On parallel machine scheduling problems with a single server

Svetlana A. Kravchenko

Institute of Engineering Cybernetics,
Surganova St. 6, 220012 Minsk, Belarus

Frank Werner

Otto-von-Guericke-Universität Magdeburg,
Postfach 4120, 39016 Magdeburg, Germany

Abstract - In this paper we consider the problem of scheduling jobs on parallel machines with setup times. The setup has to be performed by a single server. The objective is to minimize the schedule length (makespan) as well as the forced idle time. The makespan problem is known to be \( NP \)-hard even for the case of two identical parallel machines. This paper presents a pseudopolynomial algorithm for the case of two machines when all setup times are equal to one. We also show that the more general problem with an arbitrary number of machines is unary \( NP \)-hard and analyze some list scheduling heuristics for this problem. The problem of minimizing the forced idle time is known to be unary \( NP \)-hard for the case of two machines and arbitrary setup and processing times. We prove unary \( NP \)-hardness of this problem even for the case of constant setup times. Moreover, some polynomially solvable cases are given.

Keywords - production scheduling, parallel machine problems, setup times, \( NP \)-hard problems, polynomial algorithms, heuristics

\(^1\)This research was supported by the International Association for the Promotion of Cooperation with Scientists from the Independent States of the Former Soviet Union, Project INTAS-93-257, and by Deutsche Forschungsgemeinschaft, Project ScheMA. The first author was also supported by Belarussian Fundamental Research Foundation.
1 Introduction

This paper studies a deterministic scheduling environment on $m$ identical parallel machines with setup times. The problem under consideration can be described as follows. There are $m$ identical parallel machines $1,\ldots,m$ which must process $n$ given jobs of the set $\{1,\ldots,n\}$ available for processing at time zero. Each job has a known integer processing time $p_i$ and before its processing it requires to be loaded on a machine, i.e. it has a known loading time $s_i$. This loading, which is called a setup, is performed by a server. There is only one server serving all machines. Throughout the paper we suppose that travel times between machines are equal to zero. Having completed a setup, the server is free to perform other setups. Simultaneous requests by machines for the server will necessarily result in machine idle time, unless some setup times are zero. All values $s_i$ and $p_i$ are integer and known in advance. We note that zero processing times are not excluded and a job with a zero processing time can be interpreted as a job with such a small duration $\epsilon$ which can be disregarded. Each machine processes one job at a time and job preemption is not allowed. Often the objective is to sequence all jobs on the machines such that the maximum job completion time (makespan) is minimized. We denote this problem as $P, S_1|s_i|C_{\text{max}}$, where $S_1$ indicates a single server.

It is known from [2] that

- problem $P, S_1|s_i,p_i=1|C_{\text{max}}$ can be solved in $O(n)$ time,
- problem $P_2, S_1|s_i=1|C_{\text{max}}$ is binary $NP$-hard and
- problem $P_2, S_1|s_i=s|C_{\text{max}}$ is unary $NP$-hard.

Here $s_i = s$ indicates that all setup times are equal but their lengths may vary with the problem input.

Additionally, we consider the related problem $P, S_1|s_i|IT$, where the forced idle time or interference has to be minimized. Following [3], we define $IT$ as the amount of time in list scheduling when some machine is idle due to the unavailability of the server for setting up a job when needed, i.e. we disregard the idle time on a machine after all of its processing is completed but before the other machine completes its processing. The consideration of that idle time too would lead to the makespan problem. In [3] it has been shown that problem $P_2, S_1|s_i|IT$ is unary $NP$-hard. In that paper also local search and beam search algorithms have been given and compared.

The above problems have found applications in Flexible Manufacturing Systems, where a robot is shared among several pieces of equipment for tool change and part setup purposes.
There have been numerous studies of robots in different scheduling environments. A more detailed discussion of the literature on this topic can be found in [2, 3]. In [2] a complexity analysis for the described model under common scheduling objective functions has been presented. In the same paper two heuristics of makespan scheduling have also been analyzed.

Another related problem is the problem of concurrent resource scheduling, where the processing of some jobs requires the simultaneous use of more than one resource. Such a problem has been considered for instance by Dobson and Karmarkar [1] for the objective of minimizing the total weighted completion time. Sahney [4] considers also a problem with two parallel identical machines served by one server. In this problem the server must attend the machine and a fixed switching time incurs when the server moves between the machines. Each job is preallocated to a machine and the objective is to minimize the total job flow time. So Sahney’s problem can be considered as a single machine batching problem.

