

Manuscript version: Author's Accepted Manuscript

The version presented in WRAP is the author's accepted manuscript and may differ from the published version or Version of Record.

Persistent WRAP URL:

<http://wrap.warwick.ac.uk/115622>

How to cite:

Please refer to published version for the most recent bibliographic citation information. If a published version is known of, the repository item page linked to above, will contain details on accessing it.

Copyright and reuse:

The Warwick Research Archive Portal (WRAP) makes this work by researchers of the University of Warwick available open access under the following conditions.

Copyright © and all moral rights to the version of the paper presented here belong to the individual author(s) and/or other copyright owners. To the extent reasonable and practicable the material made available in WRAP has been checked for eligibility before being made available.

Copies of full items can be used for personal research or study, educational, or not-for-profit purposes without prior permission or charge. Provided that the authors, title and full bibliographic details are credited, a hyperlink and/or URL is given for the original metadata page and the content is not changed in any way.

Publisher's statement:

Please refer to the repository item page, publisher's statement section, for further information.

For more information, please contact the WRAP Team at: wrap@warwick.ac.uk.

**Submitted to *Operations Research*
manuscript (Please, provide the manuscript number!)**

Authors are encouraged to submit new papers to INFORMS journals by means of a style file template, which includes the journal title. However, use of a template does not certify that the paper has been accepted for publication in the named journal. INFORMS journal templates are for the exclusive purpose of submitting to an INFORMS journal and should not be used to distribute the papers in print or online or to submit the papers to another publication.

Online Algorithms for Multi-Level Aggregation

Marcin Bienkowski

Institute of Computer Science, University of Wrocław, Poland

Martin Böhm

CSLog, Universität Bremen, Germany and Computer Science Institute, Charles University, Czech Republic

Jarosław Byrka

Institute of Computer Science, University of Wrocław, Poland

Marek Chrobak

Department of Computer Science, University of California at Riverside, USA

Christoph Dürr

Sorbonne Université, CNRS, Laboratoire d'informatique de Paris 6, LIP6, France

Lukáš Folwarczný

Institute of Mathematics, Czech Academy of Sciences, Czech Republic and Computer Science Institute, Charles University, Czech Republic

Lukasz Jeż

Institute of Computer Science, University of Wrocław, Poland

Jiří Sgall

Computer Science Institute, Charles University, Czech Republic, sgall@iuuk.mff.cuni.cz

Nguyen Kim Thang

IBISC, Université d'Evry Val d'Essonne, France

Pavel Veselý

Department of Computer Science, University of Warwick, Coventry, UK and Computer Science Institute, Charles University, Czech Republic

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 the root of \mathcal{T} . 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 problems 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 trees of arbitrary (fixed) depth. 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.

Key words: algorithmic aspects of networks, online algorithms, scheduling and resource allocation, lot sizing, multi-stage assembly problem

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 (Dooly et al. 2001). Interestingly, TCP-AP is equivalent to the classical Lot Sizing Problem studied in the operations research literature since the 1950s. (See, for example, Wagner and Whitin (1958).)

An example in Figure 1 illustrates an instance of TCP-AP and a schedule of packet transmissions. The x-axis represents time. Messages, represented by circles, arrive at integer times. Up-arrows

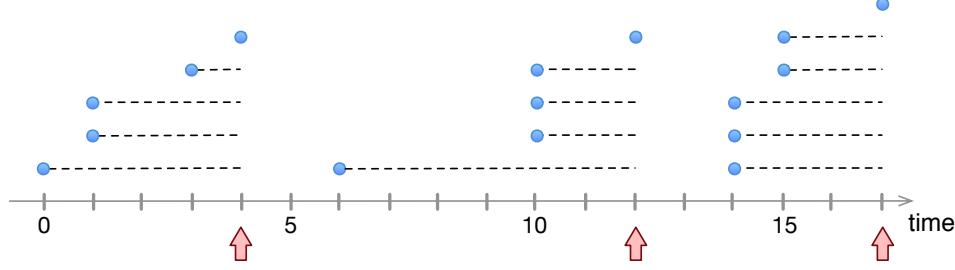


Figure 1 An example of TCP-AP with linear cost function.

point to transmission times and horizontal dashed lines represent waiting times of messages. Assume that the cost of each packet's transmission is 10 and that we charge for delay at rate 1 per time unit. The schedule consists of three transmissions whose total cost is 30. The first transmitted packet will contain 5 messages with total delay $4 + 3 + 3 + 1 + 0 = 11$. The total delay of messages in the second and third transmissions will be, respectively, 12 and 13. So the total cost of the schedule in Figure 1 is 66.

The offline variant of TCP-AP, that is computing the optimum schedule of transmissions of messages aggregated into packets, assuming that all arrival times of control messages are known beforehand, can be naturally represented by an integer linear program. The optimum solution can also be quite easily and efficiently found with dynamic programming, with the fastest known algorithm for this problem achieving running time $O(n \log n)$ (Aggarwal and Park 1993).

In practice, however, packet aggregation decisions must be done on the fly, in real time. This gives rise to the online version of TCP-AP, in which an online algorithm receives information about messages as they are released over time. At each time step, this algorithm needs to decide whether to transmit the packet with pending messages or not, without any information about future message releases. Online algorithms for a variety of other scheduling problems (and other optimization problems) have been a topic of extensive study since 1980's – see, for example, Sgall (1998), Borodin and El-Yaniv (1998). With incomplete information about the input sequence, an online algorithm cannot, in general, guarantee to compute an optimal solution. Thus research in this area focuses on designing near-optimal algorithms. The quality of solutions computed by an online algorithm is typically measured by its *competitive ratio*, which is defined (roughly) as the worst-case ratio between its cost and the optimum cost (computed offline). Naturally, the smaller the ratio the better.

The online variant of TCP-AP has been well studied: It is known that the optimal competitive ratio is 2 in the deterministic case (Dooly et al. 2001), i.e., there is an algorithm that computes a solution of cost not more than twice the optimum, and it is not possible to achieve a smaller

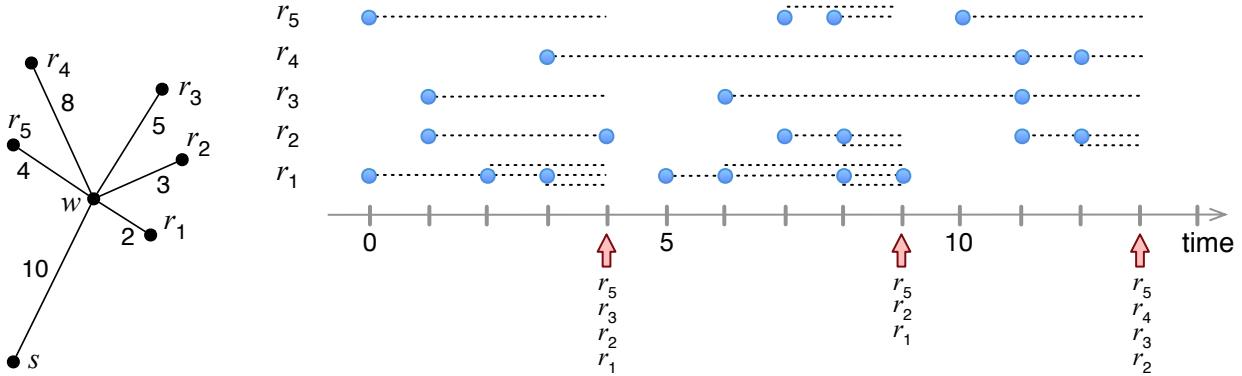


Figure 2 An example of JRP with linear cost function.

ratio online. With randomization, it is possible to reduce the ratio to $e/(e - 1) \approx 1.582$ (Karlin et al. 2003, Buchbinder and Naor 2009, Seiden 2000). Online variants of TCP-AP that use different assumptions or objective functions were also examined in the literature (Frederiksen et al. 2003, Albers and Bals 2005).

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.

Figure 2 shows an example of an instance of JRP and a schedule. This instance has five retailers r_1, r_2, \dots, r_5 , the warehouse is denoted w and the supplier is denoted s . The connections are represented by a “star” tree, shown on the left, with shipping costs associated with its edges. For example, the shipping cost from the supplier to the warehouse is 10. On the right, requests are represented by circles and are arranged in five rows corresponding to the retailers. There are three shipments, at times 4, 9 and 13, marked by up-arrows, with their participating retailers listed below. The first shipment serves retailers r_1, r_2, r_3 and r_5 and its cost is $10 + 2 + 3 + 5 + 4 = 24$. Assume that the waiting cost is equal to the time elapsed between the request and the shipment that satisfies it. Then the waiting cost of the requests served by the first shipment will be $(4 + 2 + 1) + (3 + 0) + 3 + 4 = 17$.

Similarly to TCP-AP, JRP can also be represented by an integer linear program (see, for example (Buchbinder et al. 2008)). In contrast to TCP-AP, however, JRP is known to be NP-hard (Arkin et al. 1989), and even APX-hard (Nonner and Souza 2009, Bienkowski et al. 2015). The currently best approximation algorithm, due to Bienkowski et al. (2014), achieves a factor of 1.791, improving on earlier work (Levi et al. 2005, 2008, Levi and Sviridenko 2006). 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 by Bienkowski et al. (2015), JRP-D can be approximated with ratio 1.574.

For the online variant of JRP, Buchbinder et al. (2008) gave a 3-competitive algorithm using a primal-dual scheme (improving an earlier bound of 5 by Brito et al. (2012)) and proved a lower bound of 2.64, that was subsequently improved to 2.754 (Bienkowski et al. 2014). The optimal competitive ratio for JRP-D is 2 (Bienkowski et al. 2014).

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, for example, Bortnikov and Cohen (1998), Badrinath and Sudame (2000)) can be used to reduce the so-called *ack-implosion*, the proliferation of control messages routed to the source. Such global approach is likely to be more effective than applying aggregation on each link separately (which amounts to solving an instance of the TCP-AP problem for each link). For example, the root of the tree can represent a web server that gathers TCP acknowledgements from its open TCP connections. These TCP acknowledgement messages are very small (40 bytes), yet each individual message needs to be processed by each node it traverses in order to determine its route through a routing table lookup. With aggregation, only one such processing is needed, per node, for an aggregated message that contains multiple acknowledgements. As shown experimentally by Badrinath and Sudame (2000), this approach reduces packet latency. A similar problem arises in energy-efficient data aggregation and fusion in sensor networks (Hu et al. 2005, Yuan et al. 2003). Outside of networking, tradeoffs between the cost of communication and delay arise in message aggregation in organizational hierarchies (Papadimitriou 1996). In supply-chain management, multi-level variants of lot sizing have been studied as well (Crowston and Wagner 1973, Kimms 1997). The need to consider more tree-like (in a broad sense) supply hierarchies has also been advocated by Lambert and Cooper (2000).

