Beyond Monotone Delays for Multi-Level Aggregation

Yossi Azar
Department of Computer Science, Tel Aviv University, Israel

Liad Iluz
Department of Computer Science, Tel Aviv University, Israel

Abstract

In the online Multi-Level Aggregation Problem (MLAP), requests arrive over time and are associated with nodes of a given weighted rooted tree of depth D. Each request must eventually be served by performing a service. Serving a request consists of selecting a rooted subtree that contains the request's node, incurring a service cost equal to the total weight of the selected subtree. To reduce service costs, multiple requests may be served simultaneously by selecting a single rooted subtree that spans all of them. In addition, each request is associated with a penalty function that specifies the cost incurred when the request is served at a particular time. The objective is to minimize the total cost, consisting of both service costs and penalty costs.

Most previous work on MLAP assumes monotone non-decreasing penalty functions, commonly referred to as delay functions. Only very recent results consider penalty functions that initially decrease and subsequently increase, and even then only for the special cases of depths $D = 1$ and $D = 2$, namely the Joint Replenishment Problem (JRP).

In this work, we extend previous results in two ways. First, we allow *arbitrary penalty functions*, which may decrease and increase multiple times. Second, we study the general MLAP with *arbitrary tree depth* D under these arbitrary penalty functions. We present a randomized algorithm which is $O(D \log n \log(nDW))$-competitive, where W is the maximum service window among all penalty functions after normalizing the Lipschitz parameter of each penalty function to be 1 and normalizing the minimum positive edge weight incident to the root to be 1; and n is the number of requests. We note that our algorithm runs in polynomial-time, and even for $D = 1$ the problem admits hardness of approximation of $\Omega(\log n)$ for polynomial time algorithms.

As mentioned above, prior to our work even for trees of depth $D = 1, 2$, non-monotone penalty functions have been studied only in special cases of functions that decrease and increase only once. In contrast, for such trees we obtain $O(\log n \log(nW))$-competitive algorithms for arbitrary non-monotone penalty functions.

2012 ACM Subject Classification Theory of computation → Online algorithms

Keywords and phrases online algorithms, multi-level aggregation problem, arbitrary penalty functions, multicast problem

Digital Object Identifier 10.4230/LIPIcs.ESA.2026.65

Related Version *Full version*: https://arxiv.org/abs/2607.04317

1 Introduction

The online *Multi-Level Aggregation Problem* (MLAP) is a general aggregation problem that captures the trade-off between waiting in order to batch requests, thereby sharing an expensive resource, and serving requests at an optimal time to reduce time-dependent penalties. Every request arrives at a node of a given weighted rooted tree $\mathcal{T}$ and must eventually be served. A service corresponds to selecting a rooted subtree $\mathcal{T}' \subseteq \mathcal{T}$, where all requests located in $\mathcal{T}'$ can be served simultaneously. *The service cost* or *the cost of performing a service* is defined as the total weight of the edges in $\mathcal{T}'$. In addition, each request incurs a time-dependent penalty cost, which depends on the time at which the request is served. The objective is to minimize the total cost, defined as the sum of the service costs and the penalties.

The depth of the tree D, is a key structural parameter of the problem. When $D = 1$, MLAP is equivalent to the *TCP Acknowledgment Problem* (TCP-AP), where the tree consists only of a root and a single child, and the edge weight represents the cost of performing an acknowledgment (also called single-item JRP). When $D = 2$, MLAP is equivalent to the *Joint Replenishment Problem* (JRP), which models multiple items ordered with a fixed order cost (the cost of the root) and with item-specific costs (the cost of the second level).

Most prior work on the various variants of the MLAP has focused on instances with monotone non-decreasing penalty functions, commonly referred to as *delay functions* (see, e.g., [6, 4, 12] and more in Subsection 1.3). More recent work has studied the Joint Replenishment Problem (JRP) under penalty functions that combine holding and backlog costs. In this setting, the penalty function first decreases (modeling holding cost) and then increases (modeling backlog cost), and is assumed to be identical across all requests [20]. Subsequent work extended these results to allow penalty functions that first decrease and then increase to differ across requests, while still achieving constant-competitive algorithms [3, 23].

In contrast, we allow the penalty functions to exhibit arbitrary behavior, possibly decrease and increase multiple times. This has not been considered even for the TCP-AP (single-item JRP). Moreover, we consider the depth of the tree D as an arbitrary parameter and not restricted only to a constant D, rather than restricting attention to the constant-depth case ($D = 2$) as in the JRP. For such an arbitrary D, only monotone non-decreasing penalty functions (delay functions) have been considered. Since the common term *delay* implies monotonicity, we instead adopt the more general term *penalty function*, and use *penalty* to denote the cost incurred when serving a request at a given time.

To this end, we introduce a new problem, the *online Incremental Multicast Problem* (IMP). This problem is a natural generalization of the classical online Multicast Problem (MP). Unlike the multicast problem, where the input trees are fixed in advance, in the incremental multicast problem the trees are initially empty and may grow over time as requests arrive.

We then establish two key reductions that transform an instance of MLAP with arbitrary penalty functions into a corresponding instance of IMP. Finally, to obtain a complete algorithmic solution for our problem, we show that the randomized algorithm for the online multicast problem introduced by Alon et al. [1] can be naturally adapted to solve the incremental multicast problem (IMP). We rewrite both the algorithm and its analysis to fit the context of this new variant. Combined with our reductions, this yields a full randomized algorithm for MLAP with arbitrary penalty functions.

Despite allowing penalty functions that may decrease and increase multiple times, our framework remains highly relevant to real-world applications, including classical delay-based models. For example, in supply chain and inventory management, where the JRP is commonly applied, the urgency of serving a request may vary over time. A retailer preparing for a seasonal sales period may face a penalty that increases as peak demand approaches, but then decreases afterward due to discounts, excess inventory or supplier flexibility.

1.1 Our Results

Let n be the number of requests that arrive during the algorithm and let D be the depth of the problem tree $\mathcal{T}$. Assume WLOG that all edge weights are positive (zero weight edges can be contracted), and that the root has only one connected node (otherwise, split the tree into a collection of disjoint trees, and solve the problem on each of them, as every two requests that are related to different trees can not share a service). Then, we normalize the weight of this only edge that incident to the root to be 1, by normalizing the values of the penalty function using this value.

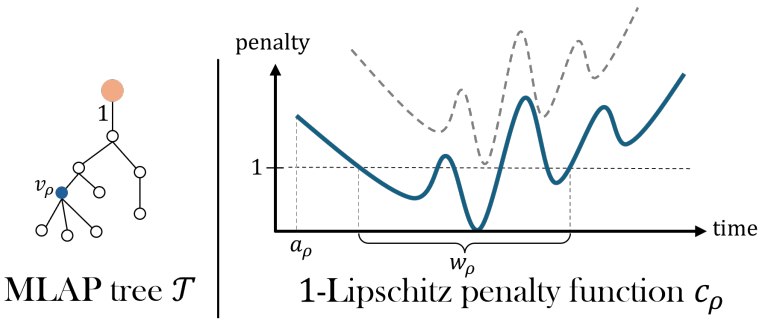

Figure 1 Request ρ arrives at time a_ρ, is associated with a node v_ρ, and has a 1-Lipschitz penalty function c_ρ, obtained by: (i) dividing by the minimum positive edge incident to the root (making it to be 1), (ii) shifting c_ρ so its minimum is 0, and (iii) rescaling to make it 1-Lipschitz. The dashed gray line represents the penalty function before normalization. The service window w_ρ is the interval between the earliest time with $c_\rho(t) \leq 1$ and the latest time with $c_\rho(t) \geq 1$.

We further assume, without loss of generality, that each penalty function attains the value 0. Otherwise, we shift the penalty function by subtracting its minimum value. This transformation does not affect the competitive-ratio analysis, since the same constant is subtracted from both the algorithm's cost and the optimal cost. Consequently, the new competitive ratio is at least as large as the original one. We then normalize the penalty function to be 1-Lipschitz[1] by appropriately rescaling the time axis. Let w_ρ denote the *service window* of request ρ, defined as the interval from the earliest time at which the penalty (after normalization) is at most 1, to the latest time at which it is at least 1. Let W be the maximum service window over all requests.

The parameter W need not be known in advance, and it captures the complexity of the problem: larger values of W correspond to penalty functions that originally vary more quickly or allow longer service horizons. Figure 1 illustrates these definitions.

Finally, we briefly define a *2-decreasing rooted tree* as a weighted rooted tree in which the weight of every edge is at most half the weight of its parent edge.

The following results are given as a worst case upper bound analysis of the competitive ratio between the value of our algorithm to the value of the optimal solution, which know the whole input sequence in advance.

- For the online MLAP with arbitrary penalty functions on 2-decreasing trees, we design a randomized algorithm which is $O(\log n \log(nDW))$-competitive.
- There exists a randomized algorithm for the online MLAP with arbitrary penalty functions for general trees that is $O(D \log n \log(nDW))$-competitive.
- For constant D (in particular, for the JRP with arbitrary penalty functions), we obtain a randomized algorithm which is $O(\log n \log(nW))$-competitive.

Remark. Although the parameter n may be large, we also show that it can be replaced by $\bar{n}$ in all the above results, which is typically much smaller than n. Here, $\bar{n}$ denotes the maximum number of requests that arrive within the same window of length W.