The remainder of the paper is the following. In Section 2 we present a pseudopolynomial algorithm for problem \( P_2, S_1 | s_i = 1 | C_{\text{max}} \). The existence of such an algorithm was indicated as an open question in [2]. In Section 3 we show that problem \( P, S_1 | s_i = 1 | C_{\text{max}} \) is unary NP-hard and analyze some list scheduling heuristics for this problem. In Section 4 we consider the \( IT \) criterion. We prove unary NP-hardness of problem \( P_2, S_1 | s_i = s | IT \) and we present some polynomially solvable cases.

## 2 Two-machine problem with setup times

First we transform problem \( P_2, S_1 | s_i = 1 | C_{\text{max}} \) into a parallel machine problem \( P_2 || C_{\text{max}} \) by including the setup time of each job into its processing time, i.e. the modified processing time \( p'_i \) of job \( i \) is equal to \( p_i + 1 \). This problem can be optimally solved by a well-known pseudopolynomial algorithm (see for instance [5]).

Now we want to transform the obtained schedule for problem \( P_2 || C_{\text{max}} \) into a feasible one for problem \( P_2, S_1 | s_i = 1 | C_{\text{max}} \) in such a way that at each time no more than one job is starting. Since the first unit of the modified processing times is used by the server to perform the setup on the corresponding machine, we would obtain an optimal schedule for problem \( P_2, S_1 | s_i = 1 | C_{\text{max}} \) by such a transformation. Otherwise, i.e. if the situation occurs that the proposed algorithm sequences two jobs so that they start at the same time, then we show that this situation is unavoidable, and it does not violate a lower bound for the optimal objective function value.

Let us start with an optimal schedule for problem \( P_2 || C_{\text{max}} \) using the modified processing
times. Initially we shift all jobs on the machine with the smaller machine load for one time unit to the right (if the machine loads are equal, a machine is chosen arbitrarily). Let $\tilde{C}_{\text{max}}$ be the makespan value of this schedule after the above shift operation.

It is clear that a lower bound for the optimal objective function value for problem $P_2, S_1 \mid s_i = 1 \mid C_{\text{max}}$ is given by $C_{\text{max}} = \max \{ n, \tilde{C}_{\text{max}} \}$ (note that in the original problem zero processing times are not excluded).

The following algorithm schedules one job at each step (except possibly the last performed step). The partial schedule obtained after step $i-1$ is denoted as $S_{i-1}$ and characterized by a profile $(T_{i-1}^1, T_{i-1}^2)$, where $T_{i-1}^j$ denotes the completion time of the job scheduled last on machine $j$ in $S_{i-1}$.

Suppose that some partial schedule $S_{i-1}$ has been obtained. The set $J_1(J_2)$ is the set of currently unscheduled jobs which are not contained in the partial schedule $S_{i-1}$ (at step 1 the initial set $J_1(J_2)$ is the set of jobs processed on machine 1 (machine 2) in the corresponding optimal schedule for problem $P_2 \parallel C_{\text{max}}$). Consider the machine $u$ with the smaller value of $T_{i-1}^u$ ($j \in \{1, 2\}$), i.e. $T_{i-1}^u = \min\{T_{i-1}^1, T_{i-1}^2\}$. A job $k$ with $1 \leq k \leq n$ is called feasible with respect to $S_{i-1}$, if this job belongs to set $J_u$ and if after scheduling this job on machine $u$ the inequality $|T_i^1 - T_i^2| \neq 0$ holds. $F_u$ is the set of feasible jobs on machine $u$ with respect to the current set $J_u$. In step $i$, the algorithm tries to schedule a feasible job $k$ on machine $u$ with a modified processing time as small as possible. More detailed, Algorithm $P_2, S_1 \mid s_i = 1 \mid C_{\text{max}}$ works as follows:

**Algorithm** $P_2, S_1 \mid s_i = 1 \mid C_{\text{max}}$

1. determine the sets $J_1$ and $J_2$ by means of an optimal schedule for problem $P_2 \parallel C_{\text{max}}$ with the modified processing times;
2. shift all jobs on the machine with the smaller machine load, say machine 2, for one time unit to the right;
3. $T_0^1 := 0; \ T_0^2 := 1$

While $J_1 \cup J_2 \neq \emptyset$ Do

Begin

4. determine machine $u$ with $T_{i-1}^u = \min\{T_{i-1}^1, T_{i-1}^2\}$ and let $v$ be the other machine;
5. determine the set $F_u \subseteq J_u$ of feasible jobs;

If $F_u \neq \emptyset$ Then

Begin

6. determine $k \in F_u$ the job $k$ with minimal $p'_k$;

End

While
If $|T_{i-1}^u - T_{i-1}^v| \neq 1$ and there is a set $Z$ of $p'_k$ unit jobs in $J_v$ Then

7. interchange the jobs of the set $Z$ and job $k$, and let now $k$ be an arbitrary job from $Z$;

8. include job $k$ into $S_{i-1}$ (which yields a partial schedule $S_i$ with $T_u^i := T_u^{i-1} + p'_k$ and $T_v^i := T_v^{i-1}$) and remove job $k$ from $J_u$.

