Putting Off the Catching Up: Online Joint Replenishment Problem with Holding and Backlog Costs*

Benjamin Moseley† Aidin Niaparast† R. Ravi†

Abstract

We study an online generalization of the classic Joint Replenishment Problem (JRP) that models the trade-off between ordering costs, holding costs, and backlog costs in supply chain planning systems. A retailer places orders to a supplier for multiple items over time: each request is for some item that the retailer needs in the future, and has an arrival time and a soft deadline. If a request is served before its deadline, the retailer pays a holding cost per unit of the item until the deadline. However, if a request is served after its deadline, the retailer pays a backlog cost per unit. Each service incurs a fixed joint service cost and a fixed item-dependent cost for every item included in a service. These fixed costs are the same irrespective of the units of each item ordered. The goal is to schedule services to satisfy all the online requests while minimizing the sum of the service costs, the holding costs, and the backlog costs.

Constant competitive online algorithms have been developed for two special cases: the make-to-order version when the deadlines are equal to arrival times [10], and the make-to-stock version with hard deadlines with zero holding costs [7]. Our general model with holding and backlog costs has not been investigated earlier, and no online algorithms are known even in the make-to-stock version with hard deadlines and non-zero holding costs. We develop a new online algorithm for the general version of online JRP with both holding and backlog costs and establish that it is 30-competitive. Along the way, we develop a 3-competitive algorithm for the single-item case that we build on to get our final result. Our algorithm uses a greedy strategy and its competitiveness is shown using a dual fitting analysis.

1 Introduction

The *JRP* is a fundamental problem in supply chain management [20, 19, 17, 23, 10, 7], and it has been studied both in the offline and online models.

Offline JRP. In the offline version of the JRP (see [20] for pointers to relevant work), there is a set of n commodities (also called items) that a retailer stocks and sells over a planing horizon T. The demand for each item at each time period is assumed to be known (either as a given set of deterministic quantities or via a description of the demand distribution for each item over time). The demand for an item at a given time t must be fulfilled by units of the item ordered at or before t, i.e., no backlogging (delaying the satisfaction of the demand) is allowed.

With no costs for ordering, the retailer will simply order as many units of each item as is demanded as they arise over time. Ordering items, however, incurs costs for the retailer: every order irrespective of the set of items ordered incurs a joint ordering cost $c(r)$; additionally, if units of item v are part of the order, there is an item-dependent cost of $c(v)$ irrespective of the number of units of item v ordered.

With nonzero ordering costs, the retailer will then order the total demand for all items at the beginning of the planning horizon in a single order, but this ignores practical constraints and costs in storing these items. JRP models this with a holding cost function $h^v_{tt'}$ for holding one unit of item v for the period t (when it is ordered) to t' (when it is used to satisfy demand). In this paper, we assume that the holding cost is h per unit time for all items, so that $h^v_{tt'} = h \cdot (t' - t)$ for all items v. The goal of JRP is to find an ordering policy that minimizes the sum of ordering costs and holding costs. This version has also been called the *make-to-stock* JRP because items are ordered to be stored until they are consumed.

In the *make-to-order* version of JRP (studied in [10]), holding inventory is not allowed. Hence, the demand for each item at t can only be satisfied by items ordered at or after t. Instead of holding costs, there are backlog or delay costs ($b^v_{tt'}$) to satisfy the orders as above, which must be minimized in addition to the ordering costs. The make-to-stock and make-to-order versions of JRP are equivalent in their offline versions [19].

The bulk of the research on JRP assumes either offline deterministic demand, where the demands for each item and each time period are known in advance, or stochastic demand, where the probability distributions of future demands are

*The full version of the paper can be accessed at https://arxiv.org/abs/2410.18535. This material is based upon work supported in part by the Air Force Office of Scientific Research under award number FA9550-23-1-0031, Google Research Award, an Infor Research Award, a Carnegie Bosch Junior Faculty Chair, NSF grants CCF-2121744 and CCF-1845146 and ONR Grant N000142212702.

†Tepper School of Business, Carnegie Mellon University, USA. moseleyb,aniapara,ravi@andrew.cmu.edu

known (see the surveys [17, 23]). However, these assumptions may be unrealistic in practical scenarios, and motivated the study of online algorithms [8] for the problem, which assumes future demands are completely unknown and generated by an adversary.

Online JRP with holding and backlog. Buchbinder et al. [10] study the online version of make-to-order JRP (with no holding), and give an elegant 3-competitive algorithm using the online primal-dual framework. They also provide a lower bound of approximately 2.64 on the competitiveness of any deterministic online algorithm for the problem. On the other hand, for the online version of the make-to-stock version of the JRP (with no backlogs) in the special case where holding costs are zero, Bienkowski et al. [7] designed a 2-competitive algorithm and showed that this competitive ratio was optimal.

In reality, the problem faced by retailers involves ordering for items jointly, while allowing for both holding inventory and backlogging some demand when it is not available. In particular, if we wish to model the adversarial (non-stochastic) demand model online, there is a specific time (arrival time) when the retailer is cognizant of the demand for a future day (as a result of either a planning exercise or a bulk order with a specific target delivery date). This naturally sets up the online arrival of demand information for each item, which is associated with an arrival time, a deadline, and associated costs for early or late fulfillment. In this paper, we model and study the online version of this general variant of JRP.

Motivated by the scheduling literature [24], we assume that each demand has an arrival time and a deadline. The arrival time can be thought of as the earliest time at which the requirement for a deterministic demand at the deadline materializes. Each demand can only be satisfied using services at times that are not earlier than its arrival time. Our model captures the scenario where a retailer anticipates at time a that it is going to run out of item v at time $d \geq a$. So, it can only send a request after the arrival time a to the supplier for the item v to try and meet the deadline d. We note that papers addressing the traditional offline versions of the JRP do not consider the notion of an arrival time for demands and assume instead that they are known at the beginning of the planning horizon (or equivalently, that their arrival time is zero). With the introduction of the arrival time in the offline models, the equivalence of the make-to-order and make-to-stock versions (described in [19]) no longer holds. However, it is natural and important to introduce this notion of an arrival time in the online formulation.

In the definition below, we follow the literature on multi-level aggregation problems [12] and model the demand requests as arising in the leaves of a tree with two levels: the root node represents the joint ordering cost and each leaf represents an item or commodity with its node cost being the item-specific ordering cost.

PROBLEM 1.1. (ONLINE JOINT REPLENISHMENT PROBLEM) *Let T be a two-level tree with root r and leaf nodes $v_1, \ldots, v_n$. Each node $v \in V(T)$ has a cost $c(v)$ associated with it: the root cost represents the joint ordering cost (also called fixed service cost), and the leaf costs represent the item-specific ordering costs (also called fixed item costs).*

A set of requests arrive at different leaves over time, with possibly more than one request arriving at the same time. Each request ρ is specified by a tuple $\rho = (v_\rho, a_\rho, d_\rho)$, where v_ρ is the leaf node corresponding to the commodity that makes the request, a_ρ is the arrival time of the request, and $d_\rho \geq a_\rho$ is the (soft) deadline of the request. We assume each request is for one unit of the commodity since we can replicate demands for more units accordingly. Moreover, we assume that there is no ordering cost associated with each unit of each item[1]*. In the online version, at each time t, only the requests with arrival time before or at time t are known, while nothing is known about future requests. In particular, the total number of requests and the length of the time horizon of the problem are unknown.*

Each request can only be satisfied at or after its arrival time, but possibly after its deadline. The holding and backlog costs are assumed to be linear and uniform among different requests. In particular, if a request ρ is satisfied at a time $t \in [a_\rho, d_\rho]$, we incur a holding cost of $h \cdot (d_\rho - t)$, and if it is satisfied at a time $t \in (d_\rho, \infty)$, we incur a backlog cost of $b \cdot (t - d_\rho)$.

The requests must be satisfied using a series of services (the term we use for orders in this paper). A service is a subtree of T' of T that contains r, and can satisfy any subset of as yet unsatisfied requests made by nodes in $V(T')$. The service cost is $\sum_{v \in V(T')} c(v)$, which is the sum of the fixed joint cost of the service (root) and the costs of the items (leaves) included in the service. The problem is to minimize the total cost of servicing all the demand, which is the sum of the service costs, holding costs, and backlog costs.

1.1 Solution Approaches and Our Contribution The online make-to-order JRP with only backlog cost that is studied in [10] is naturally suited to the online primal-dual approach surveyed in [11]. In the algorithm suggested in [10], once a new order for some item arrives, the dual variables associated with satisfying that demand start to grow over time, and when a dual constraint becomes tight, a service is triggered. However, this approach cannot be used for cases where a demand can be

[1] This is because such costs must be paid by any solution and they are the same among different solutions; hence they can be ignored.

satisfied before its deadline (i.e., when holding inventory is allowed), and completely fails if a demand can not be satisfied after a deadline (i.e., if backlog costs are infinite).

On the other hand, the (make-to-stock) JRP with deadlines model (JRP-D) studied in [7] is also considerably simpler to handle. When a deadline triggers service, all active requests can be satisfied due to the simplifying zero holding costs assumption. Their algorithm then uses future deadlines in increasing order to accumulate more items until the joint ordering cost has been spent to preemptively satisfy some items. Then, a simple analysis shows that any optimal schedule must satisfy requests at least at half the rate (in costs) as this algorithm.

To understand the fundamental difference between handling general holding cost and backlog cost, consider the special case of the problem with only one item (i.e. one leaf). When there is only backlog cost (i.e., no holding is allowed), once a service is triggered at time t, the optimal decision is to include all the requested units of the item that have arrived before or at t in that service, as more backlog cost is incurred if the algorithm satisfies them using some later service. Therefore, the algorithm only needs to decide the times at which services are triggered. However, when there is only holding cost (i.e., no backlog is allowed), the algorithm can wait until the deadline of the first unsatisfied request to trigger a service since an earlier service would be wasteful and would incur additional holding costs. But when a service is triggered, the algorithm needs to decide which unsatisfied requests with deadlines in the future it wants to include in this service. The additional decision of which active requests to include in each service is a fundamental difficulty in handling holding costs. When both holding and backlog are allowed, the algorithm faces both challenges of when to trigger service and which active requests to fulfill preemptively at this service.

To appreciate this, consider the single-item problem with just backlog: the optimal deterministic algorithm waits until it accumulates backlog cost s, where s is the service cost. This accumulated backlog easily "justifies" this service, i.e., the dual variables associated with these requests can each be set to the amount of their backlog, which sums up to s. This way, it is easily proved that this algorithm is 2-competitive. With just holding costs, the situation is already more complicated. This is because when we trigger a service, there might not be enough active requests with deadlines in the future to fulfill by paying their holding cost up to value s, thus disallowing the local accumulation of dual values for this service. The problem becomes even more algorithmically challenging when there are multiple items (i.e. multiple leaves) as in the general JRP, as the algorithm has to also decide not only additional requests but also which subset of additional *items* it wants to include in the current service.

Our main contribution is a constant competitive algorithm for this problem, the first known algorithm with non-trivial worst-case guarantees. Note that the results of [10, 7] imply a lower bound of 2.754 for the competitive ratio of deterministic online algorithms even for the special version with no holding costs. To the best of our knowledge, no online algorithm is currently known even for the very special case of Online JRP with only one item and only holding costs and a strict deadline (no backlog allowed). Along the way to developing the full algorithm, we illustrate our key ideas for the general singe-item case.

