RC24362 (W0710-003) October 1, 2007
Computer Science

IBM Research Report

Online Make-to-Order Joint Replenishment Model: Primal Dual Competitive Algorithms

N. Buchbinder[1], T. Kimbrel, R. Levi[2], K. Makarychev, M. Sviridenko
IBM Research Division
Thomas J. Watson Research Center
P.O. Box 218
Yorktown Heights, NY 10598

[1]Computer Science Department
Technion
Haifa 32000
Israel

[2]Sloan School of Management
MIT
Cambridge, MA 02139

Research Division
Almaden - Austin - Beijing - Haifa - India - T. J. Watson - Tokyo - Zurich

Online Make-to-Order Joint Replenishment Model: Primal Dual Competitive Algorithms

N. Buchbinder* T. Kimbrel† R. Levi ‡ K. Makarychev § M. Sviridenko¶

Abstract

In this paper, we study an *online make-to-order* variant of the classical joint replenishment problem (JRP) that has been studied extensively over the years and plays a fundamental role in broader planning issues, such as the management of supply chains. In contrast to the traditional approaches of the stochastic inventory theory, we study the problem using competitive analysis against a worst-case adversary.

Our main result is a 3-competitive deterministic algorithm for the online version of the JRP. We also prove a lower bound of approximately 2.64 on the competitiveness of any deterministic online algorithm for the problem. Our algorithm is based on a novel primal-dual approach using a new linear programming relaxation of the offline JRP model. The primal-dual approach that we propose departs from previous primal-dual and online algorithms in rather significant ways. We believe that this approach can extend the range of problems to which online and primal-dual algorithms can be applied and analyzed.

1 Introduction

Inventory theory provides streamlined stochastic optimization models that attempt to capture tradeoffs between opposing costs in managing the flow of multiple types of goods through a supply chain. Uncertainty about future demands is one of the prime features that make inventory management problems very complex. Most traditional stochastic inventory theory literature assumes that the probability distributions of the future demands are known, or at least partially known, as part of the input. However, in many practical scenarios the underlying probability distributions are not available. Moreover, any attempt to create a distributional model of the demands entails a significant risk of serious misspecification which may lead to very poor policies. To overcome this fundamental difficulty, we propose an online framework. We consider classical inventory models, but under the assumption that the demands are revealed to us online and are generated by a worst-case adversary. In particular, we propose a primal-dual based algorithmic approach that leads to policies with robust performance under competitive analysis. Our main result is a 3-competitive deterministic online algorithm for the online version of the classical JRP.

Our approach has significant contributions in at least two important dimensions. First, to the best of our knowledge, the concept of online algorithms has not been applied to inventory management models before. Moreover, we believe that the online framework that we propose can be extended to other classical inventory management problems. In particular, this approach can be very effective in devising robust policies that can be implemented under minimal knowledge about the evolution of future demands. Secondly, our online primal-dual approach is based on several new algorithmic ideas that significantly depart from previous primal-dual algorithms for offline problems and from online algorithms. We believe that these ideas can extend the range of problems to which online and primal-dual algorithms can be applied and analyzed.

Our lower bound also employs a technique that is not seen, to the best of our knowledge, in previous work on related problems.

The joint replenishment model. In this paper, we study the classical joint replenishment problem (JRP) that has been studied extensively over the years, and plays a fundamental role in broader planning issues, such as the management of supply chains (e.g. [3, 15]). We consider the JRP model under a *make-to-order* coordination mechanism, where the production of finished goods is triggered only by actual demands, and no finished goods are held in inventory [7, 10, 14]. This classical coordination mechanism is very common in scenarios with highly customized finished goods and high setup and changeover costs, for example in the steel and glass industries. (The other common coordination mechanism is *make-to-stock*, in which finished goods are produced before the actual demand occurs, and then

*Computer Science Department, Technion, Haifa 32000, Israel. E-mail: *nivb@cs.technion.ac.il*

†IBM T. J. Watson Research Center, P.O. Box 218, Yorktown Heights, NY 10598. `kimbrel@us.ibm.com`

‡Sloan School of Management, MIT, Cambridge, MA, 02139. Part of this research was conducted while the author was a postdoctoral fellow at the IBM, T. J. Watson Research Center. `retsef@mit.edu`

§IBM T. J. Watson Research Center, P.O. Box 218, Yorktown Heights, NY 10598. `konstantin@us.ibm.com`

¶IBM T. J. Watson Research Center, P.O. Box 218, Yorktown Heights, NY 10598. `sviri@us.ibm.com`

kept in inventory.)

The details of the *offline make-to-order* JRP model are as follows. A set $\mathbb{D}$ of demands for N different commodities is specified over a planning horizon of T discrete periods. Each demand point $(i,t) \in \mathbb{D}$ requires certain number of units of commodity i, where $i = 1,\ldots,N$ and $t = 1,\ldots,T$. Since holding inventories is not allowed, demand point (i,t) can be satisfied only from orders placed in period t or later in time. At the beginning of each period $s = 1,\ldots,T$, we can place an order for any combination of items, and this order can be used to serve pending demand points that arrived in period s or earlier in time. Each such order incurs a fixed *joint ordering cost* K_0, as well as a fixed *item ordering cost* K_i for each item (commodity) $i = 1,\ldots,N$ included in the order, regardless of the number of units being ordered. The fixed ordering costs drive us to order in large batches. This is traded off against *delay penalty cost*, which captures the costs incurred by allowing demand points to wait for some amount of time until they are satisfied. For each demand point (i,t) and a potential order $s \geq t$, we let w_{its} be the corresponding late penalty cost incurred by demand point (i,t), if served from an order in period s. The only assumption is that, for each single demand point (i,t), the delay penalty cost w_{its} is increasing in s. This includes the case in which a demand point must be served within a specified time window after its arrival. (In this case $w_{its} = \infty$ after the time window.) The goal is to find a feasible ordering policy that minimizes the overall ordering and delay penalty costs. The offline JRP model is known to be NP-hard [2]. Recently, Levi, Roundy and Shmoys have proposed a primal-dual 2-approximation algorithm for the offline make-to-stock JRP model [15], and this has been improved by Levi, Roundy, Shmoys and Sviridenko, who proposed a 1.8-approximation algorithm using an LP-rounding approach [16]. (In the offline make-to-stock JRP model there are holding costs for maintaining physical inventory instead of delay penalty costs. However, mathematically the make-to-stock and the make-to-order offline models are equivalent, and all the complexity results and algorithms can be transparently converted to the offline make-to-order variants.) The special case of the offline joint replenishment problem with a single type of commodity, also known as the *single-item lot-sizing problem*, is polynomially solvable via dynamic programming [18] or primal-dual algorithms (for example, [11, 15]).