End

Else (i.e. if $F_u = \emptyset$ holds)

Begin

9. let $k \in J_u$ be an arbitrary infeasible job;

10. interchange the sets of jobs $J_u$ and $J_v$ except job $k$;

11. determine the set $F_u \subseteq J_u$ of feasible jobs;

If $F_u = \emptyset$ Then

12. schedule job $k$ on machine $u$ and then alternatively the jobs of the sets $J_u \setminus \{k\}$ and $J_v$ on both machines at the earliest possible time → STOP!

Else

13. determine job $k \in F_u$ with minimal $p'_k$ and include it into $S_{i-1}$ (which yields a partial schedule $S_i$ with $T_u^i := T_u^{i-1} + p'_k$ and $T_v^i := T_v^{i-1}$) and remove job $k$ from $J_u$;

End

End.

To illustrate Algorithm $P2, S1 | s_i = 1 | C_{\text{max}}$, we consider the following example.

Example 1. Consider a problem with 6 jobs, 2 machines and the lengths $p'_1 = 2$, $p'_2 = p'_3 = 3$, $p'_4 = p'_5 = p'_6 = 1$. The algorithm begins with the schedule shown in Fig. 1(a). In step 1 the algorithm finds a job with minimal length among the jobs in $F_1$, which is job 2. Therefore, after step 1 we obtain the same partial schedule $S_1$ as in Fig. 1(a) with $J_1 = \{4, 5, 6\}$, $J_2 = \{1, 3\}$, $T_1^1 = 3$ and $T_2^1 = 1$. In step 2 we have $F_2 = \{3\}$ and the algorithm interchanges the sets of jobs $J_1 = \{3\}$ and $J_2 = \{4, 5, 6\}$. One of the unit time jobs, say job 4, is now scheduled on machine 2. The obtained schedule is shown in Fig. 1(b).

In step 3 job 1 is scheduled on machine 2 and in step 4 job 3 is scheduled on machine 1. Then in steps 5 and 6, jobs 5 and 6 are scheduled on machine 2, respectively. The final schedule is shown in Fig. 1(c).
Theorem 1 The above algorithm constructs an optimal schedule for problem $P_2, S1 \mid s_i = 1 \mid C_{\text{max}}$.

Proof: To prove the theorem, we show that the lower bound $\max\{u, \hat{C}_{\text{max}}\}$ of the optimal objective function value is not violated after each step of the algorithm. Let machines $u$ and $v$ be defined as in Algorithm $P2, S1 \mid s_i = 1 \mid C_{\text{max}}$. First we note that, if the situation $F_u \neq \emptyset$ occurs at some step, then the bound is obviously not violated by scheduling the next job.

We have only to prove that, if the situation $F_u = \emptyset$ occurs, then the lower bound is also not violated by scheduling the remaining jobs according to Algorithm $P2, S1 \mid s_i = 1 \mid C_{\text{max}}$. To prove this, we show that, if at some step $j$ both sets $J_1$ and $J_2$ contain only unfeasible jobs with respect to machine $u$ (i.e. if line 12 of the above algorithm applies), then

a) $J_1$ and $J_2$ contain only unit time jobs and

b) up to time $T^* = \min\{T_1^{j-1}, T_2^{j-1}\}$, the server was never idle and therefore the remaining jobs in $J_1$ and $J_2$ can be consecutively scheduled on the machines in an arbitrary way (for instance alternatively), i.e. the total required server time yields the makespan value of an optimal schedule.

First we prove a). Assume that in Algorithm $P2, S1 \mid s_i = 1 \mid C_{\text{max}}$ line 12 applies, i.e. at step $j$ no feasible job can be scheduled (see Fig. 2 with $u = 2$).

\[ T_1^{j-2} \quad T_2^{j-2} \quad T_1^j \]

\[ T_2^{j-1} \quad t \quad T_1^{j-1} \]

**Figure 2:** The case $|T_1^{j-1} - T_2^{j-1}| = t$ before step $j$
This means that the current sets $J_1$ and $J_2$ contain only jobs of equal modified processing times $t = |T^j_1 - T^j_2| \geq 1$. Assume that $t > 1$ holds. Then in step $j - 1$ a job of length $2t$ must have been scheduled, i.e. $|T^{j-2}_1 - T^{j-2}_2| = t > 1$ (otherwise we would have scheduled another job which would not be completed at time $T^{j-1}_1$). Repeating this argument, we get finally $|T^0_1 - T^0_2| > 1$ which contradicts the initial condition $|T^0_1 - T^0_2| = 1$. So $t = 1$ must hold.

