. Due to our assumption that no two sets of requests become critical at the same time, we have that e' must not have been a major edge before the start of the phase. But note that all edges in Y_e have weight at most $w(e)$, and thus the server only traversed edges of weight at most $w(e)$ since the start of the phase. Thus, we must have that $w(e') < w(e)$. Now, note that the server cannot reach any edge e' such that $w(e') < w(e)$ and $e' \notin Y_e$ from a position which is internal to Y_e without traversing an edge of weight at least $w(e)$. This is a contradiction to e' being a major edge. $\square$

Lemma 5.13. $\mathrm{ALG}^D \leq \mathrm{ALG}^B$

Proof. Let $\mathcal{Q}$ be the set of all requests in the instance. Divide $\mathcal{Q}$ into $Q_1, ..., Q_{k'}$ such that Q_i are the requests served by the algorithm in the i'th phase.

Fix the i'th phase, let t be the time of the phase and let $Q = Q_i$. Let Y_e be the saturated tree triggering the phase, with e a major edge. Due to Proposition 5.12, we have that $Q \subseteq Y_e$. Since the algorithm's server is outside Y_e, we have that $w(Y_e^Q)$ is a lower bound for the cost of moving the server to serve Q. Since e is a major edge immediately before the start of the phase, we have that $d_Q(t) \leq w(Y_e^Q)$. Thus the delay incurred by the requests of Q is bounded by the buying cost incurred by the algorithm in the phase.

Summing this conclusion over all phases yields the lemma. $\square$

Proposition 5.14. *Moving the server to touch a major edge e costs at most $2w(e)$.*

Proof. Since we are in a (≥ 2)-HST, the path from any node to another node consists of (at most) one upwards path followed by one downwards path. Since e is a major edge, each edge on the path from the server to e must have weight at most $w(e)$. Thus, the downwards path must be of length 0 – otherwise, it would contain e's parent edge, which has weight larger than $w(e)$. Consider that the weight of the upwards path is at most $2w(e)$. $\qquad\square$

Lemma 5.15. $\mathrm{ALG}^B \leq (2D + 5) \cdot \sum_{i=1}^{k} w(e_i)$

Proof. Each service triggered by the saturation of a major edge e causes a multilevel aggregation service of either T_e or R_e, plus additional server movements required to reach and traverse e. The additional movements are of at most $3w(e)$ (using Proposition 5.14), and thus $3 \cdot \sum_{i=1}^{k} w(e_i)$ over all services.

Using a very similar proof to the case for multilevel aggregation, we can show that the sum of the weight of the trees to "transmit" yielded by the calls to the multilevel aggregation algorithm are at most $(D + 1) \sum_{i=1}^{k} w(e_i)$. Since traversing a tree by DFS is twice the cost of transmission, the buying cost incurred by the OSD algorithm for that step is at most $2(D + 1) \sum_{i=1}^{k} w(e_i)$.

Overall, the buying cost of the algorithm is at most $(2D + 5) \sum_{i=1}^{k} w(e_i)$. $\qquad\square$

of Lemma 5.11. The lemma results directly from Lemmas 5.13 and 5.15. $\qquad\square$

5.3.2 Lower Bounding OPT

Definition 5.16 ($\mathbb{I}_i$)**.** We define the indicator variable $\mathbb{I}_i$ for $i \in [k]$ to be 1 if the optimum's server was on the same side of e_i at t_i as the algorithm's server (before the call to `UponCritical`), and 0 otherwise.

The following lemma provides a lower bound on the cost of the optimum.

Lemma 5.17. $\sum_{i=1}^{k} \mathbb{I}_i \cdot w(e_i) \leq 3 \cdot \mathrm{OPT}^B + 3D \cdot \mathrm{OPT}^D$

Charging nodes and incurred costs. We first define the charging nodes for the analysis of this algorithm. For every edge e, there exist three types of charging nodes:

1. *Standard root charging nodes* (SRCN), which are nodes of the form $(e, [\tau_1, \tau_2))$ where τ_1 and τ_2 are two subsequent times in which $\mathtt{Explore}(e)$ is called due to e being a major edge and T_e being saturated, triggering service.

