Equal processing and equal setup time cases of scheduling parallel machines with a single server

Amir H. Abdekhodaee\textsuperscript{a}, Andrew Wirth\textsuperscript{b,\*}, Heng Soon Gan\textsuperscript{b}

\textsuperscript{a}Operations Research Group, CSIRO Mathematical and Information Sciences, Private Bag 10, Clayton South, MDC 3169, Australia
\textsuperscript{b}Department of Mechanical and Manufacturing Engineering, The University of Melbourne, Vic 3010, Australia

Abstract

This paper considers the deterministic problem of scheduling two-operation non-preemptable jobs on two identical semi-automatic machines. A single server is available to carry out the first (or setup) operation. The second operation is executed automatically, without the server. The general problem of makespan minimization is NP-hard in the strong sense. Two special cases which arise in practice, the equal processing times and regular equal setup times problems turn out to be closely linked to the restricted set partitioning problem. This link is used to show that they are NP-hard in the ordinary sense, and to construct a tight lower bound and two highly effective $O(n \log n)$ heuristics.

© 2003 Elsevier Ltd. All rights reserved.

Keywords: Parallel machines; Single server; Setup

1. Introduction

The problem considered here is how to arrange two-operation non-preemptable jobs on two identical semi-automatic machines. For the first operation, i.e. setup, the presence of an operator and availability of a machine is necessary while the second operation, processing on the machine, can be carried out automatically and so the presence of the operator is not required. The setup times and processing times are known in advance and the objective is to arrange jobs so that the maximum completion time or makespan can be minimised. Another objective which is discussed in the literature and closely related to makespan is machine interference. Machine interference minimisation is similar to makespan minimisation except that it ignores machine idle time after the completion of all jobs on that machine.

\* Corresponding author. Tel.: +61-3-8344-4852; fax: +61-3-9347-8784.
E-mail address: wirth@mame.mu.oz.au (A. Wirth).

0305-0548/$ - see front matter © 2003 Elsevier Ltd. All rights reserved.
doi:10.1016/S0305-0548(03)00144-8
The assignment of operators to semi-automatic machines has been studied extensively and some reports even date back to the mid-1930s. However, these researchers tackled the problem using mainly a stochastic approach. Stecke and Aronson [1] provided a well classified survey of these methods. The deterministic problem has only fairly recently begun to attract some attention. Koulamas and Smith [2] were the first to discuss the problem in a way similar to that presented in this paper. They proposed a look-ahead algorithm and demonstrated that this increases the utilisation of machines. Koulamas [3] showed that minimising total interference is NP-complete in the strong sense. He introduced a reduction procedure through which a problem is simplified and any sequence for the reduced problem would alternate between machines. The reduced problem is then solved using beam search which he argued was superior to a local search algorithm.

Hall et al. [4] provided a comprehensive complexity analysis of the problem with various objective functions including makespan. However, results on solution methods for makespan were limited. Kravchenko and Werner [5] presented a pseudopolynomial time algorithm for the case of two machines when all setup times are equal to one. They studied the complexity of this problem for an arbitrary number of machines and showed that it is strongly NP-complete.

Later in another paper, Kravchenko and Werner [6] provided a pseudopolynomial time algorithm for \( m \) machines and \( m - 1 \) operators. This problem turned out to be strongly NP-complete when minimising maximum lateness. They also reported some results on the stochastic version of the unit setup time makespan minimisation problem.

In this article we follow the same line as developed in our previous paper [7] in which we presented some computational complexity results and developed an integer programming model for waiting and idle times. We also solved the equal length jobs case exactly using a sorting algorithm. Some heuristics for the general case were proposed and their performances were compared to published results. The proposed heuristics appeared to perform better than those of Koulamas [3].

Here, however, we mainly discuss the special cases of equal processing time and equal setup times, analyse their computational complexity and observe how good heuristics can be established for them.

2. The model

We follow the notation of Abdekhodaee and Wirth [7] and begin by recalling some of the results of that paper. Assume that we have \( n \) jobs with setup times \( s_i \) and processing times \( p_i \) for \( i = 1, \ldots, n \). Let \( t_i \) be the start time of job \( i \) and \( c_i \) its completion time. So \( c_i = t_i + s_i + p_i \). Denote the length of job \( i \) by \( a_i = s_i + p_i \). Assume also that we have two identical parallel machines and a single operator to carry out the setups. In the notation of Kravchenko and Werner our problem is \( P2, S1|s_i|C_{\text{max}} \) (two identical parallel machines, one common server, arbitrary setups, makespan minimisation) (Fig. 1).

As stated earlier processing does not require the server. We also introduce the convention that uppercase terms shall refer to the set of jobs after it has been scheduled, thus \( C_i \) shall refer to the completion time of the \( i \)th scheduled job and \( S_j \) is the setup time of the job that is scheduled first.

We assume that no job is unnecessarily delayed. We say a set of jobs is regular if \( p_i \leq a_j \) for all \( i, j \). Until further notice we make the weaker assumption that the jobs are processed alternately on the machines. It will be convenient to consider five dummy jobs, in positions \(-1, 0, n + 1, n + 2\).
and \( n + 3 \) which have zero setup and processing times:

\[
P_{-1} = P_0 = P_{n+1} = P_{n+2} = P_{n+3} = S_{-1} = S_0 = S_{n+1} = S_{n+2} = S_{n+3} = 0 \quad \text{with} \quad T_{-1} = T_0 = 0,
\]

\[
T_{n+1} = \max(C_{n-1}, T_n + S_n), T_{n+2} = C_n \quad \text{and} \quad T_{n+3} = T_{n+2}.
\]

It is generally impossible to avoid both machine idle time and server wait time. We wish to minimise makespan, that is \( \max_{i=1}^{n} C_i \). Let \( I_i \), the \( i \)th machine idle time, be the time the machine which has just finished the \( i \)th job is idle before it starts its next job. We assume, without loss of generality, that the first job is started on machine one and that \( T_1 = 0 \). Denote by \( W_i \) the server waiting time between the end of the setup of the \((i+1)\)th and the start of the \((i+2)\)th scheduled jobs. We recall that \( x^+ = \max(x, 0) \).

The proofs of following propositions appears in Abdekhodaee and Wirth [7].

**Proposition 1.** Under the assumption of alternating processing the machine idle time of the \( i \)th scheduled job, \( I_i = (T_{i+1} + S_{i+1} - T_i - A_i)^+ \) for \( i = 0, \ldots, n + 1 \) and \( T_{i+2} = A_i + T_i + I_i \) for \( i = 0, \ldots, n \). \( W_i = T_{i+2} - T_{i+1} - S_{i+1} \) for \( i = 0, \ldots, n \). Also \( \text{makespan} = T_{n+2} \). If the jobs are regular then they are processed alternately.

**Proposition 2.** Under the assumption of alternating processing,

\[
\text{makespan} = \sum_{i=1}^{n} S_i + \sum_{i=1}^{n} W_i = \frac{1}{2} \left( \sum_{i=1}^{n} a_i + \sum_{i=0}^{n+1} I_i \right).
\]

Also \( W_i = I_i + P_i - S_{i+1} - W_{i-1} = (P_i - S_{i+1} - W_{i-1})^+ \), \( I_i = (W_{i-1} + S_{i+1} - P_i)^+ \) and \( W_i = 0 \) or \( I_i = 0 \) for \( i = 1, \ldots, n \).

