Online Algorithms for Multi-Level Aggregation*

Marcin Bienkowski[1], Martin Böhm[2], Jaroslaw Byrka[3], Marek Chrobak[4], Christoph Dürr[5], Lukáš Folwarczný[6], Łukasz Jeż[7], Jiří Sgall[8], Nguyen Kim Thang[9], and Pavel Veselý[10]

1 **Institute of Computer Science, University of Wrocław, Wrocław, Poland**
2 **Computer Science Institute, Charles University, Prague, Czech Republic**
3 **Institute of Computer Science, University of Wrocław, Wrocław, Poland**
4 **Department of Computer Science, University of California, Riverside, USA**
5 **Sorbonne Universités, UPMC Univ Paris 06, CNRS, LIP6, Paris, France**
6 **Computer Science Institute, Charles University, Prague, Czech Republic**
7 **Institute of Computer Science, University of Wrocław, Wrocław, Poland**
8 **Computer Science Institute, Charles University, Prague, Czech Republic**
9 **IBISC, Université d'Evry Val d'Essonne, Evry, France**
10 **Computer Science Institute, Charles University, Prague, Czech Republic**

Abstract

In the *Multi-Level Aggregation Problem* (MLAP), requests arrive at the nodes of an edge-weighted tree $\mathcal{T}$, and have to be served eventually. A *service* is defined as a subtree X of $\mathcal{T}$ that contains its root. This subtree X serves all requests that are pending in the nodes of X, and the cost of this service is equal to the total weight of X. Each request also incurs waiting cost between its arrival and service times. The objective is to minimize the total waiting cost of all requests plus the total cost of all service subtrees. MLAP is a generalization of some well-studied optimization problems; for example, for trees of depth 1, MLAP is equivalent to the TCP Acknowledgment Problem, while for trees of depth 2, it is equivalent to the Joint Replenishment Problem. Aggregation problem for trees of arbitrary depth arise in multicasting, sensor networks, communication in organization hierarchies, and in supply-chain management. The instances of MLAP associated with these applications are naturally online, in the sense that aggregation decisions need to be made without information about future requests.

Constant-competitive online algorithms are known for MLAP with one or two levels. However, it has been open whether there exist constant competitive online algorithms for trees of depth more than 2. Addressing this open problem, we give the first constant competitive online algorithm for networks of arbitrary (fixed) number of levels. The competitive ratio is $O(D^4 2^D)$, where D is the depth of $\mathcal{T}$. The algorithm works for arbitrary waiting cost functions, including the variant with deadlines. We include several additional results in the paper. We show that a standard lower-bound technique for MLAP, based on so-called *Single-Phase* instances, cannot give super-constant lower bounds (as a function of the tree depth). This result is established by giving an online algorithm with optimal competitive ratio 4 for such instances on arbitrary trees. We also study the MLAP variant when the tree is a path, for which we give a lower bound of 4 on the competitive ratio, improving the lower bound known for general MLAP. We complement this with a matching upper bound for the deadline setting.

1998 ACM Subject Classification F.1.2 Modes of Computation – Online computation, F.2.2 Nonnumerical Algorithms and Problems – Sequencing and scheduling

* Research partially supported by NSF grants CCF-1536026, CCF-1217314 and OISE-1157129, Polish NCN grants DEC-2013/09/B/ST6/01538, 2015/18/E/ST6/00456, project 14-10003S of GA ČR and GAUK project 548214.

24th Annual European Symposium on Algorithms (ESA 2016).
Editors: Piotr Sankowski and Christos Zaroliagis;
Article No. 12; pp. 12:1–12:17

Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Keywords and phrases algorithmic aspects of networks, online algorithms, scheduling and resource allocation

Digital Object Identifier 10.4230/LIPIcs.ESA.2016.12

1 Introduction

Certain optimization problems can be formulated as aggregation problems. They typically arise when expensive resources can be shared by multiple agents, who incur additional expenses for accessing a resource. For example, costs may be associated with waiting until the resource is accessible, or, if the resource is not in the desired state, a costly setup or retooling may be required.

1-level aggregation

A simple example of an aggregation problem is the *TCP Acknowledgment Problem (TCP-AP)*, where control messages ("agents") waiting for transmission across a network link can be aggregated and transmitted in a single packet ("resource"). Such aggregation can reduce network traffic, but it also results in undesirable delays. A reasonable compromise is to balance the two costs, namely the number of transmitted packets and the total delay, by minimizing their weighted sum [15]. Interestingly, TCP-AP is equivalent to the classical Lot Sizing Problem studied in the operations research literature since the 1950s. (See, for example, [30].) In the offline variant of TCP-AP, that is when all arrival times of control messages are known beforehand, an optimal schedule for aggregated packets can be computed with dynamic programming in time $O(n \log n)$ [1]. In practice, however, packet aggregation decisions must be done on the fly, without any information about future message releases. This scenario is captured by the online variant of TCP-AP that has also been well studied; it is known that the optimal competitive ratio is 2 in the deterministic case [15] and $\mathrm{e}/(\mathrm{e}-1) \approx 1.582$ in the randomized case [17, 11, 28].

2-level aggregation

Another optimization problem involving aggregation is the *Joint Replenishment Problem (JRP)*, well-studied in operations research. JRP models tradeoffs that arise in supply-chain management. One such scenario involves optimizing shipments of goods from a supplier to retailers, through a shared warehouse, in response to their demands. In JRP, aggregation takes place at two levels: items addressed to different retailers can be shipped together to the warehouse, at a fixed cost, and then multiple items destined to the same retailer can be shipped from the warehouse to this retailer together, also at a fixed cost, which can be different for different retailers. Pending demands accrue waiting cost until they are satisfied by a shipment. The objective is to minimize the sum of all shipment costs and all waiting costs.

JRP is known to be $\mathbb{NP}$-hard [2], and even $\mathbb{APX}$-hard [25, 5]. The currently best approximation, due to Bienkowski et al. [6], achieves a factor of 1.791, improving on earlier work by Levi et al. [21, 23, 24]. In the deadline variant of JRP, denoted JRP-D, there is no cost for waiting, but each demand needs to be satisfied before its deadline. As shown in [5], JRP-D can be approximated with ratio 1.574.

For the online variant of JRP, Buchbinder et al. [10] gave a 3-competitive algorithm using a primal-dual scheme (improving an earlier bound of 5 in [9]) and proved a lower bound of

2.64, that was subsequently improved to 2.754 [6]. The optimal competitive ratio for JRP-D is 2 [6].

Multiple-level aggregation

TCP-AP and JRP can be thought of as aggregation problems on edge-weighted trees of depth 1 and 2, respectively. In TCP-AP, this tree is just a single edge between the sender and the recipient. In JRP, this tree consists of the root (supplier), with one child (warehouse), and any number of grandchildren (retailers). A shipment can be represented by a subtree of this tree and edge weights represent shipping costs. These trees capture the general problem on trees of depth 1 and 2, as the children of the root can be considered separately (see Section 2).

This naturally extends to trees of any depth D, where aggregation is allowed at each level. Multi-level message aggregation has been, in fact, studied in communication networks in several contexts. In multicasting, protocols for aggregating control messages (see [8, 3], for example) can be used to reduce the so-called *ack-implosion*, the proliferation of control messages routed to the source. A similar problem arises in energy-efficient data aggregation and fusion in sensor networks [16, 31]. Outside of networking, tradeoffs between the cost of communication and delay arise in message aggregation in organizational hierarchies [26]. In supply-chain management, multi-level variants of lot sizing have been studied [14, 19]. The need to consider more tree-like (in a broad sense) supply hierarchies has also been advocated in [20].