Next we prove b). If line 12 of Algorithm $P2$, $s_1 = 1 \mid C_{\text{max}}$ applies, we have only unit time jobs in both sets $J_1$ and $J_2$. We show that up to time $T^*$ the server is never idle (due to part a), the remaining jobs in $J_1$ and $J_2$ can for instance alternatively be scheduled on the machines and so the total required server time yields the optimal makespan value). Assume that the server is idle for the last time after doing step $i \leq j - 1$ when job $k$ has been scheduled, w.l.o.g. we assume that job $k$ has been scheduled on machine 1. This situation is illustrated in Fig. 3 with $l \leq j - 1$.

Let $x = |T^i_2 - T^i_1|$. Note that $x > 1$ must hold, that $J_1$ cannot contain unit time jobs at step $i$, and that $J_2$ contains unit time jobs otherwise line 12 will never occur. We show now that at each following step only the set $J_s$ for one machine can contain unit time jobs and therefore line 12 cannot occur. If after step $i$, the condition $|T^i_1 - T^i_2| \neq 1$ holds, we schedule unit time jobs on machine 2 until at some step $l$ the equality $T^l_1 - T^l_2 = 1$ holds. We now still must have unit time jobs in the current set $J_2$ since we did not use lines 7 and 10. Note that the number of unit time jobs in $J_2$ is at most equal to $x$, otherwise we would replace job $k$ in step $i$ by unit time jobs (line 7). Note also that $J_1$ contains only jobs of length $x$ or larger than $p'_k$. Therefore, if a job scheduled on machine 1 will be replaced by unit time jobs, this can only be a job of length $x$, otherwise this contradicts to scheduling job $k$ in step $i$ as in Fig. 3. However, in this case there remain only unit time jobs on machine 1 to be scheduled and not on machine 2. Therefore, after an idle time of the server, unit time jobs can be only in $J_1$ or in

\[ \text{Figure 3: The server is idle in } [T^i_1 + 1, T^i_2] \]
$J_2$, but not in both sets, and line 12 will never occur. Thus, line 12 of the proposed algorithm can be only performed if the server is not idle up to time $T^*$. \hfill \square

We now estimate the complexity of the algorithm. To transform an optimal schedule for problem $P2 \parallel C_{\text{max}}$ by Algorithm $P2, S1 \mid s_i = 1 \mid C_{\text{max}}$ we need $O(n \log n)$ time. To see that this complexity is sufficient, we note that within the algorithm one can consider the sets $J_k$ ($k = 1, 2$) as a list of pairs. The first value of each pair is the job length and the second value is the number of jobs with such a length. Within this list all pairs are arranged in non-decreasing order of the $p_i$-values. Hence, problem $P2, S1 \mid s_i = 1 \mid C_{\text{max}}$ can be solved in pseudopolynomial time.

3 The makespan problem with an arbitrary number of machines

In this section we show that problem $P, S1 \mid s_i = 1 \mid C_{\text{max}}$ is unary NP-hard and analyze several list scheduling heuristics for this problem.

First we give an NP-hardness proof for problem $P, S1 \mid s_i = 1 \mid C_{\text{max}}$, where the number of machines is arbitrary.

**Theorem 2** Problem $P, S1 \mid s_i = 1 \mid C_{\text{max}}$ is unary NP-hard.

**Proof:** Consider the problem 3-partition: given a set of integers $A = \{a_1, \ldots, a_{3m}\}$ with
\[
\sum_{i=1}^{3m} a_i = mB, \quad B/4 < a_i < B/2 \quad \text{for} \quad 1 \leq i \leq 3m.
\]

Does there exist a partition of the set $A$ into $m$ disjoint 3-element sets $A_1, \ldots, A_m$ such that for each $j$ the equality $\sum_{a_i \in A_j} a_i = B$ holds?

Consider the following instance of problem $P, S1 \mid s_i = 1 \mid C_{\text{max}}$ with $3m$ jobs and $m$ machines, where $p_i = ma_i - 1$, $s_i = 1$ for $i = 1, \ldots, 3m$. The decision version of problem $P, S1 \mid s_i = 1 \mid C_{\text{max}}$ is as follows: does there exist a schedule with an objective function value $C_{\text{max}} \leq mB + m - 1$?

