# Single-machine scheduling with release dates, due dates and family setup times

J.M.J. Schutten
S.L. van de Velde
W.H.M. Zijm
Department of Mechanical Engineering
University of Twente
P.O. Box 217, 7500 AE Enschede, The Netherlands

September 10, 1993 – Revised August 2, 1995

**Abstract**

We address the NP-hard problem of scheduling $n$ independent jobs with release dates, due dates, and family setup times on a single machine to minimize the maximum lateness. This problem arises from the constant tug-of-war going on in manufacturing between efficient production and delivery performance, between maximizing machine utilization by batching similar jobs and maximizing customers' satisfaction by completing jobs before their due dates. We develop a branch-and-bound algorithm, and our computational results show that it solves almost all instances with up to about 40 jobs to optimality. The main algorithmic contribution is our lower bounding strategy to deal with family setup times. The key idea is to see a setup time as a setup job with a specific processing time, release date, due date, and precedence relations. We develop several sufficient conditions to derive setup jobs. We specify their parameters and precedence relations such that the optimal solution value of the modified problem obtained by ignoring the setup times, not the setup jobs, is no larger than the optimal solution value of the original problem. One lower bound for the modified problem proceeds by allowing preemption. Due to the agreeable precedence structure, the preemptive problem is solvable in $O(n \log n)$ time.

*1980 Mathematics Subject Classification (Revision 1991)*: 90B35.
*Keywords and Phrases*: scheduling, maximum lateness, family setup times, branch-and-bound, setup jobs, preemption.

# 1 Introduction

In the last two decades, we have seen dramatic changes of the conditions under which manufacturing organizations have to operate and the objectives they have to meet. Next to efficiency, quality and delivery reliability have become key performance criteria (cf. Deming [5] and Blackburn [2]). In particular, the ability to cut manufacturing lead times and to meet tight due dates determines a company's competitive position.

In machining environments, such as a part manufacturing shop, the combined goal of efficient and effective production may lead to complex control problems. Efficient production in such an environment is achieved by minimizing the loss of capacity due to setups and thus by combining jobs with similar setup characteristics. Effective production in an order-driven environment is achieved by completing jobs before their due dates, or at least by minimizing lateness. Clearly, these two objectives may be conflicting: Clustering jobs with similar setup characteristics may lead to the lateness of others. Any solution to these problems should therefore be based on a combination of batching and sequencing considerations. These problems are often dealt with hierarchically. On a higher level, batch sizes (or run lengths) of jobs of the same or similar nature are determined; sequencing these batches is then a lower level, short term decision. Maintaining this hierarchical approach under the current market conditions with increasing product diversity and decreasing product life cycles, however, may lead to unacceptable results, including a poor delivery performance and obsolete stocks. This creates the need to cluster the jobs dynamically, depending on the workload.

This paper addresses the combined setup/due date problem in a relatively simple but, in our experience, highly relevant setting. We consider the following problem, in which a set $\mathcal{J}$ of $n$ independent jobs $J_1, \ldots, J_n$ need to be processed on a single-machine. Each $J_j$ $(j = 1, \ldots, n)$ needs uninterrupted processing during a given positive time $p_j$, becomes available for processing at its release date $r_j$, and should be completed by its due date $d_j$. The machine is available from time 0 onwards and can process no more than one job at a time. The jobs are partitioned into families $\mathcal{F}_1, \ldots, \mathcal{F}_m$, and $f(j)$ is the index of the family to which job $J_j$ belongs. If we schedule two jobs that belong to different families contiguously, then we need a given non-negative setup time $s_i$ in between that is completely specified by the family $\mathcal{F}_i$ to which the second job belongs. We also assume that we need a setup for the first job of each family. No setup is needed when jobs of the same family are scheduled contiguously. During a setup time no processing of jobs is possible. The machine may be set up for a particular job prior to its release date. The set of jobs between two subsequent setups are said to be scheduled in the same *batch*.