THEOREM 1.1. *There exists a 3-competitive deterministic polynomial-time algorithm for Online JRP with backlog and holding costs with a single item.*

We then extend this to the general multi-item case with a worse competitive ratio.

THEOREM 1.2. *There exists a 30-competitive deterministic polynomial-time algorithm for Online JRP with backlog and holding costs.*

The algorithm is loosely inspired by the primal-dual approach for constructing approximation algorithms [1, 15, 11]. The main idea of the algorithm is to trigger services using accumulated greedy dual variable values of a natural linear programming formulation for the JRP. The service is triggered when these greedily growing dual variables (corresponding to constraints requiring serving the requests in the primal) become tight for some dual constraint. Despite this connection, the algorithm itself does not construct any dual solution explicitly but is stated as a simple greedy scheme. In contrast, [10] explicitly maintains and updates dual values in the algorithm.

As in a traditional primal-dual method, the duals corresponding to requests are increased simultaneously at the rate of backlog costs once their deadlines are past and time advances. Broadly, the algorithm waits until enough backlog cost is accumulated in a first backlog phase to trigger a service (and makes tight a constraint on the dual values summing up to no more than the service costs). To ensure feasibility of dual-fitting, we aggregate and serve some overdue requests for additional items in a second backlog phase. Finally, we also aggregate and serve requests with deadlines in the future for all included items in both phases. This ensures that the service cost, the backlog costs, and the holding costs paid are balanced.

We use dual-fitting to define a feasible dual solution of value within a constant factor of the total algorithm's costs to prove the competitive ratio.

In more detail, the algorithm accumulates backlog costs for requests for a specific item (whose deadlines have passed) until it 'fills' the item-specific ordering cost and the item becomes 'mature'. Mature items then 'overflow' their 'surplus backlog costs' towards filling the cost of the root (the joint ordering cost) in a 'Mature Backlog Phase'. A service is triggered when the root is fully paid for by these accumulating backlog costs. At a service, all overdue requests from all mature items are satisfied. In addition, in a 'Premature Backlog Phase', a few more premature items that are close to becoming mature are accumulated in roughly increasing order of their time to maturity as a preemptive step (similar to the 'simulation' step in the algorithm of [10]). Then, in a 'Local Holding Phase', for each included item in the service, requests with future deadlines that can be satisfied by paying an overhead up to the item-dependent ordering cost are preemptively included, again in increasing order of their deadline, in this service. Finally, a second preemptive 'Global Holding Phase' includes additional requests in increasing order of future deadlines among all included items by paying an overhead of at most the joint ordering cost. The details of the four phases are in Section 3. The bulk of our proof argues that the preemptive phases allow us to fit a feasible dual of value at least a constant fraction of the total costs paid by the triggering services.

We prove the single-item result in the next section before providing the algorithm for the general multi-item case in the following.

REMARK 1.1. *Although for simplicity throughout the paper we assume the holding and backlog cost functions are linear, all the results easily extend to the more general case where holding and backlog cost functions are non-negative and increasing (as long as they are the same across different items). In particular, the results still hold when the holding and backlog costs between times s and t, denoted by h_{st} and b_{st} respectively, satisfy the following property: for each $s' \leq s \leq t \leq t'$ we have $h_{st} \leq h_{s't'}$ and $b_{st} \leq b_{s't'}$. These general results will appear in a journal version of this paper.*

1.2 Related Work As noted above, offline JRP is studied in both the make-to-stock model, where only holding is allowed (but no backlogs), and the make-to-order model, where only backlog is allowed (but no holding). The special case of the make-to-order model where the delay function is linear is called JRP-L. The special case of the make-to-stock model where there are hard deadlines but zero holding costs is called JRP-D.

Offline Approximation Ratios. The offline JRP is known to be strongly $\mathbb{NP}$-hard [2, 4, 22], even for the special cases of JRP-D and JRP-L. Also, JRP-D is $\mathbb{APX}$-hard [22, 6].

On the positive side, Levi, Roundy and Shmoys [19] gave the first approximation algorithm for (offline make-to-stock) JRP, with a ratio of 2 under a general notion of holding costs. Their algorithm uses a primal-dual approach and can incorporate both holding and backlog costs. This ratio was subsequently improved to 1.8 by Levi et al. [18, 21]. Later, Bienkowski et al. [7] improved the approximation ratio to 1.791. For offline (make-to-stock) JRP-D, the special case with zero holding costs, the ratio was reduced to 5/3 by Nonner and Souza [22] and then to ≈ 1.574 by Bienkowski et al. [6].

The special case of the make-to-order JRP with a single item is also known as *Dynamic TCP Acknowledgement*. This problem was first introduced by Dooly, Goldman, and Scott [13, 14] in 1998. In the offline setting, they gave a dynamic programming solution that solves the problem optimally in $O(n^2)$ time.

Online Competitive Ratios. For the online version of (make-to-stock) JRP-D, Bienkowski et al. [7] designed a 2-competitive online algorithm and prove that it has an optimal competitive ratio.

Brito et al. [9] gave a 5-competitive algorithm for online make-to-order JRP. Buchbinder et al. [10] gave a 3-competitive algorithm for this problem. They also proved a lower bound of 2.64 for this problem, which was later improved to 2.754 in [7]. These lower bounds apply to the more restricted version of the problem, where the delay functions are linear.

For the online version of the single-item make-to-order JRP (Dynamic TCP acknowledgment), Dooly et al.[13, 14] gave a 2-competitive algorithm, and presented a matching lower bound for any deterministic algorithm. Seiden [25] proved an $\frac{e}{e-1}$ lower bound in the online setting for any algorithm. Later, Karlin, Kenyon and Randall [16] designed a randomized algorithm achieving $\frac{e}{e-1}$ competitive ratio. As we mentioned before, the single-item variant of the problem with just holding was not studied before in the online setting.

The make-to-order and make-to-stock versions of JRP are closely related to different Online Aggregation and Inventory Routing problems, respectively, which have been studied extensively. We review additional related literature on these problems in the full version of the paper.

2 Warm Up: Single-Item Case

In this section, we study the special case of Online JRP where there is only one item and prove Theorem 1.1. The underlying tree only consists of a root r and a leaf v. The cost of each service is always $s := c(r) + c(v)$. First, we present an algorithm for single-item Online JRP. Then we write a natural linear program relaxation for the problem and obtain its dual. We then use dual fitting to prove Theorem 1.1, i.e., we come up with a feasible dual solution whose objective value is at least $\frac{1}{3}$ the cost of our algorithm. This proves that the algorithm is 3-competitive.

2.1 Algorithm In this section, we describe our online algorithm for single-item online JRP. We begin by describing the algorithmic challenges.

Suppose that only one request has arrived for this single item so far with deadline d. Clearly, it is not optimal to trigger a service before d, as we incur an unnecessary holding cost. One option is to trigger a service at d, which results in no holding or backlog cost. This is indeed optimal if no other request arrives in the near future. However, in case some other requests come shortly after d, it would be beneficial to aggregate all requests together and serve them all at once using one service after all their deadlines. This approach prevents paying a service cost for each of them, at the expense of paying backlog costs for the overdue requests.

Since the algorithm does not have access to future requests, it should hedge its bets between these two cases. When there is no holding allowed, the single-item problem is equivalent to the TCP-acknowledgment problem, for which Dooly et. al. [13] show that the best deterministic approach is to **wait until the sum of the backlog costs of the overdue requests accumulates to be equal to the service cost** s (similar to the familiar ski-rental idea in online algorithms) and then trigger a service. Our proposed algorithm follows the same stopping rule to trigger a service. Moreover, when a service is triggered, we satisfy all of overdue requests using this service since otherwise, when we fulfill them later, we will pay even more backlog cost for them.

Unlike the TCP-acknowledgment problem, after deciding when to trigger a service, we face an additional algorithmic challenge. After serving overdue requests, there may still be *active* requests with future deadlines. These requests can be fulfilled using the current service by paying the holding costs until their deadlines. The question we face is which subset of these active requests we must include in the current service.

On one extreme, assume that the algorithm does not fulfill any of these requests using the current service. Then, in the case where the backlog cost b is large enough, the holding cost h is small enough, and the deadlines of active requests are uniformly distributed in the future, the algorithm triggers a new service for each of the active requests. Alternatively, the optimal algorithm satisfies all of them using the current service with small additional holding cost. At the other extreme, the algorithm pays a significant holding cost to satisfy the active requests using the current service, which might be highly suboptimal as there is the option of waiting to get closer to the deadlines of those requests so they can be served with smaller holding/backlog costs. To address this trade-off, our algorithm **sets a budget of** s **to pay for the holding costs of aggregating a subset of currently active requests**. The natural choice is to satisfy the requests with earlier deadlines first, as these requests will trigger a service sooner if we do not fulfill them now.

Putting these two phases together, we propose Algorithm 1 for single-item online JRP.

Algorithm 1 : Single-Item Online Joint Replenishment Problem

• **Backlog Phase.** Wait until the sum of the backlog costs of unsatisfied overdue requests becomes s, the service cost. Let t be the time at which this happens. Trigger a service S at time t, and satisfy all of the unsatisfied requests with deadlines no later than t using this service.

• **Holding Phase.** Go over all the remaining active unsatisfied requests in order of their deadlines, and include them one by one in S, as long as the sum of their holding costs at time t is at most s.

