Joint replenishment meets scheduling

Péter Györgyi[1] · Tamás Kis[1] · Tímea Tamási[1,2] · József Békési[3]

Abstract

In this paper, we consider a combination of the joint replenishment problem (JRP) and single-machine scheduling with release dates. There is a single machine and one or more item types. Each job has a release date, a positive processing time, and it requires a subset of items. A job can be started at time t only if all the required item types were replenished between the release date of the job and time point t. The ordering of item types for distinct jobs can be combined. The objective is to minimize the total ordering cost plus a scheduling criterion, such as total weighted completion time or maximum flow time, where the cost of ordering a subset of items simultaneously is the sum of a joint ordering cost, and an additional item ordering cost for each item type in the subset. We provide several complexity results for the offline problem, and competitive analysis for online variants with min–sum and min–max criteria, respectively.

Keywords Joint replenishment · Single-machine scheduling · Complexity · Polynomial time algorithms · Online algorithms

1 Introduction

The joint replenishment problem (JRP) is a classical problem of supply chain management with several practical applications. In this problem, a number of demands emerge over the time horizon, where each demand has an arrival time, and an item type (commodity). To fulfill a demand, the required item must be ordered *not sooner* than the arrival time of the demand. Orders of different demands can be combined, and the cost of simultaneously ordering a subset of item types incurs a joint ordering cost and an additional item ordering cost for each item type in the order. None of these costs depends on the number of units ordered. Thus, the total ordering cost can be reduced by combining the ordering of item types of distinct demands. However, delaying a replenishment of an item type delays the fulfillment of all the demands that require it. The objective function expresses a trade-off between the total ordering cost, and the cost incurred by delaying the fulfillment of some demands, see, e.g., Buchbinder et al. (2013). There are other variants of this basic problem, see Sect. 2.

In the above variant of JRP, a demand becomes ready as soon as the required item type has been replenished after its arrival time. However, in a make-to-order manufacturing environment the demands may need some processing by one or more machines before they become ready, or in other words, a scheduling problem emerges among the demands for which the required item types have been ordered. This paper initiates the systematic study of these variants. By adopting the common terminology of scheduling theory, from now on we call the demands and item types 'jobs' and 'resource types,' respectively.

✉ Tamás Kis
   kis.tamas@sztaki.hu

   Péter Györgyi
   gyorgyi.peter@sztaki.hu

   Tímea Tamási
   timea.tamasi@sztaki.hu

   József Békési
   bekesi@inf.u-szeged.hu

[1] Institute for Computer Science and Control, Eötvös Loránd Research Network, Kende Str. 13-17., Budapest 1111, Hungary

[2] Department of Operations Research, Institute of Mathematics, ELTE Eötvös Loránd University, Budapest, Hungary

[3] Department of Computer Algorithms and Artificial Intelligence, Faculty of Science and Informatics, University of Szeged, Szeged, Hungary

We consider the simplest scheduling environment, the single machine. It means that there is one machine, which can process at most one job at a time. Each job has a release date and a processing time, and it requires one or more resource types (this is a slight generalization of JRP, where each demand has only one item type). A job becomes ready to be started only if all the required resources are replenished after its release date. We only consider non-preemptive scheduling problems in this paper. The objective function is the sum of the replenishment costs and the scheduling cost (such as the total weighted completion time, the total weighted flow time, or the maximum flow time of the jobs). Observe that if we delay a replenishment of a resource type i, then all the jobs waiting for i will occupy the machine after i arrives, thus the processing of these jobs may delay the processing of some other jobs arriving later. In other words, at each time moment we have to take care of the jobs with release dates in the future, which is not easy in the offline case, and almost impossible in an online problem, where the jobs are not known in advance.

After a literature review in Sect. 2, we provide a formal problem statement along with a brief overview of the main results of the paper in Sect. 3. Some hardness results for min–sum, and min–max type criteria are presented in Sects. 4, and 5, respectively. Polynomial time algorithms are described for some variants of the problem in Sects. 6 and 7. Then, in Sects. 8 and 9 we provide online algorithms for min–sum type criteria, and for the F_{max} objective function, respectively. We conclude the paper in Sect. 10.

2 Literature review

The first results on JRP are more than 50 years old, see, e.g., Starr and Miller (1962), for an overview of older results we refer the reader to Khouja and Goyal (2008). Since then, several theoretical and practical results appeared, this review cites only the most relevant literature.

Over the years, a number of different variants of the JRP have been proposed and studied, some of which are mathematically equivalent. Originally, JRP was an inventory management problem, where demands have due dates, and to fulfill a demand, the required item type must be ordered *before* the due-date of the demand. However, keeping the units on stock incurs an inventory holding cost beyond the total ordering costs, and we call this variant JRP-INV. In another variant, demands have release dates and deadlines, and they can be fulfilled by ordering the corresponding items in the specified time intervals. The cost of a solution is determined solely by the total ordering costs. We will refer to this variant as JRP-D. In the last variant considered in this review, demands have a delay cost function instead of deadlines, which determines the cost incurred by the delay between the release date of the demand and the time point when it is fulfilled by an order. The objective function balances the total cost incurred by delaying the fulfillment of the demands and the total ordering costs. We call this variant JRP-W. In this paper, we combine JRP-W with a scheduling problem.

The complexity of JRP-INV is studied by Arkin et al. (1989), who proved strong NP-hardness. Levi et al. (2006) give a 2-approximation algorithm, and the approximation ratio is reduced to 1.8 by Levi and Sviridenko (2006), see also Levi et al. (2008). Cheung et al. (2016) describe approximation algorithms for several variants under the assumption that the ordering cost function is a monotonically increasing, submodular function over the item types, see also Bosman and Olver (2020).

The JRP-D is shown NP-hard in the strong sense by Becchetti et al. (2009) as claimed by Bienkowski et al. (2015), and APX-hardness is proved by Nonner and Souza (2009), who also describe an 5/3-approximation algorithm. Bienkowski et al. (2015) provide an 1.574 approximation algorithm, and new lower bounds for the best possible approximation ratio. For the special case when the demand periods are of equal length, they give an 1.5-approximation algorithm, and prove a lower bound of 1.2. In Bienkowski et al. (2014), the online version of JRP-D is studied, and an optimal 2-competitive algorithm is described.

The NP-hardness of JRP-W with linear delay cost functions follows from that of JRP-INV (reverse the time line). This has been sharpened by Nonner and Souza (2009) by showing that the problem is still NP-hard even if each item admits only three distinct demands over the time horizon. Buchbinder et al. (2013) study the online variant of the problem providing a 3-competitive algorithm along with a lower bound of 2.64 for the best possible competitive ratio. Bienkowski et al. (2014) provide a 1.791-approximation algorithm for the offline problem, and they also prove a lower bound of 2.754 for the best possible competitive ratio of an algorithm for the online variant of the problem with linear delay cost function.

For scheduling problems, we use the well-known $\alpha|\beta|\gamma$ notation of Graham et al. (1979). $1|r_j|\sum C_j$ is a known strongly NP-hard problem, see, e.g., Lenstra et al. (1977). There is a polynomial time approximation scheme (PTAS) for this problem even in the case of general job weights and parallel machines (Afrati et al., 1999). However, there are other important approximation results for this problem. Chekuri et al. (2001) provide an $e/(e-1) \approx 1.5819$-approximation algorithm based on the α-point method. For the weighted version of the same problem, Goemans et al. (2002) present an 1.7451-approximation algorithm. If 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 (Baptiste, 2000).

Anderson and Potts (2004) devise a 2-competitive algorithm for the online version of $1|r_j| \sum w_j C_j$, i.e., each job j becomes known only at its due-date r_j, and scheduling decisions cannot be reversed. This is the best possible algorithm for this problem, since Hoogeveen and Vestjens (1996) proved that no online algorithm can have a competitive ratio less than 2, even if the weights of the jobs are identical.

Kellerer et al. (1999) describe 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 with unit weights is $\Theta(n)$, and it is unbounded if the jobs have distinct weights (Epstein & Van Stee, 2001). Due to these results, most of the research papers assume preemptive jobs, see, e.g., Bansal and Dhamdhere (2007), Chekuri et al. (2001), and Epstein and Van Stee (2001).

In one of the online variants of the problem, we study in this paper the only unknown parameter is the number of consecutive time periods while the jobs arrive. The ski-rental problem, introduced by L. Rudolph according to Karp (1992), is very much alike. A person would ski for an unknown number of d consecutive days, and she can either rent the equipment for unit cost every day, or buy it in some day for the remaining time for cost y. If she buys the equipment in day $t \le d$, the total cost is $t + y$, while the offline optimum is $\min\{d, y\}$. One seeks a strategy to minimize the ratio of these two values in the worst case without knowing d. The best deterministic online algorithm has a competitive ratio of 2, while the best randomized online algorithm has $e/(e - 1) \approx 1.58$, and both bounds are tight (Karlin et al., 1994, 1988).

While this paper is probably the first one to study the joint replenishment problem combined with production scheduling, there is a considerable literature on integrated production and outbound distribution scheduling. In such models, the production and the delivery of customer orders are scheduled simultaneously, while minimizing the total production and distribution costs. After the first paper by Potts (1980), there appeared several models and approaches over the years, for an overview see, e.g., Chen (2010). While most of the papers focus on the offline variants of the problem, there are a few results on the online version as well. In particular, Averbakh and Xue (2007); Averbakh and Baysan (2013) propose online algorithms for single-machine problems with linear competitive ratio in either Δ or ρ, where Δ and ρ denote the ratio of the maximum to minimum delivery costs, and job processing times, respectively. This is considerably improved by Azar et al. (2016), who propose poly-logarithmic competitive online algorithms in Δ and ρ in single as well as in parallel machine environments, and they also prove that their bounds are essentially best possible.

3 Problem statement and overview of the results

We have a set $\mathcal{J}$ of n jobs that have to be scheduled on a single machine. Each job j has a processing time $p_j > 0$, a release date $r_j \ge 0$, and possibly a weight $w_j > 0$ (in case of min–sum type objective functions). 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 $\mathcal{R}$. Let $\mathcal{J}_i \subseteq \mathcal{J}$ be the set of those jobs that require resource R_i. A job j can only be started if all the resources in $R(j)$ are replenished after r_j. Each time some resource R_i is replenished, a fixed cost K_i is incurred on top of a fixed cost K_0, which must be paid each time moment when any replenishment occurs. These costs are independent of the amount replenished. Replenishment is instantaneous.[1]

A *solution of the problem* is a pair $(S, \mathcal{Q})$, where S is a *schedule* specifying a starting time for each job $j \in \mathcal{J}$, and $\mathcal{Q} = \{(\mathcal{R}_1, t_1), \ldots, (\mathcal{R}_q, t_q)\}$ is a *replenishment structure*, which specifies time moments t_ℓ along with subsets of resources $\mathcal{R}_\ell \subseteq \mathcal{R}$ such that $t_1 < \cdots < t_q$. We say that job j is *ready to be started at time moment t in replenishment structure* $\mathcal{Q}$, if each resource $R \in R(j)$ is replenished at some time moment in $[r_j, t]$, i.e., $R(j) \subseteq \bigcup_{t_\ell \in [r_j, t]} \mathcal{R}_\ell$. The solution is *feasible* if (i) the jobs do not overlap in time, i.e., $S_j + p_j \le S_k$ or $S_k + p_k \le S_j$ for each $j \ne k$, (ii) each job $j \in \mathcal{J}$ is ready to be started at S_j in $\mathcal{Q}$.

