SIGACT News Online Algorithms Column 23

Rob van Stee
University of Leicester
Leicester, United Kingdom

For this first column of 2014, my predecessor as editor of this column, Marek Chrobak, graciously offered to write a contribution, discussing online aggregation problems. Readers might remember the TCP acknowledgement problem which has been studied in various papers. This survey considers generalizations of that problem, with emphasis on some (very) recent results. Thanks a lot to Marek for his contribution!

I would like to invite more contributions to this column, be it surveys, conference reports, or technical articles related to online algorithms and competitive analysis. If you are considering becoming a guest writer, don't hesitate to mail me at `rob.vanstee@le.ac.uk`.

Online Aggregation Problems

Marek Chrobak[*]

February 14, 2014

1 Introduction

This paper is a survey of research on online algorithms for aggregation problems. Generally, an aggregation problem can be thought of as a scheduling problem, where tasks can be executed in batches. The cost of executing a batch is fixed, independent of the number of tasks in the batch. There is also a "conformity" cost associated with a task being assigned to a batch. This cost may depend on the job and the batch. Typically, the conformity costs represent the waiting cost (delay), and we will often use this term in the paper. But other types of conformity costs may appear in some applications, for example the cost of pre- or post-processing if the batch processing does not fully meet the requirements of the task. Our objective is to minimize the total cost:

$$\text{cost} = (\text{total batch cost}) + (\text{total conformity cost}).$$

A simple example of such an aggregation problem is the well-known *Dynamic TCP Acknowledgement Problem*, where we wish to schedule transmissions of control (acknowledgement) messages along a network link. These messages are very small and any number of them can be aggregated and transmitted as a single packet. The tradeoff is that more aggregation reduces congestion but it also increases average delay. The objective is to minimize the sum of the number of transmitted packets and the total delay of all messages.

The TCP acknowledgement problem can be naturally extended from a single link to trees of arbitrary depth. Control packets can be generated at all nodes and they need to be transmitted to the root. We are allowed to aggregate them at intermediate nodes, to reduce cost. This extension was studied by Khanna *et al.* [12] and Brito *et al.* [6]. We extend it further, by allowing waiting cost functions that are not necessarily linear.

More formally, we consider the *Multi-Level Aggregation Problem (MLAP)*, defined as follows. We are given a tree $\mathcal{T}$ with root r, whose edges are assigned positive weights. We are also given a set of requests $\mathcal{R}$, where each request $\pi \in \mathcal{R}$ is specified by its arrival time a_π and the node v_π where π arrives. Any subtree X of $\mathcal{T}$ rooted at r is called a *service*. Such X serves all requests that are pending at its nodes at the cost equal to the total weight of X. For each request π we are also given its waiting cost function ω_π whose values are non-negative and non-decreasing with time. If π is served at time $t \geq a_\pi$ then its waiting cost is $\omega_\pi(t)$. The objective is to find a schedule (that is, a set of service trees) that minimizes the total service cost plus the total waiting cost.

[*]Department of Computer Science, University of California at Riverside, USA. Work partially supported by NSF Grants CCF-1217314 and OISE-1157129.

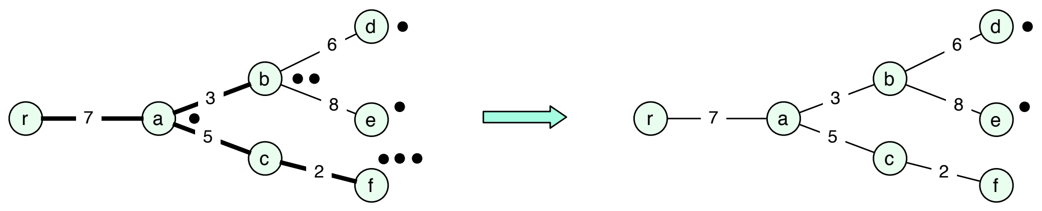


Figure 1: An example of a service subtree in MLAP. The dots next to vertices represent requests. The service subtree is marked by thick edges. After the service, all requests in the subtree are satisfied. The cost of this service is $7 + 3 + 5 + 2 = 17$.

The waiting cost function is often assumed to be linear, that is $\omega_\pi(t) = t - a_\pi$ for $t \geq a_\pi$, but most results in the literature hold for arbitrary waiting cost functions, as long as they are non-negative and non-decreasing. One other function that has been studied has the form $\omega_\pi(t) = 0$ for $t \leq d_\pi$ and ∞ for $t > d_\pi$, where d_π is called the *deadline* of π. Let MLAP-L and MLAP-D denote the variants of MLAP with, respectively, linear waiting cost functions and with deadlines.

