MIT Libraries | DSpace@MIT

MIT Open Access Articles

*The submodular joint replenishment problem*

The MIT Faculty has made this article openly available. ***Please share*** how this access benefits you. Your story matters.

**Citation:** Cheung, Maurice, Adam N. Elmachtoub, Retsef Levi, and David B. Shmoys. "The Submodular Joint Replenishment Problem." Mathematical Programming, July 3, 2015.

**As Published:** http://dx.doi.org/10.1007/s10107-015-0920-3

**Publisher:** Springer-Verlag

**Persistent URL:** http://hdl.handle.net/1721.1/98901

**Version:** Original manuscript: author's manuscript prior to formal peer review

**Terms of use:** Creative Commons Attribution-Noncommercial-Share Alike

Massachusetts Institute of Technology

**Mathematical Programming manuscript No.**
(will be inserted by the editor)

# The Submodular Joint Replenishment Problem

**Maurice Cheung** ·
**Adam N. Elmachtoub** ·
**Retsef Levi** ·
**David B. Shmoys**

Received: November 2013, December 2014, May 2015, June 2015

**Abstract** The joint replenishment problem is a fundamental model in supply chain management theory that has applications in inventory management, logistics, and maintenance scheduling. In this problem, there are multiple item types, each having a given time-dependent sequence of demands that need to be satisfied. In order to satisfy demand, orders of the item types must be placed in advance of the due dates for each demand. Every time an order of item types is placed, there is an associated joint setup cost depending on the subset of item types ordered. This ordering cost can be due to machine, transportation, or labor costs, for example. In addition, there is a cost to holding inventory for demand that has yet to be served. The overall goal is to minimize the total ordering costs plus inventory holding costs.

In this paper, the cost of an order, also known as a joint setup cost, is a monotonically increasing, submodular function over the item types. For this general problem, we show that a greedy approach provides an approximation guarantee that is logarithmic in the number of demands. Then we consider three special cases of submodular functions which we call the laminar, tree, and cardinality cases, each of which can model real world scenarios that previously have not been captured. For each of these cases, we provide a constant factor

Maurice Cheung
School of Operations Research and Information Engineering, Cornell University
E-mail: myc26@cornell.edu

Adam N. Elmachtoub
Department of Industrial Engineering and Operations Research, Columbia University
E-mail: adam@ieor.columbia.edu

Retsef Levi
Sloan School of Management, Massachusetts Institute of Technology
E-mail: retsef@mit.edu

David B. Shmoys
Department of Computer Science and School of Operations Research and Information Engineering, Cornell University
E-mail: david.shmoys@cornell.edu

approximation algorithm. Specifically, we show that the laminar case can be solved optimally in polynomial time via a dynamic programming approach. For the tree and cardinality cases, we provide two different linear programming based approximation algorithms that provide guarantees of three and five, respectively.

**Keywords** Inventory Management · Approximation Algorithm · Submodular Function · Joint Replenishment Problem

## 1 Introduction

Inventory models with deterministic and non-stationary demand have a long and rich history in supply chain management, beginning with the seminal paper of Wagner and Whitin (1958). In these models, there are demands for different item types that need to be satisfied before their respective due dates. Ordering inventory in a time period results in setup costs, and holding inventory before it is due results in holding costs. The main objective of these models is to minimize the total setup costs and holding costs to satisfy all the demand. Setup costs typically represent, among others, the use of machines, trucks, and/or laborers. When multiple item types are ordered in the same period, some of the ordering costs are typically shared among the different item types. In most inventory management models, the economies of scale are traditionally captured by a *joint setup cost* structure, and the problem is generally known as a *joint replenishment problem (JRP)*. The JRP has been used in many applications including inventory management, maintenance scheduling, logistics, and transportation problems.

The most traditional JRP model uses the *additive* joint setup cost structure. In this model, there is a one-time setup cost if *any* item type is ordered, in addition to an individual item setup cost for each item type ordered. The additive joint setup cost structure has limited modeling power in some significant practical cases, yet there is a surprising lack of results for models with more complex joint setup cost structures when demand is non-stationary. This in sharp contrast to Queyranne (1986), Federgruen and Zheng (1992), and Teo and Bertsimas (2001) who gave near-optimal algorithms for inventory models with a very general joint setup cost structure but with *constant* demand. In this work, we consider several variations and generalizations of the deterministic, *non-stationary* joint replenishment problem beyond the additive cost structure that capture many interesting settings in practice. Since joint replenishment problems with non-stationary demand are typically NP-hard, even for the additive model (Arkin et al. (1989)), it is computationally intractable to find optimal solutions quickly. However, we provide *approximation algorithms* that can efficiently find solutions provably close to optimal, and in the worst case are guaranteed to be within a small factor of the optimal cost. This factor is known as the *approximation ratio*.

In the JRP models studied in this paper, there are multiple item types, each with a sequence of demands over a discrete time horizon of finitely many periods. That is, for each item type there is a prespecified demand quantity due in each time period. Each demand must be completely satisfied by an order at or prior to its due date period, which means neither backlogging nor lost sales are allowed. For each time period, one needs to decide which item types to order, and how much to order. The joint setup cost, or ordering cost, is a function of the specific set of item types included in the order, and does not depend on the demand quantities ordered. For each unit of demand ordered, there is a holding cost that depends on the item type, the time it was ordered, and the due date of the demand point it will serve. Our holding cost structure is general enough to also capture additional per unit production costs when an order is placed. One can easily show that zero-inventory ordering policies are optimal in this model, and thus every demand of a given item type is satisfied by the latest order of that item type prior to its due date period. The goal is to satisfy all the demands on time by a sequence of orders that minimizes the total joint setup costs plus holding costs.

The joint setup cost function we study satisfies two natural properties known as *monotonicity* and *submodularity*. The monotonicity property simply means that as more item types are ordered, the total joint setup cost increases. The submodularity property captures the economies of scale in ordering more item types, i.e., the marginal cost of adding any specific item type to a given order decreases as the given order increases. Under these assumptions, we refer to this problem as the *submodular joint replenishment problem.* For the submodular JRP, we show that a simple greedy algorithm achieves an approximation ratio that is logarithmic in the total number of demands. In addition, we also describe an integer programming formulation whose linear programming relaxation can be solved efficiently and has a special structure for the optimal solution. We then consider three special cases of the submodular JRP with non-stationary demand that capture a wide range of applications. These cases are called the *tree JRP*, *laminar JRP*, and the *cardinality JRP*, none of which have any clear mathematical relation with the others.

In the *tree JRP* case, we are given a rooted tree where each node represents a process that incurs a setup cost, and the leaves of the tree represent the item types. The joint setup cost of ordering any subset of item types is the total setup costs of all the nodes on the paths from the root to the leaves corresponding to the item types being ordered. The tree JRP model captures situations where each item type requires a chain of processes to be performed, and several of those processes are shared by other item types. One application of the tree JRP is in maintenance scheduling problems (Levi et al. (2011)) for aircraft engines. Each module of the engine corresponds to a node in the tree, and to get to a certain engine part requires removing all necessary modules. The tree with only a root and leaves is identical to the additive joint setup cost structure, and thus this problem is also NP-hard. We provide an approximation algorithm that gives a solution guaranteed to be no more than three times the optimal cost. Specifically, the algorithm is based on solving a linear program, and then successively rounding the variables corresponding to the nodes in the tree in a particular fashion. Specifically, we start by opening orders containing the root process, and work our way down the tree using a breadth-first search. For each node, we round the corresponding variables to a time where there is an order containing the process corresponding to its parent node, thus ensuring that all necessary processes for an item type are accounted for when it is ordered.

In the *laminar JRP* case, the joint setup cost function is specified through a laminar family of subsets of item types, each with an associated setup cost. The family can be modeled as a tree where the nodes correspond to subsets of item types, and the children of each node must partition the corresponding subset of item types belonging to that node. This captures situations where there are machines (or laborers) with varying degrees of specialization. Any two machines have the property that either they cannot make any of the same item types, or that one machine has strictly more capabilities than the other. The joint setup cost for ordering a subset of item types is simply the setup cost of the cheapest collection of machines that is capable of producing all the item types being ordered. For the laminar JRP, we are surprisingly able to solve the problem to optimality with an efficient dynamic program. This implies that this variant of the problem belongs to the class of problems P. The subproblems of the dynamic program are finding the optimal cost of serving the demands in a specific time interval, given that a specific machine will be in use at the beginning of the interval. The correctness relies on our ability to decompose the item types into groups based on which machines they can be processed on.

Finally, the *cardinality JRP* is the case where the joint setup cost function has the property that the cost of ordering a subset of item types is simply a function of the cardinality of the subset of item types being ordered. The submodularity in this case implies that the joint setup cost function is concave in the cardinality of the subset of item types being ordered. A natural application of this model is when all the item types are very similar, but vary in only one aspect, such as color or size. Although the cardinality JRP is NP-hard, which was shown indirectly in Arkin et al. (1989), we provide an efficient algorithm with a worst-case approximation ratio of five. This algorithm is based on an innovative iterative rounding procedure that uses the variables from a linear relaxation of a novel integer programming formulation of the cardinality JRP. Our algorithm carefully builds up orders based on their size, while ensuring that the cost of any particular order can be paid for using the primal objective costs. The holding costs are accounted for using a property of the respective dual linear program. Since the linear programming relaxation is of exponential size, we also describe a nontrivial yet equivalent linear program of a polynomial size.

## 1.1 Literature Review

Joint replenishment problems are infamous for being intractable, and thus have been typically studied via the notion of *approximation algorithms.* When demand is assumed to be stationary and continuous, the additive JRP was shown by Schulz and Telha (2011) to be as hard as integer factorization. For this same problem, Roundy (1985) showed that "power-of-two" policies have approximation ratios of 1.06 and 1.02, depending on whether the base planning period is fixed or not. In Teo and Bertsimas (2001), an improved approximation ratio of 1.04 was obtained for the additive JRP with fixed base planning periods. When the time horizon is finite, Segev (2013) and Nonner and Sviridenko (2013) provide quasi-polynomial and efficient polynomial time approximation schemes, respectively. In Federgruen and Zheng (1992), the results of Roundy (1985) were generalized to the submodular JRP with constant demand. Other stationary inventory models with submodular costs were considered in Federgruen et al. (1992), Herer and Roundy (1997), and Viswanathan (2007).

The literature for the submodular JRP with non-stationary demand has focused primarily on the additive joint setup cost structure, which was shown to be NP-hard in Arkin et al. (1989). Nonner and Souza (2009) further showed that this problem is APX-hard when holding costs are nonlinear with respect to time, which is the case for the models we consider. Several heuristics for the non-stationary additive JRP have been proposed with varying degrees of theoretical performance guarantees in Veinott Jr (1969), Zangwill (1969), Kao (1979), Joneja (1990), Federgruen and Tzur (1994), Levi et al. (2006), and Stauffer et al. (2011). The current best approximation algorithms for the additive JRP with non-stationary demand are due to Levi et al. (2008) and Bienkowski et al. (2013), which have approximation ratios of 1.80 and 1.8791, respectively. Becchetti et al. (2006) and Khanna et al. (2002) have considered special cases of the tree JRP with assumptions on the holding cost structure. Chan et al. (2002) show that zero-inventory policies are near-optimal for joint replenishment problems with piecewise linear costs, however their conditions do not imply submodularity.