The *cost of a solution* is the sum of the scheduling cost c_S, and the replenishment cost $c_\mathcal{Q}$. The former can be any optimization criteria know in scheduling theory, but in this paper, we confine our discussion to the weighted sum of job completion times $\sum w_j C_j$, the sum of flow times $\sum F_j$, where $F_j = C_j - r_j$, and to the maximum flow time $F_{\max} := \max F_j$. Note that C_j is the completion time of job j in a schedule. The replenishment cost is calculated as follows: $c_\mathcal{Q} := \sum_{\ell=1}^{|\mathcal{Q}|}(K_0 + \sum_{R_i \in \mathcal{R}_\ell} K_i)$.

In the offline problem, the complete input is known in advance, and we seek a feasible solution of minimum cost over the set of all feasible solutions. In the online variant of the problem, where the jobs arrive over time, and the input becomes known only gradually, the solution is built step by step; however, decisions cannot be reversed, i.e., if a job is started, or some replenishment is made, then this cannot be altered later.

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 problems.

---

[1] This assumption is without loss of generality in offline problems, and can be handled by transforming the problem data in the online variants, see Sect. 8.1.

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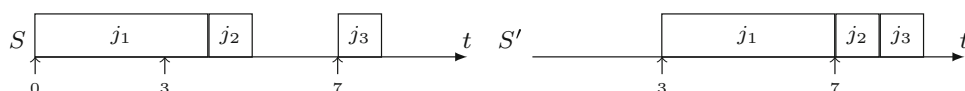

Table 1 Results of the paper

Problem	Result	Source
$1\|jrp, s = 1, r_j\|\sum C_j + c_{\mathcal{Q}}$	NP-hard	Proposition 1
$1\|jrp, p_j = 1, r_j\|\sum C_j + c_{\mathcal{Q}}$	NP-hard	Theorem 3
$1\|jrp, s = 2, r_j\|F_{\max} + c_{\mathcal{Q}}$	NP-hard	Theorem 4
$1\|jrp, s = \text{const}, p_j = 1, r_j\|\sum w_j C_j + c_{\mathcal{Q}}$	Polynomial alg.	Theorem 5
$1\|jrp, s = \text{const}, p_j = p, r_j\|\sum C_j + c_{\mathcal{Q}}$	Polynomial alg.	Theorem 6
$1\|jrp, s = 1, r_j\|F_{\max} + c_{\mathcal{Q}}$	Polynomial alg.	Theorem 7
$1\|jrp, s = \text{const}, p_j = p, r_j\|F_{\max} + c_{\mathcal{Q}}$	Polynomial alg.	Theorem 9
$1\|jrp, s = 1, p_j = 1, r_j\|\sum C_j + c_{\mathcal{Q}}$	2-Competitive alg.	Theorem 10
$1\|jrp, s = 1, p_j = 1, r_j\|\sum C_j + c_{\mathcal{Q}}$	No $\left(\frac{3}{2} - \varepsilon\right)$-competitive alg.	Theorem 11
$1\|jrp, s = 1, p_j = 1, r_j\|\sum w_j C_j + c_{\mathcal{Q}}$	No $\left(\frac{\sqrt{5}+1}{2} - \varepsilon\right)$-competitive alg.	Theorem 12
$1\|jrp, s = 1, p_j = 1, r_j\|\sum F_j + c_{\mathcal{Q}}$	2-Competitive alg.	Theorem 13
$1\|jrp, s = 1, p_j = 1, r_j\|\sum F_j + c_{\mathcal{Q}}$	No $\left(\frac{3}{2} - \varepsilon\right)$-competitive alg.	Theorem 14
$1\|jrp, s = 1, p_j = 1, \text{regular } r_j\|F_{\max} + c_{\mathcal{Q}}$	$\sqrt{2}$-Competitive alg.	Theorem 15
$1\|jrp, s = 1, p_j = 1, \text{regular } r_j\|F_{\max} + c_{\mathcal{Q}}$	No $\left(\frac{4}{3} - \varepsilon\right)$-competitive alg.	Theorem 16
$1\|jrp, s = 1, p_j = 1, r_j\|F_{\max} + c_{\mathcal{Q}}$	No $\left(\frac{\sqrt{5}+1}{2} - \varepsilon\right)$-competitive alg.	Theorem 17

We extend the notation of Graham et al. (1979) by jrp in the β field to indicate the joint replenishment of the resources, thus we denote our problem by $1|jrp, r_j|c_S + c_{\mathcal{Q}}$. In addition, $s = 1$ or $s = \text{const}$ indicate that the number of resources is 1, and constant, respectively, and not part of the input. Further on, $p_j = 1$ and $p_j = p$ imply that all the jobs have the same processing time 1 (unit-time jobs), and p, respectively.

The next example describes two feasible solutions for a one-resource problem.

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 from a single resource R, i.e., $\mathcal{Q} = ((\{R\}, 0), (\{R\}, 3), (\{R\}, 7))$, and the starting times of the jobs are $S_1 = 0$, $S_2 = 4$, and $S_3 = 7$, then $(S, \mathcal{Q})$ is a feasible solution.

However, if there are only two replenishments in $\mathcal{Q}'$ at $t_1 = 3$ and $t_2 = 7$, then we have to start the jobs later, e.g., if $S'_1 = 3$, $S'_2 = 7$, and $S'_3 = 8$, then $(S', \mathcal{Q}')$ is feasible. Observe that in the second solution we have saved the cost of a replenishment ($K_0 + K_1$); however, the total completion time of the jobs has increased from 17 to 24. See Fig. 1 for an illustration.

Results of the paper. The main results of the paper fall in 3 categories: (i) NP-hardness proofs, (ii) polynomial time algorithms, and (iii) competitive analysis of online variants of the problem, see Table 1 for an overview. We provide an almost complete complexity classification for the offline problems

with both of the $\sum w_j C_j$ and $F_{\max}$ objectives. Notice that the former results imply analogous ones for the $\sum w_j F_j$ criterion. While most of our polynomial time algorithms work only with unit-time jobs, a notable exception is the case with a single resource and the $F_{\max}$ objective, where the job processing times are arbitrary positive integer numbers. We have devised online algorithms for some special cases of the problem for both min–sum and min–max criteria. In all variants for which we present an online algorithm with constant competitive ratio, we have to assume unit-time jobs. While we have a 2-competitive algorithm with unit-time jobs for min–sum criteria, for the online problem with the $F_{\max}$ objective we also have to assume that the input is regular, i.e., in every time unit a new job arrives, but in this case the competitive ratio is $\sqrt{2}$.

Preliminaries. Let $\mathcal{T} := \{\tau_1, \ldots, \tau_{|\mathcal{T}|}\}$ be the set of different job release dates r_j such that $\tau_1 < \cdots < \tau_{|\mathcal{T}|}$. For technical reasons, we introduce $\tau_{|\mathcal{T}|+1} := \tau_{|\mathcal{T}|} + \sum_{j \in \mathcal{J}} p_j$.

Observation 1 *For any scheduling criterion, the problem $1|jrp, r_j|c_S + c_{\mathcal{Q}}$ admits an optimal solution in which all replenishments occur at the release dates of some jobs.*

4 Hardness results for min–sum criteria

In this section, we prove results for $1|jrp, r_j|\sum w_j C_j + c_{\mathcal{Q}}$. However, these results remain valid, if we replace $\sum w_j C_j$

to $\sum w_j F_j$, since the difference between the two objective functions is a constant ($\sum_{j \in \mathcal{J}} w_j r_j$).

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).

Recall the definition of $\mathcal{R}$ and $\mathcal{T}$ in Sect. 3.

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

Proof If $K_0 = K_1 = \cdots = K_s = 0$ 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 no-cost at any time moment).

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:

$$W = \{\mathcal{Q} : \mathcal{Q} = ((\mathcal{R}_1, \tau_1), \dots, (\mathcal{R}_{|\mathcal{T}|}, \tau_{|\mathcal{T}|})), \text{ where } \mathcal{R}_\ell \subseteq \mathcal{R}\}.$$

By Observation 1, there exists 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 $r'_j := \min\{t \geq r_j : R(j) \subseteq \cup_{\tau_\ell \in [r_j, t]} \mathcal{R}_\ell\}$ 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 $(\mathcal{Q}^\star, S^\star)$ is an optimal solution of I then $S^\star$ is an optimal solution for $I_{\mathcal{Q}^\star}$ and the optimal solution of $I_{\mathcal{Q}^\star}$ yields an optimal solution for I with $\mathcal{Q}^\star$. 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 $\mathcal{O}_{\tilde{\mathcal{Q}}} = \min\{\mathcal{O}_{\mathcal{Q}} : \mathcal{Q} \in W\}$. Due to our previous observation, $(\tilde{\mathcal{Q}}, S^{\tilde{\mathcal{Q}}})$ is an optimal solution of I. □

Note that Theorem 1 remains valid even if we add restrictions on the processing times of the jobs, e.g., $p_j = p$ or $p_j = 1$.

We need the following results of Lenstra et al. (1977).

Theorem 2 *The problem $1|r_j| \sum C_j$ is NP-hard.*

The next results is a direct corollary.

Proposition 1 $1|jrp, s = 1, r_j| \sum C_j + c_{\mathcal{Q}}$ *is NP-hard even if $K_0 = K_1 = 0$.*

We have another negative result when $p_j = 1$ and s is not a constant:

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

Proof We reduce the MAX CLIQUE problem to our scheduling problem.

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?

Consider an instance I of MAX CLIQUE, from which we create an instance I' of $1|jrp, p_j = 1, r_j| \sum C_j + c_{\mathcal{Q}}$. We define $|V| + M$ jobs (where M is a sufficiently large number), and $s = |E|$ resources in I'. There is a bijection between the edges in E and the resources, and likewise, there is a bijection between the nodes in V and the first $|V|$ jobs. 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 associated with 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 I, if and only if I' admits a solution of 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 corresponds 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 to the nodes of $V \setminus V'$ and have 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 in 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 $1|jrp, p_j = 1, r_j| \sum C_j + c_{\mathcal{Q}}$ with an objective function value of $\mathcal{O}$. Observe that each resource must 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 $\binom{|V|+M+2}{2} - (|V| - k) = \binom{|V|+M+1}{2} + M + 1 + k$, which is greater than $\mathcal{O}$ for M sufficiently large. Hence, there is no gap in the schedule before r''. 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. $\quad\square$

5 Hardness of $1|jrp, r_j|F_{\max} + c_{\mathcal{Q}}$

When the scheduling objective is $F_{\max}$, the problem with two resources is NP-hard, even in a very special case:

Theorem 4 *If $s = 2$, then the problem $1|jrp, r_j|F_{\max} + c_{\mathcal{Q}}$ is NP-hard even if $K_0 = K_1 = 0$ and every job requires only one resource.*

Proof We will reduce the NP-hard PARTITION problem to our scheduling problem.

PARTITION: Given a set A of n items with positive integer sizes $a_1, a_2, \ldots, a_n$ such that $B = \sum_{j \in A} a_j / 2$ is integral. Is there a subset A_1 of A such that $\sum_{j \in A_1} a_j = B$?

Consider an instance I of PARTITION, and we define the corresponding instance I' of $1|r_j, jrp|F_{\max} + c_{\mathcal{Q}}$ as follows. There are $n + 2$ jobs: n *partition jobs*, such that $p_j = a_j$, $r_j = B$, and $R(j) = R_1$ $(j = 1, \ldots, n)$. The other two jobs require resource R_2, they have unit processing times, $r_{n+1} := 0$, and $r_{n+2} := 2B$. Further on, let $K_0 = K_1 := 0$, and $K_2 := B^2$.

We prove that PARTITION instance I has a solution if and only if I' admits a solution with an objective function value of at most $\mathcal{O} := B^2 + 2B + 1$.

