Faculty of Mathematical Sciences

University of Twente

University for Technical and Social Sciences

P.O. Box 217
7500 AE Enschede
The Netherlands
Phone: +31-53-4893400
Fax: +31-53-4893114
Email: memo@math.utwente.nl

Memorandum No. 1525

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

J.L. Hurink and S. Knust[1]

May 2000

ISSN 0169-2690

[1]Universität Osnabrück, Fachbereich Mathematik/Informatik, D-49069 Osnabrück, Germany

List Scheduling in a Parallel Machine Environment with Precedence Constraints and Setup Times

Johann Hurink
University of Twente, Faculty of Mathematical Sciences,
NL-7500 AE Enschede
j.l.hurink@math.utwente.nl

Sigrid Knust *
Universität Osnabrück, Fachbereich Mathematik/Informatik,
D-49069 Osnabrück
sigrid@mathematik.uni-osnabrueck.de

May 2, 2000

Abstract

We present complexity results which have influence on the strength of list scheduling in a parallel machine environment where additionally 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.

Keywords: scheduling, list scheduling, complexity, parallel machines, setup times

Subject classification: 90B35

*supported by the Deutsche Forschungsgemeinschaft, Project 'Komplexe Maschinen-Schedulingprobleme'

1 Introduction

The concept of list scheduling - for a given list of jobs construct a corresponding schedule by planning the jobs in the order of the list - has been widely used in the scheduling area. On the one hand, several polynomial algorithms utilize such procedures (e.g. Smith's rule [8] for problem $1||\sum w_jC_j$) and approximation heuristics are often based on priority lists (cf. e.g. Baker [1], Graham [3]). 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. 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.

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 $P|prec, s_{ij}|C_{\max}$ and can be stated as follows: given are n jobs with processing times $p_1, \ldots, p_n$ 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 & Potts [6].

We are interested in the described problem since it arises as a subproblem in a job-shop environment where the jobs additionally have to be transported between the machines by transport robots (cf. Knust [5]). A job-shop problem with transportation times is a generalization of the classical job-shop problem and may be formulated as follows: We are given a set of machines and a set of jobs. Each job consists of a chain of operations which have to be processed in this order. With each operation a dedicated machine is associated on which the operation has to be processed without preemption for a given duration. Each machine can process at most one operation at a time. Additionally, transportation times are considered. They occur if a job changes from one machine to another and depend on the jobs and the machines between which the transport takes place. We assume that all these transport operations have to be done by a group of r identical transport robots where each robot can handle at most one job at a time. Furthermore, if a robot moves empty between two machines, empty moving times occur (depending on the machines between which the move takes place), which may be regarded as sequence-dependent setup times on the robots. The objective is to determine a feasible schedule which minimizes the makespan.

In Hurink & Knust [4] we studied the situation with a single transport robot ($r = 1$) and proposed a two-level approach where on the first level machine orders for the job-shop machines are fixed and on the second level a corresponding robot order is constructed by a tabu search procedure. The resulting robot scheduling problem on the second level corresponds to the single-machine problem $1 \mid prec\,(l_{ij}), r_j, s_{ij} \mid \max\{C_j + q_j\}$, where $prec\,(l_{ij})$ indicates arbitrary non-negative finish-start time-lags $l_{ij} \geq 0$, s_{ij} stands for sequence-dependent setup times, r_j for release dates, and $C_j + q_j$ denotes for each job the sum of its completion time and its delivery time (tail) q_j. In order to generalize this two-stage approach to the situation with more than one robot, we have to deal with

the more complex subproblem $P \mid prec(l_{ij}), r_j, s_{ij} \mid \max\{C_j + q_j\}$, where the r transport robots correspond to r parallel identical machines.

