The complexity of one-machine batching problems

Susanne Albers and Peter Brucker*

FB 6 Mathematik, Universität Osnabrück, D 49069 Osnabrück, Germany

Received 4 December 1990
Revised 7 July 1991

Abstract

Batching problems are combinations of sequencing and partitioning problems. For each job sequence JS there is a partition of JS into batches with optimal value opt(JS) and one has to find a job sequence which minimizes this optimal value. We show that in many situations opt(JS) is the solution of a shortest path problem in some network. An algorithm solving this special shortest path problem in linear time with respect to the number of vertices is presented. Using this algorithm some new batching results are derived. Furthermore, it is shown that most of the batching problems which are known to be polynomially solvable turn into NP-hard problems when modified slightly.

Keywords. Batching, polynomial algorithm, NP-hard, shortest path problem.

1. Introduction

Modern technologies in flexible manufacturing lead to new types of scheduling problems. While traditional scheduling methods are primarily concerned with sequencing of jobs, modern manufacturing environments provide the additional possibility to process jobs in batches. Batching problems that arise in these settings present a new and fruitful research direction and their solution enhances the ability to manage manufacturing operations efficiently. The results lead to improvement of resource usage and customer satisfaction which are important objectives in manufacturing.

A one-machine batching problem can be formulated as follows. Given are $n$ jobs $J_i (i = 1, \ldots, n)$ with processing times $p_i (i = 1, \ldots, n)$ which are to be processed on one machine. There may be given precedence relations between the jobs. Jobs are processed in so-called batches. A batch is a set of jobs which are processed jointly. The

* Correspondence to: Professor P. Brucker, FB 6 Mathematik, Universität Osnabrück, D 49069 Osnabrück, Germany.
* This work was partially supported by the Deutsche Forschungsgemeinschaft.

0166-218X/93/$06.00 © 1993—Elsevier Science Publishers B.V. All rights reserved
number of jobs in a batch is called the batch size. Batch sizes are numbers between 1 and \( n \). The formation of batches implies batch availability which means that all jobs in a batch are not available until the last job of the batch is finished. Thus, the flow time \( f_i \) of a job coincides with the completion time of the last scheduled job in its batch and all jobs in this batch have the same flow time.

The production of a batch requires a machine set-up \( S \) of \( s \geq 0 \) time units. The machine set-ups are both sequence independent and batch independent, i.e., they depend neither on the sequence of batches nor on the number of jobs in a batch.

The one-machine batching problem is to find a sequence of jobs and a collection of batches that partitions this sequence of jobs such that the objective function

\[
F = \sum \alpha_i f_i,
\]

where \( \alpha_i \geq 0 \) are given job weights, is minimized. The job weights are induced by jobs having different priorities. We also consider the special case in which all \( \alpha_i \) equal 1, such that our objective function is reduced to \( F = \sum f_i \). We choose the function \( F = \sum \alpha_i f_i \) because it is very important in practical applications. It has a positive effect on manufacturing lead times, on the minimization of work-in-process inventories and safety stocks and on the meeting of due dates.

Figure 1 shows a batching problem with six jobs. The jobs are scheduled in order of increasing job indices. The first batch contains \( J_1, J_2 \) and \( J_3 \) and \( f_1 = f_2 = f_3 = 5 \). The second batch comprises \( J_4 \) and \( J_5 \) and \( f_4 = f_5 = 11 \). The third batch just contains \( J_6 \). \( F = \sum f_i \) assumed, we have

\[
F = 3 \times 5 + 2 \times 11 + 1 \times 16 = 53.
\]

A schedule of a given batching problem may be described by a job sequence and a batch size sequence. The job sequence is the order in which the jobs are scheduled. In our example we have the job sequence \( JS = J_1 J_2 J_3 J_4 J_5 J_6 \). The batch size sequence is a sequence of positive integers \( n_1, n_2, \ldots, n_k \) with \( \sum_{j=1}^{k} n_j = n \), where \( n_j \) is the batch size of the \( j \)th batch. Although a solution of a batching problem may be given by batches and a sequence of the batches we will define a solution more specifically by a job sequence combined with a batch size sequence.

Our work continues earlier research on one-machine batching presented in [3, 5]. So far, only batching problems with \( \sum f_i \) objective function have been investigated. For problems with \( p_i = p \) for \( i = 1, \ldots, n \) an \( O(\sqrt{n}) \)-algorithm was given by Coffman,
Nozari and Yannakakis [2]. Furthermore, it was shown by Coffman et al. [3] that if the processing times are arbitrary then the same problem can be solved in polynomial time.

The purpose of this paper is to explore a number of additional batching problems and classify them as polynomially solvable or NP-hard by presenting efficient algorithms or NP-hardness proofs. In order to identify the specific problems easily, we introduce a classification scheme which is similar to the scheme used for scheduling problems. Each batching problem is described by a string of the form

\[ \alpha|\beta|\gamma. \]

- \( \alpha \) denotes the number of jobs in the problem. If \( \alpha = \cdot \) then an arbitrary value is assumed.
- \( \beta \) is a string of the form \( \beta = \beta_1 \beta_2 \) and describes restrictions that are given on the set of jobs. \( \beta_1 \) specifies precedence constraints. \( \beta_1 = \cdot \) indicates that there are given independent jobs which may be scheduled in any order. \( \beta_1 = \text{Chains} \) defines a problem in which precedence constraints form parallel chains. Furthermore, \( \beta_1 = \text{Prec} \) denotes precedence relations of an arbitrary structure. \( \beta_2 \) specifies additional restrictions, e.g. \( \cdot p = p_1 \) means that all jobs have the same processing time.
- \( \gamma \in \{ \sum f_i, \sum \alpha_i f_i \} \) denotes the objective function.

Table 1 gives a survey of the complexity results presented in this paper and shows the borderline between “easy” and “hard” batching problems. For the polynomially solvable cases the table mentions the complexity of the best known algorithm. If the algorithm is developed in an other paper we give an appropriate reference.

This paper is organized as follows. In Section 2 we examine polynomially solvable batching problems and show that the problem of finding an optimal batch size sequence for a fixed job sequence can be reduced to a special shortest path problem. We develop an algorithm that solves the shortest path problem in \( O(n) \) time, where \( n \) is the number of jobs in the given job sequence. The latter result is of interest in its own right because it shows that under certain conditions shortest path problems in networks with \( O(n^2) \) arcs can be solved in linear time.

In Section 3 we devote ourselves to NP-hard batching problems and present a number of NP-hardness proofs.
2. Polynomially solvable batching problems

In this section we develop an algorithm that solves the batch sizing problem for any batching problem with a given job sequence in $O(n)$ time, where $n$ is the number of jobs in the problem. We show how this algorithm can be applied to solve a number of batching problems in polynomial time.