We note that [23] showed that an α-approximation algorithm for the offline single-item JRP ($D = 1$) with arbitrary penalty functions implies the same approximation for the offline set-cover problem. This implies a hardness of $\Omega(\log n)$ for any polynomial time algorithm,

[1] A function f is called L-Lipschitz if for all x, y: $|f(x) - f(y)| \leq L|x - y|$.

even for a single-item JRP, unless $P = NP$. However, if we allow exponential time (or unbounded time) algorithms, the best lower bound for online deterministic algorithms is 2 for $D = 1$ and 2.754 for $D \geq 2$. We improve this result for non-monotone penalty functions. Specifically, we obtained a lower bound of 4 for $D = 1$ with penalty functions with values in $[0, 1]$. Note that this lower bound is higher than the upper bound of the JRP with penalty functions that decrease and subsequently increase considered in [23], which separates the two problems.

1.2 Our Techniques

Our solution is built upon three reductions, which can be illustrated by the following flow diagram:

$$\text{MLAP}(n, D) \Longrightarrow \text{MLAP}_2(n, D) \Longrightarrow \text{MLAP}_{\text{2-Discrete}}(n, D, T) \Longrightarrow \text{IMP}(n, m)$$

The parameter n is the number of requests in all problems, and in the MLAP variants, D refers to the depth of the tree. The first reduction transforms a general $\text{MLAP}(n, D)$ instance with an arbitrary tree into $\text{MLAP}_2(n, D)$ – the same problem with 2-decreasing trees. This step is standard in many MLAP solutions and incurs a multiplicative loss of a factor $2D$ in the competitive ratio. For more details see the full version of this paper or [6, 12, 4]. The second reduction takes an instance of MLAP_2 with arbitrary penalty functions defined over a continuous timeline and discretizes them into a set of T candidate service points in time, for appropriate parameter T. The resulting problem is called $\text{MLAP}_{\text{2-Discrete}}$. The third and final reduction converts $\text{MLAP}_{\text{2-Discrete}}$ into an instance of the Incremental Multicast Problem (IMP), a generalization of the multicast problem that we introduce in this paper. Here, the parameter m denotes the number of edges in the final IMP graph. We show the following results towards a full randomized algorithm for the MLAP with arbitrary penalty function:

1. **Reduction I:** $\text{MLAP}(n, D) \leq^{2D} \text{MLAP}_2(n, D)$.
2. **Reduction II:** $\text{MLAP}_2(n, D) \leq \text{MLAP}_{\text{2-Discrete}}(n, D, T)$ with $|T| = O(nW)$.
3. **Reduction III:** $\text{MLAP}_{\text{2-Discrete}}(n, D, T) \leq \text{IMP}(n, m)$ with $m = O(nD|T|)$.
4. **Algorithm I:** Designing a $O(\log n \log m)$-competitive randomized algorithm for the IMP, based on the randomized algorithm of Alon et. al. [1] for the online multicast problem.

Putting all the results together, we get a randomized algorithm for the online MLAP with arbitrary penalty functions and arbitrary trees which is $O(D \log n \log(nDW))$-competitive.

1.3 Related Works

The Joint Replenishment Problem (JRP)

The JRP models a retailer managing inventory for multiple items, where requests (demands) arrive in an online fashion over a continuous timeline. Orders incur 3 types of costs: a fixed joint cost, item-specific costs, and a time-dependent cost for serving a particular request at a specific time. The objective is to minimize the total cost. The JRP is a special case of the Multi-Level Aggregation Problem (MLAP) where $D = 2$. A simpler variant is the single-item JRP, where there is only one item to be ordered. This problem is equivalent to the *Dynamic TCP Acknowledgment Problem (TCP-AP)* ($D = 1$).

Classical work on JRP distinguishes two variants based on time-dependent costs: *make-to-stock*, where items can be held with per-unit holding cost, and *make-to-order*, where holding is disallowed and backlog costs are incurred. While equivalent in the offline setting, they

differ online. For the online make-to-order JRP, Brito et al. [11] presented a 5-competitive algorithm. Buchbinder et al. [13] improved this result giving a 3-competitive algorithm using the primal-dual approach and proved a lower bound of 2.64, which was later improved to 2.754 for any deterministic algorithm by Bienkowski et al. [8]. For the single-item JRP (TCP-AP), Dooly et al. [15, 16] gave a tight 2-competitive deterministic algorithm, Seiden [22] proved an $e/(e-1)$ lower bound for randomized algorithms, and Karlin et al. [17] matched it with a randomized algorithm that is $e/(e-1)$-competitive. For the online make-to-stock JRP, Bienkowski et al. [8] designed a 2-competitive algorithm in the special case where holding costs are zero, and showed that this bound is optimal.

More recently, research has shifted toward more general models. Moseley et al. [20] studied the case involving both holding and backlog costs. In their formulation, ordering too early incurs holding costs, while ordering too late incurs backlog costs. Thus, each request has a time-dependent cost function that first decreases (reflecting holding costs) and then increases (reflecting backlog costs), resulting in a unique time at which the cost of serving the request is minimized. They gave a deterministic 30-competitive algorithm for the special case where all requests share the same time-dependent function. Their results were later extended in [3, 23] to handle any such functions that may be different among requests, and also obtained constant-competitive algorithms.

Our work generalizes these results in two ways: (1) we consider arbitrary penalty functions that may exhibit multiple increases and decreases over time; and (2) we analyze the full MLAP under this model, rather than restricting to the JRP.

The Multi-Level Aggregation Problem (MLAP)

The MLAP was introduced by Bienkowski et al. [6] as a general model that captures the trade-off between delaying requests to benefit from aggregation, and serving them immediately to avoid high waiting costs. All potential requests are located at the nodes of a given rooted tree of depth D, and serving a set of requests corresponds to selecting a rooted subtree that spans those requests. Bienkowski et al. [6] presented an $O(D^4 2^D)$-competitive deterministic algorithm, which was later improved to $O(D^2)$-competitive deterministic algorithm by Azar and Touitou [4]. Azar and Touitou in [5] also show $O(\log k)$-competitive deterministic algorithm for this problem where k is the number of nodes in the tree. For the special case of MLAP with deadlines (zero cost follows by infinity cost after the deadline), Buchbinder et al. [12] gave an $O(D)$-competitive deterministic algorithm. For the special case of a path, Bienkowski et al. [9] gave an algorithm which is 5-competitive and showed a lower bound of 3.618, which was improved later to 4 by [7]. For both variants, the deadline and the delay, the best known lower bounds are only constant (as achieved under the JRP model). Additional variants of MLAP have been studied in [7, 10, 14, 18, 19, 21]. Nevertheless, all of these results assume monotone non-decreasing penalty functions (i.e., delay functions), whereas our work considers arbitrary functions that may increase and decrease multiple times over time.

The Multicast Problem (MP) in Trees

Given a collection of weighted rooted trees, and a set of clients (requests) that are associated with at most one node in each tree. The objective is to select a minimum-weight collection of rooted subtrees such that all clients are served; that is, for every client, the solution contains at least one rooted subtree that spans at least one of the nodes that associated with this client. In the online version, clients arrive one by one, and upon the arrival of a client, the

algorithm must immediately serve it. The service cost is defined as the sum of the weights of all edges in the chosen subtree. The algorithm may add new edges to its current solution but cannot remove any previously chosen edges. The goal is to minimize the total cost.

Alon et al. [1] described a randomized algorithm for the online MP in trees that achieves a competitive ratio of $O(\log n \cdot \log m)$, where n is the number of clients that arrive during the execution of the algorithm and m is the total number of edges across all trees. In a subsequent paper, Alon et al. [2] established a nearly matching lower bound of $\Omega\left(\frac{\log n \log m}{\log\log n + \log\log m}\right)$, for any deterministic algorithm, via a reduction from the online set cover problem, which is a special case of the MP.

1.4 Paper Organization

Section 2 introduces notation and definitions for MLAP with arbitrary penalty functions and the Incremental Multicast Problem (IMP) in trees, along with a standard reduction used in MLAP variants. Section 3 presents a reduction from continuous-time MLAP with arbitrary penalties to a discrete version; this reduction may be of independent interest. Section 4 gives the final reduction from discrete MLAP to IMP. Section 5 adapts the algorithm for MP of Alon et al. [1] to our IMP variant. Section 6 establishes a lower bound for this problem even for the case of $D = 1$. Finally, Section 7 concludes with directions for future research. More details will be provided in the full version of the paper.

2 Preliminaries

2.1 The MLAP with arbitrary penalty functions

The input of the MLAP consists of a weighted tree $\mathcal{T} = (V, E)$ with positive edge weights (or costs) $c : E \to \mathbb{R}^+$, rooted at a node $r \in V$ and having depth D. Each request is associated with a non-root node of the tree, representing the request's type. Serving a request means selecting a subtree rooted at r that contains the node of the request. Multiple requests may be aggregated and served simultaneously by selecting a single rooted subtree that spans all of them. The *service cost* or *the cost of performing a service*, is the sum of all weights in the selected subtree and is independent of the time at which the service is performed.

In addition, each request is associated with a penalty function that is revealed upon its arrival (clairvoyant model) and specifies the cost of serving the request at every point in continuous time. This function is arbitrary and may increase or decrease multiple times. The algorithm may perform additional services over time, but it can not cancel or modify a service once executed. The objective is to minimize the total cost: service costs plus penalty costs, while ensuring that every request is eventually served.

