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), where the only previously-known improvement was for the special case of deadlines by Buchbinder et al. (SODA '17). 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.

I. 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.

This paper is partially supported by the ISF grant no. 1506/16. A full version of this paper appears in https://arxiv.org/abs/1904.07131.

2575-8454/19/$31.00 ©2019 IEEE
DOI 10.1109/FOCS.2019.00013

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 $O(D^4 2^D)$ bound 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 II 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 III discusses the more general problem of online facility location with delay, and extending the algorithm for the deadline case in Section II to an $O(\log^2 n)$-competitive algorithm for the case of delay. Section IV presents the $O(D^2)$-competitive deterministic algorithm for online multilevel aggregation with delay. Section V 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 IV.

II. ONLINE FACILITY LOCATION WITH DEADLINES

A. 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 II.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.

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 II.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 II.3. *There exists an $O(\log^2 n)$-competitive randomized algorithm for online facility location with deadlines for any metric space of n points.*

B. 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 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.

Algorithm 1: Facility Location with Deadlines

```
Initialization.
Initialize $c_u \leftarrow 0$ for any node $u \in T \backslash \{r\}$.
Declare $b_u$ for any node $u \in T$.

Event Function UponDeadline() // Upon expired deadline of pending request at time $t$
    Explore($r$)

Function Explore($u$)
    Open($u$)
    // Spend a budget of $f$ on charging child node counters
    set $b_u \leftarrow f$
    while $b_u \neq 0$ and there remain pending requests in $T_u$ do
        // Consider pending requests by increasing deadline
        let $q$ be the pending request with earliest deadline in $T_u$
        let $v$ be the child of $u$ on the path to $v_q$
        call Invest($u$,$v$,$\delta(u, v_q)$)
        if $c_v = f$ then set $c_v \leftarrow 0$ ; call Explore($v$).
        if $q$ is still pending then connect $q$ to facility at $u$
```

Algorithm 1: Facility Location with Deadlines (cont.)

```
Function Open($u$)
    open facility at $u$.
    if $u$ is a leaf node then connect to facility all pending requests on $u$

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.
    let $y \leftarrow \min(x, b_u, f - c_v)$
    increase $c_v$ by $y$
    decrease $b_u$ by $y$
    return $y$
```

C. 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.

Theorem II.4. $\text{ALG} \leq O(D^2) \cdot \text{OPT}^B + O(D) \cdot \text{OPT}^C$.

To prove Theorem II.4, we show validity of the algorithm, an upper bound for ALG and a lower bound for OPT.

1) Validity of the Algorithm

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

Proposition II.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. ■

Corollary II.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 II.5 concludes the proof. ■

2) Upper Bounding ALG

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. We prove the following lemma.

Lemma II.7. $\mathrm{ALG} \leq 3 \cdot (D+1) \cdot kf$.

The proof of Lemma II.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 II.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 II.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 $\texttt{Explore}_t(u)$.

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.

Propositions II.10 and II.11 immediately yield Lemma II.7, and their proofs are given the full version of the paper.

Proposition II.10. $\sum_{u \in T} \bar{c}_u \leq (D+1)kf$

Proposition II.11. $\mathrm{ALG} \leq 3 \cdot \sum_{u \in T} \bar{c}_u$

3) Lower Bounding OPT

We now provide a lower bound for the cost of OPT, through proving the following lemma. Recall that k is number of calls to `Explore`(r).

Lemma II.12. $kf \leq 2(D+1) \cdot \mathrm{OPT}^B + 4 \cdot \mathrm{OPT}^C$

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 `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 `Explore`(u) is called. Similarly, we allow the charging nodes $(u, [\tau_1, \tau_2])$ in which τ_1 is the last time `Explore`(u) is called, and $\tau_2 = \infty$. We denote by M the set of charging nodes. Figure 1 visualizes 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 II.13. $\sum_\mu c(\mu) \leq 2(D+1) \cdot \mathrm{OPT}^B + 4 \cdot \mathrm{OPT}^C$

Proof: In the full version of the paper. ■

Definition II.14 (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 II.15. *For any $G = (V', E)$ and weights $\alpha : E \to \mathbb{R}^+$, we have $\sum_{v \in V'} \chi_v = 0$.*

Definition II.16. 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.