2.1. Reduction of the batch sizing problem to a special shortest path problem

Consider a fixed but arbitrary job sequence $JS = J_1 J_2 J_3 \ldots J_n$ of a given batching problem. We want to solve the batch sizing problem, i.e., we want to find an optimal sequence of batch sizes with respect to the general objective function $F = \sum \alpha_i f_i$.

Any solution is of the form

$$BS = S J_{i_1} \ldots J_{i_2 - 1}SJ_{i_2} \ldots J_{i_3 - 1}SJ_{i_3} \ldots J_{i_k - 1}SJ_{i_k} \ldots J_n,$$

where $K$ is the number of batches in $BS$ and $i_j$ is the index of the first job in the $j$th batch. Obviously, $1 = i_1 < i_2 < i_3 < \cdots < i_k \leq n$.

We now calculate the $\sum \alpha_i f_i$ value $F(BS)$ for $BS$. The processing time of the $j$th batch equals

$$P_j = s + \sum_{v = i_j}^{i_{j+1} - 1} p_v.$$

Thus,

$$F(BS) = \sum_{i = 1}^{n} \alpha_i f_i$$

$$= \sum_{j = 1}^{K} \left( \sum_{v = i_j}^{n} \alpha_v \right) P_j$$

$$= \sum_{j = 1}^{K} \left( \sum_{v = i_j}^{n} \alpha_v \right) \left( s + \sum_{v = i_j}^{i_{j+1} - 1} p_v \right).$$

In order to solve the batch sizing problem, we obviously have to find a constant $K$ and a sequence of indices $1 = i_1 < i_2 < i_3 < \cdots < i_K \leq n$ such that the above objective function value is minimized. This problem can be reduced to a shortest path problem.

Every solution $BS$ corresponds to a path of the form

$$BS = S J_{i_1} \ldots J_{i_2 - 1}SJ_{i_2} \ldots J_{i_3 - 1}SJ_{i_3} \ldots J_{i_k - 1}SJ_{i_k} \ldots J_nJ_{n+1}.$$  

Here $J_{n+1}$ is a "dummy" job. The edge length $c_{ij}$ of edge $(J_i, J_j)$ is set to

$$c_{ij} = \left( \sum_{v = i_j}^{n} \alpha_v \right) \left( s + \sum_{v = i_j}^{j-1} p_v \right).$$

$c_{ij}$ are the "costs" produced by a batch containing $J_i, J_{i+1}, \ldots, J_{j-1}$. 
The \( c_{ij} \) have a very interesting property. Let \( i < j < k \), then

\[
c_{ik} - c_{ij} = \left( \sum_{\nu=i}^{n} x_{\nu} \right) \left( \sum_{\nu=j}^{k-1} p_{\nu} \right)
= f(i)h(j,k),
\]

where \( f(i) \) and \( h(j,k) \) are set to \( \sum_{\nu=i}^{n} x_{\nu} \) and \( \sum_{\nu=j}^{k-1} p_{\nu} \), respectively. We note that \( f(i) \) is monotone nonincreasing and \( h(j,k) > 0 \) for any \( 1 \leq j < k \leq n + 1 \).

These considerations lead to the problem of finding a shortest path from vertex 1 to vertex \( n + 1 \) in a network \( N = (V, E, C) \) with the following properties:

1. \( V = \{1, 2, \ldots, n + 1\} \) is the set of vertices.
2. An edge \((i, j)\) is in \( E \) if and only if \( i < j \).
3. The edge lengths \( C = (c_{ij}) \) satisfy

\[
c_{ik} - c_{ij} = f(i)h(j,k)
\]

for all \( i < j < k \), where \( f(i) \) is monotone nonincreasing and \( h(j,k) > 0 \) for all \( j < k \).

In the next subsection we will show that this problem can be solved in \( O(n) \) time, if each of the values \( f(i) \) and \( h(j,k) \), \( i < k \), can be computed in constant time. The corresponding algorithm is based on an algorithm presented by Coffman et al. [3].

### 2.2. An efficient shortest path algorithm

The algorithm which solves the special shortest path problem introduced in Subsection 2.1 is a strong generalization of an algorithm presented by Coffman et al. [3] and it is based on a dynamic programming approach. Its complexity is reduced to \( O(n) \) due to the special properties of \( C \).

First some notation. Let

- \( F_j \) be the length of a shortest path from \( j \) to \( n + 1 \),
- \( F_j(k) \) be the length of a shortest path from \( j \) to \( n + 1 \) which contains \((j,k)\) as first edge.

Then

\[
F_j(k) = c_{jk} + F_k
\]

and

\[
F_j = \min\{F_j(k) \mid j < k \leq n + 1\}.
\]

We develop a condition under which

\[
F_j(k) \leq F_j(l)
\]

for three given vertices \( j < k < l \) holds. Simple calculations yield that the last inequality is equivalent to

\[
F_j(k) - F_j(l) = c_{jk} + F_k - c_{jl} - F_l \leq 0
\]
and
\[ f(j) \geq \frac{F_k - F_i}{h(k,l)}. \]

For any two vertices \( k < l \) we define
\[ \delta(k,l) := \frac{F_k - F_i}{h(k,l)} \]
and call \( \delta(k,l) \) the threshold of \( k \) and \( l \). If \( j \) \((j < k < l)\) is a vertex with \( f(j) \geq \delta(k,l) \) then \( F_j(k) \leq F_j(l) \) and \( l \) is called not better than \( k \) (with respect to \( j \)). On the other hand, if \( f(j) < \delta(k,l) \), then \( F_j(k) > F_j(l) \) and \( l \) is called better than \( k \) (with respect to \( j \)).

**Lemma 2.1.** Let \( 1 \leq j < k < l \) be vertices satisfying \( f(j) \geq \delta(k,l) \). Then
\[ F_i(k) \leq F_i(l) \]
for all \( i = 1, \ldots, j \).

**Proof.** Since the function \( f \) is monotone nonincreasing, \( f(i) \geq f(j) \) for all \( i = 1, \ldots, j \). Thus
\[ f(i) \geq \delta(k,l) \]
and
\[ F_i(k) \leq F_i(l) \]
for all \( i = 1, \ldots, j \). \( \square \)

**Lemma 2.2.** Suppose \( \delta(j,k) \leq \delta(k,l) \) for some vertices \( 1 \leq j < k < l \leq n + 1 \). Then, for each vertex \( i, 1 \leq i \leq j \), either \( k \) is not better than \( j \) or \( l \) is better than \( k \) with respect to \( i \).