2.2 Analysis In this section, we analyze the competitive ratio of Algorithm 1. We start by writing a linear program for single-item Online JRP:

$$P = \min \quad \sum_t s \cdot z_t + \sum_\rho \left[\sum_{a_\rho \le t \le d_\rho} x_{\rho,t} \cdot h \cdot (d_\rho - t) + \sum_{t > d_\rho} x_{\rho,t} \cdot b \cdot (t - d_\rho) \right] \tag{2.1}$$ [SJRP$_\text{P}$]

$$\text{s.t.} \quad \sum_{t \ge a_\rho} x_{\rho,t} = 1, \qquad \forall \rho \tag{2.2}$$

$$x_{\rho,t} \le z_t, \qquad \forall \rho, t \ge a_\rho \tag{2.3}$$

$$z_t, x_{\rho,t} \ge 0, \qquad \forall \rho, t \tag{2.4}$$

In [SJRP$_\text{P}$], z_t represents whether a service is triggered at time t, and $x_{\rho,t}$ determines if request ρ is satisfied at time t. Constraint (2.2) ensures that each request is satisfied eventually, and constraint (2.3) relates the two types of variables, making sure that a request is satisfied at time t only if there is a service at that time. Replacing constraint (2.4) with $z_t, x_{\rho,t} \in \{0, 1\}$ results in an integer program whose optimum value is exactly OPT, the minimum cost incurred by the best offline algorithm. Since [SJRP$_\text{P}$] is a relaxation of this IP, it follows that $P \le OPT$. Here is the dual of [SJRP$_\text{P}$], with variables α_ρ and $\beta_{\rho,t}$ assigned to constraints (2.2) and (2.3), respectively:

$$D = \max \quad \sum_\rho \alpha_\rho \tag{2.5}$$ [SJRP$_\text{D}$]

$$\text{s.t.} \quad \sum_{\rho : a_\rho \le t} \beta_{\rho,t} \le s, \qquad \forall t \tag{2.6}$$

$$\alpha_\rho - \beta_{\rho,t} \le h \cdot (d_\rho - t), \qquad \forall \rho, a_\rho \le t \le d_\rho \tag{2.7}$$

$$\alpha_\rho - \beta_{\rho,t} \le b \cdot (t - d_\rho), \qquad \forall \rho, t > d_\rho \tag{2.8}$$

$$\beta_{\rho,t} \ge 0, \qquad \forall \rho, t \tag{2.9}$$

$$\alpha_\rho \in \mathbb{R}, \qquad \forall \rho \tag{2.10}$$

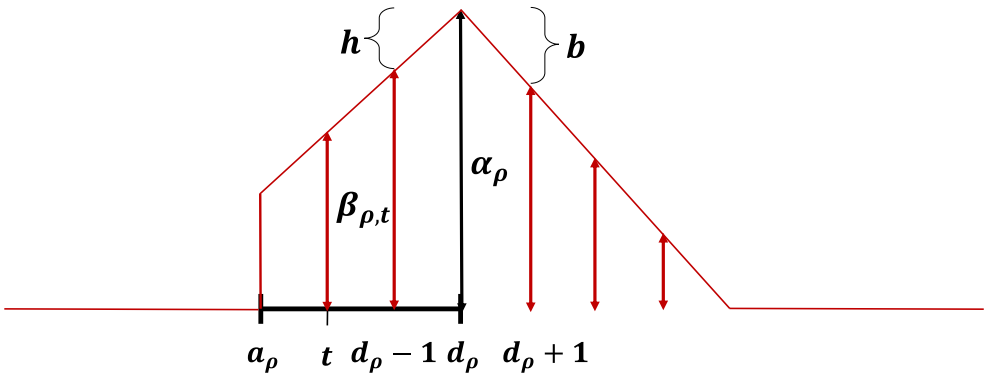
Here we give some intuition about how to approach dual fitting. The goal is to come up with a feasible solution to [SJRP$_\text{D}$] with a large objective function $\sum_\rho \alpha_\rho$. Think of α variables as *money* that we can use to *pay* for the cost of our algorithm. When $\alpha_\rho > 0$ for some request ρ, constraints (2.7) and (2.8) require us to assign nonnegative $\beta_{\rho,t}$ variables for $t \in [\max(a_\rho, d_\rho - \frac{\alpha_\rho}{h}), d_\rho + \frac{\alpha_\rho}{b}]$. So each α variable imposes a range of nonzero β variables. Constraint (2.8) can be rewritten as $\beta_{\rho,t} \ge \alpha_\rho - b \cdot (t - d_\rho)$, which means that at time $t = d_\rho$, we must set $\beta_{\rho,t} = \alpha_\rho$, and as we move past the deadline d_ρ of ρ, the value of $\beta_{\rho,t}$ can be decreased at the rate of b per unit of time, until it becomes 0 at time $t = d_\rho + \frac{\alpha_\rho}{b}$. We call this direction the *forward direction* (of time). In the other direction, which we call the *backward direction*, as we move from d_ρ towards a_ρ, Constraint (2.7) implies that we can decrease $\beta_{\rho,t}$ at the rate of h, until either it hits 0 at time $t = d_\rho - \frac{\alpha_\rho}{h}$, or hits a_ρ, whichever comes first. After that point, its value can stay at zero. Figure 1 illustrates these two cases.

In assigning these $\beta_{\rho,t}$ values as above over several requests ρ, the sum of the β variables is bounded by Constraint (2.6): at each time t, the total *budget* we have for the sum of the β's is at most the service cost s.

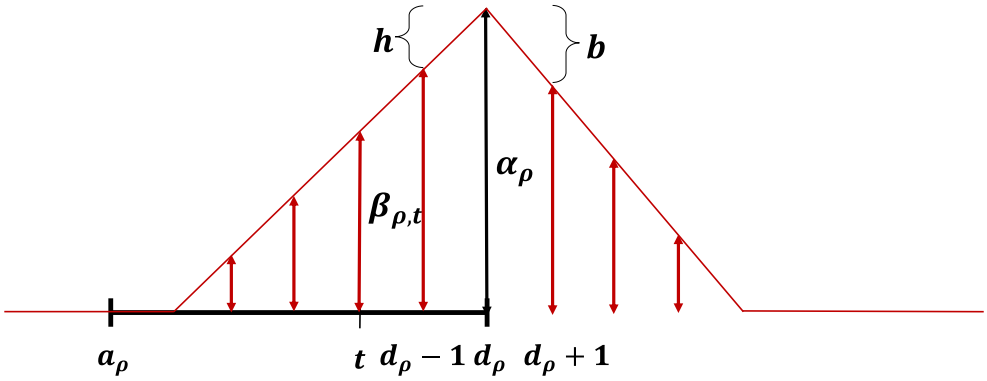
Before describing the dual solution, we need some notation. Assume Algorithm 1 triggers services $S_1, \ldots, S_N$ at times $t_1 < \ldots < t_N$. Note that the algorithm does at most one service at each time step. For service S_i, let B_i be the set of requests that trigger this service, i.e., the set of requests whose backlog costs sum up to s at time t_i and are satisfied in Backlog Phase of Algorithm 1. Also, let H_i be the set of requests with deadlines after t_i that get fulfilled by S_i in Holding Phase of Algorithm 1. By the design of the algorithm, we know $\sum_{\rho \in B_i} b \cdot (t_i - d_\rho) = s$ and $\sum_{\rho \in H_i} h \cdot (d_\rho - t_i) \le s$. For a service S_i, let B_i^1 be the set of requests in B_i that were "alive" at the time of the previous service, i.e., $B_i^1 := \{\rho \in B_i : a_\rho \le t_{i-1}\}$. Let $B_i^2 := B_i \backslash B_i^1$. Since at each of the N services, on top of the service cost, we pay an additional s as backlog costs of the requests that trigger the service in Backlog Phase and at most additional s to aggregate near-time future requests in Holding Phase, we have the following.

LEMMA 2.1. *Algorithm 1 provides a solution to the Single-Item Online JRP problem of cost at most* $3Ns$.

To complete the analysis, we give a feasible solution for [SJRP$_\text{D}$] with an objective value of at least Ns. This would show that offline $OPT \ge P \ge D \ge Ns$, which then shows that the competitive ratio of Algorithm 1 is at most $\frac{3Ns}{Ns} = 3$.



(a)



(b)

Figure 1: Illustration of how the β variables can depreciate in forward and backward directions starting from d_ρ to satisfy constraints (2.7) and (2.8) in [SJRP$_\text{D}$] for some $\alpha_\rho > 0$. Figures 1a and 1b illustrate the cases where $d_\rho - \frac{\alpha_\rho}{h} < a_\rho$ and $d_\rho - \frac{\alpha_\rho}{h} > a_\rho$, respectively.

Even though our LPs have a variable or constraint for every point of time t, it is sufficient to only include the times corresponding to events in the optimal solution. Note that this is only for the sake of analysis since the algorithm itself does not use the LPs.

We begin by outlining the obstacles when developing a suitable dual solution and describe our strategies to address them. The dual solution presented here is also a crucial building block of the analysis of Algorithm 2 for the general Online JRP problem with multiple items described in Section 3.1.

The goal is to assign a subset of requests to each service S_i such that they can pay (accumulate) the cost s using their variables α, while all the associated variables β across these subsets satisfy Constraint (2.6).

Assume for a moment that for each service S_i, all of the requests that trigger this service arrive after the previous service, which has happened at time t_{i-1}, i.e., for each $\rho \in B_i$ we have $a_\rho > t_{i-1}$ (in this case, B_i^1 is empty for each i). Then, for each service S_i and for each $\rho \in B_i$, we set $\alpha_\rho := b \cdot (t_i - d_\rho)$ to be the backlog cost of ρ at time t_i, i.e., the contribution of ρ to triggering S_i. With this assignment, the $\beta_{\rho,t}$ variables deflate to zero in the forward direction at time $d_\rho + \frac{\alpha_\rho}{b} = t_i$. Moreover, in the backward direction, each $\beta_{\rho,t}$ becomes zero at $a_\rho > t_{i-1}$. This is an ideal situation because of the following.

1. The sum of the α_ρ variables for all the requests that trigger S_i is exactly s, as by definition, a service is triggered once the sum of the backlog costs of the active overdue requests becomes s;
2. The β variables associated with paying for a request $\rho \in B_i$ can have nonzero values only in the range $(t_{i-1}, t_i]$, which means that the β variables for different services do not overlap. Since the α values for a service sum to s, this is also the largest sum of the corresponding β values at any time in their interval, ensuring Constraint (2.6) is met.

Thus, in this scenario, a feasible dual solution with an objective value of Ns can be easily constructed.

Now what if a request $\rho \in B_i$ arrives before t_{i-1}, i.e., $\rho \in B_i^1$? We cannot do the same assignment for the dual variables,

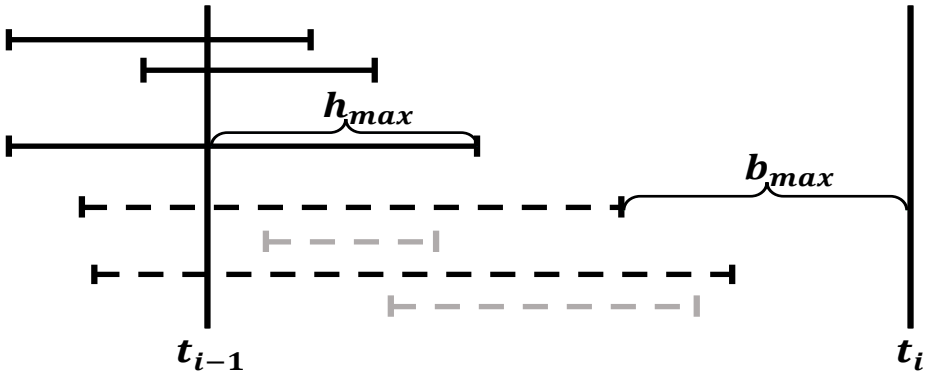


Figure 2: The requests in H_{i-1} (solid), B_i^1 (dashed black), and B_i^2 (dashed grey) for some service S_i. These are the requests that are used to pay for (one-third of) the costs of service S_i. In this figure, $h_{\max}$ is the holding cost of the request in H_{i-1} with the latest deadline, and $b_{\max}$ is the backlog cost of the request in B_i^1 which has the earliest deadline.

as the resulting nonzero β variables for requests in different service sets B_j can 'pile up' on each other at the same time, resulting in a violation of Constraint (2.6). In this case, a key property of Algorithm 1 helps us address this issue: since ρ has arrived before t_{i-1}, this request was a candidate to be served in the Holding Phase of service S_{i-1}. Since the algorithm in the Holding Phase probes the active requests in the order of their deadline, it means that service S_{i-1} has served some other requests H_{i-1} in its Holding Phase with earlier deadlines than ρ. Therefore, **all the requests in H_{i-1} have earlier deadlines than the requests in B_i^1**. See Figure 2 for an illustration.