Notation and Formal Definition

We normalize the cost of the unique edge incident to the root to be 1 (by scaling all weights). For every node $v \in V$ that is not the root r, we denote by $c(v)$ the cost of the edge between v and its parent. For a subtree $\mathcal{T}' \subseteq \mathcal{T}$, $c(\mathcal{T}')$ is the total weight of all edges in $\mathcal{T}'$.

Requests associated with different nodes arrive over time. Each request ρ is specified by a tuple (v_ρ, a_ρ, c_ρ), where $v_\rho \in V$ is the node at which the request occurs, $a_\rho \in \mathbb{R}^+$ is the arrival time, and $c_\rho : [a_\rho, \infty) \to \mathbb{R}^+$ is the penalty function of the request. It means that serving request ρ at time t incurs a penalty of $c_\rho(t)$.

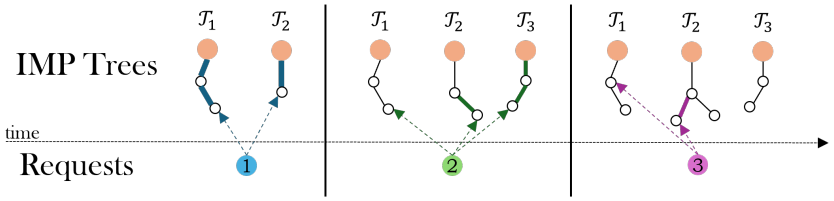

Figure 2 The figure shows three sequential requests. Each request (at the bottom) points to its associated vertices (S_ρ). Bold edges indicate newly added edges ($\{\mathcal{P}_v\}_{v\in S_\rho}$) and their costs ($c_\rho$). The first request connects to two new vertices in different trees via paths of lengths 2 and 1. The second request involves three vertices: one already in the tree (length 0), one connected by a single new edge, and one connected by a path of length 2 that creates a new tree. The third request involves two vertices: one existing (length 0) and one connected by a single new edge (length 1).

We normalize the penalty function to be 1-Lipschitz by stretching the time. Since the penalty function may increase and decrease over time, we define the *service window* of a request ρ, denoted by w_ρ, in order to bound the time period during which the request may be served. Specifically, w_ρ is the interval from the first time at which the function attains a value at most 1 to the last time at which it attains the value 1. We assume that this window is finite and denote by W the maximum length of a service window over all requests. Note that the values of these parameters need not be known in advance. Figure 1 illustrates the arrival of request ρ. The goal is to Minimize the total cost: $\sum_{i=1}^{\ell}\left(c\left(\mathcal{T}_i\right)+\sum_{\rho\in\mathcal{R}_i} c_\rho\left(t_i\right)\right)$.

Feasible Solution

Let R be a sequence of requests, each associated with exactly one node of $\mathcal{T}$. A solution to the MLAP consists of a sequence of services (rooted subtrees) $\mathcal{T}_1, \mathcal{T}_2, \ldots, \mathcal{T}_\ell \subseteq \mathcal{T}$, where each service $\mathcal{T}_i$ is performed at time t_i. Each service $\mathcal{T}_i$ has a corresponding set of requests $\mathcal{R}_i \subseteq R$ that are served by it. The solution is *feasible* if both conditions hold: (1) for every $i = 1, \ldots, \ell$ and every $\rho \in \mathcal{R}_i$: $t_i \geq a_\rho$ and $v_\rho \in \mathcal{T}_i$; and (2) every request $\rho \in R$ must eventually be served, i.e., there exists some i such that $\rho \in \mathcal{R}_i$.

The MLAP with discrete penalty functions

In this problem the penalty functions are discrete, i.e., each request ρ is associated with a discrete penalty function $c_\rho : T_\rho \to \mathbb{R}^+$, where T_ρ is a finite set of all potential time units to serve request ρ. We denote by T the set of all potential time units to serve all the requests, i.e., $T = \bigcup_{\rho\in R} T_\rho$.

2.2 The Incremental Multicast Problem (IMP)

We now introduce a new variant and a generalization of the online multicast problem. In contrast to the classical online multicast problem, in the incremental multicast problem the trees are not known in advance and may be incrementally constructed as requests arrive.

Let R be the sequence of requests and let $\mathcal{T} = (V, E)$ be a family of rooted trees. Initially, both R and $\mathcal{T}$ are empty. A new tree is introduced, together with its root, when the first request that is associated with this tree arrives, and the tree then grows incrementally as additional requests connect to it. Formally, each request ρ is represented by the tuple $(S_\rho, \{\mathcal{P}_v\}_{v\in S_\rho}, c_\rho)$, where:

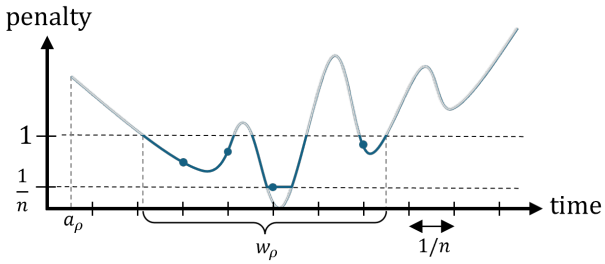

Figure 3 First, all values below $\frac{1}{n}$ get value $\frac{1}{n}$. Then, the discrete penalty function is defined at the points corresponding to the global intervals of length $\frac{1}{n}$ that lie within the service window and have values of at most 1 (shown as blue circles).

- S_ρ is a set of nodes (already in V or new nodes) that are associated with the request. Each node $v \in S_\rho$ belongs to at most one tree in $\mathcal{T}$.
- $\{\mathcal{P}_v\}_{v \in S_\rho}$ is a collection of paths in $\mathcal{T}$. For each node $v \in S_\rho$, the path $\mathcal{P}_v$ lies in some unique tree $\mathcal{T}' \in \mathcal{T}$. This path connects the node v to an existing node in $\mathcal{T}'$ or initiates a new rooted tree (a path from the node to the new root). Each such a path consists of only new edges, i.e., edges that were not previously present in this tree, and adding $\mathcal{P}_v$ to $\mathcal{T}'$ preserves the property that $\mathcal{T}'$ is a tree.
- $c_\rho : \bigcup_{v \in S_\rho} E(\mathcal{P}_v) \to \mathbb{R}^+$ is a cost function on edges assigning a positive cost to every new edge introduced by the request.

Figure 2 illustrates the incremental construction process in the IMP.

Feasible Solution

Let T_ρ denote the set of all roots that are associated with the nodes in S_ρ. When a request ρ arrives, it induces a unit demand $D_\rho = (S_\rho, T_\rho)$. This means that at any time, the solution must contain at least one path from some node $v \in S_\rho$ to the root of its corresponding tree from T_ρ. The algorithm is allowed to add new edges to the solution, but never remove edges once they have been added. The cost of the algorithm is the total cost of all edges in the final solution, and the objective is to minimize it.

The Fractional Model

In the fractional model, the algorithm may assign a fractional weight $w_e \in [0, 1]$ to any edge $e \in E$. These fractions may increase over time, but they can never decrease. A feasible solution is an assignment of weights $\{w_e\}_{e \in E}$ such that, at any moment and for every request ρ, the total flow from S_ρ to T_ρ is at least 1. Notice that since the fractional weights never decrease, the feasibility conditions of all previously arrived requests remain satisfied. The cost of the fractional solution is the weighted sum of edge costs: $\sum_{e \in E} w_e c_e$.

3 Reduction to MLAP with discrete arbitrary penalty functions

In this section, we describe the second reduction from the MLAP on 2-decreasing trees with maximum depth D and n requests ($\text{MLAP}_2(n, D)$) to $\text{MLAP}_{\text{2-Discrete}}(n, D, T)$, the same problem with discrete arbitrary penalty functions with the set T of all potential service time units. This technique is general and can be applied to other offline and online problems involving arbitrary penalty functions.

By Lipschitzness condition and bounded service window, we can assume that each penalty function has a global minimum point (with zero value as we mention earlier). For simplicity, we make two additional assumptions that will be removed at the end of this section: (1) all n requests arrive in the time interval $[0, W]$ (see Subsection 3.1); and (2) the parameters n and W are known in advance (see Subsection 3.2). Finally, we note that throughout this section we work in the setting of MLAP on 2-decreasing trees with maximum depth D. The main result of this section:

▶ **Theorem 1** (Discretization Step). *Assume there is an algorithm $\mathcal{A}$ for the $MLAP_{2\text{-}Discrete}(n, D, T)$ which is an α-approximation (offline) or competitive (online). Then, there is an algorithm $\mathcal{B}$ for the original $MLAP_2(n, D)$ which is an $O(\alpha)$-approximation (offline) or competitive (online), with $|T| = O(nW)$. Moreover, both algorithms do not need to know the parameters n and W in advance.*

We start with the offline version and then we will show that it can be maintained in online fashion, using a two-step reduction:

1. Reduce to **non-zero** penalty functions $[a_\rho, \infty) \to \left[\frac{1}{n}, \infty\right)$, incurring a factor of 2 loss in approximation.
2. Reduce to **discrete** penalty functions $T_\rho \to \left[\frac{1}{n}, 1\right]$, where T_ρ is the potentially discrete set of service time unites for ρ, incurring a factor of 6 loss in approximation.