Since this problem is a combination of a partitioning problem (assign the jobs to the machines) and a sequencing problem (determine for each machine an order in which the jobs assigned to this machine are executed), an 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. If we first assign the jobs to the machines, the remaining sequencing problems on the machines still contain the traveling salesman problem, i.e. they are strongly NP-hard. On the other hand, we could first treat the sequencing problem and try to determine an optimal assignment afterwards. Thus, in this context we have to deal with the question whether it is possible to design an efficient list scheduling algorithm which produces a dominant set of list schedules. 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 remainder of 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.

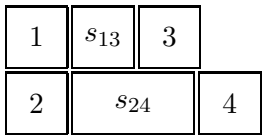
2 List scheduling algorithms

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. 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 $i \rightarrow j$ 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

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



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. 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 $3 \rightarrow 4$, the optimal solution remains feasible (job 4 starts directly after job 3 finishes). However, since the sequences $\pi = (1, 2, 4, 3)$ and $\pi = (2, 1, 4, 3)$ are no longer valid, the set of all list schedules does not contain the optimal solution anymore. □

This example shows that for problem $P|prec, s_{ij}|C_{\max}$ 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 & Drexl [9] 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 & 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 $P|prec, s_{ij}|C_{\max}$. 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 $\frac{b}{4} < a_i < \frac{b}{2}$ 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

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

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

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

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 i-th position on machine M_j and schedule the jobs $(i, 2), \dots, (i, m)$ on the i-th 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 Figure 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

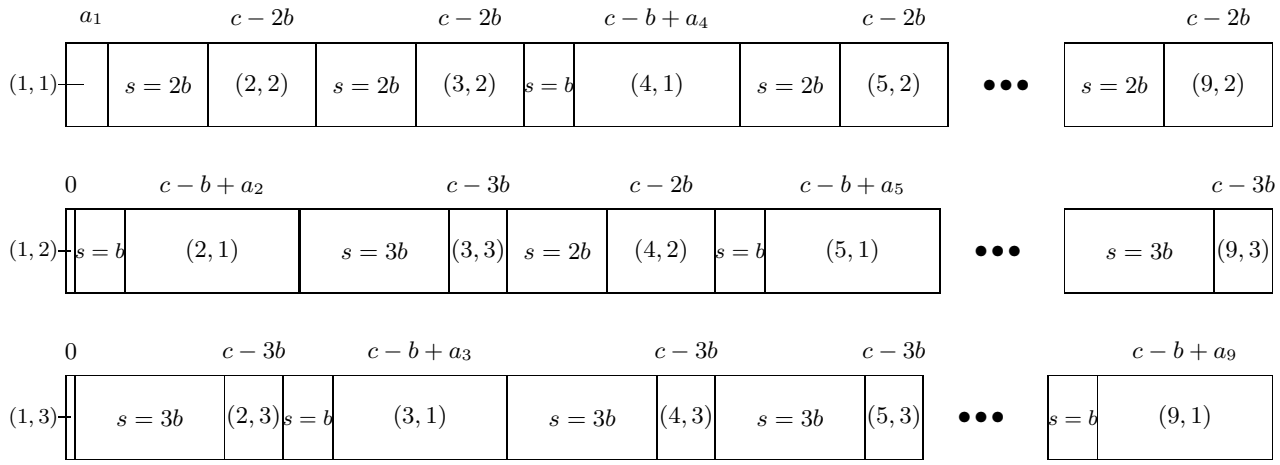


Figure 1: Schedule derived from a solution of 3-PART with $r = m = 3$

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 $q = 1, \ldots, 3r$ 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 $I_1, \ldots, I_r$ forms a feasible partition, this value is equal to $(3r-1)c + b = y$ and the resulting schedule satisfies $C_{\max} = \max_{k=1}^{m} \{C_k^M\} \leq y$.
To state that the schedule is a feasible solution for the given problem it remains to show that the starting times respect the order $\pi = ((1,1), \ldots, (1,m), (2,1), \ldots, (2,m), (3,1), \ldots, (3r,m))$.