These applications have inspired research on offline and online approximation algorithms for multi-level aggregation problems. Becchetti et al. [4] gave a 2-approximation algorithm for the deadline case. (See also [9].) Pedrosa [27] showed, adapting an algorithm of Levi et al. [22] for the multi-stage assembly problem, that there is a $(2+\varepsilon)$-approximation algorithm for general waiting cost functions, where ε can be made arbitrarily small.

In the online case, Khanna et al. [18] gave a rent-or-buy solution (that serves a group of requests once their waiting cost reaches the cost of their service) and showed that their algorithm is $O(\log \alpha)$-competitive, where α is defined as the sum of all edge weights. However, they assumed that each request has to wait at least one time unit. This assumption is crucial for their proof, as demonstrated by Brito et al. [9], who showed that the competitive ratio of a rent-or-buy strategy is $\Omega(D)$, even for paths with D edges. The same assumption of a minimal cost for a request and a ratio dependent on the edge-weights is also essential for Vaya [29], who studies a variant of the problem with bounded bandwidth (the number of packets that can be served by a single edge in a single service).

The existence of a primal-dual $(2+\varepsilon)$-approximation algorithm [27, 22] for the offline problem suggests the possibility of constructing an online algorithm along the lines of [11]. Nevertheless, despite substantial effort of many researchers, the online multi-level setting remains wide open. This is perhaps partly due to impossibility of direct emulation of the cleanup phase in the primal-dual offline algorithms in the online setting, as this cleanup is performed in the "reverse time" order.

The case when the tree is just a path has also been studied. An offline polynomial-time algorithm that computes an optimal schedule was given in [7]. For the online variant, Brito et al. [9] gave an 8-competitive algorithm. This result was improved by Bienkowski et al. [7] who showed that the competitive ratio of this problem is between $2+\phi \approx 3.618$ and 5.

Table 1 Previous and current bounds on the competitive ratios for MLAP for trees of various depths. Ratios written in bold are shown in this paper. Unreferenced results are either immediate consequences of other entries in the table or trivial observations. Asterisked ratios represent results for MLAP with arbitrary waiting cost functions, which, though not explicitly stated in the respective papers, are straightforward extensions of the corresponding results for MLAP-L.

	MLAP and MLAP-L		MLAP-D	
	upper	lower	upper	lower
depth 1	2^* [15]	2 [15]	1	1
rand. alg. for depth 1	1.582^* [17]	1.582 [28]	1	1
depth 2	3 [10]	2.754 [6]	2 [6]	2 [6]
fixed depth $D \geq 2$	$\mathbf{O(D^4 2^D)}$	2.754	$\mathbf{D^2 2^D}$	2
paths of arbitrary depth	5^* [7]	3.618 [7], **4**	**4**	**4**

1.1 Our Contributions

We study online competitive algorithms for multi-level aggregation. Minor technical differences notwithstanding, our model is equivalent to those studied in [9, 18], also extending the deadline variant in [4] and the assembly problem in [22]. We have decided to choose a more generic terminology to emphasize general applicability of our model and techniques.

Formally, our model consists of a tree $\mathcal{T}$ with positive weights assigned to edges, and a set of requests $\mathcal{R}$ that arrive in the nodes of $\mathcal{T}$ over time. These requests are served by subtrees rooted at the root of $\mathcal{T}$. Such a subtree X serves all requests pending at the nodes of X at cost equal to the total weight of X. Each request incurs a waiting cost, defined by a non-negative and non-decreasing function of time, which may be different for each request. The objective is to minimize the sum of the total service and waiting costs. We call this the *Multi-Level Aggregation Problem* (MLAP).

In most earlier papers on aggregation problems, the waiting cost function is linear, that is, it is assumed to be simply the delay between the times when a request arrives and when it is served. We denote this version by MLAP-L. However, most of the algorithms for this model extend naturally to arbitrary cost functions. Another variant is MLAP-D, where each request is given a certain deadline, has to be served before or at its deadline, and there is no penalty associated with waiting. This can be modeled by the waiting cost function that is 0 up to the deadline and $+\infty$ afterwards.

In this paper, we mostly focus on the online version of MLAP, where an algorithm needs to produce a schedule in response to requests that arrive over time. When a request appears, its waiting cost function is also revealed. At each time t, the online algorithm needs to decide whether to generate a service tree at this time, and if so, which nodes should be included in this tree.

The main result of our paper is an $O(D^4 2^D)$-competitive algorithm for MLAP for trees of depth D, presented in Section 4. A simpler $D^2 2^D$-competitive algorithm for MLAP-D is presented in Section 3. No competitive algorithms have been known so far for online MLAP for arbitrary depth trees, even for the special case of MLAP-D on trees of depth 3.

For both results we use a reduction of the general problem to the special case of trees with fast decreasing weights. For such trees we then provide an explicit competitive algorithm. While our algorithm is compact and elegant, it is not a straightforward extension of the 2-level algorithm. (In fact, we have been able to show that naïve extensions of the latter

algorithm are not competitive.) It is based on carefully constructing a sufficiently large service tree whenever it appears that an urgent request must be served. The specific structure of the service tree is then heavily exploited in an amortization argument that constructs a mapping from the algorithm's cost to the cost of the optimal schedule. We believe that these three new techniques: the reduction to trees with fast decreasing weights, the construction of the service trees, and our charging scheme, will be useful in further studies of online aggregation problems.

In Section 5 we study a version of MLAP, that we refer to as *Single-Phase* MLAP (or 1P-MLAP), in which all requests arrive at the beginning, but they also have a common *expiration time* that we denote by θ. Any request not served by time θ pays waiting cost at time θ and does not need to be served anymore. In spite of the expiration-date feature, it can be shown that 1P-MLAP can be represented as a special case of MLAP. 1P-MLAP is a crucial tool in all the lower bound proofs in the literature for competitive ratios of MLAP, including those in [10, 7], as well as in our lower bounds in Section 6. It also has a natural interpretation in the context of JRP (2-level MLAP), if we allow all orders to be canceled, say, due to changed market circumstances. In the online variant of 1P-MLAP all requests are known at the beginning, but the expiration time θ is unknown. For this version we give an online algorithm with competitive ratio 4, matching the lower bound. Since 1P-MLAP can be expressed as a special case of MLAP, our result implies that the techniques from [10, 7] cannot be used to prove a lower bound on the competitive ratio for MLAP larger than 4, and any study of the dependence of the competitive ratio on the depth D will require new insights and techniques.

In Section 6 we consider MLAP on paths. For this case, we give a 4-competitive algorithm for MLAP-D and we provide a matching lower bound. We show that the same lower bound of 4 applies to MLAP-L as well, improving the previous lower bound of 3.618 from [7].

Due to the page limit, most of the proofs will be given in the full version of the paper.

2 Preliminaries

Let $\mathcal{T}$ be a tree with root r. For any set of nodes $Z \subseteq \mathcal{T}$ and a node x, Z_x denotes the set of all descendants of x in Z; in particular, $\mathcal{T}_x$ is the subtree of $\mathcal{T}$ rooted at x. The parent of a node x is denoted $parent(x)$. The *depth of* x, denoted $depth(x)$, is the number of edges on the simple path from r to x. In particular, r is at depth 0. The depth D of $\mathcal{T}$ is the maximum depth of a node of $\mathcal{T}$.

We will deal with weighted trees in this paper. For $x \neq r$, by ℓ_x or $\ell(x)$ we denote the weight of the edge connecting node x to its parent. We assume that all these weights are positive. We extend this notation to r by setting $\ell_r = 0$. If Z is any set of nodes of $\mathcal{T}$, then the weight of Z is $\ell(Z) = \sum_{x \in Z} \ell_x$.