In this case, in addition to the requests in B_i, we also use the requests in H_{i-1} to pay s. Each request $\rho \in H_{i-1}$ can pay its holding cost that is incurred at time t_{i-1} by setting $\alpha_\rho := h \cdot (d_\rho - t_{i-1})$. In this way, $\beta_{\rho,t}$ diminishes to zero in the backward direction at $d_\rho - \frac{\alpha_\rho}{h} = t_{i-1}$. However, in the forward direction, it might go beyond t_i.

Remember that in order to control the sum of β variables at each time (to satisfy Constraint (2.6)), it is crucial to keep the β variables associated with different services completely disjoint. This means that for each $\rho \in B_i \cup H_{i-1}$ and each $t \notin [t_{i-1}, t_i]$, we want to have $\beta_{\rho,t} = 0$.

To this end, in the dual solution, we compare $h_{\max}$, the holding cost (from time t_{i-1}) of the request in H_{i-1} with the latest deadline, with $b_{\max}$, the backlog cost (at time t_i) of the request in B_i^1 with the earliest deadline (see Figure 2). Intuitively, if $h_{\max} > b_{\max}$, it means that the deadlines of the requests in B_i^1 are "far enough" from t_{i-1} that their β values will deflate to 0 in the backward direction before they reach t_{i-1}, making sure that all the nonzero β's are confined within the range $[t_{i-1}, t_i]$. Similarly, if $h_{\max} < b_{\max}$, it follows that the deadlines of the requests in H_{i-1} are "far enough" from t_i that their β variables will diminish to zero before t_i in the forward direction. Therefore, in the former case, we can use the requests in B_i to pay s, and in the latter case, we can use the requests in H_{i-1} to pay s.

But what if the holding costs of the requests in H_{i-1} are not enough to pay for s? The final observation is that in such case, we can charge the requests in B_i to pay the slack, as their β values in the backward direction definitely deflate to 0 before reaching t_{i-1}. This is because for each $\rho \in B_i^1$, since it was not satisfied at time t_{i-1} despite being active then, there was not enough budget left in Holding Phase of S_{i-1} to include it in H_{i-1}, which in turn means that its deadline has "enough distance" from t_{i-1}.

The Fitted Dual Solution. Putting everything together, here is our proposed dual solution. The proofs of Lemmas 2.2 and 2.3, which show the dual solution has an objective value of Ns and it is feasible, can be found in the full version of the paper.

For each $i = 1, \ldots, N$ do the following. Let $h_{\max}$ be the holding cost (from time t_{i-1}) of the request in H_{i-1} which has the latest deadline, and $b_{\max}$ be the backlog cost (at time t_i) of the request in B_i^1 which has the earliest deadline (see Figure 2). Set $h_{\max} := 0$ if $H_{i-1} = \emptyset$ and $b_{\max} := 0$ if $B_i^1 = \emptyset$. For ease of notation, set $H_0 = \emptyset$. There are two cases:

- **Case 1.** $h_{\max} > b_{\max}$: set $\alpha_\rho = b \cdot (t_i - d_\rho)$ for each $\rho \in B_i$, and set $\alpha_\rho = 0$ for each $\rho \in H_{i-1}$.

- **Case 2.** $h_{\max} \leq b_{\max}$: set $\alpha_\rho = h \cdot (d_\rho - t_{i-1})$ for each $\rho \in H_{i-1}$, and set $\alpha_\rho = \frac{s - h_{\text{sum}}}{s} \cdot b \cdot (t_i - d_\rho)$ for each $\rho \in B_i$, where $h_{\text{sum}} := \sum_{\rho \in H_{i-1}} h \cdot (d_\rho - t_{i-1})$ is the sum of the holding costs of requests in H_{i-1} at time t_{i-1}.

For each ρ, set $\beta_{\rho,t}$ as follows (see Figure 1):

$$\beta_{\rho,t} = \begin{cases} \max(0, \alpha_\rho - h \cdot (d_\rho - t)) & \text{for } a_\rho \le t \le d_\rho \\ \max(0, \alpha_\rho - b \cdot (t - d_\rho)) & \text{for } t > d_\rho. \end{cases}$$

LEMMA 2.2. *In the proposed dual solution, for each $i = 1, \ldots, N$, we have $\sum_{\rho \in B_i \cup H_{i-1}} \alpha_\rho = s$. This shows that the dual objective function is Ns.*

LEMMA 2.3. *The above dual solution is feasible for* [SJRP_D].

Proof. [Proof of Theorem 1.1] Follows from Lemmas 2.1, 2.2, and 2.3. ∎

REMARK 2.1. *The analysis of the Algorithm 1 is tight, i.e., there exists an instance of the problem for which the algorithm performs 3 times worse than the optimal offline algorithm: assume $b = h = 1$ and all the arrival times are 0. Assume there are s deadlines at time 0, and for each $k \ge 1$, there are $2s$ deadlines at time $2k$. Algorithm 1 makes services at times $1, 3, \ldots$ and pays $3s$ for each service. But one can do services at times $0, 2, \ldots$. and pay s for each service (no holding or backlog cost incurred).*

REMARK 2.2. *In the case where no backlog is allowed, i.e. $b = \infty$, the same analysis shows that Algorithm 1 is 2-competitive. Note that in this case, the algorithm waits until one of the requests becomes due, and then triggers a service. After that, the algorithm goes over the active requests in the order of their deadlines and includes them in the current service until their total holding cost reaches s.*

3 Online JRP with Multiple Items

In this section, we describe an algorithm for (multiple-item) Online JRP. In Online JRP, each service includes a subset $U \subseteq \{v_1, \ldots, v_n\}$ of the items, and the cost of the service is the sum of the joint service cost $c(r)$, the item costs $c(U) = \sum_{u \in U} c(u)$, and the holding and backlog costs of the requests fulfilled using this service.

In the case where $c(r) > \sum_{i=1}^{n} c(v_i)$, the problem reduces to single-item JRP with service cost $c(r)$ as the sum of the joint service cost and item costs for each service is within a constant factor of $c(r)$. On the other hand, if for some item v we have $c(r) < c(v)$, we can decouple v from the other items and treat it as a separate single-item problem, which results in the loss of only a constant factor. While we make no assumptions, the above arguments show that the hard case is where $c(r) > c(v_i)$ for each i and $\sum_{i=1}^{n} c(v_i)$ is much larger than $c(r)$. Critically, in the multi-item case, the algorithm will need to justify paying both the joint service cost and the cost of each individual item that is included in a service.

3.1 Algorithm

We start with an intuitive description of the algorithm and then give a formal definition.

Triggering a Service: Using a greedy dual-growing approach, each item type v_i is represented by a cup with a target volume $c(v_i)$ which is "filled" over time with backlog costs of requests for this item. Once an item's cup is full, the algorithm decides to include that item in the next service. We call such items *mature*. The algorithm additionally accumulates *surplus* backlog cost, which can be thought of as the "overflow" of the item cups into another cup with volume $c(r)$ which corresponds to the joint service cost. Once this joint cup is filled in a "mature" backlog accumulation phase, the algorithm *triggers* an actual service and includes all the mature items in the service, which we denote by S. The algorithm naturally serves all the overdue requests for all items in S since otherwise a larger backlog cost will be incurred to serve them in the future.

Selecting Additional Items to Serve: It is not sufficient to include only the mature items as there are instances where the algorithm incurs large costs from requests for several items that are just about to become mature (see [10]). To overcome such instances, the algorithm also serves some of the *premature* items; that is, the items that have not accumulated enough backlog cost to fill up their cups. Since we are adding premature items based on their backlog, this adds a second "premature" backlog phase to the algorithm. In this phase, the algorithm sets a budget of $\Theta(c(r))$ and "buys" premature items one by one in the order of the first time they are going to be mature assuming no new request arrives for them. It is worth mentioning that the algorithm presented in [10] for make-to-order JRP has a similar step called the "simulation step".

Selecting Additional Requests to Serve: The algorithm considers each item $v_i \in S$ and performs a "local" holding phase similar to Algorithm 1 with a budget of $c(v_i)$. As explained later in Section 3.4, this helps us accumulate the $c(v_i)$ cost using dual variables. Moreover, by the same logic, to gather the joint service cost $c(r)$ in a feasible dual, another "global" holding phase is needed jointly among all the items in the service. In this "global" holding phase, the algorithm sets a budget of $c(r)$

and goes over all the active requests for the items in S in the order of their deadlines and fulfills them by paying their holding cost.

Formally, we propose Algorithm 2 for Online JRP. Even though the algorithm uses a continuous notion of time for illustrative convenience, it is not hard to implement it in time polynomial in the number of requests since all relevant times of maturity are polynomially bounded by the number of requests.

Algorithm 2 : Multiple-item Online Joint Replenishment Problem

Start with time $t = 0$ and continuously increase t. At each time t:

• Let $\mathcal{A}_t$ be the set of all the *active* requests at time t, i.e., the requests that have arrived before or at t and are not satisfied by any of the previous services.

• Assign a variable $b_t(v)$ to each item v, which indicates the amount of backlog accumulated by the unsatisfied overdue requests for v at time t, i.e., $b_t(v) = \sum_{\rho \in \mathcal{A}_t : v_\rho = v,\ d_\rho < t} b \cdot (t - d_\rho)$. We call an item v *mature*, if $b_t(v) \geq c(v)$. Any backlog cost that is accumulated after v becomes mature is called *surplus backlog cost*.

• Assign a variable $b_t(r)$ to the root, which indicates the sum of the surplus backlog costs for the mature items at time t, i.e., $b_t(r) = \sum_{v : b_t(v) \geq c(v)} (b_t(v) - c(v))$.

• **Mature Backlog Phase** If $b_t(r) = c(r)$, i.e., the surplus backlog cost accumulated equals the joint service cost, trigger a service S at time t. Include all the mature items v in S, satisfy all their overdue requests using S, and remove them from $\mathcal{A}_t$.

• **Premature Backlog Phase.** For each *premature* item v that has at least one unsatisfied request, compute the first time $\text{mature}_t(v)$ it is going to be mature, assuming no new request shows up, i.e., $\text{mature}_t(v)$ is a time t' such that $\sum_{\rho \in \mathcal{A}_t : v_\rho = v,\ d_\rho < t'} b \cdot (t' - d_\rho) = c(v)$. Sort all of the premature items in non-decreasing order of $\text{mature}_t(v)$, and include them in S one by one, as long as the sum of their item costs $c(v)$ is at most $2c(r)$. For each newly included item, satisfy all of their overdue requests using S, and remove them from $\mathcal{A}_t$.

• **Local Holding Phase.** For each item v included in S, iterate over the remaining active requests for v in non-decreasing order of their deadlines, and satisfy them one by one using S, as long as the sum of their holding cost is at most $c(v)$ (Note that since all the overdue requests in v are already satisfied using one of the backlog phases, all the remaining active requests in v have deadlines in the future).

• **Global Holding Phase.** Go over the remaining active requests for items included in S in non-decreasing order of their deadlines, and satisfy them one by one using S, as long as the sum of their holding cost is at most $c(r)$.

