Egerváry Research Group
on Combinatorial Optimization

Technical reportS

TR-2020-20. Published by the Egerváry Research Group, Pázmány P. sétány 1/C, H–1117, Budapest, Hungary. Web site: www.cs.elte.hu/egres. ISSN 1587–4451.

Joint replenishment meets scheduling

József Békési, Péter Györgyi, and Tamás Kis

October 2020

Joint replenishment meets scheduling

József Békési*, Péter Györgyi**, and Tamás Kis***

Abstract

In this paper we consider a combination of the joint replenishment problem and a single machine scheduling problem with release dates. There is a single machine and one or more additional, non-renewable resources. Each job has a release date, a positive processing time, and some resource requirements. A job can be started at time t only if there occurred a replenishment of all the required resources between the release date of the job and time point t. The objective is to minimize the total replenishment cost plus the total weighted completion time of the jobs, where each replenishment from each resource incurs a fixed cost, and there is an additional fixed cost each time some resource is replenished. We provide several complexity results and competitive analysis for online variants of the problem.

1 Introduction

Joint replenishment is a classical problem of supply chain management with several practical applications. In this problem a number of requests (jobs, items, etc.) emerges during the time horizon, where each request requires some resources. To fulfill a request, all the required resources must be replenished, however, these replenishments cannot occur before the arrival of the request. The replenishments incur some fixed costs as follows. Each time when one or more resources are replenished, there is a fixed cost independently of the number of distinct resource types that are replenished simultaneously, and moreover, the replenishment of each resource type incurs an additional fixed cost. To save some cost, the replenishments of different requests can be combined, that is, if two or more requests require a common resource, then it can be replenished jointly to economize on the replenishment costs. Of course, delaying a replenishment of a resource delays the fulfillment of all the requests that require it.

Various objectives have been studied in the literature, but in most cases, the objective function describes somehow a trade-off between the replenishment costs and cost related to the delays in the fulfillments of the requests. The latter cost can be the

*Department of Applied Informatics, Gyula Juhász Faculty of Education, University of Szeged, Hungary. E-mail: bekesi@jgypk.szte.hu

**Institute for Computer Science and Control, Budapest, Hungary.
E-mail: gyorgyi.peter@sztaki.hu

***Institute for Computer Science and Control, Budapest, Hungary. E-mail: kis.tamas@sztaki.hu

total (weighted) completion time of the requests, the total lateness of the requests, or some inventory costs.

There are applications where every data is known from the beginning, however, in most practical cases information about a request become known only when it is released. Traditional models usually assume distributions about the future requests, but these are not available in several cases. Therefore, newer models (e.g. [9]) often deal with the online version of the problem.

It is a common assumption in joint replenishment problems that a request is ready if all the required resources have arrived. However, in several cases these requests require some processing by one or more machines, or in other words a scheduling problem emerges among the requests for which the required resources have arrived. This paper begins the examination of such variants. Due to the commonly used terminology of scheduling problems, from now on we call the requests as jobs.

We consider the simplest scheduling environment, the single machine. It means, we have one machine, which can process at most one job at a time. Each job has a processing time, and preemptions of the jobs are not allowed. The objective function is the sum of the replenishment costs and the scheduling cost (total weighted completion time or total weighted flow time of the jobs). Observe that if we delay a replenishment then the jobs waiting for their resources will occupy the machine after their resources arrive, thus they may delay some further jobs. In other words at each time moment we have to take care about the jobs with a latter release date, which is not easy in the offline case, and almost impossible in the online case.

After a literature review, a formal problem statement, and listing the results of the paper, we prove some complexity results in section 2. Then, in section 3 we show 2-competitive online algorithms for two variants, and finally, in section 4 we prove lower bounds on the best competitive ratio of different variants.

1.1 Literature review

The first results in joint replenishment are more than 50 years old (e.g., [23]), for the older results we cite [18]. Since then, several theoretical and practical results became known, this review cites only the most relevant papers.

One of the most common way to minimize the delays is to introduce delay penalty costs which increases as the delay increases. An offline variant of such a problem is APX-hard ([3],[22]), but there are approximation algorithms ([21],[20],[22]) for a special different variants. In one of the closest model to the model of this paper, Buchbinder et al. [9] consider an online variant of previous problem providing a competitive analysis against a worst-case adversary. Cheung et al. [12] consider a model where the trade-off is between a submodular joint setup cost and an inventory cost for holding the delaying requests in the inventory. They provide several approximation algorithms for different variants. A similar variant is considered by [8].

Bienkowski et al. [7] provided new approximation results for the problem of [9]. They also consider a model where the requests have deadlines and the ordering cost is to be minimized. For the latter problem, further results are presrented in [6].

For scheduling problems we use the well-known $\alpha|\beta|\gamma$ notation of [15]. $1|r_j|\sum C_j$

is a known strongly NP-hard problem, see, e.g., [19]. There is a polynomial time approximation scheme (PTAS) for this problem even in case of job weights and parallel machines ([1]). However, there are other important approximation results for this problem. Chekuri et al. [11] provide an $e/(e-1) \approx 1.5819$-approximation algorithm for this problem based on α-point method. For the weighted version of the same problem [14] presents an algorithm with a 1.7451-approximation ratio. If the each job has the same processing time then the weighted version of the problem is solvable in polynomial time even in case of constant number of parallel machines ([5]).

In [2], a 2-competitive algorithm is devised for $1|online, r_j|\sum w_j C_j$, i.e., for the online version of the previous scheduling problem. This is the best possible algorithm for this problem, since [16] proved no online algorithm can have a competitive ratio less than 2, even if the weights of the jobs are identical.

Kellerer er al. [17] provided a $O(\sqrt{n})$-approximation algorithm for $1|r_j|\sum F_j$ and proved that no polynomial time algorithm can have $O(n^{1/2-\varepsilon})$ approximation ratio for any $\varepsilon > 0$ unless $P = NP$. It is also known that the best competitive ratio of the online problem without weights is $\Theta(n)$, and it is unbounded if the jobs have weights ([13]). Due to these results, most of the research papers deal with the preemptive jobs (e.g., [10],[13],[4]).

1.2 Problem statement

We have a set $\mathcal{J}$ of n jobs that has to be scheduled on a single machine. Each job j has a processing time $p_j > 0$, a release date $r_j \geq 0$, and a weight $w_j > 0$. Let $\mathcal{T} := \{\tau_1, \ldots, \tau_{|\mathcal{T}|}\}$ be the set of different job release dates r_j such that $\tau_1 < \ldots < \tau_{|\mathcal{T}|}$. For technical reasons we introduce $\tau_{|\mathcal{T}|+1} := \tau_{|\mathcal{T}|} + \sum_{j\in\mathcal{J}} p_j$.

In addition, there is a set of resources $\mathcal{R} = \{R_1, \ldots, R_s\}$, and each job $j \in \mathcal{J}$ requires a non-empty subset $R(j)$ of the resources. Let $\mathcal{J}_i \subseteq \mathcal{J}$ be the set of those jobs that require resource R_i. Before a resource can be used for a job, it must be replenished. A *replenishment structure* $\mathcal{Q} = (\mathcal{Q}_1, \mathcal{Q}_2, \ldots)$ determines the set of resources replenished at each time moment, i.e., $\mathcal{Q}_t \subseteq \mathcal{R}$ is the set of those resources replenished at time moment t. A *schedule* S determines a start time S_j for each $j \in \mathcal{J}$. We say that job j is *available at time moment* t in replenishment structure $\mathcal{Q}$, if $R(j) \subseteq \cup_{\tau\in[r_j,t]}\mathcal{Q}_\tau$. A *feasible solution* of the problem is a pair of a schedule and a replenishment structure, i.e., $(S, \mathcal{Q})$, such that (i) the jobs do not overlap in time, i.e., $S_j + p_j \leq S_k$ or $S_k + p_k \leq S_j$ for each $j \neq k$, (ii) each job $j \in \mathcal{J}$ is available at S_j in $\mathcal{Q}$.

We seek a feasible solution $(S, \mathcal{Q})$ which minimizes the cost of the scheduling c_S, plus the cost of the replenishments $c_\mathcal{Q}$. The former cost can be $c_S := \sum_{j\in\mathcal{J}} w_j C_j$, where $C_j := S_j + p_j$ is the completion time of $j \in \mathcal{J}$, or $c_S := \sum_{j\in\mathcal{J}} w_j F_j$, where $F_j := C_j - r_j$ is the flow time of $j \in \mathcal{J}$. Note that $\sum w_j F_j = \sum w_j C_j - \sum w_j r_j$, where the last sum is a constant, thus the complexity status (polynomial or NP-hard) is the same for the two problems. However, there can be differences in approximability and the best competitive ratio of algorithms for the online version of the problem (see later).

The latter cost is determined as follows. Whenever $\mathcal{Q}_t \neq \emptyset$, a fixed cost of K_0 arises.

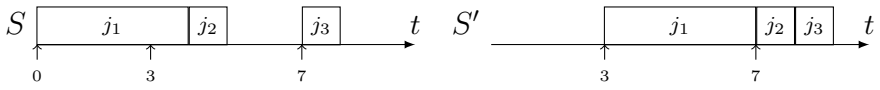

Figure 1: Two feasible solutions. The arrows below the axis denote the replenishments.

In addition, for each $R_i \in \mathcal{Q}_t$, an amount of K_i must be payed. Hence, the total cost of replenishment at time t in replenishment structure $\mathcal{Q}$ is $c_t := K_0 + \sum_{R_i \in \mathcal{Q}_t} K_i$ if $\mathcal{Q}_t \neq \emptyset$, while $c_t := 0$ otherwise. These costs are independent of the amount of the resources acquired. The total replenishment cost is $c_{\mathcal{Q}} := \sum_t c_t$. Hence, the goal is to minimize the value of the solution $v(S, \mathcal{Q}) := c_S + c_{\mathcal{Q}}$.

