ELSEVIER

Operations Research Letters 29 (2001) 231–239

operations research letters

www.elsevier.com/locate/dsw

List scheduling in a parallel machine environment with precedence constraints and setup times

Johann Hurink[a,*], Sigrid Knust[b,1]

[a]*Faculty of Mathematical Sciences, University of Twente, P.O. Box 217, NL-7500 AE Enschede, The Netherlands*
[b]*Fachbereich Mathematik/Informatik, Universität Osnabrück, D-49069 Osnabrück, Germany*

Received 2 May 2000; received in revised form 1 September 2001; accepted 20 September 2001

Abstract

We present complexity results which have influence on the strength of list scheduling in a parallel machine environment where additional precedence constraints and sequence-dependent setup times are given and the makespan has to be minimized. We show that contrary to various other scheduling problems, in this environment a set of dominant schedules cannot be calculated efficiently with list scheduling techniques. © 2002 Elsevier Science B.V. All rights reserved.

MSC: 90B35

Keywords: Scheduling; List scheduling; Complexity; Parallel machines; Setup times

1. Introduction

List scheduling is a widely used concept in the scheduling area. Basically, a list scheduling algorithm is a routine which calculates for a given order of jobs (which may be given by a list) a corresponding schedule. Mostly, such routines consider the jobs one by one in the given order and take a scheduling decision on the base of the partial schedule given by the previously scheduled jobs. Decisions for formerly scheduled jobs are not changed again.

On the one hand, several polynomial algorithms and approximation heuristics are based on list scheduling algorithms. For these methods, mostly only one list of jobs is considered and the outcome of the list scheduling algorithm for this list will be the outcome of the method. One of the most famous polynomial algorithms of this type is Smith's rule [9] for problem $1||\sum w_jC_j$ (all problems in this paper are denoted using the well-known $\alpha|\beta|\gamma$-notation of Graham et al. [4]), which sorts the jobs according to non-increasing w_j/p_j values and schedules the jobs in this order on the machine. Approximation heuristics of this type, e.g., are given by Baker [1] and Graham [3]. They also apply the list scheduling routine to one list, but in general, the outcome is not optimal for the considered problem (in the literature, such approaches are also called static priority rules; see e.g. [6]).

* Corresponding author. Tel.: +31-534893447; fax: +31-534894858.
E-mail addresses: j.l.hurink@math.utwente.nl (J. Hurink), sigrid@mathematik.uni-osnabrueck.de (S. Knust).
[1] Supported by the Deutsche Forschungsgemeinschaft, Project 'Komplexe Maschinen-Schedulingprobleme'.

On the other hand, list schedules may form the base of branch and bound methods or local search algorithms: if the set of all schedules achieved by applying an efficient list scheduling algorithm to all possible sequences of the jobs is a dominant set (i.e. the set contains at least one optimal solution), we can restrict our considerations to these schedules and use the set of all possible job sequences as solution space. Woerlee [11] has used this concept for developing a branch-and-bound method for problem $P|r_j|L_{\max}$ and Schutten and Leussink [8] did the same for problem $P|r_j, s_j|L_{\max}$.

In this paper, we present some complexity results which restrict the use of list scheduling for solving a parallel machine scheduling problem where additionally precedence constraints and sequence-dependent setup times are given and the makespan has to be minimized. This problem is denoted by $P|\text{prec}, s_{ij}|C_{\max}$ and can be stated as follows: given are n jobs with processing times $p_1, \dots, p_n$ which have to be processed on m parallel machines without preemption respecting a given set of precedence constraints. Furthermore, if jobs i and j are processed consecutively on the same machine, a setup of length s_{ij} has to be done on the machine between the two jobs. The goal is to minimize the makespan, i.e. the maximal completion time of a job. The problem is NP-hard in the strong sense since it generalizes the single-machine problem $1|s_{ij}|C_{\max}$ (traveling salesman problem) and the classical parallel machine problem $P||C_{\max}$. Some other complexity results for scheduling problems with sequence-dependent setup times can be found in Monma and Potts [5].

In the remaining of this paper, we deal with the question whether it is possible to design an efficient list scheduling algorithm for problem $P|\text{prec}, s_{ij}|C_{\max}$ which produces a dominant set of list schedules if it is applied to all sequences of jobs which are compatible with the given precedences (i.e. are linear extensions of the partial order induced by the precedences). A positive answer to this question could lead to a solution approach for the considered problem by using the set of all possible job sequences as solution space and the developed method to generate corresponding schedules. However, in this paper we will show that a positive answer to this question is very unlikely.