Without loss of generality, we assume that all data are integral. A feasible *schedule* $\sigma$ satisfies all these conditions and specifies for each $J_j$ a completion time $C_j(\sigma)$. For a given schedule $\sigma$, we compute the lateness of $J_j$ as $L_j(\sigma) = C_j(\sigma) - d_j$. If $L_j(\sigma) \leq 0$, then $J_j$ is *early*; otherwise it is *tardy*. The maximum lateness of $\sigma$ is defined as $L_{\max}(\sigma) = \max_{1 \leq j \leq n} L_j(\sigma)$. The problem is to find a schedule with the smallest maximum lateness $L^*_{\max}$ among all feasible schedules. This problem is NP-hard, even in the case of no family setup times (Lenstra, Rinnooy Kan, and Brucker [10]) and in the case of equal release dates

(Bruno and Downey [3]). In the remainder, we follow the three-field notation proposed by Graham, Lawler, Lenstra, and Rinnooy Kan [7] to classify machine scheduling problems; our problem is accordingly denoted as $1|r_j, s_i|L_{\max}$.

The presence of release dates is consistent with MRP-controlled environments. Also, the problem $1|r_j, s_i|L_{\max}$ appears as a subproblem in decomposition-based approaches for job-shop scheduling with setup times, such as the shifting bottleneck approach of Adams et al. [1]. The extension of this approach to hybrid job shops, including parallel machines at several stages, family setup times and additional resource constraints, like operators, cutting tools, and fixtures, is the focus of a research project at the University of Twente in cooperation with the part manufacturing shops of several industrial companies; see for instance Meester and Zijm [11]. The choice of minimizing maximum lateness is again motivated by industrial experiences.

Although the interest in combined batching and scheduling approaches in manufacturing is growing (see e.g. Potts and Van Wassenhove [12]), we are not aware of any research addressing this particular problem. We feel therefore that this paper fills an important gap in that it addresses a fundamental practical problem. Also, it makes a contribution in terms of algorithmic design for solving this type of NP-hard problem by branch-and-bound, in general, and in terms of lower bound computing for problems with setup times, in particular. The lower bounds that work well for the problem without family setup times, $1|r_j|L_{\max}$, including Carlier's bound (Carlier [4]) and the preemptive lower bound obtained by allowing the interruption of the processing of a job and resumption later on, can be applied to our problem only if we ignore the setup times completely, which of course may result in weak lower bounds. For instance, the preemptive lower bound obtained by solving the $1|r_j, pmtn|L_{\max}$ problem is found by Horn's algorithm in $O(n \log n)$ time (Horn [8]); in contrast, the preemptive problem $1|r_j, s_i, pmtn|L_{\max}$ is NP-hard, since $1|s_i|L_{\max}$ is (Bruno and Downey [3]).

Our key observation is that we may regard any setup as the processing of an imaginary setup job of length equal to the setup time of the family associated with it. We will develop sufficient conditions for establishing that certain jobs belonging to the same family are *not* processed in the same batch. The implication is that these jobs are *separated* by a setup job for which we can specify precedence relations, a release time, and a due date. In Section 2, we describe how the derivation and specification of the setup jobs takes place. Let $\mathcal{S}$ be the set of setup jobs that are derived in this way. For any instance $I$ of $1|r_j, s_i|L_{\max}$, we can then construct an instance $I'$ of $1|r_j, prec\,|L_{\max}$ with job set $\mathcal{J} \cup \mathcal{S}$, where *prec* indicates the presence of precedence relations between the jobs. In fact, the precedence constraints have a specific structure in our application and induce instances of what we term the $1|r_j, setup\text{-}prec|L_{\max}$ problem. The crux is that for any instance $I$ and $I'$ constructed in this way, we have that

$$L^*_{\max}(I) \geq L^*_{\max}(I'),$$

with $L^*_{\max}(I)$ and $L^*_{\max}(I')$ the optimal solution values for these instances. Hence, a lower bound on $L^*_{\max}(I)$ can be computed by computing a lower bound on $L^*_{\max}(I')$. In Section 3, we compute a lower bound on $L^*_{\max}(I')$ by solving the preemptive problem

$1|r_j$, *setup-prec*, *pmtn*$|L_{\max}$. We show that this problem is solvable in $O(n \log n)$ time due to the agreeable precedence structure. Section 4 reports on our implementation of the branch-and-bound algorithm and on our computational experiments; our results show that we can solve instances up to 40 jobs to optimality. In Section 5, we draw some conclusions and point out future research directions.

