This is a post-peer-review, pre-copyedit version of an article published in Journal of Scheduling (ISSN 1099-1425).
The final authenticated version is available online at link.springer.com:
https://dx.doi.org/10.1007/s10951-018-0561-5

Single-Machine Batch Scheduling to Minimize the Total Setup Cost in the Presence of Deadlines

Dominik Kress · Maksim Barketau · Erwin Pesch

Received: date / Accepted: date

Abstract We address the single-machine batch scheduling problem with the objective of minimizing the total setup cost. This problem arises when there are n jobs that are partitioned into F families and when setup operations are required whenever the machine switches from processing a job of one family to processing a job of another family. We assume that setups do not require time but are associated with a fixed cost which is identical for all setup operations. Each job has a processing time and an associated deadline. The objective is to schedule all jobs such that they are on time with respect to their deadlines and the total setup cost is minimized. We show that the decision version of this problem is NP-complete in the strong sense. Furthermore, we present properties of optimal solutions and an $O(n\log n + nF)$ algorithm that approximates the cost of an optimal schedule by a factor of F. The algorithm is analyzed in computational tests.

Keywords Batch scheduling · Job families · Setup cost · Strong NP-hardness · Approximation algorithm · EDD-schedule · GT-schedule

D. Kress (✉) · E. Pesch
Management Information Science
University of Siegen
Kohlbettstr. 15, 57068 Siegen, Germany
E-mail: {dominik.kress, erwin.pesch}@uni-siegen.de

M. Barketau
United Institute of Informatics Problems
National Academy of Sciences of Belarus
220012 Minsk, Belarus
E-mail: barketau@mail.ru

E. Pesch
Center for Advanced Studies in Management
HHL Leipzig
Jahnallee 59, 04109 Leipzig, Germany

1 Introduction and Problem Description

When sequentially processing jobs on a single machine in an industrial environment, the change of one job to another may result in a significant *setup cost*. This cost may, for example, be due to the need to modify the machine by installing a different set of tools or loading a new software. In such an environment it can be beneficial to group the jobs into *batches* that share the same setup requirements and can thus be processed contiguously. However, the processing of large batches may delay the processing of important jobs that are included in other batches, which may eventually result in missing the deadlines of these jobs. Hence, there is a trade-off between minimizing the total setup cost of a schedule and the need to guarantee on-time production of jobs.

In the above context, we consider the following scheduling problem. There are n jobs to be sequenced on a single machine. Each of these jobs belongs to one of F nonempty *families*. Jobs within a family are sufficiently similar to be processed on the machine without the need for intermediate setups. Family F_f, $f \in \{1,\dots,F\}$, consists of n_f jobs. Hence, $n = \sum_{f=1}^{F} n_f$. We denote the j-th job of family F_f, $f \in \{1,\dots,F\}$, by (j,f). The *processing time* of this job is denoted by $p_{j,f} \in \mathbb{N}$. Its processing must be completed no later than its *deadline* $d_{j,f} \in \mathbb{N}$ and it may not be preempted. We assume that the jobs of each family are sorted and indexed in non-decreasing order of their respective deadlines. Jobs with identical deadlines within a family are arranged in the order of increasing indices. When the processing of a job of a specific family finishes and the processing of another family's job starts, as well as at the beginning of the schedule, a setup is needed. These setups are assumed to not require time. However, there is a setup cost $s \in \mathbb{N}$ associated with each setup. This cost is identical for all setup operations. As indicated above, a *batch* is a set of jobs of the same family that is processed between two setups of a given

schedule. The formation of batches is subject of the decision making process under consideration in this paper. The processing of a batch is completed, when the processing of all jobs of the batch is completed. We assume that the starting and completion time of each job, i.e. the time instants when the job is started to be processed and when its processing is completed, is independent of the other jobs of its batch. That is, a job "becomes available" (Potts and Kovalyov, 2000) immediately after its processing on the machine is completed, even if the processing of its corresponding batch is not yet completed. This is usually referred to as a *job availability model* (see, for example, Allahverdi et al., 1999, 2008; Potts and Kovalyov, 2000). The problem is to find a schedule that minimizes the total setup cost and that is feasible with respect to the deadlines. As the setup cost is identical for all setup operations, this is equivalent to minimizing the number of batches of the schedule.

1.1 Related Literature and Applications

There exists a plenitude of papers dealing with scheduling problems that involve setup considerations. Instead of summarizing all of these papers, we refer to Błażewicz et al. (2007) and the excellent and extensive reviews by Allahverdi (2015), Allahverdi et al. (1999, 2008) and Potts and Kovalyov (2000). These review papers introduce modifications of the classical three-field notation for scheduling problems proposed by Graham et al. (1979). Based on these modifications, we denote the problem under consideration in this paper by $1 \,|\, d_{j,f}, b = n, SC_{si,b} = s \,|\, TSC$. Here, the first field indicates that we are concerned with a one-machine problem. The second field specifies the job characteristics. Each job (j,f), $f \in \{1,\ldots,F\}$, has an associated deadline $d_{j,f}$, and there is no exogenous upper bound on the maximum batch size b, i.e. the maximum batch size may be equal to the number of jobs, $b = n$. Furthermore, the setup cost is sequence-independent, i.e. it depends solely on the batch that is to be processed, and it is equal to s for all setups. This is indicated by $SC_{si,b} = s$. Finally, the third field refers to the objective of minimizing the total setup cost (TSC).

The specific problem considered in this paper has first been considered by Bruno and Downey (1978), who refer to its decision version as the *Schedule Cost Problem*. The authors provided a proof for its NP-completeness, while it remained open whether the problem is strongly NP-complete. Even though there exists closely related research on similar problem settings where setups are assumed to require time (initiated by Bruno and Downey, 1978, see Cheng et al., 2003; Lu and Yuan, 2007; Tanaev et al., 1998), this question has - to the best of the authors' knowledge - not yet been answered.

There is a wide variety of practical applications of batch scheduling problems. Examples include semiconductor wafer fabrication and semiconductor testing (see, for example, Herrmann and Lee, 1995; Mehta and Uzsoy, 1998), task scheduling on computer systems (e.g. Bruno and Downey, 1978), or the production of colors of paint (see Monma and Potts, 1989, and the references therein). Another application of the problem under consideration in this paper arises, when scheduling inspections of patients on complex medical equipment, as, for example, magnetic resonance tomographs (MRT). Here, the patients can be viewed as jobs, each of which is related to a specific section of the human body (e.g. heart or brain) that requires inspection no later than by a specified deadline. Patients that require inspection of the same body section must be processed and evaluated by a specific doctor and her assistants. The availability of this medical team is especially arranged for each scheduled batch of these patients. A fixed cost is charged for each batch. This cost may, for example, relate to the cost of preparing the team as each member of the team is to get ready to finally perform the task expected to be done. Moreover, there may be additional payments to the doctors and the assistants for processing a batch. Naturally, the total scheduling cost must be minimized.

1.2 Overview and Contribution of the Paper

The contribution and structure of this paper is as follows. First, we provide a proof for the strong NP-completeness of the decision version of $1 \,|\, d_{j,f}, b = n, SC_{si,b} = s \,|\, TSC$ and thus answer the aforementioned open question on the problem's complexity status posed by Bruno and Downey (1978). This is subject of Section 2. Second, in Section 3, we present an $O(n \log n + nF)$ algorithm that approximates the cost of an optimal schedule by a factor of F. It is based on some properties of optimal schedules that we derive in Section 3.1. The algorithm itself is described in Section 3.3 and it is analyzed in computational tests in Section 4. The paper closes with a summary in Section 5.