Overall, we get a reduction from arbitrary penalty functions $[a_\rho, \infty) \to [0, \infty)$ to discrete penalty functions $T_\rho \to \left[\frac{1}{n}, 1\right]$, incurring a factor of 12 loss in approximation. To show any of those reductions, we denote by $\mathrm{ALG}_{\mathrm{org}}$, $\mathrm{ALG}_{\mathrm{new}}$, $\mathrm{OPT}_{\mathrm{org}}$ and $\mathrm{OPT}_{\mathrm{new}}$ the values of the algorithm and the optimum on the original problem and on the new problem respectively. In each step, we denote by c_ρ the original penalty function of a request ρ, and by $\hat{c}_\rho$ its new penalty function after applying the step. Figure 3 illustrates the obtained discrete function.

Step 1: Non-zero values

Define a new **non-zero penalty function** by setting every value less than $\frac{1}{n}$ to be $\frac{1}{n}$ and all other values remain the same: $\hat{c}_\rho = \max\{c_\rho, \frac{1}{n}\}$.

▶ **Lemma 2.** *Any α-approximation (or competitive) algorithm for the problem using the new non-zero penalty functions is a 2α-approximation (or competitive) algorithm for the original penalty functions.*

Proof. Since $\hat{c}_\rho(t) \geq c_\rho(t)$ for every t, it holds that $\mathrm{ALG}_{\mathrm{org}} \leq \mathrm{ALG}_{\mathrm{new}}$. On the other hand, by keeping all service points as in $\mathrm{OPT}_{\mathrm{org}}$, the new optimum pays at most an additional $\frac{1}{n} \cdot n = 1$. Finally, since $\mathrm{OPT}_{\mathrm{org}} \geq 1$ for serving at least one request, we get $\mathrm{OPT}_{\mathrm{new}} \leq \mathrm{OPT}_{\mathrm{org}} + 1 \leq 2 \cdot \mathrm{OPT}_{\mathrm{org}}$. ◀

Step 2: Discrete functions

The discrete set of service time units for ρ, denoted by T_ρ, is the set obtained by dividing the service window w_ρ into global intervals of length $\frac{1}{n}$, restricted to points with value at most 1:

$$T_\rho = \left\{ t := \frac{1}{n} \cdot q \;\middle|\; q \in \mathbb{N},\ t \in w_\rho,\ c_\rho(t) \leq 1 \right\}.$$

The new discrete penalty function is defined as $\hat{c}_\rho = c_\rho|_{T_\rho}$.

▶ **Lemma 3.**

1. *For every request ρ the set T_ρ is not empty.*
2. *Let $T = \bigcup_\rho T_\rho$. Then, $|T| = O(nW)$.*
3. *Any α-approximation (competitive) algorithm for the problem using the new discrete penalty functions is a 6α-approximation (competitive) algorithm using the original non-zero penalty functions.*

Proof.

1. Notice that the penalty functions obtained from Step 1 are still 1-Lipschitz. Therefore, for every interval I of length $\frac{1}{n}$, we have $\max_{t\in I} c_\rho(t) - \min_{t\in I} c_\rho(t) \leq \frac{1}{n}$. Let t_0 be a time at which the original penalty function attains value 0. The endpoint of the interval containing t_0 has cost at most $\frac{1}{n} \leq 1$. By definition, this point belongs to w_ρ, and hence is included in T_ρ.
2. Since $|T_\rho| \leq \frac{w_\rho}{1/(n)} + 1$, and all requests arrived in $[0, W]$, $|T| \leq \frac{W}{1/(n)} + 1 = O(nW)$.
3. Since we only restrict the set of potential service points, $\mathrm{ALG}_{\mathrm{org}} \leq \mathrm{ALG}_{\mathrm{new}}$. Let t be the service time of request ρ in $\mathrm{OPT}_{\mathrm{org}}$. Define a new corresponding service time $t' \in T_\rho$ for $\mathrm{OPT}_{\mathrm{new}}$ as follows:
   - If $c_\rho(t) > \left(1 - \frac{1}{n}\right)$, then t' is an arbitrary point in T_ρ (e.g., the closest one).
   - Otherwise, t' is the closest point in T_ρ such that $t' \geq t$ (we will show that there exists one in this case).

   Next we analyze the cost of the new optimal solution. Let $R_{>\left(1-\frac{1}{n}\right)}$ be the set of requests served by $\mathrm{OPT}_{\mathrm{org}}$ with penalty greater than $\left(1 - \frac{1}{n}\right)$, and define $R_{\leq\left(1-\frac{1}{n}\right)}$ analogously. Denote their total penalties by $cost\left(R_{>(1-\frac{1}{n})}\right)$ and $cost\left(R_{\leq(1-\frac{1}{n})}\right)$. Then, using these notations:

   $$\mathrm{OPT}_{\mathrm{org}} = cost\left(R_{>(1-\frac{1}{n})}\right) + cost\left(R_{\leq(1-\frac{1}{n})}\right) + C,$$

   where C is the total cost for performing services. We analyze two cases:

   Case 1 – $R_{>\left(1-\frac{1}{n}\right)}$. Serving each request ρ in this set individually costs at most $\hat{c}_\rho(t') + c(P(v_\rho))$, where v_ρ is the request's node and $P(v_\rho)$ is the path from v_ρ to the root of the tree. Since $\hat{c}_\rho(t') \leq 1$ as $t' \in T_\rho$, and the cost of that path is at most 2 by the 2-decreasing property, the total cost for serving ρ is at most 3. In addition, since

   $$cost\left(R_{>(1-\frac{1}{n})}\right) > \left(1 - \frac{1}{n}\right) \cdot \left|R_{>(1-\frac{1}{n})}\right|,$$

   then the total cost in $\mathrm{OPT}_{\mathrm{new}}$ for these requests (including the cost of performing the services and the cost of the penalties) is at most

   $$3 \cdot \left|R_{>(1-\frac{1}{n})}\right| < \frac{3}{1-\frac{1}{n}} \cdot cost\left(R_{>(1-\frac{1}{n})}\right) \leq 6 \cdot cost\left(R_{>(1-\frac{1}{n})}\right)$$

   for $n \geq 2$.

   Case 2 – $R_{\leq\left(1-\frac{1}{n}\right)}$. First, we show that in this case, t' is the end of the global interval containing t. Indeed, since both t and t' lie in the same interval of length $\frac{1}{n}$, we have

   $$\hat{c}_\rho(t') \leq c_\rho(t) + \frac{1}{n} \leq \left(1 - \frac{1}{n}\right) + \frac{1}{n} = 1.$$

Therefore, $t' \in T_\rho$ as desired. Second, since $c_\rho(t) \geq \frac{1}{n}$ from the first step, we also conclude that $\hat{c}_\rho(t') \leq 2c_\rho(t)$. Since $t' \geq t$, all requests in $R_{\leq(1-\frac{1}{n})}$ that were served at time t in $\mathrm{OPT}_{\mathrm{org}}$ can be served together at time t'. This implies that these requests are served using the same number and structure of services as in $\mathrm{OPT}_{\mathrm{org}}$, and therefore no additional service cost is incurred for $R_{\leq(1-\frac{1}{n})}$. Combining both cases, we get that

$$\mathrm{OPT}_{\mathrm{new}} \leq 6 \cdot cost\left(R_{>(1-\frac{1}{n})}\right) + 2 \cdot cost\left(R_{\leq(1-\frac{1}{n})}\right) + C \leq 6 \cdot \mathrm{OPT}_{\mathrm{org}}. \quad \blacktriangleleft$$

Maintaining the reductions in an online fashion

We assume that n and W are known in advance, and later explain how to get rid of this assumption, losing only a constant factor in the competitive ratio. We also take care of maintaining Lipschitz constant 1. The **first step** is applied independently to each request, without requiring any knowledge of other requests. Therefore, this step remains valid as new requests arrive online. The **second step**, which constructs a common discrete domain T, can also be maintained incrementally in the online setting. Specifically, we build T in the order that requests arrive, by continuously adding new global intervals of length $1/n$, from each new request's penalty function.

3.1 Reduction to windows of length W

We can restrict attention to time intervals of length W only, losing a constant factor in the competitive ratio. As a result, we may assume that the number of requests is $\bar{n}$, the maximum number of requests that arrive within any window of length W. Typically, $\bar{n} \ll n$.

To do so, we maintain two sequential, disjoint *live* copies of the algorithm at any time, each spanning an interval of length $2W$ and overlapping by W. Specifically, every request arriving in the interval $[0, W)$ is assigned to the first copy, and every request arriving in $[W, 2W)$ is assigned to the second copy. Requests assigned to the first copy may be served up to time $2W$, while requests in the second copy may be served up to time $3W$.

More generally, for every $z \in \mathbb{Z}_{\geq 0}$, the even copy indexed by $2z$ contains all requests arriving in the interval $[2zW, (2z+1)W)$, and the odd copy indexed by $2z+1$ contains all requests arriving in $[(2z+1)W, (2z+2)W)$. Each copy is active for a duration of $2W$, and the algorithm is executed independently on each active copy, considering only the requests assigned to it.

Note that the feasibility of a solution is not violated by this partitioning method, as each request can still be served in its original service time. The total cost incurred by the algorithm is at most the sum of the costs over all copies. Since the windows of all even copies are disjoint, and likewise the windows of all odd copies are disjoint, the optimal solution must pay at least the maximum cost among the even copies or the odd copies. Consequently, this transformation increases the competitive ratio by at most a factor of 2.

3.2 Eliminating the need to know n and W in advance

Our algorithm requires an upper bound on the global discretization scale $\frac{1}{n}$ to extract the potential time units for service from each penalty function. Thus, we now explain why one may assume prior knowledge of the number of requests n, and the maximum length of a service window W, while incurring only a constant-factor loss in the competitive ratio. We also note that due to the normalization, we consider an update of the Lipschitz parameter by updating the parameter W.