These applications have inspired research on offline and online approximation algorithms for multi-level aggregation problems. Becchetti et al. (2009) gave a 2-approximation algorithm for the deadline case. (See also (Brito et al. 2012).) Pedrosa (2013) showed, adapting an algorithm of Levi et al. (2006) 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. (2002) 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 $O(\log W)$ -competitive, where W is 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. (2012), 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 in the work of Vaya (2012), 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 (Pedrosa 2013, Levi et al. 2006) for the offline problem suggests the possibility of constructing an online algorithm along the lines of the scheme by Buchbinder and Naor (2009). 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 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 by Bienkowski et al. (2013). For the online variant, Brito et al. (2012) gave an 8-competitive algorithm. This result was improved by Bienkowski et al. (2013) who showed that the competitive ratio of this problem is between $2 + \phi \approx 3.618$ and 5.

A related problem of integrated scheduling and distribution has also been studied in the online setting (Azar et al. 2016): It resembles JRP, but in our terms, actually corresponds to a 1-level tree with multiple leaves. While services of those are independent, there is an interplay at the root of the tree, as each request has to be processed for its own specified amount of time at the root before being serviced — like in lot-sizing or JRP, these can be thought of as re-stocking orders, but fulfilled directly by a manufacturer once the items are produced. Azar et al. (2016) consider linear waiting costs (and preemptive scheduling/production), but, in principle, one could study arbitrary waiting functions and allow complex aggregation in shipment, as we do in this work.

1.1. Our Contributions

We study online competitive algorithms for multi-level aggregation. Minor technical differences notwithstanding, our model is equivalent to those studied by Brito et al. (2012), Khanna et al.

	MLAP and MLAP-L		MLAP-D	
	upper	lower	upper	lower
depth 1	2*	[Doo01]	2	[Doo01]
rand. alg. for depth 1	1.582*	[Kar03]	1.582	[Sei00]
depth 2	3	[Buc08]	2.754	[Bie14]
fixed depth $D \geq 2$	$O(D^4 2^D)$		2.754	$D^2 2^D$
paths of arbitrary depth	5*	[Bie13]	3.618	[Bie13]
			5*	2

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. Except the second line in the table, all bounds are for deterministic algorithms. The references to particular papers were shortened in the following way: [Sei00] (Seiden 2000), [Doo01] (Dooly et al. 2001), [Kar03] (Karlin et al. 2003), [Buc08] (Buchbinder et al. 2008), [Bie13] (Bienkowski et al. 2013), and [Bie14] (Bienkowski et al. 2014). 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. Some values in the table are approximations: 1.582 represents $e/(e - 1)$ and 3.618 represents $2 + \phi$, where ϕ is the golden ratio.

(2002), also extending the deadline variant (Becchetti et al. 2009) and the assembly problem (Levi et al. 2006). We have decided to choose a more generic terminology to emphasize general applicability of our model and techniques.

Formally, in our model, an instance of the problem consists of a tree \mathcal{T} with positive weights assigned to edges, and a set \mathcal{R} of requests 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 5. A simpler $D^2 2^D$ -competitive algorithm for MLAP-D is presented in Section 4.

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, described in Section 3, of the general problem to the special case of trees with rapidly 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 rapidly decreasing weights, the construction of the service trees, and our charging scheme, will be useful in further studies of online aggregation problems.

Finally, in Section 6, we discuss several technical issues concerning the use of general functions as waiting costs in **MLAP**. In particular, when presenting our algorithms for **MLAP** we assume that all waiting cost functions are continuous (which cannot directly capture some interesting variants of **MLAP**). This is done, however, only for technical convenience; as explained in Section 6, these algorithms can be extended to left-continuous functions, which allows us to model **MLAP-D** as a special case of **MLAP**. We also consider two alternative models for **MLAP**: the discrete-time model and the model where not all requests need to be served, showing that our algorithms can be extended to these models as well.

Notes. An extended abstract of this work appeared in the proceedings of 24th Annual European Symposium on Algorithms (ESA) (Bienkowski et al. 2016). (Other results announced in (Bienkowski et al. 2016) will be published in a separate companion paper.) In a subsequent work, Azar et al. (2017) study a more general service problem with delays. This problem includes **MLAP** as a special case when in addition to the requests from **MLAP**, we repeat many requests to the root of the tree. The results of Azar et al. (2017) then imply $O(D^{O(1)})$ competitive algorithm for **MLAP**. Finally, Buchbinder et al. (2017) improve the competitive ratio for **MLAP-D** (the variant with deadlines) to $O(D)$. Their approach uses a more subtle charging argument, combined with a reduction to the case with rapidly decreasing weights (similar to ours), showing that some of the ideas introduced in this paper could indeed be helpful for ultimately determining the tight competitive ratio for **MLAP**.

2. Preliminaries

Weighted trees. Let \mathcal{T} be a tree with root r . 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} .

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 *induced subtree* of \mathcal{T} rooted at x . Furthermore, Z^i denotes the set of nodes in Z of depth i in tree \mathcal{T} . 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., $Z_x^{<i}$ is the set of all descendants of x that belong to Z and whose depth in \mathcal{T} is smaller than i .

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. (In a typical application this weight would represent the length or the cost of traversing this edge.) For the sake of convenience, we will often refer to ℓ_x as the weight of x . 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 non-negative *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 $a_\rho \leq t \leq d_\rho$ and $\omega_\rho(t) = +\infty$ for $t > d_\rho$, where d_ρ is called the *deadline* of request ρ . 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.

In our algorithm 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 Section 6; 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 $wcost(\rho, S) = \omega_\rho(t)$. 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 $cost(S)$, is defined by

$$cost(S) = scost(S) + wcost(S),$$

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

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

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

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 $c \geq 1$, we say that \mathcal{A} is c -competitive if $\text{cost}(\mathbf{S}) \leq c \cdot \text{opt}(\mathcal{J})$ for any instance \mathcal{J} of MLAP, where \mathbf{S} is the schedule computed by \mathcal{A} on \mathcal{J} and $\text{opt}(\mathcal{J})$ is the optimum cost for \mathcal{J} . (Note that the definition of competitiveness in the literature often allows an additive error term, independent of the request sequence. For our algorithms, this additive term is not needed.)

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 of r the *quasi-root* of \mathcal{T} and denote it by q . Note that q is included in every (non-trivial) service. Requests at r can be serviced immediately at cost 0, so we can simply assume that there are no such requests in \mathcal{R} .

Urgency functions. When choosing nodes for inclusion in a service, our online algorithms give priority to those that are most “urgent”. For MLAP-D, naturally, urgency of nodes can be measured by their deadlines, where a deadline of a node v is the earliest deadline of a request pending in the subtree \mathcal{T}_v , i.e., the induced subtree rooted at v . But for the arbitrary instances of MLAP we need a more general definition of urgency, which takes into account the rate of increase of the waiting cost in the future. To this end, each of our algorithms will use some *urgency function* $f : \mathcal{T} \rightarrow \mathbb{R} \cup \{+\infty\}$, which also depends on the set of pending requests and the current time step, and which assigns some time value to each node. The earlier this value, the more urgent the node is.

Formally, for MLAP-D, we define the function $d^t(v)$ for any time t as follows. For any node v , $d^t(v)$ is the earliest deadline among all requests in \mathcal{T}_v that are pending for the algorithm at time t ; if there is no pending request in \mathcal{T}_v , we set $d^t(v) = +\infty$. We use d^t as the urgency function at time t in our algorithm for MLAP-D.

DEFINITION 1. Let f be an urgency function, A be a set of nodes in \mathcal{T} and $\beta \geq 0$ be a real number. Then, $\text{Urgent}(A, \beta, f)$ is the smallest set of nodes in A such that

1. for all $u \in \text{Urgent}(A, \beta, f)$, and $v \in A - \text{Urgent}(A, \beta, f)$ we have $f(u) \leq f(v)$, and
2. either $\ell(\text{Urgent}(A, \beta, f)) \geq \beta$ or $\text{Urgent}(A, \beta, f) = A$.

Intuitively, $\text{Urgent}(A, \beta, f)$ is the set of nodes obtained by choosing the nodes from A in order of their increasing urgency value, until either their total weight exceeds β or we run out of nodes from A . In case of ties in the values of f , there may be multiple choices for $\text{Urgent}(A, \beta, f)$; we choose among them arbitrarily.

3. Reduction to α -Decreasing Trees

One basic intuition that emerges from earlier works on trees of depth 2 (Buchbinder et al. 2008, Brito et al. 2012, Bienkowski et al. 2014) 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 α -decreasing trees that captures this intuition and showing that MLAP reduces to the special case of MLAP for such α -decreasing trees, increasing the competitive ratio by a factor of at most $D\alpha$. The value of α used in our algorithms will be fixed later. This is a general result, not limited only to algorithms in our paper.

DEFINITION 2. Fix $\alpha \geq 1$. A tree \mathcal{T} is α -decreasing if for any node u different from the root of \mathcal{T} and for any child v of u , it holds that $\ell_u \geq \alpha \cdot \ell_v$.

Note that the α -decreasing property is one of the conditions of α -HST (a hierarchically well-separated tree with separation α , see, e.g., (Bartal 1996)). That is, any α -HST is also an α -decreasing tree. However, for our purposes we do not require that the edge weight from any node to its children is the same, which is required by α -HST.

THEOREM 1. *If there exists a c -competitive algorithm \mathcal{A} for MLAP (resp. MLAP-D) on α -decreasing trees (where c can be a function of D , the tree depth), then there exists a Dac -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} . In our reduction, we convert \mathcal{T} to an α -decreasing tree \mathcal{T}' on the same set of nodes. We then show that any service on \mathcal{T} is also a service on \mathcal{T}' of the same cost and, conversely, that any service on \mathcal{T}' can be converted to a slightly more expensive service on \mathcal{T} .

We start by constructing an α -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

$\ell_w \geq \alpha \cdot \ell_u$; if no such w exists, we take $w = r$. The length of an edge from u to its parent remains ℓ_u . 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 α -decreasing.