**Proposition 3.** The decision version of the makespan minimisation problem is NP-complete in the strong sense.

The following lemma turns out to be very useful in finding some optimality criteria which simplify the special cases.

**Lemma 1.** For a regular set of jobs, if we reduce a single processing time or increase a single setup then the total server waiting time remains the same or is reduced.
Proof. The proof is by induction. In general, \( W_i = (P_i - S_i + 1 - W_{i-1})^+ \). Without loss of generality, we shall assume that it is job 1 that is changed. If \( W' \) indicates the modified server waiting time we have: \( W'_1 = (P_1 - S_2)^+ \) and \( W'_1 = (P_1 - S_2 - h)^+ \) where \( h \geq 0 \). So clearly, \( W'_1 \leq W_1 \).

Before proceeding, it should be noted that from Proposition 2 \( W'_2 = (P_2 - S_3 - W'_1)^+ \leq W_2 \) since \( W'_1 \leq W_1 \):

\[
W'_1 + W'_2 = P_2 - S_3 \leq (P_2 - S_3 - W_1)^+ + W_1 = W_1 + W_2.
\]

The result is also true for \( m = 2 \).

We assume that the result is true for \( m = 1, \ldots, k \), i.e. \( \sum_{i=1}^{k} W'_i \leq \sum_{i=1}^{k} W_i \), and prove that it is then true for \( k + 1 \) server waiting times.

If \( W'_{k+1} = 0 \) then the result is true for \( m = k + 1 \).

If \( W'_{k+1} > 0 \) then \( P_{k+1} - S_{k+2} = W'_k + W'_{k+1} \) and \( P_{k+1} - S_{k+2} \leq W_k + W_{k+1} \). Since the result is true for \( m = k - 1 \) it is true for \( m = k + 1 \).

So \( \sum_{i=1}^{k+1} W'_i \leq \sum_{i=1}^{k+1} W_i \) is true and the proof is complete. \( \square \)

3. The equal processing time and equal set-up time problems

3.1. The equal processing time problem

The equal processing time case is clearly regular since \( p_i = p \leq s_j + p_j = s_j + p \). A practical example is a coating procedure where a number of parts have to be covered by a specific material. Assume the number of jobs may differ from one coating cycle to another. If loading time is a function of the number of jobs and the processing time is constant, we have an equal processing time problem.

To study the equal processing time problem we first consider the following definition and the proposition below. If \( s_j \leq p \) we say the setup is short, otherwise, if \( s_j > p \), it is long. By a run of short (long) setup jobs we mean a maximal sequence of jobs each with short (long) setups. Denote the short setups by \( S \) and the long setups by \( L \).

Lemma 2. For jobs with equal processing times, any run of purely long setup jobs can be replaced by a single job with a setup and processing time both equal to \( p \) without changing the sum of waiting times, whilst reducing the makespan by the sum of the long setups minus \( p \), that is \( \sum_{i \in \text{short setups}} L_i - p \).

Proof. Consider a sequence of jobs such as \( S_1, \ldots, S_k, L_1, \ldots, L_j, S_{k+1}, \ldots, S_n \). The makespan equals

\[
\sum_{i \in \text{short setups}} S_i + \sum_{i \in \text{long setups}} L_i + \sum_{i \in \text{short setups}} W_i + \sum_{i \in \text{long setups}} W_i.
\]

Note that the last of the above four sums is, in fact, zero. We now show that by the above replacement, the sum the waiting times will remain unchanged. The makespan will be reduced by
\[ W_{i-1} = (p - L_i - W_{i-2})^+ = 0 \]
\[ W_i = (p - S_{i+1})^+ \]
\[ W_{i+1} = (p - L_{i+2} - W_i)^+ = 0 \]
\[ W_{i+2} = (p - S_{i+3})^+ \]

\[ W'_{i-1} = (p - L_i - W_{i-2})^+ = 0 \]
\[ W'_i = (p - L_{i+2})^+ = 0 \]
\[ W'_{i+1} = (p - S_{i+1})^+ \]
\[ W'_{i+2} = (p - S_{i+3} - W'_{i+1})^+ \]

\[ \sum_{i=1}^{j} L_i - p, \text{ since the replacement results in the second of the above four sums being decreased by this amount, the other sums remaining unchanged.} \]

Consider \( W_k = (p - L_1 - W_{k-1})^+ = 0, W_{k+j} = p - S_{k+1}, W_i = 0 \) if \( k + 1 \leq i \leq k + j - 1 \), and otherwise \( W_i = (p - S_{i+1} - W'_{i-1})^+ \). If we replace this run of long setup times with a single job of (setup = \( p \) and processing = \( p \)), then we have \( W_k = (p - p - W_{k-1})^+ = 0 \), \( W_{k+j} = p - S_{k+1} \), and otherwise \( W_i = (p - S_{i+1} - W_{i-1})^+ \) which is the same as the previous set of equations. Therefore, the above replacement can be performed. The cases \( S_1, \ldots, S_k, L_1, \ldots, L_j \) and \( L_1, \ldots, L_j, S_1, \ldots, S_k \) are proved similarly. \( \square \)

The above lemma allows us to reduce the problem somewhat.

**Proposition 4.** For jobs with equal processing times, there exists an optimal sequence in which jobs with long setups form a single run.

**Proof.** We must show that if an optimal solution contains LSL, a run of long setup jobs followed by a run of short setup jobs, followed in turn by a run of long setup jobs, it may be replaced by LS. Repeating this replacement procedure it will follow that there exists an optimal solution with at most one run of long setups. We consider here a special case which outlines the type of argument required.

Consider an optimal sequence such as \( \ldots S_{i-1} L_i S_{i+1} L_{i+2} S_{i+3} \ldots \) of the setup times. Then the waiting times are given in column A of Table 1. If we swap \( L_{i+2} \) and \( S_{i+1} \) to obtain the sequence \( \ldots S_{i-1} L_i L_{i+2} S_{i+1} S_{i+3} \ldots \) then the operator waiting times are given in column B of Table 1.

By reasoning similar to that applied in the proof of Lemma 1, the total waiting time of the new arrangement is not increased. Similar arguments for runs of short and long setup jobs show that there exists an optimal sequence with at most one run of long setups. \( \square \)

Thus there exists an optimal sequence of the type SL, LS or SLS.

If all the setups are longer than the processing times of other jobs, it follows from Proposition 2 that the makespan = \( \sum_{i=1}^{n} s_i + P_n \), so any sequence is optimal provided that the last job is the one with least processing time. On the other hand, for short setup time jobs there are as we show later, efficient heuristics.