Since this paper has been made available, there have been several related results. For the submodular JRP, Shmoys and Tong (2013) provide an algorithm with an approximation ratio that is logarithmic in the number of item types. Also for the submodular JRP, Nagarajan and Shi (2015) provide an algorithm with an approximation ratio that is logarithmic in the number of time periods. Under certain assumptions on the holding costs, they show that their approximation ratio is sublogarithmic. For the tree JRP, Pedrosa and Sviridenko (2013) provide an algorithm with approximation ratio of two.

In Section 2, we give a precise mathematical description of the submodular JRP along with a greedy approximation algorithm and an integer programming formulation. We then consider the tree, laminar, and cardinality JRPs in Sections 3, 4, and 5. Finally, we offer concluding remarks in Section 6.

## 2 Submodular JRP

### 2.1 Model

In this section, we formally describe the submodular joint replenishment problem. There are $N$ different types of items, or item types, each associated with a known sequence of demands over a discrete finite planning horizon of $T$ periods. These items types are denoted by the the set $\mathcal{N} := \{1, \ldots, N\}$. The demand in a specific period $t \in \{1, \ldots T\}$ for an item type $i \in \mathcal{N}$ is denoted by $d_{it} \geq 0$. Let the set $\mathcal{D}$ contain all the demand points $(i,t)$ with $d_{it} > 0$. Each demand point $(i,t) \in \mathcal{D}$ needs to be served by an order containing item type $i$ placed no later than its due date $t$. There are no backlogging or lost sales. In every period one must decide which item types to order, and how much of each item type to order. The total cost of a solution is composed of the joint setup costs for placing an order, and the holding costs for holding inventory before demand is due.

The joint setup cost of an order is a function of the item types ordered, and does not depend on the demand quantities ordered. For any given time period and a subset of item types $S \subseteq \mathcal{N}$, we let $K(S)$ denote the joint setup cost of ordering demand for item types in $S$ in that time period. $K(\cdot)$ is assumed to be nonnegative, monotonically increasing, and submodular. The monotonicity assumption means that for every $S_1 \subseteq S_2 \subseteq \mathcal{N}$, $K(S_1) \leq K(S_2)$. The submodularity assumption means that for every pair of sets $S_1, S_2 \subseteq \mathcal{N}$, $K(S_1) + K(S_2) \geq K(S_1 \cup S_2) + K(S_1 \cap S_2)$. This definition can be shown to be equivalent to the following: for every set $S_1 \subseteq S_2 \subseteq \mathcal{N}$ and any item type $i \in \mathcal{N}$, $K(S_1 \cup \{i\}) - K(S_1) \geq K(S_2 \cup \{i\}) - K(S_2)$. This alternative definition conveys more clearly the economies of scale interpretation of submodularity, i.e., that the additional cost of adding an item type to a given order is decreasing as more item types are included in the given order. Finally we let $h^i_{st}$ be the cost of holding a unit of item type $i$ from period $s$ to $t$. Thus, if demand point $(i,t)$ is served by an order at time $s$, then this this incurs a holding cost of $d_{it}h^i_{st}$, which we conveniently denote by $H^i_{st}$. If $d_{it} = 0$, then $H^i_{st} = 0$ as well. We note that $h^i_{st}$ also allows us to easily capture a per unit production cost for ordering a unit of item $i$ in period $s$. We assume $h^i_{st}$ is nonnegative and monotonically decreasing in $s$ for a fixed $i$ and $t$, which means that holding inventory longer is always more costly. This assumption implies that any demand point $(i,t)$ is satisfied by the latest order before or at time $t$ containing item type $i$, i.e., zero-inventory policies are optimal. Thus given a sequence of orders, the inventory allocation and holding costs can be computed in a straightforward fashion. The goal is to satisfy all the demands on time by a sequence of orders that minimizes the total joint setup costs plus holding costs.

In Theorem 1, we provide an $O(\log(NT))$-approximation algorithm for the submodular JRP. The proof relies on a reduction to the set cover problem, which is similar in spirit to a reduction described in Svitkina and Tardos (2010) for a facility location type problem.

**Theorem 1** *There exists an $O(\log(NT))$- approximation algorithm for the submodular JRP using the greedy algorithm for the set cover problem.*

*Proof* We reduce the submodular JRP to the set cover problem. In the set cover problem, there are $m$ objects that need to be covered, and $n$ subsets of those objects that each have a different cost. The goal is to cover the $m$ objects using the cheapest collection of available subsets. Chvatal (1979) showed that a simple greedy algorithm is an $O(\log m)$-approximation algorithm to this problem. The algorithm works by iteratively choosing the subset that has the smallest ratio of its cost over the number of uncovered objects in the subset.

To model the submodular JRP as a set cover problem, we let each demand point $(i,t)$ be an object, which means there are at most $NT$ in total. Recall that $\mathcal{D}$ is the set of all demand points with a positive demand. The collection of possible subsets we may use in the cover is denoted by $\mathcal{U} := \{1,\ldots,T\} \times 2^{\mathcal{D}}$. The cost of a subset $(s,U) \in \mathcal{U}$, denoted by $c(s,U)$, is simply the cost of serving the demand points in $U$ from time $s$ in the submodular JRP. More specifically, $c(s,U) = \infty$ if $U$ contains a demand point with due date before $s$. Otherwise, if we let $I(U)$ denote the subset of item types in $U$, then $c(s,U) = K(I(U)) + \sum_{(i,t)\in U} H^i_{st}$. Notice that $c(s,U)$ is a submodular function in $U$ for a fixed $s$ due to the submodularity of $K(\cdot)$ and the fact that the holding costs are separable.

Now we need to show that we can find a set $(s,U) \in \mathcal{U}$ whose ratio of $c(s,U)$ over the number of uncovered demand points is smallest. Since the number of such possible sets is exponential, we cannot enumerate and thus need to define an efficient procedure. To find the set to add to the cover, we first find the set that has the smallest ratio for each time period $s$, and then choose the cheapest among them. Let $w_{(i,t)} = 1$ if demand point $(i,t)$ is still uncovered and let it be 0 otherwise. For a specific time period $s$, we aim to find a set $U \in 2^{\mathcal{D}}$ that minimizes $\frac{c(s,U)}{\sum_{(i,t)\in U} w_{(i,t)}}$. This optimization problem can be solved by finding the minimum $\alpha$ for which there exists a set $U \in 2^{\mathcal{D}}$ such that $\frac{c(s,U)}{\sum_{(i,t)\in U} w_{(i,t)}} \leq \alpha$. Equivalently, we can write this inequality as $c(s,U) - \alpha \sum_{(i,t)\in U} w_{(i,t)} \leq 0$. Since the left hand side of this expression is clearly submodular, we can find a set $U$, if it exists, that satisfies this inequality for a given $\alpha$ by solving a submodular minimization problem, which is known to be efficiently solvable (Orlin (2009)). Doing a binary search over the possible values of $\alpha$ allows us to find the minimum $\alpha$ efficiently, up to an arbitrarily close constant factor. (Note that this increases the overall approximation guarantee by the same constant factor.) Therefore, we can run the greedy algorithm efficiently and obtain an $O(\log(NT))$-approximation algorithm. $\square$

## 2.2 Integer Programming Formulation

The following is an integer programming formulation of the submodular JRP. The binary variable $y^S_s$ is 1 if the subset of item types $S$ is ordered in period

$s$ and is 0 otherwise. The binary variable $x^i_{st}$ is 1 if demand $(i,t)$ is satisfied using an order from period $s$ and is 0 otherwise.

$$\begin{aligned}
\text{minimize} \quad & \sum_{S\subseteq\mathcal{N}}\sum_{s=1}^{T} K(S)y_s^S + \sum_{(i,t)\in\mathcal{D}}\sum_{s=1}^{t} H^i_{st}x^i_{st} && && \text{(IP)}\\
\text{subject to} \quad & \sum_{s=1}^{t} x^i_{st} = 1, && (i,t)\in\mathcal{D} && (1)\\
& x^i_{st} \le \sum_{S:i\in S\subseteq\mathcal{N}} y_s^S, && (i,t)\in\mathcal{D}, s=1,\dots t && (2)\\
& x^i_{st}, y_s^S \in \{0,1\}, && (i,t)\in\mathcal{D}, s=1,\dots t, S\subseteq\mathcal{N}
\end{aligned}$$

We first argue that this is indeed a valid formulation for the submodular JRP. Constraint (1) ensures that each demand $(i,t)$ with $d_{it}>0$ is completely served by an order at some time $s\le t$. Constraint (2) ensures that if any demand $(i,t)$ is served by an order at time $s$, then there must be a subset ordered at time $s$ that includes $i$. By submodularity, the optimal solution only sets at most one $y$ variable to 1 in any time period $s$. The natural LP relaxation of (IP), denoted by (P), relaxes the integer constraints on the variables to nonnegativity constraints.

The following lemma shows that there exists an optimal solution to (P), such that in any given time period $s$, the set of $y_s$ variables that are positive have a very special nested structure. Using this lemma, one can easily show that for instances of the tree and cardinality JRPs, (P) is is as strong as the polynomial-size LP relaxations that we construct for the tree and cardinality JRPS in Sections 3 and 5, i.e., (P-T) and (P-C*). Moreover, this lemma may be useful in future results on the submodular JRP.

**Lemma 1** *There exists an optimal solution $(x,y)$ to (P) such that for any given period $s$ and any two subsets $R\subseteq\mathcal{N}$ and $S\subseteq\mathcal{N}$, if $y_s^R>0$ and $y_s^S>0$, then $R$ and $S$ are nested, i.e., either $R\subseteq S$ or $S\subseteq R$.*

*Proof* Given an arbitrary optimal solution $(\hat{x},\hat{y})$ to (P), we construct another optimal solution $(x,y)$ with the desired property. Fix a time period $s$. Let $Z^i_s := \sum_{S:i\in S}\hat{y}^S_s$ be the sum of the fractional number of sets ordered in period $s$ that contains item type $i$. Now consider the following auxiliary optimization problem which has a variable $r^S_s$ for every set $S\subseteq\mathcal{N}$.

$$\begin{aligned}
\text{minimize} \quad & \sum_{S\subseteq\mathcal{N}} K(S)r^S_s && && \text{(AUX-s)}\\
\text{subject to} \quad & \sum_{S:i\in S} r^S_s \ge Z^i_s, && i=1,\dots,N && (3)\\
& r^S_s \ge 0 && S\subseteq\mathcal{N}
\end{aligned}$$

Recall $K(S)$ is the joint setup cost of ordering the item types in $S$. Thus (AUX-s) exactly captures the cheapest way to order the item types such that the amount of $i$ ordered is at least $Z_s^i$. Since $K(S)$ is a submodular function, (AUX-s) is the dual of a polymatroid where the greedy algorithm finds an optimal solution (see Bertsimas and Weismantel (2005)). Specifically, assume without loss of generality that the items are indexed such that $Z_s^1 \geq Z_s^2 \geq \ldots \geq Z_s^N$ and let $S(i) := \{1, \ldots, i\}$. Then an optimal solution to (AUX-s) is given by $r_t^{S(N)} = Z_t^N$, and $r_s^{S(i)} = Z_s^i - Z_s^{i+1}$ for $i = 1, \ldots, N-1$, and $r_s^S = 0$ for all other $S$. Note that by construction $S(i)$ has the desired nested property. Thus, setting $x = \hat{x}$ and $y_s^S = r_s^S$, where $r_s^S$ is the optimal solution to (AUX-s), gives a feasible solution to (P) where every demand is served in exactly the same manner. Thus, the holding costs are identical for the two solutions, and the joint setup costs of $(x, y)$ are no more than that of $(\hat{x}, \hat{y})$ since $\hat{y}_s^S$ gives a feasible solution to (AUX-s). Therefore, $(x, y)$ is also an optimal solution, and in addition satisfies the desired nested property. $\square$