The construction implies that if a set of nodes X is a service subtree of \mathcal{T} , then it is also a service subtree for \mathcal{T}' of the same cost. (However, note that the actual topology of the trees with node set X in \mathcal{T} and \mathcal{T}' may be very different. For example, if $\alpha = 5$ and \mathcal{T} is a path with costs (starting from the leaf) $1, 2, 2^2, \dots, 2^D$, then in \mathcal{T}' the node of weight 2^i is connected to the node of weight 2^{i+3} , except for the last three nodes that are connected to r . Thus the resulting tree consists of three paths ending at r with roughly the same number of nodes. In particular, X in \mathcal{T}' may now contain paths without any request.) Therefore, any schedule for \mathcal{J} is also a schedule for $\mathcal{J}' = (\mathcal{T}', \mathcal{R})$, which gives us that $\text{opt}(\mathcal{J}') \leq \text{opt}(\mathcal{J})$.

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 weight less than $\alpha \cdot \ell_u$. It follows that $\ell(X') \leq ((D - 1)\alpha + 1)\ell(X) \leq D\alpha \cdot \ell(X)$.

In total, the service cost of \mathcal{B} is at most $D\alpha$ 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 c -competitive, we obtain

$$\text{cost}(\mathcal{B}, \mathcal{J}) \leq D\alpha \cdot \text{cost}(\mathcal{A}, \mathcal{J}') \leq D\alpha c \cdot \text{opt}(\mathcal{J}') \leq D\alpha c \cdot \text{opt}(\mathcal{J}),$$

and thus \mathcal{B} is $D\alpha c$ -competitive. ■

4. 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 ONLTREED that achieves competitive ratio $c_\alpha = (2 + 1/\alpha)^{D-1}$ for α -decreasing trees. Together with the reduction given in the previous section, this will imply the following result.

THEOREM 2. *There exists a $D^2 2^D$ -competitive online algorithm for MLAP-D on trees of depth D .*

Proof. Applying Theorem 1 to Algorithm ONLTREED, we obtain that there exists a $D\alpha c_\alpha = D\alpha(2 + 1/\alpha)^{D-1}$ -competitive algorithm for general trees. For $D \geq 2$, choosing $\alpha = D/2$ yields a competitive ratio bounded by $\frac{1}{2}D^2 2^{D-1} \cdot (1 + 1/D)^D \leq \frac{1}{4}D^2 2^D \cdot e \leq D^2 2^D$. For $D = 1$ there is a trivial 1-competitive algorithm. ■

4.1. Intuition

Consider the optimal 2-competitive algorithm for MLAP-D for trees of depth 2 (Bienkowski et al. 2014). Assume that the tree is α -decreasing, for some large α . (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. At the same time, 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 α -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 α -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 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 $\ell_q = \alpha^2$, $\ell_w = \alpha$, and that each leaf has weight 1. The example forces the algorithm to serve the children of w in small batches of size α with cost more than α^2 per batch or α 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 $\Omega(\alpha)$ on the competitive ratio. (The requests at other nodes can be ignored in the optimal solution, as we can keep repeating the above strategy in a manner similar to the lower-bound technique presented, e.g., by Buchbinder et al. (2008) or by Bienkowski et al. (2013).) 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.

4.2. Algorithm ONLTREED

At any time t when some request expires, that is when $t = d^t(q)$ for the quasi-root q , 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  $i = 2, \dots, D$ 
   $Z^i \leftarrow$  set of all children of nodes in  $X^{i-1}$ 
  for each  $v \in X^{<i}$ 
     $U(v, i, t) \leftarrow \text{Urgent}(Z_v^i, \ell_v, d^t)$ 
     $X \leftarrow X \cup U(v, i, t)$ 

```

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 $U(v, i, t)$. $U(v, i, t)$ itself is created by taking nodes from \mathcal{T}_v^i (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 nodes. Since \mathcal{T} is α -decreasing, each added node has weight at most ℓ_v/α , and thus the total weight of $U(v, i, t)$ is at most $\ell_v + \ell_v/\alpha$. Note that added 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.

4.3. Analysis

Intuitively, it should be clear that Algorithm ONLTREED cannot have a better cost-to-optimum 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 $c_\alpha = (2 + 1/\alpha)^{D-1}$. We now prove a bound on the cost of the service tree.

LEMMA 1. *Let X be the service tree produced by Algorithm ONLTREED at time t . Then $\ell(X) \leq c_\alpha \cdot \ell_q$.*

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

The base case of $i = 1$ is trivial, as $X^{\leq 1} = \{r, q\}$ and $\ell_r = 0$. For $i \geq 2$, X^i is the union of the sets $U(v, i, t)$ over all nodes $v \in X^{< i}$. Recall that by our construction, $\ell(U(v, i, t)) \leq \ell_v + \ell_v/\alpha = (1 + 1/\alpha)\ell_v$. Therefore, by the inductive assumption, we get that

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

proving the induction step and completing the proof that $\ell(X) \leq c_\alpha \cdot \ell_q$. \blacksquare

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 $\ell_q \geq \ell(X)/c_\alpha$, 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, each occurrence of v in the services of S^* will be charged at most once in this way. This implies that the total cost of our algorithm is at most c_α times the optimal cost, giving us an upper bound of c_α on the competitive ratio for α -decreasing trees.

In the proof, by nos_v^t we denote the time of the first service in S^* that includes v and is strictly after time t ; we also let $\text{nos}_v^t = +\infty$ if no such service exists (nos stands for *next optimal service*). For a service (X, t) of the algorithm, we say that a node $v \in X$ is *overdue* at time t if $d^t(v) < \text{nos}_v^t$. 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, nos_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 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 induced 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 $t' \in (t, \text{nos}_v^t)$ in which v is overdue. Since v is a

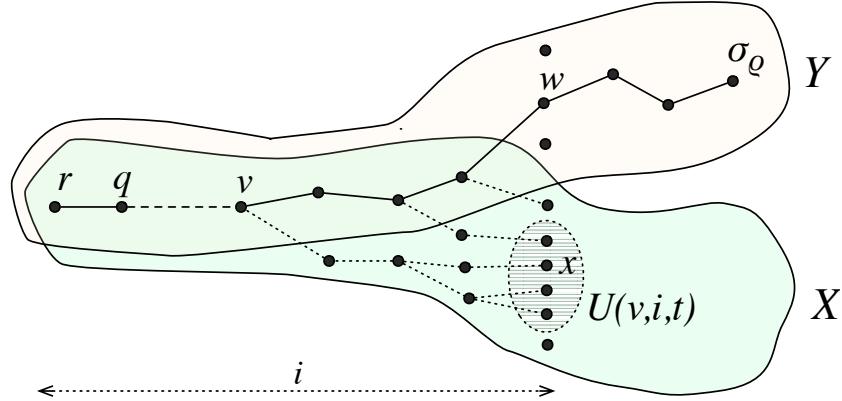


Figure 3 Illustration of the proof of Lemma 2.

leaf, after the service at time t there are no pending requests in $\mathcal{T}_v = \{v\}$. As v is overdue at time t' , we have $d^{t'}(v) < \text{nos}_v^{t'} = \text{nos}_v^t$ and this implies that there is a request ρ with $\sigma_\rho = v$ such that $t < a_\rho \leq d_\rho < \text{nos}_v^t$. But this is not possible, because S^* does not serve v in the time interval (t, nos_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 $t < t' < \text{nos}_v^t$ which contains v and such that v is overdue at t' . Thus $\text{nos}_v^t = \text{nos}_v^{t'}$.

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 $d_\rho < \text{nos}_v^{t'} = \text{nos}_v^t$. 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 3.) 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.

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

$$d^t(x) \leq d^t(w) \leq d_\rho < \text{nos}_v^{t'} = \text{nos}_v^t \leq \text{nos}_x^t,$$

and thus x is overdue at time t . By the inductive assumption, the total weight of critically overdue nodes in each induced 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 at least $\ell(U(v, i, t)) \geq \ell_v$, completing the proof. ■

Now consider a service (X, t) of the algorithm. The quasi-root q is overdue at time t , so Lemmata 2 and 1 imply that the charge from (X, t) is at least $\ell_q \geq \ell(X)/c_\alpha$. Since each node in any service in S^* is charged at most once, we conclude that Algorithm ONLTREED is c_α -competitive for any α -decreasing tree \mathcal{T} .

5. A Competitive Algorithm for MLAP

In this section, we show that there is an online algorithm for MLAP whose competitive ratio for trees of depth D is $O(D^4 2^D)$. As in Section 4, we will assume that the tree \mathcal{T} in the instance is α -decreasing and present a competitive algorithm for such trees, which will imply the existence of a competitive algorithm for arbitrary trees by using Theorem 1 and choosing an appropriate value of α .

5.1. Preliminaries and Notations

Recall that $\omega_\rho(t)$ denotes the waiting cost function of a request ρ . As explained in Section 2, we assume that the waiting cost functions are continuous. (In Section 6, we discuss how to extend our results to arbitrary waiting cost functions.) We will overload this notation, so that we can talk about the waiting cost of a set of requests or a set of nodes. 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 . We sometimes omit P , in which case the notation refers to the set of all requests in the instance, that is $\omega(Z, t) = \omega_{\mathcal{R}}(Z, t)$. Similarly, we omit Z when Z contains all nodes, that is $\omega_P(t) = \omega_P(\mathcal{T}, t)$.

Maturity time. In our algorithm for MLAP-D in Section 4, 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. Nevertheless, we can still introduce the 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 maturity time will be our urgency function, as discussed earlier in Section 2. We use the maturity time in two ways: first, the maturity times of the quasi-root determine the service times, and second, maturity times of other nodes are used to prioritize them for inclusion in the service trees. We now proceed to define these notions formally.

Consider some time t and any set $P \subseteq \mathcal{R}$ of requests and a subtree Z of \mathcal{T} (not necessarily rooted at r). Z is called P -mature at time t if $\omega_P(Z, t) \geq \ell(Z)$. Let

$$\mu_P(Z) = \arg \min_t \{\omega_P(Z, t) \geq \ell(Z)\}.$$

That is, $\mu_P(Z)$ is the earliest time τ at which Z is P -mature; if such τ does not exist, we set $\mu_P(Z) = +\infty$. Since $\omega_P(Z, 0) = 0$, $\ell(Z) \geq 0$, and $\omega_P(Z, t)$ is a non-decreasing and continuous function of t , $\mu_P(Z)$ is well-defined.

In the following definition, a condition $Z \sqsubseteq \mathcal{T}_v$ denotes that a set of nodes Z is not only a subtree of \mathcal{T}_v , but is also itself rooted at v . Consider any node v and any set $P \subseteq \mathcal{R}$ of requests. Let

$$M_P(v) = \min_{Z \sqsubseteq \mathcal{T}_v} \mu_P(Z) \quad \text{and} \tag{1}$$

$$C_P(v) = \arg \min_{Z \sqsubseteq \mathcal{T}_v} \mu_P(Z). \tag{2}$$

$M_P(v)$ is called the P -maturity time of v and $C_P(v)$ is called the P -critical subtree rooted at v ; if there are more such trees, we choose one arbitrarily. From the above definitions, we have that $\omega_P(C_P(v), M_P(v)) = \ell(C_P(v))$.

The following simple lemma guarantees that the maturity time of any node in the P -critical subtree $C_P(v)$ is upper bounded by the maturity time of v .

LEMMA 3. *Let $u \in C_P(v)$ and let $Y = (C_P(v))_u$ be the induced subtree of $C_P(v)$ rooted at u . Then $M_P(u) \leq \mu_P(Y) \leq M_P(v)$.*

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

We stress that the concepts of maturity times and critical subtrees are defined above abstractly with respect to arbitrary sets P of requests, and are independent of the online algorithm. Yet, naturally, in our algorithm and its analysis, in most cases this P will represent the set of requests pending for the algorithm at a given time. Thus, for any time t , we will also introduce simplified notations $M^t(v)$ and $C^t(v)$ to denote the time $M_P(v)$ and the P -critical subtree $C_P(v)$, where P is the set of requests pending for the algorithm at time t ; if it so happens that the algorithm schedules a service at some time t , then this P represents the set of requests that are pending at time t right before this service is executed. 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 .

5.2. Algorithm

We now describe our algorithm for α -decreasing trees. A service will occur at each maturity time of the quasi-root q (with respect to the pending requests), that is at each time t for which $t = M^t(q)$. At such a time, the algorithm chooses a service that contains the critical subtree $C = C^t(q)$ 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 Algorithm ONLTREED, where the urgency of nodes is now measured by their maturity time. In other words, our urgency function is now $f = M^t$ (see Section 2.) As before, this extra set will be a union of a system of sets $U(v, i, t)$ for $i = 2, \dots, D$, and $v \in C^{<i} \cup E^{<i}$, except that now, for technical reasons, 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  $i = 2, \dots, D$ 
     $Z^i \leftarrow$  set of all nodes in  $\mathcal{T}^i - C$  whose parent is in  $C \cup E$ 
    for each  $v \in (C \cup E)^{<i}$ 
         $U(v, i, t) \leftarrow \text{Urgent}(Z_v^i, \ell_v, M^t)$ 
         $E \leftarrow E \cup U(v, i, t)$ 
         $Z^i \leftarrow Z^i - U(v, i, t)$ 