We extend the notation of [15] by jrp in the β field to describe the joint replenishment of the resources, thus we denote our problem by $1|jrp, r_j|c_S + c_{\mathcal{Q}}$.

Since there always exists an optimal solution in which the resources are replenished only at the release dates of the jobs, we make the following assumption without loss of generality.

Assumption 1. *$\mathcal{Q}_t = \emptyset$, if $t \notin \mathcal{T}$.*

We also consider the online version of the problem, where a job becomes known upon its release date. We say an online algorithm is *α-competitive*, if it always provides a solution such that its objective function value is at most α times the offline optimum of the same problem. For technical reasons, we do not use the Assumption 1 for our online algorithms.

The next example describe a feasible solution in case of one resource.

Example 1. *Suppose there are 3 jobs, $p_1 = 4$, $p_2 = p_3 = 1$, $r_1 = 0$, $r_2 = 3$, and $r_3 = 7$. If there are 3 replenishments, $R \in (\mathcal{Q}_0 \cap \mathcal{Q}_3 \cap \mathcal{Q}_7)$, and $(S_1 = 0, S_2 = 4, S_3 = 7)$ then $(S, \mathcal{Q})$ is a feasible solution.*

However, if there are only two replenishments in $\mathcal{Q}'$ at $t = 3$ and $t = 7$, then we need to schedule the jobs later, e.g., if $(S'_1 = 3, S'_2 = 7, S'_3 = 8)$, then $(S', \mathcal{Q}')$ is feasible. Observe that in the second solution we have saved a cost of a replenishment $(K_0 + K_1)$, however, the total completion time of the jobs has increased from 17 to 24.

See Figure 1 for illustration.

1.3 Results of this paper

In the first part of the paper we consider the offline variant of the problem. Throughout this part, we assume $c_S = \sum w_j C_j$, however, the results remain valid in case of $c_S = \sum w_j F_j$. First, we prove a theorem that describes a connection between $1|jrp, r_j| \sum w_j C_j + c_{\mathcal{Q}}$ and the classic scheduling program $1|r_j| \sum w_j C_j$.

We say that two optimization problems, A and B, are *equivalent* if we can get an optimal solution for problem A in polynomial time by using an oracle which solves B optimally in constant time, and vice versa. Of course, preparing the problem instance

for the oracle takes time as well, and it must be polynomial in the size of the instance of problem A (B).

Theorem 1 helps to get complexity results and algorithms for different variants of the combined joint replenishment and scheduling problem.

Theorem 1. *If $|\mathcal{R}|$ and $|\mathcal{T}|$ are constants, then $1|jrp, r_j|\sum w_jC_j + c_{\mathcal{Q}}$ is equivalent to $1|r_j|\sum w_jC_j$.*

From the proof of Theorem 1, we can observe that the equivalence of the two problems remains valid if we introduce the same restrictions on both problems (like $p_j = 1$ for each job in both problems).

Corollary 1. *If there are the same restrictions on the jobs then the two problems of Theorem 1 remains equivalent.*

Then, we prove NP-hardness for a variant of $1|r_j|\sum w_jC_j$ to achieve complexity results for $1|jrp, r_j|\sum w_jC_j + c_{\mathcal{Q}}$.

Theorem 2. *$1|r_j|\sum w_jC_j$ is NP-hard even if $|\{w_j : j \in \mathcal{J}\}| = 2$.*

The next corollary follows from Corollary 1 and Theorem 2.

Corollary 2. *$1|jrp, r_j|\sum w_jC_j + c_{\mathcal{Q}}$ is NP-hard even if $K_0 = K_1 = \ldots = K_s = 0$, and $|\{w_j : j \in \mathcal{J}\}| = 2$.*

We have another negative result:

Theorem 3. *$1|jrp, r_j, p_j = 1|\sum C_j + c_{\mathcal{Q}}$ is NP-hard even if $K_0 = 0$, and $K_i = 1$, $i = 1, \ldots, s$.*

Then, we prove that two variants of $1|jrp, r_j|\sum w_jC_j + c_{\mathcal{Q}}$ are solvable in polynomial time.

Theorem 4. *$1|jrp, r_j, p_j = 1, s = const.|\sum w_jC_j + c_{\mathcal{Q}}$ is solvable in polynomial time.*

Theorem 5. *$1|jrp, r_j, p_j = p, s = const.|\sum C_j + c_{\mathcal{Q}}$ is solvable in polynomial time.*

The second part of the paper deals with online problems. In these problems, we seek algorithms with the best competitive ratio. The next two theorems describe our positive results. First, we prove:

Theorem 6. *There is 2-competitive algorithm for $1|jrp, online, r_j, p_j = 1, s = 1|\sum C_j + c_{\mathcal{Q}}$.*

Then, more generally, we have:

Theorem 7. *There is 2-competitive algorithm for $1|jrp, online, r_j, p_j = 1, s = 1|\sum F_j + c_{\mathcal{Q}}$.*

The last results yield lower bounds on the best competitive ratio for different variants:

Theorem 8. *There is no $(3/2 - \varepsilon)$-competitive algorithm for any constant $\varepsilon > 0$ for $1|jrp, online, r_j, p_j = 1, s = 1|\sum C_j + c_{\mathcal{Q}}$.*

Theorem 9. *There is no $\left(\frac{\sqrt{5}+1}{2} - \epsilon\right)$-competitive algorithm for any constant $\epsilon > 0$ for $1|jrp, online, r_j, p_j = 1, s = 1|\sum w_jC_j + c_{\mathcal{Q}}$.*

2 Complexity results for the offline problem

In section 2.1, we prove Theorems 1, 2 and 3. Then, in section 2.2 we describe polynomial time dynamic programs for some variants of the problem to prove Theorems 4 and 5.

Recall that throughout this section, we have assumed $c_S = \sum w_j C_j$.

2.1 Equivalence with $1|r_j|\sum w_j C_j$ and consequences

Proof of Theorem 1. If $K_0 = K_1 = \ldots = K_s = 0$ then $1|jrp, r_j|\sum w_j C_j + c_{\mathcal{Q}}$ is exactly the same as $1|r_j|\sum w_j C_j$ (each resource can be replenished at every time moment without any cost).

For the other direction consider an instance I of $1|jrp, r_j|\sum w_j C_j + c_{\mathcal{Q}}$. We prove that we can get the optimal solution for I by solving a constant number of instances of $1|r_j|\sum w_j C_j$. First, we define $\left(2^{|\mathcal{R}|}\right)^{|\mathcal{T}|}$ replenishment structures for I:

$$W = \{\mathcal{Q} : \mathcal{Q}_t \subseteq \mathcal{R}, \text{ if } t \in \mathcal{T}, \text{ otherwise, } \mathcal{Q}_t = \emptyset\}.$$

Due to Assumption 1, there exits an optimal solution of $1|jrp, r_j|\sum w_j C_j + c_{\mathcal{Q}}$ with one of the above replenishment structures.

We define an instance $I_{\mathcal{Q}}$ of $1|r_j|\sum w_j C_j$ for each $\mathcal{Q} \in W$. The jobs have the same p_j, and w_j values as in I, the differences are only in their release dates. For a given $\mathcal{Q}$, let $r'_j := \min\{t \geq r_j : R(j) \subseteq \cup_{\tau \in [r_j, t]} Q_\tau\}$ be the release date of j in $I_{\mathcal{Q}}$. If this value is infinity for any job j, then there is no feasible schedule for $I_{\mathcal{Q}}$, because some resource in $R(j)$ is not replenished at or after r_j. Observe that, if S is a feasible solution of $I_{\mathcal{Q}}$, then $(\mathcal{Q}, S)$ is a feasible solution of I, and its objective function value is $\sum w_j C_j + c_{\mathcal{Q}}$, where C_j is the completion time of j in S.

Thus, if $(\mathcal{Q}^*, S^*)$ is an optimal solution of I then S^* is an optimal solution for $I_{\mathcal{Q}^*}$ and the optimal solution of $I_{\mathcal{Q}^*}$ yields an optimal solution for I with $\mathcal{Q}^*$. Invoking the oracle for each $I_{\mathcal{Q}}$ ($\mathcal{Q} \in W$), we can determine an optimal schedule $S^{\mathcal{Q}}$ for each $I^{\mathcal{Q}}$. From these schedules we can determine the value $\mathcal{O}_{\mathcal{Q}} := c_{\mathcal{Q}} + \sum_{j \in \mathcal{J}} w_j C_j^{\mathcal{Q}}$ for each $\mathcal{Q} \in W$, where $C_j^{\mathcal{Q}}$ denotes the completion time of j in $S^{\mathcal{Q}}$. Let $\tilde{\mathcal{Q}} \in W$ denote a replenishment structure such that $\mathcal{O}_{\tilde{Q}} = \min\{\mathcal{O}_{\mathcal{Q}} : \mathcal{Q} \in W\}$. Due to our previous observation, $(\tilde{\mathcal{Q}}, S^{\tilde{\mathcal{Q}}})$ is an optimal solution of I. □

Theorem 2 deduces the hardness of a variant of $1|r_j|\sum w_j C_j$ from the following well-known NP-hard problem.

3-PARTITION: A set $A := \{a_1, a_2, \ldots, a_{3q}\}$ is given, such that $\sum_{j=1}^{3q} a_j = qB$, and $B/4 < a_j < B/2$ for each $j = 1, \ldots, 3q$. Is there a partitioning of A into q triplets $A_1, \ldots, A_q$ such that $\sum_{j \in A_i} a_j = B$, $i = 1, \ldots, q$?