The paper is organized as follows. In Section 2, we review different versions of common list scheduling algorithms for parallel machine problems. In Section 3, we consider the problem $P|s_{ij}|C_{\max}$ without additional precedence constraints, but with a given job sequence π. We show that for an arbitrary number of machines, the problem of finding a best schedule in which job π_j does not start its execution earlier than job π_i for all $i < j$ is strongly NP-hard and that the problem remains ordinary NP-hard for a fixed number m of machines (even for $m = 2$). A pseudo-polynomial algorithm for problem $Pm|s_{ij}|C_{\max}$ with a given starting time order π is presented in Section 4. Some consequences of these results for the possibilities of using list scheduling algorithms for the considered problem are discussed in Section 5. The paper ends with some concluding remarks.

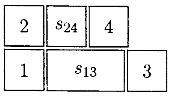
2. List scheduling algorithms

As mentioned in the Introduction, the concept of list scheduling has been widely used in the scheduling area. In order to apply this concept successfully, an efficient list scheduling algorithm has to be designed which produces a dominant set of schedules (a set of schedules is called *dominant* if it contains at least one optimal schedule). In this section, we focus on different versions of parallel machine problems and consider some possibilities for list scheduling algorithms.

Given a list of all jobs, a standard list scheduling algorithm constructs a schedule for the parallel machine problem $P||C_{\max}$ as follows: schedule the next job of the list on a machine which is available first (i.e. where the job starts its processing as early as possible). Obviously, this list scheduling algorithm is polynomial and the set of all schedules obtained in this way is a dominant set. If in addition precedence constraints are given, only lists which are compatible with the precedences are considered (i.e. if $i \to j$ holds, i is placed before j in the list). It is easy to see that the described list scheduling algorithm still produces a dominant set of schedules. If, on the other hand, sequence-dependent setup times s_{ij} are considered, Schutten [7] has shown that the list scheduling algorithm also produces a dominant set for problem $P|s_{ij}|C_{\max}$ if in each step the considered job is processed on a machine where it starts its processing (not its setup) as early as possible. However, in the case where setup times and precedence constraints are given, the result of Schutten cannot be generalized as the following example shows.

Example. Given are 2 machines and 4 jobs with unit processing times. The setup times are given by

$$s = (s_{ij})_{i,j=1,\dots,4} = \begin{pmatrix} 0 & 10 & 2 & 10 \\ 10 & 0 & 0 & 1 \\ 10 & 10 & 0 & 10 \\ 10 & 10 & 10 & 0 \end{pmatrix}.$$



The optimal solution is achieved by scheduling jobs 1 and 3 on one machine in this order and jobs 2 and 4 on the other machine in that order (see the above figure). This solution can be calculated by the list scheduling algorithm using the sequence $\pi=(1,2,4,3)$ or $\pi=(2,1,4,3)$. All other sequences lead to schedules with larger makespans.

If we now add a precedence constraint $3 \to 4$, the structure of the optimal solution remains the same (same machine assignment and same sequences on the machines, only job 4 has to start 2 units later). However, since the sequences $\pi=(1,2,4,3)$ and $\pi=(2,1,4,3)$ are no longer compatible with precedence constraints, a standard list scheduling procedure will not consider these two sequences and, thus, the set of all list schedules does not contain the optimal solution anymore (even if one would apply the list scheduling algorithm to the two sequences, the set of list schedules is no longer dominant, since the solutions resulting from the two sequences are infeasible).

This example shows that for problem $P|\text{prec}, s_{ij}|C_{\max}$ another list scheduling algorithm has to be used in order to obtain a dominant set of schedules. A class of schedule generation schemes which is often used for different scheduling problems works as follows: Given a permutation π, a schedule is constructed in which job π_j does not start its execution earlier than job π_i for all $i<j$. Sprecher and Drexl [10] used such a procedure in a branch and bound algorithm (generating a so-called "precedence tree") for the resource-constrained project scheduling problem (RCPSP). Carlier and Neron [2] showed that for multi-processor flow-shop problems a so-called "strict scheduling algorithm" produces a dominant set of schedules.