2 Computational Complexity

We will make use of the strongly NP-complete 3-Partition problem, which is defined as follows (Garey and Johnson, 1979): Given $3m+1$ integers $u_1,\ldots,u_{3m},B$ with $\sum_{i=1}^{3m} u_i = mB$ and $\frac{B}{4} < u_i < \frac{B}{2}$ $\forall i = 1,\ldots,3m$. Does there exist a partition of the set $\{1,\ldots,3m\}$ into m subsets $U_1,\ldots,U_m$, such that $\sum_{i \in U_j} u_i = B$ $\forall j = 1,\ldots,m$? Note that for every yes-instance of 3-Partition, we have $|U_i| = 3$ $\forall i = 1,\ldots,m$.

We will present a pseudo-polynomial transformation (Garey and Johnson, 1979) from 3-Partition to the decision version of $1 \,|\, d_{j,f}, b = n, SC_{si,b} = s \,|\, TSC$, which is defined in analogy to its optimization version and asks if there exists a feasible schedule with a total scheduling cost of no more

than a given K. This transformation will prove strong NP-hardness of $1 \mid d_{j,f}, b = n, SC_{si,b} = s \mid TSC$. The proof is an adaption of the proofs presented by Cheng et al. (2003) and Lu and Yuan (2007), who provide the following lemma.

Lemma 1 (Lu and Yuan (2007)) *Suppose $\mu_1, \ldots, \mu_m$ and B are $m+1$ positive integers such that $\mu_1 + \cdots + \mu_m = mB$ and*

$$(m+1)\sum_{i=1}^{j-1}\mu_i + \sum_{i=j}^{m} i\mu_i \leq (m+1)\sum_{i=1}^{j-1} B + \sum_{i=j}^{m} iB$$

holds for each j with $1 \leq j \leq m$. Then $\mu_1 = \cdots = \mu_m = B$.

We will additionally make use of the following lemma.

Lemma 2 *Let I be an instance of $1 \mid d_{j,f}, b = n, SC_{si,b} = s \mid TSC$. If there exists a feasible schedule for I, then there exists an optimal schedule S with the jobs of each family being processed in nondecreasing order of their deadlines, i.e. for any family F_f, $f \in \{1, \ldots, F\}$, and any two jobs (i,f) and (j,f), $i \neq j$, (i,f) is scheduled before (j,f) in S if $d_{i,f} \leq d_{j,f}$.*

Proof The correctness of the claim follows from some simple job shifting and interchange arguments (see the analogous proof by Gerodimos et al., 1999, for a closely related problem class; cf. also Lu and Yuan, 2007).

Consider an optimal schedule S of I and let (i,f) and (j,f), $f \in \{1, \ldots F\}$, be two jobs $i \neq j$ such that (i,f) is the last job of family F_f that is processed before (j,f) in S, where $d_{j,f} < d_{i,f}$. Now, modify S as follows. First, shift job (i,f) to be processed right before job (j,f) while not changing the respective order of other jobs. Second, interchange jobs (i,f) and (j,f). It is easy to see that the resulting schedule is feasible. Moreover, the number of batches does not increase due to the modification. Hence, the resulting schedule is optimal. By repeating this procedure for all suitable pairs of jobs, we obtain an optimal schedule with the jobs of each family being processed in non-decreasing order of their deadlines. □

In order to align with well established terms in the scheduling literature, we will refer to a sequence of jobs in nondecreasing order of their deadlines (as used in Lemma 2) as an earliest due date (EDD) order in the remainder of this paper.

While the proof of the following Theorem 1 is based on ideas of Cheng et al. (2003) and Lu and Yuan (2007) and partially follows similar arguments as these proofs, it especially differs in the use and (sub-) proofs of Properties 1–5.

Theorem 1 *The decision version of $1 \mid d_{j,f}, b = n, SC_{si,b} = s \mid TSC$ is strongly NP-complete.*

Table 1 Construction of I_B

Scheduling cost:	$K = (7m+1)s$, with $s \in \mathbb{N}$ arbitrary
Number of jobs:	$n = 3m(m+1) + 2m + 1$
Number of families:	$F = 4m+1$
Distinct deadlines:	$D_j = (2j-1)(X+Y)$ $+\frac{3m(m+1)}{2}Z + \frac{3(j-1)(2m-j+2)}{2}Z$ $+(m+1)\sum_{i=1}^{j-1} B$ $+\sum_{i=j}^{m} iB$ for $1 \leq j \leq m+1$
$f = 1, \ldots, 3m$:	$F_f = \{(j,f) \mid 1 \leq j \leq m+1\}$
	$p_{j,f} = u_f + Z \quad \forall (j,f) \in F_f$
	$d_{j,f} = D_j \quad \forall (j,f) \in F_f$
$f = 3m+1, \ldots, 4m$:	$F_f = \{(j,f) \mid 1 \leq j \leq 2\}$
	$p_{j,f} = X + Y \quad \forall (j,f) \in F_f$
	$d_{j,f} = D_{f-3m-1+j} \quad \forall (j,f) \in F_f$
$f = 4m+1$:	$F_f = \{(1,f)\}$
	$p_{1,f} = X + Y$
	$d_{1,f} = D_{m+1}$

Proof It is easy to see that the decision version of $1 \mid d_{j,f}, b = n, SC_{si,b} = s \mid TSC$ is in NP.

Given an instance I_P of 3-Partition, we construct an instance I_B of the decision version of $1 \mid d_{j,f}, b = n, SC_{si,b} = s \mid TSC$ in polynomial time as summarized in Table 1. Here, X, Y, and Z are arbitrary, nonnegative integers with

$$X > \frac{1}{2}(m^2 - m)B,$$

$$Z > \frac{1}{4}(m^2 + m)B,$$

and

$$Y \geq \frac{3}{2}(m^2 - m)Z.$$

Observe that family F_f relates to u_f, $f \in \{1, \ldots, 3m\}$. Furthermore, note that

$$D_1 = \frac{m^2+m}{2}(3Z+B) + X + Y$$

and

$$D_j = D_{j-1} + (m-j+2)(3Z+B) + 2(X+Y)$$

for $2 \leq j \leq m+1$. Hence, $D_i < D_j$ for $i, j \in \{1, \ldots, m+1\}$ and $i < j$.

Figure 1 illustrates the construction of I_B.