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

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

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

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

Thus, we get

$$S_i^1 \leq S_i^2 \leq \ldots \leq S_i^m.$$

Since, furthermore

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

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 $C_{\max} \leq y = (3r-1)c + b$. First, we consider a schedule on a single machine M_k. Let $(i_1, j_1), \ldots, (i_Q, j_Q)$ 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 $q = 1, \ldots, Q$.

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

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

- Due to (3.1) we have $C_k^M \geq p_{(i_1,j_1)} + (Q-1)c$. Thus, since the given schedule satisfies $C_{\max} \leq (3r-1)c + b$, at most $3r$ jobs are processed on each machine (i.e. $Q \leq 3r$). 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

  $$C_k^M \geq (3r-1)c + p_{(i_1,j_1)} \geq (3r-1)c + c - mb = (3r-1)c + b + 1,$$

  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

  $$C_k^M \geq (3r-1)c + b + 1,$$

  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 $C_k^M \leq y = (3r-1)c + b$ we now can conclude $\sum_{\{q|j_q=1\}} a_{i_q} \leq b$ 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: One way to prove the NP-hardness is to reduce 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. However, a more direct way (using less 'dummy' jobs) to prove the NP-hardness is to reduce the NP-hard problem EVEN-ODD-PARTITION (EO-PART) to the decision version of the given problem.

EO-PART: Given are $2r$ positive number $a_1, \ldots, a_{2r}$ with $\sum_{i=1}^{2r} = 2b$. Does there exist a partition of the index set $\{1, \ldots, 2r\}$ into two disjoint sets I_1 and I_2 such that $\sum_{i \in I_1} a_i = \sum_{i \in I_2} a_i = b$ and the indices $2j-1$ and $2j$ belong to different sets for $j = 1, \ldots, r$?

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

Let $c := 3b$ and let the number of jobs be defined by $n := 2r$. The processing times of the jobs are given by

$$p_i := \begin{cases} c - b + a_i & \text{if } i = 2j-1 \quad \text{for some } j \in \{2, \ldots, r\} \\ b + a_i & \text{if } i = 2j \quad \text{for some } j \in \{2, \ldots, r\} \\ a_i & \text{if } i = 1, 2 \end{cases}$$

and the setup times between jobs k and l by

$$s_{kl} := \begin{cases} b & \text{if } k \in \{2j-1, 2j\} \text{ and } l = 2j+1 \text{ for some } j \in \{1, \ldots, r-1\} \\ c - b & \text{if } k \in \{2j-1, 2j\} \text{ and } l = 2j+2 \text{ for some } j \in \{1, \ldots, r-1\} \\ c + b + 1 & \text{otherwise} \end{cases}$$

(cf. Figure 2).

We ask for a schedule respecting the order $\pi := (1, \ldots, n)$ with a makespan $C_{\max} \leq y := (r-1)c + b$. We show that EO-PART has a feasible solution if and only if a schedule respecting π with $C_{\max} \leq y$ exists.

Assume that I_1, I_2 is a solution of EO-PART and that $I_i = \{i_1, \ldots, i_r\}$ with $i_j < i_{j+1}$ for $j = 1, \ldots, r-1$ and $i = 1, 2$. We construct a solution of problem $P2|s_{ij}|C_{\max}$ by scheduling on M_i $(i = 1, 2)$ the jobs corresponding to I_i in the order $(i_1, \ldots, i_r)$. Due to the assumption that the indices $2j-1$ and $2j$ belong to different sets, for $j = 1, \ldots, r$, we know that for $i_k = 2j+1$ or $i_k = 2j+2$ we have $i_{k-1} \in \{2j-1, 2j\}$ for $k = 2, \ldots, r$ and $i = 1, 2$. Thus, for i_k with $k \geq 2$ we have

$$s_{i_{k-1}, i_k} + p_{i_k} = c + a_{i_k}.$$

As a result we get for the makespan of this schedule:

$$\begin{aligned} C_{\max} &= \max\{C_{1_r}, C_{2_r}\} = \max_{i=1}^{2} \{p_{i_1} + \sum_{k=2}^{r} (s_{i_{k-1}, i_k} + p_{i_k})\} \\ &= (r-1)c + \sum_{k \in I_i} a_k = (r-1)c + b = y, \end{aligned}$$

where C_i denotes the completion time of job i in the schedule.