In the *online make-to-order* JRP model, the demand points and the length of the planning horizon T are not known in advance. Instead, demand points arrive in an online manner, and the ordering decision made in each time period s is based only on demand points that arrived up to period s. We measure the performance of our algorithms by the standard competitive ratio between the cost incurred by the online algorithm and the minimum cost obtained by an offline optimal solution when the input is generated by a worst-case adversary. The *online single-item lot-sizing* problem is equivalent to the well understood *TCP-acknowledgment problem*. Dooly, Goldman and Scott gave an optimal 2-competitive deterministic algorithm for this problem [8]. Karlin, Kenyon and Randall gave an optimal $e/(e-1)$-competitive randomized algorithm [13]. Buchbinder, Jain and Naor have recently proposed an analysis of these algorithms using a primal-dual approach [4].

1.1 Our results and techniques The main result of this paper is a 3-competitive deterministic algorithm for the online make-to-order JRP model. In addition, we obtain a lower bound of 2.64 on the competitive ratio of any deterministic online algorithm.

The deterministic algorithm for the online TCP-acknowledgement problem (or the online make-to-order single-item lot-sizing problem) [8] is simply based on balancing the ordering cost with the delay penalty cost. We place an order whenever the cumulative delay penalty cost of the currently pending demands exceeds K, where K is the fixed ordering cost. However, it can be shown that any straightforward extension of this idea to the JRP model does not lead to algorithms with bounded competitive ratio. The fundamental problem is the choice of which items (commodities) should be included once the decision has been made to place a (joint) order. Intuitively, the dilemma is whether to include in the order items for which the current cumulative delay penalty costs are relatively low. Including such items runs into the risk of incurring high item ordering costs because of unnecessary orders of these items; excluding them runs the risk of incurring high joint ordering costs due to placing multiple orders, each including only a small number of items. More specifically, it is not clear whether the future arrivals of demands will justify additional joint orders that can be used to satisfy the currently pending demands of those items. Instead it may be better to serve them from the order placed in the current period.

We propose a primal-dual algorithm for the online JRP model. The primal-dual approach is a well-studied discipline in the design and analysis of optimization and approximation algorithms. Buchbinder and Naor [5, 6] have recently proposed a primal-dual framework for online packing and covering problems. This framework includes, for example, a large number of routing and load balancing problems, the online set cover problem [1], as well as other problems. Recently, the framework was

shown applicable to several other interesting problems [4]. As we shall discuss below, the algorithm we propose in this paper and the performance analysis require new and more delicate algorithmic ideas (e.g., the *simulation step* that we describe below) that significantly depart from previous work.

Our primal-dual approach is based on a novel linear programming relaxation (LP) of the offline JRP model. Our LP and its dual program guide the decisions made by the online algorithm. Specifically, we simultaneously construct online both a feasible primal integer solution and a feasible dual solution. We then show that the cost of the primal solution is at most 3 times the cost of the dual solution, obtaining a 3-competitive algorithm. For intuition, it helps to think of the dual variables as *budgets* that are being used to pay for the cost of the primal solution, which consists of the joint ordering cost, the item ordering costs and the delay penalty costs. The budget of each as-yet-unknown demand point (i,t) is 0 until time t. Starting at time t, the corresponding budget can change until this demand point is frozen.

We note that our primal-dual algorithm departs from previously known algorithms in rather significant ways. In particular, the online nature of the problem requires several fundamentally different algorithmic approaches compared to primal-dual algorithms that were previously designed for offline versions of the JRP model [15], the facility location problem [12] and other combinatorial problems (e.g., [9]). For example, all of these algorithms have a second *clean-up phase*, in which the initial primal solution is modified to make it cheaper. This is not possible in an online setting, and we need a different approach.

Now we describe the main aspects of the new algorithmic approach that we propose and contrast it with the previous offline primal-dual algorithms. Joint orders are placed when the budgets of a subset of demand points are sufficient to pay the joint ordering cost and the item ordering costs of a certain potential order, as well as the resulting delay penalty costs incurred by serving them from that order. (This corresponds to a dual constraint becoming tight.) However, this potential order may be in the past, and because the problem is online it can not be used anymore. Instead we place an order in the current period, initially including all the corresponding items, and freeze the budgets of all the demand points that will be served by this order. As already mentioned, the fundamental issue that arises at this point is whether to include additional items in the current order even though their current budgets are not sufficient to pay for their respective item ordering costs. This issue is handled through what we call a *simulation step*. We temporarily increase the algorithmic time indicator beyond the current time period, and adjust the budgets of the active (not frozen) demand points accordingly, assuming that no demand point will arrive in future periods. Throughout this phase certain dual constraints become tight and this guides the algorithm in including additional items in the current order. However, at the end of the simulation step all the dual budgets are decreased back to their original values at the beginning of the simulation step, and the algorithmic time indicator is again aligned with the current time period. In particular, it is not guaranteed that the budgets of the corresponding demand points will eventually reach their *simulation values*. Thus, unlike most other primal-dual approximation algorithms, our algorithm is not dual ascent. That is, the budgets of certain demand points can temporarily go up, and then be decreased, and go up again, until they are frozen. Intuitively, one can think of the simulation step as a 'loan' taken by the algorithm to pay for additional orders of certain items. It is not clear a-priori that it will be able to fully recap this 'loan' in the future. Thus we need to make sure that the 'loan' being taken is not too high; in other words, we need to carefully monitor the simulation step and design the right *stopping rule*. We believe that the concept of the primal-dual simulation step will have applications in other online and offline settings, and will extend the applicability of primal-dual approximation and online algorithms. In addition, unlike previous algorithms, the performance analysis of our algorithm is not based on showing how to use the dual budgets to pay the cost of each order placed by the algorithm separately. Instead, we use the algorithm to partition the orders into *clusters*, and then show how to use the dual budgets to pay for the overall cost of each cluster. Specifically, we show that each dual budget is not used more than 3 times.