3.2 Primal and Dual Problems

We use dual-fitting to analyze the algorithm. First, we start by writing a linear program and its dual for Online JRP.

$$P = \min \sum_{v \in \{r, v_1, \ldots, v_n\}} \sum_t c(v) \cdot z_{v,t} \qquad [\text{JRP}_\text{P}] \tag{3.11}$$

$$+ \sum_\rho \left[\sum_{a_\rho \le t \le d_\rho} x_{\rho,t} \cdot h \cdot (d_\rho - t) + \sum_{t > d_\rho} x_{\rho,t} \cdot b \cdot (t - d_\rho) \right] \tag{3.12}$$

$$\text{s.t.} \sum_{t \ge a_\rho} x_{\rho,t} = 1, \qquad \forall \rho \tag{3.13}$$

$$x_{\rho,t} \le z_{v_\rho,t}, \qquad \forall \rho, t \ge a_\rho \tag{3.14}$$

$$z_{v_i,t} \le z_{r,t}, \qquad \forall t, 1 \le i \le n \tag{3.15}$$

$$x_{\rho,t} \ge 0, \qquad \forall \rho, t \tag{3.16}$$

$$z_{v,t} \ge 0, \qquad \forall v, t \tag{3.17}$$

In [JRP$_\text{P}$], $z_{v,t}$ represents whether a service is made at time t that includes node v, and $x_{\rho,t}$ determines if request ρ is satisfied at time t. Constraint (3.13) ensures that each request is eventually satisfied, and Constraint (3.14) relates the two types of variables, ensuring that a request is satisfied at time t only if a service is provided at that time. Constraint (3.15) guarantees that the root is included in all services, i.e., the joint service cost is paid for each service that is triggered. If the variables are constrained to be binary, the resulting Integer Program will be an exact formulation for the problem. Thus [JRP$_\text{P}$] is a relaxation of the original problem, and it follows that $P \le OPT$.

The dual of [JRP$_\text{P}$] has variables α_ρ, $\beta_{\rho,t}$, and $\gamma_{v_i,t}$ assigned to the constraints (3.13),(3.14), and (3.15), respectively:

$$D = \max \sum_\rho \alpha_\rho \qquad [\text{JRP}_\text{D}] \tag{3.18}$$

$$\text{s.t.} \sum_{\rho : a_\rho \le t, v_\rho = v_i} \beta_{\rho,t} - \gamma_{v_i,t} \le c(v_i), \qquad \forall t, 1 \le i \le n \tag{3.19}$$

$$\sum_{i=1}^{n} \gamma_{v_i,t} \le c(r), \qquad \forall t \tag{3.20}$$

$$\alpha_\rho - \beta_{\rho,t} \le h \cdot (d_\rho - t), \qquad \forall \rho, a_\rho \le t \le d_\rho \tag{3.21}$$

$$\alpha_\rho - \beta_{\rho,t} \le b \cdot (t - d_\rho), \qquad \forall \rho, t > d_\rho \tag{3.22}$$

$$\beta_{\rho,t} \ge 0, \qquad \forall \rho, t \tag{3.23}$$

$$\gamma_{v_i,t} \ge 0, \qquad \forall t, 1 \le i \le n \tag{3.24}$$

In [JRP$_\text{D}$], variables $\beta_{\rho,t}$ are only defined for $t \ge a_\rho$. In the analysis, for simplicity, we define $\beta_{\rho,t}$ for all values of t and set $\beta_{\rho,t}$ to be 0 for each $t < a_\rho$.

Here we give some intuition about the dual problem. The goal is to find a feasible solution to [JRP$_\text{D}$] with a large objective function $\sum_\rho \alpha_\rho$. As in the single-item case, we think of α variables as *money* that can be used to *pay* for the cost of our algorithm. When α_ρ is increased for some request ρ, Constraints (3.21) and (3.22) force a feasible solution to have non-negative $\beta_{\rho,t}$ variables for $t \in [\max(a_\rho, d_\rho - \frac{\alpha_\rho}{h}), d_\rho + \frac{\alpha_\rho}{b}]$. Each nonzero α variable thus imposes a range of nonzero β variables. The sum of the β variables for requests for each item v_i is bounded by Constraint (3.19). Assume that the γ variables are all zero: then, Constraint (3.19) insists that at each time t, the total amount of *budget* that we have for the sum of the β's for requests ρ associated with item v_i is at most the cost of the item $c(v_i)$. This can be thought of as a *local budget* that we have for each item per unit of time. If the cost incurred by α variables is beyond this local budget, then the variables γ must be used, which represent a *global budget* that is shared between all items. Constraint (3.20) implies that at each time t, the total amount of the global budget is at most the cost of the joint service $c(r)$.

3.3 Notation Let $S_1, \ldots, S_N$ be the services that our algorithm triggers, and assume that they happen at times $t_1 \le \ldots \le t_N$. Let $V_{i,1}$ and $V_{i,2}$ be the set of items included in service S_i in Mature Backlog Phase and Premature Backlog Phase, respectively, and define $V_i := V_{i,1} \cup V_{i,2}$. Let $V_{i,A}$ be the set of items that have at least one active[2]

[2]Recall that an active request is one that has arrived before the current time and is as yet unsatisfied.

request immediately after the Mature Backlog Phase. Note that $V_{i,2} \subseteq V_{i,A}$. For ease of notation, we define $t_0 := 0$ and $V_0 = V_{0,A} := \emptyset$.

Let $B_{i,1}$, $H_{i,1}$, and $H_{i,2}$ be the set of requests satisfied using S_i in Mature Backlog Phase, Local Holding Phase, and Global Holding Phase, respectively. For each item v, the requests for that item satisfied in these three phases are denoted by $B^v_{i,1}$, $H^v_{i,1}$, and $H^v_{i,2}$, respectively. For each item $v \in V_{i,2}$, let $B^v_{i,2}$ be the set of requests for v that contribute to making v mature at time $\text{mature}_{t_i}(v)$, i.e., the set of active requests ρ at the beginning of Premature Backlog phase such that $a_\rho \le t_i$, $d_\rho < \text{mature}_{t_i}(v)$, and $v_\rho = v$. Let $B_{i,2} := \bigcup_{v \in V_{i,2}} B^v_{i,2}$. Note that in Premature Backlog Phase, only the requests in $B_{i,2}$ that have deadlines before t_i are satisfied. So $S_i \subseteq B_{i,1} \cup B_{i,2} \cup H_{i,1} \cup H_{i,2}$ (we abuse the notation here and use S_i to refer to both the i'th service and the set of requests included in the i'th service). For a subset U of vertices in the tree, define $c(U) := \sum_{u \in U} c(u)$.

Consider a service S_i and an item $v \in V_{i,1}$. The item v is included in the Phase I backlog of service S_i, so it has matured at some time $t \le t_i$. This implies that the sum of the backlog costs of the active requests for v at time t is at least $c(v)$. Since active requests for v at time t are all included in $B^v_{i,1}$, and $t \le t_i$, we conclude that $\sum_{\rho \in B^v_{i,1}} b \cdot (t_i - d_\rho) \ge c(v)$. Let $\rho_1, \dots, \rho_m$ be the requests in $B^v_{i,1}$ in increasing order of arrival time. Let k^* be the smallest k such that $\sum_{l=1}^{k} b \cdot (t_i - d_{\rho_l}) \ge c(v)$. Define $L^v_i := \{\rho_1, \dots, \rho_{k^*}\}$. If $\sum_{l=1}^{k^*} b \cdot (t_i - d_{\rho_l}) = c(v)$, define $R^v_i := \{\rho_{k^*+1}, \dots, \rho_m\}$; otherwise define $R^v_i := \{\rho_{k^*}, \dots, \rho_m\}$. Note that $L^v_i \cup R^v_i = B^v_{i,1}$ and $L^v_i \cap R^v_i$ is either $\emptyset$ or $\{\rho_{k^*}\}$. In the latter case, the item becomes mature in the middle of processing the dual value for request ρ_{k^*} in this order.

Just to repeat, similar to the single-item case, when $\alpha_\rho > 0$ for some request ρ, constraints (3.21) and (3.22) require us to assign nonnegative $\beta_{\rho,t}$ variables for $t \in [\max(a_\rho, d_\rho - \frac{\alpha_\rho}{h}), d_\rho + \frac{\alpha_\rho}{b}]$; see Figure 1. As we move from time d_ρ to later or earlier times, we can decrease the value of $\beta_{\rho,t}$. We call the former *forward direction*, and the latter *backward direction* (of time).

3.4 Challenges and a High-Level Description of the Dual Solution In each service S_i, all costs are within a constant factor of $c(V_{i,1}) + c(r)$; see Lemma 3.7. In general, neither of these terms dominates the other. Therefore, we will fit a dual solution that accumulates both $\Theta(c(V_1))$ and $\Theta(c(r))$.

In the single-item case, to pay s for each service S_i, the requests in $H = H_{i-1}$ and $B = B_i$ are used. There are 3 ingredients that are utilized in the dual solution for the single-item case:

(I1) Sum of the backlog costs of the requests in B at time t_i is s.

(I2) The active requests at t_{i-1} with future deadlines are added to H in order of their deadline and the budget for their holding costs is s.

(I3) Each request in B that has arrived before t_{i-1} is a candidate in the holding phase of S_{i-1}.

In the dual solution presented in Section 3.5 for $[\text{JRP}_\text{D}]$, we use a similar dual solution to the single-item case to partially pay the costs of Algorithm 2. In fact, all the cost $c(V_{i,1})$ for each service S_i is paid in this way, since all 3 ingredients needed are present. Assume $v \in V_{i,1}$ and let S_ℓ be the last service before S_i that includes v. Let $B := B^v_{i,1}$ and $H := H^v_{\ell,1}$. Item v is included in $V_{i,1}$ because the sum of backlog accumulated for it is at least $c(v)$, i.e., the sum of the backlog costs of the request in B at time t_i is $c(v)$ (Ingredient **(I1)**). Moreover, in the Local Holding Phase of S_ℓ, Algorithm 2 sets a budget of $c(v)$ to go over the active requests for v with deadlines in the future and serves them using S_ℓ in the order of their deadlines (Ingredient **(I2)**). Lastly, each request in B that has arrived before t_ℓ is a candidate in the Local Holding Phase of S_ℓ for item v (Ingredient **(I3)**). We can accumulate a dual value of $c(v)$ using a similar approach to the dual assignment for the single-item case. This is done in step **D.1** of the dual assignment in Section 3.5 using Lemma 3.1. We call this procedure **local charging**, as it is used to recover the item costs (which can be thought of as the local costs of the algorithm) only using the local budget and the β variables. In particular, the γ variables in $[\text{JRP}_\text{D}]$ are not used in the local chargings.

Paying $c(r)$ is more nuanced. At first glance, it might seem that for two consecutive services S_{i-1} and S_i, we can follow the same structure and set B to be the set of requests that caused the surplus backlog of $c(r)$ in S_i, and set $H := H_{i-1,2}$, the set of requests served in the global holding phase of S_{i-1}. In fact, with this definition of B and H, we have ingredients **(I1)** and **(I2)**; but **(I3)** might be missing. This is because the set of *items* included in the services S_{i-1} and S_i are not necessarily the same, i.e., there might be a request $\rho \in B$ that has arrived before t_{i-1} but its item v_ρ is not included in the service S_{i-1}[3].