In the following section, we will deal with the question whether such an approach is also possible for problem $P|\text{prec}, s_{ij}|C_{\max}$. We will show that it is NP-hard to determine a schedule with minimal makespan where the starting times respect a given order π.

3. NP-hardness results

In this section, we will consider the parallel machine problem $P|s_{ij}|C_{\max}$ with sequence-dependent setup times s_{ij}, no precedence relations, and a given job list π. We are interested in a schedule with minimal makespan where job π_j does not start its execution earlier than job π_i for all $i<j$.

Theorem 1. *For problem $P|s_{ij}|C_{\max}$ it is NP-hard in the strong sense to determine a schedule with minimal makespan where the starting times respect a given order π.*

Proof. To prove the NP-hardness, we will reduce the strongly NP-hard problem 3-PARTITION (3-PART) to the decision version of the given problem.

3-PART: Given are $3r$ positive number $a_1,\dots,a_{3r}$ with $\sum_{i=1}^{3r} = rb$ and $b/4 < a_i < b/2$ for $i=1,\dots,3r$. Does there exist a partition $I_1,\dots,I_r$ of the index set $\{1,\dots,3r\}$ such that $|I_j|=3$ and $\sum_{i\in I_j} a_i = b$ for $j=1,\dots,r$?

Given an arbitrary instance of 3-PART, an instance of problem $P|s_{ij}|C_{\max}$ with a given starting time order π is constructed as follows:

Let $c := (m+1)b+1$, let the number of machines be given by $m := r$, and let the number of jobs be $n := 3r^2$. For simplicity of notation, we denote the $3r^2$ jobs by pairs (i,j) with $i = 1,\dots,3r$ and $j = 1,\dots,r$. The processing times of the jobs $(1,j)$ for $j = 1,\dots,r$ are given by

$$p_{(1,j)} := \begin{cases} a_1 & \text{if } j = 1, \\ 0 & \text{otherwise} \end{cases}$$

and the processing times of the remaining jobs (i,j) for $i = 2,\dots,3r$ and $j = 1,\dots,r$ are defined as

$$p_{(i,j)} := \begin{cases} c - b + a_i & \text{if } j = 1, \\ c - jb & \text{if } j \geqslant 2. \end{cases}$$

The setup times between two jobs $(i,j),(k,l)$ are given by

$$s_{(i,j),(k,l)} := \begin{cases} lb & \text{if } k = i+1, \\ c+b+1 & \text{if } k = 1, \\ (l+1)b+1 & \text{otherwise.} \end{cases}$$

We ask for a schedule in which the starting times respect the lexicographic order

$$\pi := ((1,1),\dots,(1,m),(2,1),\dots,(2,m),(3,1),\dots,(3r,m))$$

with a makespan $C_{\max} \leqslant y := (3r-1)c + b$. We show that 3-PART has a feasible solution if and only if a schedule respecting π with $C_{\max} \leqslant y$ exists.

To do this, we first calculate the sum of the processing time of a job and the setup time preceding this job. Assume that two jobs (i,j) and (k,l) are scheduled consecutively on the same machine. Then we have

$$s_{(i,j),(k,l)} + p_{(k,l)} = \begin{cases} c + a_k & \text{if } k = i+1 \text{ and } l = 1, \\ c & \text{if } k = i+1 \text{ and } l \geqslant 2, \\ c+b+1+a_k & \text{if } k \neq i+1 \text{ and } l = 1, \\ c+b+1 & \text{otherwise.} \end{cases} \tag{3.1}$$

Now assume that $I_1,\dots,I_r$ is a feasible solution of 3-PART. We construct a corresponding schedule for the jobs (i,j) with $i = 1,\dots,3r$ and $j = 1,\dots,r$ of the instance of $P|s_{ij}|C_{\max}$ as follows: If $i \in I_j$, schedule job $(i,1)$ on the ith position on machine M_j and schedule the jobs $(i,2),\dots,(i,m)$ on the ith position of the other machines M_k, $k \neq j$, in an arbitrary way.

The construction of such a schedule is illustrated by an example in Fig. 1, where we have $r = m = 3$ and assume that $I_1 = \{1,4,6\}$, $I_2 = \{2,5,8\}$, $I_3 = \{3,7,9\}$ is a solution of 3-PART.