In the following, we will show that I_P is a yes-instance if and only if I_B is a yes-instance and that we have therefore constructed a pseudo-polynomial transformation (Garey and Johnson, 1979) from 3-Partition to the decision version of $1 \mid d_{j,f}, b = n, SC_{si,b} = s \mid TSC$, so that the latter problem is strongly NP-complete. We will refer to all jobs with deadline D_j, $j \in \{1, \ldots, m+1\}$, as D_j-jobs. Furthermore, given a schedule for I_B, we will denote the completion time of the

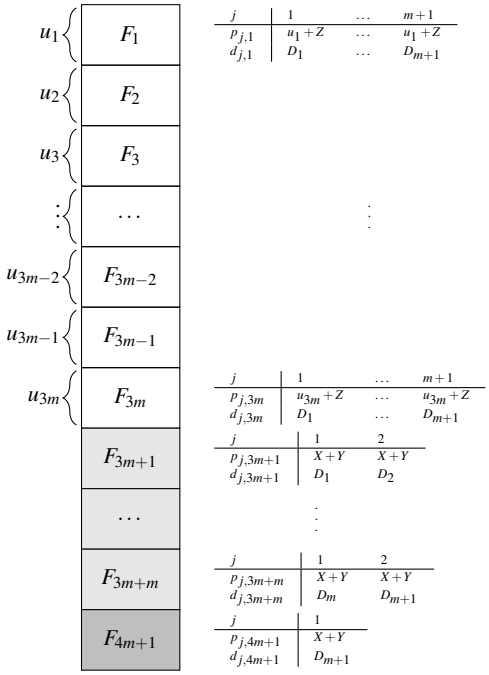

Fig. 1 Construction of I_B

last D_j-job within this schedule by C_j, $j \in \{1,\dots,m+1\}$. We will restrict ourselves to the nontrivial case $m > 1$.

Assume that I_P is a yes-instance. Then it is possible to re-label the indices of $u_1,\dots,u_{3m}$ and their corresponding families in I_B, such that $u_{3i-2}+u_{3i-1}+u_{3i} = B$ for $1 \le i \le m$. Now, let each family F_f, $f \in \{3m+1,\dots,4m+1\}$, act as a batch and divide all remaining families F_f, $f \in \{1,\dots,3m\}$, into exactly two batches $B_f = \left\{(j,f) | 1 \le j \le \left\lceil \frac{f}{3} \right\rceil \right\}$ and $A_f = F_f \setminus B_f$. Construct a schedule S by sequencing these batches as follows:

$$S = B_1, B_2, \dots, B_{3m}, F_{3m+1}, A_1, A_2, A_3, F_{3m+2}, \dots, A_{3m-2}, A_{3m-1}, A_{3m}, F_{4m+1}.$$

The jobs within each batch are sequenced in their EDD order. It is easy to see that this requires $7m+1$ setup operations so that the total scheduling cost is K. Furthermore,

$$C_1 = \sum_{i=1}^{m} i \sum_{j=3i-2}^{3i} u_j + \sum_{i=1}^{m} 3iZ + X + Y = \sum_{i=1}^{m} (iB + 3iZ) + X + Y$$
$$= (B+3Z)\sum_{i=1}^{m} i + X + Y = D_1,$$

and

$$C_j = C_{j-1} + (m-j+2)(3Z + \sum_{i=3(j-2)+1}^{3(j-1)} u_i) + 2(X+Y)$$
$$= C_{j-1} + (m-j+2)(3Z+B) + 2(X+Y) = D_j$$

for $2 \le j \le m+1$. Hence, all jobs are on time and we have established a solution to I_B.

Now assume that I_B is a yes-instance and let S be a feasible schedule with a total scheduling cost of no more than K that satisfies the EDD property of Lemma 2. Then, for each batch of a family F_f, $f \in \{1,\dots,4m+1\}$, in S and any two jobs (i,f) and (j,f) with $i < j$ that are included in the batch, all jobs of the set $\{(k,f) | i < k < j\}$ are included in the batch as well. If this were not the case, the EDD property would not be fulfilled, which can be seen when recalling that $D_i < D_j$ for $i, j \in \{1,\dots,m+1\}$ and $i < j$. Additionally, note that all families $F_1,\dots,F_{3m}$ include exactly one D_1-job. We denote the unique batch of family F_f, $f \in \{1,\dots,3m\}$, that includes a D_1-job by B_f and relabel the indices of $F_1,\dots,F_{3m}$ and their corresponding integer numbers of I_P, such that $|B_1| \le |B_2| \le \dots \le |B_{3m}|$. Hence, $B_f = \{(1,f),\dots,(|B_f|,f)\}$ for $f \in \{1,\dots,3m\}$.

We will make use of the following property:

Property 1. Let S be constructed as described above. Then $C_j < C_{j+1}$ for all $j \in \{1,\dots,m\}$.

Proof of Property 1. We will prove Property 1 by mathematical induction. Consider the base case $j = 1$ and assume to the contrary that $C_1 \ge C_2$. Then, by definition of 3-Partition and the construction of I_B, we have

$$C_1 \ge 2mB + 6mZ + 3(X+Y). \tag{1}$$

The first two summands of the right hand side of (1) relate to the processing of the D_1- and D_2-jobs of families F_f, $f = 1,\dots,3m$, while the last summand considers these jobs in families $3m+1$ and $3m+2$. It is easy to show that (1) necessarily results in $C_1 > D_1$ if

$$X > \frac{1}{4}\left(m^2 - 3m\right)B,$$

and

$$Y \ge \frac{3}{4}\left(m^2 - 3m\right)Z,$$

by some straightforward algebraic transformations. Hence, by choosing X and Y as described in the above construction of I_B, at least one D_1-job is late, which contradicts the feasibility of S. Thus, we must have $C_1 < C_2$, and we have proven the base case.

Now, let $j \ge 2$ and assume that $C_i < C_{i+1}$ for all $i < j$. Assume that $C_j \ge C_{j+1}$. Then

$$C_j \ge (j+1)mB + (j+1)3mZ + (2j+1)(X+Y)$$

in analogy to (1). One can easily show that this results in $C_j > D_j$ if

$$X > \frac{1}{4}\left(m^2 - 3m - j^2 + 3j - 2\right)B,$$

and

$$Y \ge \frac{3}{4}\left(m^2 - 3m - j^2 + 3j - 2\right)Z.$$

Additionally, note that $f(j) := -j^2 + 3j - 2$ is monotonically decreasing in j for $j \geq 2$. Hence, by choosing X and Y as above, at least one D_j-job is late, which contradicts the feasibility of S. We must therefore have $C_j < C_{j+1}$, which proves the inductive step and the correctness of Property 1.

By making use of Property 1, we can derive the following additional property:

Property 2. Let S be constructed as described above. Then, for all $j \in \{1,\dots,m\}$, S can be transformed into a feasible schedule S', where the jobs $(1,3m+j)$ and $(2,3m+j)$ are processed without intermediate setup, i.e. where $(2,3m+j)$ is processed directly after $(1,3m+j)$, without increasing the number of setup operations.

Proof of Property 2. Assume there is a family F_{3m+j}, $j \leq m$, where $(2,3m+j)$ is not processed directly after $(1,3m+j)$ in S. Then there is one setup operation immediately before and another one immediately after job $(1,3m+j)$ in S. We can assume that the same holds for job $(2,3m+j)$, because, if this job is the last job that is processed in S (which is only possible for $j = m$ because of Property 1 and the construction of I_B) we can swap the positions of jobs $(2,4m)$ and $(1,4m+1)$ in S without this swap resulting in infeasibility.

Case 1: Assume that $(1,3m+j)$ is the last D_j-job scheduled in S. As $(2,3m+j)$ is processed later than $(1,3m+j)$ because of the EDD property of Lemma 2, we may move job $(2,3m+j)$ to be the immediate successor of $(1,3m+j)$. This results in all jobs at positions between $(1,3m+j)$ and $(2,3m+j)$ being shifted towards later completion times by the processing time of $(2,3m+j)$. These shifted jobs, however, must necessarily have a deadline of at least D_{j+1}. Hence, as S is feasible and $(2,3m+j)$ is a D_{j+1}-job, the modified schedule S' that results from shifting the jobs does not violate any deadline. In S', we have a setup operation between the predecessor of $(1,3m+j)$ in S and $(1,3m+j)$, $(2,3m+j)$ and the successor of $(1,3m+j)$ in S, and a potential setup between the predecessor and the successor of $(2,3m+j)$ in S. Moreover, there is no setup between the two jobs of family F_{3m+j} in S'. All remaining setup operations remain unchanged, so that the number of setups in S' is not larger than in S.