It follows from Proposition 4 and Lemma 2 that for jobs with equal processing times we may assume that all long setup jobs form a single run and, in fact, may be replaced by a single job with setup time equal to \( p \). Hence, for jobs with equal processing times we may assume, without loss of generality, that all setups are short.
3.2. The equal setup times problem

Our problem with equal setups is closely related to the following one of scheduling with safety distances. Spieksma et al. [8] study the latter problem of scheduling jobs (without setups) on two parallel machines without idle time. Each job is already assigned to its machine. Their objective is to sequence the jobs on the two machines so as to maximize the minimum time between any two completion times. In particular, they ask if it is possible to find a schedule so that this minimum is no less than some value \( d \). It can be easily shown that this is equivalent to our problem with the changes that we assume the jobs are pre-assigned to the machines, and all but the first ones on each machine have a setup of \( d \) time units. The first jobs have no setups. We seek a schedule with no idle time.

We note that a set of equal processing time jobs is automatically a regular set, thus we have the alternating case. However, when we have equal setup times the set may not be regular and we should consider conditions which ensure that the set is regular. One such necessary and sufficient condition is \( \max p_i - \min p_j \leq s \) since then \( p_i \leq \max p_i \leq \min p_j + s \leq a_j \), hence we have a regular set where \( s \) is the constant setup time.

The problem of equal setup times has been solved pseudopolynomially for the special case of unit setup times by Kravchenko and Werner [5]. Unit setup time, in fact, means that all processing times are some multiple of the setup time. Therefore, processing times are relatively large. We firstly consider a regular version of the problem, one for which the range of variation in processing times is less than or equal to the setup time. We will present some suggestions later in this paper for the non-regular case.

**Proposition 5.** For a regular set of jobs with equal setup times there exists an optimal solution which contains at most one run of long processing times.

**Proof.** An application of Lemma 1 to an argument analogous to that in Proposition 4 for equal processing times yields the result. \( \square \)

From the comment following Proposition 4, if all processing times are shorter than the setups any sequence is optimal provided the last job has the least processing time. Therefore, in the remainder of this paper, we mainly deal with the case where there are long processing times, that is, processing times which are larger than setup times.

4. Some lower bounds

It is useful to establish tight lower bounds because of their ability to demonstrate the quality of different heuristics. Moreover, a good lower bound may also have some of the characteristics of an optimal or near optimal solution. We now propose some lower bounds for regular sets of jobs.

**Proposition 6.** For a regular set of jobs

\[
\text{makespan} \geq \frac{1}{2} \left( \sum_{i=1}^{n} a_i + s_1 + s_2 + p_{\text{minimum}} - p_{\text{maximum}} \right).
\]
If the number of jobs, \( n \), is even, processing times are equal and \( s_1 \leq s_2 \leq \cdots \leq s_n \) then

\[
\text{makespan} \geq \frac{1}{2} \left( \sum_{i=1}^{n} a_i + s_1 + s_2 + \bar{\varepsilon} \right)
\]

where \( \bar{\varepsilon} = \min_{A,A'} |\sum_{i \in A} s_i - \sum_{j \in A'} s_j| \), \( A \cup A' = \{3,4,\ldots,n\} \), \( A \cap A' = \emptyset \) and \( |A| = |A'| \).

If, in addition, \( s_i \leq p/2 \) for all \( i \) then this latter lower bound is tight.

**Proof.** For a regular set of jobs we have makespan = \( \frac{1}{2}(\sum_{i=1}^{n} a_i + \sum_{i=0}^{n+1} I_i) \), by Proposition 2. Fig. 1 illustrates this. Now \( I_0 = S_1 \) and \( I_{n-1} + I_n + I_{n+1} \geq S_n + P_n - P_{n-1} \).

So

\[
\text{makespan} \geq \frac{1}{2} \left( \sum_{i=1}^{n} a_i + S_1 + S_n + P_n - P_{n-1} \right).
\]

Hence

\[
\text{makespan} \geq \frac{1}{2} \left( \sum_{i=1}^{n} a_i + s_1 + s_2 + p_{\text{minimum}} - p_{\text{maximum}} \right).
\]

Now suppose the number of jobs, \( n \), is even and let \( n = 2k \). Consider any schedule and let \( B \) and \( B' \) be the sets of indices of the jobs (other than those in positions 1 and \( n \)) assigned to machines 1 and 2, respectively.

Then \( T_n \geq S_1 + (k-1)p + \sum_{i \in B} s_i \). So, makespan \( \geq S_1 + (k-1)p + \sum_{i \in B} s_i + S_n + p \). Similarly, since \( I_0 = S_1 \)

\[
\text{makespan} \geq S_1 + (k-1)p + \sum_{j \in B'} s_j + S_n + p.
\]

So

\[
\text{makespan} \geq S_1 + S_n + kp + \max \left( \sum_{i \in B} s_i, \sum_{j \in B'} s_j \right).
\]

Since \( 2 \max(u,v) = u + v + |u - v| \) it follows that

\[
\text{makespan} \geq \frac{1}{2} \left( \sum_{i=1}^{n} a_i + s_1 + s_2 + \mu \right),
\]

where

\[
\mu = \left| \sum_{i \in B} s_i - \sum_{j \in B'} s_j \right|.
\]

Fig. 2 illustrates this with \( I_0 = s_1 \) and \( I_{n+1} = s_2 + \mu \). Suppose for now that neither the first job, with setup \( s_a \), nor the last job in the schedule, with setup \( s_b \), is the shortest nor the second shortest job. Swap the first job and the shortest job, and the last job and the second shortest job. Consider now the new schedule. Let \( A \) be the set of indices corresponding to the jobs on machine 1 other than
the one in the first position and let \( A' \) be the set of indices of the jobs on machine two other than in the last position. Then

\[
\varepsilon \leq \sum_{i \in A} s_i - \sum_{j \in A'} s_j \leq \sum_{i \in B} s_i - \sum_{j \in B'} s_j + s_a - s_1 + s_b - s_2.
\]

A similar argument holds in the other cases. So

\[
\text{makespan} \geq \frac{1}{2} \left( \sum_{i=1}^{n} a_i + s_1 + s_2 + \varepsilon \right).
\]

Finally, we show that this lower bound is tight if \( s_i \leq p/2 \) for all \( i \). We note that \( \varepsilon \leq p/2 \) since merely partitioning the jobs in longest setup time order yields \( \mu \leq p/2 \). Without loss of generality \( \sum_{i \in A} s_i \leq \sum_{j \in A'} s_j \). Sort the jobs in \( A \) and \( A' \) in decreasing order to yield setups \( \{ x_i \} \) and \( \{ \beta_i \} \). Let \( \delta_i = x_i - \beta_i \). Now sort \( \delta_i \) in decreasing order. We note that if

\[
\delta_i \geq 0 \geq \delta_j
\]

then

\[
\delta_i + |\delta_j| \leq p/2.
\]

Suppose that \( x_i \geq x_j \) then \( \beta_i \geq \beta_j \). So \( x_i - \beta_i + \beta_j - x_j \leq x_i \leq p/2 \). The other case is proved similarly.

We have \( 0 \leq \sum_{i=1}^{k-1} \delta_i \leq p/2 \). It now follows from \( \delta_i + |\delta_j| \leq p/2 \) that we can ensure, by further sorting, that \( 0 \leq \sum_{i=1}^{h} \delta_i \leq p/2 \) for \( h = 1, \ldots, k-1 \). (Choose positive deltas until their sum is about to exceed \( p/2 \). Then, choose negative deltas until the sum is about to become strictly negative. Repeat this process until we have selected all the deltas. Relabel the alphas and betas.)