2 Preliminaries

In this section we formally define our problem. Let $\mathcal{H} = \{1,2,\ldots,N\}$ be the set of item types (commodities). In the JRP model the cost of an order does not depend on the number of units ordered, but only on the item types in the order. The cost of an order is composed of a *joint ordering cost* denoted by K_0 and a fixed *item ordering cost* of K_i for ordering any number of units of item type i. Thus, the cost of ordering a set $S \subseteq \{1,2,\ldots,N\}$ is $C(S) \triangleq K_0 + \sum_{i \in S} K_i$. New demands arrive at times $t \in \{1,2,\ldots,T\}$ for some unknown T. Let $\mathbb{T} = \{1,2,\ldots,T\}$ be the set of times in the planning horizon. Each demand specifies some item type that is needed. Let $\mathbb{D}$ be the set of demands. Each demand is thus defined by a pair (i,t), where i is the item type associated with the demand and $t \in \mathbb{T}$ is the arrival time of the demand.

Every demand $(i,t) \in \mathbb{D}$ can be satisfied in any time period $s \geq t$ after its arrival. The *delay penalty cost* of satisfying demand (i,t) at time s is w_{its} which is an increasing function of s. It is convenient to view the delay function in increments between each time s and the subsequent time $s+1$. Let $\Delta_{its} \geq 0$ be the additional penalty of delaying demand (i,t) from time s until time $s+1$. That is $\Delta_{its} = w_{i,t,s+1} - w_{its}$.

Linear Program Formulation and Its Dual. It is not difficult to verify that there exists an optimal offline solution that makes orders only at times at which new demands arrive (i.e., times $t \in \mathbb{T}$). To define a lower bound on the optimal cost we start by formulating the problem as a linear program whose minimum solution is a lower bound on the optimal integral cost. We define for each set S and time $t \in \mathbb{T}$ a variable $x(S,t)$ that is an indicator for the event of sending an order for the types of S at time t. Let $z(i,t,s)$ indicate the event of delaying demand (i,t) from time s until time $s+1$. The cost of any offline algorithm can be described as the linear cost function in line (2.1) below. The constraints in line (2.2) stipulate that for any demand (i,t) and time s greater than its arrival time, either the demand has been served by some order prior to time s, or it is delayed at time s. We omit the constraints that all variables must be non-negative. Note that the numbers of constraints and variables in this the LP formulation are exponential with respect to the number of item types. However, since our proposed algorithm never updates the variables explicitly, it can be implemented efficiently.

$$\text{(2.1)} \quad \min \sum_{t=1}^{T} \sum_{S \subseteq \{1,\ldots,r\}} C(S)x(S,t) + \sum_{(i,t)\in\mathbb{D}} \sum_{s \geq t} \Delta_{its} \cdot z(i,t,s)$$

$$\forall (i,t) \in \mathbb{D}, s \geq t:$$

$$\text{(2.2)} \quad \sum_{\tau=t}^{s} \sum_{S \mid i \in S} x(S,\tau) + z(i,t,s) \geq 1.$$

Next, we define the dual linear programming formulation for the primal linear program (2.1)-(2.2). Obtaining the dual program can be done in a technical way by defining a dual variable for any primal constraint, and a dual constraint for any primal variable. However, the dual program in this case has an intuitive meaning. In the dual program we have a variable $y(i,t,s)$ for each demand (i,t) and time $s \geq t$. The dual variable can be interpreted as a budget. The objective of the dual program is to maximize the total budget earned from all demands. Constraint (2.4) states that the budget earned by demand (i,t) between two consecutive times is at most the delay it incurred between these two consecutive times. Constraint (2.5) stipulates that the total budget earned by demands whose type is in S and which arrived prior to time s cannot exceed the cost of ordering the types in S. Thus, the intuition is that each demand of some type i earns budget that pays for its order K_i. After it has paid for its fixed item ordering cost, it starts contributing to the joint ordering cost K_0.

$$\text{(2.3)} \quad \max \sum_{(i,t)\in\mathbb{D}} \sum_{s \geq t} y(i,t,s)$$

$$\text{(2.4)} \quad \forall (i,t) \in \mathbb{D}, s \geq t : y(i,t,s) \leq \Delta_{its}$$

$$\forall S \subseteq \{1,\ldots,N\}, s \in \mathbb{T}:$$

$$\text{(2.5)} \quad \sum_{(i,t) \mid t \leq s, i \in S} \sum_{r \geq s} y(i,t,r) \leq C(S)$$

3 The Online algorithm

In this section we present our online algorithm. We start with intuition. It is instructive to think first about a much simpler setting with only a single item type. Suppose that a demand (for this one type) arrives. If no additional demand arrives in the near future, it is best to serve this demand right away. However, if an additional demand arrives soon enough then we would benefit from delaying the first demand and serving both demands together. Of course, we do not know which of the two will happen. Thus, the question is how much delay cost should we incur before making an order. It can be shown that the optimal deterministic approach is to wait until the delay cost incurred exactly equals the order cost. This approach results in an optimal 2-competitive algorithm [8]. In our terminology, this corresponds to an algorithm that waits until the accumulated budgets of all demands pay for their order. However, it is not hard to show that such an algorithm will fail in the more complex JRP problem studied here. Intuitively, the bad example would force such an algorithm to incur the ordering cost K_0 too often. Suppose we have many item types, with orders for each at time 0, and their respective waiting costs increase rapidly close to their deadlines but the deadlines are equally distributed in time. Then the previous algorithm would open an order for each demand separately very close to its deadline and incur the joint ordering cost K_0 for each demand, and the optimal algorithm will order items in batches and will not pay K_0 as often. Thus, new ideas are needed.