Case 2: Assume that $(1,3m+j)$ is not the last D_j-job scheduled in S. Choose a D_j-job j' at a later position than $(1,3m+j)$ in S. Assume that j' is the last D_j-job in S. If $(2,3m+j)$ is scheduled on a later position than j' in S, then shift jobs $(1,3m+j)$ and $(2,3m+j)$, such that $(1,3m+j)$ is scheduled immediately after job j', directly followed by job $(2,3m+j)$. In the resulting schedule S', $(1,3m+j)$ will be completed at the time when j' was completed in S and by the same arguments as in case 1 for $(2,3m+j)$, we can conclude that S' does not violate any deadline. In S', we have a setup operation before job $(1,3m+j)$ and after job $(2,3m+j)$, as well as potential setup operations between the predecessor and the successor of $(1,3m+j)$ (or at the start of the schedule) and $(2,3m+j)$ in S. Moreover, there is no setup between the two jobs of family F_{3m+j} in S'. All remaining setup operations remain unchanged, so that the number of setups in S' is not larger than in S. If, on the other hand, $(2,3m+j)$ is scheduled on a position somewhere between $(1,3m+j)$ and j', then shift $(1,3m+j)$ and $(2,3m+j)$ immediately behind j'. As before, the resulting schedule S' does not violate any deadline and the number of setups does not increase. This concludes the proof of Property 2.

Because of Property 2, we may assume that S is such that the jobs $(1,3m+j)$ and $(2,3m+j)$ are processed without an intermediate setup operation for the remainder of this proof. Then, we can make the following observation:

Property 3. Let S be constructed as described above and let $j \in \{1,\dots,m\}$. Then $(1,3m+j)$ is the last D_j-job that is processed before C_j.

Proof of Property 3. Assume the opposite. Then, by making use of Property 1, we know that we must have

$$C_j \geq jmB + 3mjZ + 2j(X+Y).$$

In line with the derivations in the proof of Property 1, it is easy to show that this necessarily results in $C_j > D_j$ if

$$X > \frac{1}{2}\left(m^2 - m\right)B,$$

and

$$Y \geq \frac{3}{2}\left(m^2 - m\right)Z,$$

Therefore, due to the choice of X and Y when generating I_B, at least one D_j-job is late, which is a contradiction. This proves the correctness of Property 3.

We will make use of another property that is based on Property 1:

Property 4. Let $j \in \{1,\dots,m\}$ and let S be constructed as described above. Then there must exist at least $l_j := \sum_{i=j}^{m} 3(m+1-i)$ jobs that belong to families F_f, $f \in \{1,\dots,3m\}$, and that have not been started to be processed at time C_j in S.

Proof of Property 4. First, note that every job that has been started to be processed at time C_j, must also be completed at time C_j. This is because preemption is not allowed and the latter completion time must correspond to a time instant at which the processing of a D_j-job has just been completed. Now, assume to the contrary that there exist less than $\sum_{i=j}^{m} 3(m+1-i)$ jobs of families F_f, $f \in \{1,\dots,3m\}$, that have not been started to be processed at time C_j. Then, by definition of 3-Partition and the construction of I_B, we have

$$\begin{aligned} C_j >& 3m(m+1)Z - \sum_{i=j}^{m} 3(m+1-i)Z + Z + (m+1)mB \\ & - \sum_{i=j}^{m} 3(m+1-i)\frac{B}{2} + \frac{B}{2} + (2j-1)(X+Y). \end{aligned} \tag{2}$$

To see this, note that each family F_f, $f \in \{1,\dots,3m\}$, is composed of exactly $m+1$ jobs. The processing time of each of these jobs is a sum with two summands: Z and u_f. The first three terms of the right hand side of (2) therefore bound C_j from below by considering the difference of the total processing time of the jobs in families F_f, $f = 1,\dots,3m$, and the processing time of the relevant jobs that have not been completed at time C_j, when only taking into account the first summands of the processing times (i.e. Z for each job). Similarly, the next three terms of the right hand side of (2) take account of the remaining summands of the processing times and make use of the fact that $\sum_{i=1}^{3m} u_i = mB$ and $u_i < \frac{B}{2}$ for all $i = 1,\dots,3m$ by definition of 3-Partition. Finally, the last term of the right hand side of (2) applies Property 1 and makes use of the fact that, for $2 \leq j \leq m$ (the case $j = 1$ is similar), families $F_{3m+(j-1)}$ and F_{3m+j} contain a D_j-job with processing time $X+Y$.

It is easy to show that (2) necessarily results in $C_j > D_j$ if

$$Z > \frac{1}{4}\left(m^2 + 3m - 2mj + j^2 - 3j\right)B$$

by some straightforward algebraic transformations. Additionally, note that $g(j) := -2mj + j^2 - 3j$ is monotonically decreasing in j for $1 \leq j \leq m+1$. Hence, by choosing Z as described in the above construction of I_B, at least one D_j-job is late, which contradicts the feasibility of S and thus proves the claim of Property 4.

We will furthermore need the following property that extends Property 4:

Property 5. Let $j \in \{1,\dots,m\}$ and let S be constructed as described above. Then there exist at most l_j jobs that belong to families F_f, $f \in \{1,\dots,3m\}$, and that have not been started to be processed at time C_j in S.

Proof of Property 5. We will refer to a setup operation after a job (i,f), with $i < m+1$ and $f \in \{1,\dots,3m\}$ as a *cut*. Note that, due to the fact that $K = (7m+1)s$, there may exist at most $3m$ cuts in S.

We will prove Property 5 by mathematical induction. Consider the base case $j = m$ and recall that every job that has been started to be processed at time C_m, must also be completed at time C_m. Now, assume to the contrary of Property 5 that there are $l_m + 1 = 4$ (any larger number of jobs can be proven in analogy) jobs of families F_f, $f \in \{1,\dots,3m\}$, that have not been started to be processed at time C_m. Because of Properties 1–3, as well as the construction of I_B and S, we know that this requires four cuts. Now, stepwise add the least possible amount of cuts in order to guarantee the values $l_{m-1},\dots,l_1$ of Property 4. Obviously, we need at least three additional cuts in order to guarantee $4 + 3 \cdot 2 = 10 > l_{m-1} = 9$ unprocessed jobs at time C_{m-1}. If $m = 2$, this results in a total of at least 7 cuts and, thus, is a contradiction. Now, note that because of the fact that $l_{i-1} - l_i = 3(m+2-i)$ for $i > 1$, the same reasoning, i.e. the need for at least three additional cuts, applies when stepwise considering all l_i with $i < m-1$. Hence, we have a total of at least $3m+1$ cuts and, thus, a contradiction for any value of m, which proves the base case.