If I has a solution, then consider the following solution $(S, \mathcal{Q})$ of I'. There are two replenishments: $\mathcal{Q} = (((\{R_1\}, B), (\{R_2\}, 2B))$ thus $c_{\mathcal{Q}} = B^2$. Let the partition jobs correspond to the items in A_1 be scheduled in S in arbitrary order in $[B, 2B]$, while the remaining partition jobs in $[2B + 1, 3B + 1]$. Let $S_{n+1} := 2B$, and $S_{n+2} := 3B + 1$ (see Fig. 2 for illustration); therefore, $F_j \leq 2B + 1$, $j = 1, \ldots, n + 2$. Observe that $(S, \mathcal{Q})$ is feasible, and its objective function value is exactly $\mathcal{O}$.

Now suppose that there is a solution $(S, \mathcal{Q})$ of $1|jrp, r_j|F_{\max} + c_{\mathcal{Q}}$ with $v(S, \mathcal{Q}) \leq \mathcal{O}$. Observe that there is only one replenishment from R_2, since two replenishments would have higher cost than $\mathcal{O}$. It means the flow time of every job

is at most $2B + 1$. Since j_{n+2} requires R_2, the replenishment time of R_2 cannot be earlier than $r_{n+2} = 2B$. However, j_{n+1} has to be completed until $2B + 1$, which means it requires R_2 not later than $2B$; therefore, the replenishment of R_2 is at $2B$, and $S_{n+1} = 2B$. Since the partition jobs has a flow time of at most $2B + 1$, thus these jobs are scheduled in the interval $[B, 3B + 1]$. Due to the position of j_{n+1}, the machine cannot be idle in this interval, which means the partition jobs are scheduled in $[B, 2B]$, and in $[2B + 1, 3B + 1]$ without idle times (j_{n+1} separates the two intervals). Hence, we get a solution for the PARTITION instance I. $\quad\square$

6 Polynomial time algorithms for min–sum criteria

In this section, we describe polynomial time algorithms, based on dynamic programming, for solving special cases of $1|jrp, r_j| \sum w_j C_j + c_Q$. Again, the same methods work for the $\sum w_j F_j + c_Q$ objective function. 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. Recall the definition of $\mathcal{T}$ in the end of Sect. 3.

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

Proof We describe a dynamic program for computing an optimal solution of the problem. The states of the dynamic program are arranged in $|\mathcal{T}| + 1$ layers, $L(\tau_1), \ldots, L(\tau_{|\mathcal{T}|+1})$, where $L(\tau_k)$ is the kth *layer* $(k = 1, \ldots, |\mathcal{T}| + 1)$. A state $N \in L(\tau_k)$ is a tuple encoding some properties of a partial solution of the problem. Each state has a corresponding partial schedule S^N such that the last job scheduled completes not later than τ_k, and also a replenishment structure $\mathcal{Q}^N$ such that replenishments occur only at $\{\tau_1, \ldots, \tau_{k-1}\}$.

More formally, a state N from $L(\tau_k)$ is a tuple $[\tau_k; \alpha_1, \ldots, \alpha_s; \beta_1, \ldots, \beta_s; \gamma_1, \ldots, \gamma_s; \delta]$, with the following definitions of the variables:

α_i—number of the jobs from $\mathcal{J}_i$ (jobs that require resource R_i) that are scheduled in S^N,

β_i—time point of the last replenishment from R_i in $\mathcal{Q}^N$ (i.e., β_i is the largest time point τ_ℓ such that $R_i \in \mathcal{R}_\ell$, and $(\mathcal{R}_\ell, \tau_\ell) \in \mathcal{Q}^N$, or '−' if there is no such replenishment in $\mathcal{Q}^N$),

γ_i—number of the replenishments from R_i in $\mathcal{Q}^N$ (the number of $(\mathcal{R}_\ell, \tau_\ell)$ pairs in $\mathcal{Q}^N$ such that $R_i \in \mathcal{R}_\ell$),
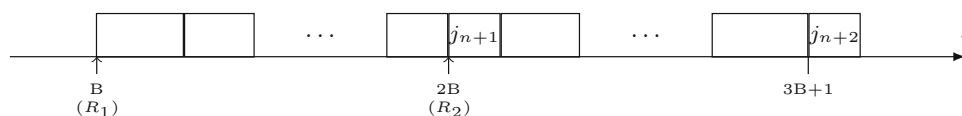

Fig. 2 Illustration of the schedule. The arrows below the axis denote the replenishments (the replenished resources are in a bracket)
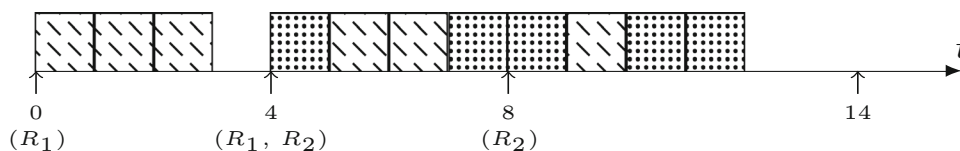
Fig. 3 A corresponding partial schedule and replenishment structure of state [14; 6, 5; 4, 8; 2, 2; 3]. Jobs of $\mathcal{J}_1$ are hatched, while the jobs of $\mathcal{J}_2$ are dotted

δ—total number of replenishments in $\mathcal{Q}^N$, i.e., the number of $(\mathcal{R}_\ell, \tau_\ell)$ pairs in $\mathcal{Q}^N$ such that $\mathcal{R}_\ell \neq \emptyset$.

See Fig. 3 for an easy example.

The schedule S^N associated with N is specified by n numbers that describe the completion time of the jobs, or -1, if the corresponding job is not scheduled in S^N.

Let $\mathcal{J}_i(\bar{\beta}_i)$ be the subset of those jobs from $\mathcal{J}_i$ that have a release date at most $\bar{\beta}_i$.

The first layer of the dynamic program has only one state N_0 where all parameters are 0 or '$-$,' and the corresponding schedule and replenishment structure is empty. The algorithm builds the states of $L(\tau_{k+1})$ from those of $L(\tau_k)$, and each state has a 'parent state' in the previous layer (except N_0 in $L(\tau_1)$). We compute the states by Algorithm Dyn-Prog_wjCj. When processing state $N \in L(\tau_k)$, the algorithm goes through all the subsets of the resources, and for each subset it generates a tuple $\bar{N}$ by selecting jobs unscheduled in S^N. The number of jobs selected is at most $\tau_{k+1} - \tau_k$, so that they can fit into the interval $[\tau_k, \tau_{k+1}]$. From each set $\mathcal{J}_i$, we consider only those unscheduled jobs that are ready to be started while taking into account the latest replenishment time of resource R_i, see step 1c. Then, S^N is extended with the selected jobs to obtain a new schedule S^{new}, and also a new replenishment structure $\mathcal{Q}^{new}$ is obtained in step 1f. There are two cases to consider:

- If $\bar{N}$ is a new state, i.e., not generated before, then it is unconditionally added to $L(\tau_{k+1})$, see step 1g.
- If $\bar{N}$ is not a new state, then we compare the total weighted completion time of the jobs in the schedule already associated with $\bar{N}$ and that in S^{new}. If S^{new} is the better, then (S^{new}, Q^{new}) replaces $(S^{\bar{N}}, Q^{\bar{N}})$, and the parent node of $\bar{N}$ is updated to N, see step 1h.

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.

Lemma 1 *Let $N = [\tau_k; \alpha_1, \ldots, \alpha_s; \beta_1, \ldots, \beta_s; \gamma_1, \ldots, \gamma_s; \delta] \in L(\tau_k)$ 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.

Algorithm DynProg_wjCj

Initialization: $k := 1$, $N_0 := [\tau_1; 0, \ldots, 0; -, \ldots, -; 0, \ldots, 0; 0]$, S^{N_0} and $\mathcal{Q}^{N_0}$ are empty sets, $L(\tau_1) := \{N_0\}$

1. For each state $N := [\tau_k; \alpha_1, \ldots, \alpha_s; \beta_1, \ldots, \beta_s; \gamma_1, \ldots, \gamma_s; \delta] \in L(\tau_k)$ and for each subset $\bar{\mathcal{R}}$ of $\{R_1, \ldots, R_s\}$ do steps 1a-1h:

    a. 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 $\bar{\gamma}_i := \gamma_i$.
    b. If $\bar{\mathcal{R}} = \emptyset$, then let $\bar{\delta} := \delta$, otherwise, let $\bar{\delta} := \delta + 1$.
    c. Determine the sets of jobs $\mathcal{J}_i(\bar{\beta}_i)$ $(i = 1, \ldots, s)$. Let $\mathcal{J}^k \subseteq \cup_{i=1}^s \mathcal{J}_i(\bar{\beta}_i)$ be the $\min\{\tau_{k+1} - \tau_k, \sum_{i=1}^s (|\mathcal{J}_i(\bar{\beta}_i)| - \alpha_i)\}$ unscheduled jobs with the largest weights.
    d. Let $\bar{\alpha}_i := \alpha_i + |\{j \in \mathcal{J}^k : R(j) = i\}|$ for $i = 1, \ldots, s$. Let $\bar{\tau} := \tau_{k+1}$.
    e. Let $\bar{N} := [\bar{\tau}; \bar{\alpha}_1, \ldots, \bar{\alpha}_s; \bar{\beta}_1, \ldots, \bar{\beta}_s; \bar{\gamma}_1, \ldots, \bar{\gamma}_s; \bar{\delta}]$.
    f. Construct the schedule S^{new} from S^N by scheduling the jobs of $\mathcal{J}^k$ in non-increasing w_j order from τ_k without idle times. Analogously, $\mathcal{Q}^{new}$ is obtained from $\mathcal{Q}^N$ by extending it with $(\bar{\mathcal{R}}, \bar{\tau})$.
    g. If $\bar{N} \notin L(\bar{\tau})$, then add $\bar{N}$ to $L(\bar{\tau})$, set N to be its parent node, and associate $(S^{new}, \mathcal{Q}^{new})$ with $\bar{N}$.
    h. If $\bar{N} \in L(\bar{\tau})$ and the total weighted completion time of S^{new} is smaller than that of $S^{\bar{N}}$, then replace $S^{\bar{N}}$ with S^{new}, $\mathcal{Q}^{\bar{N}}$ with $\mathcal{Q}^{new}$, and set the parent node of $\bar{N}$ to N.

2. If $k \leq |\mathcal{T}|$ then let $k := k + 1$ and go to the first step.
3. Determine a state $\hat{N} \in L(\tau_{|\mathcal{T}|+1})$ with the smallest $\sum_{j \in \mathcal{J}} w_j C_j(S^N) + K_0 \delta + \sum_{i=1}^s K_i \gamma_i$ value and output the corresponding solution.

(iii) If β_i is a number (i.e., $\beta_i \neq -$), then there is a replenishment from R_i at β_i in $\mathcal{Q}^N$.
(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 1c.
(ii) Follows from (i).
(iii) Follows from the definition of β_i.
(iv) Follows from Algorithm DynProg_wjCj.
(v) Each job is scheduled because $\alpha_i = |\mathcal{J}_i|$ and the jobs do not overlap due to (ii). Observe that Algorithm DynProg_wjCj cannot schedule a job before its release date. If it chooses a job $j \in \mathcal{J}_i$ to process in $[\tau_k, \tau_{k+1}]$ at step

1c, it has a release date at most $\bar{\beta}_i$, since $j \in \mathcal{J}_i(\bar{\beta}_i)$, thus the statement follows from (iii) and the definition of $\bar{\beta}_i$ in step 1a.

(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 1c, when the algorithm creates an arbitrary state $\bar{N}$ from its parent state N.