**Proof.** Let \( i \) be a vertex with \( 1 \leq i \leq j \). If \( f(i) \geq \delta(j,k) \), then \( k \) is not better than \( j \) with respect to \( i \). On the other hand, if \( f(i) < \delta(j,k) \), then \( f(i) < \delta(k,l) \) and \( l \) is better than \( k \) with respect to \( i \). \( \square \)

These two lemmas are very important for our shortest path algorithm. Given a vertex \( j \) and two vertices \( k, l \) satisfying the required properties each, Lemmas 2.1 and 2.2 ensure that for every \( i \leq j \) the edges \((i,l)\) and \((i,k)\) respectively need not be considered in search of a shortest path from \( i \) to \( n + 1 \).

We now explain the shortest path algorithm which is described in detail in Fig. 3. In order to determine a shortest path from vertex 1 to vertex \( n + 1 \), we compute shortest paths from all vertices \( j \) to \( n + 1 \). Starting with vertex \( n \) we scan the vertices in order of decreasing index. When a shortest path from \( j \) to \( n + 1 \) has to be computed, shortest paths from \( j + 1, j + 2, \ldots, n \) to \( n + 1 \) are already known and therefore, it suffices to determine an immediate successor \( k > j \) of vertex \( j \) in a shortest path from
Algorithm Shortest Path
begin
1. \( Q := \{n + 1\} \); 
2. for \( j := n \) downto 1 do begin 
3. \( \text{while } \text{HEAD}(Q) \neq \text{TAIL}(Q) \text{ and } f(j) \geq \delta(\text{NEXT}(\text{HEAD}(Q)), \text{HEAD}(Q)) \text{ do} \) 
   \( \text{DELETE}(\text{HEAD}(Q)) \); 
4. \( N(j) := \text{HEAD}(Q) \) 
5. \( \text{while } \text{HEAD}(Q) \neq \text{TAIL}(Q) \text{ and } \delta(j, TAIL(Q)) \leq \delta(TAIL(Q), \text{PREVIOUS}(TAIL(Q))) \text{ do} \) 
   \( \text{DELETE}(\text{TAIL}(Q)) \); 
6. \( \text{ADD}(j, Q) \); 
end 
end 

Fig. 3. The shortest path algorithm.

\( j \) to \( n + 1 \). Such a vertex \( k \) is computed with the help of a queue \( Q \) (see also Fig. 2) which contains all candidates for an optimal \( k \).

Initially, \( Q \) stores only vertex \( n + 1 \). During the course of the algorithm vertices are deleted from the head and from the tail of the queue and are appended only to the tail of the queue. Thus, assuming that \( Q \) is of the form

\[
\begin{array}{c|c|c|c|c|c}
 & i_t & i_{t-1} & \cdots & i_2 & i_1 \\
\hline
\text{TAIL} & & & & & \\
\text{HEAD} & & & & & \\
\end{array}
\]

the invariant

\[
i_t < i_{t-1} < \cdots < i_2 < i_1
\]

holds. Furthermore, we will ensure that also the invariant

\[
\delta(i_t, i_{t-1}) \geq \delta(i_{t-1}, i_{t-2}) > \cdots > \delta(i_2, i_1)
\]

is always satisfied.

Let \( 1, 2, \ldots, j - 1 \) be vertices not scanned so far and let \( Q \) be of the structure described above. We show how an immediate successor \( k > j \) can be determined.

First, vertices are deleted from the head of \( Q \): If \( f(j) \geq \delta(i_2, i_1) \), then (by Lemma 2.1) we have \( F_i(i_2) \leq F_i(i_1) \) for all \( i \leq j \), and \( i_1 \) can be deleted from \( Q \). We continue to delete the head from \( Q \) until finally

\[
\delta(i_{r+1}, i_r) > f(j)
\]

for some \( r \leq t - 1 \), which implies (because of invariant (2))

\[
F_j(i_{v+1}) > F_j(i_v) \quad \text{for } v = r, \ldots, t - 1,
\]

or \( r = t \) and \( Q \) contains only \( i_t \). Thus, among all vertices stored in \( Q \), \( i_r \) is the best vertex for an immediate successor \( k > j \) of \( j \) and is stored in \( N(j) \).

Then, some operations are performed at the tail of \( Q \): If \( \delta(j, i_r) \leq \delta(i_r, i_{r-1}) \) then (by Lemma 2.2) we can eliminate \( i_r \) from \( Q \). We continue to delete the tail of \( Q \) until

\[
\delta(j, i_r) > \delta(i_r, i_{r-1})
\]

for some \( r > 1 \) or \( Q \) contains only \( i_1 \). Then \( j \) is appended to \( Q \). Obviously, since \( Q \) initially contains only \( n + 1 \), the deletions at the tail guarantee that the invariant (2) is always satisfied.

We remark that due to invariant (1) the computation of thresholds is always defined.

It is easy to see that the actual shortest path from 1 to \( n + 1 \) can be recovered using the array \( N(j) \). We note that we have to initialize \( F_{n+1} := 0 \) at the beginning of the algorithm.

We now investigate the complexity of the algorithm. Each vertex \( 1 \leq i \leq n \) is added and deleted at most once from \( Q \). Therefore, the operations in steps 3–6 of the algorithm are performed at most \( n \) times, and we conclude that the algorithm runs in \( O(n) \) time, if we assume that each of the values \( f(j), h(j, k) \) \((j < k)\) and thus each threshold \( \delta(j, k) \) can be computed in constant time.

If we apply our algorithm to the batch sizing problem, we have to do some preprocessing in order to obtain \( O(n) \) time complexity. We recall that

\[
c_{ik} - c_{ij} = \left( \sum_{v = i}^{n} x_v \right) \left( \sum_{v = j}^{k-1} p_v \right)
\]

\[
= f(i) h(j, k)
\]

for \( i < j < k \). Define

\[
sp(i) := \begin{cases} 
0, & i = 1, \\
\sum_{v=1}^{i-1} p_v, & i = 2, \ldots, n + 1.
\end{cases}
\]

Obviously, the values \( f(i) \) and \( sp(i) \), \( i = 1, \ldots, n \), can be computed in \( O(n) \) time preceding the algorithm. The values \( sp(i) \) guarantee the computation of \( h(j, k) \) and \( \delta(j, k) \) \((j < k)\) in \( O(1) \) time.

2.3. Polynomialsly solvable cases

We explain how the shortest path algorithm can be used to solve batching problems in polynomial time.
Lemma 2.3. For every $\cdot |\cdot | \sum f_i$-problem there exists an optimal solution in which the jobs are scheduled in nondecreasing order of processing times.

Proof. See Coffman et al. [3].