Let us consider an arbitrary machine M_k and denote by $(i_1,j_1),\dots,(i_{3r},j_{3r})$ the job sequence on M_k in the resulting schedule. We assume that all jobs are processed consecutively and that no idle times on the machines occur due to the order π. Later on we will show that this assumption is valid since job π_j does not start before job π_i for all $i < j$, i.e. the starting time order π is respected in the schedule.

Based on the assignment, we have $i_q = q$ for $q = 1,\dots,3r$ and, thus, for the completion time C_k^M of machine M_k we get

$$\begin{aligned} C_k^M &= p_{(1,j_1)} + \sum_{q=2}^{3r} (s_{(q-1,j_{q-1}),(q,j_q)} + p_{(q,j_q)}) \\ &= (3r-1)c + \sum_{\{q|j_q=1\}} a_{i_q} = (3r-1)c + \sum_{i \in I_k} a_i \end{aligned}$$

due to the second case in (3.1). Since $I_1,\dots,I_r$ forms a feasible partition, this value is equal to $(3r-1)c+b=y$ and the resulting schedule satisfies $C_{\max} = \max_{k=1}^{m}\{C_k^M\} \leqslant y$.

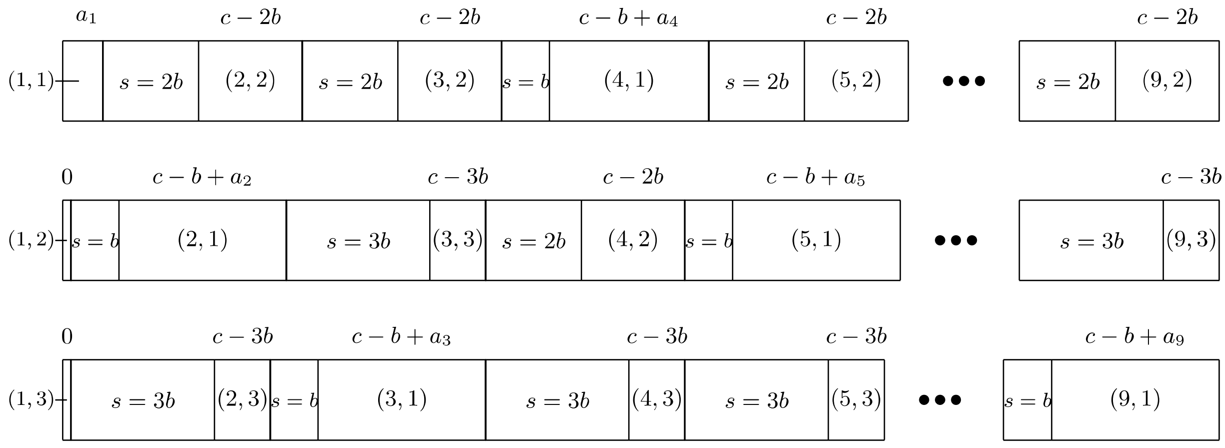


Fig. 1. Schedule derived from a solution of 3-PART with $r=m=3$.

To state that the schedule is a feasible solution for the given problem, it remains to show that the starting times respect the order $\pi=((1,1),\dots,(1,m),(2,1),\dots,(2,m),(3,1),\dots,(3r,m))$.

Let S_i^j denote the starting time of job (i,j) in the given schedule. For $i=1$ we have $S_1^1=0\leqslant S_1^2=0\leqslant\cdots\leqslant S_1^m=0$. Now we consider $i\geqslant 2$. Since job (i,j) is scheduled on the ith position on a machine, we get

$$S_i^j\geqslant (i-2)c+s_{h,(i,j)}=(i-2)c+jb,$$

where h denotes the predecessor of job (i,j) on its machine and

$$S_i^j\leqslant (i-2)c+s_{h,(i,j)}+b=(i-2)c+(j+1)b.$$

Thus, we get

$$S_i^1\leqslant S_i^2\leqslant\cdots\leqslant S_i^m.$$

Since, furthermore

$$S_i^m\leqslant (i-2)c+(m+1)b=(i-1)c-1<(i-1)c+b\leqslant S_{i+1}^1,$$

we can conclude that the constructed schedule respects π and, thus, is a feasible solution for the considered parallel machine problem.