Our schedule is the following. Let \( S_1 = s_1, S_{2k} = s_2, S_{2i} = x_i \) and \( S_{2i+1} = \beta_i \).

So, \( W_1 = p - x_1 \) and so \( I_1 = 0 \). Similarly, \( W_2 = p - S_3 - p + S_2 = x_1 - \beta_1 = \delta_1 \geq 0 \). So, \( I_2 = 0 \). Also, \( W_3 = p - S_4 - S_2 + S_3 = p - S_4 - \delta_1 \geq 0 \) since \( S_4 \leq p/2 \) and \( \delta_1 \leq p/2 \). By similar arguments we show that all idle times other than \( I_0 = s_1 \) and \( I_{n+1} = s_2 + \sum_{i=1}^{k} \delta_i \) are zero. Hence the lower bound,

\[
\frac{1}{2} \left( \sum_{i=1}^{n} a_i + s_1 + s_2 + \varepsilon \right)
\]

is tight. \( \square \)

The following example shows that we cannot replace the condition \( s_i \leq p/2 \) above by \( s_i \leq p \) and retain tightness. Consider a set of eight jobs with \( p = 10 \) and setups with durations 0, 0, 4, 8, 8, 8,
10 and 10. Clearly, \( \varepsilon = 0 \) but it is easy to show that we cannot achieve a schedule with zero idle time.

5. Complexity analysis of special cases

When a special case of an NP-complete problem is considered, there is a question as to whether the special case still remains NP-complete. In fact, sometimes apparently minor changes to a problem in P may convert it to an NP-complete problem while restricted versions of some NP-complete problems are solvable in polynomial time. Moreover, the complexity analysis might contain useful clues on how to generate powerful heuristics for a problem.

5.1. Complexity analysis of the equal processing time problem

It was noted earlier that this problem is regular and, by Proposition 4, it can be broken down into two or three sets, a set with long setups (within which the sequence is essentially immaterial except for the last job) and one or two sets with short setups. We address the sequencing of the short setup case and show that the problem is NP-complete in the ordinary sense. We achieve this by reducing an already NP-complete problem to this problem.

The argument in the following lemma is very similar to that used above in the proof of the tight lower bound result.

**Proposition 7.** The equal processing time problem with short setups is NP-complete in the ordinary sense.

**Proof.** We transform the restricted partitioning problem (partitioning into sets with equal cardinality) which is known to be NP-complete in the ordinary sense, Garey and Johnson [9], to the equal processing time problem with short setups.

The restricted partitioning problem is as follows. Let \( \{d_1, d_2, \ldots, d_{2k}\} \) be a set of \( 2k \) positive integers. Is there a partition so that \( \sum_{i \in A} d_i = \sum_{j \in A'} d_j \) subject to \( \|A\| = \|A'|\)?
We define $\delta_i = d_{2i-1} - d_{2i}$. It is clear that $\sum_{i=1}^{k} \delta_i = 0$. We arrange the $\delta_i$ in decreasing order so that after a further relabelling we have

$$\sum_{i=1}^{h} \delta_i \geq 0 \quad \text{for all} \quad 1 \leq h \leq k.$$ 

Let $S_1 = S_{2k+2} = 0$, and $S_i = d_{i-1}$ for $2 \leq i \leq 2k + 1$. We show that for this sequence all idle times are zero. Recall that $I_i = (W_i - S_{i+1} + P_i)^+ \text{ and } W_i = (P_i - S_{i+1} - W_{i-1})^+$. So

$$W_1 = (P - S_2)^+ = P - S_2 > 0 \quad \text{so} \quad I_1 = 0,$$

$$W_2 = (P - S_3 - P + S_2)^+ = S_2 - S_3 = \delta_1 \geq 0 \quad \text{and} \quad I_2 = (S_3 - S_2)^+ = 0,$$

$$W_3 = (P - S_4 - S_2 + S_3)^+ > 0 \quad \text{so} \quad I_3 = 0,$$

$$W_4 = (P - S_5 - P + S_4 + S_2 - S_3)^+ = \delta_1 + \delta_2 \geq 0 \quad \text{and} \quad I_4 = (S_3 + S_5 - S_2 - S_4)^+ = 0$$

$$\vdots$$

$$W_{2h-1} = (P - S_{2h} - \delta_1 - \cdots - \delta_{h-1})^+ > 0 \quad \text{so} \quad I_{2h-1} = 0$$

and

$$W_{2h} = \delta_1 + \cdots + \delta_h \geq 0 \quad \text{so} \quad I_{2h} = (-\delta_1 - \delta_2 - \cdots - \delta_h)^+ = 0$$

for $1 \leq h \leq k$.

Also $I_{2k+1} = 0$ since $W_{2k} = 0$ and $S_{2k+2} = 0$. So all idle times are zero, and the proof is complete.

We shall later use the argument of the above lemma to construct heuristics for the equal processing times problem.

5.2. Complexity analysis of the equal setup time problem

The general equal setup time problem has already been shown to be NP-complete in the strong sense (Hall et al. [4]). We note, however, that their proof relies on a non-regular set of jobs. We address the complexity issue when the problem is regular. We further restrict the problem to the case where there is no short processing time job.

**Proposition 8.** The regular equal setup time problem with long processing times is NP-complete in the ordinary sense.

**Proof.** The proof is very similar to the previous case with some minor changes to the transformation. As before, we transform the restricted partitioning problem (stated above) to the equal setup time problem.

Let $\{d_1, d_2, \ldots, d_{2k}\}$ be a set of $2k$ positive integers. Is there a partition so that $\sum_{i \in A} d_i = \sum_{j \in A'} d_j$ subject to $\|A\| = \|A'\|$?

Let $s = d_{\text{max}} - d_{\text{min}} + 2$, $p_i = d_i + M$ where $M > \sum_{i=1}^{2k} d_i$ for $i=1, \ldots, 2k$. Also let $p_{2k+1} = d_{\text{max}} + M + 1$ and $p_{2k+2} = d_{\text{min}} + M - 1$. Then clearly, $p_i - p_j \leq s$ for all $i, j$ so the set is regular and also all processing times are long.

Suppose that there is a schedule with $\sum_{i=0}^{2k+3} I_i \leq s$ then clearly $\sum_{i=0}^{2k+3} I_i = s$ as $I_0 = s$. Also we must have $|P_{2k+1} - P_{2k+2}| = s$ since $\sum_{i=1}^{2k+3} I_i = 0$ and the jobs are regular. Hence $P_{2k+1}$ and
\(p_{2k+2}\) must be the processing times of the last two jobs. So an appropriate partition of \(\{d_1, \ldots, d_{2k}\}\) exists.