Proof of Theorem 2. We reduce the 3-PARTITION problem to $1|r_j|\sum w_j C_j$ under the restriction of the theorem. Consider an instance I of 3-PARTITION, and let the instance I' of $1|r_j|\sum w_j C_j$ be the following. There are $4q$ jobs, among them $3q$ so called *partition jobs* ($j = 1, \ldots, 3q$) correspond to the items in I: these jobs have unit weights, $r_j = 0$, and $p_j = a_j$. There are $q - 1$ so called *separating jobs*

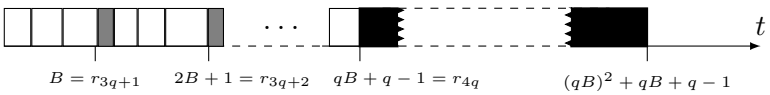

Figure 2: Illustration of the schedule. Partition jobs are white, separating jobs are gray, and the long job is black.

$(j = 3q + 1, \ldots, 4q - 1)$ with $p_j = 1, w_j = (qB)^2$, and $r_j = (j - 3q) \cdot (B + 1) - 1$. The last job (so called *long job*) has $p_{4q} = w_{4q} = (qB)^2$, and $r_{4q} = qB + q - 1$.

We prove that 3-PARTITION has a solution if and only if there is a solution for $1|r_j|\sum w_jC_j$ with an objective function value of at most

$$\mathcal{O} := (qB)^2 \cdot \left(\sum_{j=3q+1}^{4q-1} (r_j + 1) + ((qB)^2 + qB + q - 1) \right) + 3 \cdot \sum_{j=3q+1}^{4q} r_j. \qquad (1)$$

If 3-PARTITION has a solution, then schedule the separating jobs and the long job from their release dates (the total weighted completion time of these jobs is exactly the first part of (1)). After that, schedule the jobs corresponding to items of A_i $(i = 1, \ldots, q)$ in arbitrary order from $r_{3q+i} - B$ without any gap. Note that the jobs do not overlap (and there is no gap) in this schedule, since the total processing time of the jobs correspond to items of A_i is B. A job that corresponds to an item of A_i has a completion time at most r_{3q+i} in this schedule, thus the total weighted completion time of the partition jobs is at most $3 \cdot \sum_{j=3q+1}^{4q} r_j$, and $\sum w_jC_j \leq \mathcal{O}$ for the whole schedule. See Figure 2 for illustration.

Now suppose that there is a schedule S with an objective function value of at most $\mathcal{O}$. If all of the separating jobs and the long job start as soon as possible (i.e., at its release date), then their total weighted completion time is $(qB)^2 \cdot \left(\sum_{j=3q+1}^{4q-1}(r_j + 1) + ((qB)^2 + qB + q - 1)\right)$. Observe that $3 \cdot \sum_{j=3q+1}^{4q} r_j < (qB)^2$, where the latter value is the weight of a separating job or the long job. Thus, if any of these jobs starts later than its release date, the total weighted completion time of S would be greater than $\mathcal{O}$. Hence, we know $S_j = r_j$ for $j = 3q+1, \ldots, 4q$, and the total weighted completion time of the partition jobs in S is at most $3 \cdot \sum_{j=3q+1}^{4q} r_j$.

Now consider the partial schedule S', where there are no partition jobs, and $S'_j = r_j$ for the other jobs. There are q gaps in this schedule, and the length of any gap is B, thus their total length is qB. It means if there is a gap in S, then at least one partition job starts after the long job, i.e., after $qB + q - 1 + (qB)^2$. This value is greater than $3 \cdot \sum_{j=3q+1}^{4q} r_j$, therefore, we would be in contradiction with our previous observation on the total weighted completion time of the partition jobs. Thus, there is no gap in S, which means the machine is working on partition jobs in each $[r_j - B, r_j]$, $j = 3q + 1, \ldots, 4q$ (i.e., in the gaps of S'). Each of these intervals contains exactly three partition jobs, because we have $B/4 < a_j < B/2$ for $j = 1, \ldots, 3q$ in the 3-PARTITION problem. The items corresponding to a triplet of jobs in the same interval yield a solution of the 3-PARTITION problem. □

Theorem 3 shows that the $1|jrp, r_j, p_j = 1|\sum C_j + c_{\mathcal{Q}}$ is NP-hard even if $K_0 = 0$,

and $K_i = 1$ for each resource i (cf. $1|r_j, p_j = 1|\sum C_j$ is a trivial problem). We use the MAX CLIQUE problem (known NP-complete problem) for the reduction.

MAX CLIQUE: Given a graph $G = (V, E)$, and $k \in \mathbb{Z}$. Does there exist a complete subgraph (clique) of G with k nodes?

Proof of Theorem 3. We reduce the MAX CLIQUE problem to the above variant of $1|jrp, r_j, p_j = 1|\sum C_j + c_{\mathcal{Q}}$. Consider an instance I of MAX CLIQUE $(G = (V, E), k)$, from I we create an instance I' of $1|jrp, r_j, p_j = 1|\sum C_j + c_{\mathcal{Q}}$. We define $|V| + M$ jobs (where M is a sufficiently large number), and $s = |E|$ resources in I'. Each resource correspond to an edge, while there are $|V|$ jobs correspond to every node in V. The release date of these jobs is $r' = 0$, and such a job j requires a resource r if and only if the node corresponding to job j is an endpoint of the edge corresponding to resource r. We define M further jobs with a release date $r'' = |V| - k$, which require every resource.

We claim that there is a solution for the MAX CLIQUE, if and only if there is solution of $1|jrp, r_j, p_j = 1|\sum C_j + c_{\mathcal{Q}}$ with an objective function value of $\mathcal{O} = 2|E| - \binom{k}{2} + \binom{M+|V|+1}{2}$. If there is a k-clique $G' = (V', E')$ in G then we introduce two replenishments: at $r' = 0$ we replenish every resource that correspond to an edge in $E \setminus E'$, and at r'', we replenish every resource. Observe that the replenishment cost is $2|E| - |E'| = 2|E| - \binom{k}{2}$. Then, consider the following schedule S: schedule the jobs that correspond the nodes of $V \setminus V'$ and has a release date 0 from $t = 0$ in arbitrary order, after that schedule the remaining jobs in arbitrary order. It is easy to see that S is feasible since the number of the jobs scheduled at the first step is $|V| - k = r''$. The total completion time of the jobs is $\binom{|V|+M+1}{2}$, thus the total cost is $\mathcal{O}$.

Now suppose that there is a solution of $1|jrp, r_j, p_j = 1|\sum C_j + c_{\mathcal{Q}}$ with an objective function value of $\mathcal{O}$. Observe that each resource should be replenished at r'', since the jobs with a release date r'' require every resource. The cost of this replenishment is $|E|$. If there is a gap in the schedule before r'' then the total completion time of the jobs is at least $\binom{|V|+M+2}{2} - (|V| - k) = \binom{|V|+M+1}{2} + M + 1 + k$. Hence, there is no gap in the schedule before r'' (supposing M is large enough). It means the resources replenished at $r' = 0$ are sufficient for $r'' = |V| - k$ jobs. The total completion time of a schedule without any gap is $\binom{|V|+M+1}{2}$, thus at most $|E| - \binom{k}{2}$ resources can be replenished at $r' = 0$, otherwise, the objective function value would exceed $\mathcal{O}$. This means at least $|V| - k$ nodes have at most $|E| - \binom{k}{2}$ incident edges. Consider the remaining at most k nodes of the graph. According to the previous observation, there are at least $\binom{k}{2}$ edges among them, which means this subgraph is a k-clique. □

2.2 Polynomial time algorithms with dynamic programming

In this section we show two algorithms to prove Theorems 4 and 5. Both algorithms apply a dynamic programming approach. Throughout this section we assume that each job j requires a single resource only, and with a slight abuse of notation, let $R(j)$ be this resource.

Proof of Theorem 4. First, we determine several so called 'states' by an algorithm that describes some attributions of different $(\mathcal{Q}^p, S^p)$ pairs, where $\mathcal{Q}^p$ is a replenishment

structure and S^p is a partial schedule. We will prove that there exists a state from which we can determine an optimal solution of the problem.

The states are arranged in $|\mathcal{T}| + 1$ layers, $Layer(\tau_1), \ldots, Layer(\tau_{|\mathcal{T}|+1})$, where $Layer(\tau_i)$ is the i^{th} *layer* ($i = 1, \ldots, |\mathcal{T}| + 1$). The first layer has only one state N_0 (that describes the attributions of $\mathcal{Q}^{N_0} = (\emptyset, \ldots, \emptyset)$ and S^{N_0}, which is the empty partial schedule). The algorithm builds the states of the i^{th} layer from the states of the $(i-1)^{th}$ layer, i.e., each state has a 'parent state' in the previous layer, except N_0. Each state N of the i^{th} layer has a corresponding partial schedule S^N which is also recorded, and N contains some details about a replenishment structure $\mathcal{Q}^N$ such that $\mathcal{Q}^N_t = \emptyset$ if $t \notin \{\tau_1, \ldots, \tau_{i-1}\}$. For technical reasons N contains some information about S^N. Throughout the algorithm, the states of the i^{th} layer has a corresponding schedule where the last job completes not later than τ_i.

Formally, in the i^{th} layer, the states are of the form $N = [\tau_i; \alpha_1, \ldots, \alpha_s; \beta_1, \ldots, \beta_s; \gamma_1, \ldots, \gamma_s; \delta]$, where α_i ($i = 1, \ldots, s$) represents the number of the jobs from $\mathcal{J}_i$ that are scheduled in S^N, β_i ($i = 1, \ldots, s$) represents the time of the last replenishment from R_i in $\mathcal{Q}^N$ (i.e., the largest time moment t such that $R_i \in \mathcal{Q}^N_t$) or $'-'$ if there is no such replenishment in $\mathcal{Q}^N$, γ_i ($i = 1, \ldots, s$) represents the number of the replenishments from R_i in $\mathcal{Q}^N$ (the number of time moments t such that $R_i \in \mathcal{Q}^N_t$), and δ is the total number of replenishments in $\mathcal{Q}^N$ (the number of time moments t such that $\mathcal{Q}^N_t \neq \emptyset$).