Lemma 2.4. For every $\cdot |p_i = p| \sum \alpha_i f_i$-problem there exists an optimal solution in which the jobs are scheduled in nonincreasing order of job weights $\alpha_i$.

Proof. Consider an optimal solution and let $J_i$ and $J_k$ be two jobs with $\alpha_i < \alpha_k$, where $J_i$ is scheduled preceding $J_k$. Interchanging $J_i$ and $J_k$ does not increase the value of the objective function and iterating such interchanges leads to an optimal solution with the desired property.

Solving the $\cdot |\cdot | \sum f_i$- and the $\cdot |p_i = p| \sum \alpha_i f_i$-problem is a trivial matter now. First, we have to schedule the jobs in order of nondecreasing processing times and non-increasing job weights, respectively. Application of our shortest path algorithm yields optimal solutions in $O(n \log n)$ time.

The last polynomially solvable problem we mention is the $\cdot |\text{Prec}, p_i = p| \sum f_i$-problem. In this problem all jobs are identical. Thus, all feasible sequences yield the same objective value. To solve the problem we construct a topological ordering of the jobs and apply the shortest path algorithm. The overall complexity is $O(n^2)$.

3. NP-hardness proofs

In this section we prove NP-hardness for three batching problems. In this connection batching problems are formulated as decision problems, i.e., given a threshold value $\bar{F}$ we have to find a solution for which the objective value is bounded from above by $\bar{F}$.

3.1. The $\cdot |\cdot | \sum \alpha_i f_i$-problem

We show that the $\cdot |\cdot | \sum \alpha_i f_i$-problem is NP-hard in the strong sense by presenting a reduction from the 3-PARTITION problem which can be formulated as follows:

There are given $3m$ positive integers $a_1, a_2, \ldots, a_{3m}$ and a positive integral bound $B$. The integers $a_i$, $i = 1, \ldots, 3m$, satisfy the conditions

$$\frac{B}{4} < a_i < \frac{B}{2}$$

and

$$\sum_{i=1}^{3m} a_i = mB.$$
Can \( I = \{1, 2, \ldots, 3m\} \) be partitioned into \( m \) disjoint sets \( I_1, I_2, \ldots, I_m \), such that for \( j = 1, \ldots, m \)

\[
\sum_{i \in I_j} a_i = B?
\]

(Notice that the conditions \( B/4 < a_i < B/2 \) imply that every set \( I_j \) with \( \sum_{i \in I_j} a_i = B \) must contain exactly three elements.)

The \( \cdot | \cdot | \sum \alpha_i f_i \cdot \)-problem obviously lies in the class NP. Given an arbitrary instance of 3-PARTITION we define a \( \cdot | \cdot | \sum \alpha_i f_i \cdot \)-problem \( P \) as follows:

1. For every positive integer \( a_i, i = 1, \ldots, 3m \), define a partition job \( J_i \) with

\[
p_i = a_i,
\]

\[
\alpha_i = p_i.
\]

2. For \( i = 1, \ldots, m \) define a "dummy" job \( J_{3m+i} \) with

\[
p_{3m+i} = (m + 1 - i)2B + B,
\]

\[
\alpha_{3m+i} = p_{3m+i}.
\]

3. Define a "dummy" job \( J_{4m+1} \) with

\[
p_{4m+1} = 2B,
\]

\[
\alpha_{4m+1} = p_{4m+1}.
\]

4. Set the set-up time to \( s = 2B \).

5. The threshold value \( \bar{F} \) is given by the optimal objective function value of the

\[
B(m + 1)(m + 2)|p_i = 1| \sum f_i \cdot -problem
\]

with \( s = 2B \).

We shall show that \( P \) has a solution BS with \( F(\text{BS}) \leq \bar{F} \) if and only if the 3-PARTITION problem has a solution. In this reduction the dummy jobs will play a very important role. If 3-PARTITION has a solution then the dummy jobs will imply that for each index \( 1 \leq j \leq m \) all partition jobs \( J_i \) with \( i \in I_j \) must be scheduled together with one dummy job in the same batch.

We need a few preliminaries.

Consider an arbitrary solution BS of the problem \( P \). Let \( K \) be the number of batches in BS and let \( M_j \) be the set of job indices in batch \( j \), i.e.,

\[
M_j := \{ |J_i \text{ is scheduled in batch } j \}, \quad j = 1, \ldots, K.
\]

We have

\[
\sum_{i \in M_j} \alpha_i = \sum_{i \in M_j} p_i, \]

\[
\sum_{j=1}^{K} \left| \sum_{i \in M_j} \alpha_i \right| = \sum_{j=1}^{K} \left| \sum_{i \in M_j} p_i \right| = B(m + 1)(m + 2).
\]
and these sums are positive integers for all $j = 1, \ldots, K$. Thus, BS can be regarded as a solution in which

$$
n = \sum_{j=1}^{K} \sum_{l \in M_j} \alpha_l = \sum_{j=1}^{K} \sum_{l \in M_j} p_l = B(m + 1)(m + 2)
$$

jobs with processing times $p = 1$ and weights equal to one are scheduled and in which the $j$th batch contains $\sum_{l \in M_j} \alpha_l = \sum_{l \in M_j} p_l$ jobs, $j = 1, \ldots, K$.

**Lemma 3.1.** Let $P'$ be a $B(m + 1)(m + 2) | p_i = 1 | \sum f_i$-problem with set-up time $s = 2B$. Then $P'$ has a unique optimal solution $BS_K$ with $K = m + 1$ batches and batch sizes

$$
n_j = (m + 2 - j)2B, \quad j = 1, \ldots, K.
$$

**Proof.** According to Proposition 2 in Dobson et al. [4], the solution $BS_K$ is an optimal solution of $P'$. We must prove that $BS_K$ is the only optimal solution.

Let $BS_L \neq BS_K$ be another optimal solution of $P'$, where $L$ is the number of batches in $BS_L$. Let $m_j, j = 1, \ldots, L$, be the corresponding batch sizes in $BS_L$.