The online variant of MLAP is defined in a natural way: The requests arrive over time, and at time a_π, when a request π arrives, all attributes of π are revealed. At each time t, we need to decide whether to schedule a service at this time, and, if so, what service tree to use. These decisions depend only on the information revealed up to time t and are irrevocable. As computing an optimal schedule in an online fashion is in general not possible, we use the standard notion of competitive analysis to evaluate online algorithms.

2 Single-Phase Game

Almost all existing lower bound proofs for online versions of MLAP in [7, 4, 5] utilize a reduction to a variant of MLAP that we will refer to as the *Single-Phase MLAP*. The only exception we are aware of is the lower bound for the TCP Acknowledgement Problem, or 1-level MLAP (see the next section). In the Single-Phase MLAP all requests are issued at the beginning, at time 0. However, the adversary can terminate the game at any time ξ, that we will refer to as the *expiration time*, unknown to the online algorithm. All requests that are still pending at time ξ are cancelled, and they only contribute their waiting cost (from the release time until time ξ) to the overall cost. Note that Single-Phase MLAP is *not* a special case of MLAP, due to the fact that we do not require all requests to be serviced.

We claim that a lower bound of R on the competitive ratio for the Single-Phase MLAP implies a lower bound of R for MLAP. Suppose we have some adversary strategy $\mathcal{F}$ for the Single-Phase MLAP that uses an instance $\mathcal{I}$ and forces ratio R. The idea is to concatenate together a large number of copies of $\mathcal{I}$, with each consecutive copy "compressed" by some large factor. More specifically, choose some large constant M. In the i-th copy of $\mathcal{I}$ we will replace each request by M^i identical requests (think about it as assigning it weight M^i) and scaling down (dividing) all time values by M^i. So, roughly, the instances get shorter and shorter but also heavier and heavier. These two changes ensure that if we execute $\mathcal{F}$ in each instance, the waiting costs will remain the same. (In [7], the authors use the fitting term "zooming" to describe this trick.) Each copy of $\mathcal{I}$ may leave some unserved requests, but these can be all satisfied at the end with a single service that only contributes a constant additive term to the adversary cost.

We need two more observations before we continue. One is that the above reduction applies to sub-problems of MLAP where restrictions are placed on the trees that can appear in the instance, since all copies of the original instance of MLAP in this reduction use the same tree and only duplicate the requests.

The second observation is that this reduction applies as well to sub-problems of MLAP where we place some restrictions on the waiting cost function, for example when we assume that the waiting cost function is linear or a step function.

3 Single-Level Aggregation

Consider first the case when the depth of $\mathcal{T}$ is 1, that is all nodes of $\mathcal{T}$ other than r are r's children. Since no aggregation takes place in such a tree, we can process each edge separately, and thus we can assume that $\mathcal{T}$ is in fact just a single edge (r, q) of length $L = 1$. In this case, assuming further that the cost function is linear, MLAP is equivalent to the Dynamic TCP Acknowledgement model, mentioned earlier, introduced by Dooly, Goldman and Scott [8, 9] in 1998. It captures the algorithmic essence of the dilemma facing the implementers of the TCP protocol, namely how to achieve a good balance between the congestion and delay. Needless to add, the model from [8, 9] is a very rough simplification; the actual problem involves a number of other technical constraints that are difficult to model mathematically.

Recall that our objective is to minimize the total cost:

$$\text{cost} = (\text{number of acknowledgement packets}) + (\text{total delay}).$$

It is not hard to prove that no online algorithm can achieve competitive ratio better than 2. There are actually two completely different adversary strategies that we can use to prove this lower bound. The first strategy uses the reduction to Single-Phase MLAP, as described in the previous section. In the lower bound strategy, the adversary issues just one request at q. The algorithm eventually must serve this request, say at time t. Then the adversary terminates the phase right after time t. The algorithm pays $1 + t$, while the adversary's cost is $\min(1, t) \leq \frac{1}{2}(1 + t)$.

There is also a more direct proof that does not use the reduction to Single-Phase MLAP, which goes as follows. For any given online algorithm $\mathcal{A}$, the adversary issues one request at q at a time, with each new request issued right after the previous one is served. The adversary can choose one of two strategies: serve when odd-numbered requests arrive, or when even-numbered requests arrive. The cost of these two strategies together is equal to $\mathcal{A}$'s cost, so one of them achieves cost not larger than half of $\mathcal{A}$'s cost.