As we have mentioned earlier, for each state N we also record S^N, which can be stored by n numbers that describe the completion time of the jobs, or equals to 0, if the corresponding job is not scheduled in S^N. Note that recording replenishment structures could require exponential time (since there are $2^{|\mathcal{T}|}$ options even in case of $|\mathcal{R}| = 1$), thus we record only the numbers β_i, γ_i, and δ in a state. Later, we will determine a replenishment structure to each N from these numbers.

We determine states with the following procedure:

Algorithm 1

Initialization: $k = 1$, $N_0 := [\tau_1; 0, \ldots, 0; -, \ldots, -; 0, \ldots, 0; 0]$, S^{N_0} is the empty schedule, $Layer(\tau_1) := \{N_0\}$

1. For each state $N := [\tau_k; \alpha_1, \ldots, \alpha_s; \beta_1, \ldots, \beta_s; \gamma_1, \ldots, \gamma_s; \delta] \in Layer(\tau_k)$ and for each subset $\bar{\mathcal{R}}$ of $\{R_1, \ldots, R_s\}$ do steps 2-7:
2. For each resource R_i, $i \in \{1, \ldots, s\}$, determine $\bar{\beta}_i$ and $\bar{\gamma}_i$ as follows: if $R_i \in \bar{\mathcal{R}}$, then $\bar{\beta}_i := \tau_k$ and $\bar{\gamma}_i := \gamma_i + 1$; otherwise, $\bar{\beta}_i := \beta_i$ and $\bar{\gamma}_i := \gamma_i$.
3. If $\bar{\mathcal{R}} = \emptyset$, then let $\bar{\delta} := \delta$, otherwise, let $\bar{\delta} := \delta + 1$.
4. Let $\bar{\mathcal{J}}_i \subseteq \mathcal{J}_i$ denote the jobs that have a release date at most $\bar{\beta}_i$ $(i = 1, \ldots, s)$. Let $\mathcal{J}^k \subseteq \cup_{i=1}^{s} \bar{\mathcal{J}}_i$ be the $\min\{\tau_{k+1} - \tau_k, \sum_{i=1}^{s}(|\bar{\mathcal{J}}_i| - \alpha_i)\}$ unscheduled jobs with the largest weights. Let $\bar{\alpha}_i := \alpha_i + |\{j \in \mathcal{J}^k : R(j) = i\}|$ for $i = 1, \ldots, s$. Let $\bar{\tau} := \tau_{k+1}$.
5. Let $\bar{N} := [\bar{\tau}; \bar{\alpha}_1, \ldots, \bar{\alpha}_s; \bar{\beta}_1, \ldots, \bar{\beta}_s; \bar{\gamma}_1, \ldots, \bar{\gamma}_s; \bar{\delta}]$.
6. Consider the partial schedule S^N associated with N, and schedule the jobs of $\mathcal{J}^k$ in non-increasing w_j order from τ_k without idle time. Let $\bar{S}^{new}$ be the resulting partial schedule.
7. If $\bar{N} \notin Layer(\bar{\tau})$, then add $\bar{N}$ to $Layer(\bar{\tau})$, let N be its parent node and associate the schedule $\bar{S}^{new}$ with the new node $\bar{N}$. Otherwise, if $\bar{N} \in Layer(\bar{\tau})$, then let $S^{\bar{N}}$ be the schedule associated with $\bar{N}$. If the total weighted completion time of $\bar{S}^{new}$ is smaller than that of $S^{\bar{N}}$, replace $S^{\bar{N}}$ to $\bar{S}^{new}$ and modify the parent node of $\bar{N}$ to N.
8. If $k < |\mathcal{T}|$ then let $k := k + 1$ and go to the first step.

Throughout the procedure we record the *parent state* of each state $N \neq N_0$, i.e., the state of the previous layer that was used during the creation of N. Observe that the number of the states is at most n^{3s+2}, because there are at most n different options in each coordinate of a state. This observation shows that the above procedure is polynomial in the size of the input parameters of the problem.

For each state N, we determine replenishments $\mathcal{Q}^N$ as follows. If $N \in Layer(\tau_1)$ then $\mathcal{Q}^N := (\emptyset, \ldots, \emptyset)$. Otherwise, if $N \in Layer(\tau_{k+1})$ $(1 \leq k \leq |\mathcal{T}|)$, we copy the replenishments $\mathcal{Q}^{N'}$ from its parent state N', and add a new replenishment at τ_k from resources $\bar{\mathcal{R}}$, where $\bar{\mathcal{R}}$ is the selected subset of resources when the algorithm creates N from N' (cf. step 1 of the algorithm). If $\bar{\mathcal{R}} = \emptyset$, then $\mathcal{Q}^N = \mathcal{Q}^{N'}$ and there is no new replenishment. Note that determining $\mathcal{Q}^N$ for each state N requires polynomial time.

Lemma 1. *Let $N = [\tau_k; \alpha_1, \ldots, \alpha_s; \beta_1, \ldots, \beta_s; \gamma_1, \ldots, \gamma_s; \delta] \in Layer_k$ be an arbitrary state.*

(i) Each job in S^N completes not later than τ_k.

(ii) The jobs do not overlap in S^N.

(iii) *If β_i is a number (i.e., $\beta_i \neq -$), then there is a replenishment from R_i at β_i in $\mathcal{Q}^N$ ($R_i \in \mathcal{Q}^N_{\beta_i}$).*

(iv) *There are δ replenishments in $\mathcal{Q}^N$ of which there are γ_i replenishments from R_i ($i = 1, \ldots, s$).*

(v) *If each $\alpha_i = |\mathcal{J}_i|$ then the full schedule S^N is feasible for $\mathcal{Q}^N$.*

(vi) *The cost of the replenishments in $\mathcal{Q}^N$ can be determined from N in polynomial time.*

Proof. (i) Follows from the observation that $|\mathcal{J}^k| \leq \tau_{k+1} - \tau_k$ in Step 4.

(ii) Follows from (i).

(iii) Trivial.

(iv) Follows from Algorithm 1.

(v) Each job is scheduled because $\alpha_i = |\mathcal{J}_i|$ and the jobs do not overlap due to (ii). Observe that Algorithm 1 cannot schedule a job before its release date. If it chooses a job $j \in \mathcal{J}_i$ to schedule in $[\tau_k, \tau_{k+1}]$ at step 4, it has a release date at most β_i since $j \in \bar{\mathcal{J}}_i$, thus the lemma follows from (iii).

(vi) Follows from (iv). The cost of $\mathcal{Q}^N$ is $K_0 \cdot \delta + \sum_{i=1}^{s} K_i \cdot \gamma_i$.

□

Consider the set $\mathcal{J}^k$ at Step 4 (the jobs that are scheduled in that iteration) when the algorithm creates an arbitrary state N from its parent state N'.

Claim 1. *If $|\mathcal{J}^k| < \tau_{k+1} - \tau_k$, then $\mathcal{J}^k$ is the set of available jobs at τ_k in case of $\mathcal{Q}^N$ that are unscheduled in $S^{N'}$. Otherwise, it is the $\tau_{k+1} - \tau_k$ elements of this set with the largest weight.*

Proof. Follows from the definitions. □

After determining the states by Algorithm 1, and determining $\mathcal{Q}^N$ for each state N, we choose a state $\hat{N} \in \mathit{Layer}(\tau_{|\mathcal{T}|+1})$, and later we prove that $(\mathcal{Q}^{\hat{N}}, S^{\hat{N}})$ is an optimal solution. For each state N of the last layer we decide whether $(\mathcal{Q}^N, S^N)$ is feasible or not (by Lemma 1 (v)), calculate the cost of $\mathcal{Q}^N$ (by Lemma 1 (vi)), and the total weighted completion time of S^N (since S^N is recorded). We define $\hat{N}$ as the state such that $(\mathcal{Q}^{\hat{N}}, S^{\hat{N}})$ is feasible, and it has best objective function value among that of the states of the last layer with a feasible solution. Observe that this procedure requires polynomial time.

Now we prove that there is a state of the last layer such that the corresponding solution of the problem is optimal, thus the solution $(\mathcal{Q}^{\hat{N}}, S^{\hat{N}})$ calculated by our method must be optimal. Let $(\mathcal{Q}^*, S^*)$ be an optimal solution. Observe that, the jobs in $[\tau_k, \tau_{k+1}]$ in S^* are in non-increasing w_j order, and by Assumption 1 we know that $\mathcal{Q}^*_t = \emptyset$ if $t \notin \mathcal{T}$. Recall that N_0 is the only state of $\mathit{Layer}(\tau_1)$. Let $N_k \in \mathit{Layer}(\tau_{k+1})$

be the state that we get from N_{k-1}, when $\bar{\mathcal{R}}$ was set to $\mathcal{Q}^*_{\tau_k}$ in Algorithm 1 ($k = 1, \ldots, |\mathcal{T}|$). Due to the feasibility of the optimal solution, there is a replenishment from R_i not earlier than $\max_{j \in \mathcal{J}_i} r_j$, thus Algorithm 1 will schedule every job in $S^{N_{|\mathcal{T}|}}$, i.e., each $\alpha_i = |\mathcal{J}_i|$ in $N_{|\mathcal{T}|}$. Therefore, $(\mathcal{Q}^{N_{|\mathcal{T}|}}, S^{N_{|\mathcal{T}|}})$ is feasible by Lemma 1 (v).