Suppose that a set of types of items S have gathered an accumulated budget that may pay for the order cost of the set of types in S. That is, the accumulated budget is at least the joint ordering cost along with the additional fixed item order cost of each type in the set

S. We certainly want to make an order to the set of types in S. However, since this ordering already pays for the joint ordering cost, we might want to add other types that still have not fully paid for their fixed cost. Otherwise, we might need to pay the joint ordering cost several times while the optimal solution joins all these demands together and pays the joint ordering cost only once. On the other hand, it is not wise to add all the types that have not yet paid for their fixed cost, since more demands of these types may arrive in the near future. The question is how many and which ones of these additional "premature" types should be added to the current order.

The main novel idea that is used in our algorithm is the *simulation step* that uses the current known information to decide which of the types will be added to the current joint order. In the simulation step we virtually increase the algorithmic time to future time periods assuming that no additional demands arrive. During this process we increase the delay cost of the demands that have arrived and are not satisfied yet, and so additional types manage to "pay" for their order cost. These types are added to the current order, and their corresponding demands become "half active" (defined below). Continuing the simulation for a long time has the benefit of adding more types to the current order that already paid for the joint order cost. However, each additional type that is added prematurely adds to the current order cost more than its current budget. Since more demands of that type might arrive in future times, the "future" budget that convinced us to add it to the current order is not guaranteed. Therefore, the idea is to continue with the simulation step as long as the extra future budget that we use is at most the joint order cost, K_0. This extra budget exactly balances between these two options.

3.1 Algorithm Description We are now ready to formally describe our algorithm. Our online algorithm uses an *algorithmic time indicator* τ that starts at time 0 and is increased continuously as time progresses. At each discrete time s the algorithm simulates a continuous increase of the time indicator τ from time s to $s+1$. Any orders that are indicated at algorithmic time τ are actually made at time $s = \lfloor \tau \rfloor$.

Each demand (i,t) may be *active*, *half active* or *inactive* during the algorithm execution. Each unsatisfied demand is always active, so each demand is active from its arrival time until it is satisfied at some algorithmic time τ. After this time τ the demand (i,t) becomes either half active or inactive. Half active demands are demands that were already satisfied, but they still play some role in our algorithm. Intuitively, these demands were satisfied "prematurely" and we want them to pay for some other costs. When demand is satisfied it may become half active for a certain time, and then it becomes inactive forever. Let $\tau_e(i,t)$ be the time demand (i,t) becomes half active. Let $\tau_f(i,t)$ be the time demand (i,t) becomes inactive. If $\tau_f(i,t) > \tau_e(i,t)$ then during the time interval $[\tau_e(i,t), \tau_f(i,t)]$ the demand (i,t) is half active. When a new demand arrives and is not served yet, we set $\tau_e(i,t) = \tau_f(i,t) = \infty$. We remark again that $\tau_e(i,t)$ and $\tau_f(i,t)$ do not have to be integral.

Our proposed online algorithm maintains during each time τ a list of active (unsatisfied) demands $A_\tau \subseteq \mathbb{D}$ and a set $H_\tau \subseteq \mathbb{D}$ of half active demands. These sets contain, of course, only demands that arrived prior to time τ. At the beginning of the planning horizon A_0 consists of demands that arrived at time 0 and $H_0 = \emptyset$. If the algorithm changes the status of a set of demands at time τ then we denote by A_τ^- and A_τ^+ the set of active demands just before and after this change. Similarly, let H_τ^- and H_τ^+ denote the set of active demands just before and after the change.

The algorithm maintains the following assignment to each dual variable $y(i,t,s)$. Until τ becomes s the variable is zero. If demand (i,t) is active or half active at time $\tau \geq s$ then the variable is increased continuously by the linear function $y(i,t,s) \leftarrow (\tau - s)\Delta_{its}$. The increment stops whenever the demand becomes inactive or when $\tau = s+1$, the earlier event. From that point on the dual variable remains a constant with value $\min\{(\tau_f(i,t) - s)\Delta_{its}, \Delta_{its}\}$.

Orders are triggered when dual constraints become tight due to increments of the dual variables. To this end, we consider at time τ any set S and any time $s \leq \tau$ and check whether the total sum of the dual variables that contribute to the suitable dual constraint is exactly $C(S)$. Demands that *contribute* to the constraint are demands whose type is in S and their arrival time is prior to s. We may consider only times s that are integral (i.e. $s \in \mathbb{T}$), since if the constraint is tight for non-integral time τ' then moving backwards to time $\lfloor \tau' \rfloor$ can only increase the total sum. This happens since in our model no new demands arrive between time $\lfloor \tau' \rfloor$ and time τ'.

When reading the algorithm description note the similarity between the work of the algorithm during the simulation step and the work of the algorithm during the main part. The only difference is that demands that are being added to the order in the simulation step are marked as half active and not inactive as in the main part of the algorithm. The formal description of the algorithm is the following:

Inventory algorithm: Upon arrival of a demand (i,t) set $\tau_f(i,t) = \tau_e(i,t) = \infty$.
At each algorithmic time τ:

Keep the invariant that for any set of types S and time $s \leq \tau$: $\sum_{(i,t)|t\leq s, i\in S} \sum_{r\geq s} y(i,t,r) \leq C(S)$.

If there exist a set of types S and time s such that $\sum_{(i,t)|t\leq s, i\in S} \sum_{r\geq s} y(i,t,r) = C(S)$:

1. Set all half active demands that contribute to the constraint to be inactive.
   Formally, if $(i,t) \in H_\tau$ and also $i \in S, t \leq s$ then set $\tau_f(i,t) = \tau$.
2. If there are no active demands that contribute to the tight constraint return to (1). Otherwise:
3. Set all half active demands to be inactive. Note that we may change status of some demands that do not contribute to the tight constraint.
4. If s is later than the time of the previous order made by the algorithm, then set an order to the all item types in the tight constraint $O \triangleq S$.
5. If s is at most the time of the previous order made by the algorithm, then set an order to the set of types of demands that are active and contribute to the tight constraint. Formally:
$$O \triangleq \{i \mid \exists (i,t) \in A_\tau \text{ such that } i \in S, t \leq s\}$$
6. Set all active demands (i,t) whose type $i \in O$ to be inactive.