Claim 1 *If* $|\mathcal{J}^k| < \tau_{k+1} - \tau_k$, *then* $\mathcal{J}^k$ *is the set of jobs ready to be started at* τ_k *with respect to* $\mathcal{Q}^{new}$ *that are unscheduled in* S^N. *Otherwise, it is the* $\tau_{k+1} - \tau_k$ *elements of this set with the largest weights.*

Proof Follows from the definitions. □

After determining the states, Algorithm DynProg_wjCj chooses a state $\hat{N} \in L(\tau_{|\mathcal{T}|+1})$ representing a feasible solution of smallest objective function value, and later we prove that $(\mathcal{Q}^{\hat{N}}, S^{\hat{N}})$ is an optimal solution. For each state $N \in L(\tau_{|\mathcal{T}|+1})$, 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. 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 $(\mathcal{Q}^\star, S^\star)$ be an optimal solution. Notice that the jobs scheduled in the interval $[\tau_k, \tau_{k+1}]$ in $S^\star$ are in non-increasing w_j order, and replenishments occur only at the release dates of the jobs. Recall that N_0 is the only state of $L(\tau_1)$. Let $N_k \in L(\tau_{k+1})$ be the state that we get from $N_{k-1} \in L(\tau_k)$, when $\bar{\mathcal{R}}$ is set to $\mathcal{R}_k^\star$ (the set of those resources replenished at τ_k in $\mathcal{Q}^\star$) in Algorithm DynProg_wjCj ($k = 1, \ldots, |\mathcal{T}|$). 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 DynProg_wjCj 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 $\mathcal{Q}^\star \equiv \mathcal{Q}^{N_{|\mathcal{T}|}}$ (i.e., the $\mathcal{Q}^\star$ and $\mathcal{Q}^{N_{|\mathcal{T}|}}$ consists of the same pairs of resource subsets and time points). It remains to prove that the total weighted completion time is the same for $S^\star$ 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* $S^\star$ *and in* $S^{N_{|\mathcal{T}|}}$.

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

If the machine is idle from t only in $S^\star$, then there are more jobs scheduled until $t + 1$ in $S^{N_{|\mathcal{T}|}}$ than in $S^\star$. 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 $S^\star$. 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 $S^\star$, otherwise, $S^{N_{|\mathcal{T}|}}$ would not be feasible. However, if we modify the starting time of j in $S^\star$ to t (the starting time of other jobs does not change) then we would get a schedule S' such that $(\mathcal{Q}^\star, S')$ is better than $(\mathcal{Q}^\star, S^\star)$, 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 $S^\star$ than in $S^{N_{|\mathcal{T}|}}$. The machine is idle from t in $S^{N_{|\mathcal{T}|}}$, thus when Algorithm DynProg_wjCj creates N_k, it chooses a set $\mathcal{J}^k$ at step 1c such that $|\mathcal{J}^k| < \tau_{k+1} - \tau_k$. Hence, we know from Claim 1 that each job from $\mathcal{J}^k \cap \mathcal{J}_i(\tau_k)$ 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} in any feasible schedule for replenishment structure $\mathcal{Q}^\star$. It is in a contradiction with the definition of $\mathcal{J}_i$. □

From Claim 1, we know that Algorithm DynProg_wjCj always starts a job with highest weight among the unscheduled jobs that are ready to be started at any time moment t, if any, during the construction of $S^{N_{|\mathcal{T}|}}$. We claim that this property is even valid for $S^\star$, and thus for every time moment t, the job started at t in $S^{N_{|\mathcal{T}|}}$ has the same weight as the job which starts at t in $S^\star$; therefore, the total weighted completion time of the two schedules is the same.

Suppose that the above property is not met by $S^\star$, that is, let j_1 be a job of minimum starting time in $S^\star$ such that there is a job j_2 such that $w_{j_2} > w_{j_1}$, $S_{j_2}^\star > S_{j_1}^\star$, and j_2 is ready to be started at $S_{j_1}^\star$ in $\mathcal{Q}^\star$. Let S' be the schedule such that $S_{j_1}' := S_{j_2}^\star$, $S_{j_2}' := S_{j_1}^\star$, and $S_j' := S_j^\star$ for each $j \neq j_1, j_2$. Observe that (i) j_2 is ready to be started at S_{j_2}' in $\mathcal{Q}^\star$ by assumption, (ii) j_1 is ready to be started at S_{j_1}' in $\mathcal{Q}^\star$, because $r_{j_1} \leq S_{j_1}^\star \leq S_{j_1}'$, and (iii) each other job $j \notin \{j_1, j_2\}$ is ready to be started at S_j' in $\mathcal{Q}^\star$, because $S_j' = S_j^\star$. Thus, schedule S' is feasible for $\mathcal{Q}^\star$. The total weighted completion time of S' is smaller than that of $S^\star$, which contradicts that $(\mathcal{Q}^\star, S^\star)$ is optimal. This proves the theorem. □

We can prove Theorem 6 with a slightly modified version of the previous algorithm.

Theorem 6 $1|jrp, s = \text{const}, p_j = p, r_j| \sum C_j + c_\mathcal{Q}$ *is solvable in polynomial time.*

Proof *(Proof (sketch))* We sketch a similar dynamic program to that of Theorem 5. Let $\mathcal{T}' := \{\tau + \lambda \cdot p : \tau \in \mathcal{T}, \lambda = 0, 1, \ldots, n\} = \{\tau_1' < \cdots < \tau_{|\mathcal{T}'|}'\}$. 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 DynProg_wjCj 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 $L(\tau')$. The other coordinates have the same meaning as before.

Let $N = [\tau'; \alpha_1, \ldots, \alpha_s; \beta_1, \ldots, \beta_s; \gamma_1, \ldots, \gamma_s; \delta] \in L(\tau')$ be the state chosen in Step 1 of the Algorithm DynProg_wjCj. Steps 1a and 1b remain the same and we modify step 1c as follows. We define $\mathcal{J}^k$ as the set of all unscheduled job from $\bigcup_{i=1}^{s} \mathcal{J}_i(\bar{\beta}_i)$. In step 1d, we define $\bar{\alpha}_i$ in the same way as in Algorithm DynProg_wjCj. 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. In steps 1f, we schedule the jobs of $\mathcal{J}^k$ in non-increasing release date order. In steps 1h and 3, we use unit weights. The remaining steps are the same.

The proof of soundness of the modified dynamic program is analogous to that of Theorem 5. □

Remark 1 The results of Theorems 5 and 6 remain valid even if the jobs may require more than one resource. To this end, we consider all non-empty subsets of the s resources (there are $2^s - 1$ of them), and let ϕ_ℓ be the ℓth subset in some arbitrary enumeration of them. Then, we define $\mathcal{J}_\ell$ as the set of those jobs that require the subset of resources ϕ_ℓ. Clearly, the $\mathcal{J}_\ell$ constitute a partitioning of the jobs. In the dynamic program, let α_ℓ be the number of jobs scheduled from $\mathcal{J}_\ell$, and this also means that each state has $2^s - 1$ components α_ℓ. In step 1c, $\mathcal{J}^k$ is chosen from $\cup_{\ell=1}^{2^s-1} \mathcal{J}_\ell(\min_{i \in \phi_\ell} \bar{\beta}_i)$ as the $\min\{\tau_{k+1} - \tau_k, \sum_{\ell=1}^{2^s-1}(|\mathcal{J}_\ell(\min_{i \in \phi_\ell} \bar{\beta}_i)| - \alpha_\ell)\}$ unscheduled jobs of largest weight. With these modifications, the dynamic programs solve the problems, where the jobs may require more than one resource.

7 Polynomial time algorithms for $1|jrp, r_j|F_{\max} + c_Q$

In this section, we obtain new complexity results for the maximum flow time objective. We start with a special case with a single resource.

Lemma 2 *If $s = 1$, there is an optimal solution, where jobs are scheduled in non-decreasing r_j order, all replenishments occur at the release dates of some jobs, and the machine is idle only before replenishments.*

Proof Let $(S^\star, \mathcal{Q}^\star)$ be an optimal solution. Suppose there exist two jobs, i and j, such that $r_i < r_j$, and i is scheduled directly after j in $S^\star$. By swapping i and j, F_i will decrease, and F_j will not be bigger than F_i before the swap. Hence, $F_{\max}$ will not increase, and the replenishment structure $\mathcal{Q}^\star$

will be feasible for the updated schedule as well, since all replenishments occur at the release dates of some jobs (see Observation 1).

Let τ_1, τ_2 be two consecutive replenishment dates in the optimal solution, and assume the machine is idle in the time interval $[t_1, t_2] \subset [\tau_1, \tau_2)$ such that it is busy throughout the interval $[\tau_1, t_1]$, and let j_1 be the first job starting after t_2. Shift j_1 to the left until t_1. This yields a feasible schedule, since again, we may assume that all replenishments occur at the release dates of some jobs. Moreover, $F_{\max}$ will not increase. Repeating this transformation, we can transform $S^\star$ into the desired form. □

Theorem 7 *If $s = 1$, then the problem $1|jrp, r_j|F_{\max} + c_Q$ can be solved in polynomial time.*

Proof Note that if two or more jobs have the same release dates, then in every optimal solution they are replenished at the same time, and they can be scheduled consecutively in arbitrary order without any idle time. Therefore, we can assume that the release dates are distinct, i.e., $r_1 < r_2 < \cdots < r_n$.

Denote with $K = K_0 + K_1$ the cost of a replenishment. We define n layers in Algorithm DynProg_Fmax, layer i corresponding to r_i. In layer i, the resource is replenished at r_i, and the jobs arriving before r_i and having no replenishment yet are scheduled in increasing release date order.

Throughout the algorithm, we maintain several states for each layer, where a state N is a 5-tuple, i.e., $N = (\alpha_N, \beta_N, \gamma_N, F_{\max}(N), u_N)$.

α_N—release date of a job when the last replenishment occurs,

β_N—release date of a job from which the remaining jobs are scheduled consecutively without idle times,

γ_N—index of the first job scheduled from β_N,

$F_{\max}(N)$—maximum flow time,

u_N—number of replenishments.

From each layer i, the algorithm derives new states for layers $j = i + 1, \ldots, n$. We define an initial layer 0, consisting of the unique state $N_0 := (0, 0, 0, 0, 0)$.

From Lemma 2, we know that there is an optimal solution, where the jobs are scheduled in increasing release date order. In the proposed algorithm, we consider all of the replenishment structures, and the best schedule for each of them. Hence, the algorithm finds the optimal solution.

Since for each $N \in L_i$, $(\alpha_N, \beta_N, \gamma_N, u_N) \in \{r_1, \ldots, r_i\}^2 \times \{1, \ldots, i\}^2$, there are at most n^4 states in the layer n. Therefore, our algorithm is of polynomial time complexity, and it finds an optimal state $N^\star = (\alpha^\star, \beta^\star, \gamma^\star, F_{\max}^\star, u^\star)$ with objective $F_{\max}^\star + K u^\star$.

By recording the parent state of each state N in the algorithm, we can define a series of states leading to $N^\star$: $\{N_0, N_1^\star, \ldots, N_k^\star\}$, where $N_k^\star = N^\star$. The replenishment

Algorithm DynProg_Fmax

Initialization: $N_0 = (0, 0, 0, 0, 0)$, $L_0 = \{N_0\}$.