Definition of MLAP

A *request* ρ is specified by a triple $\rho = (\sigma_\rho, a_\rho, \omega_\rho)$, where σ_ρ is the node of $\mathcal{T}$ in which ρ is issued, a_ρ is the *arrival time* of ρ, and ω_ρ is the waiting cost function of ρ. We assume that $\omega_\rho(t) = 0$ for $t \leq a_\rho$ and $\omega_\rho(t)$ is non-decreasing for $t \geq a_\rho$. MLAP-L is the variant of MLAP with linear waiting costs; that is, for each request ρ we have $\omega_\rho(t) = t - a_\rho$, for $t \geq a_\rho$. In MLAP-D, the variant with deadlines, we have $\omega_\rho(t) = 0$ for $t \leq d_\rho$ and $\omega_\rho(t) = \infty$ for $t > d_\rho$, where d_ρ is called the *deadline* of request ρ.

In our algorithms for MLAP with general costs we will be assuming that all waiting cost functions are continuous. This is only for technical convenience and we discuss more general

waiting cost functions in the full version of the paper; we also show there that MLAP-D can be considered a special case of MLAP, and that our algorithms can be extended to the discrete-time model.

A *service* is a pair (X, t), where X is a subtree of $\mathcal{T}$ rooted at r and t is the time of this service. We will occasionally refer to X as the service tree (or just service) at time t, or even omit t altogether if it is understood from context.

An instance $\mathcal{J} = \langle \mathcal{T}, \mathcal{R} \rangle$ of the *Multi-Level Aggregation Problem* (MLAP) consists of a weighted tree $\mathcal{T}$ with root r and a set $\mathcal{R}$ of requests arriving at the nodes of $\mathcal{T}$. A *schedule* is a set S of services. For a request ρ, let (X, t) be the service in S with minimal t such that $\sigma_\rho \in X$ and $t \geq a_\rho$. We then say that (X, t) *serves* ρ and the *waiting cost* of ρ in S is defined as $\mathsf{wcost}(\rho, \mathsf{S}) = \omega_\rho(t)$. Furthermore, the request ρ is called *pending* at all times in the interval $[a_\rho, t]$. Schedule S is called *feasible* if all requests in $\mathcal{R}$ are served by S.

The cost of a feasible schedule S, denoted $\mathsf{cost}(\mathsf{S})$, is defined by

$$\mathsf{cost}(\mathsf{S}) = \mathsf{scost}(\mathsf{S}) + \mathsf{wcost}(\mathsf{S}),$$

where $\mathsf{scost}(\mathsf{S})$ is the total service cost and $\mathsf{wcost}(\mathsf{S})$ is the total waiting cost, that is

$$\mathsf{scost}(\mathsf{S}) = \sum_{(X,t)\in\mathsf{S}} \ell(X) \quad \text{and} \quad \mathsf{wcost}(\mathsf{S}) = \sum_{\rho\in\mathcal{R}} \mathsf{wcost}(\rho, \mathsf{S}).$$

The objective of MLAP is to compute a feasible schedule S for $\mathcal{J}$ with minimum $\mathsf{cost}(\mathsf{S})$.

Online algorithms

We use the standard and natural definition of online algorithms and the competitive ratio. We assume the continuous time model. The computation starts at time 0 and from then on the time gradually progresses. At any time t new requests can arrive. If the current time is t, the algorithm has complete information about the requests that arrived up until time t, but has no information about any requests whose arrival times are after time t. The instance includes a time horizon H that is not known to the online algorithm, which is revealed only at time $t = H$. At time H, all requests that are still pending must be served. (In the offline case, H can be assumed to be equal to the maximum request arrival time.)

If $\mathcal{A}$ is an online algorithm and $R \geq 1$, we say that $\mathcal{A}$ is *R-competitive* if $\mathsf{cost}(\mathsf{S}) \leq R \cdot \mathsf{opt}(\mathcal{J})$ for any instance $\mathcal{J}$ of MLAP, where S is the schedule computed by $\mathcal{A}$ on $\mathcal{J}$ and $\mathsf{opt}(\mathcal{J})$ is the optimum cost for $\mathcal{J}$.

Quasi-root assumption

Throughout the paper we will assume that r, the root of $\mathcal{T}$, has only one child. This is without loss of generality, because if we have an algorithm (online or offline) for MLAP on such trees, we can apply it independently to each child of r and its subtree. This will give us an algorithm for MLAP on arbitrary trees with the same performance. From now on, let us call the single child the *quasi-root* of $\mathcal{T}$ and denote it by q. Note that q is included in every (non-trivial) service.

Reduction to L-Decreasing Trees

One basic intuition that emerges from earlier works on trees of depth 2 (see [10, 9, 6]) is that the hardest case of the problem is when ℓ_q, the weight of the quasi-root, is much larger than the weights of leaves. For arbitrary depth trees, the hard case is when the weights of

nodes quickly decrease with their depth. We show that this is indeed the case, by defining the notion of L-decreasing trees that captures this intuition and showing that MLAP reduces to the special case of MLAP for such L-decreasing trees, increasing the competitive ratio by a factor of at most DL. This is a general result, not limited only to algorithms in our paper.

Formally, for $L \geq 1$, we say that $\mathcal{T}$ is *L-decreasing* if for each node $u \neq r$ and each child v of u we have $\ell_u \geq L \cdot \ell_v$. (The value of L used in our algorithms will be fixed later.)

▶ **Theorem 2.1.** *Assume that there exists an R-competitive algorithm $\mathcal{A}$ for MLAP (resp. MLAP-D) on L-decreasing trees (where R can be a function of D, the tree depth). Then there exists a (DLR)-competitive algorithm $\mathcal{B}$ for MLAP (resp. MLAP-D) on arbitrary trees.*

Proof. Fix the underlying instance $\mathcal{J} = (\mathcal{T}, \mathcal{R})$, where $\mathcal{T}$ is a tree and $\mathcal{R}$ is a sequence of requests in $\mathcal{T}$.

We start by constructing an L-decreasing tree $\mathcal{T}'$ on the same set of nodes. For any node $u \in \mathcal{T} - \{r\}$, the parent of u in $\mathcal{T}'$ will be the lowest (closest to u) ancestor w of u in $\mathcal{T}$ such that $\ell_w \geq L \cdot \ell_u$; if no such w exists, we take $w = r$. Note that $\mathcal{T}'$ may violate the quasi-root assumption, which does not change the validity of the reduction, as we may use independent instances of the algorithm for each child of r in $\mathcal{T}'$. Since in $\mathcal{T}'$ each node u is connected to one of its ancestors from $\mathcal{T}$, it follows that $\mathcal{T}'$ is a tree rooted at r with depth at most D. Obviously, $\mathcal{T}'$ is L-decreasing.

It follows that if a set of vertices X is a service subtree of $\mathcal{T}$, then it is also a service subtree for $\mathcal{T}'$. (Note that the actual topology of the trees induced by X in $\mathcal{T}$ and $\mathcal{T}'$ may be very different.) Thus, also any schedule for $\mathcal{J}$ is also a schedule for $\mathcal{J}' = (\mathcal{T}', \mathcal{R})$, which gives us that $\mathsf{opt}(\mathcal{J}') \leq \mathsf{opt}(\mathcal{J})$.