# 2 Derivation of the setup jobs

We derive two types of setup jobs: *Separating* setup jobs that have precedence relations, and *unrelated* setup jobs that have no precedence relations. We call the jobs in $\mathcal{J}$ the *real jobs* to distinguish them from the setup jobs. In the remainder, we let $\mathcal{S}$ be the set of setup jobs. Also, we let $\succ$ and $\prec$ mean 'has to follow' and 'has to precede', respectively.

In Section 2.1, we discuss the prerequisites of our approach to derive setup jobs, including a proof that a setup can indeed be seen as a setup job with a specific processing time, release date, due date, and precedence relations. We point out that the setup jobs should be consistent with each other and introduce a measure for the strength of a setup job. Finally, we also derive the so-called initial setup jobs. In Section 2.2, we discuss the logic behind the derivation of separating setup jobs and our two strategies to actually derive them. In Section 2.3, we derive a different type of setup jobs which do not involve precedence relations.

## 2.1 Preliminaries

Consider any instance $I$ of $1|r_j, s_i|L_{\max}$ and let $I'$ be the instance of $1|r_j, \textit{setup-prec}|L_{\max}$ obtained from $I$ by ignoring the family setup times. Hence, we have that $L^*_{\max}(I') \leq L^*_{\max}(I)$. Suppose now that we have established, one way or the other, that in every optimal schedule for $I$ all jobs in $\mathcal{A} \subset \mathcal{F}_i$ precede all jobs in $\mathcal{B} \subset \mathcal{F}_i$ ($\mathcal{B} \neq \emptyset$) and no job from $\mathcal{A}$ and no job from $\mathcal{B}$ are scheduled in the same batch. This then means that there must be at least one *separating* setup associated with family $\mathcal{F}_i$ between the last job belonging to $\mathcal{A}$ and the first job belonging to $\mathcal{B}$. Theorem 1 validates our key idea that this setup can be viewed as a *separating setup job* with a specific processing time, release date, due date, and precedence relations.

**Theorem 1** *We still have that $L^*_{\max}(I') \leq L^*_{\max}(I)$, if we add a setup job $J_s$ to $I'$ with*

$$
\begin{aligned}
p_s &= s_i, \\
J_s &\succ J_j, \ \textit{for all } J_j \in \mathcal{A}, \\
J_s &\prec J_j, \ \textit{for all } J_j \in \mathcal{B}, \\
r_s &= \min_{J_j \in \mathcal{F}_i \setminus \mathcal{A}} r_j - s_i, \\
d_s &= \min_{J_j \in \mathcal{B}} (d_j - p_j).
\end{aligned}
$$

**Proof** It only remains to be shown that the specification of $r_s$ and $d_s$ is correct. Consider any optimal schedule $\sigma$ for $I$ and any setup for family $\mathcal{F}_i$ that succeeds all jobs from $\mathcal{A}$ and precedes all jobs from $\mathcal{B}$ in this schedule. We associate the setup job $J_s$ with this setup. We may assume that this setup occurs immediately before the execution of the job it is needed for. Since this may be any job in $\mathcal{F}_i \setminus \mathcal{A}$, the release date of $J_s$ follows. Let $\sigma'$ be the feasible schedule for $I'$ obtained from $\sigma$ in the following way: Let the sequence of the real jobs in $\sigma'$ concur with the sequence in $\sigma$, and replace the setups with their associated setup jobs, if they have one. Note that $C_j(\sigma') \leq C_j(\sigma)$ for all $J_j \in \mathcal{J}$, and therefore $L_j(\sigma') \leq L_j(\sigma) \leq L^*_{\max}(I)$. If we assign $d_s$ as proposed, we have that $d_s = d_j - p_j$ and $J_s \prec J_j$ for some $J_j \in \mathcal{B}$, and hence that

$$L_s(\sigma') = C_s(\sigma') - d_s \leq C_j(\sigma') - p_j - (d_j - p_j) \leq C_j(\sigma) - d_j = L_j(\sigma) \leq L^*_{\max}(I).$$

Thus, we proved that $L_j(\sigma') \leq L^*_{\max}(I)$ for every job in $I'$, and therefore $L^*_{\max}(I') \leq L_{\max}(\sigma') \leq L^*_{\max}(I)$. □

The crux is that the addition of this separating setup job may improve the value $L^*_{\max}(I')$, and thus the lower bound on $L^*_{\max}(I)$. In the remainder, if we add a setup job to $I$ separating some sets $\mathcal{A}$ and $\mathcal{B}$, then we implicitly assume that it has the release date, due date and precedence relations as specified in Theorem 1.