Now we consider the dual program of (P). Let $B_t^i$ and $L_{st}^i$ be the dual variables corresponding to constraints (1) and (2) in (P), respectively. Then the dual program of (P) is

$$\begin{aligned}
\text{maximize} \quad & \sum_{(i,t)\in\mathcal{D}} B_t^i && && \text{(D)}\\
\text{subject to} \quad & B_t^i \leq H_{st}^i + L_{st}^i, && (i,t)\in\mathcal{D}, s = 1,\ldots,t && (4)\\
& \sum_{i\in S}\sum_{t=s}^{T} L_{st}^i \leq K(S), && s = 1,\ldots,T, S \subseteq \{1,\ldots,N\} && (5)\\
& L_{st}^i \geq 0 && (i,t)\in\mathcal{D}, s = 1,\ldots,t &&
\end{aligned}$$

Note that (P) contains an exponential number of variables and (D) contains an exponential number of constraints. One can solve the dual using the ellipsoid method, since there is an efficient separation oracle finding violated constraints (if there are any) using submodular function minimization. From the dual solution, a primal solution can then be found in polynomial time (see Corollary 14.1g(v) in Schrijver (1998)). In Sections 3 and 5 we focus on special cases of the submodular JRP that have polynomial-size LP relaxations. This also allows us to give efficient LP-based approximation algorithms for these cases.

## 3 Tree Joint Replenishment Problem

In this section, we consider the tree JRP, where the joint setup cost structure is represented by a tree $\mathcal{T}$ with a root $r$. Specifically, there is a setup cost $K_j > 0$ associated with each node $j$ in the tree, and each item type $i$ corresponds to a leaf in the tree. (Nodes with zero cost are avoided to simplify the discussion,

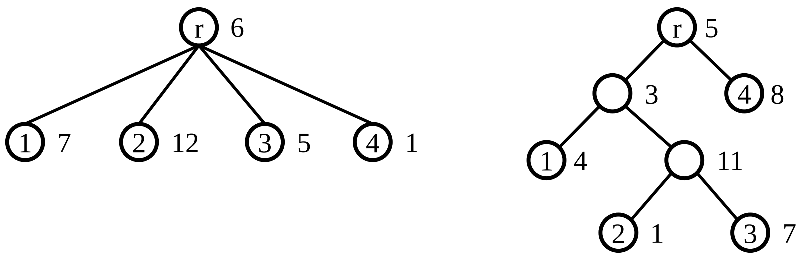

**Fig. 1** These two trees represent two examples of the tree joint setup cost structure. The leaves are labeled according to the item type they represent and the root node is labeled with an $r$. The number next to each node denotes the cost of that node. The left tree belongs to the special case of an additive joint setup cost structure. In the right tree, the cost of ordering item types 1 and 3 is 5+3+4+11+7 = 30.

but can easily be incorporated.) For every node $j \in \mathcal{T}$, let $path(j)$ denote the unique path from node $j$ to the root of the tree. The joint setup cost of ordering a subset of item types $S \subseteq \mathcal{N}$ is then equal to $\sum_{j \in \cup_{i \in S} path(i)} K_j$, i.e., $K(S)$ is the setup cost of all nodes that belong to the paths from $r$ to the leaves corresponding to the set $S$. (See Figure 1 for an example.)

We note that the tree JRP generalizes the additive JRP since the additive joint setup cost structure can be viewed as a tree. Specifically, there is a root node connected to $N$ leaves, one for each item type. The cost of the root node is $K_r$ and the cost of each leaf $i$ is $K_i$. In the remainder of this section, we advance the ideas of Levi et al. (2008) to describe a 3-approximation algorithm for the tree-JRP via LP rounding.

The following is a natural IP formulation for the tree JRP. We define the binary variable $y_s^j$ to be 1 if the cost of node $j$ is incurred in period $s$, and 0 otherwise. For convenience, we extend the notion of ordering item types to ordering nodes in the tree, so $y_s^j = 1$ if node $j$ is ordered in period $s$. The binary variable $x_{st}^i$ is 1 if demand $(i,t)$ is served from period $s$ and 0 otherwise.

$$\text{minimize} \quad \sum_{j \in \mathcal{T}} \sum_{s=1}^{T} K_j y_s^j + \sum_{(i,t) \in \mathcal{D}} \sum_{s=1}^{t} H_{st}^i x_{st}^i \qquad \text{(IP-T)}$$

$$\text{subject to} \sum_{s=1}^{t} x_{st}^i = 1, \quad (i,t) \in \mathcal{D} \qquad (6)$$

$$x_{st}^i \le y_s^j, \qquad (i,t) \in \mathcal{D}, s = 1, \dots t, j \in path(i) \qquad (7)$$

$$x_{st}^i, y_s^j \in \{0,1\}, \quad (i,t) \in \mathcal{D}, s = 1, \dots t, j \in \mathcal{T}$$

The correctness of (IP-T), and in particular for constraint (7), follows from the fact that in order to place an order for item $i$, one has to pay $\sum_{j \in path(i)} K_j$. If we relax the binary constraints in (IP-T) to nonnegativity constraints, this gives us the LP relaxation which we denote by (P-T).

Next we describe the dual of (P-T), which is used in the analysis of our LP rounding algorithm. Let $B_t^i$ and $L_{st}^{ij}$ be the dual variables corresponding to constraints (6) and (7) in (P-T) respectively. The dual of the (P-T) is then

$$
\begin{aligned}
\text{maximize}\quad & \sum_{(i,t)\in\mathcal{D}} B_t^i && && \text{(D-T)}\\
\text{subject to}\quad & B_t^i \le H_{st}^i + \sum_{j\in path(i)} L_{st}^{ij}, && (i,t)\in\mathcal{D}, s=1,\dots,t && (8)\\
& \sum_{i:j\in path(i)} \sum_{t=s}^{T} L_{st}^{ij} \le K_j, && s=1,\dots,T, j\in\mathcal{T} && (9)\\
& L_{st}^{ij} \ge 0, && (i,t)\in\mathcal{D}, s=1,\dots,t, j\in\mathcal{T} &&
\end{aligned}
$$

## 3.1 The LP Rounding Algorithm

We now show how to round an optimal solution to (P-T), denoted by $(x, y)$ to a feasible solution to the tree JRP problem with cost of at most 3 times the optimal value of (P-T), thus obtaining a 3-approximation algorithm. Our rounding procedure considers the nodes in the rooted tree one at a time, starting at the root node. The nodes can be processed in any order, as long as node $j$ is processed last among all nodes in $path(j)$ (i.e., the nodes on the path from $j$ to the root). Hence one can use, for example, a breadth first search starting from the root node.

We first describe the processing of the root node, where we decide in which time period to place orders. The rounding procedure is based on the values of $y_1^r, \dots, y_T^r$, which are the variables corresponding to fractional orders of the root node in (P-T). We place an order of the root node $r$ in period $s$ if $(\sum_{u=1}^{s-1} y_u^r, \sum_{u=1}^{s} y_u^r]$ contains an integer. Now for each node $j$, assume we have already processed all other nodes in $path(j)$, including the parent node $j'$. We place a *tentative order* in period $s$ for node $j$ if $(\sum_{u=1}^{s-1} y_u^j, \sum_{u=1}^{s} y_u^j]$ contains an integer. Motivated by the fact that we can only order $j$ if $j'$ has been ordered, we place actual orders of $j$ via a two-sided push procedure as follows. If there is a tentative order of $j$ at period $s$, then place an actual order of $j$ at the first order point of $j'$ in $(s, T]$ and place another actual order of $j$ at the latest order point of $j' \in [1, s]$ (if such orders of $j'$ exist). Notice that by construction, the orders are synchronized. See Figure 2 for an example of the rounding procedure. The pseudo-code for the algorithm is given below as Algorithm 1.

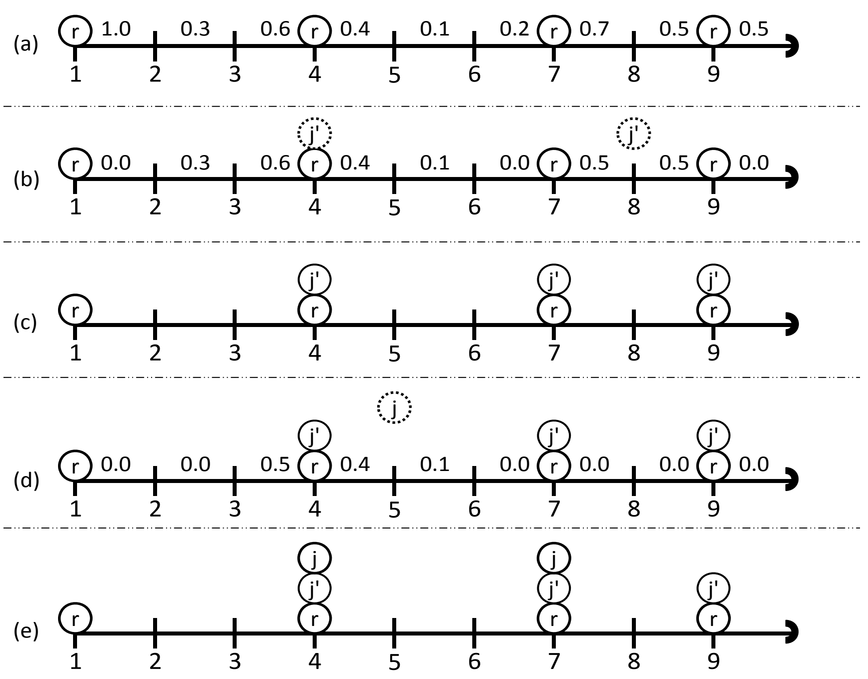

**Fig. 2** This figure demonstrates our LP rounding procedure for the first three successive nodes of the tree, which we call node r (root), node $j'$, and node $j$. There are 9 time periods in this example. In (a), each circle denotes an order placed for the root node. The values on the intervals correspond to the values of $y_t^r$, i.e., $y_1^r = 1.0$ and $y_2^r = 0.3$. In (b), each dotted circle denotes a tentative order placed for node $j'$. The values on the intervals correspond to the values of $y_t^{j'}$. In (c), we do a two sided push for the tentative orders of $j'$, which results in actual orders of $j'$ at periods 4, 7, and 9. In (d), the dotted circle denotes a tentative order placed for node $j$. The values on the intervals correspond to the values of $y_t^j$. In (e), we do a two sided push for the tentative orders of $j$, which results in actual orders of $j$ at periods 4 and 7.

## 3.2 Analysis

We first prove a structural lemma on the algorithm which is a key for the subsequent performance analysis. We refer to $(x, y)$ as the optimal solution of (P-T) for the remainder of the section.