1. For $i \in \{0, \ldots, n\}$ and $j \in \{i + 1, \ldots, n\}$, do steps 1a-1g:

   a. For each $N \in L_i$, $N = (\alpha_N, \beta_N, \gamma_N, F_{\max}(N), u_N)$, define a new state for layer j: N'.

   b. Let $P := \beta_N + \sum_{k=\gamma_N}^{i} p_k$ (the completion time of jobs scheduled consecutively).

   c. Start the jobs between $i + 1$ and j in increasing release date order from $\max(P, r_j)$ consecutively, without idle times.

   d. Calculate $F_k = C_k - r_k$ for each newly scheduled job.

   e. $\alpha_{N'} := r_i$.
   $$\beta_{N'} := \begin{cases} r_j & \text{if } P < r_j \\ \beta_N & \text{if } P \geq r_j. \end{cases}$$
   $$\gamma_{N'} := \begin{cases} i + 1 & \text{if } P < r_j \\ \gamma_N & \text{if } P \geq r_j. \end{cases}$$
   $F_{\max}(N') := \max(F_{\max}(N), \max_{k=i+1}^{j} F_k)$.
   $u_{N'} := u_N + 1$.

   f. Let $N' = (\alpha_{N'}, \beta_{N'}, \gamma_{N'}, F_{\max}(N'), u_{N'})$.

   g. Insert N' in L_j if it has no other state with the same $(\alpha, \beta, \gamma, u)$ attributes, or replace the state $N'' \in L_j$ with N' if $(\alpha_{N'}, \beta_{N'}, \gamma_{N'}, u_{N'}) = (\alpha_{N''}, \beta_{N''}, \gamma_{N''}, u_{N''})$, while $F_{\max}(N') < F_{\max}(N'')$.

2. Determine the state $N \in L_n$ with the smallest $F_{\max}(N) + Ku_N$ value and output the corresponding solution.

times are defined by $\alpha_1^\star, \ldots \alpha_k^\star$. The optimal schedule can be built in the following way: for $i = k, \ldots, 1$, jobs with indices at least $\gamma_i^\star$ are scheduled from $\beta_i^\star$. □

Now we propose a more efficient algorithm for the case, where $s = 1$, $p_j = 1$ for every job, and all the job release dates are distinct (denoted by 'distinct r_j').

Lemma 3 *The problem* $1|jrp, s = 1, p_j = 1, \text{distinct } r_j$ $|F_{\max} + c_Q$ *admits an optimal solution, where every job has the same flow time, and every replenishment occurs at the release date of a job.*

Proof Consider an optimal solution $(S^\star, Q^\star)$ of maximum flow time $F_{\max}^\star$, where every replenishment occurs at the release date of a job (see Observation 1). We define the schedule S with job starting times $S_j = r_j + F_{\max}^\star - 1$. Then $S_j \geq S_j^\star$ for each $j \in \mathcal{J}$, and the S_j are distinct by the assumption on the r_j. Consequently, $(S, Q^\star)$ is a feasible solution of the same cost as $(S^\star, Q^\star)$. □

Theorem 8 *There is an $O(n^3)$ time algorithm that finds the optimal solution for* $1|jrp, s = 1, p_j = 1, \text{distinct } r_j$ $|F_{\max} + c_Q$.

Proof Using Lemma 3, if we try every possible flow time for scheduling the jobs, and then find the best replenishment structure for this fixed schedule, we obtain the optimal solution by picking the solution that gives the minimal objective function value. The same lemma implies that the optimum flow time is of the form $r_k - r_j + 1$ for $j, k \in \mathcal{J}$ such that $r_j \leq r_k$. Consider the following algorithm:

Algorithm Distinct_rj_Fmax

1. For each $F \in \{r_k - r_j + 1 \mid \forall j, k \in \mathcal{J} \text{ such that } r_j \leq r_k\}$, do steps 1a-1c:

   a. Construct a feasible schedule, where every job has flow time of F, by starting every job j at $r_j + F - 1$. Let $u_F := 0$.

   b. While there is a job without a resource ordered for it, let j be the last such job. Replenish the resource at r_j, let $u_F := u_F + 1$, and repeat.

   c. The solution for F has objective function value $F + Ku_F$.

2. Output the best solution obtained in the first step.

An example for six jobs with $r_1 = 1, r_2 = 2, r_3 = 3, r_4 = 5, r_5 = 7, r_6 = 8$ and fixed $F = 3$ is shown on Fig. 4.

The running time of the algorithm is $O(n^3)$.

We only have to show that for a fixed F, the replenishment structure defined by the algorithm gives a solution with a minimal number of replenishments. We proceed by induction on the number of jobs. For $n = 1$, the statement clearly holds. Consider an input I with n jobs, and denote the optimal number of replenishments for I for the fixed flow time F with $u_F(I)$.

For the last job in the schedule, there must be a replenishment at r_n. Delete the jobs which start at or after r_n in the schedule, and let I' be the input obtained by deleting the same jobs from I. By the induction hypothesis, the algorithm determines the optimal number of replenishments for I'. Hence, $u_F(I) \leq u_F(I') + 1$. On the other hand, by the construction of I', for all $j \in I'$, $r_j + F \leq r_n$. Hence, $u_F(I) \geq u_F(I') + 1$, and thus $u_F(I) = u_F(I') + 1$. □

Finally, we mention that it is easy to modify the dynamic program of Sect. 6 for the $F_{\max}$ objective, so that we have:

Theorem 9 $1|jrp, s = \text{const}, p_j = p, r_j|F_{\max} + c_Q$ *is solvable in polynomial time.*

Proof (Proof (sketch))

Use the same dynamic program as in the proof of Theorem 6, except that in steps 1h and 3, consider the maximum flow time instead of the total completion time.
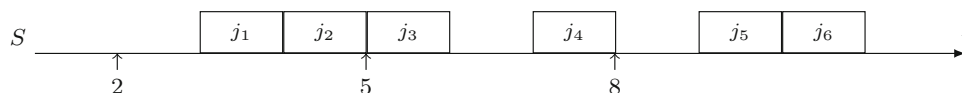

Fig. 4 Schedule and replenishment structure created by Algorithm Distinct_rj_Fmax for six jobs for $F = 3$

The proof of soundness of this algorithm is analogous to those of Theorems 5 and 6. □

8 Competitive analysis of the online problem with min–sum criteria

First, we provide a 2-competitive algorithm for the case where the scheduling cost is the total completion time ($\sum C_j$), then we improve this algorithm to achieve the same result for the case where the scheduling cost is the total flow time ($\sum F_j$). The proof of the second result requires more sophisticated analysis, but the main idea is the same as that of the proof of the first result.

8.1 Online algorithm for $1|jrp, s = 1, p_j = 1, r_j| \sum C_j + c_Q$

In the online version of the problem, we do not know the number of the jobs or anything about them before their release dates.

Since there is only one resource and the processing time and the weight of each job is one, 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 have to decide whether to replenish the resource or not, and finally we can start a job from t for which the resource is replenished. Note that if we have to decide about the replenishment at time point t *before* we get information about the newly released jobs, then the problem is the same as the previous one with each job release date increased by one, by assuming that ordering takes one time unit.

In the following algorithm, let $B_t \subseteq \{j \in \mathcal{J} \mid r_j \le t\}$ denote the set of unscheduled jobs at time t. When we say that 'start the jobs of B_t from time t,' then it means that the jobs in B_t are put on the machine from time t on without any delays between them in increasing release date order. Observe that their total completion time will be $t \cdot |B_t| + G(|B_t|)$, where $G(a) := a(a+1)/2$.

Algorithm Online_SumCj

Initialization: $t := 0$

1. Determine the set B_t of unscheduled jobs at time t.
2. If $B_t = \emptyset$ and no more jobs are left in the input, then stop.
3. If $t \cdot |B_t| + G(|B_t|) \ge K$, then replenish the resource, start the jobs of B_t from t, and let $t := t + |B_t|$. Otherwise, $t := t + 1$.
4. Go to step 1.

Let $(S, \mathcal{Q})$ denote the solution created by Algorithm Online_SumCj, while $(S^\star, \mathcal{Q}^\star)$ an arbitrary optimal solution.

Recall the notation $v(sol)$ denoting the objective function value of a solution sol. Let t_i be the time moment of the ith replenishment in $\mathcal{Q}$ and $t_0 := 0$. To simplify our notations, we will use $B_{(i)} := B_{t_i}$ for the set (block) of jobs that start in $[t_i, t_{i+1})$ in S, see Fig. 5.

Clearly, the release date r_j of a job $j \in B_{(i)}$ has to be in $[t_{i-1}+1, t_i]$. For technical reasons, we introduce $b_i := |B_{(i)}|$, $z_i := |\{j \in B_{(i)} : r_j = t_i\}|$, and $y_i := b_i - z_i$. The next observation follows from the condition of step 3 of Algorithm Online_SumCj.

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

We divide $v(S, \mathcal{Q})$ among the blocks $B_{(i)}$ in the following way: for $i = 1, 2, \ldots,$ let

$$\text{ALG}_i := K + \sum_{j \in B_{(i)}} C_j = K + t_i b_i + G(b_i),$$

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

$$\text{OPT}_i := \begin{cases} K + \sum_{j \in B_{(i)}} C_j^\star, & \text{if } \exists (\mathcal{R}_\ell^\star, \tau_\ell) \in \mathcal{Q}^\star \text{ such that} \\ & \quad\quad t_{i-1} < \tau_\ell < t_i. \\ \sum_{j \in B_{(i)}} C_j^\star, & \text{otherwise.} \end{cases}$$

Observe that $v(S^\star, \mathcal{Q}^\star) \ge \sum_{i \ge 1} \text{OPT}_i$. Now we prove that Algorithm Online_SumCj is 2-competitive.

Theorem 10 *Algorithm Online_SumCj is 2-competitive for the online $1|jrp, s = 1, p_j = 1, r_j| \sum C_j + c_Q$ problem.*

Proof Suppose that there is a gap in S before t_i ($i \ge 1$), and the next gap starts in $[t_{i+\ell}, t_{i+\ell+1})$. Then, we have $\sum_{\mu=0}^{\ell} b_{i+\mu}$ jobs that start between the two neighboring gaps, and their completion times are $t_i + \nu$, where $\nu = 1, 2, \ldots, \sum_{\mu=0}^{\ell} b_{i+\mu}$. Hence,

$$\sum_{\mu=0}^{\ell} \text{ALG}_{i+\mu} = (\ell+1)K + t_i \sum_{\mu=0}^{\ell} b_{i+\mu} + G\left(\sum_{\mu=0}^{\ell} b_{i+\mu}\right).$$

Now we verify that $\sum_{\mu=0}^{\ell} \text{ALG}_{i+\mu} \le 2 \cdot \sum_{\mu=0}^{\ell} \text{OPT}_{i+\mu}$, thus the theorem follows from the previous observations on ALG_i and OPT_i. Let $0 \le \ell' \le \ell$ be the smallest index such that there is no replenishment in $[t_{i+\ell'-1} + 1, t_{i+\ell'} - 1]$ in $\mathcal{Q}^\star$. If there is no such index, then let $\ell' := \ell + 1$.