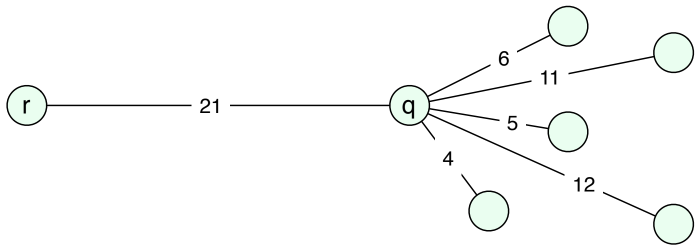
To match this lower bound, we can employ a strategy that resembles the one for the ski-rental problem. After each service we wait until the new accumulated waiting cost becomes equal 1, and then we serve q again. The cost of any phase, namely the waiting interval plus the subsequent service cost, is 2. Within any phase the adversary cost must be at least 1, since the adversary must either serve q or pay 1 for waiting. So the competitive ratio is at most 2.

The randomized case is more interesting. Seiden [16] gave a lower bound of $e/(e-1)$ (this result was independently obtained by Noga), but proving a matching upper bound turned out to be more challenging. A natural approach would be to schedule each service according to some probability distribution on the accumulated waiting cost, but no solutions along these lines are known. Using a more sophisticated approach, Karlin, Kenyon and Randall [11] were able to design a randomized algorithm with ratio $e/(e-1)$.

For a bit of historical perspective, a few comments about the offline version of the 1-level MLAP, or the TCP Acknowledgement Problem. In [8, 9], the authors also showed that the optimum schedule can be computed in time $O(n^2)$ with dynamic programming. Interestingly, as it turned out, this is not a new problem. The 1-level MLAP is also mathematically equivalent to the economic lot sizing problem, that was formulated by Wagner and Whitin [17] in 1958. (The history of the lot-sizing problem can be traced, in fact, as far back as 1913, to the paper by Ford Whitman Harris [10].) The problem is concerned with scheduling orders for a commodity to satisfy a collection of demands. These demands appear at specified times and they need to be satisfied from the current inventory. The inventory can be replenished by orders, with each order having a fixed cost, independent of its size. The objective is to minimize the sum of the ordering costs and the inventory holding cost, where the holding costs accrue at the rate proportional to the inventory size. You can think of the lot sizing problem as the TCP Acknowledgement Problem with the time direction reversed, with demands corresponding to packets and orders corresponding to transmissions. In their paper, Wagner and Whitin also gave an $O(n^2)$-time algorithm. As shown by Aggarwal and Park [1] in early 1990s, the running time can be improved to $O(n \log n)$ by exploiting the Monge property.

4 Two-Level Aggregation

Just like for the single level, for two-level trees we can assume that the root has only one child. The resulting tree has a broom-like shape, with root r connected to the central node q, and q connected to the leaves:



The length of edge (r, q) will be denoted by L and the length of each edge (q, v), where v is a leaf, will be denoted ℓ_v.

This model has been studied in several contexts. Brito *et al.* [6] called such trees *flat trees* and studied the online TCP Acknowledgement Problem in this model. They developed an 8-competitive algorithm for such trees.

Mathematically, the 2-level MLAP is equivalent to the *Joint Replenishment Problem (JRP)*, well studied in the operations research literature. The offline version of JRP is $\mathbb{NP}$-hard [2], and, recently, there has been substantial effort in designing offline approximation algorithms for JRP, see [13, 14, 15, 3]. JRP can be thought of as a generalization of the lot sizing problem to the case of multiple retailers. Each retailer has demands that arrive at certain times. These demands need to be satisfied from their inventories. The inventories can be replenished with orders, but now each order is realized in two stages: first, the items are delivered from the supplier to the warehouse, at some cost L, from which they are redistributed to retailers that issued these orders, paying cost ℓ_x for each retailer x involved in the order. As before, we also have holding cost functions for the inventories. Similar to the single-level case, the correspondence between JRP and the 2-level MLAP involves reversing the time direction.

As stated, JRP does not quite make sense as an online problem, since the orders take place before the demand times. However, there is also a natural version of JRP when orders (or shipments) take place *after* the demands are issued. In this case there are no inventories, but instead of holding costs we have waiting costs that reflect the urgency of different demands. This model was studied in the online scenario by Buchbinder *et al.* [7], who gave a 3-competitive algorithm, improving the ratio from [6], and proved a lower bound of 2.64. This lower bound has been recently improved in [4] to 2.75. (These lower bounds hold even for the linear cost function.) The same paper gives a 2-competitive algorithm and a matching lower bound for JRP-D, the version of JRP with deadlines.