**Lemma 2** *For each node $j$ and time interval of periods $[s, t]$ where $\sum_{u=s}^{t} y_u^j \geq 1$, the algorithm places an order for node $j$ somewhere in $[s, t]$.*

*Proof* We first describe simple properties of the optimal solution $(x, y)$. For a given node $j$ in the tree that is not a leaf, let $C(j)$ be the set of children of $j$. Then for each time period $u$, we know $y_u^j = \max_{k \in C(j)} y_u^k$ or otherwise we can lower the objective cost by decreasing $y_u^j$ while remaining feasible. It then

**Algorithm 1:** LP Rounding Algorithm for the tree JRP

```
Solve (P-T) for $(x, y)$
For each possible $s$, place an order of the root node in period $s$ if
$(\sum_{u=1}^{s-1} y_u^r, \sum_{u=1}^{s} y_u^r]$ contains an integer

// Process in breadth first search order
for each node $j$ do
    For each possible $s$, place a tentative order of the node $j$ in period $s$
    if $(\sum_{u=1}^{s-1} y_u^j, \sum_{u=1}^{s} y_u^j]$ contains an integer

    for each time period $s$ that contains a tentative order of $j$ do
        if $\exists$ an order of $parent(j)$ in $(s, T]$ then
            Place an order for node $j$ at the earliest order point of
            $parent(j)$ in $(s, T]$
        if $\exists$ an order of $parent(j)$ in $[1, s]$ then
            Place an order for node $j$ at the latest order point of
            $parent(j)$ in $[1, s]$
Serve each demand point $(i, t)$ from the latest order up to time $t$ that
includes item type $i$
```

immediately follows that for each pair of nodes $j$ and $j'$ such that $j' \in path(j)$, and any time period $u$, we have that $y_u^j \leq y_u^{j'}$. Now we prove the lemma by induction on $path(j)$, starting from the root.

*Base case (root node):* For any time interval $[s, t]$ where $\sum_{u=s}^{t} y_u^r \geq 1$, the interval $(\sum_{u=1}^{s-1} y_t^r, \sum_{u=1}^{t} y_t^r]$ must contain an integer. By construction of the algorithm, an order for the root node must then exist somewhere in $[s, t]$.

*Inductive case:* Consider node $j$ and any time interval $[s, t]$ where $\sum_{u=s}^{t} y_u^j \geq 1$. Let $j'$ be the parent of $j$. Since $y_u^{j'} \leq y_u^j$ for any $u$, then clearly $\sum_{u=s}^{t} y_u^{j'} \geq 1$. Using the induction hypothesis, we know there is an order for node $j'$ in $[s, t]$, namely at period $u'$. Also, by a similar reasoning as in the base case, we know there is a tentative order for node $j$ in $[s, t]$, namely at period $u$. We assume $u'$ was chosen to be the closest order of $j'$ to $u$ that is in $[s, t]$. Now since the algorithm opens order of $j$ by a two sided push to the two closest orders of $j'$ in opposite directions, then $j$ will be ordered at $u'$ as well. Thus we have proven the inductive case and the claim follows. □

The correctness of the algorithm follows from Lemma 2 above. Specifically, for each demand point $(i, t)$, it follows from constraints (6) and (7) in (P-T) that $\sum_{s=1}^{t} y_s^i \geq 1$. By Lemma 2, there exists a time in $[1, t]$ where all the nodes in $path(i)$ are ordered. This implies that the solution is indeed feasible. Next we analyze the cost of the solution produced by the algorithm. We start by considering the ordering cost. Since the number of orders made is at most

$\lfloor\sum_{u=1}^{T} y_u^r\rfloor$ for the root node and $2\lfloor\sum_{u=1}^{T} y_u^j\rfloor$ for all other nodes $j$ (we make up to two orders for every tentative order of $j$), the next lemma follows directly.

**Lemma 3** *The total ordering cost for the solution by Algorithm 1 is at most* $2\sum_{s=1}^{T}\sum_{j\in\mathcal{T}} K_j y_s^j$.

Finally we analyze the holding cost incurred by the solution constructed by the algorithm. We show that the total holding cost incurred is at most $\sum_{(i,t)\in\mathcal{D}} B_t^i$, the optimal value of (D-T).

**Lemma 4** *The total holding cost for the solution by Algorithm 1 is at most* $\sum_{(i,t)\in\mathcal{D}} B_t^i$.

*Proof* For any demand point $(i,t)$, consider the set of orders $s$ that serve $(i,t)$ fractionally in the optimal solution for (P-T), i.e., $x_{st}^i > 0$. Let $s_1$ be the earliest of such orders and we define $[s_1,t]$ as the *active interval* for demand $(i,t)$, specifically $x_{s_1t}^i > 0$ and $\sum_{s=s_1}^{t} x_{st}^i = 1$. Since $x_{s_1t}^i > 0$, by the complementary slackness conditions, the corresponding dual constraint must be tight, i.e., $B_t^i = H_{s_1t}^i + \sum_{j\in path(i)} L_{s_1t}^{ij}$. This expression, combined with the nonnegativity constraints on $L_{s_1t}^{ij}$, implies that $B_t^i \geq H_{s_1t}^i$. However, we also assume that the holding cost $H_{st}^i$ is monotonically decreasing in $s$. It follows that for any time $s$ in the active interval, we have that $B_t^i \geq H_{st}^i$. Hence, it suffices to show that there exists an order for $i$ in the active interval for $(i,t)$. However, by the definition of active interval and constraints (6) and (7) in (P-T), we have that $\sum_{s=s_1}^{t} y_s^i \geq 1$. Thus, using Lemma 2 for the interval $[s_1,t]$ shows that there exists an order of item $i$ in the active interval of $(i,t)$, as desired. $\square$

By Lemmas 3 and 4, and the fact that the optimal value of (P-T) and (D-T) are both lower bounds on the value of the optimal solution to the tree JRP, we obtain a 3-approximation for the Tree JRP as stated in Theorem 2.

**Theorem 2** *Algorithm 1 is a 3-approximation algorithm for the tree JRP.*

## 4 Laminar Joint Replenishment Problem

In this section, we study the laminar joint replenishment problem. In this setting, the joint setup cost function corresponds to a laminar family. A laminar family $\mathcal{F}$ is a collection of subsets of $\{1,\ldots,N\}$ such that for any $S_1, S_2 \in \mathcal{F}$, either $S_1 \cap S_2 = \emptyset$, $S_1 \subseteq S_2$, or $S_2 \subseteq S_1$. Each subset $F \in \mathcal{F}$ represents a machine (or laborer) that can produces the item types in $F$. The laminar JRP captures situations where any two machines (subsets) either produce completely different item types, or one machine is strictly more capable than the other. The joint setup cost of ordering any subset (machine) $F \in \mathcal{F}$ is $\kappa_F \geq 0$, i.e., $\kappa_F$ is the cost of using the machine corresponding to $F$. In the laminar JRP, the cost of ordering a set of item types $S$ in any time period is the cost of the cheapest collection of subsets in $\mathcal{F}$ whose union contains $S$. In Lemma 5 below, we show that the laminar JRP corresponds to a special case of the submodular JRP.

**Lemma 5** *The laminar JRP is a special case of the submodular JRP.*

*Proof* We need to show that the joint setup cost structure for the laminar JRP is monotonic and submodular. The joint setup cost $K(S)$ of ordering a subset of item types $S$ in the laminar JRP corresponds to the cost of the cheapest collection of machines that are capable of producing all the item types in $S$. From this definition, monotonicity follows immediately.

We prove submodularity by showing that for any $S_1, S_2 \subseteq \mathcal{N}$, $K(S_1) + K(S_2) \geq K(S_1 \cup S_2) + K(S_1 \cap S_2)$. Let $F_1^1, \ldots, F_{i_1}^1$ be the collection of machines that corresponds to ordering $S_1$, i.e., $K(S_1) = K_{F_1^1} + \ldots + K_{F_{i_1}^1}$. Similarly, let $F_1^2, \ldots, F_{i_2}^2$ correspond to the machines used to order $S_2$. Now we show that the collection of machines $C = \{F_1^1, \ldots, F_{i_1}^1, F_1^2, \ldots, F_{i_2}^2\}$, including duplicates, can be allocated to satisfy orders of $S_1 \cup S_2$ and $S_1 \cap S_2$, and thus provide an upper bound on $K(S_1 \cup S_2) + K(S_1 \cap S_2)$. We allocate the machines in decreasing order of size, with ties broken arbitrarily. Let $F$ be the current largest cardinality machine in $C$ that has yet to be allocated. If there is an item type in $F$ that is not currently covered by the collection for $S_1 \cup S_2$, then allocate $F$ to the collection for $S_1 \cup S_2$. Otherwise, allocate $F$ to the collection for $S_1 \cap S_2$.

By construction, the collection for $S_1 \cup S_2$ covers all item types covered by $C$, and therefore is a feasible cover for $S_1 \cup S_2$. Now consider an item type $j \in S_1 \cap S_2$. We know there must exist machines $F_{j_1}^1$ and $F_{j_2}^2$ that contain item type $j$. By the laminar property, either $F_{j_1}^1 \subseteq F_{j_2}^2$ or $F_{j_2}^2 \subset F_{j_1}^1$. By construction, we know that when the smaller of the two sets was processed, it must have been allocated to the collection for $S_1 \cap S_2$. Thus the allocation procedure correctly partitions $C$ into disjoint covers for $S_1 \cup S_2$ and $S_1 \cap S_2$. □

A laminar family can be conveniently represented by a tree graph such as in Figure 3, where every node corresponds to a subset $F \in \mathcal{F}$. The graph can be constructed iteratively in the following manner. First, create a node for every subset $F \in \mathcal{F}$ that contains only 1 item type. Then, for $k = 2, \ldots, N$, create a node for every subset $F \in \mathcal{F}$ of size $k$, and let this be a parent for all subgraphs (subtrees) whose root currently corresponds to item types belonging in $F$. Without loss of generality, we can always convert a laminar family to a new laminar family that is represented by a *connected binary* tree graph.

Transforming the tree graph of the original laminar family into a connected binary graph for a new laminar family can be accomplished without affecting $K(S)$. Given the original tree graph, we merge two disjoint subgraphs with roots $F_1$ and $F_2$ via a new subset $F = F_1 \cup F_2$ with cost $\kappa_F = \kappa_{F_1} + \kappa_{F_2}$, that serves as the parent of both subtrees. If a node $F_0$ has more than 2 children, we take two children $F_1$ and $F_2$ and create a new subset $F = F_1 \cup F_2$ with cost $\kappa_F = \kappa_{F_1} + \kappa_{F_2}$ that is the parent of both $F_1$ and $F_2$ and child of $F_0$. If a node $F_0$ has only 1 child $F_1$, we create a new subset $F = F_0 \backslash F_1$ with cost $\kappa_F = \kappa_{F_0}$ that becomes the second child of $F_0$. It is easy to see that repeating these three operations iteratively transforms any graph into a connected binary tree graph that corresponds to an equivalent cost structure. See Figure 3 for an example of a converted graph.