The algorithm $\mathcal{B}$ for $\mathcal{T}$ is defined as follows: On a request sequence $\mathcal{R}$, we simulate $\mathcal{A}$ for $\mathcal{R}$ in $\mathcal{T}'$, and whenever $\mathcal{A}$ contains a service X, $\mathcal{B}$ issues the service $X' \supseteq X$, created from X as follows: Start with $X' = X$. Then, for each $u \in X - \{r\}$, if w is the parent of u in $\mathcal{T}'$, then add to X' all inner nodes on the path from u to w in $\mathcal{T}$. By the construction of $\mathcal{T}'$, for each u we add at most $D - 1$ nodes, each of cost less than $L \cdot \ell_u$. It follows that $\ell(X') \leq ((D-1)L + 1)\ell(X) \leq DL \cdot \ell(X)$.

In total, the service cost of $\mathcal{B}$ is at most DL times the service cost of $\mathcal{A}$. Any request served by $\mathcal{A}$ is served by $\mathcal{B}$ at the same time or earlier, thus the waiting cost of $\mathcal{B}$ is at most the waiting cost of $\mathcal{A}$ (resp. for MLAP-D, $\mathcal{B}$ produces a valid schedule for $\mathcal{J}$). Since $\mathcal{A}$ is R-competitive, we obtain

$$\mathsf{cost}(\mathcal{B}, \mathcal{J}) \leq DL \cdot \mathsf{cost}(\mathcal{A}, \mathcal{J}') \leq DLR \cdot \mathsf{opt}(\mathcal{J}') \leq DLR \cdot \mathsf{opt}(\mathcal{J}),$$

and thus $\mathcal{B}$ is DLR-competitive. ◀

3 A Competitive Algorithm for MLAP-D

In this section we present our online algorithm for MLAP-D with competitive ratio at most $D^2 2^D$. To this end, we will give an online algorithm that achieves competitive ratio $R_L = (2 + 1/L)^{D-1}$ for L-decreasing trees. Taking $L = D/2$ and using the reduction to L-decreasing trees from Theorem 2.1 then leads to a $D^2 2^D$-competitive algorithm for arbitrary trees.

3.1 Intuitions

Consider the optimal 2-competitive algorithm for MLAP-D for trees of depth 2 [6]. Assume that the tree is L-decreasing, for some large L. (Thus $\ell_q \gg \ell_v$, for each leaf v.) Whenever a

pending request reaches its deadline, this algorithm serves a subtree X consisting of r, q and the set of leaves with the earliest deadlines and total weight of about ℓ_q. This is a natural strategy: We have to pay at least ℓ_q to serve the expiring request, so including an additional set of leaves of total weight ℓ_q can at most double our overall cost. But, assuming that no new requests arrive, serving this X can significantly reduce the cost in the future, since servicing these leaves individually is expensive: it would cost $\ell_v + \ell_q$ per each leaf v, compared to the incremental cost of ℓ_v to include v in X.

For L-decreasing trees with three levels (that is, for $D = 3$), we may try to iterate this idea. When constructing a service tree X, we start by adding to X the set of most urgent children of q whose total weight is roughly ℓ_q. Now, when choosing nodes of depth 3, we have two possibilities: (1) for each $v \in X - \{r, q\}$ we can add to X its most urgent children of combined weight ℓ_v (note that their total weight will add up to roughly ℓ_q, because of the L-decreasing property), or (2) from the set of *all* children of the nodes in $X - \{r, q\}$, add to X the set of total weight roughly ℓ_q consisting of (globally) most urgent children.

It is not hard to show that the option (1) does not lead to a constant-competitive algorithm: The counter-example involves an instance with one node w of depth 2 having many children with requests with early deadlines and all other leaves having requests with very distant deadlines. Assume that $\ell_q = L^2$, $\ell_w = L$, and that each leaf has weight 1. The example forces the algorithm to serve the children of w in small batches of size L with cost more than L^2 per batch or L per each child of w, while the optimum can serve all the requests in the children of w at once with cost $O(1)$ per request, giving a lower bound $\Omega(L)$ on the competitive ratio. (The requests at other vertices can be ignored in the optimal solution, as we can keep repeating the above strategy, similar to the lower-bound technique for 1P-MLAP that will be described in the full version of the paper. Reissuing requests at the vertices other than w will not increase the cost of the optimum.) A more intricate example shows that option (2) by itself is not sufficient to guarantee constant competitiveness either.

The idea behind our algorithm, for trees of depth $D = 3$, is to do *both* (1) and (2) to obtain X. This increases the cost of each service by a constant factor, but it protects the algorithm against both bad instances. The extension of our algorithm to depths $D > 3$ carefully iterates the process of constructing the service tree X, to ensure that for each node $v \in X$ and for each level i below v we add to X sufficiently many urgent descendants of v at that level.

3.2 Notations

To give a formal description, we need some more notations. For any set of nodes $Z \subseteq \mathcal{T}$, let Z^i denote the set of nodes in Z of depth i in tree $\mathcal{T}$ (recall that r has depth 0, q has depth 1, and leaves have depth at most D). Let also $Z^{<i} = \bigcup_{j=0}^{i-1} Z^j$ and $Z^{\leq i} = Z^{<i} \cup Z^i$. These notations can be combined with the notation Z_x, so, e.g., $Z_x^{\leq i}$ is the set of all descendants of x in Z whose depth in $\mathcal{T}$ is smaller than i.

Let $f : \mathcal{T} \to \mathbb{R} \cup \{+\infty\}$ be some function that measures urgency of nodes, so that the nodes with smaller values of f are more urgent. (For MLAP-D, urgency is measured by the deadlines, in which case the function f is set to d^t, while for MLAP we need to use a different measure.) For any set A of nodes in $\mathcal{T}$ and a real number β, let $\mathsf{Urgent}(A, \beta, f)$ be a set of nodes obtained by choosing the nodes from A in order of urgency, until either their total weight exceeds β or we run out of nodes. More precisely, we define $\mathsf{Urgent}(A, \beta, f)$ as a smallest set of nodes in A such that (i) for all $u \in \mathsf{Urgent}(A, \beta, f)$, and $v \in A - \mathsf{Urgent}(A, \beta, f)$ we have $f(u) \leq f(v)$, and (ii) either $\ell(\mathsf{Urgent}(A, \beta, f)) \geq \beta$ or $\mathsf{Urgent}(A, \beta, f) = A$.

We assume that all the deadlines in the given instance are distinct. This may be done

without loss of generality, as in case of ties we can modify the deadlines by infinitesimally small perturbations and obtain an algorithm for the general case.

At any given time t during the computation of the algorithm, for each node v, let $d^t(v)$ denote the earliest deadline among all requests in $\mathcal{T}_v$ (i.e., among all descendants of v) that are pending for the algorithm; if there is no pending request in $\mathcal{T}_v$, we set $d^t(v) = +\infty$. We will use the function d^t as the urgency of vertices at time t, i.e., a node u will be considered more urgent than a node v if $d^t(u) < d^t(v)$.

3.3 Algorithm OnlTreeD

At any time t when some request expires, that is when $t = d^t(r)$, the algorithm serves a subtree X constructed by first initializing $X = \{r, q\}$, and then incrementally augmenting X according to the following pseudo-code:

for each depth $i = 2, \ldots, D$
  $Z^i \leftarrow$ set of all children of nodes in X^{i-1}
  for each $v \in X^{<i}$
    $U(v, i, t) \leftarrow \mathsf{Urgent}(Z^i_v, \ell_v, d^t)$
    $X \leftarrow X \cup U(v, i, t)$

In other words, at depth i, we restrict our attention to Z^i, the children of all the nodes in X^{i-1}, i.e., of the nodes that we have previously selected to X at level $i-1$. (We start with $i = 2$ and $X^1 = \{q\}$.) Then we iterate over all $v \in X^{<i}$ and we add to X the set $U(v, i, t)$ of vertices from $\mathcal{T}^i_v$ (descendants of v at depth i) whose parents are in X, one by one, in the order of increasing deadlines, stopping when either their total weight exceeds ℓ_v or when we run out of such vertices. Note that these sets do not need to be disjoint.