In this section, we will sketch some of these results. First, we will discuss the proofs for lower and upper bounds of 2 for JRP-D. We then explain the ideas behind proving a lower bound of 2.75 for the linear cost function, and the 3-competitive algorithm for arbitrary cost functions.

In this section we will use terminology inspired by JRP, referring to the root r as the supplier, to the central node q as the warehouse, to the leaves as retailers. Terms "request" and "demand", as well as "service" and "shipment" will be used interchangeably. L will be called the warehouse shipping cost and each ℓ_v, for a retailer v, will be called the retailer v's shipping cost.

4.1 Lower Bounds for Two Levels

Lower bound of 2 for the deadline case. As shown in [3], no online algorithm for JRP can have competitive ratio better than 2, even for the deadline case. The instance used in the proof has some large number $N+1$ of retailers $v_0, v_1, ..., v_N$. The warehouse shipping cost is $L = 1$, the retailer shipping costs are $\ell_{v_0} = 0$ and $\ell_{v_i} = 1$ for $i = 1, ..., N$. Each v_i has a request issued at time 0 with deadline i. So any online algorithm $\mathcal{A}$ must serve v_i no later than at time i. By a simple exchange argument, we can assume that each v_i is served earlier than or at the same time as v_{i+1}. If $\mathcal{A}$ serves each request separately, its total cost is $2N + 1$, while the adversary can serve all requests at the beginning paying only $N + 1$, so the ratio approaches 2 for $N \to \infty$. So suppose that some v_k is served together with v_{k+1} and let k be the smallest such index. Then v_{k+1} is served strictly before its deadline $k+1$. The adversary will stop the phase right after this shipment. The algorithm's cost is at least $1 + 2(k-1) + 3 = 2k + 2$ (note that this is correct even for $k = 0$), while the adversary at time 0 can serve $v_0, ..., v_k$ (but not v_{k+1}), paying $k + 1$, so the ratio is 2.

A better lower bound. For the linear waiting cost function, Buchbinder *et al.* [7] gave a lower bound proof of 2.64, which was recently improved to 2.75 in [4]. The proof in [7] uses only a constant number of retailers, while the strategy in [4] follows the general strategy for the deadline case, outlined above, with an unbounded number of retailers. We will now sketch the idea of this proof.

We will assume the cost function to be linear, although the argument works as well for any continuous, non-negative and increasing function. One way to think about this proof is as "simulating" the strategy for deadlines. We use essentially the same instance as before, but with a little twist. Instead of just one request at v_i, we will issue ω_i requests at v_i at time 0, where the numbers ω_i are quickly decreasing, $\omega_0 \gg \omega_1 \gg ... \gg \omega_i \gg ...$. It is more convenient to think about this as having one request in v_i with weight ω_i, so that its waiting cost function is $\omega_i t$, for $t \geq 0$. As before, we can assume that the requests are served in order $v_1, v_2, ...$, possibly several at the same time. Note that to be competitive, an online algorithm needs to serve v_i no later than at time $O(1/\omega_i)$. Also, at this time the waiting cost of the remaining requests at $v_{i+1}, v_{i+2}, ...$ is negligible. The adversary strategy is very similar to that for the deadlines: finish the phase if either (i) $\mathcal{A}$ serves v_i and v_{i+1} together, or (ii) $\mathcal{A}$ serves v_i before some threshold time σ_i.

Think about this game from the point of view of just a single retailer v_i, as if we only wanted to be competitive with respect to v_i's cost. Then this game is equivalent to the single-level MLAP, for which the optimal competitive ratio is 2. But here, the adversary can also serve many requests together at the beginning of the phase, while $\mathcal{A}$ can do this only once, since as soon as $\mathcal{A}$ serves more than one retailer, the phase ends. So the intuition is, we should be able to beat ratio 2.

The rest is basically just calculation. In the proof in [3], to optimize the lower bound, the costs ℓ_{v_i} are actually taken to be some constant c (not necessarily 1). Equalizing the ratios for all i and both types of endings (whether $\mathcal{A}$ serves v_i too early, or together with v_{i+1}), and solving the resulting recurrence, we get that the best choice of c is the root of $c^3 + c^2 - 1 = 0$, $c \approx 0.7548$, and the lower bound on the competitive ratio is $2 + c \approx 2.7548$.