It remains to show that in the constructed schedule the starting times respect the order $\pi = (1, \ldots, n)$. We do this by showing that

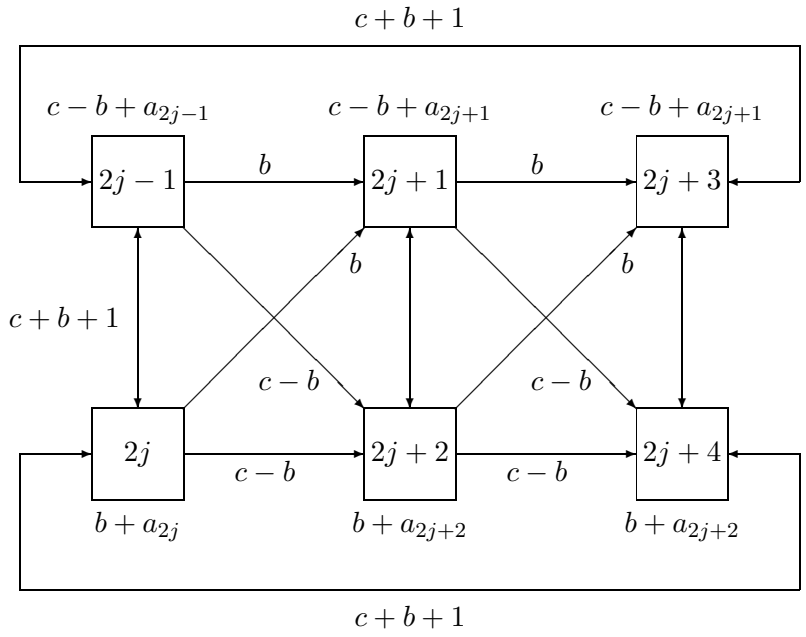


Figure 2: Definition of processing and setup times

- job $2j+1$ starts after job $2j$ for $j = 2, \ldots, r-1$ and
- job $2j$ starts after job $2j-1$ for $j = 2, \ldots, r$.

Jobs 1 and 2 both start at time 0 and, thus, job 3 obviously starts after job 2. Now we consider the case $j \geq 2$. If jobs $2j$ and $2j+1$ belong to the same set I_i, they are scheduled on the same machine, i.e. the starting times clearly satisfy $S_{2j+1} \geq S_{2j}$. Otherwise we have $2j+1 = i_{j+1}$ and $2j = \overline{i}_j$, where $\{\overline{i}_1, \ldots, \overline{i}_r\}$ with $\overline{i}_j < \overline{i}_{j+1}$ denotes the complement of the set I_i.

- The starting time S_{2j+1} of job $2j+1$ is given by

$$\begin{aligned} S_{2j+1} &= S_{i_{j+1}} = p_{i_1} + \sum_{k=2}^{j} (s_{i_{k-1},i_k} + p_{i_k}) + s_{i_j,2j-1} \\ &= (j-1)c + \sum_{k=1}^{j} a_{i_k} + b \geq (j-1)c + b. \end{aligned}$$

  Furthermore, we have

$$\begin{aligned} S_{2j} &= S_{\overline{i}_j} = p_{\overline{i}_1} + \sum_{k=2}^{j-1} (s_{\overline{i}_{k-1},\overline{i}_k} + p_{\overline{i}_k}) + s_{\overline{i}_{j-1},2j-2} \\ &= (j-2)c + \sum_{k=1}^{j} a_{i_k} + c - b \leq (j-1)c + b - b = (j-1)c. \end{aligned}$$

  Thus, job $2j+1$ starts after job $2j$ for $j = 2, \ldots, r-1$.