```

At the end of the instance (when $t = H$, the time horizon), if there are any pending requests, ONLTREE issues the last service that contains all nodes v with a pending request in \mathcal{T}_v .

Note that $X = C \cup E$ is indeed a service tree, as it contains r, q and we are adding to it only nodes that are children of the nodes already in X . The initial choice and further changes of Z^i imply that the sets $U(v, i, t)$ are pairwise disjoint and disjoint from C — a fact that will be useful in our analysis.

We also need the following fact.

LEMMA 4. Suppose that Algorithm ONLTREE issues a service at a time t , that is $M^t(q) = t$. Let P' denote the set of requests pending at time t and not served at time t . Then $M_{P'}(q) > t$.

Proof. Consider any subtree Y of \mathcal{T} rooted at q . It is sufficient to show that $\omega_{P'}(Y, t) < \ell(Y)$.

We claim that the following relations hold:

$$\omega(C^t(q) \cup Y, t) \geq \omega(C^t(q), t) + \omega_{P'}(Y, t) \quad (3)$$

$$\omega(C^t(q), t) = \ell(C^t(q)) \quad (4)$$

$$\omega(C^t(q) \cup Y, t) \leq \ell(C^t(q) \cup Y) \quad (5)$$

$$\ell(C^t(q) \cup Y) < \ell(C^t(q)) + \ell(Y) \quad (6)$$

Indeed, inequality (3) is true because each request pending at time t in $C^t(q) \cup Y$ contributes to at most one of the terms on the right-hand side: if it is in $C^t(q)$ then it's served at time t , so it cannot be in P' . Inequalities (4) and (5) follow from the assumption that $M^t(q) = t$ and from the definition of $C^t(q)$. (Observe that $C^t(q) \cup Y \sqsubseteq \mathcal{T}_q$, that is $C^t(q) \cup Y$ is also a candidate for a critical set in (2).) Inequality (6) holds because $C^t(q)$ and Y both contain q with $\ell(q) > 0$.

Combining (3)-(6), we obtain

$$\begin{aligned} \ell(C^t(q)) + \omega_{P'}(Y, t) &= \omega(C^t(q), t) + \omega_{P'}(Y, t) \\ &\leq \omega(C^t(q) \cup Y, t) \\ &\leq \ell(C^t(q) \cup Y) < \ell(C^t(q)) + \ell(Y), \end{aligned}$$

and $\omega_{P'}(Y, t) < \ell(Y)$ follows. ■

COROLLARY 1. *At any time t we have $M^t(q) \geq t$.*

Proof. The statement holds trivially at the beginning, at time $t = 0$. In any time interval without new requests released nor services, the inequality $M^t(q) \geq t$ is preserved by the definition of the service times and continuity of waiting cost functions. Releasing a request ρ at a time $a_\rho = t$ cannot decrease $M^t(q)$ to below t , because the waiting cost function of ρ is identically 0 up to t , and thus releasing ρ does not change the waiting costs at time t or before. Finally, Lemma 4 implies that the inequality is also preserved when a service occurs. ■

Corollary 1 shows that the sequence of service times is non-decreasing and thus the definition of the algorithm is sound. In fact Lemma 4 even shows that no two services can occur at the same time.

5.3. Competitive Analysis

We now present the proof that there is a $O(D^4 2^D)$ -competitive algorithm for **MLAP** for trees of depth D . The overall argument is quite intricate, so we will start by summarizing its main steps:

- First, as explained earlier, we will assume that the tree \mathcal{T} in the instance is α -decreasing. For such \mathcal{T} we will show that Algorithm **ONLTREE** has competitive ratio $O(D^2 c_\alpha)$, where $c_\alpha = (2 + 1/\alpha)^{D-1}$. Our bound on the competitive ratio for arbitrary trees will then follow, by using Theorem 1 and choosing an appropriate value of α (see Theorem 3).

- For α -decreasing trees, the bound of the competitive ratio of Algorithm ONLTREE involves four ingredients:

— We show (in Lemma 5) that the total cost of Algorithm ONLTREE is at most twice its service cost.

— Next, we show that the service cost of Algorithm ONLTREE can be bounded (within a constant factor) by the total cost of all critical subtrees $C^t(q)$ of the service trees in its schedule.

— To facilitate the estimate of the adversary cost, we introduce the concept of a *pseudo-schedule* denoted \bar{S} . The pseudo-schedule \bar{S} is a collection of *pseudo-services*, which include the services from the original adversary schedule S^* . We show (in Lemma 7) that the adversary pseudo-schedule has service cost not larger than D times the cost of S^* . Using the pseudo-schedule allows us to ignore the waiting cost in the adversary's schedule.

— With the above bounds established, it remains to show that the total cost of critical subtrees in the schedule of Algorithm ONLTREE is within a constant factor of the service cost of the adversary's pseudo-schedule. This is accomplished through a charging scheme that charges nodes (or, more precisely, their weights) from each critical subtree of Algorithm ONLTREE to their appearances in some earlier adversary pseudo-services.

Two auxiliary bounds. We now assume that \mathcal{T} is α -decreasing and proceed with our proof, according to the outline above.

The definition of the maturity time implies that the waiting cost of all the requests served is at most the service cost $\ell(X)$, as otherwise X would be a good candidate for a critical subtree at some earlier time. Denoting by S the schedule computed by Algorithm ONLTREE, we thus obtain:

LEMMA 5. $\text{cost}(S) \leq 2 \cdot \text{scost}(S)$.

Using Lemma 5, we can restrict ourselves to bounding the service cost, losing at most a factor of 2. We now bound the cost of a given service X ; recall that $c_\alpha = (2 + 1/\alpha)^{D-1}$.

LEMMA 6. *Each service tree $X = C \cup E$ constructed by the algorithm satisfies $\ell(X) \leq c_\alpha \cdot \ell(C)$.*

Proof. Since \mathcal{T} is α -decreasing, the weight of each node that is a descendant of v is at most ℓ_v/α and thus $\ell(U(v, i, t)) \leq (1 + 1/\alpha)\ell_v$.

We now estimate $\ell(X)$. We claim and prove by induction for $i = 1, \dots, D$ that

$$\ell(X^{\leq i}) \leq (2 + 1/\alpha)^{i-1} \ell(C^{\leq i}). \quad (7)$$

The base case for $i = 1$ is trivial, as $X^{\leq 1} = C^{\leq 1} = \{r, q\}$. For $i \geq 2$, the set X^i consists of C^i and the sets $U(v, i, t)$, for $v \in X^{< i}$. Each of these sets $U(v, i, t)$ has weight at most $(1 + 1/\alpha)\ell_v$. Therefore

$$\ell(X^i) \leq (1 + 1/\alpha)\ell(X^{< i}) + \ell(C^i). \quad (8)$$

Now, using (8) and the inductive assumption (7) for $i - 1$, we get

$$\begin{aligned}\ell(X^{\leq i}) &= \ell(X^{<i}) + \ell(X^i) \\ &\leq (2 + 1/\alpha)\ell(X^{<i}) + \ell(C^i) \\ &\leq (2 + 1/\alpha)^{i-1}\ell(C^{<i}) + \ell(C^i) \leq (2 + 1/\alpha)^{i-1}\ell(C^{\leq i}).\end{aligned}$$

Taking $i = D$ in (7), the lemma follows. ■

Waiting costs and pseudo-schedules. Our plan is to charge the cost of Algorithm ONLTREE to the optimal (or the adversary's) cost. Let S^* be an optimal schedule. To simplify this charging, we extend S^* by adding to it pseudo-services, where a *pseudo-service from a node v* is a partial service of cost ℓ_v that consists only of the edge from v to its parent. We denote this modified schedule \bar{S} and call it a *pseudo-schedule*, reflecting the fact that its pseudo-services are not necessarily subtrees of \mathcal{T} rooted at r . Adding such pseudo-services will allow us to ignore the waiting costs in the optimal schedule.