Claim 3 $\text{ALG}_{i+\mu} \le 2 \cdot \text{OPT}_{i+\mu}$ *for all $\mu \in [\ell', \ell]$.*

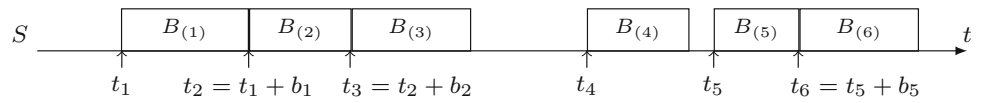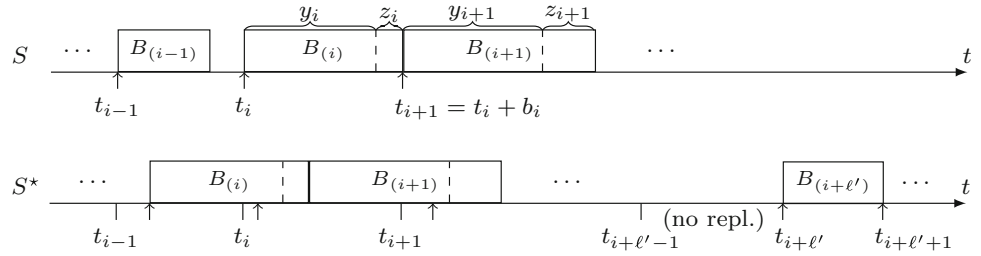**Fig. 5** Schedule S created by Algorithm Online_SumCj

Fig. 6 Schedule S created by Algorithm Online_SumCj and a possible realization of $S^\star$

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 $j \in \cup_{\mu=\ell'}^{\ell} B_{(i+\mu)}$; hence, they cannot be started before the first replenishment after t'. The first replenishment after t' in $\mathcal{Q}^\star$ is not earlier than $t_{i+\ell'}$ in $S^\star$ due to the definition of ℓ'. We have assumed that the order of the jobs is the same in every schedule, and there is no idle time among the jobs in $\cup_{\mu=\ell'}^{\ell} B_{(i+\mu)}$ in S by the choice of i and ℓ, thus we have $C_j^\star \geq C_j$ for each $j \in \cup_{\mu=\ell'}^{\ell} B_{(i+\mu)}$. Since $\mathrm{OPT}_{i+\mu} \geq \sum_{j \in B_{(i+\mu)}} C_j^\star \geq \sum_{j \in B_{(i+\mu)}} C_j \geq K$ for all $\mu \in [\ell', \ell]$, where the last inequality follows from the condition of step 3 of Algorithm Online_SumCj, we have $\mathrm{ALG}_{i+\mu} = K + \sum_{j \in B_{(i+\mu)}} C_j \leq 2 \cdot \mathrm{OPT}_{i+\mu}$ 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 remains to prove $\sum_{\mu=0}^{\ell'-1} \mathrm{ALG}_{i+\mu} \leq 2 \cdot \sum_{\mu=0}^{\ell'-1} \mathrm{OPT}_{i+\mu}$, if there is a replenishment in every interval $[t_{i+\mu-1} + 1, t_{i+\mu} - 1]$, $\mu = 0, \ldots, \ell'-1$ in $\mathcal{Q}^\star$. See Fig. 6 for an illustration of a possible realization of $S^\star$.

Let $\beta := \sum_{\mu=1}^{\ell'-1} b_{i+\mu}$. Observe that

$$\sum_{\mu=0}^{\ell'-1} \mathrm{OPT}_{i+\mu} \geq \ell' K + (t_{i-1} + 1) y_i + G(y_i)$$
$$+ t_i (z_i + \beta) + G(z_i + \beta), \tag{1}$$

since $\mathrm{OPT}_{i+\mu} = K + \sum_{j \in \mathcal{J}_{i+\mu}^S} C_j^\star$ 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 $z_i + \beta$ jobs with a release date at least t_i.

On the other hand, we have

$$\sum_{\mu=0}^{\ell'-1} \mathrm{ALG}_{i+\mu} = \ell' K + t_i (y_i + z_i + \beta) + G(y_i + z_i + \beta), \tag{2}$$

therefore,

$$2 \cdot \sum_{\mu=0}^{\ell'-1} \mathrm{OPT}_{i+\mu} - \sum_{\mu=0}^{\ell'-1} \mathrm{ALG}_{i+\mu}$$
$$\geq \ell' K + 2(t_{i-1} + 1) y_i + t_i (z_i + \beta) - t_i y_i + 2G(y_i)$$
$$+ 2G(z_i + \beta) - G(y_i + z_i + \beta)$$
$$\geq \ell' K + (2t_{i-1} + 1) y_i + t_i (z_i + \beta) - (t_i - 1) y_i$$
$$\geq \ell' K + (2t_{i-1} + 1) y_i + t_i (z_i + \beta) - K \geq 0,$$

where the first inequality is a direct consequence of (1) and (2), the second one follows from the fact that $2G(a) + 2G(b) \geq G(a + b)$ for any a and b. The first inequality of the last line follows from Observation 2, and the second from $\ell' \geq 1$.

Finally, we show that the above analysis is tight. Suppose that there is only one job with a release date 0. Algorithm Online_SumCj starts this job from $t = K - 1$, thus $v(S, \mathcal{Q}) = 2K$. In the optimal solution $(S^\star, \mathcal{Q}^\star)$, this job starts at $t = 0$, thus $v(S^\star, \mathcal{Q}^\star) = K + 1$. As K tends to infinity $2K/(K + 1)$ tends to 2, which shows that Algorithm Online_SumCj is not α-competitive for any $\alpha < 2$. □

We close this section by lower bounds on the best possible competitive ratios:

Theorem 11 *There is no $\left(\frac{3}{2} - \varepsilon\right)$-competitive algorithm for any constant $\varepsilon > 0$ for $1|jrp, s = 1, p_j = 1, r_j| \sum C_j + c_\mathcal{Q}$.*

Proof Suppose that there is only one job arriving at 0. If an algorithm starts 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 starts the first job at t, then it is possible that another job arrives at $t + 1$. In this case starting the two jobs at time $t + 1$ and $t + 2$, respectively, with one replenishment yields a solution of 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 ratio increases, while the second one decreases as t increases. This means that 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 then $\frac{K+\bar{t}+1}{K+1}$ 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 $\frac{(3/2)K - 1/4}{K+1}$. Therefore, for any $\varepsilon > 0$, there is a sufficiently large K, such that there is no $(3/2 - \varepsilon)$-competitive algorithm for the problem. $\square$

When the job weights are arbitrary, then a slightly stronger bound can be derived.

Theorem 12 *There is no* $\left(\frac{\sqrt{5}+1}{2} - \epsilon\right)$-*competitive algorithm for any constant* $\epsilon > 0$ *for* $1|jrp, s = 1, p_j = 1, r_j| \sum w_j C_j + c_{\mathcal{Q}}$.

Proof 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 starting 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:

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

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

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

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

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

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. $\square$

8.2 Online algorithm for $1|jrp, s = 1, p_j = 1, r_j| \sum F_j + c_Q$

In this section, we describe a 2-competitive algorithm for the online version of the total flow time minimization problem for the special case, where there is only one resource, and $p_j = 1$ for all jobs. 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 almost the same as in the previous section and several parts of the proof are analogous. We also use the notations of the previous section, e.g., B_t, $G(a)$, etc. Observe that if we start the jobs of B_t from t, then their total flow time will be $\sum_{j \in B_t}(t - r_j) + G(|B_t|)$.

Algorithm Online_SumFj

Initialization: $t := 0$

1. Determine the set B_t of unscheduled jobs at time t.
2. If $B_t = \emptyset$ and no more jobs are left in the input, then stop.
3. If $\sum_{j \in B_t}(t - r_j) + G(|B_t|) \geq K$, then replenish the resource, start the jobs of B_t from t and let $t := t + |B_t|$. Otherwise, $t := t + 1$.
4. Go to step 1.

Let $(S, \mathcal{Q})$ denote the solution created by Algorithm Online_SumFj, while $(S^\star, \mathcal{Q}^\star)$ an arbitrary optimal solution. The notations t_i, $B_{(i)}$, b_i, y_i, z_i have the same meaning as in the previous section, see again Fig. 5 for illustration. The next observation is analogous to Observation 2:

Observation 3 *If the machine is idle in* $[t_i - 1, t_i]$, *then* $\sum_{j \in B_{t_i-1}}(t_i - 1 - r_j) + G(y_i) < K$.

Let $\text{ALG}_i := K + \sum_{j \in B_{(i)}} F_j$ $(i \geq 1)$, thus we have $v(S, \mathcal{Q}) = \sum_{i \geq 1} \text{ALG}_i$. Let s_i denote the number of replenishments in $\mathcal{Q}^\star$ in $[t_{i-1} + 1, t_i - 1]$, and let $\text{OPT}_i := s_i K + \sum_{j \in B_{(i)}} F_j^\star$. Observe that $v(S^\star, \mathcal{Q}^\star) \geq \sum_{i \geq 1} \text{OPT}_i$. We are ready to prove the main result of this section.
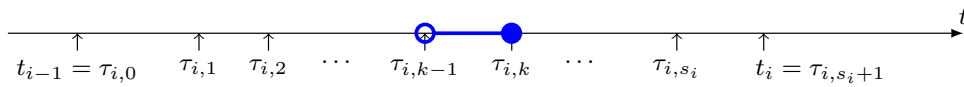
Fig. 7 Replenishment times in $[t_{i-1}+1, t_i-1]$ in $\mathcal{Q}^\star$. The release dates of the jobs of $B_{(i)}(k)$ are from the blue interval

Theorem 13 *Algorithm Online_SumFj is 2-competitive for the online $1|jrp, s=1, p_j=1, r_j|\sum F_j + c_Q$ problem.*

Proof Analogously to the proof of Theorem 10, we prove $\sum_{\mu=0}^{\ell} \text{ALG}_{i+\mu} \leq 2 \cdot \sum_{\mu=0}^{\ell} \text{OPT}_{i+\mu}$ 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 10. Then, the next claim is analogous to Claim 3.

Claim 4 $\text{ALG}_{i+\mu} \leq 2 \cdot \text{OPT}_{i+\mu}$ *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 $j \in \cup_{\mu=\ell'}^{\ell} B_{(i+\mu)}$; hence, they cannot be started before the first replenishment after t'. The first replenishment after t' in $\mathcal{Q}^\star$ is not earlier than $t_{i+\ell}$ in $S^\star$ due to the definition of ℓ'. We have assumed that the order of the jobs is the same in every schedule, and there is no idle time among the jobs in $\cup_{\mu=\ell'}^{\ell} B_{(i+\mu)}$ in S, thus, we have $C_j^\star \geq C_j$ and $F_j^\star \geq F_j$ for each $j \in \cup_{\mu=\ell'}^{\ell} B_{(i+\mu)}$. Since $\text{OPT}_{i+\mu} \geq \sum_{j \in B_{(i+\mu)}} F_j^\star \geq \sum_{j \in B_{(i+\mu)}} F_j \geq K$ for all $\mu \in [\ell', \ell]$, where the last inequality follows from the condition of step 3 of Algorithm Online_SumFj, we have $\text{ALG}_{i+\mu} = K + \sum_{j \in B_{(i+\mu)}} F_j \leq 2 \cdot \text{OPT}_{i+\mu}$ for all $\mu \in [\ell', \ell]$. □

If $\ell' = 0$, then our work is done, thus from now on we suppose $\ell' \geq 1$. It remains to prove $\sum_{\mu=0}^{\ell'-1} \text{ALG}_{i+\mu} \leq 2 \cdot \sum_{\mu=0}^{\ell'-1} \text{OPT}_{i+\mu}$, if there is at least one replenishment in every interval $[t_{i+\mu-1}+1, t_{i+\mu}-1]$, $\mu = 0, \dots, \ell'-1$ in $\mathcal{Q}^\star$, i.e., $s_{i+\mu} \geq 1$ for every $\mu = 0, \dots, \ell'-1$.