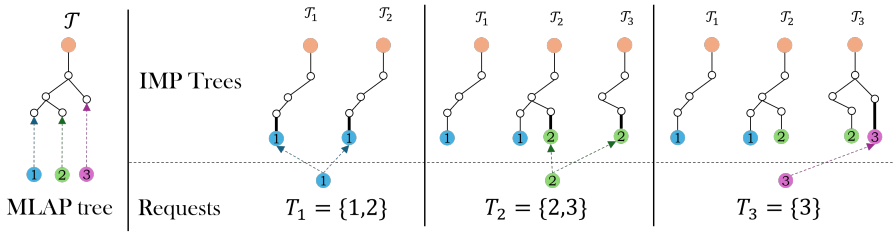

Figure 4 Each discrete time point induces a copy of the MLAP tree containing the path from the request's node to the root, plus a new node and edge (in bold) encoding the penalty. The first (blue) request has two service points, corresponding to trees $\mathcal{T}_1$ and $\mathcal{T}_2$. The second (green) request also has two service points: the first reuses an existing tree, adding only the missing part of the path. The third (purple) request has a single service point, corresponding to a new node added to an existing tree.

▶ **Theorem 4.** *Let ALG be an $O(\beta)$-competitive algorithm for MLAP under the assumption that the parameters n and W are known in advance, where $\beta = \log(n)\log(nDW)$, and where D is fixed and known in advance. Then, there exists an algorithm ALG′ for MLAP that achieves the same asymptotic bound $O(\beta)$ without prior knowledge of n and W.*

Since this proof is technical, we presented it in Appendix A.

4 Reduction from MLAP to IMP

In this section, we present the third reduction, from MLAP with 2-decreasing trees and discrete penalty functions to the Incremental Multicast Problem. This reduction is novel and establishes a connection between these two classic online problems.

▶ **Theorem 5.** *Let ALG_{IMP} be an α-competitive algorithm for IMP(n,m), where n is the number of requests and m is the total number of edges in the resulting trees. Then, there exists an α-competitive algorithm ALG_{MLAP} for MLAP$_{2\text{-}Discrete}(n,D,T)$, where D is the maximum depth of the trees in $\mathcal{T}$, T is the final set of potential service points across all penalty functions and n is the number of requests. Moreover, the number of edges satisfies $m = O(nD|T|)$.*

The key idea behind this reduction is the analogy between potential service time units and facilities. A client's connection cost to a facility is derived from the penalty cost of serving the corresponding request at that time unit.

Proof.

(1) Input MLAP $\Longrightarrow$ IMP. Let $(\mathcal{T}_{MLAP}, \mathcal{R}_{MLAP})$ be an instance of MLAP$_{\text{2-Discrete}}$, where $\mathcal{T}_{MLAP}$ is a 2-decreasing weighted rooted tree and $\mathcal{R}_{MLAP}$ is a sequence of requests arriving over time. For simplicity, we assume that $\mathcal{T}_{MLAP}$ consists of a single 2-decreasing tree rooted at node r, with edge cost function c. We denote by $c(v)$ the cost of the edge connecting node v to its parent. The reduction naturally extends to a family of disjoint trees.

Algorithm ALG_{IMP} starts with an empty family of trees $\mathcal{T}_{IMP}$ and an empty sequence of requests $\mathcal{R}_{IMP}$. Remark that each request $\rho \in \mathcal{R}_{MLAP}$ is defined by the tuple (v_ρ, a_ρ, c_ρ), where v_ρ is a leaf node in $\mathcal{T}_{MLAP}$ associated with the request, a_ρ is the arrival time of the request, and c_ρ is a discrete penalty function defined over the set of potential service time units T_ρ. We denote by $P(v_\rho)$ the path from v_ρ to the root r.

Each time unit $t \in T_\rho$ corresponds to a tree $\mathcal{T}'_t \in \mathcal{T}_{IMP}$ rooted at node r_t. In each such tree, the request is associated with a new leaf node $v_\rho^{(t)}$. This tree is a duplication of the original MLAP tree $\mathcal{T}$, restricted to the path $P(v_\rho)$ plus the new leaf node.

Formally, for every request $\rho \in \mathcal{R}_{MLAP}$ (in the order of arrival), we create a corresponding request $\rho' \in \mathcal{R}_{IMP}$ defined as $(S_{\rho'}, \{\mathcal{P}_v\}_{v \in S_{\rho'}}, c_{\rho'})$ with the following components:

- $S_{\rho'} = \{v_\rho^{(t)} \mid t \in T_\rho\}$.
- $\{\mathcal{P}_{v_\rho^{(t)}}\}_{v_\rho^{(t)} \in S_{\rho'}}$ is the collection of the incremental paths. Each path $\mathcal{P}_{v_\rho^{(t)}}$ is constructed by duplicating the path $P(v_\rho)$ from $\mathcal{T}$ into the tree $\mathcal{T}'_t \in \mathcal{T}_{IMP}$, taking only the portion of the path that does not already exist in that tree. Finally, a new leaf node $v_\rho^{(t)}$ is introduced, connected to the original node v_ρ.
- The cost function $c_{\rho'}$ on the new edges is inherited from the original edge costs in $\mathcal{T}$, except the only one edge connecting $v_\rho^{(t)}$ to v_ρ which has the corresponding penalty cost in that time $c_\rho(t)$.

We denote by $T_{\rho'} = \{r^{(t)} \mid t \in T_\rho\}$ the set of all roots corresponding to each node in $S_{\rho'}$. Note that T_ρ represents potential times to serve request ρ in the MLAP, while $T_{\rho'}$ represents the corresponding roots in the constructed IMP trees. Although the notation may seem confusing at first, the precise correspondence between these sets is the key to the reduction. Figure 4 illustrates the main idea of the reduction.

(2) Apply algorithm ALG_{IMP}. Upon the arrival of a request ρ, after converting it as described above, algorithm ALG_{MLAP} simulates algorithm ALG_{IMP} on the new request ρ' using the demand $(S_{\rho'}, T_{\rho'})$.

(3) Output IMP $\Longrightarrow$ MLAP. Let $E_{\rho'}$ denote the set of edges from $\mathcal{T}_{IMP}$ that ALG_{IMP} adds to the solution to satisfy ρ'. ALG_{MLAP} collects all these edges. At each time t for which the IMP solution contains edges from a tree $\mathcal{T}'_t$, ALG_{MLAP} performs a service. This service consists of the rooted subtree formed by those edges, and a request ρ is included in this service only if its extra edge also appears in the IMP solution. In particular, if the IMP algorithm adds edges belonging to a tree corresponding to a past time unit, those edges are ignored in the MLAP solution. Therefore, the feasibility condition of making a service before time is expired, is preserved.

In addition, each new request connects only to trees corresponding to its arrival time and future time units. Since the IMP algorithm satisfies each request immediately and never removes edges once added, all such edges remain in both solutions and collectively yield a feasible solution for the MLAP.

The cost of algorithm ALG_{MLAP} is at most the cost of algorithm ALG_{IMP}, since every payment of ALG_{MLAP} corresponds directly to a payment of ALG_{IMP}, either as part of the service cost (inner edges copied from $\mathcal{T}$) or as a penalty (the extra edge corresponding to the request). Note that some payments of ALG_{IMP} may be ignored in ALG_{MLAP}. From the opposite direction, using the reduction, any solution for the MLAP can be directly transformed into a valid solution for the corresponding IMP instance. Therefore, $OPT_{\text{IMP}} \leq OPT_{\text{MLAP}}$, so, $ALG_{\text{MLAP}} \leq ALG_{\text{IMP}} \leq \alpha \cdot OPT_{\text{IMP}} \leq \alpha \cdot OPT_{\text{MLAP}}$. Finally, the total number of edges in the resulting IMP instance is at most $O(nD|T|)$, since each request contributes at most $D + 1$ edges across at most $|T|$ trees. ◀

5 Algorithm for the online IMP

To complete the discussion on the MLAP with arbitrary penalty functions and to obtain a full algorithm for that problem, it remains to describe an algorithm for the IMP. In this section, we present a randomized algorithm for the online IMP, based on the randomized algorithm of Alon et al. for the online multicast problem [1]. The adapted algorithm yields the following theorem:

▶ **Theorem 6.** *There exists a randomized $O(\log n' \log m')$-competitive algorithm for the Incremental Multicast Problem in trees, where n' is the current number of requests and m' is the current number of edges in the trees.*

Most of the algorithmic components and their analysis are naturally adapted from the work of Alon et al. [1]. We present the algorithm construction together with all relevant lemmas and theorems. Since our setting is a generalization of the multicast problem, we also rewrite the proofs and include them in Appendix B.

The most fundamental difference between the problems, due to the incremental nature of the IMP, concerns the knowledge of the structure of the graph in advance. The approach by Alon et al. [1] relies on constructing a fractional feasible solution, which is subsequently rounded to yield a randomized integral solution. This algorithm uses the number of edges m to normalize each edge's cost and weight (fraction). To address this, we apply the same doubling technique as mentioned in Subsection 3.2: we start with an estimate $m = 2$. Each time the estimation of m is exceeded, we square its current real value. After every such an update, we start a new copy of the problem and simulate the algorithm from the beginning. We take to the solution all edges that were taken in all the copies. This technique introduces an additional constant factor to the performance compared with the setting where the final m is known in advance. Notice that at each moment, the algorithm uses the current number of edges m', and this parameter also reflected in the competitive ratio.