We may not just derive setup jobs as we please. We have to make sure that the setup jobs are *consistent* with each other. For instance, if we have already derived a setup job between job sets $\mathcal{A}$ and $\mathcal{B}$, then we may not add another setup job between the subsets $\mathcal{A}' \neq \emptyset$ and $\mathcal{B}'$ if $\mathcal{A}' \subseteq \mathcal{A}$ and $\mathcal{B}' \subseteq \mathcal{B}$. To ensure the derivation of consistent setup jobs, we introduce the notion of *induction*. We say that the jobs in $\mathcal{A}$ *left-induce* $J_s$, the jobs in $\mathcal{B}$ *right-induce* $J_s$, and the setup job $J_s$ is induced by family $\mathcal{F}_i$. We construct a so-called *induction graph* $\mathcal{G} = (\mathcal{J} \cup \mathcal{S}, \mathcal{H})$, in which there is an arc $(J_s, J_j)$ in $\mathcal{H}$ with $J_s \in \mathcal{S}$ and $J_j \in \mathcal{J}$ if and only if $J_j$ right-induces $J_s$. Similarly, there is an arc $(J_j, J_s)$ in $\mathcal{H}$ if and only if $J_j$ left-induces $J_s$. The induction graph then corresponds to a set of consistent setup jobs if in its *transitive reduction*, obtained from $\mathcal{G}$ by removing all arcs that are implied by transitivity, each $J_j \in \mathcal{J}$ has at most one ingoing and at most one outgoing arc in $\mathcal{H}$. Accordingly, we may add a setup job to $I'$ if this condition remains satisfied. Throughout this section, we assume that this is so.

The *rank* of a setup job is defined as the number of jobs it separates. If $\mathcal{A} \cup \mathcal{B} = \mathcal{F}_i$, then the separation, and thereby the setup job $J_s$, is the strongest possible: We then say that $J_s$ has *full rank*. If $|\mathcal{A} \cup \mathcal{B}| < |\mathcal{F}_i|$, then in fact $J_s$ separates *at least* the job sets $\mathcal{A}$ and $\mathcal{B}$: We do not know yet on which side of $J_s$ the other jobs in $\mathcal{F}_i$ will be scheduled. The rank of $J_s$ is then equal to $|\mathcal{A} \cup \mathcal{B}|$. Intuitively, we prefer setup jobs of high rank. The aim of this section is to derive such setup jobs in the root node of the branch-and-bound tree.

One particular type of setup job of full rank is a sitting duck: For every family $\mathcal{F}_i$, we need a setup job just before the processing of its first job. Accordingly, we may introduce an *initial setup job* $J_s$ for family $\mathcal{F}_i$ with

$$p_s \;=\; s_i,$$

$$
\begin{aligned}
d_s &= \min_{J_j \in \mathcal{F}_i} (d_j - p_j), \\
J_s &\prec J_j, \text{ for all } J_j \in \mathcal{F}_i, \\
r_s &= \min_{J_j \in F_i} r_j - s_i.
\end{aligned}
$$

## 2.2 Deriving separating setup jobs

The separating setup jobs are derived through sufficient conditions for having an optimal schedule in which particular jobs of the same family are not scheduled in the same batch. We stipulate these conditions in terms of a lower bound $lb$ and an incumbent upper bound $ub$ on $L^*_{\max}(I)$, each proceeding from the assumption that $L^*_{\max}(I) < ub$. It is irrelevant how these $lb$ and $ub$ are obtained. However, the tighter $lb$ and $ub$ are, the more effective these conditions will be. In fact, there is a strong interaction between deriving setup jobs and computing lower bounds; after all, the more setup jobs are derived, the stronger the lower bound is likely to be.

The logic behind the derivation of separating setup jobs is the following. Suppose we want to put two jobs in the same batch. If the release and due dates of these jobs prohibit that these jobs are scheduled contiguously, then the machine is idle in between their processing, if no other job belonging to the same family is available for processing. If this idle time period $T$ is too long, then *saving a single setup does not make up for what is essentially a loss of machine capacity.* We have two strategies to conclude that $T$ is effectively too long: (i) If $T$ is so long that we can perform a setup for family $\mathcal{F}_i$ in the meantime, and (ii) If a lower bound for the case that we leave the machine idle during period $T$ is equal to or larger than the incumbent upper bound. We formalize these strategies below.