Simulation step:

- Simulate the execution of the algorithm in future times $\tau(\text{future}) > \tau$. If there are no active demands left, or the accumulated value of the dual variables $y(i,t,s)$ from time τ until time $\tau(\text{future})$ is at least K_0 then **stop** the simulation step and return to line (1).
- Otherwise: if there exists a set S' and time $s(\text{past})$ such that
$$\sum_{(i,t)|t\leq s(\text{past}), i\in S'} \sum_{r\geq s} y(i,t,r) = C(S')$$

  1. Add to the order O all types in
  $$O' \triangleq \left\{ i \;\middle|\; \begin{array}{l} \exists (i,t) \in A_{\tau(\text{future})} \\ \text{ such that } i \in S', t \leq s(\text{past}) \end{array} \right\}$$
  2. Set all active demands (i,t) whose type $i \in O'$ to be half active at time τ and inactive at time $\tau(\text{future})$ ($\tau_e(i,t) = \tau$, $\tau_f(i,t) = \tau(\text{future})$) and continue the simulation.

3.2 Analysis The main idea in the analysis is to construct a primal solution whose cost is that of the solution obtained by the online algorithm. Then we construct a corresponding dual solution based on the variables used by the online algorithm and prove that this dual solution is indeed feasible. The main difficulty is proving that the cost of the primal solution is at most 3 times the value of the dual solution. To prove this we bound the total delay penalty cost and the total order cost of the algorithm as a function of the dual variables. We then sum up all costs and prove that each dual variable appears in the sum at most 3 times. We conclude the proof using the standard weak duality theorem.

The primal solution is defined simply by setting for each order O made by the online algorithm at time $s = \lfloor \tau \rfloor$, $x(O,s) = 1$. Also we set $z(i,t,s) = 1$ for any demand (i,t) and time s such that $\tau_e(i,t) \geq s+1$ and otherwise we set $z(i,t,s) = 0$. Note when a demand becomes half active it is already served and so we can set $z(i,t,s) = 0$ for any s such that $\tau_e(i,t) < s+1$. The dual solution is given by the assignments to the dual variables $y(i,t,s)$ determined during the execution of the algorithm. The following Lemma proves that the dual solution produced this way is feasible.

LEMMA 3.1. *The assignment $y(i,t,s)$ for each demand (i,t) and time s that is maintained during the execution of the algorithm defines a feasible dual assignment to (2.4)-(2.5).*

Proof. First, since each $y(i,t,s) \leq \Delta_{its}$ all constraints (2.4) are satisfied. Assume to the contrary that the assignment violates one of the constraints (2.5). Then there exists a set S and time s such that:
$$\sum_{(i,t)|t\leq s, i\in S} \sum_{r\geq s} y(i,t,s) > C(S)$$
Thus, by this (false) assumption there must be a time τ during the execution of the algorithm for which
$$\sum_{(i,t)|t\leq s, i\in S} \sum_{r\geq s} y(i,t,s) = C(S).$$
Also, there must be some demand (i,t) with type $i \in S$ that arrived prior to time s and is active or half active at time $> \tau$. This follows because the dual constraint is violated at time $> \tau$. However, by the algorithm behavior (lines (1),(6)), at the algorithmic time τ all active and half active demands that contribute to the constraint become inactive, and so we are done.

In order to prove that the cost of the primal solution produced by the algorithm is at most 3 times the cost of the dual solution we need further notations. We

consider the dual constraints that become tight during the execution of the online algorithm. Note that dual constraints may become tight either in the main section of the algorithm or in the simulation step. Some of the constraints that become tight during the simulation step may not become tight later during the real execution of the algorithm. We elaborate on this issue later on. Let R be any constraint (2.5) in the dual LP that became tight during the algorithm execution. Such a constraint is defined by its starting time period $\tau_s(R)$, its set of types $W(R)$ and the time period $\tau_e(R)$ when this constraint becomes tight. Note again that we may assume that $\tau_s(R)$ is integral. However, $\tau_e(R)$ is an algorithmic time that may be any real number. In our dual solution only dual variables $y(i,t,s)$ that belong to active or half active demands have a positive value, i.e., variables $y(i,t,s)$ with $t \leq s \leq \tau_f(i,t)$. Let $D(R)$ be the set of demands that contribute to the constraint R. Formally,

$$D(R) = \{(i,t) \mid i \in W(R), s \leq \tau_s(R), \tau_f(i,t) > \tau_s(R)\}$$

For each constraint R let $\tilde{W}(R) \subseteq W(R)$ be the set of types of demands that are active or half active at the time moment when the constraint R becomes tight. Formally,

$$\tilde{W}(R) = \left\{i' \mid \exists (i,t) \in D(R), i' = i, \tau_f(i,t) \geq \tau_e(R)\right\}.$$

Finally, let $D(\tilde{W}(R))$ be the set of demands in $D(R)$ of types from the set $\tilde{W}(R)$. Formally,

$$D(\tilde{W}(R)) = \left\{(i,t) \mid (i,t) \in D(R), i \in \tilde{W}(R)\right\}$$

Next, we consider the orders made by the algorithm. Let $\tau_1 \leq \tau_2 \leq \tau_3 \leq \ldots$ be the times in which the algorithm makes an order. The algorithm makes an order whenever it encounters a tight constraint that contains active demands, i.e. $D(R) \cap A^-_{\tau_e(R)} \neq \emptyset$. Let $R_1, R_2, \ldots$ be the tight constraints that caused these orders. We partition the time interval into phases. A new phase p starts whenever the starting time of R_i is later then the ending time of the next constraint R_{i-1} (i.e. $\tau_s(R_i) > \tau_e(R_{i-1})$). The first phase starts at $\tau_s(R_1)$. An example of such a partition appears in Figure 1 where we view each constraint R as an interval between $\tau_s(R)$ and $\tau_e(R)$.