In case $L > K$ there must exist an index $1 \leq j \leq K$ with $n_j > m_j$. Let $BS'_{K}$ be the solution obtained from $BS_K$ if one job in batch $j$ is scheduled in a new batch $K + 1$ at the end of $BS_K$. Due to the optimality of $BS_K$,

$$
F(BS'_{K}) - F(BS_K) = (K + 1 - j)2B - (n_j - 1) \geq 0. \quad (3)
$$

Furthermore, let $BS'_{L}$ be the solution obtained from $BS_L$ if one job in the $L$th batch is scheduled in batch $j$. Again, due to the optimality of $BS_L$ we have

$$
F(BS'_{L}) - F(BS_L) = -(L - j)2B + m_j - (m_L - 1) \geq 0. \quad (4)
$$

Adding these two inequalities we get

$$(K + 1 - L)2B + (m_j + 2 - m_L - n_j) \geq 0.
$$

We assumed $L > K$ and thus have $K + 1 - L \leq 0$. Since $n_j > m_j$ and $m_L \geq 1$, the inequality $m_j + 2 - m_L - n_j = m_j - m_L + 1 - n_j + 1 \leq 0$ must hold. Hence, $K + 1 = L$ and $m_j - (m_L - 1) = n_j - 1$. The substitution of these terms in the inequalities (3) and (4) yields

$$
(K + 1 - j)2B - (n_j - 1) \geq 0
$$

and

$$
-(K + 1 - j)2B + (n_j - 1) \geq 0.
$$

Hence,

$$
(K + 1 - j)2B - (n_j - 1) = 0
$$

and

$$
(K + 1 - j)2B = n_j - 1.
$$
Substituting the original values for \( K \) and \( n_j \) we obtain by

\[
(m + 2 - j)2B = (m + 2 - j)2B - 1,
\]
a contradiction.

If, on the other hand, \( L \leq K \) then there exists an index \( 1 \leq j \leq L \) with \( n_j < m_j \). If \( L = K \), then there also must be an index \( 1 \leq k \leq L \) with \( n_k > m_k \). In case \( L < K \) we set \( k = L + 1 \) and \( m_k = 0 \) for the remainder of this proof.

Let \( BS'_k \) be the solution obtained from \( BS_k \) if one job in batch \( k \) is scheduled in batch \( j \). Furthermore, let \( BS'_{L} \) be the solution obtained from \( BS_L \) if one job in the \( j \)th batch is scheduled in batch \( k \).

If \( j < k \), then we have

\[
F(BS'_k) - F(BS_K) = -(k - j)2B + n_j - (n_k - 1) \geq 0
\]
and

\[
F(BS'_{L}) - F(BS_L) = (k - j)2B - (m_j - 1) + m_k \geq 0.
\]

Adding the two inequalities we obtain \( n_j + 1 - m_j + m_k + 1 - n_k \geq 0 \). According to our assumption \( n_j + 1 - m_j \leq 0 \) and \( m_k + 1 - n_k \leq 0 \). Thus, the equation \( n_j - (n_k - 1) = (m_j - 1) - m_k \) must be satisfied. Hence,

\[
-(k - j)2B + n_j - (n_k - 1) \geq 0
\]
and

\[
(k - j)2B - n_j + (n_k - 1) \geq 0.
\]

We conclude \( 0 = (k - j)2B + (n_k - n_j) - 1 = (k - j)2B + (j - k)2B - 1 = -1 \) and obtain a contradiction.

If \( k < j \) then

\[
F(BS'_k) - F(BS_K) = (j - k)2B - (n_k - 1) + n_j \geq 0
\]
and

\[
F(BS'_{L}) - F(BS_L) = -(j - k)2B + m_k - (m_j - 1) \geq 0.
\]

Addition of the two inequalities yields \( m_k - (m_j - 1) = (n_k - 1) - n_j \). Applying this equation to the second inequality we conclude

\[
(j - k)2B - (n_k - 1) + n_j = 0.
\]

In this equation we replace \( n_k \) and \( n_j \) by the original values and obtain

\[
0 = (j - k)2B - n_k + n_j + 1 = (j - k)2B + (k - j)2B + 1 = 1.
\]

This contradiction completes the proof that \( BS_K \) is the only optimal solution of \( P' \). \( \square \)
\textbf{Theorem 3.2.} The problem $P$ has a solution BS with $F(\text{BS}) \leq \bar{F}$ if and only if 3-PARTITION has a solution.

\textbf{Proof.} Let BS be an arbitrary solution of $P$ and let $K$ be the number of batches in BS. Again, we define

\[ M_j = \{ l | J_l \text{ is scheduled in batch } j \}, \quad j = 1, \ldots, K. \]

As explained above, BS can be considered as a solution in which there are scheduled $n = B(m + 1)(m + 2)$ jobs with processing times $p_i = 1$ and job weights $\alpha_i = 1$, $i = 1, \ldots, n$. Due to Lemma 3.1, $P$ has a solution BS with $F(\text{BS}) \leq \bar{F}$ if and only if it is possible to schedule the partition jobs and the dummy jobs such that the resulting solution consists of exactly $m + 1$ batches and such that the equations

\[ \sum_{l \in M_j} p_i = (m + 2 - j)2B \]

are satisfied for $j = 1, \ldots, m + 1$. In order to obtain such a solution it is necessary to schedule dummy job $J_{3m+1}$ in batch $i$, $i = 1, \ldots, m + 1$. In addition, in the first $m$ batches partition jobs with a total processing time of $B$ must be scheduled. The latter is possible if and only if 3-PARTITION has a solution, i.e., if and only if there are sets $I_1, I_2, \ldots, I_m$ with $\sum_{i \in I_j} a_i = B$, such that the partition jobs $J_i$ with $i \in I_j$ can be scheduled in the $j$th batch. Figure 4 shows the structure of this optimal solution BS with $F(\text{BS}) \leq \bar{F}$. \( \square \)

Since the transformation from 3-PARTITION to $P$ requires $O(m)$ steps, our NP-hardness proof for the $\cdot | \cdot | \sum \alpha_i f_i$-problem is complete.

\subsection{3.2. The $\cdot | \text{Chains}, p_i = p | \sum \alpha_i f_i$-problem}

In this and the next subsection we examine batching problems in which precedence constraints between the jobs consist of parallel chains. Parallel chains form a graph in which each node has at most one predecessor and at most one successor. Figure 5 shows an example with seven jobs divided into three parallel chains.

We present a polynomial reduction from the $\cdot | \cdot | \sum \alpha_i f_i$-problem to the $\cdot | \text{Chains}, p_i = p | \sum \alpha_i f_i$-problem and thus prove that $\cdot | \text{Chains}, p_i = p | \sum \alpha_i f_i$ is NP-hard in the strong sense.
Fig. 5. An example for parallel chains.

Obviously, the \( \cdot | \text{Chains}, p_i = p | \sum \alpha_i f_i \) -problem belongs to the class NP.