Conversely, suppose that a partition, satisfying the condition \(\|A\| = \|A'\|\) exists. We must show that \(\sum_{i=0}^{2k+3} I_i \leq s\) for some schedule. Partition the set so that \(\delta_i = d_{2i} - d_{2i-1}\), and arrange \(\{\delta_i\}\) in decreasing order, as before. Thus

\[W_1 = (P_1 - S)^+ > 0\]
\[W_2 = (P_2 - S - W_1)^+ = P_2 - P_1 = \delta_1 \geq 0\]

so
\[I_1 = 0,\]
and
\[I_2 = (P_1 - P_2)^+ = 0\]

and so on. Hence \(I_i = 0\) for \(i = 1, \ldots, 2k\). Since \(W_{2k} = 0\) (as \(\sum_{i=1}^{k} \delta_i = 0\)) and \(s + P_{2k+2} = P_{2k+1}\) we also have \(I_{2k+1} = I_{2k+2} = I_{2k+3} = 0\). Hence \(\sum_{i=0}^{2k+3} I_i = s\) (as \(I_0 = s\)), concluding the proof. \(\square\)

### 6. Effective and efficient heuristics

In this section we show that effective and efficient heuristics can be developed for our special cases. We use the lower bounds given above to show that these heuristics are asymptotically optimal.

#### 6.1. A heuristic for equal processing times

In this section we use results from the complexity analysis of the problem to develop a heuristic for the equal processing time problem. First, let us assume that there is an even number of jobs. If we can obtain a sequence of jobs in which there is minimal idle time throughout the sequence, and the two jobs with the smallest setup times are sequenced as the first and the last jobs to achieve the lower bound which we introduced earlier, then indeed a good solution has been obtained.

The complexity analysis argument for this case described how to avoid idle times by selecting the positive differences, \(\delta_i\), first, followed by negative difference. In the complexity analysis argument we could assume processing times to be as large as we liked so all odd waiting times were positive and all idle times were avoided. However, this may not be possible for many equal processing time problems.

Nevertheless we can still use the argument of the complexity proof and the lower bounds to construct two effective heuristics.

**Proposition 9.** For the equal processing times and short setup times problem, the sequence with the two shortest jobs in position 1 and \(2k\) and the remainder in longest setup time order has a makespan at most \(\frac{1}{2} \sum_{i=2}^{k} (s_{2i-1} - s_{2i-2})\) above the minimum (where \(s_1 \leq s_2 \leq \cdots \leq s_{2k}\)). The sequence is asymptotically optimal. Furthermore, the sequence is optimal if there are even numbers of identical jobs.

**Proof.** We follow the argument in the complexity analysis proof. \(W_1 = (p - S_2)^+ = p - S_2 \geq 0\) so \(I_1 = 0\). \(W_2 = (p - S_3 - p + S_2)^+ = S_2 - S_3 = s_{2k} - s_{2k-1} \geq 0\) and \(I_2 = (S_3 - S_2)^+ = 0\). \(W_3 = (p - S_4 - S_2 + S_3)^+ = (p - S_2 + (S_3 - S_4)) \geq 0\) so \(I_3 = 0, \ldots\), \(W_{2k-1} = (p - S_{2k} - (S_2 - S_3) - (S_4 - S_5) \cdots - (S_{2k-2} - S_{2k-3})) \geq 0\) so \(I_{2k-1} = 0, \ldots\), and \(W_{2k} = (p - S_{2k+1} - (S_2 - S_3) - \cdots - (S_{2k-1} - S_{2k})) \geq 0\) so \(I_{2k} = 0, \ldots\). For more details, please refer to the original paper. \(\square\)
\[ S_{2k-1} = p - S_2 + (S_3 - S_4) + \cdots + (S_{2k-1} - S_{2k}) \geq 0 \text{ so } I_{2k-1} = 0. \text{ Now } I_{2k+1} = W_{2k} = p - (p - S_2 + (S_3 - S_4) + \cdots + (S_{2k-1} - S_{2k})) = s_2 + \sum_{i=2}^{k} (s_{2i} - s_{2i-1}) = s_2 + s_{2k} - s_3 + \sum_{i=2}^{k-1} (s_{2i} - s_{2i-1}) \leq s_2 + s_{2k} - s_3. \]

Also \( I_0 = s_1, \) so

\[
\text{makespan} \leq \frac{1}{2} \left( \sum_{i=1}^{n} a_i + s_1 + s_2 + s_{2k} - s_3 \right).
\]

Also \( LB \geq \frac{1}{2} np \) hence the heuristic is asymptotically optimal. If there are even numbers of identical jobs then \( \text{makespan} = \frac{1}{2}(s_1 + s_2 + \sum_{i=1}^{n} a_i) \leq LB \) and the result follows. \( \square \)

Generally, when we do not have even numbers of identical or near identical jobs the following heuristic is highly effective.

\textit{An }O(n \log n)\textit{ heuristic for equal processing time (even number of jobs)}

Step 1: Place the shortest job in position 1 and the second shortest job in position 2k.

Step 2: Partition the remaining \( 2k - 2 \) jobs into two sets \( G \) and \( H \) with \( k - 1 \) jobs each, so that the sums of their lengths are approximately equal (say, by sorting in decreasing order and allocating alternately to \( G \) and \( H \)).

Step 3: Sort each set in decreasing order and denote the job lengths \( g_i \) and \( h_i \).

Step 4: Let \( D = \{ \delta_i: \delta_i = g_i - h_i, i = 1, \ldots, k - 1 \} \), \( \delta^+_i = \max(\delta_i, 0) \) and \( \delta^-_i = \max(-\delta_i, 0) \) so \( \delta_i = \delta^+_i - \delta^-_i \). Let \( D^+ = \{ \delta^+_i: i = 1, \ldots, k - 1 \} \), \( D^- = \{ \delta^-_i: i = 1, \ldots, k - 1 \} \) Also let \( \Delta = 0 \). Sort \( D^+ \) and \( D^- \) in decreasing order.

Step 5: Find the largest \( \delta^+_i \), say \( \delta^+_1 \), and replace \( \Delta \) by \( \Delta + \delta^+_1 \), delete \( \delta^+_1 \) from \( D^+ \) and \( \delta^-_1 \) from \( D^- \), and schedule the related jobs \( g_1 \) and \( h_1 \) in the next two available positions, respectively.

Step 6: Find the largest \( \delta^-_1 \), say \( \delta^-_1 \). If \( \Delta - \delta^-_1 < 0 \) then go to step 7 otherwise go to step 8.

Step 7: Find the smallest \( \delta^+_i \), say \( \delta^+_m \) so that \( \Delta + \delta^+_m - \delta^-_i \geq 0 \). Replace \( \Delta \) by \( \Delta + \delta^+_m \), delete \( \delta^-_i \) and schedule the related jobs. If the condition cannot be met, select the largest \( \delta^+_i \) and add it to \( \Delta \), schedule the related jobs and repeat step 7 until the condition is met or there are no remaining \( \delta^+_i \).

Step 8: If \( \Delta - \delta^-_i \geq 0 \), position the related jobs \( (g_1 \text{ and } h_1) \), replace \( \Delta \) by \( \Delta - \delta^-_i \) and delete \( \delta^-_i \).

Step 9: Find the largest \( \delta^-_i \) so that \( \Delta - \delta^-_i \geq 0 \), if there is such a \( \delta^-_i \), sequence the related jobs in next positions and delete them. Repeat while there are still some \( \delta^-_i \) satisfying the condition.