We prove that $(\mathcal{Q}^{N_{|\mathcal{T}|}}, S^{N_{|\mathcal{T}|}})$ is an optimal solution. Observe that $\mathcal{Q}^* \equiv \mathcal{Q}^{N_{|\mathcal{T}|}}$ (i.e., the coordinates of the $\mathcal{Q}^*$ and that of $\mathcal{Q}^{N_{|\mathcal{T}|}}$ are the same). It remained to prove that the total weighted completion time is the same for S^* and $S^{N_{|\mathcal{T}|}}$. The next claim describes an important observation on these schedules.

Claim 2. *The machine is working in the same periods in S^* and in $S^{N_{|\mathcal{T}|}}$.*

Proof. Suppose for contradiction that $\tau_k \leq t < \tau_{k+1}$ is the first time point, when the machine becomes idle either in S^* or in $S^{N_{|\mathcal{T}|}}$, but not in the other.

On the one hand, if the machine is idle from t only in S^*, then there are more jobs scheduled until $t+1$ in $S^{N_{|\mathcal{T}|}}$ than in S^*. Hence, there exists a resource R_i such that there are more jobs scheduled from $\mathcal{J}_i$ (the set of jobs that require R_i) until $t+1$ in $S^{N_{|\mathcal{T}|}}$ than in $S^{N_{|\mathcal{T}|}}$. This means there is at least one job $j \in \mathcal{J}_i$ with a release date not later than the last replenishment before $t+1$ from R_i, which starts later than t in S^*, otherwise, $S^{N_{|\mathcal{T}|}}$ would not be feasible. However, if we modify the starting time of j in S^* to t (the starting time of other jobs does not change) then we would get a schedule S' such that $(\mathcal{Q}^*, S')$ is better than $(\mathcal{Q}^*, S^*)$, a contradiction.

On the other hand, if the machine is idle from t only in $S^{N_{|\mathcal{T}|}}$, then there is a resource R_i such that there are more jobs scheduled from $\mathcal{J}_i$ until $t+1$ in S^* than in $S^{N_{|\mathcal{T}|}}$. The machine is idle from t in $S^{N_{|\mathcal{T}|}}$, thus when Algorithm 1 creates N_k, it chooses a set $\mathcal{J}^k$ at step 4 such that $|\mathcal{J}^k| < \tau_{k+1} - \tau_k$. Hence, we know from Claim 1 that each job from $\bar{\mathcal{J}}_i$ is scheduled until t in S^{N_k} at that step. Observe that these jobs are scheduled until t also in $S^{N_{|\mathcal{T}|}}$, and all the other jobs from $\mathcal{J}_i$ must be scheduled after τ_{k+1} any feasible schedule for replenishment structure $\mathcal{Q}^*$. It is in a contradiction with the definition of $\mathcal{J}_i$. □

From Claim 1 we know that Algorithm 1 always schedules a job with highest weight among the unscheduled available jobs during the creation of $S^{N_{|\mathcal{T}|}}$, if the latter set is non-empty. We claim that this property is even valid for S^*, thus for every time moment t, the job scheduled at t in $S^{N_{|\mathcal{T}|}}$ has the same weight as the job scheduled at t in S^*, therefore the total weighted completion time is the same for these schedules.

Suppose for contradiction that this property is not true S^*. Let j_1 be the first job that contradicts this property, i.e., there is a job j_2 such that $w_{j_2} > w_{j_1}$, $S^*_{j_2} > S^*_{j_1}$, and j_1 is available at $S^*_{j_1}$ in $\mathcal{Q}^*$. Let S' be the schedule such that $S'_{j_1} := S^*_{j_2}$, $S'_{j_2} := S^*_{j_1}$, and $S'_j := S^*_j$ for each $j \neq j_1, j_2$. Observe that (i) j_2 is available at S'_{j_2} in $\mathcal{Q}^*$, because it was supposed, (ii) j_1 is available at S'_{j_1} in $\mathcal{Q}^*$, because $r_{j_1} \leq S^*_{j_1} \leq S'_{j_1}$, and $\exists t \in [r_{j_1}, S^*_{j_1}]$ such that there is a replenishment from the resource required by j_1, and (iii) each other job j is available at S'_j in $\mathcal{Q}^*$, because $S'_j = S^*_j$. Thus schedule S' is feasible for $\mathcal{Q}^*$. The total weighted completion time of S' is smaller than that of S^*, which contradicts that $(\mathcal{Q}^*, S^*)$ is optimal.

Therefore, we have proved the theorem. □

We can prove Theorem 5 with a slightly modified version of the previous algorithm.

Proof of Theorem 5 (sketch). We sketch a similar dynamic program to that of Theorem 4. Let $\mathcal{T}' := \{\tau + \lambda \cdot p : \tau \in \mathcal{T}, 0 \leq \lambda \leq n\} = \{\tau'_1 < \ldots < \tau'_{|\mathcal{T}'|}\}$. Observe that $|\mathcal{T}'| \in O(n^2)$, and $\mathcal{T}'$ contains all the possible starting and completion times of the jobs in any schedule without unnecessary idle times.

We modify Algorithm 1 as follows. There are $|\mathcal{T}'| + 1$ layers and each state is of the form $[\tau'; \alpha_1, \ldots, \alpha_s; \beta_1, \ldots, \beta_s; \gamma_1, \ldots, \gamma_s; \delta]$, where $\tau' \in \mathcal{T}'$, and all the states with the same τ' constitute $Layer(\tau')$. The other coordinates have the same meaning as before.

Let $N = [\tau'; \alpha_1, \ldots, \alpha_s; \beta_1, \ldots, \beta_s; \gamma_1, \ldots, \gamma_s; \delta] \in Layer(\tau')$ be the state chosen in Step 1 of the Algorithm 1. Steps 2 and 3 remain the same and we modify step 4 as follows. We define $\mathcal{J}^k$ as the set of every unscheduled job from $\cup_{i=1^s} \bar{\mathcal{J}}_i$, and $\bar{\alpha}_i$ in the same way as in Algorithm 1. If $\mathcal{J}^k \neq \emptyset$, then $\bar{\tau}' = \tau' + |\mathcal{J}^k| \cdot p$ (clearly, $\bar{\tau}' \in \mathcal{T}'$ by the definition of $\mathcal{T}'$). If $\mathcal{J}^k = \emptyset$, then $\bar{\tau}'$ of $\bar{N}$ equals the next member of $\mathcal{T}'$ after τ' of N. The remaining steps are the same.

The proof of soundness of the modified dynamic program is analogous to that of Theorem 4. □

Remark 1. *The results of theorems 4 and 5 remain valid even if the jobs may require more than one resource, i.e., we do not suppose that the sets $\mathcal{J}_i$ are disjoint. The only difference in the algorithms is in Step 4. We allows $j \in \mathcal{J}^k$ only if j is available at τ_k.*

3 Competitive analysis of the online problem

In the online version of the problem, we do not know the number of the jobs or anything about them before their release dates. In the next section, we describe a 2-competitive algorithm for two online variants of the problem.

3.1 Minimizing the total completion time and the total replenishment cost

To prove Theorem 6, we show a 2-competitive algorithm for the online problem in case of $s = p_j = w_j = 1$, i.e., there is only one resource and the processing time and the weight of each job is one. Due to the latter assumption, we can suppose that the order of the jobs is the same in any schedule (a fixed non-decreasing release date order). To simplify the notation, we introduce $K := K_0 + K_1$ for the cost of a replenishment.

At each time point t, first we can examine the jobs released at t, then we need to decide whether we want a replenishment or not, and finally we can schedule a job from t such that we have the required resources. Note that, if we need to decide about the replenishment *before* we know about the freshly released jobs, then the problem is the same as the previous problem where each of the release dates is increased by 1.

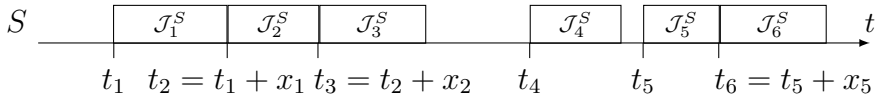

Figure 3: Schedule S created by Algorithm 2.

Algorithm 2

Initialization: $t := 0$

1. If there is an available job, then schedule it, and go to step 3.
2. Let $\mathcal{J}_{us}^t$ denote the set of unscheduled jobs at t. If $t \cdot |J_{us}^t| + \frac{|\mathcal{J}_{us}^t| \cdot (|\mathcal{J}_{us}^t|+1)}{2} \geq K$, then replenish the resource and schedule a job.
3. $t := t + 1$, and go to step 1.

Observe that $t \cdot |J_{us}^t| + \frac{|J_{us}^t| \cdot (|J_{us}^t|+1)}{2}$ is the total completion time of the jobs of J_{us}^t, if they are scheduled from t without any gap.

Let $(S, \mathcal{Q})$ denote the solution created by Algorithm 2, while $(S^*, \mathcal{Q}^*)$ an arbitrary optimal solution. Let t_i denote the time moment of the i^{th} replenishment in $\mathcal{Q}$ and $t_0 := 0$. The notation $\mathcal{J}_i^S \subseteq \mathcal{J}$, $i = 1, 2, \ldots$ denotes the set of jobs scheduled in $[t_i, t_{i+1})$ in S, see Figure 3.

Clearly, the release date r_j of a job $j \in \mathcal{J}_i^S$ has to be in $[t_{i-1} + 1, t_i]$. For technical reasons we introduce $x_i := |\mathcal{J}_i^S|$, $z_i := |\{j \in \mathcal{J}_i^S : r_j = t_i\}|$, and $y_i := x_i - z_i$. The next observation follows from the condition of step 2 of Algorithm 2.