4.2 Upper Bounds for Two Levels

In this section we will review upper bounds for the competitive ratio for the 2-level MLAP, a.k.a. JRP. We will start with the version with deadlines, JRP-D, for which we present a 2-competitive algorithm. We then show how to extend it to JRP with an arbitrary waiting cost function, achieving ratio 3. We will formulate the algorithms for arbitrary costs, but the ideas for the competitive analysis proofs will be presented only for instances where all retailer shipping costs ℓ_v are very small compared to L, the warehouse shipping cost.

4.2.1 Upper Bound of 2 for JRP-D

The 2-competitive algorithm for JRP-D appeared in [4]. The idea is to schedule shipments in batches, with each shipment taking place when some request is about to expire. Suppose we already shipped batches $B_1, ..., B_j$, with each B_i serviced at time t_i. We then wait until we reach the deadline d_v of some retailer v, where the *deadline of a retailer* is the earliest deadline of the pending requests in this retailer. Then $t_{j+1} = d_v$ and B_{j+1} is the maximum set of retailers with earliest deadlines whose total retailer cost does not exceed L. This retailer v, as well as the request in v with deadline d_v, is said to *trigger* batch B_{j+1}.

As mentioned earlier, to present the idea behind the competitive analysis of this algorithm, we will focus on the case when L is a very large integer and $\ell_v = 1$ for all v. In this case each batch B_i will have retailer cost exactly L.

For each batch B_i, define D_i to be the earliest deadline of a request that is pending at time t_i but is not included in batch B_i. We will say that batch B_i is *timely* if $t_i = D_{i-1}$. To see why this notion is useful, suppose that batch B_i is not timely, that is $t_i < D_{i-1}$. (Note that t_i cannot be larger than D_{i-1}.) This batch B_i is triggered by some request with deadline t_i. But then the definition of D_{i-1} implies that this request could not have been pending at time t_{i-1}, so it must have been released between times t_{i-1} and t_i. In such a case, we know that the adversary must have a shipment scheduled between t_{i-1} and t_i. This observation will help us estimate the adversary cost.

We divide the computation into phases. The first phase starts with the first batch B_1 (so t_1 is the first deadline in the instance). Suppose we have already defined some number of phases and that the next phase starts at batch B_g. Then this phase will end with the earliest batch B_h, $h \geq g$, such that the adversary has a shipment between times t_h and t_{h+1}. Naturally, the next phase starts with batch B_{h+1}.

Consider a phase $B_g, ..., B_h$. We now estimate the adversary cost associated with this phase. By definition, the adversary's schedule must have a shipment between t_{g-1} and t_g. We associate the warehouse cost of L of this shipment with this phase. For any batch B_j, $j = g, ..., h-1$, since the adversary does not have a shipment between t_j and t_{j+1}, B_{j+1} must be timely, which implies that each retailer v in B_j has deadline no later than at time t_{j+1}. If ρ is the request in v with this deadline, the adversary must have served ρ at some time t between a_ρ and d_ρ. Also, the algorithm could not have had any shipment for v between r_ρ and t_j, for otherwise ρ would not be pending at time t_j. We can thus associate, in a unique way, the adversary cost of serving v at t to the phase $B_g, ..., B_h$. (Note that we are not claiming that t is actually within this phase; in general, t could be in some earlier phase.) These costs will add up to $(h-g)L$, and adding the warehouse shipment cost of L right before time t_g, the total adversary cost associated with this phase is $(h-g+1)L$. But the algorithm pays $2(h-d+1)L$, only twice as much, so the competitive ratio is at most 2.

4.2.2 Upper Bound of 4 for JRP

We now show how to generalize the algorithm for JRP-D to arbitrary waiting cost functions. We first show how to get competitive ratio 4, and then later outline the idea for reducing it to 3. The basic idea is that now, instead of transmitting at a deadline, we transmit when the total waiting time of some subset of retailers (called the critical set) is equal to its shipping cost. The ratio is 4 because, intuitively, our cost doubles (compared to JRP-D), since each shipment is first paid for with the waiting cost, while the optimum algorithm can only pay one cost (for waiting or shipping.