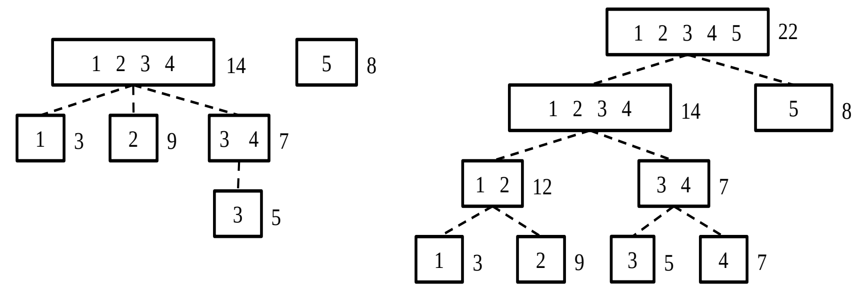

**Fig. 3** These two graphs represent two laminar families that are equivalent. Each box represents a machine that can produce the item types inside of it. The number to the right of each box is the setup cost of the machine. The dashed lines denote which machines are strictly more capable than the others. The left representation is a typical input to this problem, and the right representation is a transformation to a binary tree graph. In both representations, the cost of ordering any subset of item types is the cost of the cheapest collection of machines containing those item types.

## 4.1 Dynamic Programming Formulation

We now describe a polynomial-size dynamic programming formulation to solve the laminar JRP. We assume that the laminar family corresponds to a connected binary tree graph for simplicity. For each node/subset $F \in \mathcal{F}$, let $c_1(F)$ and $c_2(F)$ be its children nodes. If a node is a leaf, then $c_1(F) = c_2(F) = \emptyset$. Let $J(F, s, t)$ be the optimal cost of serving the demands from $s$ to $t$ for all of the item types in $F$, given that the item types in $F$ have already been ordered at time $s$ (i.e., there is a 'free' order of $F$ at time $s$). For convenience, we create a dummy time period 0, with $d^i_0 = 0$ and $H^i_{0t} = \infty$ for all $i$ and $t$. Without the dummy time period, an order of $\mathcal{N}$ would be placed at time 1 free of charge. Thus, the optimal overall cost of the laminar JRP is $J(\mathcal{N}, 0, T)$. (Note that this expression assumes there is a given order of $\mathcal{N}$ at time 0, and thus we do not charge for it.) The base case of the dynamic program is $J(\emptyset, s, t) = 0$ for all $s \le t$. Now we present the dynamic programming recursion for $|F| \ge 1$,

$$\begin{aligned} J(F,s,t) = \min\Big\{ & \sum_{i \in F \setminus \{c_1(F) \cup c_2(F)\}} \sum_{v=s}^{t} H^i_{sv} + J(c_1(F), s, t) + J(c_2(F), s, t), \\ & \min_{u=s+1,\dots,t} \big\{ \kappa_F + J(F,u,t) + \sum_{i \in F \setminus \{c_1(F) \cup c_2(F)\}} \sum_{v=s}^{u-1} H^i_{sv} + \\ & \qquad J(c_1(F), s, u-1) + J(c_2(F), s, u-1) \big\} \Big\} \end{aligned}$$

The first term in the outer min is the case where no additional orders of $F$ are placed between $s+1$ and $t$. (There is no need to place an order of $F$ at $s$

since we assume that the item types in $F$ have already been ordered at $s$.) In this case, we now know the holding costs for all the items in $F \backslash \{c_1(F) \cup c_2(F)\}$ since there are no more orders of $F$ in $[s,t]$. Furthermore, the remaining cost of serving the item types in $c_1(F)$ and $c_2(F)$ can be decomposed due to the laminar property. By construction, the item types in $c_1(F)$ and $c_2(F)$ are currently ordered at $s$, so the respective costs are $J(c_1(F), s, t)$ and $J(c_2(F), s, t)$. The second term in the outer min is the case where we place additional orders of $F$ between $s+1$ and $t$. If there are additional orders, then there must be an optimal earliest additional order, which we denote by $u$. If the next additional order occurs at $u$, then the cost decomposes into a setup cost $\kappa_F$ for the order at $u$, the remaining cost of serving $F$ between $u$ and $t$, i.e. $J(F, u, t)$, and the optimal cost of satisfying $F$ between $s$ and $u-1$ if there are no more further orders of $F$ between $s$ and $u-1$. Note that computing the last part of the cost is computed in the same manner as the first term in the outer min.

In order to solve the complete dynamic program, we start at the leaves of the binary tree and work up towards the root. At each node, solve the intervals of length 0, 1, 2, and so on. Since $\mathcal{F}$ can have size at most $2N$, there are at most $O(NT^2)$ values of $J$. Since each $J$ is computed in $O(T)$ time, this gives an overall runtime of $O(NT^3)$. Based on this dynamic programming analysis, we obtain the following theorem.

**Theorem 3** *There exists a polynomial-time dynamic programming algorithm to solve the laminar JRP.*

## 5 Cardinality Joint Replenishment Problem

In this section, we consider the cardinality JRP, where the joint setup cost is a function of the cardinality of the subset of item types being ordered. Specifically, we let $g(k)$ be a monotonically increasing, concave function which denotes the cost of ordering $k$ item types in any given period. Thus the cost of ordering the item types in $S$ in any given period is $K(S) = g(|S|)$. It is relatively straightforward to see that the concavity of $g(\cdot)$ implies that $K(\cdot)$ is submodular, and therefore the cardinality JRP is a special case of the submodular JRP. Without loss of generality, we also assume that $g(0) = 0$. In Section 5.1, we describe different formulations for the cardinality JRP and some key properties. In Section 5.2, we describe the LP-based approximation algorithm for the cardinality JRP, and in Section 5.3 we prove that our algorithm has an approximation ratio of 5. In Section 5.4, we derive a smaller, polynomial size LP relaxation that may be more useful in practical applications.

### 5.1 Linear Programming Relaxations

The following is an IP formulation for the cardinality JRP. We let $x^i_{st}$ be 1 if the demand point $(i,t)$ is served from period $s$ and 0 otherwise. The variable $z^i_s$ is 1 if an order with item type $i$ is placed in period $s$ and 0 otherwise.

Finally, the variable $q_s^k$ is 1 if there is an order in period $s$ of size at least $k$ and 0 otherwise.

$$\text{minimize} \sum_{s=1}^{T}\sum_{k=1}^{N}(g(k)-g(k-1))q_s^k + \sum_{(i,t)\in\mathcal{D}}\sum_{s=1}^{t} H_{st}^i x_{st}^i \tag{IP-C}$$

$$\text{subject to} \sum_{s=1}^{t} x_{st}^i = 1 \qquad (i,t)\in\mathcal{D} \tag{10}$$

$$x_{s,t}^i \le z_s^i \qquad (i,t)\in\mathcal{D}, s=1,\ldots,t \tag{11}$$

$$q_s^{k+1} \le q_s^k \qquad k=1,\ldots,N-1, s=1,\ldots,T \tag{12}$$

$$\sum_{i=1}^{N} z_s^i = \sum_{k=1}^{N} q_s^k, \qquad s=1,\ldots,T \tag{13}$$

$$x_{st}^i, z_s^i, q_s^k \in \{0,1\}, (i,t)\in\mathcal{D}, s=1,\ldots,t, k=1,\ldots,N$$

**Lemma 6** *(IP-C) is a correct integer programming formulation for the cardinality JRP.*

*Proof* First, we show that there is a one-to-one correspondence between solutions of (IP-C) and the cardinality JRP. Given a solution to (IP-C), we simply order and serve demand according to the variables $x_{st}^i$. Conversely, given a solution to the cardinality JRP, we let $x_{st}^i$ be defined according to the solution. For every $s=1,\ldots,T$, if there are $k$ item types ordered in $s$, we let $z_s^i = 1$ for those $k$ item types and otherwise set $z_s^i = 0$. Also, we set $q_s^1 = \ldots = q_s^k = 1$ and $q_s^{k+1} = \ldots = q_s^N = 0$. It is easy to check that all the constraints are satisfied.

Now we just need to show that the cost of a solution to (IP-C) correctly models the cost of a solution of the cardinality JRP. The holding cost is modeled by the second term of the objective of (IP-C), and is clearly accurate for some solution $x$. For a solution $x$, we know that (IP-C) sets as few of the $z$ variables as possible to 1 since this requires as few as possible $q$ variables to be 1. This property holds due to the fact that $g(k)-g(k-1)$ is always nonnegative due to the monotonicity of $g$. Constraints (12) and (13) ensure that if $k$ item types are ordered in period $s$, then only $q_s^1 = \ldots = q_s^k = 1$. Thus the joint setup cost in this time period is exactly equal to $g(k)$ due to the telescoping sum. □

Although (IP-C) is a correct and polynomial size formulation of the cardinality JRP, the LP relaxation exhibits poor properties. Specifically, one can easily see that in an optimal solution to the LP relaxation of (IP-C), the variables $q_s^1 = \ldots = q_s^N = \frac{1}{N}\sum_{i=1}^{N} z_s^i$ due to the concavity of $g(\cdot)$. Ideally, we would like an LP relaxation where the optimal solution has the intuitive property that $q_s = (q_s^1,\ldots,q_s^N)$ is a permutation of $z_s = (z_s^1,\ldots,z_s^N)$. In order to achieve this property, we add the following valid inequalities to (IP-C).

$$\sum_{i\in S} z_s^i \le \sum_{k=1}^{|S|} q_s^k, \quad \forall S \subset \mathcal{N}, s=1,\dots,T \tag{14}$$

Constraint (14) must be satisfied by any solution $(q,x,z)$ for (IP-C) since if a subset of item types $S$ is ordered in a given period $s$, then clearly $q_s^1 = \ldots = q_s^{|S|} = 1$ in order to account for the joint setup cost. Now define (P-C) to be the the linear programming relaxation of (IP-C) with the valid inequalities (14) included, which is given below.

$$\begin{aligned}
\text{minimize} & \sum_{s=1}^{T}\sum_{k=1}^{N}(g(k)-g(k-1))q_s^k + \sum_{(i,t)\in\mathcal{D}}\sum_{s=1}^{t} H_{st}^i x_{st}^i & \text{(P-C)}\\
\text{subject to} & \sum_{s=1}^{t} x_{st}^i = 1 \qquad (i,t)\in\mathcal{D} \\
& x_{s,t}^i \le z_s^i \qquad (i,t)\in\mathcal{D}, s=1,\dots,t \\
& q_s^{k+1} \le q_s^k \qquad k=1,\dots,N-1, s=1,\dots,T \\
& \sum_{i=1}^{N} z_s^i = \sum_{k=1}^{N} q_s^k, \quad s=1,\dots,T \\
& \sum_{i\in S} z_s^i \le \sum_{k=1}^{|S|} q_s^k, \quad \forall S\subset\mathcal{N}, s=1,\dots,T \\
& x_{st}^i, z_s^i, q_s^k \ge 0, (i,t)\in\mathcal{D}, s=1,\dots,t, k=1,\dots,N
\end{aligned}$$

In the next lemma, we show that there exists an optimal solution to (P-C) which has the desired property that $q_s$ is a permutation of $z_s$. The proof also describes a technique to obtain such a solution.