- Since the jobs $2j-1,\ 2j$ are scheduled on different machines, the previous formulas imply

$$S_{2j} \geq (j-2)c + 0 + c - b = (j-1)c - b$$

and

$$S_{2j-1} \leq (j-2)c + b + b = (j-2)c + 2b.$$

Due to $c = 3b$ we obtain

$$S_{2j} \geq (j-1)c - b \geq (j-2)c + 3b - b \geq S_{2j-1}$$

and, thus, job $2j$ starts after job $2j-1$ for $j = 2, \ldots, r$.

Conversely, assume that problem $P|s_{ij}|C_{\max}$ has a solution respecting $\pi = (1, \ldots, n)$ with $C_{\max} \leq y = (r-1)c + b$. Let $I_i = \{i_1, \ldots, i_{n_i}\}$ be the set of jobs processed on machine M_i $(i = 1, 2)$ in the order $(i_1, \ldots, i_{n_i})$. Since the schedule respects π, we must have $i_j < i_{j+1}$ for $j = 1, \ldots, n_i$ and $i = 1, 2$.

For the completion time C_i^M of machine M_i in the given schedule we get

$$C_i^M \geq p_{i_1} + \sum_{j=2}^{n_i} s_{i_{j-1},i_j} + p_{i_j} \geq p_{i_1} + (n_i - 1)c.$$

Since we have $C_{\max} \leq (r-1)c + b$ for this schedule, this implies $n_i \leq r$ and thus $n_i = r$ for $i = 1, 2$. Furthermore, if two jobs $2j-1$, $2j$ are processed on the same machine M_i, we get $C_i^M \geq (r-1)c + b + 1$, which is a contradiction. Thus, for $j = 1, \ldots, r$ the indices $2j-1$ and $2j$ belong to different sets I_i.

Since the jobs $2j-1$ and $2j$ are assigned to different machines, we know that for $k \geq 2$ the predecessor of job $i_k = 2j+1$ or $i_k = 2j+2$ is a job from the set $\{2j-1, 2j\}$ and, thus, $s_{i_{k-1},i_k} + p_{i_k} = c + a_{i_k}$. Therefore, for the completion time C_i^M of machine M_i in the given schedule we have

$$C_i^M \geq \sum_{j \in I_i} a_j + (r-1)c \quad \text{for } i = 1, 2.$$

Since the makespan of the given schedule is at most $(r-1)c + b$, this implies

$$\sum_{j \in I_i} a_j \leq b \quad \text{for } i = 1, 2.$$

From $\sum_{j=1}^{2r} a_j = 2b$ we now can conclude $\sum_{j \in I_i} a_j = b$ for $i = 1, 2$ and therefore I_1, I_2 is a feasible solution for EO-PART. □

Since the reduction presented in the above theorem starts from the ordinary NP-hard problem EVEN-ODD-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 (also in the job-shop application such an inequality is satisfied, cf. Hurink & Knust [4]). The setup times satisfy the so-called weak triangle inequality if

$$s_{ih} + p_h + s_{hj} \geq s_{ij}$$

for all jobs i, j, h holds. If we have even $s_{ih} + s_{hj} \geq s_{ij}$ 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. Furthermore, it is easy to see that the defined setup and processing times of the instances in the previous two NP-hardness proofs satisfy the weak triangle inequality. For the instances in Theorem 1 the inequality $\hat{s} := s_{(ij),(k,l)} + p_{(k,l)} + s_{(k,l),(g,h)} \geq s_{(i,j),(g,h)}$ can be shown as follows.

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

For the instances in Theorem 2 analogously the inequality $s_{ih} + p_h + s_{hj} \geq s_{ij}$ can be shown (cf. Figure 2).