At any time t, denote by $w_t(v)$ the number of requests at a retailer v, that we also refer to as the *weight* of v. We use notation $twc_t(v)$ for the total waiting cost of the requests pending in v at time t (counting only the cost accumulated until time t). A retailer v is called *mature* if $twc_t(v) \geq \ell_v$. For immature retailers, the time $mtr_t(v) = t + (\ell_v - twc_t(v))/w_t(v)$ is called the *maturity time* of v. Intuitively, this is the time when the requests that are currently at v will pay ℓ_v for waiting. Note that if the set of requests in v does not change in some time interval then v's maturity time remains constant. When t is understood from context, we will omit the subscript t and write simply $w(v)$, $twc(v)$, and $mtr(v)$. We will also extend some of these notations to sets of retailers.

At a time t, a set C of retailers is called *critical* if (i) $twc_t(u) \geq \ell_u$ for all $u \in C$, (ii) $twc_t(u) < \ell_u$ for all $u \notin C$, and (iii) $\sum_{v \in C} twc_t(v) = L + \ell(C)$. The last condition says that the waiting cost of C is equal to its shipping cost. It is easy to see that if a critical set exists, then it is unique, and that all retailers in the critical set are mature.

Algorithm $\mathcal{A}4$. If there is no critical set, do nothing. Otherwise, let C be the critical set. Let X be the maximum set of retailers not in C, chosen in order of increasing maturity times, such that $\ell(X) \leq L$. Transmit the batch $B = C \cup X$.

As before, we will analyze this algorithm only for the special case when the instance has the warehouse shipping cost L, for some large L, and all retailer shipping costs are $\ell = 1$.

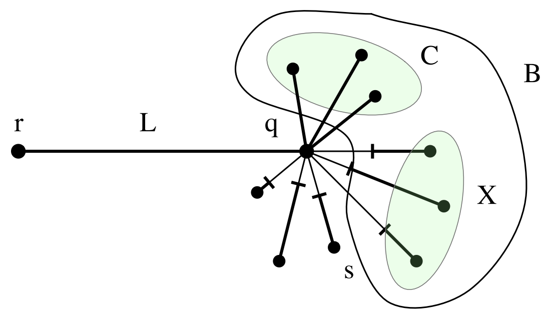


Figure 2: An example of a batch $B = C \cup X$. Thick lines represent portions of the edges (shipping costs) paid for with the waiting cost. Retailers are ordered clockwise according to their maturity times. C is the critical set, X is the set of extra retailers, and s is the special retailer.

We number the batches $B_1, B_2, \ldots$, and their transmission times are denoted $t_1, t_2, \ldots$. The corresponding critical sets and extra sets are denoted C_i and X_i, respectively. So $B_i = C_i \cup X_i$. (See Figure 2.) The vertex $s \notin B_i$ with minimum $mtr_{t_i}(s)$ is called the *special node at time* t_i and is denoted s_i and let $D_i = mtr_{t_i}(s_i)$. Generalizing the definitions from the previous section, batch B_i will be called *timely* if $t_i \geq D_{i-1}$.

We will again divide the batch sequence into phases. The first phase starts with batch B_1. If some phase starts with batch B_g, it ends with the first batch B_h, for $h \geq g$, such that either the adversary has a shipment between times t_h and t_{h+1} or batch B_{h+1} is untimely. This is very similar to what we did for the deadline case, but we need to include the second option (ending the phase before untimely batches), because the adversary may not schedule any shipments for a long time and only pay for waiting. This second option will also allow us to associate the "extra" cost of L with a phase.

We now focus on one phase $B_g, \ldots, B_h$ and define the adversary cost associated with this phase. Consider some retailer $v \in B_j$, where $j \in [g, h]$, except that we assume that $v \notin C_h$. Let t_i be the last time when the algorithm served v before time t_j. If $v \in C_j$ then v accumulated waiting cost at least 1, because v is mature at time t_j. So the adversary must either serve v between t_i or t_j or pay for waiting, paying the cost of 1 in either case. We associate this adversary cost, in a unique way, with this service of v at time t_j. If $v \in X_j$, then we can use the same argument to associate an adversary cost of 1 with our service of v, but we need to be a bit more careful, since v is not mature at time t_j. But, by the definition of phases, v will mature before time t_{j+1} and the adversary does not have any shipments before time t_{j+1}, so the earlier argument still applies.

We claim that the adversary must pay some extra cost of L for each phase, which is not included in the costs discussed in the previous paragraph. If the adversary has a shipment between t_{g-1} and t_g then this extra cost of L is the warehouse shipping cost of this shipment. Otherwise, B_g is not timely, that is $D_{g-1} > t_g$. This implies that the requests in each node $u \in C_g$ that were pending at time t_{g-1}, by time t_g will accumulate waiting at most 1. Since the total waiting cost of C_g is $L + |C_g|$, this additional waiting cost of L must have been paid by the new requests, released between t_{g-1} and t_g. We then use this waiting cost of L as the extra cost associated with phase $B_g, \ldots, B_h$.