**Lemma 7** *There exists an optimal solution to (P-C) such that for every $s = 1,\dots,T$, $q_s = (q_s^1,\dots,q_s^N)$ is an ordered permutation of $z_s = (z_s^1,\dots,z_s^N)$.*

*Proof* Let $(q,x,z)$ be an optimal solution to (P-C). For a given time period $s$, assume without loss of generality that $z_s^1 \ge \ldots \ge z_s^N$. Further assume that $q_s$ is not an ordered permutation of $z_s$. From constraint (13), we can choose the largest $i$ such that $q_s^i > z_s^i$ and the smallest $j$ such that $q_s^j < z_s^j$. Now define $\delta = \min(q_s^i - z_s^i, z_s^j - q_s^j)$. We now consider a new solution that decreases $q_s^i$ by $\delta$ and increases $q_s^j$ by $\delta$. Clearly this new solution still satisfies constraints (10), (11), and (13). By the choice of $i$, we know that $q_s^i > q_s^i - \delta \ge z_s^i \ge z_s^{i+1} \ge q_s^{i+1}$. Similarly, by the choice of $j$ we know that $q_s^j < q_s^j + \delta \le z_s^j \le z_s^{j-1} \le q_s^{j-1}$. Therefore, the new solution also satisfies constraint (12). If $j < i$, then constraint (14) is also clearly satisfied. If $i < j$, then checking constraint (14) reduces to checking that $\sum_{k=1}^{i} z_s^k \le \sum_{k=1}^{i} q_s^k$ is

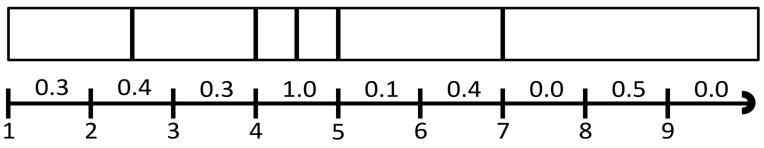

**Fig. 4** This figure shows the half-weight intervals that are constructed for $P^i$ for a problem with 9 time periods. The values on the time axis correspond to the values of $z^i_s$. The partitioned rectangle above the time interval denotes the corresponding intervals, each with a weight of 0.5, that would be added to $P^i$; namely, $(1, 2.5], (2.5, 4], (4, 4.5], (4.5, 5], (5, 7]$, and $(7, 10]$.

satisfied. By the choice of $j$, we know that for $k < i < j$, $q^k_s \geq z^k_s$. Therefore the constraint must still hold after reducing $q^i_s$ to $q^i_s - \delta \geq z^i_s$. Finally, we observe that the cost of the new solution can only decrease since $g(\cdot)$ is a monotonically increasing, concave function. Repeating this procedure will eventually result in an optimal solution that satisfies the desired property. $\square$

We remark that although (P-C) has exponentially many constraints, it can be solved in polynomial time via the ellipsoid method because there is a simple and efficient separation oracle. In Section 5.2, we will use the optimal solution to (P-C) as a basis for an LP rounding algorithm. In Section 5.4, we describe an equivalent but polynomial size formulation of (P-C) which may be of more practical use.

## 5.2 Algorithm

We now describe the LP rounding algorithm for the cardinality JRP. This algorithm first finds an optimal solution to (P-C), which we denote by $(q, x, z)$. If necessary, we modify this solution to satisfy Lemma 7, which can be done via the technique described in the proof. For each item type $i$, we define the density/weight at any (fractional) time $\tau \in (1, T+1]$ to be $z^i_{\lfloor\tau\rfloor}$. Thus, the weight of a time interval $(a, b]$ for item type $i$ is equal to $(\lceil a\rceil - a)z^i_{\lfloor a\rfloor} + \sum_{s=\lceil a\rceil}^{\lfloor b\rfloor-1} z^i_s + (b - \lfloor b\rfloor)z^i_{\lfloor b\rfloor}$. Using this density to measure weights of intervals, for each item type $i$ we partition the time horizon $(1, T+1]$ starting from time 1 into intervals with a weight of exactly 0.5, according to the following procedure. Specifically, let $P^i$ denote the set of intervals we will generate for item type $i$. Initialize $a := 1$. Now let $b$ be a time in $(a, T+1]$ where the weight of $(a, b]$ is exactly 0.5 (if such a $b$ exists). Now add $(a, b]$ to $P^i$. Reset $a := b$ and repeat the procedure until no further intervals can be added. (Note that there may be a portion at the end of $(1, T+1]$ that does not get added to $P^i$.) See Figure 4 for an example of this procedure.

Based on the previous interval partitioning procedure, let $P$ denote the set of all intervals generated using the previous procedure, across all $N$ item types, i.e., $P = \cup_{i\in\mathcal{N}} P^i$. Sort the intervals of $P$ by increasing values of the right endpoints of the intervals, with ties broken arbitrarily. Each interval in $P$ will result in an order of the corresponding item type somewhere in the respective interval. These orders will occur over a series of $N$ rounds. If an interval generates or adds to an order in round $k$, then that interval will receive a label of $k$. In the next paragraph, we describe precisely how to generate these orders and labels.

In each round, we process all the intervals in $P$ by increasing order of the right endpoints. In round 1, for an interval $(a, b]$, an order of the corresponding item type is placed at $b$ if there are currently no other orders in $(a, b]$. If such an order occurs, then $(a, b]$ is given label 1. For now, we assume orders can be placed at any continuous time, and we defer rounding until the end. In general, for labeling round $k$ we process all intervals of $P$ in increasing order of the right endpoints. Let $(a, b]$ in $P$ be the current interval in consideration for label $k$, and let $i$ be the corresponding item type. If $(a, b]$ already has a label, then no action is required. If $(a, b]$ does not have a label but there is an order of size $k$ in $(a, b]$, then leave $(a, b]$ unlabeled and do not modify any order. If $(a, b]$ does not have a label and there is no order of size $k$ in $(a, b]$, then add item type $i$ to the latest order of size $k - 1$ within $(a, b]$ and assign label $k$ to $(a, b]$. (Note that when an order is placed in round 1, it always occurs at the right endpoint since orders of size 0 exist everywhere.) Note that if $(a, b]$ has label $k$, then the size of the order after $i$ was added was exactly $k$. Finally, after all $N$ labeling rounds are complete, we round every order at time $\tau$ to $\lceil \tau - 1 \rceil$. We refer to the orders before the rounding step as tentative orders. All the demand points are then served myopically from the nearest available order of the corresponding item type. The pseudocode for the algorithm is given as Algorithm 2. See Figure 5 for an example of the algorithm.

## 5.3 Analysis

We now show that Algorithm 2 is a 5-approximation for the cardinality JRP. Throughout this section, we refer to $(q, x, z)$ as an optimal solution to (P-C) satisfying Lemma 7. We first prove a pair of useful facts in Lemmas 8 and 9 that will allow us to bound the holding costs in Lemma 10.

**Lemma 8** *Each interval $j \in P$ receives a label.*

*Proof* Assume there exists $(a, b] \in P$ of item type $i$ without a label. Let $k$ be the size of the largest order in $(a, b]$ after all the labeling rounds are complete. Since $(a, b]$ is unlabeled, then by construction the algorithm does not place an order of item type $i$ in $(a, b]$. Therefore, we know that $k < N$. Since $(a, b]$ was not labeled in round $k$, then the size $k$ order (or another order of size $k$ in $(a, b]$) must have existed when $(a, b]$ was processed in round $k$. Therefore, during the $(k+1)^{st}$ labeling round of the algorithm, item type $i$ contains only orders of size at most $k$ (with at least one of size $k$) and should have been added to an order of size $k$ in $(a, b]$. Thus, $(a, b]$ would end up labeled during round $k + 1$, which is a contradiction. □

**Algorithm 2:** Labeling Algorithm for cardinality JRP

```
Solve (P-C) for an optimal solution $(q, x, z)$ that satisfies Lemma 7
// Generate intervals of $P^i$

for $i \in \mathcal{N}$ do
    For all $\tau \in (1, T+1]$, let the density be $z^i_{\lfloor \tau \rfloor}$
    Initialize $a$:=1
    Choose $b$, if it exists, so that the weight of $(a, b]$ is 0.5 and add $(a, b]$
    to $P^i$
    Repeat previous step with $a := b$ until no more intervals can be
    added to $P^i$
Let $P = \cup_{i \in \mathcal{N}} P^i$
Sort $P$ by increasing values of the right endpoints
// Generate orders and labels

for $k \leftarrow 1$ to $N$ do
    Process $P$ in increasing order of the right endpoints
    for $(a, b] \in P$ do
        Let $i$ be the corresponding item type for interval $(a, b]$
        if $(a, b]$ is unlabeled and $\nexists$ an order of size $k$ in $(a, b]$ then
            Add item type $i$ to the latest order of size $k-1$ in $(a, b]$
            Give label $k$ to the interval $(a, b]$
Round every order at time $\tau$ to time $\lceil \tau - 1 \rceil$
Serve every demand point $(i, t)$ from the latest order up to time $t$ that
includes item type $i$
```

As in the analysis of the algorithm for tree JRP, we define the *active interval* for demand point $(i, t)$ to be $[s_1, t]$, where $s_1$ is the earliest integer time period $s$ where $x^i_{st} > 0$.

**Lemma 9** *Every demand $(i, t)$ is served from its active interval.*

*Proof* Consider a demand point $(i, t)$ and its active interval $[s_1, t]$. From feasibility of $(q, x, z)$ and the fact that $x^i_{st} \leq z^i_s$, we know that there must be at least one interval $(a, b] \in P$ of item type $i$ such that $(a, b] \subset (s_1, t+1]$. This is because the $z^i$-weight of $(a, b]$ is 0.5, while the $z^i$-weight of $(s_1, t+1]$ is at least 1, which means the partition corresponding to item type $i$ must contain an interval strictly in $(s_1, t+1]$. By Lemma 8, we know that there was a tentative order of $i$ at some time $\tau \in (a, b]$. Therefore, $\tau \in (s_1, t+1]$ and since the tentative order is rounded down in time to $\lceil \tau - 1 \rceil$, then there is an actual order of $i$ in $[s_1, t]$. □

Using Lemmas 8 and 9, we now bound the holding cost in Lemma 10.

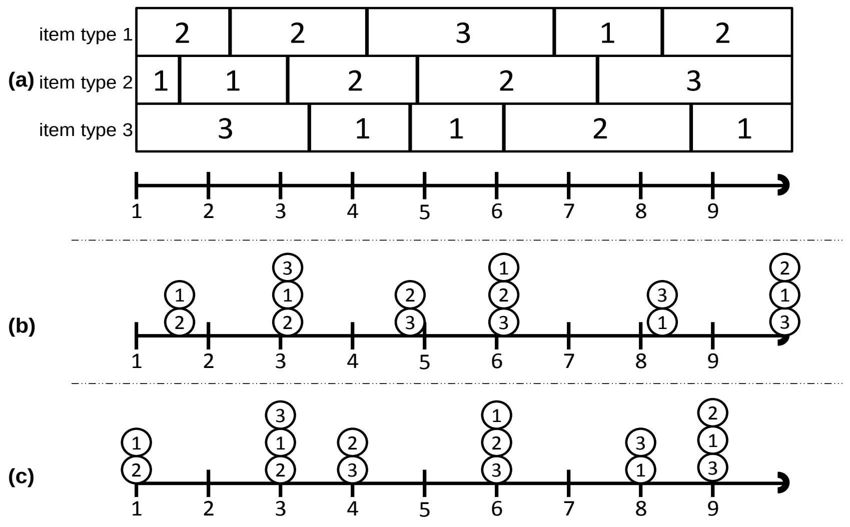

**Fig. 5** This figure shows an example of the LP rounding algorithm. In (a), each rectangle correspond to an interval in $P$. Each row of intervals corresponds to an item type. The intervals were generated using the partition procedure depicted in Figure 4. The number in each rectangle/interval corresponds to the label it was given using Algorithm 2. In (b), we show how these labels correspond to orders. The number inside each circle denotes the item type that was ordered. The height of each circle corresponds to its label. In (c), we show how to round each order to an integral time.