Now, let $2 \leq k \leq m$ be such that $j = m-(k-1)$ and assume that there are exactly (due to Property 4) l_i jobs that belong to families F_f, $f \in \{1,\dots,3m\}$, and that have not been started to be processed at time C_i in S for all $i \in \{m-k+2,\dots,m\}$. Because of Properties 1–3, as well as the construction of I_B and S, this requires a total of at least $3(k-1)$ cuts. We will proceed in analogy to the base case. Assume to the contrary of Property 5, that there are $l_{m-(k-1)} + k$ jobs of families F_f, $f \in \{1,\dots,3m\}$, that have not been started to be processed at time C_j. As above, this requires at least four additional cuts. As in the base case, when considering the remaining l_i with $i < j-1$, the need for at least three additional cuts for each i arises. Hence, we have a total of at least $3(k-1)+4+3(m-k) = 3m+1$ cuts in S and, thus, a contradiction, which proves the inductive step and the correctness of Property 5.

Finally, based on Properties 1–5, it is straightforward to observe that we may assume that each family F_f, $f = 3m+1,\dots,4m$, is processed in one batch in S, while each family F_f, $f = 1,\dots,3m$, is divided into exactly two batches, so that $|B_{3i-2}| = |B_{3i-1}| = |B_{3i}|$ for all $i \in \{1,\dots,m\}$, $B_f = \left\{(j,f) | 1 \leq j \leq \left\lceil \frac{f}{3} \right\rceil \right\}$, and $A_f := F_f \setminus B_f$ defines a batch of S.

Now, denote the sum $u_{3i-2} + u_{3i-1} + u_{3i}$ by μ_i for all $i \in \{1,\dots,m\}$. Then, by definition of 3-Partition, $\sum_{i=1}^{m} \mu_i = mB$. Furthermore, based on the above deliberations, we have

$$C_j = (2j-1)(X+Y) + \sum_{i=1}^{m} i(3Z+\mu_i) + \sum_{i=1}^{j-1}(m-i+1)(3Z+\mu_i)$$

for all $j \in \{1,\dots,m\}$. By additionally demanding $C_j \leq D_j$ for all $j \in \{1,\dots,m\}$, we derive

$$(m+1)\sum_{i=1}^{j-1}\mu_i + \sum_{i=j}^{m} i\mu_i \leq (m+1)\sum_{i=1}^{j-1} B + \sum_{i=j}^{m} iB,$$

for all $j \in \{1,\dots,m\}$ as necessary conditions for feasibility of S after some straightforward algebraic transformations. By Lemma 1, we must therefore have $\mu_1 = \dots = \mu_m = B$, so that we have established a solution to I_P. This concludes the proof. □

3 Approximation Algorithm

We will now present an approximation algorithm for $1\,|\,d_{j,f}, b = n, SC_{si,b} = s\,|\,TSC$. It is based on some properties of optimal solutions that are subject of Section 3.1. These properties complement the results presented in Section 2. Before describing and analyzing the algorithm in Section

3.3, we will slightly modify the notation and present a mathematical model for $1 \mid d_{j,f}, b = n, SC_{si,b} = s \mid TSC$ in Section 3.2. This model will later be used in our computational tests.

3.1 Properties of Optimal Solutions

Consider an optimal job sequence with the EDD property within families as presented in Lemma 2. Let (i,f) and (k,f), $f \in \{1,\ldots,F\}$, with $i < k$ be two jobs with identical deadline. Then we assume without loss of generality that (i,f) is scheduled before (k,f) within the sequence. Now, let (j,f) and $(j-1,f)$, $f \in \{1,\ldots,F\}$, be two jobs of this sequence. Obviously, (j,f) completes no later than $d_{j,f}$. Thus, $(j-1,f)$ completes no later than $d_{j,f} - p_{j,f}$. If $d_{j-1,f} > d_{j,f} - p_{j,f}$, we modify the problem instance under consideration by redefining $d_{j-1,f} = d_{j,f} - p_{j,f}$. Note that, after this modification, the optimal job sequence remains feasible and that there are no additional feasible schedules. Thus, the optimal number of batches does not change. By repeating this procedure for a finite number of times, we obtain a modified problem instance with the following property:

$$d_{j,f} - d_{j-1,f} \geq p_{j,f} \quad \forall f \in \{1,\ldots,F\}, j \in \{2,\ldots,n_f\}. \quad (3)$$

The modified problem is equivalent to the initial problem in the sense that the cost of an optimal schedule is identical in both problems and every feasible schedule of the modified problem is feasible for the initial problem. We will therefore assume that (3) holds for any problem instance considered in the remainder of this section.

Now, consider a general *EDD-schedule*, i.e. a job sequence with all jobs (in contrast to the jobs within each family as used in Lemma 2) being processed in non-decreasing order of their deadlines, which can obviously be determined in $O(n \log n)$ time. Without loss of generality, we assume that ties among jobs are broken in favor of the job with the smaller family index or, if this index is identical, the smaller job index. Hence, there is a unique EDD-schedule for a given problem instance. We observe:

Lemma 3 *Let I be an instance of $1 \mid d_{j,f}, b = n, SC_{si,b} = s \mid TSC$. There exists a feasible schedule for I if and only if the EDD-schedule of I is feasible.*

Proof The proof is in analogy to the proof of Lemma 2. □

The EDD-schedule does, however, not in general provide a good approximation for the optimal schedule, as can be seen in the following Lemma 4.

Lemma 4 *An EDD-schedule for an instance of $1 \mid d_{j,f}, b = n, SC_{si,b} = s \mid TSC$ can have $n/2$ times more batches than a corresponding optimal schedule.*

Proof Construct an instance I of $1 \mid d_{j,f}, b = n, SC_{si,b} = s \mid TSC$ by defining two families ($F = 2$), each of which contains the same number of jobs, i.e. $n_1 = n_2 = n/2$. Set $p_{j,1} = p_{j,2} = 2$, $d_{j,1} = n + 2j - 1$, and $d_{j,2} = n + 2j$ for all $j \in \{1,\ldots,n/2\}$. It is easy to see, that the number of batches in the EDD-schedule of I is n. However, because it is feasible to process all jobs of the first family before processing the jobs of the second family, the optimal schedule has exactly two batches. Thus, the EDD-schedule has $n/2$ times more batches than the optimal schedule. □

We will now turn our attention to group technology schedules (*GT-schedules*). Here, one requires precisely F setups in the schedule, so that each family must form exactly one batch (see, for example, Baker and Trietsch, 2009). A specific class of GT-schedules processes the families F_f, $f = 1,\ldots,F$, in non-decreasing order of the values $d_{1,f} + T_f - p_{1,f}$, where T_f is defined to be the total processing time of the jobs in family F_f, i.e. $T_f := \sum_{j=1}^{n_f} p_{j,f}$. We assume that ties among families are broken by favoring the family with the smaller family index. The jobs within each batch are processed in their EDD order with ties being broken as described above. We refer to the resulting schedule of a given problem instance as the *GTD-schedule*. Intuitively, the value $d_{1,f} + T_f - p_{1,f}$, $f \in \{1,\ldots,F\}$, represents the largest possible completion time of the batch of family F_f in a feasible *GT*-schedule.

The following lemma provides a feasibility criterion for a GT-schedule.

Lemma 5 *Let I be an instance of $1 \mid d_{j,f}, b = n, SC_{si,b} = s \mid TSC$. There exists a GT-schedule that is feasible (and thus optimal) for I if and only if the GTD-schedule is feasible for I.*

Proof By definition, the GTD-schedule is a specific GT-schedule. Hence, if the GTD-schedule is feasible for I, then it defines a feasible GT-schedule for I. As this schedule has the smallest possible number of batches, it must be optimal.