We now define more precisely how to obtain \bar{S} from S^* . For each node v independently we define the times when new pseudo-services of v occur in \bar{S} . Intuitively, we introduce these pseudo-services at intervals such that the waiting cost of the requests that arrive in \mathcal{T}_v during these intervals adds up to ℓ_v . The formal description of this process is given in the pseudo-code below, where we use notation $\mathcal{R}(>t)$ for the set of requests $\rho \in \mathcal{R}$ with $a_\rho > t$ (i.e., requests issued after time t). Recall that H denotes the time horizon.

```

 $t \leftarrow -\infty$ 
while  $\omega_{\mathcal{R}(>t)}(\mathcal{T}_v, H) \geq \ell_v$ 
  let  $\tau$  be the earliest time such that  $\omega_{\mathcal{R}(>t)}(\mathcal{T}_v, \tau) = \ell_v$ 
  add to  $\bar{S}$  a pseudo-service of  $v$  at  $\tau$ 
   $t \leftarrow \tau$ 

```

We apply the above procedure to all the nodes $v \in \mathcal{T} - \{r\}$ such that \mathcal{R} contains a request in \mathcal{T}_v . The new pseudo-schedule \bar{S} contains all the services of S^* (treated as sets of pseudo-services of all served nodes) and the new pseudo-services added as above. The service cost of the pseudo-schedule, $\text{scost}(\bar{S})$, is defined naturally as the total weight of the nodes in all its pseudo-services and we bound it in the next lemma for $D \geq 2$ (recall that for $D = 1$ a constant-competitive algorithm is already known).

LEMMA 7. *For $D \geq 2$ it holds $\text{scost}(\bar{S}) \leq D \cdot \text{cost}(S^*)$.*

Proof. It is sufficient to show that the total service cost of the new pseudo-services added inside the while loop is at most $\text{scost}(S^*) + D \cdot \text{wcost}(S^*)$: Adding $\text{scost}(S^*)$ once more to account for the

service cost of the services of S^* that are included in \bar{S} , and using our assumption that $D \geq 2$, we obtain $\text{scost}(\bar{S}) \leq 2 \cdot \text{scost}(S^*) + D \cdot \text{wcost}(S^*) \leq D \cdot \text{cost}(S^*)$, thus the lemma follows.

To prove the claim, consider some node v , and a pair of times t, τ from one iteration of the while loop, when a new pseudo-service was added to \bar{S} at time τ . This pseudo-service has cost ℓ_v . In S^* , either there is a service in $(t, \tau]$ including v , or the total waiting cost of the requests within \mathcal{T}_v released in this interval is equal to $\omega_{\mathcal{R}(>t)}(\mathcal{T}_v, \tau) = \ell_v$. In the first case, we charge the cost of ℓ_v of this pseudo-service to any service of v in S^* in $(t, \tau]$. Since we consider here only the new pseudo-services, created by the above pseudo-code, this charging will be one-to-one. In the second case, we charge ℓ_v to the total waiting cost of the requests in \mathcal{T}_v released in the interval $(t, \tau]$. For each given v , the charges of the second type from pseudo-services at v go to disjoint sets of requests in \mathcal{T}_v , so each request in \mathcal{T}_v will receive at most one charge from v . Therefore, for each request ρ , its waiting cost in S^* will be charged at most D times, namely at most once from each node v on the path from σ_ρ to q . From the above argument, the total cost of the new pseudo-services is at most $\text{scost}(S^*) + D \cdot \text{wcost}(S^*)$, as claimed. ■

Using the bound in Lemma 7 will allow us to use $\text{scost}(\bar{S})$ as an estimate of the optimal cost in our charging scheme, losing at most a factor of D in the competitive ratio.

Charging scheme. According to Lemma 5, to establish constant competitiveness it is sufficient to bound only the service cost of Algorithm ONLTREE. By Lemma 6 for any service tree X of the algorithm we have $\ell(X) \leq c_\alpha \cdot \ell(C)$. Therefore, it is in fact sufficient to bound the total weight of the critical sets in the algorithm's services. Further, using Lemma 7, instead of using the optimal cost in this bound, we can use the pseudo-service cost. Following this idea, we will show how we can charge, at a constant rate, the cost of all critical sets C in the algorithm's services to the adversary pseudo-services.

The basic idea of our charging method is similar to that for MLAP-D. The argument in Section 4 can be interpreted as an iterative charging scheme, where we have a charge of ℓ_q that originates from q , and this charge is gradually distributed and transferred down the service tree, through overdue nodes, until it reaches critically overdue nodes that can be charged directly to adversary services. For MLAP with general waiting costs, the charge of $\ell(C)$ will originate from the current critical subtree C . Several complications arise when we attempt to distribute the charges to nodes at deeper levels. First, due to gradual accumulation of waiting costs, it does not seem possible to identify nodes in the same service tree that can be used as either intermediate or final nodes in this process. Instead, when defining a charge from a node v , we will charge descendants of v in *earlier* services of v . Specifically, the weight ℓ_v will be charged to the set $U(v, i, t^-)$ for some $i > \text{depth}(v)$, where t^- is the time of the previous service of the algorithm that includes v . The nodes — or,

more precisely, services of these nodes — that can be used as intermediate nodes for transferring charges will be called *depth-timely*. As before, we will argue that each charge will eventually reach a node u in some earlier service that can be charged to some adversary pseudo-service directly. Such service of u will be called *u -local*, where the name reflects the property that this service has an adversary pseudo-service of u nearby (to which its weight ℓ_u will be charged).

We now formalize these notions. Let (X, t) be some service of Algorithm ONLTREE that includes v , that is $v \in X$. By $\text{Prev}^t(v)$ we denote the time of the last service of v before t in the schedule of the algorithm; if it does not exist, set $\text{Prev}^t(v) = -\infty$. By $\text{Next}^t(v, i)$ we denote the time of the i th service of v following t in the schedule of the algorithm; if it does not exist, set $\text{Next}^t(v, i) = +\infty$.

We say that the service of v at time $t < H$ is *i -timely*, if $M^t(v) < \text{Next}^t(v, i)$; furthermore, if v is $\text{depth}(v)$ -timely, we will say simply that this service of v is *depth-timely*. We say that the service of v at time $t < H$ is *v -local*, if this is either the first service of v by the algorithm, or if there is an adversary pseudo-service of v in the interval $(\text{Prev}^t(v), \text{Next}^t(v, \text{depth}(v)))$.

Given an algorithm's service (X, t) , we now define the outgoing charges from X . For any $v \in X - \{r\}$, its outgoing charge is defined as follows:

- (C1) If $t < H$ and the service of v at time t is both depth-timely and v -local, charge ℓ_v to the first adversary pseudo-service of v after time $\text{Prev}^t(v)$.
- (C2) If $t < H$ and the service of v at time t is depth-timely but not v -local, charge ℓ_v to the algorithm's service at time $\text{Prev}^t(v)$.
- (C3) If $t < H$ and the service of v at time t is not depth-timely, the outgoing charge is 0.
- (C4) If $t = H$, we charge ℓ_v to the first adversary pseudo-service of v .

We first argue that the charging is well-defined. To justify (C1) suppose that this service is depth-timely and v -local. If (X, t) is the first service of v then $\text{Prev}^t(v) = -\infty$ and the charge goes to the first pseudo-service of v which exists as all the requests must be served. Otherwise there is an adversary pseudo-service of v in the interval $(\text{Prev}^t(v), \text{Next}^t(v, \text{depth}(v)))$ and rule (C1) is well-defined. For (C2), note that if the service (X, t) of v is depth-timely but not v -local then there must be an earlier service including v . (C3) is trivial. For (C4), note again that an adversary service of v must exist, as all requests must be served.

The following lemma implies that all nodes in the critical subtree will have an outgoing charge, as needed.

LEMMA 8. *Suppose there is a service at a time $t < H$. The service of each $v \in C^t(q)$ at time t is 1-timely, and thus also depth-timely.*

Proof. From Lemma 3, each $v \in C^t(q)$ satisfies $M^t(v) \leq M^t(q) = t < \text{Next}^t(q, 1) \leq \text{Next}^t(v, 1)$, where the sharp inequality follows from Lemma 4. ■

The following lemma captures the key property of our charging scheme. For any depth-timely service of $v \in X$ that is not v -local, it identifies a subset $U(v, i, t^-)$ of the previous service (X^-, t^-) including v that is suitable for receiving a charge from v . It is important that each such set is used only once, has sufficient weight, and contains only depth-timely nodes. As we show later, these properties imply that in this charging scheme the net charge (the difference between the outgoing and incoming charge) from each service X is at least as large as the total weight of its critical subtree.

As in the argument for MLAP-D, we need to find an urgent node $w \in X_v$ which is not in X^- and has its parent in X^- . There are two important issues caused by the fact that the urgency is given by the maturity times instead of deadlines. The first issue is that the maturity time can decrease due to new request arrivals — to handle this, we argue that if the new requests had large waiting costs, they would guarantee the existence of a pseudo-service of node v in the given time interval and thus the algorithm's service of v would be v -local. The second issue is that the maturity time is not given by a single descendant but by adding the node contributions from the whole tree — thus instead of searching for w on a single path, we need a more subtle, global argument to identify such w .

LEMMA 9. *Assume that the service of v at time $t < H$ is depth-timely and not v -local. Let $i = \text{depth}(v)$, and let (X^-, t^-) be the previous service of Algorithm ONLTREE including v , that is $t^- = \text{Prev}^t(v)$. Then there exists $j > i$ such that all the nodes in the set $U(v, j, t^-)$ from the construction of X^- in the algorithm are depth-timely and $\ell(U(v, j, t^-)) \geq \ell_v$.*

Proof. Let $t^* = M^t(v)$ and let $C' = C^t(v)$ be the critical subtree of v at time t . Since the service of v at time t is i -timely, we have $t^* < \text{Next}^t(v, i)$. (It may be the case that $t^* < t$, but that does not hamper our proof in any way.) Also, since the service of v at time t is not v -local, it is not the first service of v , thus t^- and X^- are defined.

Let P^- be the set of requests pending right after time t^- (including those with arrival time t^- but not those served at time t^-), and let P be the set of requests with arrival time in the interval $(t^-, t]$. The key observation is that the total waiting cost of all the requests in C' that arrived after t^- satisfies

$$\omega_P(C', t^*) < \ell_v. \quad (9)$$

To see this, simply note that $\omega_P(C', t^*) \geq \ell_v$ would imply that $\omega_{\mathcal{R}(>t^-)}(\mathcal{T}_v, t^*) \geq \ell_v$. This in turn would imply the existence of a pseudo-service of v in the interval $(t^-, t^*] \subseteq (\text{Prev}^t(v), \text{Next}^t(v, i)]$,

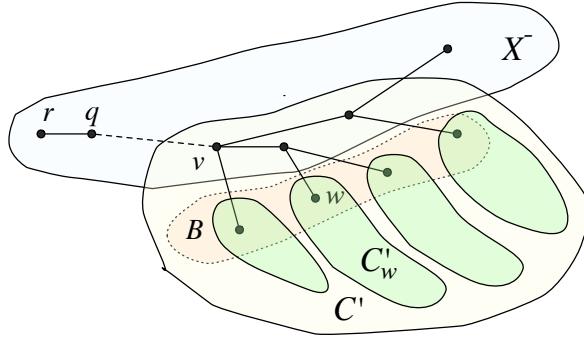


Figure 4 Illustration of the proof of Lemma 9.

which would contradict the assumption that the service of v at time t is not v -local. (Note that if $t^* \leq t^-$ then $\omega_P(C', t^*) = 0$ as t^* is before the arrival time of any request in P and the inequality holds trivially.)