Step 10: If either \( \{ \delta^-_i \} \) or \( \{ \delta^+_i \} \) are exhausted, schedule the remaining jobs as paired, otherwise go to step 5.

It is clear that Steps 2, 3 and 4 are \( O(n \log n) \) and Steps 6, 7 and 9 are \( O(\log n) \) (binary cuts). Also Steps 5 to 9 are executed \( O(n) \) times so overall the heuristic is \( O(n \log n) \).

\textit{An example for equal processing times}

Consider a set of twelve jobs with the following setups and a constant processing time equal to 100. Also without loss of generality, consider them to have been arranged in increasing order of setup times.

<table>
<thead>
<tr>
<th>Jobs</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
<th>11</th>
<th>12</th>
</tr>
</thead>
<tbody>
<tr>
<td>Setups</td>
<td>18</td>
<td>26</td>
<td>29</td>
<td>35</td>
<td>40</td>
<td>46</td>
<td>69</td>
<td>81</td>
<td>82</td>
<td>91</td>
<td>91</td>
<td>93</td>
</tr>
</tbody>
</table>
Assigning the first and final positions for the shortest and the second shortest setup jobs, we use a modified LPT (LPT with the limitation that the sets should have an equal number of jobs) to partition the set. We then obtain the following two sets:

\[ G = \{12, 9, 8, 5, 4\} \quad \text{and} \quad H = \{11, 10, 7, 6, 3\}. \]

Please note that since we used LPT, reordering of jobs in decreasing order is not necessary. The calculated values are presented in Table 2.

In step 1, the first job and the second job are set aside. The results of steps 2–4 appear in Table 2. By step 5, \( A = \delta_3^+ \) and the pair of \( \{8, 7\} \) will be selected. By steps 6 and 8 \( A = 12 - \delta_3^- = 12 - 9 = 3 \) and \( \{9, 10\} \) is selected. Then repeating step 5, \( A = 3 + \delta_5^- = 3 + 6 = 9 \) (pair \( \{4, 3\} \)). The \( A \) is reduced by selecting \( \{5, 6\} \) and finally jobs \( \{12, 11\} \) are selected. The final sequence is 1-8-7-9-10-4-3-5-6-12-11-2. The makespan achieved is 975.

6.2. Computational analysis

We examined the quality of this heuristic by studying its performance using three criteria: average performance, worst case performance and variation in performance.

Average performance may be considered as the most widely used measure, representing the expected performance of a heuristic. However, this might be unreliable if that heuristic also contains some extremely poor results. Thus, worst case analysis is another widely used measure.

We are unaware of any other heuristic for this particular problem, therefore, we have added a random sequence solution so that more meaningful diagrams can be presented.

We considered the following scenario. The number of jobs varied from 4 to 200, for each number of jobs we examined ten different problems. Processing times of jobs were equal to 100, while setup times were drawn from a uniform distribution U(0,100). Results are given in Fig. 3.

As is clear from Fig. 3, the average of the makespan/(lower bound) ratio converges rapidly to one with the increase in the number of jobs so that for about 20 jobs and more this ratio did not exceed its lower bound by more than 0.1 percent.

The largest deviation, in fact, occurs with four and six jobs. However, the first author has proved, in his doctoral dissertation, Abdekhodaee [10], that for four jobs this heuristic is optimal. (The argument is straightforward and is not reproduced here.) Also by examining the results of thirty
samples for six jobs and solving them by CPLEX, we observed that this heuristic also provided optimal solutions. Perhaps this was predictable due to the fact that the LPT scheduling rule provides optimal answers for four jobs.

The random sequence ratios also decrease as the number of jobs per sample increases, however, it seems that a steady state is reached after the number of jobs exceeds 100.

We also considered setups times drawn from U(50,100) and found that the performance of the heuristic did not deteriorate.

6.3. Required modification for an odd number of jobs

Since the previous heuristic appears to perform sufficiently well, we modified it only slightly for the problem with an odd number of jobs. We now aim at having the server waiting time which occurs prior to the last job equal to zero. We also want the longest job to be scheduled as soon as possible (Fig. 4).

For example if \( n = 5 \) we wish to have:

\[ W_1 = p - S_2, \]
In general, let $S_1 = s_1$, $S_{2k+1} = s_2$, $S_2 = s_{2k+1}$, $A = p - s_{2k+1}$ and schedule the remaining $2k - 2$ jobs using the heuristic (starting at Step 6) for an even number of jobs.

The modified heuristic was tested on randomly generated setup times. The number of jobs per problem ranged from 3 to 201 for each ten random problems considered. The results are summarised in Fig. 5.

6.4. Backward scheduling

It is interesting to note that the heuristic for equal processing times may be easily modified to be a backward scheduling one. Basically we again attempt to partition setups so that eventually they have almost equal sums.

In the previous heuristic we only presented a forward balancing mechanism in which a positive difference is depleted by later negative differences so that eventually the remaining positive difference at the end becomes small. Another procedure can be designed in exactly the reverse order so that a negative difference can be exhausted by positive differences.

6.5. A heuristic for equal setup times: the regular case

Equal setup times can be observed in practice. For example, the loading of a part on a machine may be independent of the particular job characteristics.
Compared with the case of equal processing times, the equal setup times case has been considered in a few papers. As has been mentioned earlier, Hall et al. [4] reported that this problem is NP-complete in the strong sense. Kravchenko and Werner [5] presented a pseudopolynomial time algorithm for the case when all setup times are equal to one. We first consider the problem in which setup times are equal and not necessarily one, but the set of jobs is regular. This might appear somewhat restrictive, however some modifications will be offered later to easily cope with a non-regular set.

Being a regular set of jobs with equal setup times case implies that $p_i + s \leq p_j + s$ for all $i, j$ which means that $p_i - p_j \leq s$. Depending on where the processing times are scheduled various possibilities might be considered. If all processing times are less than or equal $s$, then it is trivially solvable in polynomial time. (See the comments following Proposition 4.) If there are some long processing times we know by Proposition 5 that they form one sequence.

First we present an analogue of Proposition 9.

**Proposition 10.** For the equal setup and long processing times problem, the sequence with the shortest processing time job in position $2k$ and the longest processing time job in position $2k - 1$ and jobs in positions 1 and $2k - 2$ in shortest processing time order has a makespan at most

$$\frac{1}{2} \sum_{i=1}^{k-1} (p_{2i+1} - p_{2i}) + s + p_1 - p_2 + \sum_{i=1}^{2k} (p_{2i+1} - p_{2i}) \leq s + p_1 - p_2 + p_{2k}.$$

The sequence is optimal if there are even numbers of identical jobs. It is asymptotically optimal if processing times are bounded.