At a high level, the randomized algorithm for the IMP proceeds as follows:

Randomized Algorithm for the IMP on $(S_\rho, \{\mathcal{P}_v\}_{v \in S_\rho}, c_\rho)$

1. Preprocessing step. Remove every edge $e \in \{\mathcal{P}_v\}_{v \in S_\rho}$ with $c_\rho(e) > \alpha$, together with all edges in the path downwards until the request node. Add to the solution all edges with $c_\rho(e) \leq \frac{\alpha}{m}$. Normalize the cost of the remaining edges to lie in $[1, m]$. Further details and explanations can be found in Appendix B.1

2. Graph increment. Add to the current trees $\mathcal{T}$ all new nodes and edges in $\{\mathcal{P}_v\}_{v \in S_\rho}$. Each new edge is assigned an initial weight (fraction) of $\frac{1}{m}$.

3. Fractional solution. Perform a *weight augmentation step* for the demand (S_ρ, T_ρ) as described in Algorithm B.2.

4. Integral solution. Perform a *rounding step* on every tree rooted at a node in T_ρ as described in Algorithm B.3.

More details of the algorithm and its analysis are provided in Appendix B.

6 Lower Bound

We establish a lower bound of 4 on the competitive ratio for any deterministic algorithm, even in the special case where $D = 1$ (i.e., the TCP Acknowledgment Problem) and even when all penalty functions have only the values 0 or 1. One can show the same result with continuous penalty function (piecewise linear) that gets values in $[0, 1]$. Moseley et al. in [20] and Shmoys et al. [23] designed deterministic algorithms for the single-item JRP (equivalent to the TCP-AP) under the assumption that the penalty functions initially decrease and then increase, that are less than 4 competitive. Therefore, our lower bound shows that allowing *arbitrary* penalty functions, even with only two possible values, makes the problem strictly harder.

▶ **Theorem 7.** *Every deterministic algorithm for the TCP-AP with arbitrary penalty functions, even when each function has only values* 0 *and* 1*, is at least* $(4 - \epsilon)$*-competitive for every* $\epsilon > 0$*.*

The proof relies on the following arithmetic lemma, whose full proof is provided in the full version.

▶ **Lemma 8.** *For every* $\epsilon > 0$ *and for every sequence of real numbers* $\{c_k\}_{k=1}^{\infty}$ *(for consistency we define* $c_0 = 0$*) with* $c_k \geq 1$ *for all* $k \geq 1$*, there exists an* $n \in \mathbb{N}$ *such that*

$$\frac{s_n}{c_{n-1} + 1} > 4 - \epsilon,$$

where $s_n := \sum_{k=1}^{n} c_k$ *denote the partial sums of the sequence.*

We present a sketch of the proof here. The complete proof is provided in the full version.

Proof Sketch. Assume, toward a contradiction, that there exists a $(4 - \epsilon)$-competitive deterministic online algorithm ALG for some $\epsilon > 0$. We construct an adversarial input consisting of groups of requests released sequentially over epochs of increasing lengths. Within each epoch, the requests of a group are partitioned into subgroups, each having a distinct zero-penalty service time. Whenever ALG completes serving one group, the next group is released and the same recursive partitioning process is repeated. The sizes of the groups are chosen sufficiently large so that every partition contains an integer number of requests throughout the construction.

Using several standard simplifications, we may assume without increasing the cost of ALG that every request is served at a zero-penalty time, every service covers all requests whose penalty is zero at that time, and each group is completely served within a single epoch. Let c_k denote the epoch in which ALG finishes serving the k-th group. A key lemma shows that the competitiveness assumption implies the recurrence $c_k < 3c_{k-1} + 4$, and therefore $c_k \leq 4^k$ for every k. Consequently, serving group k forces ALG to incur a service cost of exactly c_k, implying that its total cost after serving the first k groups is $\sum_{i=1}^{k} c_i$.

On the other hand, by construction, all requests from previous groups remain free to serve at the first time unit of every subsequent epoch. Thus, an optimal offline algorithm can postpone all services and serve every request released up to group k in the first epoch in which group k appears, incurring cost at most $c_{k-1} + 1$. Applying the arithmetic lemma to the sequence c_k yields an index n for which

$$\frac{\sum_{i=1}^{n} c_i}{c_{n-1} + 1} > 4 - \epsilon,$$

contradicting the assumed competitiveness of ALG. Therefore, no deterministic online algorithm can achieve a competitive ratio smaller than 4. ◀

7 Conclusions and Open Problems

In this work, we initiate the study of the online Multi-Level Aggregation Problem (MLAP) with *arbitrary penalty functions*, which may increase and decrease multiple times. This setting goes beyond the standard assumption of monotone non-decreasing (delay) functions. To the best of our knowledge, this is the first work to analyze such general penalty functions in the context of MLAP. A central contribution is a discretization of continuous-time penalty functions, which enables a reduction to a new generalization of the Multicast Problem that we introduce, the *Incremental Multicast Problem* (IMP).

Several open questions remain. In particular, there is a significant gap between the competitive ratio of our algorithm and the lower bound we establish. Closing this gap, either by designing improved algorithms or by strengthening the lower bounds, remains an interesting direction, even for the special case $D = 1$, corresponding to the TCP Acknowledgment Problem. More broadly, our discretization-and-reduction framework suggests that similar techniques may be applicable to other online problems with arbitrary penalty functions, potentially leading to a general and clean methodology for handling such settings.

References

1 Noga Alon, Baruch Awerbuch, Yossi Azar, Niv Buchbinder, and Joseph Naor. A general approach to online network optimization problems. *ACM Trans. Algorithms*, 2(4):640–660, 2006. doi:10.1145/1198513.1198522.

2 Noga Alon, Baruch Awerbuch, Yossi Azar, Niv Buchbinder, and Joseph Naor. The online set cover problem. *SIAM J. Comput.*, 39(2):361–370, 2009. doi:10.1137/060661946.

3 Yossi Azar and Shahar Lewkowicz. Online joint replenishment problem with arbitrary holding and backlog costs. *CoRR*, abs/2507.16096, 2025. doi:10.48550/arXiv.2507.16096.

4 Yossi Azar and Noam Touitou. General framework for metric optimization problems with delay or with deadlines. In *60th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2019, Baltimore, Maryland, USA, November 9-12, 2019*, pages 60–71. IEEE Computer Society, 2019. doi:10.1109/FOCS.2019.00013.

5 Yossi Azar and Noam Touitou. Beyond tree embeddings - a deterministic framework for network design with deadlines or delay. In Sandy Irani, editor, *61st IEEE Annual Symposium on Foundations of Computer Science, FOCS 2020, Durham, NC, USA, November 16-19, 2020*, pages 1368–1379. IEEE, 2020. doi:10.1109/FOCS46700.2020.00129.

6 Marcin Bienkowski, Martin Böhm, Jaroslaw Byrka, Marek Chrobak, Christoph Dürr, Lukás Folwarczný, Lukasz Jez, Jirí Sgall, Kim Thang Nguyen, and Pavel Veselý. Online algorithms for multilevel aggregation. *Oper. Res.*, 68(1):214–232, 2020. doi:10.1287/opre.2019.1847.

7 Marcin Bienkowski, Martin Böhm, Jaroslaw Byrka, Marek Chrobak, Christoph Dürr, Luk'avs Folwarczn'y, Lukasz Jez, Jirí Sgall, Kim Thang Nguyen, and Pavel Veselý. New results on multi-level aggregation. *Theor. Comput. Sci.*, 861:133–143, 2021. doi:10.1016/J.TCS.2021.02.016.

8 Marcin Bienkowski, Jaroslaw Byrka, Marek Chrobak, Lukasz Jez, Dorian Nogneng, and Jirí Sgall. Better approximation bounds for the joint replenishment problem. In Chandra Chekuri, editor, *Proceedings of the Twenty-Fifth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2014, Portland, Oregon, USA, January 5-7, 2014*, pages 42–54. SIAM, 2014. doi:10.1137/1.9781611973402.4.

9 Marcin Bienkowski, Jaroslaw Byrka, Marek Chrobak, Lukasz Jez, Jirí Sgall, and Grzegorz Stachowiak. Online control message aggregation in chain networks. In Frank Dehne, Roberto Solis-Oba, and Jörg-Rüdiger Sack, editors, *Algorithms and Data Structures - 13th International Symposium, WADS 2013, London, ON, Canada, August 12-14, 2013. Proceedings*, volume 8037 of *Lecture Notes in Computer Science*, pages 133–145. Springer, 2013. doi:10.1007/978-3-642-40104-6_12.

10 Thomas Bosman and Neil Olver. Improved approximation algorithms for inventory problems. In Daniel Bienstock and Giacomo Zambelli, editors, *Integer Programming and Combinatorial Optimization - 21st International Conference, IPCO 2020, London, UK, June 8-10, 2020, Proceedings*, volume 12125 of *Lecture Notes in Computer Science*, pages 91–103. Springer, 2020. doi:10.1007/978-3-030-45771-6_8.

11 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. doi:10.1007/S00453-011-9567-5.

12 Niv Buchbinder, Moran Feldman, Joseph (Seffi) Naor, and Ohad Talmon. O(depth)-competitive algorithm for online multi-level aggregation. In Philip N. Klein, editor, *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. SIAM, 2017. doi:10.1137/1.9781611974782.80.