First, we prove $\text{ALG}_i \leq 2 \cdot \text{OPT}_i$. Let $\tau_{i,0} := t_{i-1}$, and $\tau_{i,1} < \tau_{i,2} < \cdots < \tau_{i,s_i}$ be the time points of the replenishments in $[t_{i-1}+1, t_i-1]$ in $\mathcal{Q}^\star$. Let $\tau_{i,s_i+1} := t_i$. For $k = 1, 2, \dots, s_i + 1$, let $B_{(i)}(k) := \{j \in B_{(i)} : r_j \in (\tau_{i,k-1}, \min\{t_i-1, \tau_{i,k}\}]\} \subseteq B_{t_i-1}$ and $b_i(k) := |B_{(i)}(k)|$. See Fig. 7 for illustration. Note that these sets are pairwise disjoint and $\bigcup_{k=1}^{s_i+1} B_{(i)}(k) = \{j \in B_{(i)} : r_j < t_i\} = B_{t_i-1}$, and their total size is y_i.

Each job in $B_{(i)}(k)$ gets resource at $\tau_{i,k}$ in $\mathcal{Q}^\star$, if $k \leq s_i$, thus they cannot start earlier than $\tau_{i,k}$ in $S^\star$. The earliest start time of a job from $B_{(i)}(s_i+1)$ is t_i. Since $F_j = (\tau_{i,k} - r_j) + (C_j - \tau_{i,k})$, thus we have $\sum_{j \in B_{(i)}(k)} F_j \geq \sum_{j \in B_{(i)}(k)} (\tau_{i,k} - r_j) + G(b_{(i)}(k))$ for any $1 \leq k \leq s_i + 1$. Applying the previous inequality for each k and $\sum_{j \in B_{(i)} \setminus B_{t_i-1}} F_j \geq G(z_i)$, we get

$$\text{OPT}_i \geq s_i K + \sum_{k=1}^{s_i+1} \sum_{j \in B_{(i)}(k)} (\tau_{i,k} - r_j)$$

$$+ \sum_{k=1}^{s_i+1} G(b_{(i)}(k)) + G(z_i)$$

$$\geq s_i K + \sum_{k=1}^{s_i+1} \sum_{j \in B_{(i)}(k)} (\tau_{i,k} - r_j)$$

$$+ G\left(\frac{y_i}{s_i+1}\right) \cdot (s_i + 1) + G(z_i),$$

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

$$\text{ALG}_i = K + \sum_{j \in B_{(i)}} F_j = K + \sum_{j \in B_{t_i-1}} (t_i - r_j)$$

$$+ G(y_i) + y_i z_i + G(z_i),$$

thus

$$2 \cdot \text{OPT}_i - \text{ALG}_i \geq (2s_i - 1)K + 2 \cdot \sum_{k=1}^{s_i+1} \sum_{j \in B_{(i)}(k)} (\tau_{i,k} - r_j) +$$

$$+ y_i\left(\frac{y_i}{s_i+1} + 1\right) + G(z_i) - \sum_{j \in B_{t_i-1}} (t_i - r_j) - G(y_i) - y_i z_i$$

$$\geq (2s_i - 2)K + 2 \cdot \sum_{k=1}^{s_i+1} \sum_{j \in B_{(i)}(k)} (\tau_{i,k} - r_j) + y_i\left(\frac{y_i}{s_i+1} + 1\right)$$

$$+ G(z_i) - y_i - y_i z_i = (2s_i - 2)K + 2 \cdot \sum_{k=1}^{s_i+1} \sum_{j \in B_{(i)}(k)} (\tau_{i,k} - r_j)$$

$$+ \frac{y_i^2}{s_i+1} + G(z_i) - y_i z_i,$$

where the second inequality follows from Observation 3 and from $|B_{t_i-1}| = y_i$.

On the one hand, if $s_i = 1$, then $2 \cdot \text{OPT}_i - \text{ALG}_i \geq y_i^2/2 + z_i^2/2 - y_i z_i \geq 0$. On the other hand, if $s_i \geq 2$, then we use again Observation 3, and we get $2 \cdot \text{OPT}_i - \text{ALG}_i \geq (2s_i-3)K + G(y_i) + G(z_i) - y_i z_i \geq y_i^2/2 + z_i^2/2 - y_i z_i \geq 0$. Therefore, we have proved $\text{ALG}_i \leq 2 \cdot \text{OPT}_i$.

Now we prove $\text{ALG}_{i+\mu} \leq 2 \cdot \text{OPT}_{i+\mu}$ for any $1 \leq \mu \leq \ell' - 1$. Let $1 \leq \mu \leq \ell' - 1$, and $j' \in B_{(i+\mu)}$ be arbitrary. Suppose that j' has the $n_{j'}$th smallest release date among the jobs in $B_{(i+\mu)}$, i.e., it is the $\left(\sum_{v=1}^{i+\mu-1} b_{(v)} + n_{j'}\right)$th job

in the fixed non-decreasing release date order that determine the order of the jobs in any schedule. Observe that $C_{j'} = t_i + \sum_{v=0}^{\mu-1} b_{(i+v)} + n_{j'}$, because the algorithm starts all of the jobs of $\bigcup_{v=0}^{\mu-1} B_{(i+v)}$ from t_i without any gap, and after that, it starts the jobs from $B_{(i+\mu)}$ also without any gap. However, it is possible that y_i jobs from $B_{(i)}$ are started before t_i in $S^\star$, but every other job from $\bigcup_{v=0}^{\mu} B_{(i+v)}$ has a release date at least t_i. Hence, we have $C_{j'}^\star \geq t_i + z_i + \sum_{v=1}^{\mu-1} b_{(i+v)} + n_{j'}$, since the order of the jobs is fixed. Since $b_{(i)} = y_i + z_i$, we have $C_{j'} \leq C_{j'}^\star + y_i$ and $F_{j'} \leq F_{j'}^\star + y_i$.

Let $h := b_{(i+\mu)}$. Since $\sum_{j \in B_{(i+\mu)}} F_j^\star \geq G(h)$, $\text{OPT}_{i+\mu} \geq K + \sum_{j \in B_{(i+\mu)}} F_j^\star$, and $\text{ALG}_i \leq K + \sum_{j \in B_{(i+\mu)}} (F_j^\star + y_i)$, we have

$$2 \cdot \text{OPT}_{i+\mu} - \text{ALG}_{i+\mu}$$
$$\geq K + G(h) - h y_i \geq G(y_i) + G(h) - h y_i \geq 0,$$

where the second inequality follows from Observation 3.

Finally, observe that the problem instance used at the end of Theorem 10 (there is only one job j, with $r_j = 0$) shows that the above analysis is tight, i.e., Algorithm Online_SumFj is not α-competitive for any $\alpha < 2$. □

The next theorem gives a lower bound on the best possible competitive ratio.

Theorem 14 *There does not exist an online algorithm for $1|jrp, s = 1, p_j = 1, r_j| \sum F_j + c_Q$ with competitive ratio better than $\frac{3}{2} - \epsilon$ for any constant $\epsilon > 0$.*

Proof Suppose that one job is released at 0, and some online algorithm ALG starts it at time $t \geq 0$, then another job at time $t + 1$ is released. The offline optimum for this problem instance with 2 jobs is $OPT = \min(2K + 2, K + t + 2)$, while the online solution has a cost of $2K + t + 2$.

1. If $K < t$, then $OPT = 2K + 2$; therefore, $\frac{ALG}{OPT} \geq \frac{3}{2}$.
2. If $K \geq t + 2$, then $OPT = K + t + 2$, $\frac{ALG}{OPT} = \frac{2K+t+2}{K+t+2} \geq \frac{3}{2}$.
3. If $t \leq K \leq t + 1$, then for any $\varepsilon > 0$ we have $\frac{ALG}{OPT} \geq \frac{3K+2}{2K+2} > \frac{3}{2} - \varepsilon$ if K is sufficiently large.

□

9 Competitive analysis for the online $1|jrp, r_j|F_{\max} + c_Q$ problem

Throughout this section, we assume $p_j = 1$ for each job j, and there is a single resource only ($s = 1$). We will describe a $\sqrt{2}$-competitive algorithm for a semi-online variant, which we call *regular input*, where $r_j = j$ for $j \geq 1$. That is, the

job release dates are known in advance, but we do not know how many jobs will arrive. We also provide lower bounds for the competitive ratio of any algorithm for regular as well as general input.

For the competitive analysis, we need a lower bound for the offline optimum, which is the topic of the next section.

9.1 Lower bounds for the offline optimum

Let $p_{\text{sum}} = \sum_{j=1}^n p_j$.

Lemma 4 $\min_F (K \lceil \frac{p_{\text{sum}}}{F} \rceil + F)$ *is a lower bound for the offline optimum.*

Proof If in a solution the maximum flow time of jobs is F, then at least $\lceil \frac{p_{\text{sum}}}{F} \rceil$ replenishment is needed, which gives the lower bound above. □

Lemma 5 $2\sqrt{K p_{\text{sum}}}$ *is a lower bound for the offline optimum.*

Proof Consider the formula from Lemma 4 without the ceiling function, which is also a lower bound for the optimum: $\min_F (K \frac{p_{\text{sum}}}{F} + F)$. The minimum is obtained in $F = \sqrt{K p_{sum}}$, and it has a value of $2\sqrt{K p_{\text{sum}}}$. □

If $p_j = 1$ for every job, we get $\min_F (K \lceil \frac{n}{F} \rceil + F)$ and $2\sqrt{Kn}$ as lower bounds.

Note that for a regular input, the first lower bound will give exactly the offline optimum. However, for a general input, the above lower bounds can be weak.

9.2 Online algorithm for regular input

For regular input, we can give the exact formula for the offline optimum using the lower bound obtained in Lemma 5. However, in the online problem we do not know the total number of jobs in the input; therefore, we propose an algorithm that determines the replenishment times in advance.
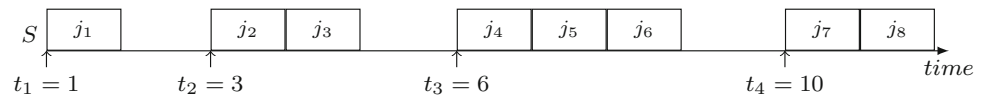
We distinguish the cases $K = 1$ and $K > 1$. Let

$$t_i := \begin{cases} \frac{i(i+1)}{2} & \text{if } K = 1, \\ K \frac{i^2+3i}{2} & \text{if } K > 1. \end{cases}$$

In Algorithm Max_Flowtime, the replenishments occur at the time points t_i, and also after the release date of the last job. At each replenishment, all yet unscheduled jobs are put on the machine in increasing release date order. The schedule obtained for the first few jobs when $K = 1$ is shown in Fig. 8.

Theorem 15 *Algorithm Max_Flowtime is $\sqrt{2}$-competitive on regular input.*

Proof We will prove the theorem separately for $K = 1$ and $K > 1$.

Fig. 8 Schedule S created by Algorithm Max_Flowtime for $K = 1$

For $K = 1$, first we check the competitive ratio for the case $n = t_i$. The maximum flow time is given by the job started in t_i; therefore, the total cost of the algorithm is: $i + t_i - t_{i-1} = 2i$.

Using the lower bound for the offline optimum obtained in Lemma 5, it is enough to check if $2\sqrt{2}\sqrt{\frac{i(i+1)}{2}} \geq 2i$, which holds for each $i \geq 1$.

Now suppose we have $n = t_i + j$ jobs, where $1 \leq j < t_{i+1} - t_i = i + 1$. In the online algorithm, the last replenishment is at $t_i + j$, and the cost is: $i + 1 + \max\{t_i - t_{i-1}, j\} = 2i + 1$. It is easy to verify that $2\sqrt{2}\sqrt{\frac{i(i+1)}{2}} + j \geq 2i + 1$, hence, the algorithm is $\sqrt{2}$-competitive for $K = 1$.