Now, let S be a feasible GT-schedule for I. By interchanging neighbored jobs within the batches of S, ensure that the jobs within each batch are processed in their EDD order, breaking ties as described above. Because of (3) and because all jobs of the same family are processed continuously and without idle times in S, all jobs of family F_f are completed no later than their respective deadlines if and only if $(1,f)$ is completed before or at its deadline for all $f \in \{1,\ldots,F\}$. This is equivalent to the processing of all families F_f, $f \in \{1,\ldots,F\}$, being completed no later than $d_{1,f} + T_f - p_{1,f}$. Hence, we can easily construct the feasible GTD-schedule by interchanging batches of S that are not sequenced in their GTD order, according to the tie breaking rules stated above (see also Tanaev et al., 1994, 1998). □

3.2 Notation and Model Formulation

For the sake of notational convenience, we will slightly modify the notation for the remainder of this paper. Define $n_0 := 0$ and denote job (j,f), $f \in \{1,\dots,F\}$, $j \in \{1,\dots,n_f\}$, with the index $n_0 + \dots + n_{f-1} + j$. Then a schedule of an instance of $1 \,|\, d_{j,f}, b = n, SC_{si,b} = s \,|\, TSC$ corresponds to a sequence of the elements of the job set $\{1,\dots,n\}$. Furthermore, we define a parameter $a_{j,f}$ for all $j \in \{1,\dots,n\}$ and all $f \in \{1,\dots,F\}$. $a_{j,f}$ is set to one if job j belongs to family f and to zero otherwise. Finally, the processing time and the deadline of job j, $j \in \{1,\dots,n\}$, are denoted by p_j and d_j, respectively. The jobs are assumed to be ordered in non-decreasing order of their respective deadlines with ties being broken in favor of smaller job indices. Hence, the sequence $(1,\dots,n)$ is an EDD-schedule of a given instance.

Based on the above notation, define the following variables:

$$x_{j,k} := \begin{cases} 1, & \text{if job } j \text{ is scheduled in } k\text{-th} \\ & \text{position of the schedule} \\ 0, & \text{else} \end{cases} \qquad \forall\, j,k \in \{1,\dots,n\},$$

and

$$y_k := \begin{cases} 1, & \text{if the } k\text{-th job of the} \\ & \text{schedule directly} \\ & \text{precedes a setup} \\ 0, & \text{else} \end{cases} \qquad \forall k \in \{1,\dots,n-1\},$$

Then a mathematical model for $1 \,|\, d_{j,f}, b = n, SC_{si,b} = s \,|\, TSC$ is as follows.

$$\min_{\mathbf{x},\mathbf{y}} \quad \sum_{k=1}^{n-1} y_k + 1 \tag{4}$$

$$\text{s.t.} \quad \sum_{k=1}^{n} x_{j,k} = 1 \qquad \forall\, j \in \{1,\dots,n\}, \tag{5}$$

$$\sum_{j=1}^{n} x_{j,k} = 1 \qquad \forall k \in \{1,\dots,n\}, \tag{6}$$

$$\sum_{l=1}^{k} \sum_{j=1}^{n} p_j x_{j,l} \leq \sum_{j=1}^{n} d_j x_{j,k} \qquad \forall k \in \{1,\dots,n\}, \tag{7}$$

$$y_k \geq \sum_{j=1}^{n} x_{j,k} a_{j,f} - \sum_{j=1}^{n} x_{j,k+1} a_{j,f} \qquad \forall k \in \{1,\dots,n-1\},\ f \in \{1,\dots,F\}, \tag{8}$$

$$x_{j,k} \in \{0,1\} \qquad \forall\, j,k \in \{1,\dots,n\}, \tag{9}$$

$$y_k \geq 0 \qquad \forall k \in \{1,\dots,n-1\}. \tag{10}$$

The objective function (4) minimizes the number of batches, which is equivalent to minimizing the total scheduling cost. Conditions (5) and (6) ensure that every job is part of the solution sequence and that at most one job can be processed at a time, respectively. Restrictions (7) guarantee that each job completes no later than its deadline. Constraints (8) enforce a setup operation between the processing of two jobs of different families. Finally, conditions (9) and (10) define the domains of the variables. Note that we need not explicitly restrict the variables y_k, $k \in \{1,\dots,n-1\}$, to be binary because this is implicitly guaranteed by the objective function (4) and restrictions (8).

3.3 Algorithm Description

Let I be an instance of $1 \,|\, d_{j,f}, b = n, SC_{si,b} = s \,|\, TSC$. Furthermore, let $i,k \in \{1,\dots,n\}$, $i \leq k$, and define a related instance I' by only considering jobs $i,\dots,k$ of I. The corresponding processing times remain unchanged, while the deadlines are set to their value in I minus the total processing time of jobs 1 to $i-1$. We denote the resulting problem I' as the (i,k)-*problem* of I. It is clear that I has a feasible schedule if and only if the (i,k)-problem has a feasible schedule for every i and k with $1 \leq i \leq k \leq n$. Moreover, we observe:

Lemma 6 *Let I be an instance of $1 \,|\, d_{j,f}, b = n, SC_{si,b} = s \,|\, TSC$ and consider any (i,k)-problem of I. If the GTD-schedule for the (i,k)-problem is feasible, then the GTD-schedule is feasible for every (l,k)-problem, where $i < l \leq k$.*

Proof It is easy to see that, in order to obtain the feasible GTD-schedule for the (l,k)-problem, one must simply remove jobs $i,\dots,l-1$ from the GTD-schedule for the (i,k)-problem. □

We can now state our approximation algorithm for $1 \,|\, d_{j,f}, b = n, SC_{si,b} = s \,|\, TSC$.

Algorithm 1 returns a schedule σ^0. If there exists no feasible schedule, the algorithm returns $\sigma^0 := \emptyset$. Our main result regarding the algorithm is as follows.

Theorem 2 *Let I be an instance of $1 \,|\, d_{j,f}, b = n, SC_{si,b} = s \,|\, TSC$. Algorithm 1 checks if there exists a feasible schedule for I. If a feasible schedule exists, it determines a schedule that approximates the cost of an optimal schedule by a factor of F.*

Proof In Step 1, Algorithm 1 checks whether the GTD-schedule of the input instance I is feasible. If this is the case, then this schedule is also optimal for I (Lemma 5) and the algorithm terminates by returning this schedule. Similarly, in Step 2, the algorithm makes use of Lemma 3 to check if I has a feasible solution. If the input instance is infeasible, the algorithm returns an empty schedule.

Algorithm 1 Approximation algorithm

0. Initialization: Set $k := n$, $\sigma := \emptyset$, and $\sigma^0 := \emptyset$.

1. GTD-schedule: Determine the GTD-schedule σ^{GTD} of the input instance. If σ^{GTD} is feasible, then it is also optimal (see Lemma 5). In this case, set $\sigma^0 = \sigma^{GTD}$ and stop.

2. EDD-schedule: Determine the EDD-schedule σ^{EDD} of the input instance. If σ^{EDD} is infeasible, then there exists no feasible schedule (see Lemma 3). In this case, stop.