Putting it together, the adversary cost associated with phase $B_g, \ldots, B_h$ is $L + \sum_{j=g}^{h} |C_j| + \sum_{j=g}^{h-1} |X_j| = (h - g + 1)L + \sum_{j=g}^{h} |C_j|$. (Recall that $|X_j| = L$, by our simplifying assumption.)

For each batch B_j the algorithm pays $L + |B_j|$ for shipping and at most $L + |B_j|$ for waiting. The algorithm's cost is at most $\sum_{j=g}^{h}(2L + 2|C_j| + 2|X_j|) = (h - g + 1)4L + 2\sum_{j=g}^{h}|C_j|$, which is not more than 4 times the adversary cost. This argument could be easier to visualize in terms of charging: For each batch B_j, where $j \neq h$, charge its cost of $2L + 2|B_j|$ to the adversary cost associated with the nodes in B_j. Since $|B_j| \geq L$, the charging ratio is at most 4. As for B_h, charge the cost of C_h (at most $2|C_h|$) to the adversary cost associated with C_h, and the remaining cost of at most $2L + 2|X_h|$ to the extra adversary cost of L associated with the phase. Here, the charging ratio is also at most 4.

4.2.3 Improving the Upper Bound for JRP to 3

A 3-competitive algorithm for JRP, for arbitrary waiting cost functions, was given by Buchbinder *et al.* [7]. They formulated their algorithm as a primal-dual algorithm. Below, we sketch how Algorithm $\mathcal{A}4$ above can be modified to give the same ratio of 3. Whether there is a better-than-3-competitive algorithm online algorithm for JRP remains an open problem.

To improve the ratio, note that in some situations we can schedule shipments earlier, before a critical set appears. Specifically, if we already scheduled a batch B_j, we would like to schedule the next batch at the maturity time of the special node s_j. The hope is that this should save us a waiting cost of L associated with a critical set. In the previous algorithm, the last batch B_h of a phase was charged partly to mature nodes in C_h, and partly to the extra cost of L for this phase. This we can still do. But in any earlier batch B_j in this phase we needed that the nodes in X_j mature within this phase so that we can charge to the adversary cost associated with this phase. We can still do this, as long as B_{j+1} is timely. This means that we can schedule B_{j+1} at the maturity time of s_j, and we are okay. The only potential problem is that new requests arriving after time t_j could create a critical set before s_j matures, in which case we cannot wait any longer and we need to schedule a shipment when this critical set appears. But in the analysis this starts a new phase, with the adversary paying an extra cost of L for this phase, so the analysis should extend to this case.

Algorithm $\mathcal{A}3$. Suppose we have already shipped batches $B_1, ..., B_j$. The next shipment B_{j+1} occurs either when a critical set appears or at time D_j, whichever happens first. If a critical set appears at some time $t < D_j$, we let $t_{j+1} = t$ and C_{j+1} is this critical set. If there is no critical set until time D_j, then $t_{j+1} = D_j$ and C_{j+1} is the set of mature nodes at that time. The transmitted batch is $B_{j+1} = C_{j+1} \cup X_{j+1}$, where X_{j+1} is the maximum set of nodes not in B_j, ordered in order of increasing maturity times, such that $\ell(X_{j+1}) \leq L$.

5 Aggregation on the Line

Another natural special case of MLAP is that when the tree is just a path. It is more convenient to think about this problem as aggregation on a line, with the "root" node being at $r = 0$ and all requests at points $x > 0$. Brito *et al.* [6] gave an 8-competitive online algorithm. Recently, Bienkowski *et al.* [5] improved this result by proving that the competitive ratio of this problem is between $2 + \phi \approx 3.618$ and 5, and they provided a polynomial time algorithm that computes an optimal schedule.

Getting any constant ratio is quite simple: Divide the positive half-line into intervals between powers of two, $J_i = (2^i, 2^{i+1}]$, for all integers i (including negative integers). For each interval apply the ski-rental strategy. Specifically, if the waiting cost of all requests in J_i is at least 2^{i-1} then serve all pending requests in $(0, 2^{i+1}]$.

The cost associated with this service is then at most $3 \cdot 2^i$; this includes 2^{i+1} for service, 2^{i-1} for the waiting cost in J_i, and at most another 2^{i-1} for the waiting cost at all intervals J_p, for $p < i$.