**Lemma 10** *The total holding cost for the solution produced by Algorithm 2 is at most the optimal cost of (P-C).*

*Proof* We bound the holding cost of our solution using the dual of (P-C). Let $B^i_t$ be the dual variables for constraint (10) and let $L^i_{st} \geq 0$ be the dual variables for constraint (11). Then we know the objective of the dual is $\max \sum_{(i,t)\in\mathcal{D}} B^i_t$. Furthermore, we know the dual has constraints corresponding to $x^i_{st}$ of the form $B^i_t \leq H^i_{st} + L^i_{st}$ for $(i,t) \in \mathcal{D}, s = 1, \ldots, t$. Now for each demand $(i,t) \in \mathcal{D}$, we bound the holding cost incurred by the algorithm by $B^i_t$. By Lemma 9, $(i,t)$ is served from an integer time $s \in [s_1, t]$, the active interval of $(i,t)$. From complementary slackness and the fact that $x^i_{s_1 t} > 0$, we know that $B^i_t = H^i_{s_1 t} + L^i_{s_1 t}$. Since $L^i_{s_1 t} \geq 0$, then the holding cost paid by $(i,t)$ is $H^i_{st} \leq H^i_{s_1 t} \leq B^i_t$. This implies the holding cost of our solution is at most the optimal cost of (P-C), since the sum of $B^i_t$ over all demand points gives the dual objective. □

We now prove another pair of useful facts in Lemmas 11 and 12 that will allow us to bound the ordering costs in Lemma 13.

**Lemma 11** *For any time $\tau \in (1, T+1]$ and $k \in 1, \ldots, N$, there are at most 2 intervals in $P$ that are both labeled $k$ and contain $\tau$.*

*Proof* First we show that for any two intervals $(a_1, b_1]$ and $(a_2, b_2]$ in $P$ such that $(a_1, b_1] \subseteq (a_2, b_2]$, their labels must be different. Let $k$ be the label of $(a_1, b_1]$. Observe that this implies that at the beginning of round $k$, $(a_1, b_1]$ contains an order of size $k-1$. If $(a_1, b_1]$ is processed first in $P$, then $(a_2, b_2]$ has an order of size $k$ in it when it is processed in labeling round $k$ and therefore cannot receive label $k$. If $(a_2, b_2]$ is processed first in $P$, then it must be the case that $b_1 = b_2$ by the ordering of $P$. If $(a_2, b_2]$ is already labeled, then it must have a label smaller than $k$. Otherwise, $(a_2, b_2]$ will be assigned a label $k$ and its corresponding item type is added to a size $k-1$ order in $(a_1, b_1]$ by construction. However, this is a contradiction because when $(a_1, b_1]$ is later processed in round $k$, it cannot be labeled $k$ since it contains an order of size $k$.

Now assume for contradiction that there are three intervals $(a_1, b_1], (a_2, b_2]$ and $(a_3, b_3]$ with label $k$, all of which contain a common time $\tau$. Denote $i_1$, $i_2$, and $i_3$ as the corresponding item types of these intervals. From the previous argument, we can assume that $a_1 < a_2 < a_3 < \tau \leq b_1 < b_2 < b_3$. Observe that the order where $i_1$ was added must occur in $(a_1, a_2]$, or else there would be a size $k$ order in $(a_2, b_2]$ when it was processed in the $k^{th}$ labeling round and $(a_2, b_2]$ would be skipped. Now consider the following two cases of where $i_2$ was ordered. If it occurs in $(a_2, b_1]$, this implies that $i_1$ was not ordered at the latest possible order of size $k-1$. If the order of $i_2$ occurred within $(b_1, b_2]$, this implies that $(a_3, b_3]$ is skipped when it is processed in the $k^{th}$ labeling round since there is an order of size $k$ in its interval. Therefore, we cannot give a label of $k$ to $(a_2, b_2]$, which is a contradiction. □

**Lemma 12** *Let $X$ and $Y$ be vectors in $\mathbb{R}^N$ such that $X^1 \geq \ldots \geq X^N$ and $Y^1 \geq \ldots \geq Y^N$. Let $\Pi$ be the set of all permutations of $\{1, \ldots, N\}$. Then $\max_{\sigma \in \Pi} \sum_{i=1}^{N} X^i Y^{\sigma(i)}$ is exactly $\sum_{i=1}^{N} X^i Y^i$.*

*Proof* We prove this by induction over $N$, with a base case when $N = 2$. In the base case, we know that $(X^1 - X^2)(Y^1 - Y^2) \geq 0$ since $X^1 \geq X^2$ and $Y^1 \geq Y^2$. Expanding this inequality implies that $X^1Y^1 + X^2Y^2 \geq X^1Y^2 + X^2Y^1$ and therefore the result holds for $N = 2$.

Now assume the result holds for $N = k$ and we want to prove that it also holds for $N = k+1$. Let $\sigma \in \Pi$ and choose $i$ so that $\sigma(i) = 1$. Now focus on two summands from the objective, namely $X^1 Y^{\sigma(1)}$ and $X^i Y^{\sigma(i)}$. Based on the choice of $i$, we know that $X^1 \geq X^i$ and $Y^{\sigma(i)} \geq Y^{\sigma(1)}$. From the base case, we know that $X^1Y^1 + X^iY^{\sigma(1)} \geq X^1Y^{\sigma(1)} + X^iY^1$. Therefore, if we switch $\sigma(1)$ and $\sigma(i)$, the objective can only increase, and furthermore $X^1$ and $Y^1$ are now matched in the objective. Now for the rest of the $k$ summands (all except $X^1Y^1$), we apply the inductive hypothesis for $N = k$, and obtain the desired result. □

Using Lemmas 11 and 12, we now bound the ordering cost as described in Lemma 13. The idea of the proof is to charge each interval the marginal ordering cost it incurred in the algorithm. If an interval was labeled as $k$, then this marginal cost is exactly $g(k) - g(k-1)$. This accounting is done using the $z_s^i$ variables, which directly ties to the objective of (P-C) by Lemma 7. Since Lemma 11 bounds the number of times each label is used, this allows to bound the number of times each $z_s^i$ variable, and therefore $q_s^i$ variable, is used.

**Lemma 13** *The total ordering cost for the solution produced by Algorithm 2 is at most 4 times the optimal cost of (P-C).*

*Proof* Consider an arbitrary interval $(a, b] \in P$, and let $k$ be its label and $i$ be the corresponding item type. By the definition of a label, when $i$ was added to a tentative order in the interval $(a, b]$, it was the $k^{th}$ item type added. Therefore, we would like to charge interval $(a, b]$ a cost of $g(k) - g(k-1)$ to account for its contribution to the overall ordering cost. To do this, it suffices to charge $2z^i_{\lfloor\tau\rfloor}[g(k) - g(k-1)]$ for all $\tau \in (a, b]$. This is enough to cover the cost of $g(k) - g(k-1)$ since the $z^i$-weight of the interval $(a, b]$ is 0.5 (recall the density at a time $\tau$ is $z^i_{\lfloor\tau\rfloor}$). More generally, let $l(i, \tau)$ be the label of the interval in $P$ of item type $i$ that contains $\tau$ (if it exists). (Note that if $\tau$ is near the end of the horizon, there may be no interval in $P$ of item type $i$ that contains $\tau$ and thus we can ignore $i$.) From the previous discussion, for each $\tau \in (1, T+1]$, we would like to charge $\sum_{i\in\mathcal{N}} 2z^i_{\lfloor\tau\rfloor}[g(l(i,\tau)) - g(l(i,\tau)-1)]$ to account for the total ordering cost of the algorithm. Next, we describe how to bound this charging scheme by four times the optimal cost of (P-C).

For any time $\tau \in (1, T+1]$, we do the following analysis. Partition the item types into two sets, $S_\tau$ and $\mathcal{N}\backslash S_\tau$, such that for any two different item types $i, j$ in the same set, $l(i,\tau) \neq l(j,\tau)$. In other words, all the item types in $S_\tau$ have unique labels and all the item types in $\mathcal{N}\backslash S_\tau$ have unique labels. This procedure is feasible due to Lemma 11. Now for all $i \in \mathcal{N}$, we will assign a fake label $l'(i,\tau)$. For all $i \in \mathcal{N}\backslash S_\tau$, uniquely assign the unused labels of $S_\tau$, i.e. $\mathcal{N}\backslash \bigcup_{i\in S_\tau} l(i,\tau)$, as the fake labels of $\mathcal{N}\backslash S_\tau$. Similarly, for all $i \in S_\tau$, uniquely assign the unused labels of $\mathcal{N}\backslash S_\tau$, i.e. $\mathcal{N}\backslash \bigcup_{i\in \mathcal{N}\backslash S_\tau} l(i,\tau)$, as the fake labels of $S_\tau$. Then our charging scheme at time $\tau$ is upper bounded by

$$\begin{aligned}
&\sum_{i\in\mathcal{N}} 2z^i_{\lfloor\tau\rfloor}[g(l(i,\tau)) - g(l(i,\tau)-1)] \\
&\leq \sum_{i\in S_\tau} 2z^i_{\lfloor\tau\rfloor}[g(l(i,\tau)) - g(l(i,\tau)-1)] + \sum_{i\in\mathcal{N}\backslash S_\tau} 2z^i_{\lfloor\tau\rfloor}[g(l'(i,\tau)) - g(l'(i,\tau)-1)] \\
&+ \sum_{i\in\mathcal{N}\backslash S_\tau} 2z^i_{\lfloor\tau\rfloor}[g(l(i,\tau)) - g(l(i,\tau)-1)] + \sum_{i\in S_\tau} 2z^i_{\lfloor\tau\rfloor}[g(l'(i,\tau)) - g(l'(i,\tau)-1)]
\end{aligned} \tag{15}$$

$$\leq 4\sum_{k=1}^{N} q^k_{\lfloor\tau\rfloor}[g(k) - g(k-1)] \tag{16}$$

The first inequality follows simply from adding nonnegative terms. For the second inequality, by construction we know that all labels $1, \ldots, N$ are used exactly once over the first two summations in (15), and exactly once over the second two summations in (15) by construction. Since $g(k) - g(k-1) \geq g(k+1) - g(k)$, $q^k_{\lfloor \tau \rfloor} \geq q^{k+1}_{\lfloor \tau \rfloor}$, and $q_{\lfloor \tau \rfloor}$ is a permutation of $z_{\lfloor \tau \rfloor}$ (Lemma 7), then the second inequality follows from applying Lemma 12 twice. Integrating our upper bound on the charging scheme (16) over $\tau$ from 1 to $T+1$ clearly gives a total ordering cost of at most four times the optimal cost of (P-C). □

Since the optimal cost of (P-C) is a lower bound on the total cost (Lemma 6), combining Lemmas 10 and 13 proves our main theorem below.