**Proof.** We repeat the argument in the proof of Proposition 9. $W_1 = (P_1 - s)^+ = P_1 - s \geq 0$ so $I_1 = 0$. $W_2 = (P_2 - (P_1 - s))^+ = P_2 - P_1 \geq 0$ and $I_2 = (P_1 - P_2)^+ = 0$. Now $I_0 = s$. Also $I_{2k+1} = W_{2k} + s + P_{2k} - P_{2k-1} + (P_2 - P_1) + \cdots + (P_{2k-2} - P_{2k-3}) = s + p_1 - P_{2k} + p_{2k-1} - p_{2k-2} + \cdots + p_3 - p_2 \leq s + p_1 - p_2 + \sum_{i=1}^{k-1} (p_{2i+1} - p_{2i}) \leq s + p_1 - p_2 + p_{2k} - p_{2k-1}$.

Now $LB = \frac{1}{2}(\sum_{i=1}^{n} a_i + s_1 + s_2 + p_{minimum} - p_{maximum})$. So, the sequence is optimal if there are even numbers of identical jobs. In any case, if processing times are bounded then the heuristic is asymptotically optimal.

We now present a general heuristic for long processing times.

For an even number of jobs, ideally, we should be able to put jobs in groups so that we not only avoid idle times but also the difference on completion of the final two jobs of each group, so to say the tail of the group, equals $s$. In this way it would be possible to join the blocks of jobs without creating idle time between groups, see Fig. 6. Further, we want the last two jobs in the sequence to be such that $s + P_n - P_{n-1}$ is minimised. If such a grouping of jobs, as above, were possible, then we could argue that the resulting sequence would be optimal.
The following special case suggests the heuristic structure. Assume, for now, that we have four jobs and that they are arranged so that there is no idle time in between and the final server waiting time is $s$. Then we have:

$$W_1 = P_1 - s,$$
$$W_2 = P_2 - s - (P_1 - s) = P_2 - P_1,$$
$$W_3 = P_3 - s - (P_2 - P_1) = P_3 + P_1 - s - P_2,$$
$$W_4 = P_4 - 0 - (P_3 + P_1 - s - P_2) = s \text{ so } P_4 - P_3 + P_2 - P_1 = 0.$$

If we continue for six jobs we would have $P_6 - P_5 + P_4 - P_3 + P_2 - P_1 = 0$. This is very similar to what has been achieved for equal processing times. In fact, the following heuristic is in the spirit of the heuristic presented for the equal processing time case. It uses the same compensatory mechanism but deals with processing times rather than setup times and uses different predesignated positions (the last two positions are predetermined here).

A heuristic for equal setup times and long processing times

**Step 1:** Place the job with the shortest processing time in position $2k$ and the job with the longest processing time in position $2k - 1$.

**Step 2:** Partition the remaining $2k - 2$ jobs into two sets $G$ and $H$ with $k - 1$ jobs each so that the sums of their processing time are approximately equal (For example, use LPT with the proviso that at no stage do the cardinalities of the two sets differ by more than 1.)

**Step 3:** (If necessary). Sort each of $G$ and $H$ in decreasing order of processing times, call the lengths $g_i$ and $h_i$.

**Step 4:** If the processing time of $g_i$ is less than the processing time of $h_i$, then swap the $g_i$ and $h_i$ jobs.

**Step 5:** Let $D = \{d_i: d_i = g_i - h_i\}$.

**Step 6:** Partition $D$ into sets $E$ and $F$ so that $\sum_{i \in E} d_i$ and $\sum_{j \in F} d_j$ are approximately equal. (Again, LPT may be applied this time with no restriction on the number of elements in $E$ and $F$.)

Set $\Delta = 0$

**Step 7:** Sort $E$ and $F$ in decreasing order of $d_i$. If $E = \phi$ then let $q^* = |F|$. If $F = \phi$ then let $\rho^* = |E|$. If $E = F = \phi$ then stop.

Otherwise let $\rho^* = \max\{\rho: \Delta + \sum_{i=1}^{\rho} e_i \leq f_1\}$ (If $\Delta + e_1 > f_1$ then set $\rho^* = 1$.)

Now find $q^* = \max\{q: \sum_{j=1}^{q} f_j \leq \sum_{i=1}^{\rho^*} e_i\}$ (If $f_1 > \sum_{i=1}^{\rho^*} e_i$ then set $q^* = 1$.)

**Step 8:** Assuming $e_i$ is related to $g'_i$ and $h'_i$ and $f_j$ is related to $g''_j$ and $h''_j$, allocate jobs to the first available positions thus: $P_1 = h'_1, P_2 = g'_1, \ldots, P_{2\rho^* - 1} = h''_{\rho^*}, P_{2\rho^*} = g''_{\rho^*}$ and $P_{2\rho^* + 1} = g''_{1}, P_{2\rho^* + 2} = h''_1, \ldots, P_{2\rho^* + 2q^*-1} = g''_{q^*}, P_{2\rho^* + 2q^*} = h''_q$. Let $\Delta = \sum_{i=1}^{\rho^*} e_i - \sum_{i=1}^{q^*} f_i$.

**Step 9:** Delete $\{e_1, \ldots, e_{\rho^*}\}$ from $E$ and $\{f_1, \ldots, f_{q^*}\}$ from $F$ and repeat the process until $E$ and $F$ are empty.

An example for equal setup times

Consider a set of 12 jobs with the following processing times and a constant setup time equal to 100. Also, without loss of generality, consider them to have been arranged in increasing order of processing times.
Assign the first job to the last position and the last job to the second last position. Using a modified LPT we have two sets as \( G = \{11, 8, 7, 4, 2\} \) and \( H = \{10, 9, 6, 5, 3\} \). Please note that since we used LPT, reordering of jobs in decreasing order is not necessary. So the following difference list can be calculated and sorted in the second section of Table 3.

Then, they are partitioned into sets \( E \) and \( F \), based on the difference list. The Table 4 will result from sorting \( E \) and \( F \).

The final sequence would be 6-7-9-8-10-11-3-2-5-4-12-1. The makespan is 1535 and the lower bound is 1532. Clearly, the outcome is very close to the optimal solution.

### 6.6. Computational analysis

Again, we examined the efficiency and effectiveness of our heuristic. From an efficiency point of view it is clear that this algorithm is \( O(n \log n) \) (and hence large size problems can be solved with our simple program within fractions of a second). But more important is its effectiveness. Our results show that it approached the lower bound as the number of jobs per sample increased.

From Fig. 7, it seems that the largest deviation of the makespan from the lower bound occurs with the lowest number of jobs per sample. Interestingly, however, it was observed that the heuristic provided optimal solutions for the cases of six or fewer jobs, which we solved using CPLEX.
6.7. Required modification for an odd number of jobs

An approach similar to the equal processing times problem with an odd number of jobs was followed for this problem. We schedule the second smallest processing time job in the last possible position (i.e. \((n - 2)\)th position), the initial balancing variable is set at \(\Delta = P_{n-2} - s\), and we used a backward heuristic.

The computational results show that the effectiveness of the heuristic does not change with these modifications and so the heuristic has, in fact, been extended to cover the case of an odd number of jobs.

6.8. A heuristic for equal setup times: the general case

The first modification is based on the idea that if some jobs in the optimal sequence are processed on the same machine and do not alternate between machines, then without loss of generality they can be assumed to be in SPT order. We show this in the following lemma.

**Lemma 3.** If an optimal schedule contains a sequence of jobs processed consecutively on the same machine then an SPT rearrangement of that sequence is also optimal.