Consider any two consecutive services from 2^{i+1}, say at time t and t'. If the adversary does not have a service from J_i between t and t', his waiting cost is at least 2^{i-1}. Otherwise, the adversary has at least one service from some point in J_i between t and t'. We will associate the portion J_{i-1}, of cost 2^{i-1}, of this adversary service with our service at t'. (Think about this service as an interval $(0, x]$, for $x \in J_i$, and divide it into intervals J_p, for $p < i$, plus interval $(2^i, x]$. We then associate the interval J_{i-1} with the service at t'.) Note that with different services we associate different portions of the adversary cost.

So, overall, we pay at most $3 \cdot 2^i$, and charging it to the associated adversary cost of 2^{i-1}, the ratio is 6. Thus the competitive ratio is at most 6. With a slightly different algorithm and charging, this ratio can be improved to 5 [5].

The lower-bound proof in [5] is a bit technical. It uses again the argument based on Single-Phase MLAP. The requests are initially issued at points b_i, for all positive integers i, with b_i's increasing more or less like a geometric sequence. Also, the number of requests w_i issued at b_i (or weights of requests) are fast decreasing. The adversary waits until either some request is served too early (earlier than an appropriate threshold), or the algorithm serves two requests at the same. If either of these two events happens, the phase ends. Equalizing the resulting ratios and setting this up as a recurrence, yields the lower bound $2 + \phi \approx 3.618$ on the competitive ratio.

6 Summary

Very little is known about the general case of MLAP, for arbitrary trees. Khanna *et al.* [12] considered an equivalent problem phrased in terms of message aggregation in networks, and they gave an online algorithm with competitive ratio $O(\log C)$, where C is the total cost of the tree. The waiting cost function in [12] is assumed to be linear, so their result applies to MLAP-L. To our knowledge, there are no other results in the literature on this problem.

	deterministic		randomized	
	lower bound	upper bound	lower bound	upper bound
1-Level (TCP Ack.)	2	2	$e/(e-1)$	$e/(e-1)$
2-Levels, deadlines (JRP-D)	2	2		
2-Levels (JRP)	2.754	3		
Line	3.618	5		

The table above summarizes the known bounds on the competitive ratios for different variants of MLAP. The upper bounds for JRP and the line are valid for arbitrary waiting cost functions [7, 5], while the lower bounds hold even for linear waiting cost functions [4, 5].

The most intriguing open problem at this time is whether there is a constant-competitive online algorithm for MLAP. This is open even for MLAP-D and MLAP-L, that is special cases with deadlines and linear waiting costs, respectively. Also, what is the best competitive ratio for JRP? Improving the upper bound of 3 [7] or the lower bound of 2.75 [4] will require significant new insights.

References

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

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

[3] Marcin Bienkowski, Jaroslaw Byrka, Marek Chrobak, Neil Dobbs, Tomasz Nowicki, Maxim Sviridenko, Grzegorz Swirszcz, and Neal Young. Algorithms for the joint replenishment problem with deadlines. In *Proc. 40th Int. Colloquium on Automata, Languages, and Programming (ICALP'13)*, pages 135–147, 2013.

[4] Marcin Bienkowski, Jaroslaw Byrka, Marek Chrobak, Łukasz Jeż, Dorian Nogneng, and Jiri Sgall. Better approximation bounds for the joint replenishment problem. In *Proceedings 25th Symposium on Discrete Algorithms (SODA'14)*, pages 42–54, 2014.

[5] Marcin Bienkowski, Jaroslaw Byrka, Marek Chrobak, Łukasz Jeż, Jiří Sgall, and Grzegorz Stachowiak. Online control message aggregation in chain networks. In *Proc. of the 13th Algorithms and Data Structures Symposium (WADS)*, pages 133–145, 2013.

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

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

[8] Daniel Dooly, Sally A. Goldman, and Stephen D. Scott. TCP dynamic acknowledgment delay: theory and practice. In *Proceedings of the Thirtieth Annual ACM Symposium on Theory of Computing*, pages 389–398. ACM Press, 1998.

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

[10] Ford Whitman Harris. How many parts to make at once. *Factory: The Magazine of Management*, 10:135–136, 152, 1913.

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

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

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

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

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

[16] Steven S. Seiden. A guessing game and randomized online algorithms. In *Proceedings of the Thirty-Second Annual ACM Symposium on Theory of Computing (STOC), 2000*, pages 592–601, 2000.

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