[3]This difficulty does not arise in the single item case since all requests are for the same item and they are considered in the previous service if they were active.

This means the request was not considered in the Global Holding Phase of S_{i-1}. However, this shows us that we can pay for the surplus backlog costs of the requests that were among the candidates in Global Holding Phase of S_{i-1}, i.e., the requests for the items in $V_{i,1} \cap V_{i-1}$ (This is done in step **D.2.2.2** of the dual assignment using Lemma 3.2).

To recover the remainder of the surplus backlog cost, namely for the items in $V_{i,1} \setminus V_{i-1}$, we need a new idea. This is where the Premature Backlog Phase of Algorithm 2 is exploited.

In the Premature Backlog Phase of service S_{i-1}, for each premature item v, the algorithm computes the first time $\text{mature}_{t_{i-1}}(v)$ that v becomes mature using only active requests at time t_{i-1}. Suppose $\text{mature}_{t_{i-1}}(v_1) \leq \ldots \leq \text{mature}_{t_{i-1}}(v_m)$. The algorithm iterates over the nodes $v_1, \ldots, v_m$, in this order, and includes them one by one in service S_{i-1} by paying their item costs $c(v)$, as long as the sum of the item costs is at most $2c(r)$. Assume that the algorithm includes items $V_{i-1,2} = \{v_1, \ldots, v_k\}$ in the Premature Backlog Phase of service S_{i-1}, and for simplicity, assume $\sum_{j=1}^{k} c(v_j) = \Theta(c(r))$.

There are two cases to consider:

- $t_i \geq \text{mature}_{t_{i-1}}(v_{k+1})$. In this case, for each item $v \in \{v_1, \ldots, v_k\}$, using Lemma 3.1, we pay for $c(v_i)$. This allows us to pay $\sum_{j=1}^{k} c(v_j) = \Theta(c(r))$. The only difference between this case and the local charging is that we set B to be equal to the set of requests that make v mature at time $\text{mature}_{t_{i-1}}(v)$. We set $H = H^v_{\ell,1}$ where ℓ is the last service before $i-1$ that includes v. This case corresponds to step **D.2.1** in the dual assignment below. The fact that $\text{mature}_{t_{i-1}}(v) \leq \text{mature}_{t_{i-1}}(v_{k+1}) \leq t_i$ ensures that the non-zero β variables set in Lemma 3.1 are restricted within $[t_\ell, t_i]$. This prevents the β variables used to pay for different services from accumulating at some fixed time.

  However, there is one important detail that we need to be careful about: some of the requests that contribute to v becoming mature at time $\text{mature}_{t_{i-1}}(v)$ have deadlines after t_{i-1}, which means that they are not served using S_{i-1} (and they might still be active at time t_i). This means that we are charging a request at time t_{i-1} that has not yet been served. To address this issue, we show that each request is charged at most *twice* during the local chargings in the construction of the dual solution.

- $t_i < \text{mature}_{t_{i-1}}(v_{k+1})$. Note that we only need to pay for the surplus backlog cost of the items in $V_{i,1} \setminus V_{i-1}$, which does not include any of the items in $V_{i-1,2} = \{v_1, \ldots, v_k\}$ since $V_{i-1,2} \subseteq V_{i-1}$. Let $v \in V_{i,1} \setminus V_{i-1}$. We show that all the requests that contribute to the surplus backlog cost of v have arrived after time t_{i-1} (Lemma 3.3). Consider setting α_ρ for each such request to be equal to the backlog cost of ρ at time t_i. In this case, we know that the corresponding β variables can only have nonzero values in $[t_{i-1}, t_i]$. Lemma 3.3 holds because either v does not have any active requests at time t_{i-1}, or v is in $\{v_{k+1}, \ldots, v_m\}$. This means that the active requests at time t_{i-1} can only accumulate enough backlog cost to make v mature at $\text{mature}_{t_{i-1}}(v_{k+1}) > t_i$. Thus, all the requests responsible for the surplus backlog cost of v at time t_i have arrived after t_{i-1}. This case corresponds to **D.2.2.1** in the dual assignment.

Finally, we mention one other detail that needs to be taken into account. In the above, for an item $v \in V_{i,1}$, we talk about requests that contribute to the surplus backlog cost of v at time t_i. Here we implicitly assume that the set of requests in $B^v_{i,1}$ is naturally partitioned into two sets: one set consisting of the earlier requests (in terms of arrival times) that make v mature, and another set comprised of the later requests that are responsible for the surplus backlog cost. However, in reality, a request can contribute to both the backlog cost that makes v mature and the surplus backlog cost. To partition the requests into two sets having the aforementioned properties, we "artificially" partition the set $B^v_{i,1}$ into two sets L^v_i and R^v_i which play the roles of the two sets described above. Note that this partitioning of the requests in $B^v_{i,1}$ is crucial as the charging used in **D.2.2.1** can only be done on the requests in R^v_i which we know have arrived after t_{i-1}.

Next we formally describe the dual solution.

3.5 Dual Solution Our proposed dual solution is constructed in a modular way. First, we describe the building blocks of the dual solution. The (partial) dual solutions presented in the proofs of Lemmas 3.1 and 3.2 are very similar to our proposed dual solution for the single-item case. The proofs can be found in the full version of our paper.

LEMMA 3.1. (LOCAL CHARGING LEMMA FOR v AND S_i) *Let S_i be a service that contains item v, and let S_ℓ be the last service before S_i that includes v. Define B and t^* as follows:*

- *Case 1 (mature items): $v \in V_{i,1}$. Define $B := L^v_i$, and $t^* := t_i$.*
- *Case 2 (premature items): $v \in V_{i,2}$. Define $B := B^v_{i,2}$ and let t^* be the time $\text{mature}_{t_i}(v)$ defined in Premature Backlog Phase of service S_i in Algorithm 2*

Also, let $H := H^v_{\ell,1}$ be the set of requests for item v that are satisfied in the Local Holding Phase of service S_ℓ. If S_i is the first service that includes v, set $t_\ell := 0$ and $H := \emptyset$. There is an assignment for the variables $\alpha_\rho, \beta_{\rho,t}$ for each $\rho \in B \cup H$ such that:

(I) $\sum_{\rho \in B \cup H} \alpha_\rho = c(v)$,

(II) $\beta_{\rho,t} = 0$ *for each* $\rho \in B \cup H$ *and* $t \notin (t_\ell, t^*)$.

(III) $\beta_{\rho,t} \le \alpha_\rho$ *for each* $\rho \in B \cup H$ *and each time* t,

(IV) α_ρ *and* $\beta_{\rho,t}$ *for* $\rho \in B \cup H$ *satisfy constraints* (3.21), (3.22), *and* (3.23).

(V) *In Case 1,* $\alpha_{\rho^*} \le c(v) - \sum_{\rho \in L^v_i \setminus \{\rho^*\}} b \cdot (t_i - d_\rho)$, *where* ρ^* *is the request in* L^v_i *with the largest arrival time.*

LEMMA 3.2. (TWO-SIDED GLOBAL CHARGE FOR S_i) *Let S_i be one of the services triggered by Algorithm 2, $i = 2, \ldots, N$, and let S_{i-1} be its previous service. Let $H := H_{i-1,2}$ and $B := \bigcup_{v \in V_{i,1} \cap V_{i-1}} R^v_i$. Then there is an assignment for the variables α_ρ, $\beta_{\rho,t}$ and $\gamma_{v,t}$ for each $\rho \in H \cup B$ and $v \in V_{i,1} \cap V_{i-1}$ such that:*

(I) $\sum_{\rho \in B \cup H} \alpha_\rho \ge \sum_{v \in V_{i,1} \cap V_{i-1}} \left(\sum_{\rho \in B^v_{i,1}} b \cdot (t_i - d_\rho) - c(v) \right)$.

(II) $\beta_{\rho,t} = 0$ *and* $\gamma_{v,t} = 0$ *for each* $\rho \in B \cup H$, $v \in V_{i,1} \cap V_{i-1}$, *and* $t \notin (t_{i-1}, t_i)$.

(III) $\beta_{\rho,t} \le \alpha_\rho$ *for each* $\rho \in B \cup H$ *and each time* t,

(IV) α_ρ, $\beta_{\rho,t}$ *and* $\gamma_{v,t}$ *for each* $\rho \in B \cup H$ *and* $v \in V_{i,1} \cap V_{i-1}$ *satisfy all the constraints in* [*JRP*$_D$].

(V) $\sum_{\rho : a_\rho \le t, v_\rho = v} \beta_{\rho,t} = \gamma_{v,t}$, *for each* $v \in V_{i,1} \cap V_{i-1}$ *and each time* t.

We build the dual solution using the following three routines.

DEFINITION 3.1. (LOCALCHARGE(v, S_i)) *Assume S_i is a service that includes item v. Do the dual assignment for item v and service S_i described in Lemma 3.1, and denote the variables by α^* and β^*. For each request ρ and time t, increase the current values of α_ρ and $\beta_{\rho,t}$ by $\frac{1}{4}\alpha^*_\rho$ and $\frac{1}{4}\beta^*_{\rho,t}$, respectively.*

DEFINITION 3.2. (UNIQUEGLOBALCHARGE(ρ, S_i)) *Assume ρ is a request in $B_{i,1}$. Let α^0_ρ be the current value of α_ρ, and define $\Delta := b \cdot (t_i - d_\rho) - \alpha^0_\rho$. Increase α_ρ by Δ, and for each time $a_\rho \le t \le t_i$, increase $\beta_{\rho,t}$ and $\gamma_{v_\rho,t}$ by Δ.*

DEFINITION 3.3. (COMMONGLOBALCHARGE(S_i)) *Let α^*, β^* and γ^* be the dual assignment of Lemma 3.2 for service S_i, where $i = 2, \ldots, N$. For each item v, request ρ, and time t, increase the values of α_ρ, $\beta_{\rho,t}$, and $\gamma_{v,t}$ by $\frac{1}{2}\alpha^*_\rho$, $\frac{1}{2}\beta^*_{\rho,t}$, and $\frac{1}{2}\gamma^*_{v,t}$, respectively.*

Now we are ready to describe the dual solution.

Dual Assignment. Consider a service S_i for $i = 2, \ldots, N$. Without loss of generality, assume after doing Mature Backlog Phase for S_{i-1}, the set of premature items that have at least one active request is $V_{i-1,A} = \{v_1, \ldots, v_m\}$, and $\text{mature}_{t_{i-1}}(v_1) \le \ldots \le \text{mature}_{t_{i-1}}(v_m)$. Suppose in Premature Backlog Phase for S_{i-1}, the items $v_1, \ldots, v_k$ are included, i.e., $V_{i-1,2} = \{v_1, \ldots, v_k\}$. So either these are all the items with active requests, i.e., $m = k$, or we did not have enough budget to include v_{k+1} in the service, i.e., $\sum_{j=1}^{k+1} c(v_j) > 2c(r)$. In the former case, define $\text{mature}_{t_{i-1}}(v_{k+1}) := \infty$.