In any optimal schedule, each $J_j$ is scheduled somewhere in the interval $[r_j, d_j + L^*_{\max}(I)]$ $(j = 1, \ldots, n)$. Accordingly, if $L^*_{\max}(I) < ub$, then the largest possible completion time of $J_j$ is $\bar{d}_j = d_j + ub - 1$. We call job $J_j$ *safely scheduled* if $r_j + p_j \leq C_j \leq d_j + lb$; note that if each job is safely scheduled, then we have an optimal solution, since $L^*_{\max}(I) \leq lb$. For any job set $\mathcal{A}$, let $r(\mathcal{A}) = \min_{J_j \in \mathcal{A}} r_j$, and $\bar{d}(\mathcal{A}) = \max_{J_j \in \mathcal{A}} \bar{d}_j$; note that a necessary condition for having $L^*_{\max}(I) < ub$ is that all jobs in $\mathcal{A}$ are completed by time $\bar{d}(\mathcal{A})$.

We are now ready to make the following observation, which plays a key role in the derivation of the setup jobs.

**Observation 1** *Consider disjoint subsets $\mathcal{A} \subset \mathcal{F}_i$ and $\mathcal{B} \subset \mathcal{F}_i$ with $\bar{d}(\mathcal{A}) < r(\mathcal{B})$. If there exists a schedule $\sigma$ with $L_{\max}(\sigma) < ub$ that puts jobs from both $\mathcal{A}$ and $\mathcal{B}$ in the same batch, then it has the following properties:*

- *The machine is idle during the period $T = [\bar{d}(\mathcal{A}), r(\mathcal{B})]$, if there is no job $J_j \in \mathcal{F}_i \setminus (\mathcal{A} \cup \mathcal{B})$ available for processing during period $T$. This means that the machine is definitely idle during period $T$ if $\mathcal{A} \cup \mathcal{B} = \mathcal{F}_i$.*

- *The batch spans at least the interval $[\max_{J_j \in \mathcal{A}}(\bar{d}_j - p_j), \min_{J_j \in \mathcal{B}}(r_j + p_j)]$.*

As pointed out before, a long idle time period $T$ makes it unlikely that there indeed exists an optimal schedule in which some job from $\mathcal{A}$ and some job from $\mathcal{B}$ are scheduled in the same batch. Or equivalently, a long period $T$ makes it likely that there exists an optimal schedule in which no job from $\mathcal{A}$ and no job from $\mathcal{B}$ are scheduled in the same batch.

It is not sensible, even if it were possible, to consider all possible $\mathcal{A}$ and $\mathcal{B}$. The following subsets enable systematic procedures for deriving setup jobs. Let $J^i_{[j]} \in \mathcal{F}_i$ denote the job with the $j$th smallest release date in family $\mathcal{F}_i$, and let $r^i_{[j]}$, $\bar{d}^i_{[j]}$, and $p^i_{[j]}$ be its release date, largest deadline possible, and processing time, respectively. For any family $\mathcal{F}_i$ and any $a = 1, \ldots, |\mathcal{F}_i|$ and $b = a, \ldots, |\mathcal{F}_i|$ define

$$\mathcal{P}^i_{a,b} = \{J^i_{[a]}, \ldots, J^i_{[b]}\}.$$

Note that by definition we have that $r^i_{[k+1]} = r(\mathcal{P}^i_{k+1,|\mathcal{F}_i|})$. The advantage of considering these subsets is that there is a fair chance of having $\bar{d}(\mathcal{P}^i_{1,k}) < r^i_{[k+l]}$, for some $l \geq 1$, a prerequisite for deriving a separating setup job.

The following theorem gives an effective means for deriving setup jobs. It says that if $T$ is large enough to accommodate a setup for family $\mathcal{F}_i$, then we may already introduce a setup job of full rank.