**Proof.** We need only show that a rearrangement in SPT order does not harm the optimality of the solution. So firstly, it is obvious that a rearrangement of a subset of jobs being processed consecutively on one machine does not change the length of that subset. Second, if we want to
consider an instance of a general problem where jobs have been partitioned in two sets. Clearly, our assumption of regularity may not apply here because jobs may be matched with others which have short processing times so their arrangement creates idle times. Further, this appearance of idle times disrupts our compensatory mechanism.

One solution might be to insert jobs with short processing times whenever possible. See for example Fig. 9. Therefore, a mechanism for inserting jobs with short processing times has been considered in a heuristic below to tackle the general case.

We also develop a heuristic for the general case based on the earlier one, using the partitioning concept. We present here a sketch of the procedure, which has, in fact, been programmed. In simple terms, the heuristic consists of the following stages:

In stage 1, we decide on a final job group, for example, by selecting the job with the largest processing time and matching it with a group of short processing time jobs whose total length is equal to or slightly smaller than this largest processing time. (If the total is smaller, a suitable value for $\Delta$ could compensate for the difference in a backward heuristic.)

In stage 2, we partition the remaining jobs into sets $G$ and $H$, as described earlier.

In stage 3, we insert the jobs with $p_j < s$, from the end of the $G$ and $H$ lists, at appropriate locations, in order to minimise idle time. (See Fig. 9.)

In stage 4, we apply a modified version of the earlier heuristic to the groups of jobs generated in stages 1 to 3.
To elaborate on stage 3, consider jobs \( x \) and \( y \) in Fig. 9. Some jobs (from the same group as job \( y \)) whose total length is less than \( p_x - s \) can be inserted before job \( y \) on machine two. Short processing times have insertion priority as otherwise they produce idle times. On the other hand, some jobs (from the same group as job \( x \)) can be reinserted after job \( x \) if their total length is less than \( p_y - s \). The following example illustrates this procedure.

**An example for equal setup**

We present the important elements of the heuristic in the following example. Consider the set of jobs given below with setup times equal to 10.

<table>
<thead>
<tr>
<th>Job Number</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
<th>11</th>
<th>12</th>
<th>13</th>
<th>14</th>
<th>15</th>
<th>16</th>
</tr>
</thead>
<tbody>
<tr>
<td>Processing Time</td>
<td>3</td>
<td>4</td>
<td>4</td>
<td>4</td>
<td>5</td>
<td>5</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>8</td>
<td>8</td>
<td>30</td>
<td>40</td>
<td>46</td>
<td>55</td>
<td>70</td>
</tr>
</tbody>
</table>

At the first stage we decide on the final jobs, so one solution is matching the largest job with the short processing times jobs. Therefore, jobs \{1, 2, 3, 4, 5\} can be matched with job 16, let us call this group the final group.

In the second stage the remaining jobs are divided into sets \( G \) and \( H \) so that their lengths are nearly equal. Using LPT, set \( G \) includes jobs \{15, 12, 11, 9, 7\} and set \( H \) includes \{14, 13, 10, 8, 6\}.

In the third stage we insert short processing jobs from set \( H \) on machine 2 to follow job 15, from set \( G \), so that the total length of these inserted jobs does not exceed \( 55 - 10 = 45 \). Thus jobs 6 and 8 are inserted as shown in Fig. 10. Similarly, jobs 7 and 9 are inserted on machine 1 as shown.

Now we consider the jobs of Fig. 10 as two composite jobs \( g'_1 \) and \( h'_1 \), so:

\[
d_1 = g'_1 - h'_1 = \sum_{i \in g'_1} a_i - \sum_{i \in h'_1} a_i = 10.
\]

Insert the remaining short jobs, 10 and 11 as illustrated in Fig. 11 to form two new composite jobs \( g'_2 \) and \( h'_2 \). Call this the second group. We now have

\[
d_2 = g'_2 - h'_2 = \sum_{i \in g'_2} a_i - \sum_{i \in h'_2} a_i = -10.
\]

In the fourth stage we arrange the groups (other than the group of final jobs) formed so far. We use the following modified version of the earlier heuristic. Sort the positive and negative \( d_i \) values separately in decreasing absolute value order. Then sequence the job groups using negative and positive \( d_i \) groups alternately, starting with the group with the largest magnitude and negative \( d_i \) value. We thus obtain the solution: the second group, followed by the first group and finally the last.
That is we obtain the sequence \{12, 10, 13, 11, 15, 6, 8, 14, 7, 9, 16, 1, 2, 3, 4, 5\}. This sequence is, in fact, optimal since it has the minimal idle time of \( s \).

7. **Forward heuristic vs. backward scheduling**

We noted earlier that the heuristics for equal processing times and equal setup times can be applied both as forward and also as backward heuristics. That is, we can find a difference and compensate for the difference in a forward order or find a difference and compensate for it in a backward order. In translating the algorithms to a computer program, we observed that the backward heuristic is sometimes more convenient to use than the forward heuristic. This is because in both heuristics we first schedule one or two final jobs and then apply the compensating mechanism. As a result the final job creates a difference that somehow affects the makespan. In backward scheduling we can start by compensating for this difference and continue toward the start of the sequence. But in forward scheduling this difference should be ignored or additional programming is needed. This is especially the case with regular equal setups with an odd number of jobs.

These heuristics were expressed formally as forward procedures. But we observe that a backward heuristic is a very good alternative especially in the case of equal setup times.

8. **Some extensions of the special cases**

We can use the previous results to tackle some new problems in which jobs require not only to be setup by a server, but may also need to be unloaded from the machines. In other words, a server should setup a job on a machine, the machine then processes the job, and finally a server is required to unload the job. We assume that processing immediately follows loading (setup) and that neither the setup nor the processing procedure can be interrupted. Now we assume that unloading also cannot be interrupted once started but it can be delayed. This is assumed to avoid deadlock. However, unloading of a processed job has priority over a job to be loaded (setup) on the other machine.

Three special cases constitute simple extensions of earlier work:

(a) variable loading–fixed processing–fixed unloading.
(b) fixed loading–variable processing–fixed unloading.
(c) fixed loading–fixed processing–variable unloading.
By assuming that each unloading (except the first and last ones) is followed by a loading the above cases can be essentially reduced to the previously studied cases of equal processing times and equal setup times.

Cases (a) and (c) are very similar to the equal processing times problem except for the first and last jobs in a sequence and case (b) is similar to the equal setup time case. The previously developed heuristics may also be applied to these cases but some modification may be needed for the first and last jobs.

9. Conclusions

In this paper we mainly discussed two important special cases of equal processing and equal setup times. The equal processing time problem is always regular while the equal setup time can also be non-regular as well. We showed that the equal processing time problem with short setups and the regular equal setup time problem with long processing times are both NP-complete in the ordinary sense. We developed a powerful heuristic for equal processing times and examined the performance of the heuristic with randomly generated problems. Equal setup times were also examined and a similar, suitably modified, heuristic was proposed for the latter problem. We used the results for the regular case of equal setups to tackle the general problem. Further, we briefly discussed the issue of applying the heuristics in forward and backward versions.

References