Here is how we construct the dual solution for [JRP$_D$]. Initially, set all the dual variables to zero. For each $i = 1, \ldots, N$, do the following (note that $V_{0,A} = \emptyset$):

Dual Assignment i

D.1 For each $v \in V_{i,1}$ do LocalCharge(v, S_i).

D.2 There are two cases:

D.2.1 Case I: $V_{i-1,A} \neq \emptyset$ and $t_i > \text{mature}_{t_{i-1}}(v_{k+1})$.
For each node $v \in V_{i-1,2}$, do LocalCharge(v, S_{i-1}).

D.2.2 Case II: $V_{i-1,A} = \emptyset$ or $t_i \le \text{mature}_{t_{i-1}}(v_{k+1})$.

Define $R_i^1 := \bigcup_{v \in V_{i,1} \setminus V_{i-1}} R_i^v$ and $R_i^2 := \bigcup_{v \in V_{i,1} \cap V_{i-1}} R_i^v$.

D.2.2.1 For each $\rho \in R_i^1$, do UniqueGlobalCharge(ρ, S_i).

D.2.2.2 If $R_i^2 \neq \emptyset$, do CommonGlobalCharge(S_i).

D.2.2.3 If the total increase of the α variables in steps **D.2.2.1** and **D.2.2.2** is more than $c(r)$, scale down all the increases in these steps (for all the variables involved) so that the total increase of the α variables becomes exactly $c(r)$.

Note that for $i = 1$, since $V_{i-1,A} = V_{i-1} = \emptyset$, only steps **D.1**, **D.2.2.1** and **D.2.2.3** are called, where $R_i^1 = \bigcup_{v \in V_{1,1}} R_1^v$. Intuitively, Dual Assignment i is responsible for paying a constant fraction of the costs of service S_i.

Before calculating the objective value of the above dual solution and proving its feasibility, we prove the following key structural lemma, which is used in the next two sections.

LEMMA 3.3. *All the requests that are involved in step **D.2.2.1** of Dual Assignment i, i.e., the requests in $R_i^1 = \bigcup_{v \in V_{i,1} \setminus V_{i-1}} R_i^v$, arrive after t_{i-1}.*

Proof. In step **D.2** of the Dual Assignment i, Case II happens when $V_{i-1,A} = \emptyset$ or $t_i < \text{mature}_{t_{i-1}}(v_{k+1})$. Recall that $V_{i-1,A} = \{v_1, \ldots, v_m\}$ is the set of all the items that have at least one active request at time t_{i-1} and are not included in Mature Backlog Phase in service S_{i-1}. The times at which these items are going to mature, considering only their active requests at time t_{i-1} immediately after Mature Backlog Phase, are $\text{mature}_{t_{i-1}}(v_1) \leq \ldots \leq \text{mature}_{t_{i-1}}(v_m)$. Among these items, $V_{i-1,2} = \{v_1, \ldots, v_k\}$ are included in Premature Backlog Phase for service S_{i-1}. The requests involved in **D.2.2.1** are the requests in $R_i^1 = \bigcup_{v \in V_{i,1} \setminus V_{i-1}} R_i^v$. Since $\{v_1, \ldots, v_k\} = V_{i-1,2} \subseteq V_{i-1}$, none of the items $\{v_1, \ldots, v_k\}$ are in R_i^1.

Let $v \in V_{i,1} \setminus V_{i-1}$ be one of the items involved in R_i^1. We want to show that all the requests in R_i^v have arrived after t_{i-1}. There are two cases:

- $v \in \{v_{k+1}, \ldots, v_m\}$. Since $t_i < \text{mature}_{t_{i-1}}(v_{k+1})$, we conclude that the sum of the backlog costs of the requests for v that are active at time t_{i-1} is less than $c(v)$ at time t_i, i.e., these requests are not enough to make v mature at time t_i. In Mature Backlog Phase of service S_i, only mature items are included. So since $v \in V_{i,1}$, i.e., it was chosen in Mature Backlog Phase of service S_i, it is impossible for all the requests in L_i^v to have arrived before t_{i-1} (recall that the requests in L_i^v, by definition, can make v mature at time t_i). In particular, it implies that the request with the latest arrival time in L_i^v has arrived after t_{i-1}, which means that all the requests in R_i^v have arrived after t_{i-1}.

- $v \notin V_{i,A}$. This means that v does not have any active requests at time t_{i-1} after service S_{i-1}. Therefore all the requests for item v that are served during Mature Backlog Phase of service S_i, i.e., $B_{i,1}^v$, have arrived after t_{i-1}. In particular, all the requests in $R_i^v \subseteq B_{i,1}^v$ have arrived after t_{i-1}.

Note that since all the requests arrive after time 0, for $i = 1$ we have that all the requests in R_i^1 arrive after $t_{i-1} = 0$. ∎

3.6 Dual Objective Value In this section, we calculate the objective value of the dual solution described in Section 3.5, and compare it to the cost of Algorithm 2.

LEMMA 3.4. *The total increase of the α variables if Step **D.2.1** of Dual Assignment i is invoked is at least $c(r)/2$.*

Proof. When this case happens, it means that v_{k+1} has some active requests at time t_{i-1}, i.e., $\text{mature}_{t_{i-1}}(v_{k+1}) \neq \infty$. But we have not included v_{k+1} in $V_{i-1,2}$, which means that we could not afford to pay for $c(v_{k+1})$ in the Premature Backlog Phase of service S_{i-1}. So $\sum_{\ell=1}^{k+1} c(v_\ell) > 2c(r)$. On the other hand, $\text{mature}_{t_{i-1}}(v_{k+1})$ is the time at which v_{k+1} becomes mature using only the backlog costs of the requests that are active at time t_{i-1}. Since v_{k+1} is not included in S_{i-1}, none of these requests are fulfilled using service S_{i-1}, and they remain active after this service. Thus, the actual time v_{k+1} becomes mature after service S_{i-1} is not more than $\text{mature}_{t_{i-1}}(v_{k+1}) < t_i$. Since S_i has happened at time t_i, it means that v_{k+1} is included in $V_{i,1}$, as it was mature at time t_i. So in **D.1**, we have called LocalCharge(v_{k+1}, S_i), which increases the sum of the α variables by $c(v_{k+1})/4$. We also call LocalCharge(v, S_i) for all $v \in V_{i-1,2} = \{v_1, \ldots, v_k\}$ in **D.2.1**, which increases the sum of the α variables by $\sum_{\ell=1}^{k} c(v_\ell)/4$. Therefore, $\alpha(S_i) \geq \sum_{\ell=1}^{k+1} c(v_\ell)/4 > 2c(r)/4 = c(r)/2$. ∎

LEMMA 3.5. *The total increase of the α variables if step **D.2.2** of Dual Assignment i is invoked is at least $c(r)/2$.*

Proof. For $i = 2, \ldots, N$, in step **D.2.2.1** of Dual Assignment i, for each $\rho \in R_i^1 = \bigcup_{v \in V_{i,1} \setminus V_{i-1}} R_i^v$, we call UniqueGlobalCharge(ρ, S_i). This function increases the value of α_ρ by $b \cdot (t_i - d_\rho) - \alpha_\rho^0$, where α_ρ^0 is the value of α_ρ before calling UniqueGlobalCharge. Note that for each node $v \in V_{i,1} \setminus V_{i-1}$, only the requests in L_i^v are used in the local chargings, which means that for each $\rho \in R_i^v \setminus L_i^v$, we have $\alpha_\rho^0 = 0$. Also, $L_i^v \cap R_i^v$ can only have one member, and if $\rho^* \in L_i^v \cap R_i^v$, Lemma 3.3 shows that $a_{\rho^*} > t_{i-1}$, which means that ρ^* was not used in any of the previous dual assignments, and the first time it was potentially used is in step **D.1** of the current Dual Assignment i. In this case, LocalCharge(v, S_i) is called, which sets the value of α_{ρ^*} to $\frac{1}{4}\alpha_{\rho^*}^*$, where $\alpha_{\rho^*}^*$ is the α value assigned to ρ^* in Lemma 3.1. Lemma Lemma 3.1(V) ensures that $\alpha_{\rho^*}^* \le c(v) - \sum_{\rho \in L_i^v \setminus \{\rho^*\}} b \cdot (t_i - d_\rho)$, which means that

$$\alpha_{\rho^*}^0 \le \frac{1}{4}\left(c(v) - \sum_{\rho \in L_i^v \setminus \{\rho^*\}} b \cdot (t_i - d_\rho)\right) \le c(v) - \sum_{\rho \in L_i^v \setminus \{\rho^*\}} b \cdot (t_i - d_\rho).$$

Also, when $L_i^v \cap R_i^v = \emptyset$, we have $\sum_{\rho \in L_i^v} b \cdot (t_i - d_\rho) = c(v)$. Thus the total increase of the alpha variables in **D.2.2.1** is

$$\begin{aligned}
\sum_{\rho \in R_i^1} (b \cdot (t_i - d_\rho) - \alpha_\rho^0) &= \sum_{v \in V_{i,1} \setminus V_{i-1}} \sum_{\rho \in R_i^v} (b \cdot (t_i - d_\rho) - \alpha_\rho^0) \\
&= \sum_{v \in V_{i,1} \setminus V_{i-1}} \left[\sum_{\rho \in R_i^v \setminus L_i^v} (b \cdot (t_i - d_\rho) - \alpha_\rho^0) + \sum_{\rho \in R_i^v \cap L_i^v} (b \cdot (t_i - d_\rho) - \alpha_\rho^0) \right] \\
&\ge \sum_{v \in V_{i,1} \setminus V_{i-1}} \left[\sum_{\rho \in R_i^v \setminus L_i^v} b \cdot (t_i - d_\rho) + \sum_{\rho \in L_i^v} b \cdot (t_i - d_\rho) - c(v) \right] \\
&= \sum_{v \in V_{i,1} \setminus V_{i-1}} \left[\sum_{\rho \in B_{i,1}^v} b \cdot (t_i - d_\rho) - c(v) \right].
\end{aligned}$$

In **D.2.2.2**, we call CommonGlobalCharge(S_i), where due to Lemma Lemma 3.2(I), increases $\sum_\rho \alpha_\rho$ by at least $\frac{1}{2}\sum_{v \in V_{i,1} \cap V_{i-1}} \left(\sum_{\rho \in B_{i,1}^v} b \cdot (t_i - d_\rho) - c(v)\right)$.