Given a \( \cdot | \sum \alpha_i f_i \) -problem \( P \) and a bound \( \bar{F} \) we define a \( \cdot | \text{Chains}, p_i = p | \sum \alpha_i f_i \) -problem \( P' \) as follows:

Let \( n \) be the number of jobs in \( P \). For each job \( J_i \) with processing time \( p_i \) we construct a chain \( C(i) \) with \( p_i \) jobs.

\[
C(i) : J_{i1} \rightarrow J_{i2} \rightarrow \ldots \rightarrow J_{ip_i}
\]

All jobs \( J_{ij} \) have a processing time \( p_{ij} = 1, j = 1, \ldots, p_i \). The first \( p_i - 1 \) jobs have a job weight

\[
\alpha_{ij} = 0, \quad j = 1, \ldots, p_i - 1.
\]

The last job \( J_{ip_i} \) has a weight

\[
\alpha_{ip_i} = \alpha_i.
\]

We will show that \( P \) has a solution BS with \( F(\text{BS}) \leq \bar{F} \) if and only if \( P' \) has a solution BS' with \( F(\text{BS'}) \leq \bar{F} \).

**Lemma 3.3.** There exists an optimal solution of \( P' \) in which jobs belonging to the same chain are scheduled together in the same batch.

**Proof.** Let BS' be an optimal solution of \( P' \). Suppose there is some chain \( C(i) \) whose jobs are not scheduled in the same batch. Let \( j \) be the batch in which \( J_{ip_i} \) is scheduled. Suppose \( J_{il}, 1 \leq l < p_i \), is the last job of chain \( C(i) \) not scheduled in batch \( j \). If we move \( J_{il} \) from its original batch to the beginning of batch \( j \) we do not raise the weighted flow time of BS' and maintain feasibility of the solution. Iterating such movements leads to a solution with the desired property. \( \square \)

**Theorem 3.4.** \( P \) has a solution BS with \( F(\text{BS}) \leq \bar{F} \) if and only if \( P' \) has a solution BS' with \( F(\text{BS'}) \leq \bar{F} \).

**Proof.** \( (\Rightarrow) \) Let BS be a solution of \( P \) with \( F(\text{BS}) \leq \bar{F} \). In BS we replace job \( J_i \) by chain \( C(i) \) for all \( i = 1, \ldots, n \). Let BS' be the resulting solution, then BS' is a feasible solution for \( P' \) and \( F(\text{BS'}) \leq \bar{F} \).
Consider an optimal solution $BS'$ of $P'$ which satisfies the property described in Lemma 3.3. We have $F(BS') \leq \bar{F}$. Within each batch we may interchange the jobs such that jobs belonging to the same batch are scheduled consecutively. Obviously, $F(BS')$ is not affected by these rearrangements. For $i = 1, \ldots, n$ we now replace $C(i)$ by job $J_i$. The resulting solution is a solution for $P$ with $F(BS) \leq \bar{F}$. □

We notice that the construction of $P'$ requires polynomial time only if we encode each chain $C(i)$ efficiently, e.g. by

$$[(p_i - 1), 0], [1, \alpha_i]$$

This string describes that $C(i)$ consists of $p_i$ jobs with processing times equal to one. The first $p_i - 1$ jobs have a weight $\alpha_{ij} = 0$ and the weight of the last job equals $\alpha_i$.

3.3. The $\cdot |\text{Chains}| \sum f_i$-problem

We present a reduction from the well-known PARTITION problem to the $\cdot |\text{Chains}| \sum f_i$-problem. The PARTITION problem can be formulated as follows:

There are given $m$ positive integers $a_1, a_2, \ldots, a_m$. Is there a subset $I' \subseteq I = \{1, 2, \ldots, m\}$, such that

$$\sum_{i \in I'} a_i = \sum_{i \in I \setminus I'} a_i?$$

The $\cdot |\text{Chains}| \sum f_i$-problem obviously lies in the class NP.

For a given PARTITION problem we construct a $\cdot |\text{Chains}| \sum f_i$-problem $P$ as follows:

Define

$$B = \frac{1}{2} \sum_{i=1}^{m} a_i.$$  

(1) For each $a_i$, $i = 1, \ldots, m$, construct a chain $C(i)$ consisting of $a_i$ jobs. The first job $J_{i1}$ of this chain has a processing time $p_{i1} = a_i + 1$. This job is followed by $a_i - 1$ jobs $J_{i2}, J_{i3}, \ldots, J_{ia_i}$ with

$$p_{ij} = 1, \quad j = 2, \ldots, a_i.$$  

Hence, $C(i)$ is of the structure

$$C(i) : \quad J_{i1} \rightarrow J_{i2} \rightarrow \ldots \rightarrow J_{ia_i}$$

The jobs of these chains are called chain jobs.

(2) For $i = 1, \ldots, B$ define a "dummy" job $J_i$ with $p_i = 1$.

(3) Set the set-up time to $s = 2B$.

(4) Set $\bar{F} = 19B^2$.

We shall show that $P$ has a solution $BS$ with $F(BS) \leq \bar{F}$ if and only if the partition problem has a solution. We need a few preliminaries.
Suppose BS is an optimal solution of $P$. Let $K$ be the number of batches in BS and let $n_j$ be the batch size of batch $j, j = 1, \ldots, K$. We show that w.l.o.g. all dummy jobs are scheduled in the first batch.

Let $K > 1$ and let $J_i, 1 \leq i \leq B$, be a dummy job scheduled in batch $j$, where $1 < j \leq K$. We assume first that $n_1 \leq B$. In this case we schedule $J_i$ at the end of the first batch and obtain a new solution BS' with

$$F(\text{BS'}) - F(\text{BS}) \leq \sum_{t=1}^{j-1} n_t - (j - 1)2B - \sum_{i=2}^{j} n_t + 1$$

$$= n_1 - n_j - (j - 1)2B + 1,$$

because the flow times of the jobs in the first $j - 1$ batches increase by 1 and the flow time of job $J_i$ decreases at least by $(j - 1)2B + \sum_{i=2}^{j} n_t - 1$. Since $n_1 \leq B$ we have

$$F(\text{BS'}) - F(\text{BS}) \leq n_1 - n_j - (j - 1)2B + 1 < 0,$$

i.e., a contradiction to the optimality of BS. Thus, $n_1 > B$.

If $n_1 > B$, then there must be some $1 \leq l \leq m$ such that the chain job $J_{l1}$ is scheduled in the first batch. Let $J_{l1}, J_{l2}, \ldots, J_{lt}, 1 \leq t \leq a_l$, be those jobs of $C(l)$ which are scheduled preceding $J_i$. If among these jobs only $J_{l1}$ is scheduled in the first batch, it is easy to derive a contradiction to the optimality of BS:

Let $\pi(J_{lu})$ be the position where $J_{lu}$ is scheduled in BS, $u = 1, \ldots, t$, and let $\pi(J_i)$ be the position of $J_i$ in BS. We apply cyclic interchanges to these jobs, i.e., we set

$$\pi(J_{lu}) = \pi(J_{lu+1}), \; u = 1, \ldots, t - 1,$$

$$\pi(J_{l1}) = \pi(J_i),$$

$$\pi(J_i) = \pi(J_{l1}),$$

and thereby reduce the flow time of the solution by at least $(n_1 - 1)(p_{l1} - 1) > 0$, since $p_{l1} > 1$.