The constructed set X is a service tree, as we are adding to it only nodes that are children of the nodes already in X.

Let ρ be the request triggering the service at time t, i.e., satisfying $d_\rho = t$. (By the assumption about different deadlines, ρ is unique.) Naturally, all the nodes u on the path from r to σ_ρ have $d^t(u) = t$ and qualify as the most urgent, thus the node σ_ρ is included in X. Therefore every request is served before its deadline.

Intuitively, it should be clear that the described algorithm cannot have a better competitive ratio than $\ell(X)/\ell_q$: If all requests are in q, the optimum will serve only q, while our algorithm uses a set X with many nodes that turn out to be useless. As we will show, via an iterative charging argument, the ratio $\ell(X)/\ell_q$ is actually achieved by the algorithm.

Recall that $R_L = (2 + 1/L)^{D-1}$. We now prove a bound on the cost of the service tree.

▶ **Lemma 3.1.** *Let X be the service tree produced by Algorithm* OnlTreeD *at time t. Then $\ell(X) \le R_L \cdot \ell_q$.*

Proof. We prove by induction that $\ell(X^{\le i}) \le (2 + 1/L)^{i-1}\ell_q$ for all $i \le D$.

The base case of $i = 1$ is trivial, as $X^{\le 1} = \{r, q\}$ and $\ell_r = 0$. For $i \ge 2$, X^i is a union of the sets $U(v, i, t)$ over all nodes $v \in X^{<i}$. Since $\mathcal{T}$ is L-decreasing, each node in the set $U(v, i, t)$ has weight at most ℓ_v/L. Thus the total weight of $U(v, i, t)$ is at most $\ell(U(v, i, t)) \le \ell_v + \ell_v/L = (1 + 1/L)\ell_v$. Therefore, by the inductive assumption, we get that

$$\begin{aligned}\ell(X^{\le i}) &\le (1 + (1 + 1/L)) \cdot \ell(X^{<i}) \\ &\le (2 + 1/L) \cdot (2 + 1/L)^{i-2}\ell_q = (2 + 1/L)^{i-1}\ell_q\,,\end{aligned}$$

proving the induction step and completing the proof that $\ell(X) \le R_L \cdot \ell_q$. ◀

3.4 Analysis

The competitive analysis uses a charging scheme. Fix some optimal schedule S^*. Consider a service (X, t) of Algorithm OnlTreeD. We will identify in X a subset of "critically overdue" nodes (to be defined shortly) of total weight at least $\ell_q \geq \ell(X)/R_L$, and we will show that for each such critically overdue node v we can charge the portion ℓ_v of the service cost of X to an earlier service in S^* that contains v. Further, any node in service of S^* will be charged at most once. This implies that the total cost of our algorithm is at most R_L times the optimal cost, giving us an upper bound of R_L on the competitive ratio for L-decreasing trees.

In the proof, by opt_v^t we denote the time of the first service in S^* that includes v and is strictly after time t; we also let $\mathsf{opt}_v^t = +\infty$ if no such service exists. For a service (X, t) of the algorithm, we say that a node $v \in X$ is *overdue* at time t if $d^t(v) < \mathsf{opt}_v^t$. Servicing of such v is delayed in comparison to S^*, because S^* must have served v before (or at) time t. Note also that r and q are overdue at time t, as $d^t(r) = d^t(q) = t$ by the choice of the service time. We define $v \in X$ to be *critically overdue* at time t if (i) v is overdue at t, and (ii) there is no other service of the algorithm in the time interval (t, opt_v^t) in which v is overdue.

We are now ready to define the charging for a service (X, t). For each $v \in X$ that is critically overdue, we charge its weight ℓ_v to the last service of v in S^* before or at time t. This charging is well-defined as, for each overdue v, there must exist a previous service of v in S^*. The charging is obviously one-to-one because between any two services in S^* that involve v there may be at most one service of the algorithm in which v is critically overdue. The following lemma shows that the total charge from X is large enough.

▶ **Lemma 3.2.** *Let (X, t) be a service of Algorithm* OnlTreeD *and suppose that $v \in X$ is overdue at time t. Then the total weight of critically overdue nodes in X_v at time t is at least ℓ_v.*

Proof. The proof is by induction on the depth of $\mathcal{T}_v$, the subtree rooted at v.

The base case is when $\mathcal{T}_v$ has depth 0, that is when v is a leaf. We show that in this case v must be critically overdue, which implies the conclusion of the lemma. Towards contradiction, suppose that there is some other service at time $t' \in (t, \mathsf{opt}_v^t)$ in which v is overdue. Since v is a leaf, after the service at time t there are no pending requests in $\mathcal{T}_v = \{v\}$. This would imply that there is a request ρ with $\sigma_\rho = v$ such that $t < a_\rho \leq d_\rho < \mathsf{opt}_v^t$. But this is not possible, because S^* does not serve v in the time interval (t, opt_v^t). Thus v is critically overdue and the base case holds.

Assume now that v is not a leaf, and that the lemma holds for all descendants of v. If v is critically overdue, the conclusion of the lemma holds.

Thus we can now assume that v is not critically overdue. This means that there is a service (Y, t') of Algorithm OnlTreeD with $t < t' < \mathsf{opt}_v^t$ which contains v and such that v is overdue at t'. Thus $\mathsf{opt}_v^t = \mathsf{opt}_v^{t'}$.

Let ρ be the request with $d_\rho = d^{t'}(v)$, i.e., the most urgent request in $\mathcal{T}_v$ at time t'.

We claim that $a_\rho \leq t$, i.e., ρ arrived no later than at time t. Indeed, since v is overdue at time t', it follows that $d_\rho < \mathsf{opt}_v^{t'} = \mathsf{opt}_v^t$. The optimal schedule S^* cannot serve ρ after time t, as S^* has no service from v in the interval $(t, d_\rho]$. Thus S^* must have served ρ before or at t, and hence $a_\rho \leq t$, as claimed.

Now consider the path from σ_ρ to v in Y. (See Figure 1.) As ρ is pending for the algorithm at time t and ρ is not served by (X, t), it follows that $\sigma_\rho \notin X$. Let w be the last node on this path in $Y - X$. Then w is well-defined and $w \neq v$, as $v \in X$. Let i be the depth of w. Note that the parent of w is in $X_v^{<i}$, so $w \in Z^i$ in the algorithm when X is constructed.

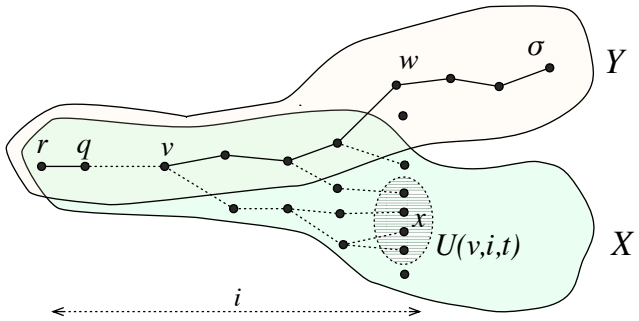

Figure 1 Illustration of the proof of Lemma 3.2.