Since $P^- \cup P$ contains all the requests pending at time t , the choice of t^* and C' implies that

$$\omega_{P^- \cup P}(C', t^*) = \ell(C') . \quad (10)$$

P^- does not contain any requests in $C' \cap X^-$, as those were served at time t^- ; therefore $\omega_{P^-}(C', t^*) = \omega_{P^-}(C' - X', t^*)$. Letting B be the set of all nodes $w \in C' - X'$ for which $\text{parent}(w) \in X^-$, we have $C' - X' = \bigcup_{w \in B} C'_w$, where all sets C'_w , for $w \in B$, are disjoint. (See Figure 4.) Also, $v \in C' \cap X^-$. Combining these observations, and using inequalities (9) and (10), we get

$$\begin{aligned} \sum_{w \in B} \omega_{P^-}(C'_w, t^*) &= \omega_{P^-}(\bigcup_{w \in B} C'_w, t^*) \\ &= \omega_{P^-}(C' - X', t^*) \\ &= \omega_{P^-}(C', t^*) \\ &= \omega_{P^- \cup P}(C', t^*) - \omega_P(C', t^*) \\ &> \ell(C') - \ell_v \\ &\geq \ell(C') - \ell(C' \cap X^-) \\ &= \ell(C' - X^-) = \sum_{w \in B} \ell(C'_w) . \end{aligned}$$

It follows that there exists $w \in B$ such that

$$\omega_{P^-}(C'_w, t^*) > \ell(C'_w) . \quad (11)$$

Equation (11) implies that $M^{t^-}(w) \leq t^*$, using also the fact that w was not served at t^- , so P^- contains exactly all the requests used to define $M^{t^-}(w)$. Let $j = \text{depth}(w)$; note that $j > i$ as w is

a descendant of v . Since $w \notin X^-$ but $\text{parent}(w) \in X^-$, and $M^{t^-}(w)$ is finite, the definition of the extra sets for X^- implies that $U(v, j, t^-)$ has sufficient weight and all its nodes are more urgent than w . More precisely, $\ell(U(v, j, t^-)) \geq \ell_v$ and any $z \in U(v, j, t^-)$ has $M^{t^-}(z) \leq M^{t^-}(w) \leq t^*$.

It remains to show that every $z \in U(v, j, t^-)$ is depth-timely at time t^- . Indeed, since $\text{depth}(z) = j \geq i + 1$ and any service containing z contains also v , we get

$$\text{Next}^{t^-}(z, j) \geq \text{Next}^{t^-}(z, i+1) \geq \text{Next}^{t^-}(v, i+1) = \text{Next}^t(v, i) > t^* \geq M^{t^-}(z),$$

where the last step uses the inequality $t^* \geq M^{t^-}(z)$ derived in the previous paragraph. Thus z is depth-timely, as needed. The proof of the lemma is now complete. ■

Competitive analysis. We are now ready to complete our competitive analysis of MLAP.

THEOREM 3. *There exists an $O(D^{42D})$ -competitive algorithm for MLAP on trees of depth D .*

Proof. We will show that Algorithm ONLTREE's competitive ratio for α -decreasing trees of depth $D \geq 3$ is at most $4D^2 c_\alpha$, where $c_\alpha = (2 + 1/\alpha)^{D-1}$. By applying Theorem 1, this implies that there is an online algorithm for arbitrary trees with ratio at most $4D^3 \alpha (2 + 1/\alpha)^{D-1}$. For $\alpha = D/2$, this ratio is bounded by $3D^4 2^D$, implying the theorem (together with the fact that for $D = 1, 2$, constant-competitive algorithms are known).

Next, fix an α -decreasing tree \mathcal{T} and focus our attention on Algorithm ONLTREE's schedule S and on the adversary pseudo-schedule \bar{S} . Define the *net charge from* a service (X, t) in S to be the difference between the outgoing and incoming charge of (X, t) . Our goal is to show that each pseudo-service in \bar{S} is charged only a constant number of times and that the net charge from each service (X, t) in S is at least $\ell(X)/c_\alpha$.

Consider first an adversary pseudo-service of v at a time τ . We argue that it is charged at most $(D+3)\ell_v$: If this is the first pseudo-service of v , it may be charged once from both the first service of v by rule (C1) and from the last service of v at time $t = H$ by rule (C4). In addition, by rule (C1) it may be charged D times from the last D services of v before τ , and once from the first service at or after τ . All the charges are equal to ℓ_v .

Now consider a service (X, t) of Algorithm ONLTREE. For $t = H$, all the nodes of X have an outgoing charge by rule (C4) and there is no incoming charge. Thus the net charge from X is $\ell(X) \geq \ell(X)/c_\alpha$.

For $t < H$, let $X = C \cup E$, where C is the critical subtree and E is the extra set. From Lemma 8, all nodes in C are depth-timely, so they generate outgoing charge of at least $\ell(C)$ from X .

We now consider the remaining balance of charges associated with X , namely the outgoing charges from E minus the total incoming charge to X . We claim that this quantity is non-negative.

Recall that E is a disjoint union of sets of the form $U(w, k, t)$ and E is disjoint from C . If a future service of a node v generates the charge of ℓ_v to X by rule (C2), it must be serviced at time $\text{Next}^t(v, 1)$, so such a charge is unique for each v . Furthermore, Lemma 9 implies that one of the extra sets $U(v, j, t)$, for $j > i$, has $\ell(U(v, j, t)) \geq \ell_v$ and consists of depth-timely nodes only. Thus these nodes have outgoing charges adding up to at least ℓ_v ; these charges go either to the adversary's pseudo-services or the algorithm's services before time t . We have shown that the net charge from each extra set $U(w, k, t)$ is non-negative; therefore, the net charge from E is non-negative as well. We conclude that the net charge from X is at least $\ell(C)$. Applying Lemma 6, we obtain that this net charge is at least $\ell(X)/c_\alpha$.

Summing over all the services (X, t) in S , we get a bound for the service cost of schedule S : $\text{scost}(S) \leq (D + 3)c_\alpha \cdot \text{scost}(\bar{S})$. Applying Lemmata 5 and 7, we get

$$\begin{aligned}\text{cost}(S) &\leq 2 \cdot \text{scost}(S) \\ &\leq 2(D + 3)c_\alpha \cdot \text{scost}(\bar{S}) \\ &\leq 2D(D + 3)c_\alpha \cdot \text{cost}(S^*) \leq 4D^2c_\alpha \cdot \text{cost}(S^*).\end{aligned}$$

We have thus shown that Algorithm ONLTREE's competitive ratio for α -decreasing trees is at most $4D^2c_\alpha$, which, as explained earlier, is sufficient to complete the proof. ■

6. General Waiting Costs

Our model of MLAP assumes full continuity, namely that the time is continuous and that the waiting costs are continuous functions of time, while in some earlier literature authors use the discrete model. Thus, we still need to show that our algorithms can be applied in the discrete model without increasing their competitive ratios. We also consider the model where some request may remain unserved. We explain how our results can be extended to these models as well. We will also show that our results can be extended to functions that are left-continuous, and that MLAP-D can be represented as a special case of MLAP with left-continuous functions. While those reductions seem intuitive, they do involve some pesky technical challenges, and they have not been yet formally treated in the literature.

Extension to the discrete model. In the discrete model (see, e.g., (Buchbinder et al. 2008)), requests arrive and services may happen only at integral points $t = 1, \dots, H$, where H is the time horizon. The waiting cost functions ω_ρ are also specified only at integral points. (The model in (Buchbinder et al. 2008) also allows waiting costs to be non-zero at the release time. However we can assume that $\omega_\rho(a_\rho) = 0$, since increasing the waiting cost function uniformly by an additive constant can only decrease the competitive ratio.) We now show how to reduce the discrete time model to the model where time and waiting costs are continuous.

THEOREM 4. Suppose that \mathcal{A} is a c -competitive online algorithm for the model with continuous time and continuous waiting cost functions. Then, there exists a c -competitive algorithm \mathcal{B} for the discrete time model.

Proof. Algorithm \mathcal{B} is constructed as follows. Let $\mathcal{J} = \langle \mathcal{T}, \mathcal{R} \rangle$ be an instance given to \mathcal{B} . We extend each waiting cost function ω_ρ to non-integral times as follows: for each integral $t = a_\rho, \dots, H-1$ we define $\omega_\rho(\tau)$ for $\tau \in (t, t+1)$ so that it continuously increases from $\omega_\rho(t)$ to $\omega_\rho(t+1)$ (e.g., by linear interpolation); $\omega_\rho(\tau) = 0$ for all $\tau < a_\rho$; and $\omega_\rho(\tau) = \omega_\rho(H)$ for all $\tau > H$.

Algorithm \mathcal{B} presents the instance $\mathcal{J} = \langle \mathcal{T}, \mathcal{R} \rangle$ with these continuous waiting cost functions to \mathcal{A} . At each integral time $t = 1, \dots, H-1$, \mathcal{B} simulates \mathcal{A} on the whole interval $[t, t+1]$. If \mathcal{A} makes one or more services, \mathcal{B} makes a single service at time t which is their union. This is possible, since no request arrives in $(t, t+1)$. At time H , algorithm \mathcal{B} issues the same service as \mathcal{A} .

Overall, \mathcal{B} produces a feasible schedule in the discrete time model. The cost of \mathcal{B} does not exceed the cost of \mathcal{A} . On the other hand, any feasible (offline) schedule \mathbf{S} in the discrete time model is also a feasible schedule in the continuous time model with the same cost. Thus \mathcal{B} is c -competitive. ■

Unserved requests with bounded waiting costs. In our definition of MLAP we require that all requests are eventually served. However, if the waiting cost of a request ρ is bounded, it is natural to allow a possibility that ρ is not served in a schedule \mathbf{S} ; in that case ρ 's contribution to the waiting cost of \mathbf{S} is $w\text{cost}(\rho, \mathbf{S}) = \lim_{t \rightarrow +\infty} \omega_\rho(t)$. In this variant, there is no time horizon in the instance, and the total cost of \mathbf{S} is defined as before, as the sum of its service and waiting costs.