First of all we show that we can handle the given instance just like a problem with parallel machines, where each machine $i$ is available for processing at time $i - 1$, i.e. any schedule has no forced idle times when the machines are available for processing.
Assume that this is not true, i.e., there is some schedule for the latter parallel machine problem, where each machine $i$ is available only at time $i - 1$, and there are at least two machines, say $k$ and $h$, $k < h$, which finish some jobs at time $t$, i.e., in the intervals $[k - 1, t]$ and $[h - 1, t]$ exactly a certain number of jobs is fully processed on the machines $k$ and $h$, respectively. It means that the length of the interval $[k - 1, t]$ as well as the length of the interval $[h - 1, t]$ are divisible by $m$ since each modified processing time is divisible by $m$. Hence, the difference of the interval lengths $(t - k + 1) - (t - h + 1) = h - k$ is divisible by $m$, too. But this is impossible since $h < m$ and $k < m$ hold. Therefore, in the remaining part of the proof we assume that any schedule has no forced idle times (after time $i - 1$ on machine $i$).

Now we show that, if problem 3-partition has a solution, then a schedule with $C_{max} = mB + m - 1$ can be constructed. Really, assume that we know a required partition $A_1, \ldots, A_m$. Then we schedule the jobs of the considered instance in the following way: all three jobs with the length $p'_i = ma_i$, where $a_i \in A_j$, are scheduled one by one in the interval $[j - 1, mB + j - 1]$ on machine $j$ with $1 \leq j \leq m$. Thus, we have constructed a feasible schedule with $C_{max} = mB + m - 1$.

Now we show that, if there is a schedule with $C_{max} \leq mB + m - 1$, then problem 3-partition has a solution. First we show that in this case each machine must process exactly three jobs. Suppose some machine $k$ with $1 \leq k \leq m$ processes more than three jobs. Then the last job on this machine can be completed only at time $t > mB + k - 1$. However, since the sum of the lengths of the jobs processed on machine $k$ is divisible by $m$, the inequality $t \geq m(B+1)+k-1 = mB + m - 1 + k$ must hold. The latter value is larger than $mB + m - 1$ since $k \geq 1$ holds. Therefore, each machine cannot process more than three jobs if $C_{max} \leq mB + m - 1$ holds. Taking into account that the number of jobs is equal to $3m$, we conclude that each machine processes exactly three jobs.

Now we show that each machine is loaded for exactly $mB$ time units. Suppose that some machine is loaded for more than $mB$ time units. Since the machine load for each machine must be divisible by $m$, we conclude that this machine is loaded for at least $m(B + 1)$ time units, which is larger than $C_{max} = mB + m - 1$. The obtained contradiction shows that each machine is loaded for exactly $mB$ time units, and this schedule gives the required partition.

Next we analyze two simple list scheduling heuristics. The list scheduling philosophy is to balance the work load among parallel machines which tries to yield smooth production schedules. A list schedule is described by a permutation $\pi = (\pi_1, \ldots, \pi_n)$ of the jobs which describes the sequence in which the server performs the setups for the jobs. When considering job $\pi_i$, we
look which machine becomes free first, and then we assign job $\pi_i$ to this machine and perform the setup by the server as early as possible.

In [2] the following results have been given:

a) The LPT list scheduling heuristic for problem $P, S_1 | s_i | C_{max}$ has a worst-case performance ratio of $2 - 1/m$.

b) An arbitrary list scheduling heuristic for problem $P, S_1 | s_i | C_{max}$ has a worst-case performance ratio of $3 - 2/m$.

Clearly, the same is true for problem $P, S_1 | s_i = 1 | C_{max}$, and we show that the above bounds are tight even for the case of unit setup times. First, we consider the LPT heuristic and show that the bound $2 - 1/m$ is tight for problem $P, S_1 | s_i = 1 | C_{max}$.

Consider the following instance of problem $P, S_1 | s_i = 1 | C_{max}$ with $mk - 1$ jobs, where $k > m$. There are $m(m - 1)$ jobs with lengths

$$p'_1 = p'_2 = \ldots = p'_{m(m-1)} = k$$

and $m(k - m) + m - 1$ jobs with unit lengths

$$p'_{m(m-1)+1} = \ldots = p'_{mk-1} = 1.$$  

Then $\pi^L = (1, 2, \ldots, mk - 1)$ is an LPT list schedule (i.e. all jobs are scheduled in the order of non-increasing lengths) with the objective function value

$$C^L_{max} = 2mk - k - m^2 + m - 1.$$ 

On the other hand, an optimal schedule with $C^*_{max} = mk - 1$ is obtained when all jobs are scheduled by the following list scheduling procedure $LISTA$.