2. *Relative root charging nodes* (RRCN), which are nodes of the form $(e, [\tau_1, \tau_2))$ where τ_1 and τ_2 are two subsequent times in which $\mathtt{Explore}(e)$ is called due to e being a major edge and R_e being saturated, triggering service.

3. *Normal charging nodes* (NCN), which are nodes of the form $(e, [\tau_1, \tau_2))$ where τ_1 and τ_2 are two subsequent times in which $\mathtt{Explore}(e)$ is called due to the counter c_e reaching $w(e)$.

Nodes of types 1 and 2 correspond to root charging nodes in the multilevel aggregation case, while nodes of type 3 correspond to non-root nodes.

For a charging node $\mu = (e, [\tau_1, \tau_2))$ we say that:

- OPT incurs a *buying cost* of $w(e)$ in μ if OPT traversed the edge e during $[\tau_1, \tau_2)$. We denote the buying cost that OPT incurs in μ by $c_b(\mu)$.

- If μ is an SRCN or an NCN, OPT incurs a *delay cost* in μ equal to the delay incurred by OPT on the set of requests $Q = \{q \in T_e | r_q \in [\tau_1, \tau_2)\}$

- If μ is an RRCN, OPT incurs a *delay cost* in μ equal to the delay incurred by OPT on the set of requests $Q = \{q \in R_e | r_q \in [\tau_1, \tau_2)\}$ <u>if</u> OPT's server remained internal to T_e during $[\tau_1, \tau_2)$.

We denote the total delay cost incurred by OPT in μ be $c_d(\mu)$. We denote the total cost that OPT incurs in μ by $c(\mu) = c_b(\mu) + c_d(\mu)$.

Lemma 5.18. $\sum_{\mu \in M} c(\mu) \geq 3 \cdot \text{OPT}^B + 3D \cdot \text{OPT}^D$

Proof. Observe that any edge traversal by the optimum's server can be counted in three charging nodes relating to that edge (one SRCN, one RRCN and one NCN).

Any delay cost incurred by the optimum due to a request q can be counted in NCNs and SRCNs along the depth of the tree, yielding $2D$ such charging nodes. In addition, the delay of q can be counted in at most D RRCNs along the path from the root to the location of the optimum's server at time r_q.

These observations yield the lemma. $\qquad\square$

Denote by M the set of all charging nodes. To prove Lemma 4.12, we show a preflow on the set of vertices $M \cup \{s\}$, where s is the source node.

The following definition of charging node investment is nearly identical to the definition in the multilevel aggregation case.

Definition 5.19 (Investing). For a charging node $\mu_1 = (e_1, [\tau_1^1, \tau_2^1])$ and an NCN $\mu_2 = (e_2, [\tau_1^2, \tau_2^2])$, we say that μ_1 *invested* x *in* μ_2 if $\texttt{Explore}_{\tau_1^1}(e_1)$ raised the counter c_{e_2} by x during the counter phase $[\tau_1^2, \tau_2^2)$ (not including recursive calls made by $\texttt{Explore}_{\tau_1^1}(e_1)$).

Procedure 7 is used to build the preflow. As in the previous analyses, we define $\bar{E}$ to be the set of possible edges between nodes of M to themselves. As before, an edge σ exists in $\bar{E}$ from a charging node μ to a charging node μ' if μ invested in μ', and $\alpha(\sigma)$ is set to be the total invested amount.

We use the following definition for ease.

Definition 5.20 (Y_μ). For a charging node $\mu = (e, [\tau_1, \tau_2])$, we define Y_μ to be R_e if μ is a RRCN. Otherwise, we define Y_μ to be T_e.

Observation 5.21. *If a node $\mu = (e, [\tau_1, \tau_2])$ invested in a node $\mu' = (e', [\tau_1, \tau_2])$, then $Y_{\mu'} \subseteq Y_\mu$.*

Proposition 5.22 (analogue of Proposition 4.21). *Let $\mu = (e, [\tau_1, \tau_2])$ such that $\texttt{Color}[\mu] = \mu^\star$ for some charging node $\mu^\star = (e^\star, [\tau_1^\star, \tau_2^\star])$. Then OPT did not enter Y_μ during $[\tau_1, \tau_2^\star)$.*