Observation 1. *If the machine is idle in $[t_i - 1, t_i]$, then $y_i(t_i - 1) + \frac{y_i \cdot (y_i+1)}{2} < K$.*

The divide the $v(S, \mathcal{Q})$ among the set of jobs $\mathcal{J}_i^S$ in the following way: for $i = 1, 2, \ldots$, let

$$ALG_i := K + \sum_{j \in \mathcal{J}_i^S} C_j = K + t_i x_i + \frac{x_i \cdot (x_i + 1)}{2},$$

i.e., the total completion time of the jobs of $\mathcal{J}_i^S$ in S plus K, which is the cost of the replenishment at t_i. Since the sets $\mathcal{J}_i^S$ are disjoint, we have $v(S, \mathcal{Q}) = \sum_{i \geq 1} ALG_i$. Finally, note that any gap in S has to be finished at t_i for some $i \geq 1$. We divide the optimum value in a similar way: we introduce values OPT_i, $i = 1, 2 \ldots$, where

$$OPT_i := \begin{cases} K + \sum_{j \in \mathcal{J}_i^S} C_j^*, & \text{if } \cup_{t=t_{i-1}+1}^{t_i - 1} \mathcal{Q}_t^* \neq \emptyset. \\ \sum_{j \in \mathcal{J}_i^S} C_j^*, & \text{otherwise.} \end{cases}$$

Observe that the $v(S^*, \mathcal{Q}^*) \geq \sum_{i \geq 1} OPT_i$.

Now we show that Algorithm 2 is 2-competitive to prove Theorem 6.

Proof of Theorem 6. Suppose that there is a gap in S before t_i ($i \geq 1$), and the next gap starts in $[t_{i+\ell}, t_{i+\ell+1})$. Then, we have $\sum_{\mu=0}^{\ell} x_{i+\mu}$ jobs scheduled between the two

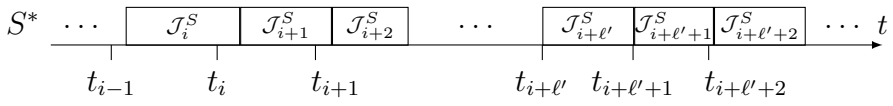

Figure 4: A possible realization of S^*.

neighboring gaps, and their completion time is $t_i + \nu$, where $\nu = 1, 2, \ldots, \sum_{\mu=0}^{\ell} x_{i+\mu}$. Hence,

$$\sum_{\mu=0}^{\ell} ALG_{i+\mu} = (\ell+1)K + t_i \sum_{\mu=0}^{\ell} x_{i+\mu} + \frac{(\sum_{\mu=0}^{\ell} x_{i+\mu}) \cdot (\sum_{\mu=0}^{\ell} x_{i+\mu} + 1)}{2}.$$

We prove $\sum_{\mu=0}^{\ell} ALG_{i+\mu} \leq 2 \cdot \sum_{\mu=0}^{\ell} OPT_{i+\mu}$, thus the theorem follows from the previous observations on ALG_i and OPT_i.

Let $0 \leq \ell' \leq \ell$ be the smallest index such that there is no replenishment in $[t_{i+\ell'-1}+1, t_{i+\ell'}-1]$ in $\mathcal{Q}^*$, i.e., $\cup_{t=t_{i+\ell'-1}+1}^{t_{i+\ell'}-1} \mathcal{Q}_t^* = \emptyset$. If there is no such index then let $\ell' := \ell+1$.

Claim 3. *$ALG_{i+\mu} \leq 2 \cdot OPT_{i+\mu}$ for all $\mu \in [\ell', \ell]$.*

Proof. If $\ell' = \ell + 1$, then the claim is trivial.

Otherwise, we have $r_j \geq t_{i+\ell'-1} + 1 =: t'$ for each job $j \in \cup_{\mu=\ell'}^{\ell} \mathcal{J}_{i+\mu}^S$, hence they cannot be scheduled before the first replenishment after t'. The first replenishment after t' in $\mathcal{Q}^*$ is not earlier than $t_{i+\ell'}$ in S^* due to the definition of ℓ'. We have assumed that the order of the jobs is the same in every schedule, thus, we have $C_j^* \geq C_j$ for each $j \in \cup_{\mu=\ell'}^{\ell} \mathcal{J}_{i+\mu}^S$. Since $OPT_{i+\mu} \geq \sum_{j \in \mathcal{J}_{i+\mu}^S} C_j^* \geq \sum_{j \in \mathcal{J}_{i+\mu}^S} C_j \geq K$ for all $\mu \in [\ell', \ell]$, where the last inequality follows from the condition of step 2 of Algorithm 2, we have $ALG_{i+\mu} = K + \sum_{j \in \mathcal{J}_{i+\mu}^S} C_j \leq 2 \cdot OPT_{i+\mu}$ for all $\mu \in [\ell', \ell]$. □

If $\ell' = 0$, then we have proved the theorem. From now on, we suppose $\ell' \geq 1$. Claim 3 shows that it remained to prove $\sum_{\mu=0}^{\ell'-1} ALG_{i+\mu} \leq 2 \cdot \sum_{\mu=0}^{\ell'-1} OPT_{i+\mu}$, if there is a replenishment in every interval $[t_{i+\mu-1}+1, t_{i+\mu}-1]$, $\mu = 0, \ldots, \ell'-1$ in $\mathcal{Q}^*$. See Figure 4 for an illustration of a possible realization of S^*.

Observe that

$$\sum_{\mu=0}^{\ell'-1} OPT_{i+\mu} \geq \ell' K + (t_{i-1}+1) y_i + \frac{y_i \cdot (y_i+1)}{2} + \\ + t_i \left(z_i + \sum_{\mu=1}^{\ell'-1} x_{i+\mu} \right) + \frac{(z_i + \sum_{\mu=1}^{\ell'-1} x_{i+\mu}) \cdot (z_i + \sum_{\mu=1}^{\ell'-1} x_{i+\mu} + 1)}{2},$$

since $OPT_{i+\mu} = K + \sum_{j \in \mathcal{J}_{i+\mu}^S} C_j^*$ for all $0 \leq \mu \leq \ell'-1$, there are y_i jobs with a release date at least $t_{i-1}+1$, and another $z_i + \sum_{\mu=1}^{\ell'-1} x_{i+\mu}$ jobs with a release date at least t_i.

Therefore,

$$
\begin{aligned}
&2 \cdot \sum_{\mu=0}^{\ell'-1} OPT_{i+\mu} - \sum_{\mu=0}^{\ell'-1} ALG_{i+\mu} \\
&\geq \ell' K + 2(t_{i-1}+1)y_i + t_i \left(z_i + \sum_{\mu=1}^{\ell'-1} x_{i+\mu} \right) - t_i y_i + y_i \cdot (y_i + 1) + \\
&+ \left(z_i + \sum_{\mu=1}^{\ell'-1} x_{i+\mu} \right) \cdot \left(z_i + \sum_{\mu=1}^{\ell'-1} x_{i+\mu} + 1 \right) - \frac{(y_i + z_i + \sum_{\mu=1}^{\ell'-1} x_{i+\mu}) \cdot (y_i + z_i + \sum_{\mu=1}^{\ell'-1} x_{i+\mu} + 1)}{2} \\
&\geq \ell' K + (2t_{i-1} + 1 - (t_i - 1))y_i + t_i \left(z_i + \sum_{\mu=1}^{\ell'-1} x_{i+\mu} \right) \\
&\geq \ell' K + (2t_{i-1}+1)y_i - K + t_i \left(z_i + \sum_{\mu=1}^{\ell'-1} x_{i+\mu} \right) \geq 0,
\end{aligned}
$$

where the first inequality of the last line follows from Observation 1, and the second from $\ell' \geq 1$. □

Suppose that there is only one job with a release date 0. The algorithm schedules this job from $t = K - 1$, thus $v(S, \mathcal{Q}) = 2K$. The optimal solution is to schedule this job from $t = 0$, thus $v(S^*, \mathcal{Q}^*) = K + 1$. As K tends to infinity $2K/(K+1)$ tends to 2, which shows Theorem 6 is tight, i.e., Algorithm 2 cannot be an α-competitive algorithm for any $\alpha < 2$.

3.2 Minimizing the total flow time and the total replenishment cost

To prove Theorem 7, we describe a 2-competitive algorithm for the online version of the problem when there is only one resource, $p_j = 1$, and the objective is to minimize $\sum F_j + c_{\mathcal{Q}}$. Observe that this problem differs from the problem of the previous section only in the objective function. Hence, we can also suppose that the order of the jobs is the same in any schedule, and $K := K_0 + K_1$.

The algorithm is nearly the same as in the previous section and several parts of the proof are analogous.

Algorithm 3

Initialization: $t := 0$

1. If there is an available job, then schedule it, and go to step 3.
2. Let $\mathcal{J}_{us}^t$ denote the set of unscheduled jobs at t. If $\sum_{j \in \mathcal{J}_{us}^t} (t - r_j) + \frac{|\mathcal{J}_{us}^t| \cdot (|\mathcal{J}_{us}^t| + 1)}{2} \geq K$, then replenish the resource and schedule a job.
3. $t := t + 1$, and go to step 1.

Observe that $\sum_{j\in\mathcal{J}_{us}^t}(t-r_j)+\frac{|\mathcal{J}_{us}^t|\cdot(|\mathcal{J}_{us}^t|+1)}{2}$ is the total flow time of the jobs of $\mathcal{J}_{us}^t$ if they are scheduled from t without any gap.

Let $(S,\mathcal{Q})$ denote the solution created by Algorithm 3, while $(S^*,\mathcal{Q}^*)$ an arbitrary optimal solution. The notations $t_i,\mathcal{J}_i^S,x_i,y_i,z_i$ have the same meaning as in the previous section, see again Figure 3 for illustration. The next observation is analogous to Observation 1:

Observation 2. *If the machine is idle in $[t_i-1,t_i]$, then $\sum_{j\in\mathcal{J}_{us}^{t-1}}(t_i-1-r_j)+\frac{y_i\cdot(y_i+1)}{2}<K$.*

Let $ALG_i:=K+\sum_{j\in\mathcal{J}_i^S}F_j$ $(i\geq 1)$, thus we have $v(S,\mathcal{Q})=\sum_{i\geq 1}ALG_i$. Let s_i denote the number of replenishments in $\mathcal{Q}^*$ in $[t_{i-1}+1,t_i-1]$, and let $OPT_i:=s_iK+\sum_{j\in\mathcal{J}_i^S}F_j^*$. Observe that $v(S^*,\mathcal{Q}^*)\geq\sum_{i\geq 1}OPT_i$.