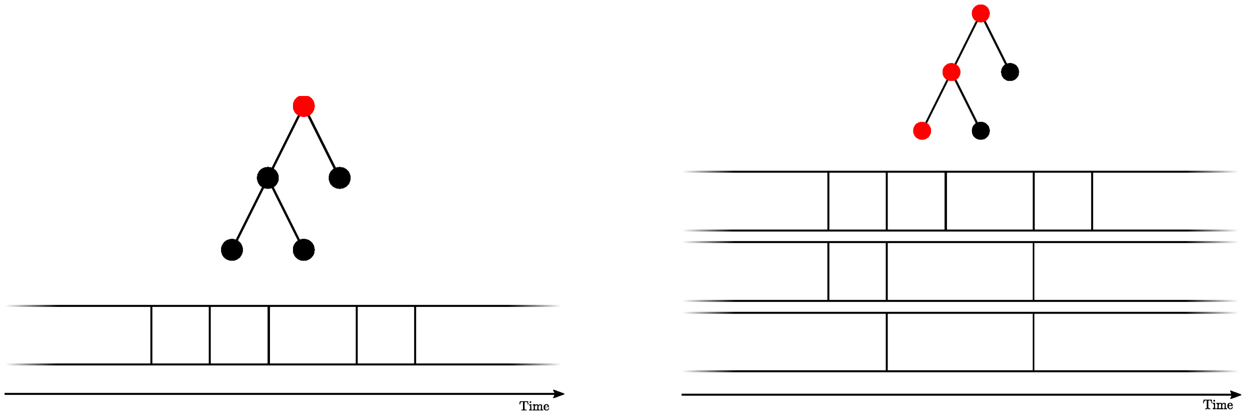
Observation $II.15$ yields that $\chi_s \leq 0$ for every preflow $Z = (G, s, \alpha)$. We write $\omega_Z = -\chi_s$.

Proposition II.17. *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 II.15 and the definition of a preflow, we get $\sum_{v \in S} \chi_v \leq \sum_{v \in V} \chi_v = -\chi_s = \omega_Z$. ■

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,



(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

since c_u was empty at τ_1 and was filled and emptied again until time τ_2.

Definition II.18 (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.

Definition II.19 (λ^t_u and λ_μ)**.** For every function call $\texttt{Explore}_t(u)$ for some $u \in T$ and time t, we denote by λ^t_u 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 $\lambda^t_u = \infty$).

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

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 the preflow-building procedure 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 $\mu_1 = (u_1, [\tau^1_1, \tau^1_2])$ to any charging node $\mu_2 = (u_2, [\tau^2_1, \tau^2_2])$ if μ_1 invested in μ_2. We set the weight $\alpha(\sigma)$ to be the amount that μ_1 invested in μ_2.

The procedure which constructs the preflow, as well as the proof of Lemma II.12, appear in the full version of the paper. Figure 2 gives a visualization of this procedure.