Proof. Since $\texttt{Color}[\mu] = \mu^\star$, we must have that for the RCN $\mu^\star$ we have that $\texttt{Color}[\mu^\star] = \mu^\star$. Thus, we have that $\mathbb{I}_i = 1$ for i such that $\tau_2^\star = t_i$, and thus the optimum's server was on the same side of $e^\star$ as the algorithm's server before the service at $\tau_2^\star$. Since we only consider critical trees on the other side of the major edge, we have that the optimum's server was not internal to $Y_{\mu^\star}$ at time $\tau_2^\star$. Since $\texttt{Color}[\mu^\star] \neq \texttt{Special}$, the optimum's server did not traverse $e^\star$ during $[\tau_1^\star, \tau_2^\star)$, and thus was not internal to $Y_{\mu^\star}$ during $[\tau_1^\star, \tau_2^\star)$.

What follows is a similar inductive argument to that of Proposition 2.23. For the base case that $\mu = \mu^\star$, we are done. We now prove the proposition by induction on the depth of the propagation of the color $\mu^\star$ to μ. Observe that the color $\mu^\star$ was propagated to μ from another charging node $\mu' = (e', [\tau_1', \tau_2'])$. By

---

Procedure 7: PreflowBuilder - Online Service with Delay

---

1 **Initialization.**

2 Let the set of vertices of G be $M \cup \{s\}$, and initialize the edge set to be $E = \emptyset$.

3 Initialize dictionary $\texttt{Color}[w] = \texttt{None}$ for every $\mu \in M$.

4 **foreach** $\mu = (e, [\tau_1, \tau_2)) \in M$ *such that* OPT *traversed edge e during* $[\tau_1, \tau_2)$ **do**

5   set $\texttt{Color}[\mu] \leftarrow \texttt{Special}$

6 **foreach** $\mu \in M$ *such that* $c(\mu) > 0$ **do**

7   add a new edge $\sigma = (s, \mu)$ to E, and set $\alpha(\sigma) = c(\mu)$

8

9 **Function** $\texttt{PreflowBuilder()}$

10   **for** i *from* 1 *to* k **do**

11   let $\mu \leftarrow (e_i, [t_{i-1}, t_i))$ be the RCN of the i'th service.

12   **if** $\mathbb{I}_i = 1$ **then** $\texttt{SetColor}(\mu, \mu)$

13   **for** j *from* 1 *to* D **do**

14   **foreach** *NCN* $\mu = (e, [\tau_1, \tau_2)) \in M$ *such that e is of depth j* **do**

15   **foreach** *edge $\sigma \in E_\mu^-$ incoming to a node μ'* **do**

16   **if** $\texttt{SetColor}(\mu, \texttt{Color}[\mu']) \neq None$ **then break**

17

18 **Function** $\texttt{SetColor}(\mu, \mu^\star)$ *// As in Procedure 2*

---

induction, the optimum's server was not internal to $Y_{\mu'}$ during $[\tau_1', \tau_2^\star)$. From Observation 5.21, we have that the optimum's server was not internal to Y_μ during $[\tau_1', \tau_2^\star)$.

Since $\texttt{Color}[\mu] \neq \texttt{Special}$, the optimum's server did not traverse e during $[\tau_1, \tau_2)$. Since μ' invested in μ, we have that $\tau_1' \leq \tau_2$, and thus the optimum's server was not internal to Y_μ during $[\tau_1, \tau_2^\star)$ as required. $\qquad\square$

Observation 5.23. *Corollary 4.19 from the multilevel aggregation case holds in this case as well. That is, if $\sum_{\sigma \in E_\mu^+} \alpha(\sigma) \geq w(e)$, then $\chi_\mu \geq 0$.*

Lemma 5.24. *The preflow defined by Procedure 7 is valid.*

Proof. As in previous versions of this lemma, we need to show that $\chi_\mu \geq 0$ for every $\mu \in M$. We separate according to cases.

Case 1: $\texttt{Color}[\mu] = \texttt{Special}$. In this case, OPT incurs a buying cost of $w(e)$ at μ, completing the case according to Observation 5.23.

Case 2: $\texttt{Color}[\mu] = \mu^\star$ for some charging node $\mu^\star$. In this case, observe that Observation 4.20 applies for OSD as well. Thus, μ has invested in other nodes a total of exactly $w(e)$, and thus $\sum_{\sigma \in E_\mu^+} \alpha(\sigma) \geq w(e)$. Observation 5.23 completes the proof for this case.