The node σ_ρ is in $\mathcal{T}_w$ and ρ is pending at t, thus we have $d^t(w) \le d_\rho$. Since $w \in Z^i$ but w was not added to X at time t, we have that $\ell(U(v,i,t)) \ge \ell_v$ and each $x \in U(v,i,t)$ is at least as urgent as w. This implies that such x satisfies

$$d^t(x) \le d^t(w) \le d_\rho < \mathsf{opt}_v^{t'} = \mathsf{opt}_v^t \le \mathsf{opt}_x^t,$$

thus x is overdue at time t. By the inductive assumption, the total weight of critically overdue nodes in each subtree X_x is at least ℓ_x. Adding these weights over all $x \in U(v,i,t)$, we obtain that the total weight of critically overdue nodes in X_v is least $\ell(U(v,i,t)) \ge \ell_v$, completing the proof. ◀

Now consider a service (X,t) of the algorithm. The quasi-root q is overdue at time t, so Lemmas 3.2 and 3.1 imply that the charge from (X,t) is at least $\ell_q \ge \ell(X)/R_L$. Since each node in any service in S^* is charged at most once, we conclude that Algorithm OnlTreeD is R_L-competitive for any L-decreasing $\mathcal{T}$.

From the previous paragraph, using Theorem 2.1, we now obtain that there exists a $DLR_L = DL(2+1/L)^{D-1}$-competitive algorithm for general trees. For $D \ge 2$, choosing $L = D/2$ yields a competitive ratio bounded by $\frac{1}{2}D^2 2^{D-1} \cdot (1+1/D)^D \le \frac{1}{4}D^2 2^D \cdot \mathrm{e} \le D^2 2^D$. (For $D = 1$ there is a trivial 1-competitive algorithm for MLAP-D.) Summarizing, we obtain the following result.

▶ **Theorem 3.3.** *There exists a $D^2 2^D$-competitive online algorithm for MLAP-D.*

4 A Competitive Algorithm for MLAP

In this section we give an online algorithm for MLAP that achieves a constant competitive ratio for trees of bounded depth. Specifically, for trees of depth D the competitive ratio of this algorithm is at most $O(D^4 2^D)$. As before, Theorem 2.1 allows us to assume that the tree in the instance is L-decreasing.

4.1 Preliminaries and notations

We use some of the notation introduced in Section 3 and introduce some more.

Recall that $\omega_\rho(t)$ is the waiting cost function of a request ρ which is assumed to be continuous. We will overload this notation, so that we can talk about the waiting cost

function for a set of requests or a set of vertices. Specifically, for a set P of requests and a set Z of nodes, let

$$\omega_P(Z,t) = \sum_{\rho \in P : \sigma_\rho \in Z} \omega_\rho(t).$$

Thus $\omega_P(Z,t)$ is the total waiting cost of the requests from P that are issued in Z.

Maturity time

In the algorithm for MLAP-D, the times of services and the urgency of nodes are both naturally determined by the deadlines. For MLAP with continuous waiting costs there are no hard deadlines; however, we can still introduce a useful notion of *maturity time* of a node, which is, roughly speaking, the time when some subtree rooted at this node has its waiting cost equal to its service cost; this subtree is then called *mature*. This turns out to be natural both for the quasi-root as the time to service it and for the other vertices as a measure of their urgency in the algorithm. We proceed to define these notions.

Consider some time t and any set $P \subseteq \mathcal{R}$ of requests. A subtree Z of $\mathcal{T}$ (not necessarily rooted at r) is called *P-mature* at time t if $\omega_P(Z,t) \geq \ell(Z)$. Also, let $\mu_P(Z)$ denote the minimal time τ such that $\omega_P(Z,\tau) = \ell(Z)$; we let $\mu_P(Z) = \infty$ if such τ does not exist. In other words, $\mu_P(Z)$ is the earliest τ at which Z is P-mature. Since $\omega_P(Z,0) = 0$ and $\omega_P(Z,t)$ is non-decreasing and continuous function of t, $\mu_P(Z)$ is well-defined.

For a node v, let the *P-maturity time of v*, denoted $M_P(v)$, be the minimum of values $\mu_P(Z)$ over all subtrees Z of $\mathcal{T}$ rooted at v. The tree Z that achieves this minimum will be denoted $C_P(v)$ and called the *P-critical tree rooted at v*; if there are more such trees, choose one arbitrarily.

The following simple lemma guarantees that the maturity time of any node in the P-critical tree $C_P(v)$ is at most the maturity time of v.

▶ **Lemma 4.1.** *Let $u \in C_P(v)$ and let $Y = (C_P(v))_u$ be the subtree of the $C_P(v)$ rooted at u. Then $M_P(u) \leq \mu_P(Y) \leq M_P(v)$.*

Proof. The first inequality is trivial. To show the second inequality, we proceed by contradiction. If the second inequality does not hold, then $u \neq v$ and $\omega_P(Y, M_P(v)) < \ell(Y)$. Take $Y' = C_P(v) \setminus Y$, which is a tree rooted at v. Since $\omega_P(C_P(v), M_P(v)) = \ell(C_P(v))$, we have that $\omega_P(Y', M_P(v)) = \omega_P(C_P(v), M_P(v)) - \omega_P(Y, M_P(v)) > \ell(C_P(v)) - \ell(Y) = \ell(Y')$. This in turn implies that $\mu_P(Y') < M_P(v)$, which is a contradiction with the definition of $M_P(v)$. ◀

Most of the references to maturity of a node or to its critical set will be made with respect to the set of requests pending for our algorithm at a given time. (In particular, if the algorithm schedules a service at some time t, the maturity times are computed with respect to the requests that are pending at time t *before* the service is executed.) Thus, for any time t, we will use notation $M^t(v)$ and $C^t(v)$ to denote the time $M_P(v)$ and the P-critical tree $C_P(v)$, where P is the set of requests pending for the algorithm at time t. Note that in general it is possible that $M^t(v) < t$. However, our algorithm will maintain the invariant that for the quasi-root q we will have $M^t(q) \geq t$ at each time t.

4.2 Algorithm

We now describe our algorithm. A service will occur at any maturity time of the quasi-root q, that is at a time t for which $t = M^t(q)$. At such a time, the algorithm chooses a service

that contains the critical tree C of q and an extra set E, whose service cost is not much more expensive than that of C. The extra set is constructed similarly as in OnlTreeD, where the urgency of nodes is now measured by their maturity time. As before, this extra set will be a union of a system of sets $U(v,i,t)$ for $i = 2,\ldots,D$, and $v \in C^{<i} \cup E^{<i}$, except that now the sets $U(v,i,t)$ will be mutually disjoint and also disjoint from C.

Algorithm OnlTree

At any time t such that $t = M^t(q)$, serve the set $X = C \cup E$ constructed according to the following pseudo-code:

$C \leftarrow C^t(q) \cup \{r\}$
$E \leftarrow \emptyset$
for each depth $i = 2,\ldots,D$
  $Z^i \leftarrow$ set of all nodes in $\mathcal{T}^i \setminus C$ whose parent is in $C \cup E$
  for each $v \in (C \cup E)^{<i}$
    $U(v,i,t) \leftarrow \mathsf{Urgent}(Z^i_v, \ell_v, M^t)$
    $E \leftarrow E \cup U(v,i,t)$
    $Z^i \leftarrow Z^i \setminus U(v,i,t)$

At the end of the instance (when $t = H$, the time horizon) issue the last service that contains all vertices v with a pending request in $\mathcal{T}_v$.

The analysis of the algorithm will be given in the full version of the paper. We obtain:

▶ **Theorem 4.2.** *There exists an $O(D^4 2^D)$-competitive algorithm for MLAP on trees of depth D.*

5 Single-Phase MLAP

We now consider a restricted variant of MLAP that we refer to as *Single-Phase* MLAP, or 1P-MLAP. In 1P-MLAP all requests arrive at the beginning, at time 0. The instance also includes a parameter θ representing the common *expiration time* for all requests. We do not require that all requests are served. Any unserved request pays only the cost of waiting until the expiration time θ.

In the online variant of 1P-MLAP, all requests, including their waiting cost functions, are known to the online algorithm at time 0. The only unknown is the expiration time θ.