We separate the first constraint that started each phase and other constraints that became tight in the same phase. Note that each phase contains at least one tight constraint that triggered the first order. Let $R_{p,1}$ be the first constraint that became tight in phase p. Let $\mathbb{R}_p$ be all the other constraint that became tight during phase p. The set $\mathbb{R}_p$ also includes the constraints that became tight during the simulation step of the algorithm and possibly did not become tight during the actual execution of the algorithm due to the beginning of phase $p+1$ (See figure 1). Let $\tau_s(p) = \tau_s(R_{p,1})$ be the start of a phase p that is equal to the starting time of the first constraint that became tight during that phase. Let $\tau_e(p) = \tau_e(R_{p,1})$ be the time in which $R_{p,1}$ became tight. Finally, let $\tau_f(p)$ be the ending time period of the phase p that is the time period in which the last constraint that caused an order in the phase p to become tight. This definition induces the following partition of the time interval $\tau_s(p) \leq \tau_e(p) \leq \tau_f(p) < \tau_s(p+1) \leq \tau_e(p+1) \leq \ldots$.

We next want to bound the total cost of the online algorithm using the value of the dual solution. We start with some intuition that will make the technical lemmas and the algorithm behavior more clear. We show in Lemma 3.3 that the delay cost incurred by the online algorithm is at most the value of the dual solution. Bounding the order cost is harder. The proof reveals the ideas behind the simulation step and the partition into phases. The first constraint in each phase is tight and thus the total "budget" inside it is enough to pay for both the joint order cost of the first order in the phase and the fixed order cost of all types ordered in this first order (excluding the types that are added in the simulation step). The main observation is that if the phase did not end after the first order, then all the constraints that become tight during the simulation step also become tight during the actual execution of the algorithm and with the same order. We formalize this in the following lemma:

LEMMA 3.2. *For each phase p, the constraints that become tight in the interval $(\tau_e(p), \tau_f(p)]$ are exactly the same constraints that became tight during the simulation step done by the algorithm.*

Proof. All the tight constraints that become tight during this time interval have a starting time earlier than the ending time of the previous order; otherwise, a new phase starts by definition. Therefore, only demands that arrived prior to the previous order (in the phase p) may contribute to these constraints. Since the simulation step simulates exactly the events that involve demands that already arrived, the constraints that became tight are exactly the constraints that became tight during the simulation step, and they become tight in the same order as in the simulation step.

In particular, note that the constraint that became tight and caused the second order in the phase is

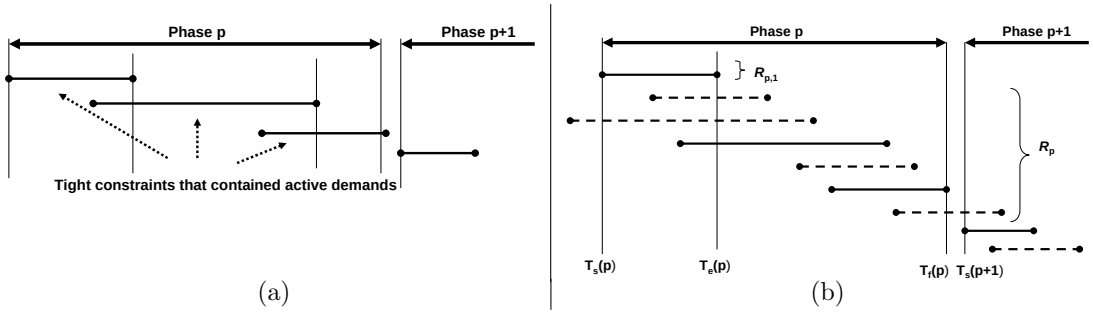

Figure 1: Phase division (a) and the constraint $R_{p,1}$ and $\mathbb{R}_p$ The dashed constraints are constraints that became tight during the simulation step and did not cause an order when they became tight.

constraint that became tight after accumulating of value of at least K_0 in the dual. This is true for each successive order as long as the current phase did not end. The value accumulated this way is enough to pay the joint ordering cost of each of the orders from the second to the last. Finally, we need to pay the fixed cost of the the types that were ordered during the second to the last order in the phase and also for the types that were added due to the simulation step in the first order. We want to sum the dual profit only until the last order of the phase in order to bound the number of times we use each dual variable. We prove that each type that was added due to a constraint that actually became tight before the last order of the phase can pay for its fixed ordering cost. The only problem is with types that were added in the last order of the phase due to the simulation step. These types are using future tight constraints. Since the phase ended, this future budget cannot be used since it might never become available. However, by the behavior of the algorithm during the simulation step the total future budget that is used is bounded by at most K_0. Therefore, we may charge this extra cost to the budget of the first tight constraint in the phase.

LEMMA 3.3. *The total delay cost of the online algorithm is at most:*

$$\sum_{(i,t)\in\mathbb{D}} \sum_{s=t}^{s<\tau_e(i,t)} y(i,t,s) \tag{3.6}$$

Proof. By the construction of the primal and dual solutions, each time the algorithm defines $z(i,t,s) = 1$ of an active demand (i,t), then also $y(i,t,s) = \Delta_{its}$. Note that in this case $s+1 \leq \tau_e(i,t)$.

The following more involved lemma bounds the total order costs made by the online algorithm.