Case 3: $\texttt{Color}[\mu] = \texttt{None}$. If there are no outgoing edges from μ, then clearly $\chi_\mu \geq 0$ and we are done. Otherwise, μ is an NCN, and there exists an outgoing edge σ to some node $\mu' = (e', [\tau_1', \tau_2'))$ with $\texttt{Color}[\mu'] = \mu^\star$ for some charging node $\mu^\star = (e^\star, [\tau_1^\star, \tau_2^\star))$. Observe that since μ' invested in μ, we must have that $\tau_1' \leq \tau_2$. Using Proposition 5.22, and the fact that $\texttt{Bought}[\mu] = False$, we have that OPT was not internal to Y_μ during $[\tau_1, \tau_2^\star)$. As in Case 3 of Lemma 4.22, we locate a set of requests internal to Y_μ due to which OPT incurs delay cost of $w(e)$ in μ.

43

Claim – There exists a set of requests $Q' \subseteq Y_\mu$ such that $r_q \in [\tau_1, \tau_2)$ such that $d_{Q'}(\tau_2^\star) \geq w(e)$.

Proof of claim. Identical to the proof for the corresponding claim in the multilevel aggregation analysis. $\square$

Using the claim, observe that since the optimum's server was not internal to Y_μ during $[\tau_1, \tau_2^\star)$, it has incurred $w(e)$ delay due to the requests of Q'. Due to the definition of delay cost on an NCN, we have that $c_d(\mu) \geq w(e)$. This completes the analysis of the case due to Observation 5.23. $\square$

Lemma 5.25. *For every root charging node $\mu = (e_i, [t_{i-1}, t_i))$ we have that $\chi_\mu \geq \mathbb{I}_i \cdot w(e_i)$.*

Proof. If $\texttt{Color}[\mu] \neq \texttt{None}$, we have that $\chi_\mu \geq w(e_i)$ using identical arguments to Cases 1 and 2 of Lemma 5.24.

Otherwise, $\texttt{Color}[\mu] = None$. Observe that $\chi_\mu \geq 0$, due to Lemma 5.24, which covers the case that $\mathbb{I}_i = 0$. Now, suppose that $\mathbb{I}_i = 1$. We show that OPT incurred a delay cost of at least $w(e_i)$ in μ.

Claim – There exists a set of requests $Q' \subseteq Y_\mu$ such that $r_q \in [t_{i-1}, t_i)$ such that $d_{Q'}(t_i) \geq w(e)$.

Proof of Claim. We denote by Q the set of requests that became critical at t_i, triggering the service. Observe that $d_Q(t_i) \geq w(Y_\mu)$, and that $r_q < t_i$ for every $q \in Q$. Since $\texttt{Color}[\mu] = \texttt{None}$, we must have that either $t_{i-1} = -\infty$ or $\lambda_\mu > t_i$.

If $t_{i-1} = -\infty$, then $r_q \in [t_{i-1}, t_i)$ and choosing $Q' = Q$ yields the claim. Otherwise, $t_{i-1} \neq -\infty$, and $\lambda_\mu > t_i$. In this case, we choose $\hat{Q} \subseteq Q$ to be the set of pending requests immediately after the service at t_{i-1}. Since $\lambda_\mu > t_i$, $d_{\hat{Q}}(t_i) \leq w(Y_\mu^{\hat{Q}}) - w(e_i) \leq w(Y_\mu^Q) - w(e_i)$. Thus, we have that $d_{Q\setminus\hat{Q}}(t_i) \geq w(e_i)$. Observe that $r_q \geq t_{i-1}$ for every $q \in Q\setminus\hat{Q}$, and thus $r_q \in [t_{i-1}, t_i)$ for every $q \in Q\setminus\hat{Q}$. Thus choosing $Q' = Q\setminus\hat{Q}$ yields the claim. $\square$

We now use this claim. Observe that the optimum's server was not internal to Y_μ at t_i (due to $\mathbb{I}_i = 1$), and since $\texttt{Color}[\mu] \neq \texttt{Special}$, the optimum's server was not internal to Y_μ during $[t_{i-1}, t_i)$. Thus, the optimum incurs a delay cost of $w(e_i)$ due to Q'. Now observe that:

- If μ is an SRCN, then $c_d(\mu) \geq w(e_i)$.