Now we prove Algorithm 3 is 2-competitive for the problem with one resource and $p_j=1$, if $c_S=\sum F_j$.

Proof of Theorem 7. Analogously to the proof of Theorem 6, we prove $\sum_{\mu=0}^{\ell}ALG_{i+\mu}\leq 2\cdot\sum_{\mu=0}^{\ell}OPT_{i+\mu}$ for each i such that in the schedule constructed by the algorithm, the machine is idle in $[t_i-1,t_i)$, and the next idle period starts in $[t_{i+\ell},t_{i+\ell+1})$, from which the theorem follows.

Recall the definition of ℓ' from the proof of Theorem 6. Then, the next claim is analogous to Claim 3.

Claim 4. *$ALG_{i+\mu}\leq 2\cdot OPT_{i+\mu}$ for all $\mu\in[\ell',\ell]$.*

Proof. If $\ell'=\ell+1$, then the claim is trivial.

Otherwise, we have $r_j\geq t_{i+\ell'-1}+1=:t'$ for each job $j\in\cup_{\mu=\ell'}^{\ell}\mathcal{J}_{i+\mu}^S$, hence they cannot be scheduled before the first replenishment after t'. The first replenishment after t' in $\mathcal{Q}^*$ is not earlier than $t_{i+\ell'}$ in S^* due to the definition of ℓ'. We have assumed that the order of the jobs is the same in every schedule, thus, we have $C_j^*\geq C_j$ and $F_j^*\geq F_j$ for each $j\in\cup_{\mu=\ell'}^{\ell}\mathcal{J}_{i+\mu}^S$. Since $OPT_{i+\mu}\geq\sum_{j\in\mathcal{J}_{i+\mu}^S}F_j^*\geq\sum_{j\in\mathcal{J}_{i+\mu}^S}F_j\geq K$ for all $\mu\in[\ell',\ell]$, where the last inequality follows from the condition of step 2 of Algorithm 3, we have $ALG_{i+\mu}=K+\sum_{j\in\mathcal{J}_{i+\mu}^S}F_j\leq 2\cdot OPT_{i+\mu}$ for all $\mu\in[\ell',\ell]$. □

If $\ell'=0$, then we are ready, thus from now on we suppose $\ell'\geq 1$. It remained to prove $\sum_{\mu=0}^{\ell'-1}ALG_{i+\mu}\leq 2\cdot\sum_{\mu=0}^{\ell'-1}OPT_{i+\mu}$, if there is at least one replenishment in every interval $[t_{i+\mu-1}+1,t_{i+\mu}-1]$, $\mu=0,\ldots,\ell'-1$ in $\mathcal{Q}^*$, i.e., $s_{i+\mu}\geq 1$ for every $\mu=0,\ldots,\ell'-1$.

First, we prove $ALG_i\leq 2\cdot OPT_i$. Let $\tau_{i,0}:=t_{i-1}$, and $\tau_{i,1}\leq\tau_{i,2}\leq\ldots\leq\tau_{i,s_i}$ be the time points of the replenishments in $[t_{i-1}+1,t_i-1]$ in $\mathcal{Q}^*$. Let $\tau_{i,s_i+1}:=t_i$. For $k=1,2,\ldots,s_i+1$, let $\mathcal{J}_i^S(k):=\{j\in\mathcal{J}_i^S:r_j\in(\tau_{i,k-1},\min\{t_i-1,\tau_{i,k}\}]\}\subseteq\mathcal{J}_{us}^{t_i-1}$. See Figure 5 for illustration. Note that these sets are pairwise disjoint and $\bigcup_{k=1}^{s_i+1}\mathcal{J}_i^S(k)=\{j\in\mathcal{J}_i^S:r_j<t_i\}=\mathcal{J}_{us}^{t_i-1}$, and their common size is y_i.

Each job in $\mathcal{J}_i^S(k)$ gets resource at $\tau_{i,k}$ in $\mathcal{Q}^*$, if $k\leq s_i$, thus they cannot start earlier than $\tau_{i,k}$ in S^*. The earliest start time of a job from $\mathcal{J}_i^s(s_i+1)$ is t_i. Since $F_j=(\tau_{i,k}-r_j)+(C_j-\tau_{i,k})$, thus we have $\sum_{j\in\mathcal{J}_i^S(k)}F_j\geq\sum_{j\in\mathcal{J}_i^S(k)}(\tau_{i,k}-r_j)+$

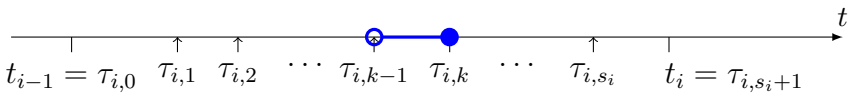

Figure 5: Replenishment times in $[t_{i-1}+1, t_i-1]$ in $\mathcal{Q}^*$. The release dates of the jobs of $\mathcal{J}_i^S(k)$ are from the blue interval.

$\frac{|\mathcal{J}_i^S(k)|\cdot(|\mathcal{J}_i^S(k)|+1)}{2}$ for any $1 \le k \le s_i+1$. Applying the previous inequality for each k and $\sum_{j\in\mathcal{J}_i^S\setminus\mathcal{J}_{us}^{t_i-1}} F_j \ge \frac{z_i\cdot(z_i+1)}{2}$, we get

$$OPT_i \ge s_iK + \sum_{k=1}^{s_i+1}\sum_{j\in\mathcal{J}_i^S(k)}(\tau_{i,k}-r_j) + \sum_{k=1}^{s_i+1}\frac{|\mathcal{J}_i^S(k)|\cdot(|\mathcal{J}_i^S(k)|+1)}{2} + \frac{z_i\cdot(z_i+1)}{2}$$
$$\ge s_iK + \sum_{k=1}^{s_i+1}\sum_{j\in\mathcal{J}_i^S(k)}(\tau_{i,k}-r_j) + \frac{\frac{y_i}{s_i+1}\cdot(\frac{y_i}{s_i+1}+1)}{2}\cdot(s_i+1) + \frac{z_i\cdot(z_i+1)}{2},$$

where the last inequality follows from simple algebraic rules. Furthermore, we have

$$ALG_i = K + \sum_{j\in\mathcal{J}_i^S} F_j = K + \sum_{j\in\mathcal{J}_{us}^{t_i-1}}(t_i-r_j) + \frac{y_i\cdot(y_i+1)}{2} + y_iz_i + \frac{z_i\cdot(z_i+1)}{2},$$

thus

$$2\cdot OPT_i - ALG_i \ge (2s_i-1)K + 2\cdot\sum_{k=1}^{s_i+1}\sum_{j\in\mathcal{J}_i^S(k)}(\tau_{i,k}-r_j) + y_i\left(\frac{y_i}{s_i+1}+1\right) +$$
$$+\frac{z_i\cdot(z_i+1)}{2} - \sum_{j\in\mathcal{J}_{us}^{t_i-1}}(t_i-r_j) - \frac{y_i\cdot(y_i+1)}{2} - y_iz_i$$
$$\ge (2s_i-2)K + 2\cdot\sum_{k=1}^{s_i+1}\sum_{j\in\mathcal{J}_i^S(k)}(\tau_{i,k}-r_j) + y_i\left(\frac{y_i}{s_i+1}+1\right) + \frac{z_i\cdot(z_i+1)}{2} -$$
$$-y_i - y_iz_i = (2s_i-2)K + 2\cdot\sum_{k=1}^{s_i+1}\sum_{j\in\mathcal{J}_i^S(k)}(\tau_{i,k}-r_j) + \frac{y_i^2}{s_i+1} + \frac{z_i\cdot(z_i+1)}{2} - y_iz_i,$$

where the second inequality follows from Observation 2 and from $|\mathcal{J}_{us}^{t_i-1}| = y_i$.

On the one hand, if $s_i = 1$, then $2\cdot OPT_i - ALG_i \ge y_i^2/2 + z_i^2/2 - y_iz_i \ge 0$. On the other hand, if $s_i \ge 2$, then we use again Observation 2, and we get $2\cdot OPT_i - ALG_i \ge (2s_i-3)K + \frac{y_i\cdot(y_i+1)}{2} + \frac{z_i\cdot(z_i+1)}{2} - y_iz_i \ge y_i^2/2 + z_i^2/2 - y_iz_i \ge 0$. Therefore, we have proved $ALG_i \le 2\cdot OPT_i$.

Now we prove $ALG_{i+\mu} \le 2\cdot OPT_{i+\mu}$ for any $1 \le \mu \le \ell'-1$. Let $1 \le \mu \le \ell'-1$, and $j' \in \mathcal{J}_{i+\mu}^S$ be arbitrary. Suppose that j' has the $n_{j'}^{th}$ smallest release date among the jobs in $\mathcal{J}_{i+\mu}^S$, i.e., it is the $\left(\sum_{\nu=1}^{i+\mu-1}\mathcal{J}_\nu^S + n_{j'}\right)^{th}$ job in the fixed non-decreasing

release date order that determine the order of the jobs in any schedule. Observe that $C_{j'} = t_i + \sum_{\nu=0}^{\mu-1} |\mathcal{J}_{i+\nu}^S| + n_{j'}$, because the algorithm schedules all of the jobs of $\bigcup_{\nu=0}^{\mu-1} \mathcal{J}_{i+\nu}^S$ from t_i without any gap, and after that, it starts to schedule the jobs from $\mathcal{J}_{i+\mu}^S$ also without any gap. However, it is possible that y_i jobs from $\mathcal{J}_i^S$ are scheduled before t_i in S^*, but every other jobs from $\bigcup_{\nu=0}^{\mu} \mathcal{J}_{i+\nu}^S$ has a release date at least t_i. Hence, we have $C_{j'}^* \geq t_i + z_i + \sum_{\nu=1}^{\mu-1} |\mathcal{J}_{i+\nu}^S| + n_{j'}$, since the order of the jobs is fixed. Since $|\mathcal{J}_i^S| = y_i + z_i$, we have $C_{j'} \leq C_{j'}^* + y_i$ and $F_{j'} \leq F_{j'}^* + y_i$.