LEMMA 3.4. *The total order cost of the online algorithm is at most:*

$$\begin{aligned}\sum_{p\in\mathbb{P}} \Bigg(& 2 \cdot \sum_{s=\tau_s(p)}^{s<\tau_e(p)} \sum_{(i,t)\in D(W(R_{p,1}))} y(i,t,s) \\ & + \sum_{s=\tau_e(p)}^{s<\tau_f(p)} \sum_{(i,t)\in\mathbb{D}} y(i,t,s) \\ & + \sum_{R\in\mathbb{R}_p} \sum_{(i,t)\in D(\tilde{W}(R))} \sum_{s=\tau_s(R)}^{s<\min\{\tau_e(R),\tau_f(p)\}} y(i,t,s) \Bigg)\end{aligned} \tag{3.7}$$

Proof. For every order of the set S of types, we pay $K_0 + \sum_{i\in S} K_i$. We prove that for each phase p, the total sum of terms that correspond to phase p is at least the total order cost of the algorithm during that phase. As previously observed, the accumulated value of the dual variables between each two orders after the first order in a phase is at least K_0. Thus, the sum of all dual variables from the time of the first order of a phase until the time of the last order in a phase fully pays for all joint order costs from the second to the last order of the phase. Formally, if there are M orders during the phase then: $\sum_{s=\tau_e(p)}^{s<\tau_f(p)} \sum_{(i,t)\in\mathbb{D}} y(i,t,s) \geq (M-1)K_0$.

The second observation is that since the first constraint in each phase p is tight then

$$\sum_{s=\tau_s(p)}^{s<\tau_e(p)} \sum_{(i,t)\in D(W(R_{p,1}))} y(i,t,s) = K_0 + \sum_{i\in W(R_{p,1})} K_i \tag{3.8}$$

Up to now, we charged to the dual variables all joint ordering costs during the phase and also the fixed ordering cost of types that were ordered during the first order of phase p, excluding the types that were added to the first order due to the simulation step. We are left with the fixed ordering cost of all types that were ordered from the second to last order of phase p and the types that were added to the first order in the phase due to the simulation step. We claim that for each constraint

$R \in \mathbb{R}_p$ that became tight in the actual execution of the algorithm or in the simulation step:

$$\sum_{(i,t)\in D(\tilde{W}(R))} \sum_{s=\tau_s(R)}^{s<\tau_e(R)} y(i,t,s) \geq \sum_{i\in\tilde{W}(R)} K_i. \tag{3.9}$$

The inequality (3.9) means that the dual variables corresponding to active demands of the inequality R are enough to pay for item ordering costs. Assume that this is not true. Then there exists some tight constraint $R \in \mathbb{R}_p$ such that

$$\sum_{(i,t)\in D(\tilde{W}(R))} \sum_{s=\tau_s(R)}^{s<\tau_e(R)} y(i,t,s) < \sum_{i\in\tilde{W}(R)} K_i$$

However, since constraint R became tight we get that $\sum_{(i,t)\in D(R)} \sum_{s=\tau_s(R)}^{s<\tau_e(R)} y(i,t,s) = K_0 + \sum_{i\in W(R)} K_i$. Since $D(\tilde{W}(R))$ contains all the demands of type $\tilde{W}(R)$ that contribute positive value to the demand R we get by subtracting the two equations that:

$$\begin{aligned}
&\sum_{(i,t)\ \mid i\in(W(R)\setminus\tilde{W}(R)), t\leq\tau_s(R)} \sum_{s=\tau_s(R)}^{s<\tau_e(R)} y(i,t,s) \\
&= \sum_{(i,t)\in D(R),(i,t)\notin D(\tilde{W}(R))} \sum_{s=\tau_s(R)}^{s<\tau_e(R)} y(i,t,s) \\
&> K_0 + \sum_{i\in(W(R)\setminus\tilde{W}(R))} K_i
\end{aligned}$$

This contradicts the fact that the online algorithm holds at each time a feasible dual solution since the dual constraint that is suitable to the set of types $W(R) \setminus \tilde{W}(R)$ and the time $t = \tau_s(R)$ is violated. The online algorithm also maintains a feasible dual solution during the simulation step, so this holds also in the simulation step.

All constraints that became tight during the simulation steps made before the last order in the phase also became tight in the actual running of the algorithm and therefore are able to pay for their item ordering cost by the corresponding dual variables. The only problem is with constraints that became tight during the simulation step made in the last order of a phase. We can only make use of the values of the dual variables until the time of the last order. The main observation is that by the algorithm's behavior the extra budget that is used is at most K_0. Thus, if there are M orders during the phase we obtain

$$\begin{aligned}
\text{Order cost} \leq & \sum_{(i,t)\in D(W(R_{p,1}))} \sum_{s=\tau_s(p)}^{s<\tau_e(p)} y(i,t,s) + (M-1)K_0 \\
& + \sum_{R\in\mathbb{R}_p} \sum_{(i,t)\in D(\tilde{W}(R))} \sum_{s=\tau_s(R)}^{s<\min\{\tau_e(R),\tau_f(p)\}} y(i,t,s) + K_0 \qquad (3.10) \\
\leq\ & 2\sum_{s=\tau_s(p)}^{s<\tau_e(p)} \sum_{(i,t)\in D(W(R_{p,1}))} y(i,t,s) + \sum_{s=\tau_e(p)}^{s<\tau_f(p)} \sum_{(i,t)\in\mathbb{D}} y(i,t,s) \\
& + \sum_{R\in\mathbb{R}_p} \sum_{(i,t)\in D(\tilde{W}(R))} \sum_{s=\tau_s(R)}^{s<\min\{\tau_e(R),\tau_f(p)\}} y(i,t,s) \qquad (3.11)
\end{aligned}$$

The first term in inequality (3.10) bounds the joint ordering cost K_0 of the first order in the phase and the fixed ordering cost of each of the types in the first constraint that became tight. The second term in (3.10) is the joint ordering cost of all orders from the second to the last. The third term is the fixed ordering cost of all types that were ordered in the first order due to the simulation step and the fixed costs of types ordered in the second to last order of the phase. Note that we sum only until the minimum between the time each constraint became tight and the last order in the phase. Thus, we add an extra cost of K_0 of the extra future budget that was used. Inequality (3.11) follows directly from equation (3.8).

We next prove a useful claim that helps us proving our final result. An order of an item with type i at time s by our algorithm is always caused by some dual constraint R becoming tight. One option is that constraint R became tight at time $s \leq \tau < s+1$ and contained an active demand of type i. Another option is that during the simulation step done at the algorithmic time τ some constraint R became tight at time $\tau_e(R) > s$ and caused an order at time s. In the next Lemma we take two successive times t_1 and t_2 in which the algorithm makes an order to an arbitrary type i. We prove that the starting time $\tau_s(R)$ of the second constraint that caused an order of type i must be larger than t_1. Intuitively, this means that an order to type i "resets" all the constraints that contain type i.