- If μ is an RRCN, then the algorithm's server was internal to T_{e_i} at time t_i. Since $\mathbb{I}_i = 1$, the optimum's server was internal to T_{e_i} as well at t_i. Since $\texttt{Color}[\mu] \neq \texttt{Special}$, the optimum's server stayed internal to T_{e_i} during $[t_{i-1}, t_i)$. Thus, $c_d(\mu) \geq w(e_i)$.

In both cases, $c_d(\mu) \geq w(e_i)$, completing the proof of the case and lemma. $\square$

of Lemma 5.17. The proof of the lemma results from observing the subset $N \subseteq M$ of all root charging nodes. Lemma 5.25 implies that $\sum_{\mu \in N} \chi_\mu \geq \sum_{i=1}^{k} \mathbb{I}_i \cdot w(e_i)$.

We now use Proposition 2.18 and Lemma 5.18 to obtain

$$\sum_{i=1}^{k} \mathbb{I}_i \cdot w(e_i) \leq \omega_Z = \sum_{\mu \in M} c(\mu) \leq 3 \cdot \text{OPT}^B + 3D \cdot \text{OPT}^D$$

proving the lemma. $\square$

5.3.3 Proof of Main Theorem

In this part of the analysis, we prove Theorem 5.10.

From Lemma 5.11, we have that $\mathrm{ALG} \leq \gamma D \cdot \sum_{i=1}^{k} w(e_i)$ for some constant γ.

Definition 5.26 (Potential function $\phi(t)$). We define the potential function $\phi(t)$ to be γD times the distance between the algorithm's server and the optimum's server at time t.

Observe that $\phi(-\infty) = 0$.

For every $i \in [k]$, define the difference in potential $\Delta_i \phi = \phi(t_i^+) - \phi(t_i^-)$, where t_i^- is time t_i immediately before the i'th service and t_i^+ is time t_i immediately after the i'th service.

We define $\mathrm{ALG}_i = \gamma D w(e_i)$, and $\mathrm{OPT}_i = \mathbb{I}_i \cdot w(e_i)$. Observe from Lemmas 5.11 and 5.17 that $\sum_i \mathrm{ALG}_i \geq \mathrm{ALG}$ and $\sum_i \mathrm{OPT}_i \leq 3 \cdot \mathrm{OPT}^B + 3D \cdot \mathrm{OPT}^D$.

Lemma 5.27. *For every $i \in [k]$, we have that* $\mathrm{ALG}_i \leq 4\gamma D \cdot \mathrm{OPT}_i - \Delta_i \phi$.

Proof. If $\mathbb{I}_i = 1$, then $\mathrm{OPT}_i = w(e_i)$. Using Proposition 5.14, we have that $\Delta_i \phi \leq 3\gamma D \cdot w(e_i)$. Thus

$$\mathrm{ALG}_i = \gamma D w(e_i) = 4\gamma D \cdot \mathrm{OPT}_i - 3\gamma D \cdot w(e_i) \leq 4\gamma D \cdot \mathrm{OPT}_i - \Delta_i \phi$$

as required.

Otherwise, $\mathbb{I}_i = 0$. Then, $\mathrm{OPT}_i = 0$. Since the optimum's server is on the other side of the edge e_i than the algorithm's server before the i'th service, and the algorithm finishes the service on that other side of e_i, it must be that $\Delta_i \phi \leq -\gamma D \cdot w(e_i)$. Thus,

$$\mathrm{ALG}_i = \gamma D w(e_i) \leq 4\gamma D \cdot \mathrm{OPT}_i - \Delta_i \phi$$

finishing the proof of the lemma. $\qquad\square$

Proposition 5.28. *Denote the final value of ϕ by $\phi(\infty)$. Then*

$$\sum_i \Delta_i \phi \geq \phi(\infty) - \gamma D \cdot \mathrm{OPT}^B$$

Proof. Consider that $\phi(\infty) = \phi(\infty) - \phi(-\infty)$ can be constructed by summing the changes to the potential function caused by moves of the algorithm's server (which are the $\Delta_i \phi$) and changes caused by moves of the optimum's server. Note that moving the optimum's server by x can increase ϕ by at most $\gamma D x$. Thus,