13 Niv Buchbinder, Tracy Kimbrel, Retsef Levi, Konstantin Makarychev, and Maxim Sviridenko. Online make-to-order joint replenishment model: Primal-dual competitive algorithms. *Oper. Res.*, 61(4):1014–1029, 2013. doi:10.1287/OPRE.2013.1188.

14 Maurice Cheung, Adam N. Elmachtoub, Retsef Levi, and David B. Shmoys. The submodular joint replenishment problem. *Math. Program.*, 158(1-2):207–233, 2016. doi:10.1007/S10107-015-0920-3.

15 Daniel R. Dooly, Sally A. Goldman, and Stephen D. Scott. TCP dynamic acknowledgment delay: Theory and practice (extended abstract). In Jeffrey Scott Vitter, editor, *Proceedings of the Thirtieth Annual ACM Symposium on the Theory of Computing, Dallas, Texas, USA, May 23-26, 1998*, pages 389–398. ACM, 1998. doi:10.1145/276698.276792.

16 Daniel R. Dooly, Sally A. Goldman, and Stephen D. Scott. On-line analysis of the TCP acknowledgment delay problem. *J. ACM*, 48(2):243–273, 2001. doi:10.1145/375827.375843.

17 Anna R. Karlin, Claire Kenyon, and Dana Randall. Dynamic TCP acknowledgement and other stories about e/(e-1). In Jeffrey Scott Vitter, Paul G. Spirakis, and Mihalis Yannakakis, editors, *Proceedings on 33rd Annual ACM Symposium on Theory of Computing, July 6-8, 2001, Heraklion, Crete, Greece*, pages 502–509. ACM, 2001. doi:10.1145/380752.380845.

18 Mathieu Mari, Michal Pawlowski, Runtian Ren, and Piotr Sankowski. Online multi-level aggregation with delays and stochastic arrivals. In Julián Mestre and Anthony Wirth, editors, *35th International Symposium on Algorithms and Computation, ISAAC 2024, December 8-11, 2024, Sydney, Australia*, volume 322 of *LIPIcs*, pages 49:1–49:20. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2024. doi:10.4230/LIPIcs.ISAAC.2024.49.

19 Jeremy McMahan. A d-competitive algorithm for the multilevel aggregation problem with deadlines. *CoRR*, abs/2108.04422, 2021. arXiv:2108.04422.

20 Benjamin Moseley, Aidin Niaparast, and R. Ravi. Putting off the catching up: Online joint replenishment problem with holding and backlog costs. In *Proceedings of the 2025 Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2025, New Orleans, LA, USA, January 12-15, 2025*, pages 3865–3883. SIAM, 2025. doi:10.1137/1.9781611978322.130.

21 Viswanath Nagarajan and Cong Shi. Approximation algorithms for inventory problems with submodular or routing costs. *Math. Program.*, 160(1-2):225–244, 2016. doi:10.1007/S10107-016-0981-Y.

22 Steven S. Seiden. A guessing game and randomized online algorithms. In F. Frances Yao and Eugene M. Luks, editors, *Proceedings of the Thirty-Second Annual ACM Symposium on Theory of Computing, May 21-23, 2000, Portland, OR, USA*, pages 592–601. ACM, 2000. doi:10.1145/335305.335385.

23 David B. Shmoys, Varun Suriyanarayana, and Seeun William Umboh. Improved online algorithms for inventory management problems with holding and delay costs: Riding the wave makes things simpler, stronger, & more general. In Kasper Green Larsen and Barna Saha, editors, *Proceedings of the 2026 Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2026, Vancouver, BC, Canada, January 11-14, 2026*, pages 2726–2742. SIAM, 2026. doi:10.1137/1.9781611978971.100.

A Eliminating the need to know n and W in advance: Proof of Theorem 3.2

Proof. We apply a variant of the doubling technique that simultaneously handles all unknown parameters. The algorithm dynamically maintains estimates of n and W, updating them whenever they are violated. At any moment, the algorithm tracks the true values of these parameters, but uses them only to update the estimates at the beginning of a new phase.

Let n_i, and W_i denote the estimates used in phase i, and let $n(t)$, and $W(t)$ be the true parameter values at time t.

▶ Note 9. The parameter W is defined only after normalizing the penalty function via the following steps:

1. Divide the function by the minimum positive edge weight incident to the root.
2. Subtract the minimum value of the function from all values.
3. Rescale the function to be 1-Lipschitz. That is, if $f(x)$ is an L-Lipschitz penalty function, we instead use the function $f\left(\frac{x}{L}\right)$, and define W accordingly.

After the arrival of the first request, we set W_1 to the true values revealed by that request, and set $n_1 = 2$. Whenever one of the estimates is violated, a new phase begins, the algorithm is restarted and the estimates are updated as follows:

- If $n(t) > n_i$, update $n_{i+1} \leftarrow (n(t))^2$.
- If $W(t) > W_i$, update $W_{i+1} \leftarrow n(t) \cdot D \cdot (W(t))^2$.

Two key facts:

1. For every request ρ, since the original arbitrary penalty function have values from 0 to 1 within its service window, it follows that $w_\rho \geq 1$ and hence at every time t, $W(t) = \max_{\rho \leq t} w_\rho \geq 1$.
2. If several parameters are updated simultaneously, since the update rules use the *current* value of each parameter, there is no circular dependency.

To show that all updates are feasible, we use the following lemma:

▶ **Lemma 10.** *For every phase $i \geq 1$:*

1. *Each parameter strictly increases when updated: $x_{i+1} > x_i, \quad x \in \{n, W\}$.*
2. *For every time t during phase i, each estimate upper bounds the true value: $n(t) \leq n_i, \quad W(t) \leq W_i$.*

Proof. In the first phase, W_1 is equal its true initial value and $n_1 = 2$ trivially upper bounds the single request observed so far. We check each possible update:

1. If $n(t) > n_i$, then since $n_1 = 2$, $n_{i+1} = (n(t))^2 > n(t) > n_i$.
2. If $W(t) > W_i$, then using $W(t) \geq W_1$, $n(t) \geq 1$, $D \geq 1$, and $W(t) \geq 1$, $W_{i+1} = n(t) \cdot D \cdot (W(t))^2 \cdot \geq W(t) > W_i$. ◀

Next we analyze the change in the competitive ratio after each update. The competitive ratio of the algorithm in phase i is $\gamma\beta_i$, where γ is some constant, $\beta_i = \log(n_i)\,\log(n_i D W_i)$, while using estimates n_i and W_i. Then,

$$ALG' = \sum_i ALG_i \;\leq\; \sum_i \gamma\beta_i \cdot OPT_i \;\leq\; OPT \cdot \gamma \sum_i \beta_i. \tag{1}$$

We now show that the competitiveness at least doubles in every phase, i.e., for every phase i, $\beta_{i+1} \geq 2\beta_i$. We analyze all three update cases.

Case 1: Update triggered by n. Using $n(t) > n_i$,

$$\beta_{i+1} = \log((n(t))^2)\,\log\left((n(t))^2 D W_i\right) \geq 2\log(n_i)\,\log(D n_i W_i) = 2\beta_i.$$

Case 2: Update triggered by W. Using $W(t) > W_i \geq W_1$,

$$\beta_{i+1} = \log(n_i)\,\log\left(n_i D \cdot n(t) \cdot D \cdot (W(t))^2\right) \geq \log(n_i)\,\log\left(n_i^2 D^2 W_i^2\right) = 2\beta_i.$$

Let $\hat{n}$ and $\hat{W}$ be the final parameter values at the last phase, and let $\hat{\beta}$ denote the competitive ratio of the algorithm at the last phase. By the update rules, $\hat{n} \leq n^2$ and $\hat{W} \leq nDW^2$. Thus,

$$\begin{aligned}\hat{\beta} &= \log(\hat{n})\log\left(\hat{n}\hat{W}\right) \leq \log(n^2)\,\log\left(n^2 \cdot nDW^2\right)\\ &\leq 2\log(n)\,\log\left(n^3 D^3 W^3\right) = 6\log(n)\,\log(nDW) = 6\beta.\end{aligned}$$

Since $\beta_{i+1} \geq 2\beta_i$ and $\hat{\beta} \leq 6\beta$, $\sum_i \beta_i \leq 12\,\beta$. Combining with equation 1 completes the proof. ◀

B Randomized algorithm for the IMP: Section 5

B.1 Preprocessing step

The original algorithm assumes that the cost of the optimal fractional solution, denoted by α, is known in advance. As in the case of assuming knowledge of the number of edges m, we can apply a doubling technique and assume that α is known within a factor of 2. After the first request arrives, we initialize α to be the smallest cost among all new edges introduced by that request. Whenever this estimate is violated, we double α. This increases the overall cost of the solution by at most a constant factor of 2.

After any violation of the estimates of either α or m, we discard the current weights assigned to the edges, update our estimate accordingly, and resume the algorithm. Note that the online setting does not allow edge weights to decrease. Thus, when we say we "forget" past weights, we mean that in each iteration, the algorithm only uses the weights assigned during that round. However, at any time, the actual weight of an edge is the maximum weight it has ever been assigned in any previous iteration.