For $K > 1$, there are three cases to consider: $n < 2K$, $n = t_i$, and $n = t_i + j$ for $1 \leq j < t_{i+1} - t_i = K(i + 2)$.

If $n < 2K$, we obtain that $ALG = K + n = OPT$.

If $n = t_i$, the total cost of the algorithm is $Ki + t_i - t_{i-1} = K(2i + 1)$. Thus, we have to compare it to the lower bound from Lemma 5, which is $2K\sqrt{\frac{i^2+3i}{2}}$. It is easy to verify that $2\sqrt{2}K\sqrt{\frac{i^2+3i}{2}} \geq K(2i + 1)$ for every $i, K \geq 1$.

If $n = t_i + j$ for $1 \leq j < (t_{i+1} - t_i) = K(i + 2)$, the total cost of the algorithm is $K(i + 1) + \max\{j, t_i - t_{i-1}\} = K(i + 1) + \max\{j, K(i + 1)\}$.

For $1 \leq j \leq K(i + 1)$, one can easily verify that $2\sqrt{2}K\sqrt{\frac{i^2+3i}{2}} + j \geq K(2i + 2)$, and for $K(i + 1) < j < K(i + 2)$, $2\sqrt{2}K\sqrt{\frac{i^2+3i}{2}} + j \geq K(2i + 3)$ hold, from which $\sqrt{2}OPT \geq ALG$ follows.

Therefore, we have shown that the algorithm is $\sqrt{2}$-competitive. The analysis is tight, because one can upper bound the offline optimum for n jobs with $2\sqrt{Kn} + K$ for any fixed K, and for $n = t_i$ we have:

1. If $K = 1$, then $\frac{ALG}{OPT} \geq \frac{2i}{\sqrt{2i(i+1)}+1} \to \sqrt{2}$ if $i \to \infty$.
2. If $K > 1$, then $\frac{ALG}{OPT} \geq \frac{K(2i+1)}{K\sqrt{2(i^2+3i)}+K} \to \sqrt{2}$ if $i \to \infty$.   □

Algorithm Max_Flowtime

Initialization: $t := 1$ and $i := 1$.

1. If $t = t_i$ or no job arrives at t, then
   a. Replenish the resource, and start the yet unscheduled jobs from t.
   b. If there are no more jobs, then stop; otherwise let $t := t + (t_i - t_{i-1})$, $i := i + 1$.

   Else $t := t + 1$.
2. Go to step 1.

Algorithm Max_Flowtime belongs to a family of online algorithms, where the first replenishment occurs at t_1 (assuming $n \geq t_1$), and after scheduling the jobs ready to be started, it leaves a gap of length $K\delta$, where δ is some fixed positive integer, before replenishing again (unless there are no more jobs, in which case the algorithm replenishes immediately, and start the remaining jobs).

In Algorithm Max_Flowtime, we have $t_1 = 1$ for $K = 1$ and $t_1 = 2K$ for $K > 1$, and $\delta = 1$ for both cases. By simple calculations, it can also be shown that these type of algorithms can have the best competitive ratio of $\sqrt{2}$; therefore, our algorithm is the best possible among them.

Theorem 16 *On regular input, there does not exist an online algorithm with competitive ratio better than $\frac{4}{3}$.*

Proof Suppose an online algorithm makes the first replenishment at time point t, and consider a regular input with $t + 1$ jobs. Then $ALG = 2K + t$, since there is one replenishment in t serving t jobs with maximum flow time of t, and there is another replenishment in $t + 1$ serving only one job. We distinguish three cases for the possible values of t:

1. If $t \leq 2K - 4$, then $OPT \leq K + t + 1$, since this is the cost of a solution with one replenishment in $t + 1$, therefore:
$$\frac{ALG}{OPT} \geq \frac{2K + t}{K + t + 1} \geq \frac{4}{3}.$$

2. If $2K - 3 \leq t \leq 2K + 3$, then $ALG = 4K + i$, where $i = t - 2K \in [-3, 3]$, and $OPT \leq 3K + 3$, since this is the cost of a solution with one replenishment. Therefore,
$$\frac{ALG}{OPT} \to \frac{4}{3}, \text{ if } K \to \infty.$$

3. If $t \geq 2K + 4$, then $OPT \leq 2K + \lceil\frac{t+1}{2}\rceil$, since this is the cost of a solution with two replenishments, therefore:
$$\frac{ALG}{OPT} \geq \frac{2K + t}{2K + \lceil\frac{t+1}{2}\rceil} \geq \frac{2K + t}{2K + \frac{t+1}{2} + 1} \geq \frac{4}{3}.$$
                                                                                   □

To reduce the upper bound of $\sqrt{2}$ on the competitive ratio of online algorithms for $1|jrp, s = 1, p_j = 1$, regular $r_j|F_{\max} + c_Q$, we believe that the t_i should be chosen in a more complicated manner than in Algorithm Max_Flowtime, which may have to be paired with a more complicated analysis as well.

9.3 Lower bound on the competitive ratio

If the input is not regular (i.e., the release dates of the jobs are arbitrary), then we have a higher lower bound:

Theorem 17 *There is no* $\left(\frac{\sqrt{5}+1}{2} - \epsilon\right)$*-competitive online algorithm for any constant* $\epsilon > 0$ *for* $1|jrp, s = 1, p_j = 1, r_j|F_{\max} + c_Q$.

Proof Consider an arbitrary algorithm. Suppose that job j_1 arrives at time 0. If no more jobs arrive, and the algorithm starts j_1 at t, then it is at least $c_1(t, K)$-competitive, where $c_1(t, K) = \frac{K+t+1}{K+1}$. If t further jobs arrive at $t + 1$, then the algorithm is at least $c_2(t, K)$-competitive, where $c_2(t, K) = \frac{2K+t+1}{K+t+2}$. To get a lower bound for an arbitrary online algorithm, we want to calculate the value of

$$\max_K \min_t \max(c_1(t, K), c_2(t, K)).$$

It is easy to see that $c_1(t, K)$ is an increasing function of t, while $c_2(t, K)$ is a decreasing function of t. Therefore, $\max(c_1(t, K), c_2(t, K))$ is minimal for a $\bar{t} \geq 0$ if $c_1(\bar{t}, K) = c_2(\bar{t}, K)$. Some algebraic calculations show that this happens if $\bar{t} = \frac{\sqrt{5K^2+4K}-K-2}{2}$. Substituting $\bar{t}$ into $c_1(t, K)$, we get $\lim_{K\to\infty} c_1(\bar{t}, K) = \frac{\sqrt{5}+1}{2}$. Therefore, for any $\varepsilon > 0$, there is a sufficiently large K, such that there is no $\left(\frac{\sqrt{5}+1}{2} - \epsilon\right)$-competitive algorithm for the problem. □

10 Conclusions

In this paper, we have combined single-machine scheduling with the joint replenishment problem, where the processing of any job on the machine can only be started if the required item types are ordered after the release date of the job. We have proved complexity results, and devised polynomial time algorithms both for the offline and the online variant of the problem. However, several open questions remained, and we list some of the most intriguing ones. Is the offline problem with the $\sum w_j C_j$ objective solvable in polynomial time when $p_j = p$ and s is constant, while the w_j are arbitrary? What is the best competitive ratio for the considered online problems? In particular, for the online problem with regular input and maximum flow time objective, only the number of the jobs is unknown, yet there is a gap between the best upper and lower bound. What can we say in case of more complex machine environments?

Declarations

References

Afrati, F., Bampis, E., Chekuri, C., Karger, D., Kenyon, C., Khanna, S., Milis, I., Queyranne, M., Skutella, M., Stein, C., & Sviridenko, M. (1999). Approximation schemes for minimizing average weighted completion time with release dates. In *40th annual symposium on foundations of computer science (Cat. No. 99CB37039)* (pp. 32–43). IEEE

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

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

Averbakh, I., & Baysan, M. (2013). Approximation algorithm for the online multi-customer two-level supply chain scheduling problem. *Operations Research Letters, 41*(6), 710–714.

Averbakh, I., & Xue, Z. (2007). On-line supply chain scheduling problems with preemption. *European Journal of Operational Research, 181*(1), 500–504.

Azar, Y., Epstein, A., Jeż, L., & Vardi, A. (2016). Make-to-order integrated scheduling and distribution. In *Proceedings of the twenty-seventh annual ACM-SIAM symposium on discrete algorithms* (pp. 140–154). SIAM.

Bansal, N., & Dhamdhere, K. (2007). Minimizing weighted flow time. *ACM Transactions on Algorithms (TALG), 3*(4), 39-es.

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

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

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

Bienkowski, M., Byrka, J., Chrobak, M., Jeż, Ł., Nogneng, D., & Sgall, J. (2014). Better approximation bounds for the joint replenishment

problem. In *Proceedings of the twenty-fifth annual ACM-SIAM symposium on discrete algorithms* (pp. 42–54). SIAM.

Bosman, T., & Olver, N. (2020). Improved approximation algorithms for inventory problems. In *International conference on integer programming and combinatorial optimization* (pp. 91–103). Springer.

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

Chekuri, C., Khanna, S., & Zhu, A. (2001). Algorithms for minimizing weighted flow time. In *Proceedings of the thirty-third annual ACM symposium on theory of computing* (pp. 84–93).

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

Chen, Z. L. (2010). Integrated production and outbound distribution scheduling: Review and extensions. *Operations Research, 58*(1), 130–148.

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

Epstein, L., & Van Stee, R. (2001). Lower bounds for on-line single-machine scheduling. In *International symposium on mathematical foundations of computer science* (pp. 338–350). Springer.

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

Graham, R. L., Lawler, E. L., Lenstra, J. K., & Rinnooy Kan, A. (1979). Optimization and approximation in deterministic sequencing and scheduling: A survey. *Annals of Discrete Mathematics, 5*, 287–326. https://doi.org/10.1016/S0167-5060(08)70356-X

Hoogeveen, J. A., & Vestjens, A. P. (1996). Optimal online algorithms for single-machine scheduling. In *International conference on integer programming and combinatorial optimization* (pp. 404–414). Springer.

Karlin, A. R., Manasse, M. S., McGeoch, L. A., & Owicki, S. (1994). Competitive randomized algorithms for nonuniform problems. *Algorithmica, 11*(6), 542–571.

Karlin, A. R., Manasse, M. S., Rudolph, L., & Sleator, D. D. (1988). Competitive snoopy caching. *Algorithmica, 3*(1), 79–119.

Karp, R. M. (1992). On-line algorithms versus off-line algorithms: How much is it worth to know the future? In *IFIP congress (1)* (Vol. 12, pp. 416–429).

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

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

Lenstra, J. K., Kan, A. R., & Brucker, P. (1977). Complexity of machine scheduling problems. *Annals of Discrete Mathematics, 1*, 343–362.

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

Levi, R., Roundy, R. O., & Shmoys, D. B. (2006). Primal–dual algorithms for deterministic inventory problems. *Mathematics of Operations Research, 31*(2), 267–284.

Levi, R., & Sviridenko, M. (2006). Improved approximation algorithm for the one-warehouse multi-retailer problem. In J. Díaz, K. Jansen, J. D. P. Rolim, & U. Zwick (Eds.), *Approximation, randomization, and combinatorial optimization. Algorithms and techniques* (pp. 188–199). Springer.

Nonner, T., & Souza, A. (2009). A 5/3-approximation algorithm for joint replenishment with deadlines. In *International conference on combinatorial optimization and applications* (pp. 24–35). Springer.

Potts, C. N. (1980). Analysis of a heuristic for one machine sequencing with release dates and delivery times. *Operations Research, 28*(6), 1436–1441.

Starr, M. K., & Miller, D. W. (1962). *Inventory control: Theory and practice*. Prentice-Hall.