Conversely, assume that problem $P|s_{ij}|C_{\max}$ has a solution respecting π with $C_{\max}\leqslant y=(3r-1)c+b$. First, we consider a schedule on a single machine M_k. Let $(i_1,j_1),\dots,(i_Q,j_Q)$ be the sequence of jobs scheduled on M_k. In three steps we will show that:

- exactly $Q=3r$ jobs are processed on M_k,
- the first job (i_1,j_1) is a job of the type $(1,j)$, and
- all first indices i_q of the jobs (i_q,j_q) are numbered consecutively, i.e. $i_q=q$ holds for $q=1,\dots,Q$.

For the completion time C_k^M of machine M_k we get

$$C_k^M\geqslant p_{(i_1,j_1)}+\sum_{q=2}^{Q}(s_{(i_{q-1},j_{q-1}),(i_q,j_q)}+p_{(i_q,j_q)}).$$

- Due to (3.1) we have $C_k^M\geqslant p_{(i_1,j_1)}+(Q-1)c$. Thus, since the given schedule satisfies $C_{\max}\leqslant(3r-1)c+b$, at most $3r$ jobs are processed on each machine (i.e. $Q\leqslant 3r$). However, since the total number of jobs to be scheduled on the m machines is given by $3rm$, exactly $3r$ jobs have to be processed on each machine (i.e. $Q = 3r$).

- If we now assume that for the first job (i_1, j_1) we have $i_1 \neq 1$, this induces
$$C_k^M \geqslant (3r-1)c + p_{(i_1,j_1)} \geqslant (3r-1)c + c - mb = (3r-1)c + b + 1,$$
which is a contradiction. Since we have exactly m jobs of type $(1, j)$, on each machine first a job $(1, j)$ and afterwards $3r - 1$ other jobs are processed (i.e. $i_1 = 1$).
- Next, assume that the indices i_{q-1}, i_q of two consecutive jobs (i_{q-1}, j_{q-1}) and (i_q, j_q) are not numbered consecutively, i.e. we have $i_q \neq i_{q-1} + 1$. Due to (3.1), this yields
$$C_k^M \geqslant (3r-1)c + b + 1,$$
which is a contradiction. Therefore, we have $i_q = q$ for $q = 1, \ldots, 3r$, which implies
$$C_k^M = (3r-1)c + \sum_{\{q|j_q=1\}} a_{i_q}.$$

From $C_k^M \leqslant y = (3r-1)c + b$ we can now conclude $\sum_{\{q|j_q=1\}} a_{i_q} \leqslant b$ for each machine. However, since $\sum_{i=1}^{3r} a_i = mb$ and since each job $(i, 1)$ for $i = 1, \ldots, 3r$ is scheduled on one of the m machines, we get $\sum_{\{q|j_q=1\}} a_{i_q} = b$. Thus, if we associate with machine M_k the set $I_k = \{i_q | j_q = 1\}$, we obtain a feasible solution for problem 3-PART. □

The above theorem shows that problem $P|s_{ij}|C_{\max}$ with a given starting time order π is NP-hard in the strong sense if the number of machines is arbitrary. The next theorem shows that for a fixed number m of machines (even for $m = 2$) the problem stays NP-hard.

Theorem 2. *For problem $P2|s_{ij}|C_{\max}$ it is NP-hard to determine a schedule with minimal makespan respecting a given starting time order π.*

Proof. The NP-hardness can be proven by reducing the NP-hard problem PARTITION to the decision version of problem $P2|s_{ij}|C_{\max}$ in a similar way as presented in Theorem 1. □

Since the reduction from Theorem 2 starts from the ordinary NP-hard problem PARTITION, the theorem only states that problem $Pm|s_{ij}|C_{\max}$ is NP-hard in the ordinary sense. In the following section, we will show that it is unlikely to expect that the problem is NP-hard in the strong sense, since a pseudo-polynomial dynamic programming algorithm to solve the problem is presented.

In the literature, it is often assumed that setup times satisfy a triangle inequality. The setup times satisfy the so-called weak triangle inequality if
$$s_{ih} + p_h + s_{hj} \geqslant s_{ij}$$
for all jobs i, j, h holds. If we have even $s_{ih} + s_{hj} \geqslant s_{ij}$ for all i, j, h, the strong triangle inequality holds.