3. Stop criterion: If $k \leq F$, set $\sigma^{max} := (1,\ldots,k)$, $\sigma^0 := (\sigma^{max},\sigma)$ and stop. Else, set $i := k - F$ and $\sigma^{max} := (i+1,\ldots,k)$.

4. $(\mathbf{i},\mathbf{k})$-problem: Determine the GTD-schedule σ^{GTD} for the (i,k)-problem.

Case 1, σ^{GTD} is feasible: Set $\sigma^{max} := \sigma^{GTD}$. If $i = 1$, set $\sigma^0 := (\sigma^{max},\sigma)$ and stop. Else, if $i > 1$, set $i := i - 1$ and repeat Step 4.

Case 2, σ^{GTD} is infeasible: Set $\sigma := (\sigma^{max},\sigma)$, $k := i$, and goto Step 3.

For the remainder of the proof, we will assume that the input instance I is such that Step 3 is executed, i.e. such that the GTD-schedule is infeasible (Step 1) while the EDD-schedule is feasible (Step 2).

In Steps 3 and 4, Algorithm 1 considers the jobs in their reverse EDD order to construct a schedule σ^0 by successively inserting non-empty job sequences σ^{max} at the beginning of a partial schedule σ. These steps can be interpreted as iterations of the algorithm, where each iteration ends with such an insertion operation. Denote the total number of these iterations by m. Then, obviously, $m \leq n$. Furthermore, in order to ease the notation, denote the sequence of jobs added in iteration i, $i \in \{1,\ldots,m\}$, by σ_i^{max} and the index of the first job considered in this iteration by l_i. Define $l_{m+1} := 0$. Then, $l_{m+1} < l_m < \cdots < l_2 < l_1 = n$ and $\sigma^0 = (\sigma_m^{max},\sigma_{m-1}^{max},\ldots,\sigma_1^{max})$, where σ_i^{max}, $i \in \{1,\ldots,m\}$, is a feasible schedule for the $(l_{i+1}+1,l_i)$-problem by construction of Steps 3 and 4 of Algorithm 1. Hence, σ^0 is feasible for I.

Note that the sequence σ_i^{max}, $i \in \{1,\ldots,m\}$, has at most F batches, because it either has at most F jobs (Step 3) or it is a GTD-schedule (Step 4). Therefore, σ^0 has at most mF batches.

By Lemma 2, there exists an optimal schedule σ^{opt} for I with the jobs of each family being processed in their EDD order. It is easy to see that σ^{opt} can be modified such that all jobs of a specific family that share the same deadline are processed in exactly one batch in the order of their indices (smaller index first) and such that the resulting schedule is optimal. The argumentation is in analogy to the proof of Lemma 2. When referring to σ^{opt} in the remainder of this proof, we assume that this modification has been implemented.

Define $\sigma_i^0 := (\sigma_i^{max},\sigma_{i-1}^{max},\ldots,\sigma_1^{max})$, $i = 1,\ldots,m$, and let $f(i)$ be the number of batches of σ^{opt} that exclusively contain jobs that are also included in σ_i^0. Define $f(0) := 0$. In the remainder of the proof we will show that $f(m) > f(m-1) > \cdots > f(1) > f(0)$, so that σ^{opt} has at least m batches. This will prove the claim of the theorem.

Because sequence $\sigma_m^0 = \sigma^0$ contains all n jobs, $f(m)$ is equal to the number of batches in σ^{opt}. Furthermore, $f(m) > f(m-1)$, because at least one batch of σ^{opt} must contain a job of sequence σ_m^{max}, which is not part of σ_{m-1}^0. Additionally, note that an insertion of a sequence into σ^0 in Step 3 can only take place in iteration m, so that we must solely consider Step 4 of Algorithm 1 in the remainder of the proof.

Now, consider iteration i, $1 \leq i \leq m-1$, of Algorithm 1. Construct a feasible schedule σ_i^{opt} for the $(1,l_i)$-problem by modification of σ^{opt} as follows. If $i > 1$, remove all jobs that are included in σ_{i-1}^0 from σ^{opt}. The relative order of the remaining jobs $1,\ldots,l_i$ remains unchanged. Note that the modification is such that $f(i) > f(i-1)$ if the jobs of sequence σ_i^{max} constitute at least one complete batch of σ_i^{opt}.

Let p be the largest index, such that job p is the first job of a batch in σ_i^{opt}. We will make use of two properties:

1. All jobs of the batch that starts with job p in σ_i^{opt} are included in the job set $\{p,p+1,\ldots,l_i\}$. This is immediately implied by the fact that the jobs of each family are processed in non-decreasing order of their deadlines in σ_i^{opt} and the above assumptions regarding σ^{opt}.
2. Among all batches in σ_i^{opt} that include jobs of the set $\{p,p+1,\ldots,l_i\}$, there exists at most one batch of each family. This can be seen by assuming to the contrary that σ_i^{opt} contains at least two of these batches of the same family. Because of the above assumptions regarding σ^{opt}, the jobs of these batches are processed in increasing order of their indices. Hence, the second batch must start with job $j \geq p+1$, which contradicts the method of choice of index p.

Because $d_1 \leq \cdots \leq d_p \leq \cdots \leq d_{l_i}$, we can transform σ_i^{opt} into a feasible schedule $\hat{\sigma}_i^{opt}$ for the $(1,l_i)$-problem by shifting jobs $p,p+1,\ldots,l_i$ to the end of σ_i^{opt} without changing their relative order. According to the second of the above properties, the resulting final sequence of the jobs $p,p+1,\ldots,l_i$ within $\hat{\sigma}_i^{opt}$ is a feasible GT-schedule for the (p,l_i)-problem. Then, because of Lemma 5, the GTD-schedule for the (p,l_i)-problem is feasible. It follows further from Lemma 6, that the GTD-schedule is feasible for every (k,l_i)-problem, with $p \leq k \leq l_i$. Thus, in iteration i of Algorithm 1, Step 4 will be repeated until all jobs of the set $\{p,p+1,\ldots,l_i\}$ are included in the sequence σ_i^{max}. According to the first of the above properties, all jobs of the batch that starts with job p in σ_i^{opt} are included in the job set $\{p,p+1,\ldots,l_i\}$. Thus, $f(m) > f(m-1) > \cdots > f(0)$ and the statement of the theorem is true. □

We close this section with analyzing the time complexity of Algorithm 1 by making use of the notation defined in the proof of Theorem 2. In Step 1, the algorithm needs $O(n+F\log F)$ time to construct the GTD-schedule and check its feasibility. Similarly, constructing and checking the feasibility of the EDD-schedule in Step 2 requires $O(n\log n)$ time. Now, consider iteration i, $1 \le i \le m$, of Algorithm 1, where the algorithm analyzes (k,l_i)-problems, $k = r, r-1, \ldots, \max\{l_{i+1},1\}$, with $r = \max\{l_i - F, 1\}$. For each of these problems, the GTD-schedule is checked for feasibility, which can be done in $O(F)$ time if the corresponding values $d_{1,f} + T_f - p_{1,f}$ are stored in a sorted list for all families. Updating this list requires $O(F)$ time when considering a new job within the algorithm. Thus, Iteration i, $1 \le i \le m$, needs $O((\max\{l_i - F, 1\} - \max\{l_{i+1}, 1\} + 1)F)$ time. Therefore, all iterations of Algorithm 1 will require $O(F\sum_{i=1}^{m}(\max\{l_i - F, 1\} - \max\{l_{i+1}, 1\} + 1) = O(nF)$ time. The running time of Algorithm 1 therefore sums up to $O(n\log n + nF)$.