Our algorithm ONLTREE works in this model as well, with the competitive ratio increased at most by one. The only modification of the algorithm is that there is no final service at the time horizon. Instead we let the time proceed to infinity, issuing services at the maturity times of q (the quasi-root of \mathcal{T}).

Let \mathbf{S}^* be the optimal schedule. Using the charging scheme described earlier in the paper, all costs of ONLTREE other than the waiting cost of unserved requests will be charged to \mathbf{S}^* . To extend the charging scheme to unserved requests, for each node v we consider the service times of v in \mathbf{S}^* and in ONLTREE, and we define v to be one of three types:

Type 1: v is not serviced neither by \mathbf{S}^* nor by ONLTREE,

Type 2: the last service of v is by ONLTREE (possibly tied with \mathbf{S}^*),

Type 3: the last service of v is by \mathbf{S}^* .

Nodes of Type 1 pay the same waiting cost in \mathbf{S}^* and ONLTREE's schedule, so their contributions can only decrease the competitive ratio and we can ignore them in the cost calculations. The same argument applies to nodes of Type 2, because there ONLTREE's cost of unserved requests is not larger than that of \mathbf{S}^* . It thus remains to show how we can charge the waiting cost of nodes v of

Type 3 to S^* . Let Y be the subtree of \mathcal{T} rooted at r induced by nodes of Type 3 and let Q be the set of requests in Y that are never served. ONLTREE's schedule satisfies $\lim_{\tau \rightarrow \infty} \text{wcost}_Q(\tau) \leq \ell(Y)$, for otherwise Y would become eventually mature and the requests in Q would be served by ONLTREE. On the other hand, $\ell(Y) \leq \text{scost}(S^*)$, because all nodes in Y are served in S^* at least once, by the definition of Type 3 nodes. So we can charge $\lim_{\tau \rightarrow \infty} \text{wcost}_Q(\tau)$ to S^* , increasing the competitive ratio by at most 1.

Extension to left-continuous waiting costs. We now argue that we can modify our algorithms to handle *left-continuous* waiting cost functions, i.e., functions that satisfy $\lim_{\tau \nearrow t} \omega_\rho(\tau) = \omega_\rho(t)$ for each time $t \geq 0$. Left-continuity enables an online algorithm to serve a request at the last time when its waiting cost is at or below some given threshold.

Some form of left-continuity is also necessary for constant competitiveness. To see this, think of a simple example of a tree of depth 1 and with $\ell_q = 1$, and a sequence of requests in q with release times approaching 1, and waiting cost functions defined by $\omega_\rho(1) = K \gg 1$ and $\omega_\rho(t) = 0$ for $t < 1$. If an online algorithm serves one such request before time 1, the adversary immediately releases another. The sequence stops either after K requests or after the algorithm serves some request at or after time 1, whichever comes first. The optimal cost is at most $\ell_q = 1$, while the online algorithm pays at least K .

The basic (but not quite correct) idea of our argument for left-continuous waiting cost functions is this: For any time point h where some waiting cost function has a discontinuity, we replace point h by a “gap interval” $[h, h + \epsilon]$, for some $\epsilon > 0$. The release times after time h and the values of all waiting cost functions after h are shifted to the right by ϵ . In the interval $[h, h + \epsilon]$, for each request ρ , its waiting cost function is filled in by any non-decreasing continuous curve with value ν^- at h and ν^+ at $h + \epsilon$, for $\nu^- = \omega_\rho(h)$ and $\nu^+ = \lim_{\tau \searrow h} \omega_\rho(\tau)$. Thus the waiting cost functions that are continuous at h are simply “stretched” in this gap interval, where their values remain constant. This will convert the original instance \mathcal{J} into an instance \mathcal{J}' with continuous waiting cost functions; then we can apply a reduction similar to the one for the discrete model, with the behavior of an algorithm \mathcal{A} on \mathcal{J}' inside $[h, h + \epsilon]$ mimicked by the algorithm \mathcal{B} on \mathcal{J} while staying at time h .

The above construction, however, has a flaw: as \mathcal{B} is online, for each newly arrived request ρ it would need to know the future requests in order to correctly modify ρ 's waiting cost function (which needs to be fully revealed at the arrival time). Thus, inevitably, \mathcal{B} will need to be able to modify waiting cost functions of earlier requests, but the current state of \mathcal{A} may depend on these functions. Such changes could make the computation of \mathcal{A} meaningless. To avoid this problem, we will focus only on algorithms \mathcal{A} for continuous cost functions that we call *stretch-invariant*. Roughly, those are algorithms whose computation is not affected by the stretching operation described above.

To formalize this, let $\mathbb{I} = \{[h_i, h_i + \varepsilon_i] \mid i = 1, \dots, k\}$ be a finite set of *gap intervals*, where all times h_i are distinct. (For now we can allow the ε_i 's to be any positive reals; their purpose will be explained later.) Let $shift(t, \mathbb{I}) = t + \sum_{i:h_i < t} \varepsilon_i$ denote the time t shifted right by inserting intervals \mathbb{I} on the time axis. We extend this operation to requests in a natural way: for any request ρ with a continuous waiting cost function, $shift(\rho, \mathbb{I})$ denotes the request modified by inserting \mathbb{I} on the time axis and filling in the values of ω_ρ in the inserted intervals by constant functions, as described earlier. For a set of requests $P \subseteq \mathcal{R}$, the stretched set of requests $shift(P, \mathbb{I})$ is the set consisting of requests $shift(\rho, \mathbb{I})$, for all $\rho \in P$.

Consider an online algorithm \mathcal{A} for MLAP with continuous waiting cost functions. We say that \mathcal{A} is *stretch-invariant* if for every instance $\mathcal{J} = \langle \mathcal{T}, \mathcal{R} \rangle$ and any set of gap intervals \mathbb{I} , the schedule produced by \mathcal{A} for the instance $\langle \mathcal{T}, shift(\mathcal{R}, \mathbb{I}) \rangle$ is obtained from the schedule produced by \mathcal{A} for \mathcal{J} by shifting it according to \mathbb{I} , namely every service (X, t) is replaced by service $(X, shift(t, \mathbb{I}))$.

Most natural algorithms for MLAP are stretch-invariant. In case of ONLTREE, observe that its behavior depends only on the maturity times $M_P(v)$ where P is the set of pending requests and $M_{shift(P, \mathbb{I})}(v) = shift(M_P(v), \mathbb{I})$; in particular stretching does not change the order of the maturity times. Using induction on the current time t , we observe ONLTREE creates a service (X, t) in its schedule for the request set \mathcal{R} if and only if ONLTREE creates a service $(X, shift(t, \mathbb{I}))$ in its schedule for the request set $shift(\mathcal{R}, \mathbb{I})$.

THEOREM 5. *Suppose that \mathcal{A} is a c -competitive online algorithm for continuous waiting cost functions that is stretch-invariant. Then, there exists a c -competitive algorithm \mathcal{B} for left-continuous waiting costs.*

Proof. Let $\mathcal{J} = \langle \mathcal{T}, \mathcal{R} \rangle$ be an instance given to \mathcal{B} . Algorithm \mathcal{B} maintains the set of gap intervals \mathbb{I} , and a set of requests \mathcal{P} presented to \mathcal{A} ; both sets are initially empty. Algorithm \mathcal{B} at time t simulates the computation of \mathcal{A} at time $shift(t, \mathbb{I})$.

If a new request $\rho \in \mathcal{R}$ is released at time $t = a_\rho$, algorithm \mathcal{B} obtains ρ' from $shift(\rho, \mathbb{I})$ by replacing the discontinuities of ω_ρ by new gap intervals \mathbb{I}_ρ on which $\omega_{\rho'}$ is defined so that it continuously increases. (If a gap interval already exists in \mathbb{I} at the given point, it is used instead of creating a new one, to maintain the starting points distinct.) We set $a_{\rho'} = shift(t, \mathbb{I})$, which is the current time in \mathcal{A} . We update \mathbb{I} to $\mathbb{I} \cup \mathbb{I}_\rho$; this does not change the current time in \mathcal{A} as all new gap intervals start at or after t . We stretch the set of requests \mathcal{P} by \mathbb{I}_ρ ; this does not change the past output of \mathcal{A} , because \mathcal{A} is stretch-invariant. (Note that the state of \mathcal{A} at time t may change, but this does not matter for the simulation.) Finally, we add the new request ρ' to \mathcal{P} .

If the current time t in \mathcal{B} is at a start point of a gap interval, i.e., $t = h_i$, algorithm \mathcal{B} simulates the computation of \mathcal{A} on the whole shifted gap interval $\langle shift(h_i, \mathbb{I}), \varepsilon_i \rangle$. If \mathcal{A} makes one or more services in $\langle shift(h_i, \mathbb{I}), \varepsilon_i \rangle$, \mathcal{B} makes a single service at time t which is their union.

The cost of \mathcal{B} for requests \mathcal{R} does not exceed the cost of \mathcal{A} for requests \mathcal{P} , since the service cost can only be smaller for \mathcal{B} due to the merging of \mathcal{A} 's services in gap intervals, and the waiting cost of a request in \mathcal{B} 's schedule is at most its waiting cost in \mathcal{A} 's schedule by left-continuity. Any adversary schedule \mathbf{S} for \mathcal{R} induces a schedule \mathbf{S}' for \mathcal{P} with the same cost. Since \mathcal{A} 's cost is at most $c \cdot \text{cost}(\mathbf{S}')$, we obtain that \mathcal{B} 's cost is at most $c \cdot \text{cost}(\mathbf{S})$; hence \mathcal{B} is c -competitive.

In the discussion above we assumed that the instance has a finite number of discontinuities. Arbitrary left-continuous waiting cost functions may have infinitely many discontinuity points, but the set of these points must be countable. The construction described above extends to arbitrary left-continuous cost functions, as long as we choose the ε_i values so that their sum is finite. ■

Reduction of MLAP-D to MLAP. We now argue that MLAP-D can be expressed as a variant of MLAP with left-continuous waiting cost functions. The idea is simple: a request ρ with deadline d_ρ can be assigned a waiting cost function $\omega_\rho(t)$ that is 0 for times $t \in [0, d_\rho]$ and $+\infty$ for $t > d_\rho$ – except that we cannot really use $+\infty$, so we need to replace it by some sufficiently large number. If $\sigma_\rho = v$, we let $\omega_\rho(t) = \ell_v^*$, where ℓ_v^* is the sum of all weights on the path from v to r (the “distance” from v to r). This will convert an instance \mathcal{J} of MLAP-D into an instance \mathcal{J}' of MLAP with left-continuous waiting cost functions.