Although not explicitly named, variants of 1P-MLAP have been considered in [10, 7], where they were used to show lower bounds on competitive ratios for MLAP. These proofs consist of two steps, first showing a lower bound for online 1P-MLAP and then arguing that, in the online scenario, 1P-MLAP can be expressed as a special case of MLAP. (A corresponding property holds in the offline case as well, but is quite trivial.)

Since most lower bounds on competitive ratio of online algorithms for variants of MLAP use Single-Phase instances, a natural approach is to try such a construction in an attempt to construct super-constant lower bounds for MLAP. We show that this approach cannot be successful in the context of MLAP.

▶ **Theorem 5.1.** *There exists a 4-competitive algorithm for the Single-Phase MLAP.*

A detailed description of the algorithm will be given in the full version of the paper.

6 MLAP on Paths

We now consider the case when the tree is just a path. For simplicity we will assume a generalization to the continuous case, that we refer to as *the MLAP problem on the line*, when the path is represented by the half-line $[0, \infty)$; that is the requests can occur at any point $x \in [0, \infty)$. Then the point 0 corresponds to the root, each vertex is a point $x \in [0, \infty)$, and each service is an interval of the form $[0, x]$. We say that an algorithm *delivers from* x if it serves the interval $[0, x]$.

We provide several results for the MLAP problem on the line. We first prove that the competitive ratio of MLAP-D (the variant with deadlines) on the line is exactly 4, by providing matching upper and lower bounds. Then later we will show that the lower bound of 4 can be modified to work for MLAP-L (that is, for linear waiting costs).

Algorithm OnlLine

The algorithm creates a service only when a deadline of a pending request is reached. If a deadline of a request at x is reached, then OnlLine delivers from $2x$.

▶ **Theorem 6.1.** *Algorithm* OnlLine *is* 4*-competitive for MLAP-D on the line.*

Proof. The proof uses a charging strategy. We represent each adversary service, say when the adversary delivers from a point y, by an interval $[0, y]$. The cost of each service of OnlLine is then charged to a segment of one of those adversary service intervals.

Consider a service triggered by a deadline t of a request ρ at some point x; that is, OnlLine delivered from $2x$. The adversary must have served ρ between its arrival time and its deadline t. Fix the last such service of the adversary, where at a time $t' \leq t$ the adversary delivered from a point $x' \geq x$. We charge the cost $2x$ of the algorithm's service to the segment $[x/2, x]$ of the adversary's service interval $[0, x']$ at time t'.

We now claim that no part of the adversary's service is charged twice. To justify this claim, suppose that there are two services of OnlLine, at times $t_1 < t_2$, triggered by requests from points x_1 and x_2, respectively, that both charge to an adversary's service from x' at time $t' \leq t_1$. By the definition of charging, the request at x_2 was already present at time t'. As x_2 was not served by OnlLine's service at t_1, it means that $x_2 > 2x_1$, and thus the charged segments $[x_1/2, x_1]$ and $[x_2/2, x_2]$ of the adversary service interval at time t' are disjoint.

Summarizing, for any adversary service interval $[0, y]$, its charged segments are disjoint. Any charged segment receives the charge equal to 4 times its length. Thus this interval receives the total charge at most $4y$. This implies that the competitive ratio is at most 4. ◀

Lower bounds

We now show lower bounds of 4 for MLAP-D and MLAP-L on the line. In both proofs we show the bound for the corresponding variant of 1P-MLAP, using a reduction from the online bidding problem [13, 12]. Roughly speaking, in online bidding, for a given universe $\mathcal{U}$ of real numbers, the adversary chooses a secret value $u \in \mathcal{U}$ and the goal of the algorithm is to find an upper-bound on u. To this end, the algorithm outputs an increasing sequence of numbers $x_1, x_2, x_3, \ldots$. The game is stopped after the first x_k that is at least u and the bidding ratio is then defined as $\sum_{i=1}^{k} x_i/u$.

Chrobak et al. [12] proved that the optimal bidding ratio is exactly 4, even if it is restricted to sets $\mathcal{U}$ of the form $\{1, 2, \ldots, B\}$, for some integer B. More precisely, they proved the following result.

▶ **Lemma 6.2.** *For any $R < 4$, there exists B, such that any sequence of integers $0 = x_0 < x_1 < x_2 < \ldots < x_{m-1} < x_m = B$ has an index $k \geq 1$ with $\sum_{i=0}^{k} x_i > R \cdot (x_{k-1} + 1)$.*

▶ **Theorem 6.3.** *There is no online algorithm for MLAP-D on the line with competitive ratio smaller than 4.*

Proof. We show that no online algorithm for 1P-MLAP-D (the deadline variant of 1P-MLAP) on the line can attain competitive ratio smaller than 4. Assume the contrary, i.e., that there exists a deterministic algorithm ALG that is R-competitive, where $R < 4$. Let B be the integer whose existence is guaranteed by Lemma 6.2. We create an instance of 1P-MLAP-D, where for every $x \in \{1, \ldots, B\}$ there is a request at x with deadline x.

Without loss of generality, ALG issues services only at integer times $1, 2, ..., B$. The strategy of ALG can be now defined as a sequence of services at times $t_1 < t_2 < \ldots < t_m$, where at time t_i it delivers from $x_i \in \{t_i, t_i + 1, ..., B\}$. Without loss of generality, $x_1 < x_2 < \ldots < x_m$. We may assume that $x_m = B$ (otherwise the algorithm is not competitive at all); we also add a dummy service from $x_0 = 0$ at time $t_0 = 0$.

The adversary now chooses $k \geq 1$ and stops the game at the expiration time that is right after the algorithm's kth service, say $\theta = t_k + \frac{1}{2}$. ALG's cost is then $\sum_{i=0}^{k} x_i$. The request at $x_{k-1} + 1$ is not served at time t_{k-1}, so, to meet the deadline of this request, the schedule of ALG must satisfy $t_k \leq x_{k-1} + 1$. This implies that $\theta < x_{k-1} + 2$, that is, all requests at points $x_{k-1} + 2, x_{k-1} + 3, ..., B$ expire before their deadlines and do not need to be served. Therefore, to serve this instance, the optimal solution may simply deliver from $x_{k-1} + 1$ at time 0. Hence, the competitive ratio of ALG is at least $\sum_{i=0}^{k} x_i / (x_{k-1} + 1)$. By Lemma 6.2, it is possible to choose k such that this ratio is strictly greater than R, a contradiction with R-competitiveness of ALG. ◀

Next, we show that the same lower bound applies to MLAP-L, the version of MLAP where the waiting cost function is linear. This improves the lower bound of 3.618 from [7].

▶ **Theorem 6.4.** *There is no online algorithm for MLAP-L on the line with competitive ratio smaller than 4.*

Proof. Similarly to the proof of Theorem 6.3, we create an instance of 1P-MLAP-L (the variant of 1P-MLAP with linear waiting cost functions) that does not allow a better than 4-competitive online algorithm. Fix any online algorithm ALG for 1P-MLAP-L and, towards a contradiction, suppose that it is R-competitive, for some $R < 4$. Again, let B be the integer whose existence is guaranteed by Lemma 6.2. In our instance of 1P-MLAP-L, there are 6^{B-x} requests at x for any $x \in \{1, 2, \ldots, B\}$.

Without loss of generality, we make the same assumptions as in the proof of Theorem 6.3: algorithm ALG is defined by a sequence of services at times $0 = t_0 < t_1 < t_2 < \ldots < t_m$, where at each time t_i it delivers from some point x_i. Without loss of generality, we can assume that $0 = x_0 < x_1 < \ldots < x_m = B$.