4 Computational Experiments

We ran computational tests to analyze the effectiveness of Algorithm 1. These tests were performed on a computer with 16GB of RAM and an Intel Core i7-4770 CPU, running at 3.4GHz under Windows 8, 64bit. The algorithm was implemented in C++ (Microsoft Visual Studio 2010). We additionally used IBM ILOG CPLEX in version 12.5 with 64bit to solve model (4)–(10).

Our testbed is composed of 10 groups of instances. For each group, the number of families of the corresponding instances is fixed, $F = 5, 10, 15, \ldots, 50$. Each group features 20 randomly generated instances. The number of jobs in family F_f, $f \in \{1, \ldots, F\}$, is set to $n_f = 2 + \lfloor f/2 \rfloor$. Integer job processing times p_j, $j = 1, \ldots, n$, were drawn from a uniform distribution over the interval $[1, 100]$. Similarly, integer deadlines d_j, $j = 1, \ldots, n$, were drawn from uniform distributions over $[\sum_{k=1}^{j} p_k, \sum_{k=1}^{j} p_k + 10n]$. Hence, each instance is feasible. Furthermore, the deadlines are such that it is unlikely that the GTD-schedule of an instance is feasible. Indeed, our testbed contains no instance with a feasible GTD-schedule. The jobs were randomly assigned to the families.

Table 2 presents the computational results on the quality of the solutions determined by Algorithm 1. For each set of instances, it presents the percentage of instances for which CPLEX returned a feasible solution after a time limit of 2 hours (third column) and the corresponding percentage of instances that CPLEX was able to solve to optimality (fourth column). Furthermore, in order to asses the solution quality of Algorithm 1, the table includes results for three ratios in columns five to seven. Each cell presents the average value as well as the maximum value (in parentheses) of these ratios over all relevant instances of a set. $qual_F$ is defined as the objective function value of a solution determined by the approximation algorithm divided by the number of families F, the latter number being a lower bound on the number of batches in an optimal solution. Similarly, the denominator of $qual_{opt}$ ($qual_{sol}$) corresponds the optimal (best) objective function value determined by CPLEX (within the time limit of two hours). The average and maximum values in the last two columns of the table solely relate to the instances for which optimal or feasible solutions have been determined, respectively.

Table 2 Computational results - solution quality

Inst. set		CPLEX		Algorithm 1		
F	n	sol [%]	opt [%]	$qual_F$	$qual_{opt}$	$qual_{sol}$
5	16	100	100	2.26 (3)	1.47 (1.86)	1.47 (1.86)
10	45	100	0	2.67 (3.4)	-	1.12 (1.68)
15	86	100	0	3.09 (3.47)	-	0.97 (1.12)
20	140	100	0	3.31 (3.7)	-	0.5 (0.56)
25	206	80	0	3.5 (4)	-	0.44 (0.48)
30	285	15	0	3.86 (4.3)	-	0.41 (0.44)
35	376	0	0	4.01 (4.34)	-	-
40	480	0	0	4.24 (4.63)	-	-
45	596	0	0	4.42 (4.98)	-	-
50	725	0	0	4.66 (5.34)	-	-

Table 3 Computational results - runtime in milliseconds

F	5	10	15	20	25	30	35	40	45	50
t_{avg}	0.01	0.03	0.08	0.19	0.38	0.73	1.2	4.98	30.65	32.61
t_{max}	0.01	0.07	0.14	0.26	0.48	1.16	1.46	16.32	101.01	56.55

The results of Table 2 indicate that Algorithm 1, on average, results in significantly better solutions than indicated by Theorem 2. Furthermore, it clearly outperforms CPLEX for problem instances of realistic size.

Table 3 complements the above results by presenting the average (t_{avg}) and maximum (t_{max}) computational times of Algorithm 1 for each set of instances in milliseconds. It shows that Algorithm 1 terminates after only a few milliseconds, even for the large instances of our testbed.

5 Summary

In this article we have addresses the single-machine batch scheduling problem with the objective of minimizing the total setup cost. We have presented a proof for the problem's strong NP-hardness and introduced an approximation algorithm that approximates the cost of an optimal schedule by a factor of F, where F denotes the number of families. Computational tests on randomly generated instances have shown that the algorithm, on average, results in significantly better solutions than indicated by this relative performance

guarantee. Furthermore, the runtimes of the algorithm are in the range of only a few milliseconds for small to medium sized instances.

Acknowledgements Maksim Barketau is partially supported by the $\Phi 10\Phi\Pi - 001$ project of the Belorussian Fund of Fundamental Research.

References

Allahverdi A (2015) The third comprehensive survey on scheduling problems with setup times/costs. European Journal of Operational Research 246(2):345–378

Allahverdi A, Gupta JND, Aldowaisan T (1999) A review of scheduling research involving setup considerations. Omega 27(2):219–239

Allahverdi A, Ng CT, Cheng TCE, Kovalyov MY (2008) A survey of scheduling problems with setup times or costs. European Journal of Operational Research 187(3):985–1032

Baker KR, Trietsch D (2009) Principles of Sequencing and Scheduling. Wiley, Hoboken, New Jersey

Błażewicz J, Ecker KH, Pesch E, Schmidt G, Węglarz J (2007) Handbook on Scheduling: From Theory to Applications. Springer, Berlin

Bruno J, Downey P (1978) Complexity of task sequencing with deadlines, set-up times and changeover costs. SIAM Journal on Computing 7(4):393–404

Cheng TCE, Ng CT, Yuan JJ (2003) The single machine batching problem with family setup times to minimize maximum lateness is strongly NP-hard. Journal of Scheduling 6(5):483–490

Garey MR, Johnson DS (1979) Computers and Intractability - A Guide to the Theory of NP-Completeness. Freeman, New York

Gerodimos AE, Glass CA, Potts CN, Tautenhahn T (1999) Scheduling multi-operation jobs on a single machine. Annals of Operations Research 92:87–105

Graham RL, Lawler EL, Lenstra JK, Rinnooy Kan AHG (1979) Optimization and approximation in deterministic sequencing and scheduling: a survey. Annals of Discrete Mathematics 5:287–326

Herrmann JW, Lee CY (1995) Solving a class scheduling problem with a genetic algorithm. ORSA Journal on Computing 7(4):443–452

Lu LF, Yuan JJ (2007) The single machine batching problem with identical family setup times to minimize maximum lateness is strongly NP-hard. European Journal of Operational Research 177(2):1302–1309

Mehta SV, Uzsoy R (1998) Minimizing total tardiness on a batch processing machine with incompatible job families. IIE Transactions 30(2):165–178

Monma CL, Potts CN (1989) On the complexity of scheduling with batch setup times. Operations Research 37(5):798–804

Potts CN, Kovalyov MY (2000) Scheduling with batching: a review. European Journal of Operational Research 120(2):228–249

Tanaev VS, Gordon VS, Shafransky Y (1994) Scheduling Theory. Single-Stage Systems. Springer-Science & Business Media, B.V., Dordrecht

Tanaev VS, Kovalyov MY, Shafransky YM (1998) Scheduling Theory. Group Technologies. Institute of Engineering Cybernetics, National Academy of Sciences of Belarus, Minsk, (in Russian)