Thus, $t > 1$ and at least $J_{l2}$ is scheduled in the first batch. Now we apply cyclic interchanges to $J_{l2}, J_{l3}, \ldots, J_{lt}$ and $J_i$ and set

$$\pi(J_{lu}) = \pi(J_{lu+1}), \; u = 2, \ldots, t - 1,$$

$$\pi(J_{l1}) = \pi(J_i),$$

$$\pi(J_i) = \pi(J_{l2}).$$

Obviously, all interchanged jobs have a processing time $p = 1$. Therefore, the processing times within each batch do not change and the new solution BS' has the same structure like BS. In particular, BS' has the same flow time as BS. But job $J_i$ is now scheduled in the first batch.

Iterating such cyclic interchanges leads to a solution which has the same structure as the original solution BS and in which all dummy jobs are scheduled in the first batch.
In the remainder of this subsection, when we consider an optimal solution, we will assume that all dummy jobs are scheduled in the first batch.

**Lemma 3.5.** Let BS be an optimal solution of P. Then BS consists of exactly two batches.

**Proof.** In BS all dummy jobs are scheduled in the first batch. Let $K$ be the number of batches in BS.

If $K = 1$, then we schedule the last job in BS in a new second batch. The processing time of this job is $p \geq 1$. Let BS′ be the new resulting solution. We have

$$F(\text{BS}′) - F(\text{BS}) = 2B - (3B - 1)p < 0$$

and obtain a contradiction to the optimality of BS.

Now we consider the case $K \geq 3$. Suppose $k_j$ is the number of chain jobs in the $j$th batch, $j = 1, \ldots, K$. In a first step we show that all jobs scheduled in batch $K$ are independent of those jobs scheduled in the preceding $K - 1$ batches. In this connection we call two jobs independent if they do not belong to the same chain. Suppose we have some chain $C(i)$, $1 \leq i \leq m$, whose first job $J_{i1}$ is scheduled in a batch $j < K$ and whose $l$th job $J_{il}$, $1 < l \leq a_i$, is scheduled in batch $K$. W.l.o.g. we may assume that $J_{il}$ is the first job of $C(i)$ scheduled in the $K$th batch. Hence, in batch $K$ there exists one job of $C(i)$ which depends on at least one job in the first $K - 1$ batches. If we schedule $J_{il}$ at the end of batch $K - 1$ we obtain a new solution BS′ with

$$F(\text{BS}′) - F(\text{BS}) \leq k_{K - 1} - 2B < 0,$$

because all dummy jobs are in the first batch and thus $k_{K - 1} < 2B$. We have a contradiction to the optimality of BS.

Next we show that $k_{K - 1} \geq B$ is satisfied. We merge batch $K - 1$ and batch $K$ into a single batch by deleting the $K$th machine set-up. Let BS′ be the resulting solution. Due to the optimality of BS we have

$$0 \leq F(\text{BS}′) - F(\text{BS}) = k_{K - 1}(2k_K) - 2Bk_K = 2k_K(k_{K - 1} - B).$$

Since all jobs in batch $K$ are independent of those in the first $K - 1$ batches, the jobs in the $K$th batch have a mean processing time $p = 2$. Therefore, the flow times of the jobs in batch $K - 1$ increase by $2k_K$ and the flow times of the jobs in batch $K$ decrease by $s = 2B$. We conclude $k_{K - 1} \geq B$ because $k_K \geq 1$.

With the help of the last conclusion we are able to show that all jobs scheduled in the first batch are independent of the chain jobs in the following $K - 1$ batches. Suppose there is a chain $C(i)$, $1 \leq i \leq m$, whose first job $J_{i1}$ is scheduled in the first batch and whose $l$th job, for some $1 < l \leq a_i$, is scheduled in a subsequent batch. Let $J_{il}$, $1 < l \leq a_i$, be the first job of this chain not scheduled in the first batch but in a batch $j$, $2 \leq j \leq K$. If we schedule $J_{il}$ at the end of the first batch, the flow time of
$J_i$ decreases at least by $(j - 1)2B + \sum_{i=2}^{j} k_i - 1$. The flow times of the jobs in the batches $1, \ldots, j - 1$ increase by 1. Let BS' be the new solution. Then we have

$$F(\text{BS'}) - F(\text{BS}) \leq B + k_1 - (j - 1)2B - k_j + 1.$$  

Since $k_{k-1} \geq B$, both $k_1 < B$ and $B + k_1 - (j - 1)2B - k_j + 1 < 0$ must hold. Again, this is a contradiction to the optimality of BS.

In the last step of this proof we show that there exists a solution BS' which consists of less than $K > 2$ batches and which has a smaller flow time than BS. We consider our original solution BS. All jobs in batch $K$ are independent of the jobs in the first $K - 1$ batches and thus have a mean processing time of $p = 2$. Furthermore, the jobs in the first batch are independent of the jobs scheduled in the other batches, and thus the jobs in the batches $2, \ldots, K - 1$ also have a mean processing time $p = 2$. In the given solution BS we now move all jobs contained in batch $K$ to the end of the first batch. Let BS' be the new solution. We have

$$F(\text{BS'}) - F(\text{BS}) = \left( B + \sum_{j=1}^{K-1} k_j \right) 2k_K - \left( \sum_{j=2}^{K-1} 2k_j + (K - 1)2B \right) k_K$$

$$= 2k_K(k_1 - B(K - 2)),$$

because the flow times of the jobs in the batches $j = 1, \ldots, K - 1$ increase by $2k_K$ and the flow times of the jobs in batch $K$ decrease by $\sum_{j=2}^{K-1} 2k_j + (K - 1)2B$. Since $k_{k-1} \geq B$ and $k_1 < B$, we have $F(\text{BS'}) < F(\text{BS})$. This contradiction proves that an optimal solution of $P$ consists of exactly two batches. \[\Box\]

Now we show that the partition problem can be reduced to $P$.

**Theorem 3.6.** The problem $P$ has a solution BS with $F(\text{BS}) \leq 19B^2$ if and only if the partition problem has a solution.