Again, the strategy of the adversary is to stop the game at some expiration time θ that is right after some time t_k, say $\theta = t_k + \epsilon$, for some small $\epsilon > 0$. The algorithm pays $\sum_{i=0}^{k} x_i$ for serving the requests. The requests at $x_{k-1} + 1$ waited for time t_k in ALG's schedule and hence ALG's waiting cost is at least $6^{B - x_{k-1} - 1} \cdot t_k$.

The adversary delivers from point $x_{k-1} + 1$ at time 0. The remaining, unserved requests at points $x_{k-1} + 2, x_{k-1} + 3, \ldots, B$ pay time θ each for waiting. There are $\sum_{j=x_{k-1}+2}^{B} 6^{B-j} \leq \frac{1}{5} \cdot 6^{B - x_{k-1} - 1}$ such requests and hence the adversary's waiting cost is at most $\frac{1}{5} \cdot 6^{B - x_{k-1} - 1} \cdot (t_k + \epsilon)$.

Therefore, the algorithm-to-adversary ratio on the waiting costs is at least $5t_k/(t_k + \epsilon)$. For any k we can choose a sufficiently small ϵ so that this ratio is larger than 4. By Lemma 6.2, it is possible to choose k for which the ratio on servicing cost is strictly greater than R. This yields a contradiction to the R-competitiveness of ALG. ◀

We point out that the analysis in the proof above gives some insight into the behavior of any 4-competitive algorithm for 1P-MLAP-L (we know such an algorithm exists, by the results in Section 5), namely that, for the type of instances used in the above proof, its waiting cost must be negligible compared to the service cost.

References

1 Alok Aggarwal and James K. Park. Improved algorithms for economic lot sizing problems. *Operations Research*, 41:549–571, 1993.

2 Esther Arkin, Dev Joneja, and Robin Roundy. Computational complexity of uncapacitated multi-echelon production planning problems. *Operations Research Letters*, 8(2):61–66, 1989.

3 B. R. Badrinath and Pradeep Sudame. Gathercast: the design and implementation of a programmable aggregation mechanism for the internet. In *Proc. 9thInternational Conference on Computer Communications and Networks (ICCCN)*, pages 206–213, 2000.

4 Luca Becchetti, Alberto Marchetti-Spaccamela, Andrea Vitaletti, Peter Korteweg, Martin Skutella, and Leen Stougie. Latency-constrained aggregation in sensor networks. *ACM Transactions on Algorithms*, 6(1):13:1–13:20, 2009.

5 Marcin Bienkowski, Jaroslaw Byrka, Marek Chrobak, Neil B. Dobbs, Tomasz Nowicki, Maxim Sviridenko, Grzegorz Swirszcz, and Neal E. Young. Approximation algorithms for the joint replenishment problem with deadlines. *Journal of Scheduling*, 18(6):545–560, 2015.

6 Marcin Bienkowski, Jaroslaw Byrka, Marek Chrobak, Łukasz Jeż, Dorian Nogneng, and Jiří Sgall. Better approximation bounds for the joint replenishment problem. In *Proc. 25th ACM-SIAM Symp. on Discrete Algorithms (SODA)*, pages 42–54, 2014.

7 Marcin Bienkowski, Jaroslaw Byrka, Marek Chrobak, Łukasz Jeż, Jiří Sgall, and Grzegorz Stachowiak. Online control message aggregation in chain networks. In *Proc. 13th Int. Workshop on Algorithms and Data Structures (WADS)*, pages 133–145, 2013.

8 Edward Bortnikov and Reuven Cohen. Schemes for scheduling of control messages by hierarchical protocols. In *Proc. 17th IEEE Int. Conference on Computer Communications (INFOCOM)*, pages 865–872, 1998.

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

10 Niv Buchbinder, Tracy Kimbrel, Retsef Levi, Konstantin Makarychev, and Maxim Sviridenko. Online make-to-order joint replenishment model: Primal-dual competitive algorithms. In *Proc. 19th ACM-SIAM Symp. on Discrete Algorithms (SODA)*, pages 952–961, 2008.

11 Niv Buchbinder and Joseph (Seffi) Naor. The design of competitive online algorithms via a primal-dual approach. *Foundations and Trends in Theoretical Computer Science*, 3(2–3):93–263, 2009.

12 Marek Chrobak, Claire Kenyon, John Noga, and Neal E. Young. Incremental medians via online bidding. *Algorithmica*, 50(4):455–478, 2008.

13 Marek Chrobak and Claire Kenyon-Mathieu. SIGACT news online algorithms column 10: competitiveness via doubling. *SIGACT News*, 37(4):115–126, 2006.

14 Wallace B. Crowston and Michael H. Wagner. Dynamic lot size models for multi-stage assembly systems. *Management Science*, 20(1):14–21, 1973.

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

16 Fei Hu, Xiaojun Cao, and C. May. Optimized scheduling for data aggregation in wireless sensor networks. In *Int. Conference on Information Technology: Coding and Computing (ITCC)*, volume 2, pages 557–561, 2005.

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

18 Sanjeev Khanna, Joseph Naor, and Danny Raz. Control message aggregation in group communication protocols. In *Proc. 29th Int. Colloq. on Automata, Languages and Programming (ICALP)*, pages 135–146, 2002.

19 Alf Kimms. *Multi-Level Lot Sizing and Scheduling: Methods for Capacitated, Dynamic, and Deterministic Models*. Springer-Verlag, 1997.

20 Douglas M Lambert and Martha C Cooper. Issues in supply chain management. *Industrial Marketing Management*, 29(1):65–83, 2000.

21 Retsef Levi, Robin Roundy, and David B. Shmoys. A constant approximation algorithm for the one-warehouse multi-retailer problem. In *Proc. 16th ACM-SIAM Symp. on Discrete Algorithms (SODA)*, pages 365–374, 2005.

22 Retsef Levi, Robin Roundy, and David B. Shmoys. Primal-dual algorithms for deterministic inventory problems. *Mathematics of Operations Research*, 31(2):267–284, 2006.

23 Retsef Levi, Robin Roundy, David B. Shmoys, and Maxim Sviridenko. A constant approximation algorithm for the one-warehouse multiretailer problem. *Management Science*, 54(4):763–776, 2008.

24 Retsef Levi and Maxim Sviridenko. Improved approximation algorithm for the one-warehouse multi-retailer problem. In *Proc. 9th Int. Workshop on Approximation Algorithms for Combinatorial Optimization (APPROX)*, pages 188–199, 2006.

25 Tim Nonner and Alexander Souza. Approximating the joint replenishment problem with deadlines. *Discrete Mathematics, Algorithms and Applications*, 1(2):153–174, 2009.

26 Christos Papadimitriou. Computational aspects of organization theory. In *Proc. 4th European Symp. on Algorithms (ESA)*, pages 559–564, 1996.

27 Lehilton L. C. Pedrosa. Private communication. 2013.

28 Steven S. Seiden. A guessing game and randomized online algorithms. In *Proc. 32nd ACM Symp. on Theory of Computing (STOC)*, pages 592–601, 2000.

29 Shailesh Vaya. Brief announcement: Delay or deliver dilemma in organization networks. In *Proc. 31st ACM Symp. on Principles of Distributed Computing (PODC)*, pages 339–340, 2012.

30 H.M. Wagner and T. Whitin. Dynamic version of the economic lot size model. *Management Science*, 5:89–96, 1958.

31 Wei Yuan, Srikanth V. Krishnamurthy, and Satish K. Tripathi. Synchronization of multiple levels of data fusion in wireless sensor networks. In *Proc. Global Telecommunications Conference (GLOBECOM)*, pages 221–225, 2003.