**Theorem 4** *Algorithm 2 is a 5-approximation algorithm for the cardinality JRP.*

## 5.4 Polynomial Size LP Relaxation

Although constraint (14) of (P-C) allows us to prove Lemma 7 and therefore Theorem 4, it makes the (P-C) formulation exponentially sized. Since our algorithm for the cardinality JRP relies on using an optimal solution for (P-C), for practical purposes we would like to be able to reformulate (P-C) using only a polynomial number of constraints and variables. In order to do this, let

$$F^k_s = \{q_s \in \mathbb{R}^N, z_s \in \mathbb{R}^N | \sum_{i \in S} z^i_s \leq \sum_{i=1}^{k} q^i_s \;\; \forall S \subset \{1, \ldots, N\} \text{ s.t. } |S| = k\}.$$

Based on the definition of $F^k_s$, we can describe constraint (14) in (P-C) by $(q_s, z_s) \in \cap_{k=1}^{N-1} F^k_s$ for all $s = 1, \ldots, T$. (Note that the subscript $s$ in $F^k_s$ is used for notational convenience, and is technically not necessary.) Now observe that the left-hand side of the constraints in $F^k_s$ can be upper bounded by $\max_{S \subseteq N} \sum_{i \in S} z^i_s$. This maximization problem is equal to the following integer program.

$$\max \sum_{i=1}^{N} z^i_s a^i \text{ s.t. } \sum_{i=1}^{N} a^i = k, \; a^i \in \{0,1\}, \; i = 1, \ldots N \qquad \text{(IP-k)}$$

Let (P-k) be the linear programming relaxation of (IP-k) where the binary constraints are replaced by $0 \leq a^i \leq 1$. Note (P-k) has integral optimal solutions and is equivalent to (IP-k). By strong duality, (P-k) is also equivalent to its dual program (D-k), which is

$$\min \;\; k\beta_s + \sum_{i=1}^{N} \alpha^i_s \text{ s.t. } \alpha^i_s + \beta_s \geq z^i_s, \; \alpha_i \geq 0, \; i = 1, \ldots N \qquad \text{(D-k)}$$

Motivated by these observations, we define the polyhedron $G_s^k$ by replacing the LHS of the constraint of $F_s^k$ with the dual objective in (D-k) and add the corresponding constraints and variables.

$$G_s^k = \{q_s \in \mathbb{R}^N, z_s \in \mathbb{R}^N, \alpha_s \in \mathbb{R}_+^N, \beta_s \in \mathbb{R} |\ k\beta_s + \sum_{i=1}^N \alpha_s^i \le \sum_{i=1}^k q_s^i,$$
$$\alpha_s^i + \beta_s \ge z_s^i, \ \ \forall i = 1, \ldots, N\}$$

In Lemma 14, we show that $F_s^k = \text{proj}_{(q_s,z_s)} G_s^k$. (Again, note that the subscript $s$ is used for notational convenience, and is technically not necessary to define $F_s^k$ and $G_s^k$.) Therefore, if we replace the constraints corresponding to $F_s^k$ in (P-C) with the constraints and variables of $G_s^k$, this results in an equivalent and polynomial sized LP formulation that can be solved in polynomial time, which we denote by (P-C*).

**Lemma 14** *For any $k = 1, \ldots, N-1$, $F_s^k = \mathit{proj}_{(q_s,z_s)} G_s^k$.*

*Proof* We first show that any $(q_s, z_s) \in F_s^k$ is also in $\text{proj}_{(q_s,z_s)} G_s^k$. Without loss of generality, assume that $z_s^1 \ge \ldots \ge z_s^N$. Let $\beta_s = z_s^k$, $\alpha_s^i = z_s^i - z_s^k$ for $i \le k$, and $\alpha_s^i = 0$ for $i > k$. Then $k\beta_s + \sum_{i=1}^N \alpha_s^i = \sum_{i=1}^k z_s^i \le \sum_{i=1}^k q_k^i$, where the inequality holds by the constraint from $F_s^k$. Moreover, for any $i$ we have that $\beta_s + \alpha_s^i = \max\{z_s^i, z_s^k\} \ge z_s^i$ by construction. Therefore, $(q_s, z_s, \alpha_s, \beta_s) \in G_s^k$ and thus $(q_s, z_s) \in \text{proj}_{(q_s,z_s)} G_s^k$.

Now we show that any $(q_s, z_s) \in \text{proj}_{(q_s,z_s)} G_s^k$ is also in $F_s^k$. Without loss of generality, assume that $z_s^1 \ge \ldots \ge z_s^N$. Since $(q_s, z_s) \in \text{proj}_{(q_s,z_s)} G_s^k$, there exists $\alpha_s$ and $\beta_s$ such that $(q_s, z_s, \alpha_s, \beta_s) \in G_s^k$. We know that for any $S$ such that $|S| = k$, $\sum_{i \in S} z_s^i \le \sum_{i=1}^k z_s^i \le k\beta_s + \sum_{i=1}^k \alpha_s^i \le k\beta_s + \sum_{i=1}^N \alpha_s^i \le \sum_{i=1}^k q_k^i$. The first inequality follows from the ordering of $z_s$; the second inequality follows from the second constraint in $G_s^k$; the third inequality follows from the nonnegativity of $\alpha_s$; and the last inequality follows from the first constraint in $G_k^s$. Thus, we have shown that $(q_s, z_s)$ is indeed in $F_s^k$. □

## 6 Conclusion

We believe that this work advances the existing research for an important class of interesting inventory management problems with non-stationary demand. The submodular JRP and the special cases we consider can capture a wide variety of real world problems and allows for substantially more modeling flexibility. Since most variants of the JRP are NP-hard, it is intractable to compute the optimal solutions efficiently. Our algorithms are computationally efficient and provide theoretical worst case guarantees. Moreover, we provided strong and polynomial size LP formulations for the tree and cardinality JRP which may be useful for solving the IP formulations in practice. One major open question which we have left unanswered is whether or not there exists a constant factor approximation algorithm for the submodular JRP.

**Acknowledgements** We thank the refereeing team for their numerous insightful comments that significantly improved the quality and exposition of the paper. The research of the first author was supported by NSF grants CCF-0832782, CCF-1017688 and NSERC grant PGS-358528. The research of the second author was partially conducted while he was a student at the Operations Research Center at Massachusetts Institute of Technology and was supported by a National Defense Science and Engineering Graduate Fellowship from the Air Force Office of Scientific Research. The research of the third author was partially supported by NSF grant CMMI-0846554 (CAREER Award), an AFOSR award FA9550-11-1-0150, an SMA grant and the Buschbaum Research Fund of MIT. The research of the fourth author was supported by NSF grants CCF-0832782 and CCF-1017688.

## References

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

Becchetti L, Korteweg P, Marchetti-Spaccamela A, Skutella M, Stougie L, Vitaletti A (2006) Latency constrained aggregation in sensor networks. Springer

Bertsimas D, Weismantel R (2005) Optimization over integers, vol 13. Dynamic Ideas Belmont

Bienkowski M, Byrka J, Chrobak M, Jez L, Sgall J (2013) Better approximation bounds for the joint replenishment problem. arXiv preprint arXiv:13072531

Chan L, Muriel A, Shen Z, Simchi-Levi D, Teo C (2002) Effective zero-inventory-ordering policies for the single-warehouse multiretailer problem with piecewise linear cost structures. Management Science pp 1446–1460

Chvatal V (1979) A greedy heuristic for the set-covering problem. Mathematics of operations research 4(3):233–235

Federgruen A, Tzur M (1994) The joint replenishment problem with time-varying costs and demands: Efficient, asymptotic and $\varepsilon$-optimal solutions. Operations Research pp 1067–1086

Federgruen A, Zheng Y (1992) The joint replenishment problem with general joint cost structures. Operations Research 40(2):384–403

Federgruen A, Queyranne M, Zheng YS (1992) Simple power-of-two policies are close to optimal in a general class of production/distribution networks with general joint setup costs. Mathematics of Operations Research 17(4):951–963

Herer Y, Roundy R (1997) Heuristics for a one-warehouse multiretailer distribution problem with performance bounds. Operations Research 45(1):102–115

Joneja D (1990) The joint replenishment problem: new heuristics and worst case performance bounds. Operations Research 38(4):711–723

Kao E (1979) A multi-product dynamic lot-size model with individual and joint set-up costs. Operations Research pp 279–289

Khanna S, Naor JS, Raz D (2002) Control message aggregation in group communication protocols. In: Automata, Languages and Programming, Springer, pp 135–146

Levi R, Roundy R, Shmoys D (2006) Primal-Dual Algorithms for Deterministic Inventory Problems. Mathematics of Operations Research 31(2):267–284

Levi R, Roundy R, Shmoys D, Sviridenko M (2008) A Constant Approximation Algorithm for the One-Warehouse Multiretailer Problem. Management Science 54(4):763

Levi R, Magnanti T, Zarybnisky E (2011) Maintenance scheduling for modular systems-models and algorithms. PhD thesis, Massachusetts Institute of Technology

Nagarajan V, Shi C (2015) Approximation Algorithms for Inventory Problems with Submodular or Routing Costs. arXiv preprint arXiv:1504.06560

Nonner T, Souza A (2009) A 5/3-approximation algorithm for joint replenishment with deadlines. In: Combinatorial Optimization and Applications, Springer, pp 24–35

Nonner T, Sviridenko M (2013) An efficient polynomial-time approximation scheme for the joint replenishment problem. In: Integer Programming and Combinatorial Optimization, Springer, pp 314–323

Orlin JB (2009) A faster strongly polynomial time algorithm for submodular function minimization. Mathematical Programming 118(2):237–251

Pedrosa L, Sviridenko M (2013) Private communivation

Queyranne M (1986) A polynomial-time, submodular extension to roundy's 98% effective heuristic for production/inventory. In: Robotics and Automation. Proceedings. 1986 IEEE International Conference on, IEEE, vol 3, pp 1640–1640

Roundy R (1985) 98%-effective integer-ratio lot-sizing for one-warehouse multi-retailer systems. Management science pp 1416–1430

Schrijver, A (1998) Theory of linear and integer programming. John Wiley & Sons

Schulz AS, Telha C (2011) Approximation algorithms and hardness results for the joint replenishment problem with constant demands. In: Algorithms–ESA 2011, Springer, pp 628–639

Segev D (2013) An approximate dynamic-programming approach to the joint replenishment problem. Mathematics of Operations Research

Shmoys D, Tong C (2013) Private communivation

Stauffer G, Massonnet G, Rapine C, Gayon J (2011) A simple and fast 2-approximation algorithm for the one-warehouse multi-retailers problem. In: Proceedings of the nineteenth annual ACM-SIAM symposium on Discrete algorithms, Society for Industrial and Applied Mathematics

Svitkina Z, Tardos E (2010) Facility location with hierarchical facility costs. ACM Transactions on Algorithms (TALG) 6(2):37

Teo C, Bertsimas D (2001) Multistage lot sizing problems via randomized rounding. Operations Research pp 599–608

Veinott Jr A (1969) Minimum concave-cost solution of leontief substitution models of multifacility inventory systems. Operations Research pp 262–291

Viswanathan S (2007) An algorithm for determining the best lower bound for the stochastic joint replenishment problem. Operations research 55(5):992–996

Wagner H, Whitin T (1958) Dynamic Version of the Economic Lot Size Model. Management Science 5(1):89–96

Zangwill W (1969) A backlogging model and a multi-echelon model of a dynamic economic lot size production system-a network approach. Management Science pp 506–527