We claim that, without loss of generality, any online algorithm \mathcal{A} for \mathcal{J}' serves any request ρ before or at time d_ρ . Otherwise, \mathcal{A} would have to pay waiting cost of ℓ_v^* for ρ (where $v = \sigma_\rho$), so we can modify \mathcal{A} to serve ρ at time d_ρ instead, without increasing its cost. We can then treat \mathcal{A} as an algorithm for \mathcal{J} . \mathcal{A} will meet all deadlines in \mathcal{J} and its cost on \mathcal{J} will be the same as its cost on \mathcal{J}' , which means that its competitive ratio will also remain the same.

Note that algorithm ONLTREE (or rather its extension to the left-continuous waiting costs, as described above) does not need this modification, as it already guarantees that when the waiting cost of a request at v reaches ℓ_v^* , all the nodes on the path from v to r are mature and thus the whole path is served.

Acknowledgments

Research partially supported by NSF grants CCF-1536026, CCF-1217314 and OISE-1157129, Polish National Science Centre grants 2016/21/D/ST6/02402, 2016/22/E/ST6/00499 and 2015/18/E/ST6/00456, project 17-09142S of GA ČR, GAUK project 634217, FMJH PGMO, ANR OATA, and RFSI.

References

- Aggarwal A, Park JK (1993) Improved algorithms for economic lot sizing problems. *Operations Research* 41:549–571.
- Albers S, Bals H (2005) Dynamic TCP acknowledgment: Penalizing long delays. *SIAM Journal on Discrete Mathematics* 19(4):938–951.

- Arkin E, Joneja D, Roundy R (1989) Computational complexity of uncapacitated multi-echelon production planning problems. *Operations Research Letters* 8(2):61–66.
- Azar Y, Epstein A, Jeż L, Vardi A (2016) Make-to-Order Integrated Scheduling and Distribution. *Proc. 24th ACM-SIAM Symp. on Discrete Algorithms (SODA)*, 140–154.
- Azar Y, Ganesh A, Ge R, Panigrahi D (2017) Online service with delay. *Proc. 49th ACM Symp. on Theory of Computing (STOC)*, 551–563.
- Badrinath BR, Sudame P (2000) Gathercast: the design and implementation of a programmable aggregation mechanism for the internet. *Proc. 9th International Conference on Computer Communications and Networks (ICCCN)*, 206–213.
- Bartal Y (1996) Probabilistic approximations of metric spaces and its algorithmic applications. *Proc. 37th IEEE Symp. on Foundations of Computer Science (FOCS)*, 184–193.
- Beccetti L, Marchetti-Spaccamela A, Vitaletti A, Korteweg P, Skutella M, Stougie L (2009) Latency-constrained aggregation in sensor networks. *ACM Transactions on Algorithms* 6(1):13:1–13:20.
- Bienkowski M, Böhm M, Byrka J, Chrobak M, Dürr C, Folwarczný L, Jeż L, Sgall J, Thang NK, Veselý P (2016) Online algorithms for multi-level aggregation. *Proc. 24th European Symp. on Algorithms (ESA)*, 12:1–12:17.
- Bienkowski M, Byrka J, Chrobak M, Dobbs NB, Nowicki T, Sviridenko M, Swirszcz G, Young NE (2015) Approximation algorithms for the joint replenishment problem with deadlines. *Journal of Scheduling* 18(6):545–560.
- Bienkowski M, Byrka J, Chrobak M, Jeż L, Nogneng D, Sgall J (2014) Better approximation bounds for the joint replenishment problem. *Proc. 25th ACM-SIAM Symp. on Discrete Algorithms (SODA)*, 42–54.
- Bienkowski M, Byrka J, Chrobak M, Jeż L, Sgall J, Stachowiak G (2013) Online control message aggregation in chain networks. *Proc. 13th Int. Workshop on Algorithms and Data Structures (WADS)*, 133–145.
- Borodin A, El-Yaniv R (1998) *Online Computation and Competitive Analysis* (Cambridge University Press).
- Bortnikov E, Cohen R (1998) Schemes for scheduling of control messages by hierarchical protocols. *Proc. 17th IEEE Int. Conference on Computer Communications (INFOCOM)*, 865–872.
- Brito C, Koutsoupias E, Vaya S (2012) Competitive analysis of organization networks or multicast acknowledgement: How much to wait? *Algorithmica* 64(4):584–605.
- Buchbinder N, Feldman M, Naor JS, Talmon O (2017) O(depth)-competitive algorithm for online multi-level aggregation. *Proceedings of the Twenty-Eighth Annual ACM-SIAM Symposium on Discrete Algorithms (SODA '17)*, 1235–1244.
- Buchbinder N, Kimbrel T, Levi R, Makarychev K, Sviridenko M (2008) Online make-to-order joint replenishment model: Primal-dual competitive algorithms. *Proc. 19th ACM-SIAM Symp. on Discrete Algorithms (SODA)*, 952–961.

- Buchbinder N, Naor JS (2009) The design of competitive online algorithms via a primal-dual approach. *Foundations and Trends in Theoretical Computer Science* 3(2–3):93–263.
- Crowston WB, Wagner MH (1973) Dynamic lot size models for multi-stage assembly systems. *Management Science* 20(1):14–21.
- Dooly DR, Goldman SA, Scott SD (2001) On-line analysis of the TCP acknowledgment delay problem. *Journal of the ACM* 48(2):243–273.
- Frederiksen JS, Larsen KS, Noga J, Uthaisombut P (2003) Dynamic TCP acknowledgment in the LogP model. *Journal of Algorithms* 48(2):407–428.
- Hu F, Cao X, May C (2005) Optimized scheduling for data aggregation in wireless sensor networks. *Int. Conference on Information Technology: Coding and Computing (ITCC)*, volume 2, 557–561.
- Karlin AR, Kenyon C, Randall D (2003) Dynamic TCP acknowledgement and other stories about $e/(e - 1)$. *Algorithmica* 36(3):209–224.
- Khanna S, Naor J, Raz D (2002) Control message aggregation in group communication protocols. *Proc. 29th Int. Colloq. on Automata, Languages and Programming (ICALP)*, 135–146.
- Kimms A (1997) *Multi-Level Lot Sizing and Scheduling: Methods for Capacitated, Dynamic, and Deterministic Models* (Springer-Verlag).
- Lambert DM, Cooper MC (2000) Issues in supply chain management. *Industrial Marketing Management* 29(1):65–83.
- Levi R, Roundy R, Shmoys DB (2005) A constant approximation algorithm for the one-warehouse multi-retailer problem. *Proc. 16th ACM-SIAM Symp. on Discrete Algorithms (SODA)*, 365–374.
- Levi R, Roundy R, Shmoys DB (2006) Primal-dual algorithms for deterministic inventory problems. *Mathematics of Operations Research* 31(2):267–284.
- Levi R, Roundy R, Shmoys DB, Sviridenko M (2008) A constant approximation algorithm for the one-warehouse multiretailer problem. *Management Science* 54(4):763–776.
- Levi R, Sviridenko M (2006) Improved approximation algorithm for the one-warehouse multi-retailer problem. *Proc. 9th Int. Workshop on Approximation Algorithms for Combinatorial Optimization (APPROX)*, 188–199.
- Nonner T, Souza A (2009) Approximating the joint replenishment problem with deadlines. *Discrete Mathematics, Algorithms and Applications* 1(2):153–174.
- Papadimitriou C (1996) Computational aspects of organization theory. *Proc. 4th European Symp. on Algorithms (ESA)*, 559–564.
- Pedrosa LLC (2013) Private communication.
- Seiden SS (2000) A guessing game and randomized online algorithms. *Proc. 32nd ACM Symp. on Theory of Computing (STOC)*, 592–601.

- Sgall J (1998) On-line scheduling. *Online Algorithms: The State of the Art*, 196–231 (Springer).
- Vaya S (2012) Brief announcement: Delay or deliver dilemma in organization networks. *Proc. 31st ACM Symp. on Principles of Distributed Computing (PODC)*, 339–340.
- Wagner H, Whitin T (1958) Dynamic version of the economic lot size model. *Management Science* 5:89–96.
- Yuan W, Krishnamurthy SV, Tripathi SK (2003) Synchronization of multiple levels of data fusion in wireless sensor networks. *Proc. Global Telecommunications Conference (GLOBECOM)*, 221–225.

Marcin Bienkowski obtained a PhD degree from the University of Paderborn, Germany, where he worked in the Algorithms and the Complexity Theory group. He is currently an associate professor and the head of the Combinatorial Optimization Group at the Computer Science Institute of the University of Wrocław, Poland. His research interests are focused on online and approximation algorithms, especially for network problems.

Martin Böhm is a postdoc researcher at CSLog, Universität Bremen, Germany. He obtained his PhD at Charles University, Prague in 2018. His research focuses approximation and online algorithms.

Jarosław Byrka is an associate professor in CS Institute, University of Wrocław, where he is a faculty since 2010. Since 2016 he is a vice dean of Department of Mathematics and Computer Science. He obtained his PhD from TU Eindhoven in 2008. His research focuses on algorithmic techniques in combinatorial optimization.

Marek Chrobak is a professor of computer science at University of California, Riverside, where he has been faculty member since 1987. He received his PhD degree from Warsaw University in 1985. His research interests are in algorithms and theoretical computer science, with main focus on online competitive algorithms and approximation algorithms for combinatorial optimization problems.

Christoph Dürr graduated in 1997 from the University Paris-South, in the area of quantum computing. He is currently a senior researcher at the CNRS, affiliated to Sorbonne University, in the Operation Research group of the lab LIP6. His research focuses on scheduling, more generally on online algorithms and combinatorial optimization.

Lukáš Folwarczný is a PhD student at Institute of Mathematics, Czech Academy of Sciences, and Charles University, Prague. His research interests are in complexity theory and online algorithms.

Łukasz Jeż is an assistant professor in CS Institute, University of Wrocław since 2016. He received his PhD from University of Wrocław in 2011. His research focuses on online algorithms.

Jiří Sgall is a professor of computer science and a vice-dean at Faculty of Mathematics and Physics, Charles University, Prague. He received his PhD from Carnegie Mellon University in 1994.

His current research focuses on online and approximation algorithms for scheduling and related problems.

Nguyen Kim Thang obtained his PhD in 2009 from Ecole Polytechnique France. He is currently associate professor in Univ Evry, University Paris-Saclay. His main research is the design and analysis of algorithms, especially the interaction between algorithms, game theory and online optimization.

Pavel Veselý is a research fellow at University of Warwick. He obtained his PhD at Charles University, Prague in 2018. His research interests are in algorithms, namely in online competitive algorithms and streaming algorithms.