Given that the number of edges m and the optimal value α are known, we may assume that all edge costs lie between 1 and m. To see it, we use the following observation:

▶ **Observation 11.** *In any fractional optimal solution, no edge with cost larger than α receives positive weight.*

Proof. Suppose otherwise, let edge e have $w(e) > 0$ while $c(e) > \alpha$. Because the total cost is at most α, it must hold that $w(e) < 1$. Now, define a modified fractional solution w' by setting $w'(e) = 0$ and scaling all other weights as

$$w'(e') = \frac{w(e')}{1 - w(e)}, \quad e' \neq e.$$

This yields a feasible solution of cost $\frac{\alpha - c(e)w(e)}{1-w(e)} < \frac{\alpha(1-w(e))}{1-w(e)} = \alpha$, contradicting the optimality of OPT. ◀

Therefore, all edges with cost greater than α can be safely ignored by removing them and all nodes connected below them in the tree. Conversely, all edges with cost less than α/m may be included automatically in the solution, increasing the solution cost by at most an additive α. Thus, after this preprocessing step, the remaining edges have costs in the range $[\alpha/m, \alpha]$. Finally, by multiplying with m/α, we normalize these costs to lie between 1 and m.

B.2 Fractional Solution

We present a fractional algorithm that is $O(\log m')$-competitive, where m is the number of edges in all trees in $\mathcal{T}$. To satisfy a demand (S_ρ, T_ρ), the algorithm performs the *weight augmentation* step described in Algorithm B.2. We denote the cost of an edge by c_e and the weight of an edge by w_e.

Algorithm: Weight Augmentation Step for a demand (S_ρ, T_ρ)

while maximum flow from S_ρ to T_ρ is less than 1
  for each edge e in a minimum cut between S_ρ and T_ρ
    $w_e \leftarrow w_e \cdot \left(1 + \frac{1}{c_e}\right)$

The performance of the fractional solution is summarized in the following theorem:

▶ **Theorem 12.** *The fractional algorithm satisfies all requests and $O(\log m)$-competitive for the fractional IMP.*

To prove this theorem, we use the following lemma:

▶ **Lemma 13.** *The total number of weight-augmentation steps performed by the algorithm is at most $\alpha + \alpha \log m = O(\alpha \log m)$, where α is the cost of an optimal solution, and m is the number of edges in $\mathcal{T}$.*

This lemma is the heart of the analysis of the fractional solution. It is based on the following potential function: $\Phi = \sum_{e \in E} c_e w_e^* \log w_e$, where w_e^* is the weight of edge e in the optimal solution. Although the full graph is not known in advance, for analysis purposes the lemma considers the final set of edges E retrospectively. If an edge does not exist yet in the graph at a given moment, its weight is treated as the initial weight.

Proof of Lemma 13. We define the following potential function: $\Phi = \sum_{e \in E} c_e w_e^* \log(w_e)$, where E is the set of edges in the graph, and w_e^* is the weight assigned to edge e in the optimal (fractional) solution. We show 3 properties of Φ:

1. **Initial value:** Initially, $w_e = \frac{1}{m}$ for every e, therefore:

$$\Phi_0 = \sum_{e \in E} c_e w_e^* \log\left(\frac{1}{m}\right) \geq -\alpha \log m.$$

2. **Upper bound:** During the execution of the algorithm, no weight w_e exceeds $1 + 1/c_e$, since otherwise it would not be a part of an augmentation step. Since $c_e \geq 1$, for every edge, we have $w_e \leq 2$. Thus,

$$\Phi \leq \sum_{e \in E} c_e w_e^* \log(2) \leq \alpha$$

3. **Augmentation increases Φ by at least 1:** Let $\mathcal{C}$ be a minimal cut where augmentation occurs. Then,

$$\begin{aligned}\Delta\Phi &= \sum_{e \in \mathcal{C}} c_e w_e^* \log\left(w_e\left(1 + \frac{1}{c_e}\right)\right) - \sum_{e \in \mathcal{C}} c_e w_e^* \log(w_e) \\ &= \sum_{e \in \mathcal{C}} c_e w_e^* \log\left(1 + \frac{1}{c_e}\right) \geq \sum_{e \in \mathcal{C}} w_e^* \geq 1,\end{aligned}$$

where the last two inequalities holds since (1) $x \log(1 + \frac{1}{x})$ is increasing function where $x \geq 1$, and (2) the optimal solution satisfy the request so every cut has total weight of at least 1.

From properties (1) and (2), the value of Φ increases by at most $\alpha + \alpha \log m$ during the run of the algorithm. From property (3), each augmentation increases Φ by at least 1. Therefore, the total number of augmentation steps during the run of the algorithm is at most $\alpha + \alpha \log m = O(\alpha \log m)$. ◀

Combining the last lemma with the simple observation that any weight augmentation step increases the value of the algorithm by at most 1, gives us the desired result.

Proof of Theorem 12. We show that during the execution of the algorithm, the total cost incurred by the algorithm satisfies $ALG \leq \alpha \log m + 2\alpha = O(\alpha \log m)$ We analyze the change in the algorithm's cost throughout the different steps performed during its execution. Specifically, we consider the impact of each of the following operations:

- **Initial Condition:** At the beginning, every edge has weight $1/m$. Therefore, $ALG \leq 1 \leq \alpha$.
- **Augmentation Step:** Let $\mathcal{C}$ be the set of edges involved in a weight augmentation step (a minimal cut). These edges have a total weight strictly less than 1. In the augmentation, each edge e in $\mathcal{C}$ is increased by $\frac{w_e}{c_e}$, incurring a cost of $c_e \cdot \frac{w_e}{c_e} = w_e$. Hence, the increase in cost during this step is $\sum_{e \in \mathcal{C}} w_e < 1$. From the previous lemma, we know that the number of augmentation steps is at most $\alpha \log m + \alpha$. Therefore, the total increase in ALG due to augmentations is at most $\alpha \log m + \alpha$.

Overall, the value of ALG is at most $\alpha \log m + 2\alpha = O(\alpha \log m)$. ◀

B.3 Randomized Solution

We now introduce the *rounding step*, which is performed after each weight augmentation step. For every tree T associated with a request, the algorithm executes the rounding step on T. Specifically, a random threshold is sampled for that tree, and this threshold determines which of its edges are rounded (i.e., included in the solution). The procedure is described in Algorithm B.3.

Algorithm: Rounding Step on a Tree $T \in \mathcal{T}$

1. Random thresholds. Sample $q = 2\lceil \log(n' + 1) \rceil$ independent random variables $X(T, j)$ for $1 \leq j \leq q$, each drawn uniformly from $[0, 1]$. Define the threshold of T as $\theta(T) := \min_{1 \leq j \leq q} X(T, j)$.
2. Edge rounding. Include in the solution every edge $e \in T$ whose fractional weight satisfies $w_e > \theta(T)$.

Let α denote the value of the optimal solution, m' the current number of edges in the trees, and n' the current number of requests. Note that the algorithm does not require knowledge of the final number of requests. Instead, it uses the current value of n', and as new requests arrive, more random variables are added to each tree. Consequently, the threshold of each tree can only decrease over time, which means that additional edges may be added to the solution, but once an edge is included, it remains in the solution.

▶ **Lemma 14.** *At any moment during the run of the algorithm, the following hold:*
1. *The expected cost of the algorithm is $O(\alpha \log n' \log m')$.*
2. *For any request ρ, the probability that ρ is not served is at most $\frac{1}{n'^2}$.*

Proof of Lemma 14. For the first part, for an edge e, define $Y(e,j) := [w_e > X(T,j)]$. Since each random variable is chosen uniformly in $[0,1]$, the probability and the expectation of the indicator $Y(e,j)$ is at most w_e. Therefore,

$$ALG = \mathbb{E}\left[\sum_{e\in E} c_e\right] \le \sum_{e\in E}\sum_{j=1}^{q} c_e \cdot \mathbb{E}\left[Y(e,j)\right] \le \sum_{e\in E}\sum_{j=1}^{q} c_e w_e = q\sum_{e\in E} w_e c_e$$
$$\le q \cdot O(\alpha \log m) = O(\alpha \log n' \log m),$$

where the last inequality uses the result of the fractional solution. For the second part, fix a request ρ and let T_ρ be the roots of the trees that associated with S_ρ (the nodes of ρ). From the feasibility of the fractional solution, we know that the total flow from S_ρ to T_ρ is at least 1. For every $v \in S_\rho$, let $f_\rho^{T_v}$ be the total flow from v to its corresponded root in the tree T_v. So, we can write: $\sum_{v\in S_\rho} f_\rho^{T_v} \ge 1$. The probability that request ρ is not satisfied using the node v is at most the probability that $f_\rho^{T_v} < \theta(T_v)$. For any $1 \le j \le q$, the probability that the value $f_\rho^{T_v}$ is at most the value of $X(T,j)$ is $1 - f_\rho^{T_v}$. Since all random variables are independent, the probability that ρ is not satisfied by any of its associated trees due to the j'th random variable is at most

$$\prod_{v\in S_\rho}(1 - f_\rho^{T_v}) \le e^{-\sum_{v\in S_\rho} f_\rho^{T_v}} \le \frac{1}{e}.$$

Therefore, after $q = 2\lceil \log(n'+1)\rceil$ repetitions of independent random variables for each tree, the probability that ρ is not served is at most $\left(\frac{1}{e}\right)^q \le \frac{1}{n'^2}$. ◀

The lemma shows that the randomized IMP algorithm yields a feasible solution with high probability while preserving a good competitive ratio. To ensure feasibility, if a request ρ is not served, the algorithm serves it using the tree with minimum path cost, adding at most OPT. Since this occurs with probability at most $\frac{1}{n'^2}$, the expected extra cost is negligible and does not affect the asymptotic bound. The full proof appears in Appendix B.3. Overall, this gives an $O(\log n' \log m')$-competitive randomized algorithm for IMP.