$$\phi(\infty) \leq \sum_i \Delta_i \phi + \gamma D \cdot \mathrm{OPT}^B$$

yielding the proposition. $\qquad\square$

Corollary 5.29. $\sum_i \Delta_i \phi \geq -\gamma D \cdot \mathrm{OPT}^B$

of Theorem 5.10. Due to Lemma 5.27, we have that

$$\mathrm{ALG} \leq \sum_i \mathrm{ALG}_i \leq \sum_i 4\gamma D \cdot \mathrm{OPT}_i - \sum_i \Delta_i \phi$$

Since $\sum_i \mathrm{OPT}_i \leq 3\mathrm{OPT}^B + 3D \cdot \mathrm{OPT}^D$, and using Corollary 5.29, we have that

$$\mathrm{ALG} \leq 4\gamma D \cdot \left(3\mathrm{OPT}^B + 3D \cdot \mathrm{OPT}^D\right) + \gamma D \cdot \mathrm{OPT}^B$$
$$\leq 13\gamma D \cdot \mathrm{OPT}^B + 12\gamma D^2 \cdot \mathrm{OPT}^D$$

proving the theorem. $\qquad\square$

References

[1] Aris Anagnostopoulos, Russell Bent, Eli Upfal, and Pascal Van Hentenryck. A simple and deterministic competitive algorithm for online facility location. *Inf. Comput.*, 194(2):175–202, 2004.

[2] Itai Ashlagi, Yossi Azar, Moses Charikar, Ashish Chiplunkar, Ofir Geri, Haim Kaplan, Rahul M. Makhijani, Yuyi Wang, and Roger Wattenhofer. Min-cost bipartite perfect matching with delays. In *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques, APPROX/RANDOM 2017, August 16-18, 2017, Berkeley, CA, USA*, pages 1:1–1:20, 2017.

[3] Yossi Azar, Yuval Emek, Rob van Stee, and Danny Vainstein. The price of clustering in bin-packing with applications to bin-packing with delays. In *The 31st ACM Symposium on Parallelism in Algorithms and Architectures, SPAA 2019*, 2019. To appear.

[4] Yossi Azar and Amit Jacob Fanani. Deterministic min-cost matching with delays. In *Approximation and Online Algorithms - 16th International Workshop, WAOA 2018, Helsinki, Finland, August 23-24, 2018, Revised Selected Papers*, pages 21–35, 2018.

[5] Yossi Azar, Arun Ganesh, Rong Ge, and Debmalya Panigrahi. Online service with delay. In *Proceedings of the 49th Annual ACM SIGACT Symposium on Theory of Computing, STOC 2017, Montreal, QC, Canada, June 19-23, 2017*, pages 551–563, 2017.

[6] Nikhil Bansal, Niv Buchbinder, Aleksander Madry, and Joseph Naor. A polylogarithmic-competitive algorithm for the k-server problem. *J. ACM*, 62(5):40:1–40:49, 2015.

[7] Marcin Bienkowski, Martin Böhm, Jaroslaw Byrka, Marek Chrobak, Christoph Dürr, Lukáš Folwarczný, Lukasz Jez, Jiri Sgall, Nguyen Kim Thang, and Pavel Veselý. Online algorithms for multi-level aggregation. In *24th Annual European Symposium on Algorithms, ESA 2016, August 22-24, 2016, Aarhus, Denmark*, pages 12:1–12:17, 2016.

[8] Marcin Bienkowski, Jaroslaw Byrka, Marek Chrobak, Lukasz Jez, Dorian Nogneng, and Jirí Sgall. Better approximation bounds for the joint replenishment problem. In *Proceedings of the Twenty-Fifth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2014, Portland, Oregon, USA, January 5-7, 2014*, pages 42–54, 2014.

[9] Marcin Bienkowski, Artur Kraska, Hsiang-Hsuan Liu, and Pawel Schmidt. A primal-dual online deterministic algorithm for matching with delays. In *Approximation and Online Algorithms - 16th International Workshop, WAOA 2018, Helsinki, Finland, August 23-24, 2018, Revised Selected Papers*, pages 51–68, 2018.