Obviously, the setup times in the example in Section 2 satisfy the strong triangle inequality. For instance in Theorem 1, the weak triangle inequality $\hat{s} := s_{(ij),(k,l)} + p_{(k,l)} + s_{(k,l),(g,h)} \geqslant s_{(i,j),(g,h)}$ can be shown as follows:

- If $g = i + 1$, we have $\hat{s} \geqslant s_{(k,l),(g,h)} = (h+1)b + 1 > hb = s_{(i,j),(g,h)}$,
- if $g = k + 1$, we get $\hat{s} \geqslant s_{(ij),(k,l)} + p_{(k,l)} \geqslant c \geqslant (m+1)b + 1 \geqslant (h+1)b + 1 = s_{(i,j),(g,h)}$,
- and in all other cases we obtain $\hat{s} \geqslant s_{(k,l),(g,h)} = s_{(i,j),(g,h)}$.

For the instance in Theorem 2 analogously the weak triangle inequality can be shown. Note that both instances can be modified in such a way that even the strong triangle inequality holds (by adding a large constant to all setup times and adjusting the threshold value y appropriately). Thus, also in this case the problems remain NP-hard.

4. A pseudo-polynomial algorithm

In this section, we will provide a pseudo-polynomial algorithm for problem $Pm|s_{ij}|C_{\max}$ with a given starting time order π. Let $T \leqslant \sum_{j=1}^{n}(p_j + \sum_{i=1}^{n} s_{ij})$ denote an upper bound for the optimal makespan, which is pseudo-polynomially bounded with respect to the input length of the instance. The key issue of the dynamic programming algorithm is the observation that for adding job π_k to a partial schedule of the jobs $\pi_1, \dots, \pi_{k-1}$ we only need to know the finishing times of all machines $M_1, \dots, M_m$ and the jobs which are scheduled last on them. Following this observation, the stages of the dynamic program can be chosen as follows:

$$(k, t_1, \dots, t_m, l_1 \dots, l_m),$$

where

$k \in \{1, \dots, n\}$	denotes the number of scheduled jobs,
$t_j \in \{0, \dots, T\}$	for $j = 1, \dots, m$ denotes the completion time of the last job on M_j,
$l_j \in \{1, \dots, n\}$	for $j = 1, \dots, m$ denotes the last job on M_j.

For a stage $(k, t_1, \dots, t_m, l_1 \dots, l_m)$ we will set $f(k, t_1, \dots, t_m, l_1 \dots, l_m) := 1$ if a feasible schedule of the jobs $\pi_1, \dots, \pi_k$ exists

- which respects the order π,
- where job l_j is scheduled last on machine j ($l_j = 0$ indicates that no job is scheduled on machine j), and
- where job l_j completes at time t_j (if $l_j = 0$, t_j must be zero too).

Otherwise, $f(k, t_1, \dots, t_m, l_1 \dots, l_m)$ will be defined as 0. Following this definition, we may restrict the considerations to stages $(k, t_1, \dots, t_m, l_1 \dots, l_m)$ with:

- $l_j \in \{0, \pi_1, \dots, \pi_k\}$, i.e. the last jobs belong to the set of scheduled jobs,
- $l_j \neq l_i$ if $l_j \neq 0$, i.e. on different machines different jobs are scheduled last,
- $t_j = 0$ if $l_j = 0$, i.e. the completion time of M_j is 0 if no job is scheduled on it,
- $\pi_k \in \{l_1, \dots, l_m\}$, i.e. the job π_k added in the stage is scheduled last on a machine,
- $t_i - p_{l_i} \leqslant t_j - p_{l_j}$ if l_i precedes l_j in π, i.e. the starting time of job l_i is not larger than the starting time of job l_j.

Stages fulfilling these conditions will be called feasible stages. It is straightforward to see that the number of stages is bounded by $n(T+1)^m n^m$, which is pseudo-polynomially bounded in the input length of the instance.