**Procedure $LISTA$**

1. $i := 1$; 
   
   **While** $i \leq m$ **Do** 
   
   Begin 
   
   2. schedule $m - 1$ jobs with the length $k$; 
   3. schedule $k - m$ jobs with the length 1; 
   4. $i := i + 1$ 
   
   **End**; 
   
   5. schedule $m - 1$ jobs with the length 1.
To illustrate, we consider the case $m = 3$ and $k = 5$. Then we have $p'_1 = p'_2 = p'_3 = p'_4 = p'_5 = p'_6 = 5$ and $p'_7 = p'_8 = p'_9 = p'_{10} = p'_{11} = p'_{12} = p'_{13} = p'_{14} = 1$.

A schedule of all jobs determined by procedure $LISTA$, which corresponds to the list $\pi^* = (1, 2, 7, 8, 3, 4, 9, 10, 5, 6, 11, 12, 13, 14)$ is shown in Fig. 4 (the dashed parts illustrate the setup times).

\begin{figure}[h]
\centering
\begin{tabular}{cccccc}
1 & 4 & 11 & 12 & 13 & 14 \\
2 & 9 & 10 & 5 \\
7 & 8 & 3 & 6 \\
\end{tabular}
\caption{A schedule according to procedure $LISTA$}
\end{figure}

Therefore, we have

$$C^L_{\max}/C^*_\max = (2mk - k - m^2 + m - 1)/(mk - 1)$$

which tends to $2 - 1/m$ for $k \to \infty$.

Now consider an arbitrary heuristic and show that the bound $3 - 2/m$ is tight for problem $P, S1 \mid s_i = 1 \mid C_{\max}$. We consider an instance with $(k - m + 1)m + (m - 2)m + 1$ jobs, namely $(k - m + 1)m$ jobs numbered as $1, ..., (k - m + 1)m$ with length 1, $(m - 2)m$ jobs numbered as $(k - m + 1)m + 1, ..., (k - m + 1)m + (m - 2)m$ with length $k$, where $k > m$, and one job $(k - m + 1)m + (m - 2)m + 1$ with length $mk - 1$.

An optimal schedule with $C^*_\max = mk - 1$ is obtained when all jobs are scheduled by the following list scheduling procedure $LISTB$.

\begin{algorithm}
1. $i := 1$;
2. schedule the job with length $mk - 1$;
\begin{algorithmic}
\While {$i \leq m$}
\State{Begin}
\State{3. schedule $m - 2$ jobs with the length $k$;}
\State{4. schedule $k - m + 1$ jobs with the length 1;}
\State{5. $i := i + 1$}
\End
\end{algorithmic}
\end{algorithm}
For an instance with \( m = 4 \) and \( k = 5 \) the above procedure determines the schedule shown in Fig. 5 which corresponds to the list \( \pi^* = (17, 9, 10, 1, 2, 11, 12, 3, 4, 13, 14, 5, 6, 15, 16, 7, 8) \).

![Figure 5: A schedule according to procedure LISTB](image)

If we schedule the jobs in an arbitrary way, we can schedule at first all unit time jobs, then all jobs with the length \( k \) and then the jobs with the length \( mk - 1 \). As a result, we obtain a schedule with the makespan value \( \tilde{C}_{max} = 3km - 2k - 1 - m^2 + m \). So, we get

\[
\frac{\tilde{C}_{max}}{C_{max}^*} = \frac{(3km - 2k - 1 - m^2 + m)}{(mk - 1)}
\]

which tends to \( 3 - \frac{2}{m} \) for \( k \to \infty \).

The above tightness proofs of both bounds given in a) and b) show that from the point of a worst case analysis the problem with unit setup times is not easier than the problem with arbitrary setup times in the case of LPT and arbitrary list scheduling.

4 The forced idle time criterion

In this section we deal with the \( IT \) criterion, where \( IT \) is the forced idle time. First of all we show that, unlike problem \( P2, S1 | s_i = 1 | C_{max} \), problem \( P2, S1 | s_i = 1 | IT \) can be solved in \( O(n \log n) \) time. For this purpose we divide the set of all jobs into two subsets \( A = \{ i | s_i = 1, p_i = 0 \} \) and \( B = \{ i | s_i = 1, p_i \neq 0 \} \) and arrange all jobs from \( B \) in non-decreasing order of their \( p_i \)-values. Then we use the following algorithm \( P2, S1 | s_i = 1 | IT \), where \( T_1 \) and \( T_2 \) again describe the profile of the current partial schedule, i.e. \( T_j \) denotes the completion time of the job scheduled last on machine \( j \) in the current partial schedule.