Let $h := |\mathcal{J}_{i+\mu}^S|$. Since $\sum_{j \in \mathcal{J}_{i+\mu}^S} F_j^* \geq \frac{h \cdot (h+1)}{2}$, $OPT_{i+\mu} \geq K + \sum_{j \in \mathcal{J}_{i+\mu}^S} F_j^*$, and $ALG_i \leq K + \sum_{j \in \mathcal{J}_{i+\mu}^S} (F_j^* + y_i)$, we have

$$\begin{aligned} 2 \cdot OPT_{i+\mu} - ALG_{i+\mu} &\geq K + \frac{h \cdot (h+1)}{2} - hy_i \\ &\geq \frac{y_i \cdot (y_i+1)}{2} + \frac{h \cdot (h+1)}{2} - hy_i \geq 0, \end{aligned}$$

where the second inequality follows from Observation 2. □

4 Lower bounds on the best competitive ratio

In this section we prove Theorems 8 and 9.

Proof of Theorem 8. Suppose that there is only one job arrives at 0. If an algorithm schedules it at some time point t, then that algorithm cannot be better than $\frac{K+t+1}{K+1}$-competitive, because it is possible that no other jobs will arrive. However, if it schedules the first job at t, then it is possible that another job arrives at $t+1$. In this case the scheduling the jobs from $t+1$ with one replenishment requires a value $K + 2t + 5$, while $v(S, \mathcal{Q}) = 2K + 2t + 3$, thus the competitive ratio cannot be better than $\frac{2K+2t+3}{K+2t+5}$. Observe that if K is given, then the first ration is increases, while the second decreases as t increases. This means, we have to find a time point $\bar{t} \geq 0$ such that $\frac{K+\bar{t}+1}{K+1} = \frac{2K+2\bar{t}+3}{K+2\bar{t}+5}$, because the latter value is a lower bound on the best competitive ratio.

Some algebraic calculations show that $\bar{t} \in [K/2 - 5/4, K/2 - 1]$, thus the lower bound is at least $\frac{3/2K-1/4}{K+1}$. Therefore, for any $\varepsilon > 0$, there is a sufficiently large K, such that there is no $(3/2 - \varepsilon)$-competitive algorithm for the problem. □

Proof of Theorem 9. Suppose that job J_1 arrives at time 0 with weight $w_1 = 1$. If no other jobs arrive and the algorithm waits until time t before scheduling J_1 then it is at least $c_1(t)$-competitive where $c_1(t) = \frac{K+t+1}{K+1}$. If another job J_2 arrives at time $t+1$ with weight w_2, then the algorithm is at least $c_2(t)$-competitive where $c_2(t) = \frac{2K+t+1+(t+2)w_2}{K+t+2+(t+3)w_2}$. To get a lower bound for an arbitrary online algorithm, we want to calculate the value of

$$\max_{K, w_2} min_t max(c_1(t), c_2(t)).$$

It is easy to see that $c_1(t)$ is an increasing function of t, and if $K > w_2 + 1$ then $c_2(t)$ is a decreasing function of t. The second part can be proved by the following simple calculation:

$$c_2(t+1) - c_2(t) = \frac{(w_2+1)(w_2+1-K)}{(K+t+2+(t+3)w_2)(K+t+3+(t+4)w_2)} < 0$$

if $K > w_2+1$. So we get the best value $\bar{t} \geq 0$ when $c_1(t) = c_2(t)$. Solving the equation we get that

$$\bar{t} = \frac{\sqrt{4K^2w_2 - 4Kw_2^2 + 5K^2 + 2Kw_2 + 5w_2^2 + 4K + 4w_2} - K - 3w_2 - 2}{2(w_2+1)}$$

Substituting $\bar{t}$ into $c_1(t)$ we get the following formula

$$c(K, w_2) := \frac{\sqrt{4K^2w_2 - 4Kw_2^2 + 5K^2 + 2Kw_2 + 5w_2^2 + 4K + 4w_2} + K + 2Kw_2 - w_2}{2(w_2+1)(K+1)}.$$

So

$$\lim_{K\to\infty} c(K, w_2) = \frac{\sqrt{4w_2+5} + 2w_2 + 1}{2(w_2+1)},$$

and

$$\lim_{w_2\to 0} \frac{\sqrt{4w_2+5} + 2w_2 + 1}{2(w_2+1)} = \frac{\sqrt{5}+1}{2},$$

which gives the desired result. □

5 Conclusions

We have examined several variants of $1|jrp, r_j|c_S + c_{\mathcal{Q}}$, where the requests (jobs) require some processing by a single machine. We have proved complexity results, and showed polynomial time algorithms both for the offline and the online variant of the problem. However, several open questions remained, we list the ones that are the most interesting (for us). Is the offline problem solvable in polynomial time when $p_j = p$ and s is constant? What is the best competitive ratio for the considered online problems? What can we say in case of more complex machine environments?

Acknowledgment

This work has been supported by the National Research, Development and Innovation Office – NKFIH, grant no. SNN 129178, and ED_18-2-2018-0006. The research of Péter Györgyi was supported by the János Bolyai Research Scholarship of the Hungarian Academy of Sciences and by the MTA-ELTE Egerváry Research Group.

References

[1] F. Afrati, E. Bampis, C. Chekuri, D. Karger, C. Kenyon, S. Khanna, I. Milis, M. Queyranne, M. Skutella, C. Stein, and M. Sviridenko. Approximation schemes for minimizing average weighted completion time with release dates. In *40th Annual Symposium on Foundations of Computer Science (Cat. No. 99CB37039)*, pages 32–43. IEEE, 1999.

[2] E. J. Anderson and C. N. Potts. Online scheduling of a single machine to minimize total weighted completion time. *Mathematics of Operations Research*, 29(3):686–697, 2004.

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

[4] N. Bansal and K. Dhamdhere. Minimizing weighted flow time. *ACM Transactions on Algorithms (TALG)*, 3(4):39–es, 2007.

[5] P. Baptiste. Scheduling equal-length jobs on identical parallel machines. *Discrete Applied Mathematics*, 103(1-3):21–32, 2000.

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

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

[8] T. Bosman and N. Olver. Improved approximation algorithms for inventory problems. In *International Conference on Integer Programming and Combinatorial Optimization*, pages 91–103. Springer, 2020.

[9] N. Buchbinder, T. Kimbrel, R. Levi, K. Makarychev, and M. Sviridenko. Online make-to-order joint replenishment model: Primal-dual competitive algorithms. *Operations Research*, 61(4):1014–1029, 2013.

[10] C. Chekuri, S. Khanna, and A. Zhu. Algorithms for minimizing weighted flow time. In *Proceedings of the thirty-third annual ACM symposium on Theory of computing*, pages 84–93, 2001.

[11] C. Chekuri, R. Motwani, B. Natarajan, and C. Stein. Approximation techniques for average completion time scheduling. *SIAM Journal on Computing*, 31(1):146–166, 2001.

[12] M. Cheung, A. N. Elmachtoub, R. Levi, and D. B. Shmoys. The submodular joint replenishment problem. *Mathematical Programming*, 158(1-2):207–233, 2016.

[13] L. Epstein and R. Van Stee. Lower bounds for on-line single-machine scheduling. In *International Symposium on Mathematical Foundations of Computer Science*, pages 338–350. Springer, 2001.

[14] M. X. Goemans, M. Queyranne, A. S. Schulz, M. Skutella, and Y. Wang. Single machine scheduling with release dates. *SIAM Journal on Discrete Mathematics*, 15(2):165–192, 2002.

[15] R. L. Graham, E. L. Lawler, J. K. Lenstra, and A. Rinnooy Kan. Optimization and approximation in deterministic sequencing and scheduling: a survey. *Annals of Discrete Mathematics*, 5:287–326, 1979.

[16] J. A. Hoogeveen and A. P. Vestjens. Optimal online algorithms for single-machine scheduling. In *International Conference on Integer Programming and Combinatorial Optimization*, pages 404–414. Springer, 1996.

[17] H. Kellerer, T. Tautenhahn, and G. Woeginger. Approximability and nonapproximability results for minimizing total flow time on a single machine. *SIAM Journal on Computing*, 28(4):1155–1166, 1999.

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

[19] J. K. Lenstra, A. R. Kan, and P. Brucker. Complexity of machine scheduling problems. In *Annals of discrete mathematics*, volume 1, pages 343–362. Elsevier, 1977.

[20] R. Levi, R. Roundy, D. Shmoys, and M. Sviridenko. A constant approximation algorithm for the one-warehouse multiretailer problem. *Management Science*, 54(4):763–776, 2008.

[21] R. Levi, R. O. Roundy, and D. B. Shmoys. Primal-dual algorithms for deterministic inventory problems. *Mathematics of operations research*, 31(2):267–284, 2006.

[22] T. Nonner and A. Souza. A 5/3-approximation algorithm for joint replenishment with deadlines. In *International Conference on Combinatorial Optimization and Applications*, pages 24–35. Springer, 2009.

[23] M. K. Starr and D. W. Miller. *Inventory control: theory and practice*. Prentice-Hall, 1962.