[10] Marcin Bienkowski, Artur Kraska, and Pawel Schmidt. A match in time saves nine: Deterministic online matching with delays. In *Approximation and Online Algorithms - 15th International Workshop, WAOA 2017, Vienna, Austria, September 7-8, 2017, Revised Selected Papers*, pages 132–146, 2017.

[11] Marcin Bienkowski, Artur Kraska, and Pawel Schmidt. Online service with delay on a line. In *Structural Information and Communication Complexity - 25th International Colloquium, SIROCCO 2018, Ma'ale HaHamisha, Israel, June 18-21, 2018, Revised Selected Papers*, pages 237–248, 2018.

[12] Carlos Fisch Brito, Elias Koutsoupias, and Shailesh Vaya. Competitive analysis of organization networks or multicast acknowledgment: How much to wait? *Algorithmica*, 64(4):584–605, 2012.

[13] Niv Buchbinder, Moran Feldman, Joseph (Seffi) Naor, and Ohad Talmon. O(depth)-competitive algorithm for online multi-level aggregation. In *Proceedings of the Twenty-Eighth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2017, Barcelona, Spain, Hotel Porta Fira, January 16-19*, pages 1235–1244, 2017.

[14] Niv Buchbinder, Kamal Jain, and Joseph Naor. Online primal-dual algorithms for maximizing ad-auctions revenue. In *Algorithms - ESA 2007, 15th Annual European Symposium, Eilat, Israel, October 8-10, 2007, Proceedings*, pages 253–264, 2007.

[15] Niv Buchbinder, Tracy Kimbrel, Retsef Levi, Konstantin Makarychev, and Maxim Sviridenko. Online make-to-order joint replenishment model: primal dual competitive algorithms. In *Proceedings of the Nineteenth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2008, San Francisco, California, USA, January 20-22, 2008*, pages 952–961, 2008.

[16] Rodrigo A. Carrasco, Kirk Pruhs, Cliff Stein, and José Verschae. The online set aggregation problem. In *LATIN 2018: Theoretical Informatics - 13th Latin American Symposium, Buenos Aires, Argentina, April 16-19, 2018, Proceedings*, pages 245–259, 2018.

[17] Daniel R. Dooly, Sally A. Goldman, and Stephen D. Scott. TCP dynamic acknowledgment delay: Theory and practice (extended abstract). In *Proceedings of the Thirtieth Annual ACM Symposium on the Theory of Computing, Dallas, Texas, USA, May 23-26, 1998*, pages 389–398, 1998.

[18] Yuval Emek, Shay Kutten, and Roger Wattenhofer. Online matching: haste makes waste! In *Proceedings of the 48th Annual ACM SIGACT Symposium on Theory of Computing, STOC 2016, Cambridge, MA, USA, June 18-21, 2016*, pages 333–344, 2016.

[19] Yuval Emek, Yaacov Shapiro, and Yuyi Wang. Minimum cost perfect matching with delays for two sources. In *Algorithms and Complexity - 10th International Conference, CIAC 2017, Athens, Greece, May 24-26, 2017, Proceedings*, pages 209–221, 2017.

[20] Jittat Fakcharoenphol, Satish Rao, and Kunal Talwar. A tight bound on approximating arbitrary metrics by tree metrics. *J. Comput. Syst. Sci.*, 69(3):485–497, 2004.

[21] Dimitris Fotakis. On the competitive ratio for online facility location. *Algorithmica*, 50(1):1–57, 2008.

[22] Anna R. Karlin, Claire Kenyon, and Dana Randall. Dynamic TCP acknowledgment and other stories about e/(e-1). *Algorithmica*, 36(3):209–224, 2003.

[23] Adam Meyerson. Online facility location. In *42nd Annual Symposium on Foundations of Computer Science, FOCS 2001, 14-17 October 2001, Las Vegas, Nevada, USA*, pages 426–431, 2001.

A   Additional Figures

(a) After Open(u).

(b) Time forwarding reaches t', the earliest deadline of a pending request in T_u.

(c) The request is connected to u.

(d) The new earliest deadline t'' is reached in time forwarding.

(e) Investment triggers Explore(u_2), serving the earliest deadline request.

Figure 6: Visualization of Algorithm 1 – the operation of Explore(u) at time t.