**Proof.** ($\Leftarrow$) If there exists a subset $I' \subseteq I$ with $\sum_{i \in I'} a_i = \sum_{i \in I \setminus I'} a_i$, then we group the chains $C(i)$ with $a_i \in I'$ in a batch $B_1$. The chains $C(i)$ with $a_i \in I \setminus I'$ are grouped in a second batch $B_2$. Obviously, every batch now contains exactly $B$ chain jobs and has a total processing time of $p = 2B$. In the first batch we also schedule the remaining $B$ dummy jobs. We have

$$F(\text{BS}) = (2B + 3B)(2B) + (4B + 5B)B = 19B^2.$$  

($\Rightarrow$) Consider an optimal solution BS with $F(\text{BS}) \leq 19B^2$. W.l.o.g. we assume that all dummy jobs are scheduled in the first batch. Within the first batch we may rearrange the jobs such that the dummy jobs are scheduled first. Obviously, $F(\text{BS})$ is not affected by these interchanges. According to Lemma 3.5, BS contains exactly two batches. Thus, we may assume that BS has the structure given in Fig. 6. Let $k_i, i = 1, 2$, be the number of chain jobs in the $i$th batch. We have $k_1 + k_2 = 2B$. First, we show that $k_1 = k_2 = B$ must hold.
If $k_1 < B$, then at most $2B - 1$ jobs are scheduled in the first batch. We show that chains, having some job scheduled in the first batch, are completely scheduled in the first batch. Suppose there is a chain $C(i)$ whose first job is in batch one and whose $l$th job is in batch two. Let $J_{il}, 2 \leq l \leq a_i$, be the first job of chain $C(i)$ scheduled in batch two. Moving this job to the end of the first batch we obtain a solution $BS'$ with

$$F(\text{BS'}) - F(\text{BS}) \leq 2B - 1 - 2B < 0.$$ 

We have a contradiction to the optimality of $BS$. Thus, both the chain jobs in the first and in the second batch have a mean processing time of $p = 2$. Since $k_1 = 2B - k_2$, it follows

$$F(\text{BS}) = (3B + 2k_1)(B + k_1) + (9B)k_2$$

$$= (3B + 2(2B - k_2))(3B - k_2) + 9Bk_2$$

$$= 21B^2 - 4Bk_2 + 2k_2^2.$$ 

For $x \in \{1, \ldots, 2B\}$ we define

$$f(x) := 2x^2 - 4Bx + 21B^2.$$ 

Then

$$f'(x) = 4x - 4B$$

and

$$f''(x) = 4.$$ 

Thus, for $x = B$ the function $f$ has an absolute minimum and $f(B) = 19B^2$. Since $k_2 > B$, we have $F(\text{BS}) > 19B^2$.

If $k_1 > B$ then let $p$ be the mean processing time of the $k_1$ chain jobs in the first batch. We have $p \geq 2$.

$$F(\text{BS}) = (2B + B + k_1p)(B + k_1) + 9Bk_2$$

$$= 21B^2 - 6Bk_1 + Bk_1p + k_1^2p.$$ 

We show

$$-6Bk_1 + Bk_1p + k_1^2p > -2B^2.$$
Since \( k_1 > B \) we have \( k_1 = B + i \) for some \( 1 \leq i < B \). Thus,
\[
-6Bk_1 + Bk_1 p + k_1^2 p = -6B(B + i) + B(B + i)p + (B + i)^2 p
= -6B^2 - 6Bi + B^2 p + Bip + B^2 p + 2Bip + i^2 p
\geq -2B^2 + Bi(-6 + 3p) + i^2 p
\geq -2B^2 + i^2 p
\]

Hence, \( F(BS) > 19B^2 \).

It follows that \( k_1 = k_2 = B \). Again, we show that chains, having one job scheduled in the first batch, are completely scheduled in batch one. Consider a chain \( C(i) \), \( 1 \leq i \leq m \), whose first job belongs to batch one and whose \( l \)th job, \( 1 < l \leq a_i \), belongs to batch two. Let \( p \) be the mean processing time of the \( k_1 \) chain jobs in the first batch. We have \( p > 2 \) and thus
\[
\]

We set
\[
I' = \{i| C(i) \text{ is scheduled in the first batch}\}.
\]

Since \( k_1 = B \), \( I' \) solves the partition problem, i.e.,
\[
\sum_{i \in I'} a_i = \sum_{i \notin I \setminus I'} a_i.
\]

If we encode the \( B \) dummy jobs by
\[
[B, 1]
\]
and each chain \( C(i) \) by
\[
[1, a_i + 1], [a_i - 1, 1]
\]
then the construction of the problem \( P \) requires a polynomial number of steps. Our NP-hardness proof is complete.

4. Concluding remarks

The results presented in this paper provide a reasonable idea of the borderline between polynomially solvable and NP-hard batching problems. We mention that using the proofs developed in Section 3 it is easy to show that the problems
\[
|\text{Tree}| \sum f_i, \quad |\text{Tree}, p_i = p| \sum \alpha_i f_i,
\]
\[
|\text{s.p.}| \sum f_i \quad \text{and} \quad |\text{s.p.}, p_i = p| \sum \alpha_i f_i
\]
are NP-hard, too. Tree and s.p. denote precedence constraints with a tree and series parallel structure, respectively.

Concerning the \( \cdot |\text{Chains}| \sum f_i \) problem we cite an interesting result developed in [1]. If the number of chains is fixed we can solve this batching problem in polynomial time. More precisely, the time complexity is \( O(nr \prod_{j=1}^{r} n_j) \), where \( r \) is the number of chains and \( n_j \) is the number of jobs in chain \( j \), \( \sum_{j=1}^{r} n_j = n \). The idea of the algorithm is to reduce the \( \cdot |\text{Chains}| \sum f_i \) problem to a shortest path problem in a network with \( \prod_{j=1}^{r} n_j \) vertices and \( O((\prod_{j=1}^{r} n_j)^2) \) edges. The methods presented in Section 2.2 can be extended and lead to a shortest path algorithm which is linear in the number of vertices. In the context of this algorithm two interesting open problems arise. First, it might be interesting to resolve whether it is possible to generalize the ideas sketched above in such a way that batching problems with series parallel precedence constraints can be solved. Second, it is unknown if the \( \cdot |\text{Chains}| \sum f_i \) problem is NP-hard in the strong sense.

Another topic for future research is the development of heuristics for NP-hard batching problems. Investigating \( \cdot |\text{Chains}| \sum f_i \) problems with a small number of jobs we observed that in optimal solutions the underlying job sequence slightly differs from optimal schedules of the corresponding scheduling problems. Thus, it might be a good heuristic for the \( \cdot |\text{Chains}| \sum f_i \) problem to solve the adjunct scheduling problem first and apply the shortest path algorithm of Section 2.2 in a second step.

Naturally, NP-hard batching problems can be solved exactly by branch and bound methods. In this connection, the development of good lower bounds is a prime concern.

Finally, we would like to mention that no significant results are known about the complexity of batching problems with parallel machines.

References