Now, assume that for a fixed value of k all the f-values for all feasible stages of the form $(k - 1, t'_1, \dots, t'_m, l'_1 \dots, l'_m)$ are known. Based on these values, the f-value of a feasible stage $(k, t_1, \dots, t_m, l_1 \dots, l_m)$ can be calculated as follows:

$f(k, t_1, \dots, t_m, l_1 \dots, l_m) = 1$ if and only if job π_k can be added to the partial schedule as a last job, i.e. if a feasible stage $(k-1, t_1, \dots, t_{j-1}, t'_j, t_{j+1}, \dots, t_m,\ l_1 \dots, l_{j-1}, l'_j, l_{j+1}, \dots, l_m)$ exists with

- $t'_j \leqslant t_j - p_{\pi_k} - s_{l'_j \pi_k}$ ($s_{l'_j \pi_k} = 0$ if $l'_j = 0$), i.e. job π_k can be scheduled last on M_j, and
- $f(k-1, t_1, \dots, t_{j-1}, t'_j, t_{j+1}, \dots, t_m, l_1 \dots, l_{j-1}, l'_j, l_{j+1}, \dots, l_m) = 1$.

Calculating this value takes an effort of at most $O(Tn)$.

If, initially, we define $f(0, 0, \dots, 0, 0, \dots, 0) = 1$, we can calculate successively the f-values of all feasible states in $O(n^{m+2}T^{m+1})$ and, thus, in pseudo-polynomial time. The makespan of a best schedule is then given by the value $\max_{j=1}^{m}\{t_j\}$ of a feasible stage $(n, t_1, \dots, t_m, l_1 \dots, l_m)$ with $f(n, t_1, \dots, t_m, l_1 \dots, l_m) = 1$.

5. Consequences for list scheduling

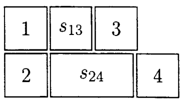
In Section 2, we discussed two different ways of using list scheduling to deal with the subproblem resulting after fixing a sequence π of the jobs for problem $P|\text{prec}, s_{ij}|C_{\max}$:

- consider the jobs in the order π and schedule them such that they start processing as early as possible,
- schedule the jobs such that their starting times respect the order π.

The example in Section 2 shows that the first possibility does not lead to a dominant set. The results of Sections 3 and 4 indicate that it is hard to find an efficient method for the second possibility. It seems that a negative answer to the second possibility already would imply a negative answer to the first possibility. However, this must not be the case. As mentioned in Section 2, Schutten [7] showed that for problem $P|s_{ij}|C_{\max}$ the set of list schedules calculated with the first possibility is a dominant set, whereas our results from Section 3 state that the problem of finding a best schedule respecting a given starting time order is NP-hard. The reason for this is that the list scheduling algorithm using a sequence π does not necessarily result in a schedule where the starting times respect π.

Example. Given are 2 machines and 4 jobs with unit processing times. The setup times are given by

$$s = (s_{ij})_{i,j=1,\dots,4} = \begin{pmatrix} 0 & 10 & 1 & 10 \\ 10 & 0 & 0 & 2 \\ 10 & 10 & 0 & 10 \\ 10 & 10 & 10 & 0 \end{pmatrix}$$



The optimal solution is achieved by scheduling jobs 1 and 3 on one machine in this order and jobs 2 and 4 on the other machine in that order (see the above figure). This solution can be calculated by the list scheduling algorithm using the sequence $\pi = (1,2,4,3)$ or $\pi = (2,1,4,3)$. However, for both sequences in the resulting schedule the starting times of job 3 and 4 do not respect the order $(4,3)$.

It still remains open whether another type of list scheduling algorithm is able to produce a dominant set in an efficient way. However, a closer look at the proof of Theorem 1 indicates that this is very unlikely:

In the proof of Theorem 1 *no precedences are used and, thus, a list scheduling algorithm may consider all possible sequences of the jobs and calculate for each of these sequences a corresponding schedule (which does not have to respect the given sequence of the jobs). In the following, we show that by the introduction of precedences an instance may be constructed which still may be used for the proof of Theorem* 1, *but which allows only one relevant sequence to be considered by a list scheduling algorithm.*

The instance of the scheduling problem corresponding to an arbitrary instance of 3-*PART in the proof of Theorem* 1 *has to be changed as follows:*

Replace each job (i,j) by two jobs $(i,j)^s$ and $(i,j)^p$ which are linked by a precedence constraint $(i,j)^s \to (i,j)^p$ and which have processing times $p_{(i,j)^s} := 0$ and $p_{(i,j)^p} := p_{(i,j)}$. The s-job represents the starting point of the original job and the p-job represents the real processing of the original job. This construction is necessary, to enable us to translate a starting time sequence into precedences (note that precedences are finish–start relations between jobs and the sequence π in Theorem 1 *corresponds to start–start relations between jobs).*