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 $T \leq \sum_{j=1}^{n}(p_j + \sum_{i=1}^{n} s_{ij})$ 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 $\pi_1, \ldots, \pi_{k-1}$ we only need to know the finishing times of all machines $M_1, \ldots, M_m$ and the jobs which are scheduled last on them. Following this observation, the stages of the dynamic program can be chosen as follows:

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

where

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

For a stage $(k, t_1, \ldots, t_m, l_1 \ldots, l_m)$ we will set $f(k, t_1, \ldots, t_m, l_1 \ldots, l_m) := 1$ if a feasible schedule of the jobs $\pi_1, \ldots, \pi_k$ 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, $f(k, t_1, \ldots, t_m, l_1 \ldots, l_m)$ will be defined as 0. Following this definition, we may restrict the considerations to stages $(k, t_1, \ldots, t_m, l_1 \ldots, l_m)$ with:

- $l_j \in \{0, \pi_1, \ldots, \pi_k\}$, 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,
- $\pi_k \in \{l_1, \ldots, l_m\}$, i.e. the job π_k added in the stage is scheduled last on a machine,
- $t_i - p_{l_i} \leq t_j - p_{l_j}$ 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 $(k-1, t'_1, \ldots, t'_m, l'_1 \ldots, l'_m)$ are known. Based on these values, the f-value of a feasible stage $(k, t_1, \ldots, t_m, l_1 \ldots, l_m)$ can be calculated as follows:

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

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

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

If, initially, we define $f(0, 0, \ldots, 0, 0, \ldots, 0) = 1$, 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 $(n, t_1, \ldots, t_m, l_1 \ldots, l_m)$ with $f(n, t_1, \ldots, t_m, l_1 \ldots, l_m) = 1$.

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 $P|prec, s_{ij}|C_{\max}$.

- 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, however, for the optimal sequence the list scheduling algorithm will give the optimal solution.

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. If we may use precedences in the instance of the scheduling problem corresponding to an arbitrary instance of 3-PART, we may change the instance in the proof of Theorem 1 as follows:

Replace each job (i,j) by two jobs $(i,j)^s$ and $(i,j)^p$ which are linked by a precedence constraint $(i,j)^s \rightarrow (i,j)^p$ and which have processing times $p_{(i,j)^s} := 0$ and $p_{(i,j)^p} := p_{(i,j)}$. The setup times between two jobs (i,j) and (k,l) are transfered to the jobs $(i,j)^p$ and $(k,l)^s$ and setup times between jobs $(i,j)^s$ and $(k,l)^p$ are defined such that

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

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

$$(1,1)^s \rightarrow \ldots \rightarrow (1,m)^s \rightarrow (2,1)^s \rightarrow \ldots \rightarrow (2,m)^s \rightarrow (3,1)^s \rightarrow \ldots \rightarrow (3r,m)^s$$

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 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 $P|prec, s_{ij}|C_{\max}$ 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. Developing such methods is the topic of further research.

References

[1] **Baker, K.R. [1974]** Introduction to sequencing and scheduling, Wiley, New York.

[2] **Carlier, J., Neron, E. [2000]** An exact method for solving the multi-processor flow-shop, to appear in RAIRO.

[3] **Graham, R.L. [1969]** Bounds on multiprocessing timing anomalies, SIAM Journal Applied Mathematics 17, 416-429.

[4] **Hurink, J., Knust, S. [1999]** A tabu search algorithm for scheduling a single robot in a job-shop environment, Osnabrücker Schriften zur Mathematik, Reihe P, Nr. 213, to appear in Discrete Applied Mathematics.

[5] **Knust, S. [1999]** Shop-scheduling problems with transportation, Ph.D. thesis, Fachbereich Mathematik/Informatik Universität Osnabrück.

[6] **Monma, C.L., Potts, C.N. [1989]** On the complexity of scheduling with batch setup times, Operations Research 37, 798-804.

[7] **Schutten, J.M.J [1996]** List scheduling revisited, Operations Research Letters 18, 167-170.

[8] **Smith, W.E. [1956]** Various optimizers for single-stage production, Naval Res. Logist. Quart. 3, 59-66.

[9] **Sprecher, A., Drexl, A. [1998]** Solving multi-mode resource-constrained project scheduling problems by a simple, general and powerful sequencing algorithm, European Journal of Operational Research 107, 431-450.