Therefore, the total increase of the α variables in steps **D.2.2.1** and **D.2.2.2** of Dual Assignment i is at least

$$\begin{aligned}
&\sum_{v \in V_{i,1} \setminus V_{i-1}} \left[\sum_{\rho \in B_{i,1}^v} b \cdot (t_i - d_\rho) - c(v) \right] + \frac{1}{2} \sum_{v \in V_{i,1} \cap V_{i-1}} \left[\sum_{\rho \in B_{i,1}^v} b \cdot (t_i - d_\rho) - c(v) \right] \\
&\ge \frac{1}{2} \sum_{v \in V_{i,1}} \left[\sum_{\rho \in B_{i,1}^v} b \cdot (t_i - d_\rho) - c(v) \right] \\
&= \frac{1}{2} c(r),
\end{aligned}$$

where the last equality is derived by the fact that Algorithm 2 has triggered S_i in Mature Backlog Phase after it has accumulated exactly $c(r)$ surplus backlog cost. ∎

LEMMA 3.6. *Let $\alpha(S_i)$ be the total increase of the α variables in Dual Assignment i. We have $\alpha(S_i) \ge \max(\frac{1}{4}c(V_{i,1}), \frac{1}{2}c(r))$ for each service S_i.*

Proof. Each time LocalCharge(v, S_i) is called for an item v and a service S_i that includes v, $\sum_\rho \alpha_\rho$ increases by $\frac{1}{4}\sum_\rho \alpha_\rho^*$, where α_ρ^* is the α variable assigned to ρ in Lemma 3.1. By Lemma Lemma 3.1(I), we know that $\sum_\rho \alpha_\rho^* = c(v)$, which means that the increase in $\sum_\rho \alpha_\rho$ during LocalCharge(v, S_i) is exactly $\frac{1}{4}c(v)$. For $i = 1, \ldots, N$, in Dual Assignment i, step **D.1** is called, in which LocalCharge(v, S_i) is invoked for each $v \in V_{i,1}$. Thus, $\alpha(S_i) \ge \frac{1}{4}c(V_{i,1})$.

Now we prove that $\alpha(S_i) \ge c(r)/2$. Consider the two cases in **D.2**:

- Case I: In this case, by Lemma 3.4, in step **D.2.1** the α variables increase by $c(r)/2$.

- Case II: In this case, by Lemma 3.5, in step **D.2.2** the α variables increase by $c(r)/2$.

□

LEMMA 3.7. *The cost of the algorithm for service S_i is at most $3c(V_{i,1}) + 9c(r)$.*

Proof. We break down the costs into three parts:

- Service Cost: We pay $c(r)$ as the joint service cost, and $c(V_{i,1}) + c(V_{i,2})$ as the item-dependent costs. From the design of the algorithm, we know $c(V_{i,2}) \leq 2c(r)$. So the service cost is at most $c(V_{i,1}) + 3c(r)$.
- Backlog Cost: When S_i is triggered, it means that in Mature Backlog Phase we have $b_t(r) = c(r)$, which means that the surplus backlog cost accumulated after the items in $V_{i,1}$ have become mature is $c(r)$. The backlog cost needed for each item $v \in V_{i,1}$ to become mature is $c(v)$. So the total amount of backlog cost in Mature Backlog Phase is $c(V_{i,1}) + c(r)$. In Premature Backlog Phase, all of the new items that are picked up are premature, which means that the sum of the backlog costs for their requests at time t_i is at most their item cost. This means the sum of the backlog cost incurred in Premature Backlog Phase is at most $c(V_{i,2}) \leq 2c(r)$. Thus the total backlog cost paid in service S_i is at most $c(V_{i,1}) + 3c(r)$.
- Holding Cost: For each item v in $V_{i,1} \cup V_{i,2}$, a holding cost of at most $c(v)$ is incurred in the Local Holding Phase of the algorithm. Moreover, a total cost of at most $c(r)$ is paid in the global holding phase. Therefore, the holding cost paid is at most $(c(V_{i,1}) + c(V_{i,2})) + c(r) \leq c(V_{i,1}) + 3c(r)$.

□

LEMMA 3.8. *The total cost incurred by the algorithm is at most $30 \sum_\rho \alpha_\rho$, for the proposed dual solution.*

Proof. The proof follows immediately from Lemmas 3.6 and 3.7. □

3.7 Dual Feasibility In this section we show that the dual solution presented in Section 3.5 is feasible. The proofs can be found in the full version of our paper.

LEMMA 3.9. *Each request ρ is involved in at most two local chargings and one global charging, i.e., its corresponding variables α_ρ, $\beta_{\rho,t}$ and $\gamma_{v_\rho,t}$ are modified at most two times during the calls to LocalCharge(.) and at most one time during the calls to UniqueGlobalCharge(.) or CommonGlobalCharge(.).*

LEMMA 3.10. *For each item v and each time t, there are only two local chargings that can increase $\sum_{\rho:v_\rho=v} \beta_{\rho,t}$.*

LEMMA 3.11. *The dual solution presented in Section 3.5 is feasible for [JRP$_D$].*

3.8 Proof of Theorem 1.2

Proof. [Proof of Theorem 1.2] For a particular instance I of Online JRP, in Section 3.5, we construct a solution to [JRP$_D$], which by Lemma 3.11, is feasible, and by Lemma 3.6, has an objective value of at least $\frac{1}{30}$Alg(I), where Alg(I) is the cost of Algorithm 2 for instance I. This objective value is a lower bound for D, the optimal value of [JRP$_D$], which in turn, by weak duality, is a lower bound for P, the optimal value of [JRP$_P$]. Since [JRP$_P$] is a relaxation of the original problem, we have the following:

$$\frac{1}{30}\text{Alg}(I) \leq D \leq P \leq OPT(I),$$

where $OPT(I)$ is the cost of the optimal solution for instance I. This shows that Algorithm 2 is a 30-competitive algorithm for Online JRP. □

4 Conclusion

This paper considers the Joint Replenishment Problem (JRP) in the online setting with holding and backlog costs for the first time. The main result is a new constant competitive greedy algorithm. Technically, this paper introduces a dual fitting approach that gives insight into the combinatorial structure of the problem.

There are several interesting directions for future work. One is to consider the case where each request can have a general monotonically increasing cost function for the holding and backlog costs. An example provided in the full version of the paper shows that Algorithms 1 and 2 do not give bounded competitive ratios for this case, even when the cost functions are linear. Another direction is to consider multilevel trees. There is a line of work on online algorithms for multilevel trees with hard deadlines or with just backlog costs (and no holding) [3, 5, 12]. The dual fitting approach here could give insights into improving this line of work and, further, can be useful for giving the first results for handling backlog and holding costs in arbitrarily deep trees. Finally, giving tight competitive ratios for different versions of Online JRP, either by strengthening the upper bounds or the lower bounds, is an intriguing open question. There are no known lower bounds on the competitive ratio of the Online JRP model presented in this paper except for the ones that already hold for the case where just backlogging is allowed.

References

[1] A. Agrawal, P. Klein, and R. Ravi, *When trees collide : An approximation algorithm for the generalized Steiner problem on networks*, SIAM Journal on Computing, 24(3) (1995), pp. 445–456.

[2] E. Arkin, D. Joneja, and R. Roundy, *Computational complexity of uncapacitated multi-echelon production planning problems*, Operations research letters, 8 (1989), pp. 61–66.

[3] Y. Azar and N. Touitou, *General framework for metric optimization problems with delay or with deadlines*, in 2019 IEEE 60th Annual Symposium on Foundations of Computer Science (FOCS), IEEE, 2019, pp. 60–71.

[4] L. Becchetti, A. Marchetti-Spaccamela, A. Vitaletti, P. Korteweg, M. Skutella, and L. Stougie, *Latency-constrained aggregation in sensor networks*, ACM Transactions on Algorithms (TALG), 6 (2009), pp. 1–20.

[5] M. Bienkowski, M. Böhm, J. Byrka, M. Chrobak, C. Dürr, L. Folwarczný, Ł. Jeż, J. Sgall, N. K. Thang, and P. Veselỳ, *Online algorithms for multilevel aggregation*, Operations Research, 68 (2020), pp. 214–232.

[6] M. Bienkowski, J. Byrka, M. Chrobak, N. Dobbs, T. Nowicki, M. Sviridenko, G. Świrszcz, and N. E. Young, *Approximation algorithms for the joint replenishment problem with deadlines*, Journal of Scheduling, 18 (2015), pp. 545–560.

[7] M. Bienkowski, J. Byrka, M. Chrobak, Ł. Jeż, D. Nogneng, and J. Sgall, *Better approximation bounds for the joint replenishment problem*, in Proceedings of the twenty-fifth annual ACM-SIAM symposium on discrete algorithms, SIAM, 2014, pp. 42–54.

[8] A. Borodin and R. El-Yaniv, *Online computation and competitive analysis*, cambridge university press, 2005.

[9] C. F. Brito, E. Koutsoupias, and S. Vaya, *Competitive analysis of organization networks or multicast acknowledgment: How much to wait?*, Algorithmica, 64 (2012), pp. 584–605.

[10] N. Buchbinder, T. Kimbrel, R. Levi, K. Makarychev, and M. Sviridenko, *Online make-to-order joint replenishment model: Primal-dual competitive algorithms*, Operations Research, 61 (2013), pp. 1014–1029.

[11] N. Buchbinder, J. S. Naor, et al., *The design of competitive online algorithms via a primal–dual approach*, Foundations and Trends® in Theoretical Computer Science, 3 (2009), pp. 93–263.

[12] M. Chrobak, *Online aggregation problems*, ACM SIGACT News, 45 (2014), pp. 91–102.

[13] D. R. Dooly, S. A. Goldman, and S. D. Scott, *Tcp dynamic acknowledgment delay (extended abstract) theory and practice*, in Proceedings of the thirtieth annual ACM symposium on Theory of computing, 1998, pp. 389–398.

[14] ———, *On-line analysis of the tcp acknowledgment delay problem*, Journal of the ACM (JACM), 48 (2001), pp. 243–273.

[15] M. X. Goemans and D. P. Williamson, *A General Approximation Technique for Constrained Forest Problems*, SIAM J. Comput., 24 (1995), pp. 296–317.

[16] A. R. Karlin, C. Kenyon, and D. Randall, *Dynamic tcp acknowledgement and other stories about e/(e-1)*, in Proceedings of the thirty-third annual ACM symposium on Theory of computing, 2001, pp. 502–509.

[17] M. Khouja and S. Goyal, *A review of the joint replenishment problem literature: 1989–2005*, European journal of operational Research, 186 (2008), pp. 1–16.

[18] R. Levi, R. Roundy, D. Shmoys, and M. Sviridenko, *A constant approximation algorithm for the one-warehouse multiretailer problem*, Management Science, 54 (2008), pp. 763–776.

[19] R. Levi, R. Roundy, and D. B. Shmoys, *Primal-dual algorithms for deterministic inventory problems*, Math. Oper. Res., 31 (2006), pp. 267–284.

[20] R. Levi, R. Roundy, D. B. Shmoys, and M. Sviridenko, *A constant approximation algorithm for the one-warehouse multiretailer problem*, Management Science, 54 (2008), pp. 763–776.

[21] R. Levi and M. Sviridenko, *Improved approximation algorithm for the one-warehouse multi-retailer problem*, in International Workshop on Approximation Algorithms for Combinatorial Optimization, Springer, 2006, pp. 188–199.

[22] T. Nonner and A. Souza, *Approximating the joint replenishment problem with deadlines*, Discrete Mathematics, Algorithms and Applications, 1 (2009), pp. 153–173.

[23] L. Peng, L. Wang, and S. Wang, *A review of the joint replenishment problem from 2006 to 2022*, Management System Engineering, 1 (2022).

[24] M. L. PINEDO, *Scheduling: Theory, Algorithms, and Systems*, Springer US, 2012.
[25] S. S. SEIDEN, *A guessing game and randomized online algorithms*, in Proceedings of the thirty-second annual ACM symposium on Theory of computing, 2000, pp. 592–601.