The original setup times between two jobs (i,j) and (k,l) are transfered to the jobs $(i,j)^p$ and $(k,l)^s$. Furthermore, we define additional setup times which guarantee that the s-job and the p-job corresponding to the same original job are always scheduled directly after each other on the same machine. This is done by setting

$$s_{(i,j)^s,(k,l)^p} := \begin{cases} 0 & \text{if } (i,j) = (k,l), \\ \infty & \text{otherwise.} \end{cases}$$

This forces that job $(i,j)^p$ has to be scheduled immediately after job $(i,j)^s$ on the same machine in each schedule of finite length. Since the setup time between these two jobs is 0, *each feasible schedule of the new instance with finite length is also a feasible schedule of the instance used in the proof of Theorem* 1 *and vice versa.*

If we now introduce precedence constraints

$$(1,1)^s \to \cdots \to (1,m)^s \to (2,1)^s \to \cdots \to (2,m)^s \to (3,1)^s \to \cdots \to (3r,m)^s$$

each feasible schedule with finite length will respect the sequence π given in the proof of Theorem 1. *However, in the new instance only the decisions for the jobs $(i,j)^s$ are relevant (the p-jobs have to be processed immediately behind the corresponding s-job on the same machine). Since the precedence constraints allow just one sequence for these s-jobs, only one relevant sequence is available for list scheduling and, thus, if list scheduling would result in a dominant set, the application of the algorithm to only one sequence would solve an NP-hard problem.*

6. Conclusions

The considered parallel machine problem $P|\text{prec}, s_{ij}|C_{\max}$ is a combination of a partitioning and a sequencing problem. Thus, a possible optimization algorithm for it may be based on a two-stage approach where first decisions for one of the subproblems are fixed and afterwards the remaining part of the problem is treated. This raises the question whether fixing decisions for the partition part or the sequencing part leads to easy solvable remaining subproblems. Obviously, fixing the partition part leads to an NP-hard subproblem generalizing the traveling salesman problem. In this paper, we focused on the opposite approach where the sequencing part is fixed first.

The presented results show that it is unlikely that an efficient list scheduling algorithm exists which leads to a dominant set of schedules. As a consequence, larger instances of the parallel machine problem with precedence constraints and sequence-dependent setup times cannot be solved by considering only the decisions for one of its two parts as solution space and solving the corresponding remaining subproblem afterwards. For alternative methods, one either has to develop solution approaches which consider both parts of the solutions simultaneously or one has to relax the goal of solving the resulting subproblems to optimality.

References

[1] K.R. Baker, Introduction to Sequencing and Scheduling, Wiley, New York, 1974.
[2] J. Carlier, E. Neron, An exact method for solving the multi-processor flow-shop, RAIRO-Rech. Oper. 34 (2000) 1–25.
[3] R.L. Graham, Bounds on multiprocessing timing anomalies, SIAM J. Appl. Math. 17 (1969) 416–429.
[4] R.L. Graham, E.L. Lawler, J.K. Lenstra, A.H.G. Rinnooy Kan, Optimization and approximation in deterministic sequencing and scheduling: a survey, Ann. Disc. Math. 5 (1979) 287–326.
[5] C.L. Monma, C.N. Potts, On the complexity of scheduling with batch setup times, Oper. Res. 37 (1989) 798–804.
[6] M. Pinedo, X. Chao, Operations scheduling with applications in manufacturing and services, Irwin/McGraw-Hill, Boston, 1999.
[7] J.M.J. Schutten, List scheduling revisited, Oper. Res. Lett. 18 (1996) 167–170.
[8] J.M.J. Schutten, R.A.M. Leussink, Parallel machine scheduling with release dates, due dates and family setup times, Internat. J. Prod. Econom. 46 (1996) 119–125.
[9] W.E. Smith, Various optimizers for single-stage production, Naval Res. Logist. Quart. 3 (1956) 59–66.
[10] A. Sprecher, A. Drexl, Solving multi-mode resource-constrained project scheduling problems by a simple, general and powerful sequencing algorithm, European J. Oper. Res. 107 (1998) 431–450.
[11] A.P. Woerlee, Decision support systems for production scheduling, Ph.D. Thesis, Erasmus University, Rotterdam, 1991.