**Algorithm** \( P2, S1 | s_i = 1 | IT \)

1. \( T_1 := 0; T_2 := 1; \) determine set \( A \) and the ordered set \( B \);
While $A \cup B \neq \emptyset$ Do

Begin

2. determine machine $u$ with $T_u = \min\{T_1, T_2\}$;

3. If $(|T_1 - T_2| = 1$ and $B \neq \emptyset)$ or $A = \emptyset$ Then

   schedule the next job from set $B$ on machine $u$

Else schedule an arbitrary job from set $A$ on machine $u$;

4. actualize $T_u$ and remove the scheduled job from the corresponding set

End.

Theorem 3 The above algorithm determines an optimal schedule for problem $P2, S1 \mid s_i = 1 \mid IT$.

Proof: After scheduling the first job, we obviously have $IT = 0$. After scheduling two jobs by Algorithm $P2, S1 \mid s_i = 1 \mid IT$, we have $IT = 1$. Suppose that after scheduling $k$ jobs $IT = 1$ holds. If we have at the current step $|T_1 - T_2| \neq 1$, then scheduling an arbitrary job does not change the objective function value $IT$. Moreover, since the jobs of $B$ are scheduled in non-decreasing order of their processing times, we always have $|T_1 - T_2| \neq 0$ after scheduling a job from the set $B$. Suppose now that $|T_1 - T_2| = 1$. If we schedule a job from set $B$, we still have $IT = 1$ and $|T_1 - T_2| \neq 0$ holds after scheduling this job. If the set $B$ is empty, then we schedule all jobs from the set $A$ consecutively. As a result, we obtain $IT = n - 1 - \sum_{i \in B} p_i$ which is obviously a lower bound for the optimal objective function value.

Now we prove that problem $P2, S1 \mid s_i = s \mid IT$ is unary NP-hard like the corresponding problem $P2, S1 \mid s_i = s \mid C_{\text{max}}$. This results strengthens a recent result by Koulamas [3], who proved unary NP-hardness only for the case of arbitrary setup times.

Theorem 4 Problem $P2, S1 \mid s_i = s \mid IT$ is unary NP-hard.

Proof: Consider the problem 3-partition: given a set of integers $A = \{a_1, \ldots, a_{3k}\}$ with

$$\sum_{i=1}^{3k} a_i = kB, \ B/4 < a_i < B/2 \quad \text{for} \quad 1 \leq i \leq 3k.$$  

Does there exist a partition of $A$ into $k$ disjoint 3-element sets $A_1, A_2, \ldots, A_k$ such that for each $j$ the equality $\sum_{a_i \in A_j} a_i = B$ holds?

Consider the following instance of problem $P2, S1 \mid s_i = s \mid IT$ with $n = 4k + 2$ jobs partitioned into three subsets.
The first subset $D$ contains $3k$ jobs $i, i = 1, \ldots, 3k$, with $s_i = B$ and $p_i = a_i$.
The second subset $E$ is a collection of $k$ jobs $i, i = 3k + 1, \ldots, 4k$, with $s_i = B$ and $p_i = 5B$.
The third subset $F$ consists of two jobs $i, i = 4k + 1, 4k + 2$, with $s_i = B$ and $p_i = B - \delta$, where $\delta < B/4$. The decision version of problem $P2, S1 \mid s_i = s \mid IT$ is as follows: does there exist a schedule with the objective function value $IT \leq B$.

First we show that, if problem 3-partition has a solution, then a schedule with $IT = B$ can be constructed. Since 3-partition has a solution, we know all sets $A_1, A_2, \ldots, A_k$. Then we schedule the jobs by the following procedure $SCHEDULE$, where each selected job is scheduled on the machine that becomes free first.

**Procedure $SCHEDULE$**

1. $i := 1$
   
   **While** $i \leq k$ **Do**
   
   Begin
   2. schedule a job $i$ from the set $E$;
   3. delete $i$ from the set $E$;
   4. schedule all three jobs from the set $A_i$
   End
   5. schedule both jobs from the set $F$.

To illustrate procedure $SCHEDULE$, we give in Fig. 6 the resulting schedule for $n = 10$ jobs, i.e. we have $k = 2$.

![Figure 6: The case $k = 2$](image)

It is easy to see that, if problem 3-partition has a solution, then a schedule obtained by the above procedure has the objective function value $IT = B$.

Now we show that, if there is a schedule with $IT \leq B$, then problem 3-partition has a solution. First we prove that in the case $IT \leq B$, both jobs from the set $F$ must be scheduled last.
It is immediately clear that in the case of $IT \leq B$, the first served job must be some job from $E$. Let $L$ denote the set of all jobs except the two jobs served last. Since scheduling the second job already leads to $IT = B$, all jobs from the set $L$ must be processed without idle time between them. First observe that each job from the set $\{D \cup F\} \cap L$ cannot overlap the setup part of some job from $E$, i.e. for each job from the set $\{D \cup F\} \cap L$, its setup and processing interval must be fully covered by the processing interval of some job from the set $E$, and the setup interval of any job from $E$ (except the first served job) must be fully covered by the processing interval of some job from $E$.