Proof of Theorem II.4: Combining Lemmas II.7 and II.12, we have that

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

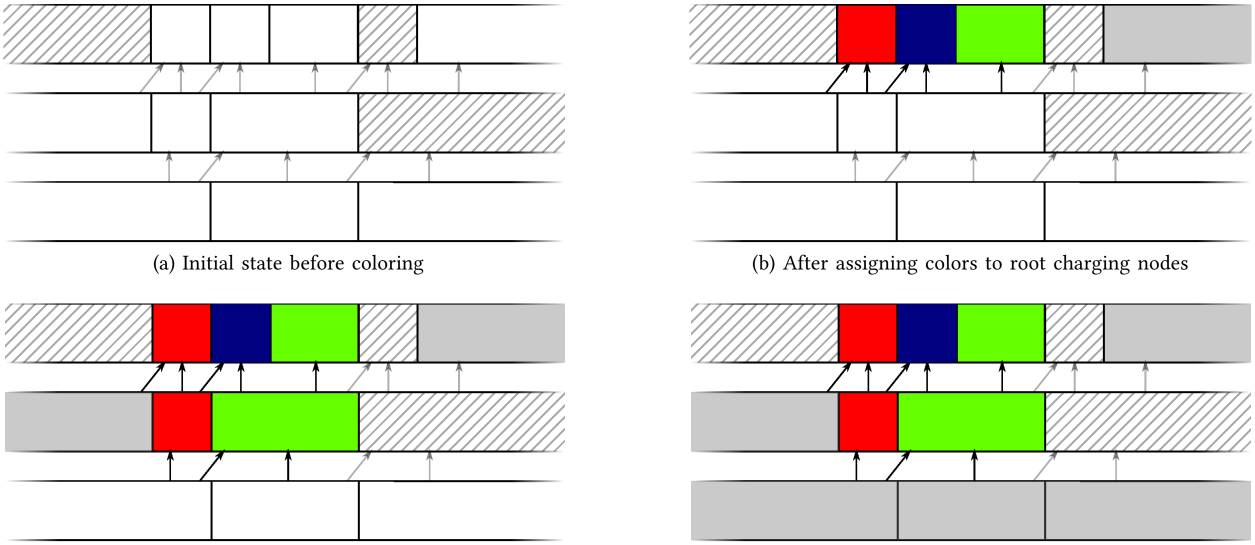
■

Remark II.20. 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, the analysis would no longer hold.

III. Online Facility Location with Delay

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



(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 the preflow-building procedure, 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 nodes with the `Special` color appear 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

problem with requests Q, the algorithm incurs the delay cost $\mathrm{ALG}^D = \sum_{q \in Q} d_q(t_q)$, where t_q is the time in which q is served by the algorithm.

The algorithm's goal is to minimize the total cost $\mathrm{ALG} = \mathrm{ALG}^B + \mathrm{ALG}^C + \mathrm{ALG}^D$.

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 the full version of the paper, we prove the following theorem.

Theorem III.1. *There exists an $O(\log^2 n)$-competitive randomized algorithm for facility location with delay for a general metric space of size n.*

IV. ONLINE MULTILEVEL AGGREGATION WITH DELAY

A. 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,

Algorithm 2: Online Multilevel Aggregation with Delay

```
**Initialization.**
Initialize $c_e \leftarrow 0$ for any edge $e \in T \backslash \{r\}$
Declare $b_e$ for every edge $e \in T$.
Declare $\mathcal{T}$.

**Event Function** UponCritical() // Upon request set becoming critical as per Definition IV.3
    set $\mathcal{T} \leftarrow \emptyset$
    Explore($r$)
    transmit $\mathcal{T}$

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

Algorithm 2: Online Multilevel Aggregation with Delay (cont.)

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

**Function** Invest($e$,$e'$)
    let $y \leftarrow \min(b_e, w(e') - c_{e'})$
    increase $c_{e'}$ by $y$
    decrease $b_e$ by $y$
    **return** $y$.
```

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 IV.1 (Similar to Definition II.1)**.** For every tree edge $e \in T$, we use the following notations:

- For $e \neq r$, 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 $Q \subseteq T_e$, 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 prove the following theorem.

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

B. Algorithm for HSTs

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

Definition IV.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 IV.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 2. 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 $\mathcal{T}$. The analysis of Algorithm 2, as well as the proof of Theorem IV.2, appear in the full version of the paper.

V. Online Service with Delay

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 the full version of the paper, we prove the following theorem.

Theorem V.1. *There exists a randomized $O(\log^2 n)$-competitive algorithm for online service with delay on a general metric space of n points.*

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.