LEMMA 3.5. *Let t_1, t_2 be two successive times in which the online algorithm makes an order for some type i. Let $\tau_s(R_2)$ be the beginning time of the second constraint that became tight (during the real time or during the simulation) and caused an order of type i at time $t_2 \leq \tau_e(R_2)$. Then $\tau_s(R_2) > t_1$*

Proof. The second constraint became tight in real time or during the simulation at time $\tau_e(R_2) \geq t_2$. This constraint must contain at time t_2 an active demand (i,t) with type i (otherwise, it doesn't cause an order

of type i). In addition, by the properties of the dual constraints the constraint only contains variables of demands that arrived prior to $\tau_s(R_2)$. Thus, demand (i,t) is active from time $\tau_s(R_2)$ until time t_2. At time $t_1 \leq t_2$ there was an order of type i, and so the algorithm changed the status of all the demands of type i that arrived prior to t_1 to either half active or inactive. Thus, it is not possible that $\tau_s(R_2) \leq t_1$.

The next lemma bounds the number of times each dual variable is used in Equation (3.7). This will allow us to bound the competitive ratio of the algorithm. Due to space constraints we defer the proof to the full version of the paper.

LEMMA 3.6. *The total number of times each $y(i,t,s)$ appears in Equation 3.7 at most twice if $s < \tau_e(i,t)$ and at most 3 times if $s \geq \tau_e(i,t)$.*

Combining Lemmas 3.3, 3.4 and 3.5 we obtain

THEOREM 3.1. *The algorithm is 3-competitive.*

4 Lower Bounds

Known lower bound techniques for the TCP acknowledgement problem do not extend readily to JRP and thus we required new ideas. In the full version of the paper we prove the following lower bounds.

THEOREM 4.1. *No deterministic online algorithm for the single-item case has competitive ratio less than 2.*

THEOREM 4.2. *No deterministic online algorithm for the 2-item case has competitive ratio less than 2.46.*

THEOREM 4.3. *No deterministic online algorithm for the 3-item case has competitive ratio less than 2.64.*

Acknowledgements: We thank Andreas Wächter for his great help in using his non-linear programming solver IPOPT [17] and Grzegorz Swirszcz for finding several errors in the exposition of the lower bound.

References

[1] N. Alon, B. Awerbuch, Y. Azar, N. Buchbinder, and J. Naor. The online set cover problem. In *Proceedings of the 35th annual ACM Symposium on the Theory of Computation*, pages 100–105, 2003.

[2] E. Arkin, D. Joneja, and R. Roundy. Computational complexity of uncapacitated multi-echelon production planning problems. *Operations Research Letters*, 8:61–66, 1989.

[3] Y. Askoy and S. S. Erenguk. Multi-item inventory models with coordinated replenishment: a survey. *International Journal of Operations and Production Management*, 8:63–73, 1988.

[4] Niv Buchbinder, Kamal Jain, and Joseph (Seffi) Naor. Online primal-dual algorithms for maximizing ad-auctions revenue. In *15th Annual European Symposium on Algorithms (ESA 2007)*, 2007.

[5] Niv Buchbinder and Joseph (Seffi) Naor. Online primal-dual algorithms for covering and packing problems. In *13th Annual European Symposium on Algorithms - ESA 2005*, 2005.

[6] Niv Buchbinder and Joseph (Seffi) Naor. Improved Bounds for Online Routing and Packing Via a Primal-Dual Approach. In *47th annual ieee Symposium on Foundations of Computer Science (FOCS 2006)*, 2006.

[7] N. P. Dellaert1 and M. T. Melo. Heuristic procedures for a stochastic lot-sizing problem in make-to-order manufacturing. *Annals of Operations Research*, 59(1):227–258, 2005.

[8] Daniel R. Dooly, Sally A. Goldman, and Stephen D. Scott. Tcp dynamic acknowledgment delay (extended abstract): theory and practice. In *STOC '98: Proceedings of the thirtieth annual ACM symposium on Theory of computing*, pages 389–398, 1998.

[9] M. X. Goemans and D. P. Williamson. A general approximation technique for constrained forest problems. *SIAM Journal on Computing*, 24:296–317, 1995.

[10] Qi-Ming He, E.M. Jewkes, and J. Buzacott. The value of information used in inventory control of a make-to-order inventory-production system. *IIE Transactions*, 34(11):999–1013, 2002.

[11] S. Van Hoesel and A. Wagelmans. A dual algorithm for the economic lot-sizing problem. *European Journal of Operatioms Research*, 52:315–325, 1991.

[12] K. Jain and V. V. Vazirani. Approximation algorithms for metric facility location and k-median problems using the primal-dual schema and Lagrangian relaxation. *Journal of the ACM 48*, pages 274–296, 2001.

[13] Anna R. Karlin, Claire Kenyon, and Dana Randall. Dynamic TCP acknowledgement and other stories about e/(e-1). In *ACM Symposium on Theory of Computing*, pages 502–509, 2001.

[14] Wei-Min Lan and Tava Lennon Olsen. Multiproduct systems with both setup times and costs: Fluid bounds and schedules. *Operations Research*, 54(3):505–522, 2006.

[15] R. Levi, R. O. Roundy, and D. B. Shmoys. Primal-dual algorithms for deterministic inventory problems. *Mathematics of Operations Research*, 31:267–284, 2006.

[16] R. Levi, R. O. Roundy, D. B. Shmoys, and M. Sviridenko. First constant approximation algorithm for the single-warehouse multi-retailer problem. To appear in Management Science, extended abstracts appeared in SODA 2005 and APPROX 2006., 2004.

[17] A. Wächter and L. T. Biegler. On the implementation of an interior-point filter line-search algorithm for large-scale nonlinear programming. *Mathematical Programming*, 106(1):25–57, 2006.

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