The sum of the setup times of the jobs in $E$ is equal to $kB$ and the sum of the processing times of the jobs in $E$ is equal to $5kB$. The sum of the modified processing times of the jobs of the set $D$ is equal to $4kB$ and the sum of the modified processing times of the jobs in the set $F$ is equal to $4B - 2\delta$. Since the total processing part of the jobs of set $E$ must cover the total setup part of the jobs of set $E$ (without the first served interval) and the total time for the jobs of the sets $D \cap L$ and $F \cap L$, the value $5kB$ must exceed $(k-1)B + 4kB + 4B - 2\delta - \Delta$ by $B$, where $\Delta \in \{2B + a_i + a_j, 3B - \delta + a_i, 4B - 2\delta\}$ is the sum of the lengths of both last served jobs (notice that both last served jobs must belong to set $D$ or to set $F$), i.e. we must have $5kB - B \geq 5kB + 3B - 2\delta - \Delta$ and therefore $2\delta + \Delta \geq 4B$.

Now we consider all possible cases for $\Delta$. If $\Delta = 2B + a_i + a_j$ (i.e. both last served jobs are from the set $D$), then by a direct substitution we obtain $2\delta + a_i + a_j \geq 2B$ which is not possible since $\delta < B/4$ and $a_i < B/2$ for all $i$ with $1 \leq i \leq 3k$. If $\Delta = 3B - \delta + a_i$ (i.e. one of the two last served jobs is from the set $D$ and the other one from the set $F$), then we obtain $\delta + a_i \geq B$, which is also not possible again due to $\delta \leq B/4$ and $a_i \leq B/2$. If $\Delta = 4B - 2\delta$ (i.e. both last served jobs are from the set $F$), then we have $2\delta + 4B - 2\delta \geq 4B$. Therefore, the only possibility for the two last served jobs is to belong to set $F$. At the same time, since the last inequality is a strict equality, we can assert that any two jobs from the set $E$ do not overlap each other in their processing parts, i.e. if $IT = B$ then a job from $E$ is overlapped exactly by an interval of the length $B$. So, for processing all remaining jobs of set $D$, we have $k$ intervals, where each of these intervals has the length $4B$. Therefore, if we have a schedule with $IT \leq B$, then each of these $k$ intervals must be filled by three jobs from the set $D$, which are defined by a set $A_j$ with $1 \leq j \leq k$.

Now we consider some polynomially solvable cases of problem $P2, S1 \mid s_i = s \mid IT$. First we consider the case when all processing times are larger than the constant setup time ($p_i > s$).
Proposition 1  Problem $P_2, S_1 \mid s_i = s, p_i > s \mid IT$ can be solved in $O(n \log n)$ time by the SPT (shortest processing time) rule.

Proof: We show that in this case $IT = s$ holds. Assume that all jobs are numbered in non-decreasing order of their $p_i$-values. It is clear that, if $p_i \geq p_{i-1}$ holds, then $s + p_i \geq p_{i-1} + s$ holds. Therefore, if the setup interval for job $i$ is fully covered by the processing interval of job $i-1$, then the processing interval of job $i$ fully covers the setup interval for job $i+1$, see Fig. 7.

Figure 7: The setup interval for job $i$ is covered by the processing interval of job $i - 1$

Now consider the case with constant setup times and $p_i < s$ for all $i$.

Proposition 2  Problem $P_2, S_1 \mid s_i = s, p_i < s \mid IT$ can be solved in $O(n)$ time.

Proof: Consider an arbitrary list schedule $\pi = (\pi_1, \ldots, \pi_n)$. Obviously, after scheduling the first job $IT = 0$ holds. After scheduling the second job we have $IT = s$, after scheduling the third job we get $IT = s + (s - p_{\pi_1})$ and so on. Thus, after scheduling the last job $IT = s + (s - p_{\pi_1}) + (s - p_{\pi_2}) + \ldots + (s - p_{\pi_{n-2}})$ holds, i.e. we have $IT = (n - 1)s - \sum_{i=1}^{n-2} p_{\pi_i}$. This value is minimal if and only if $\sum_{i=1}^{n-2} p_{\pi_i}$ is maximal. Thus, it is only necessary to know the jobs with the two smallest processing times to determine an optimal solution and problem $P_2, S_1 \mid s_i = s, p_i < s \mid IT$ can be solved in $O(n)$ time.

References