**Theorem 2** *Suppose $L^*_{\max}(I) < ub$. If there is a family $\mathcal{F}_i$ $(i = 1, \ldots, m)$ and an index $k$ $(k = 1, \ldots, |\mathcal{F}_i| - 1)$, for which*

$$\bar{d}(\mathcal{P}^i_{1,k}) + s_i \leq r(\mathcal{P}^i_{k+1,|\mathcal{F}_i|}), \tag{1}$$

*then we may introduce a setup job $J_s$ of full rank that separates $\mathcal{P}^i_{1,k}$ from $\mathcal{P}^i_{k+1,|\mathcal{F}_i|}$.*

**Proof** Let $\sigma$ be any optimal schedule. There are two cases to consider:

(i) There is no batch in $\sigma$ that contains a job from $\mathcal{P}^i_{1,k}$ as well as a job from $\mathcal{P}^i_{k+1,|\mathcal{F}_i|}$. In this case, there is a setup between $\mathcal{P}^i_{1,k}$ and $\mathcal{P}^i_{k+1,|\mathcal{F}_i|}$.

(ii) There is a batch in $\sigma$ that contains a job from $\mathcal{P}^i_{1,k}$ as well as a job from $\mathcal{P}^i_{k+1,|\mathcal{F}_i|}$. In this case, the machine is idle between $\bar{d}(\mathcal{P}^i_{1,k})$ and $r^i_{[k+1]}$; see Observation 1. We can then transform $\sigma$ into an equivalent schedule in which a setup, performed during period $T = [\bar{d}(\mathcal{P}^i_{1,k}), r^i_{[k+1]}]$, splits this batch into two consecutive batches of the same family.

Therefore, we may assume that in every optimal solution a setup separates $\mathcal{P}^i_{1,k}$ and $\mathcal{P}^i_{k+1,|\mathcal{F}_i|}$. □

The next theorem is a generalization of Theorem 2 to derive setup jobs of smaller rank. If we cannot separate the sets $\mathcal{P}^i_{1,k}$ and $\mathcal{P}^i_{k+1,|\mathcal{F}_i|,}$, then we may try to separate the sets $\mathcal{P}^i_{1,k}$ and $\mathcal{P}^i_{k+l,|\mathcal{F}_i|,}$ for some $l \geq 2$. After all, the larger $l$ is, the longer the idle time period $T$ gets if we want to put some jobs belonging to these sets in the same batch. The condition for testing if $T$ gets too long is similar to condition (1), albeit period $T$ should also have room to accommodate the 'in-between jobs' $J^i_{[k+1]}, \ldots, J^i_{[k+l-1]}$.

**Theorem 3** *Suppose $L^*_{\max}(I) < ub$. If there is a family $\mathcal{F}_i$ $(i = 1, \ldots, m)$, an index $k$ $(k = 1, \ldots, |\mathcal{F}_i| - 1)$, and an index $l$ $(l = 1, \ldots, |\mathcal{F}_i| - k)$ such that the interval*

$$[\bar{d}(\mathcal{P}^i_{1,k}), r(\mathcal{P}^i_{k+l,|\mathcal{F}_i|})] \tag{2}$$

*is large enough to safely schedule each of the jobs $J^i_{[k+1]}, \ldots, J^i_{[k+l-1]}$ and a setup for family $\mathcal{F}_i$ in it, then we may introduce a setup job $J_s$ of rank $|\mathcal{F}_i| - l + 1$ that separates the job sets $\mathcal{P}^i_{1,k}$ and $\mathcal{P}^i_{k+l,|\mathcal{F}_i|}$.* □

We now come to our second strategy to derive setup jobs. Suppose $lb(\mathcal{A}, \mathcal{B})$ is a lower bound for the case that some unspecified job from $\mathcal{A}$ and some unspecified job from $\mathcal{B}$ are scheduled in the same batch. If $lb(\mathcal{A}, B) \geq ub$, then the sets $\mathcal{A}$ and $\mathcal{B}$ are obviously separated in any optimal schedule if $L^*_{\max}(I) < ub$. In Section 3, we show how we compute such a bound.

**Theorem 4** *Suppose $L^*_{\max}(I) < ub$. If $\bar{d}(\mathcal{P}^i_{1,k}) < r(\mathcal{P}^i_{k+l,|\mathcal{F}_i|}) - s_i$ and*

$$lb(\mathcal{P}^i_{1,k}, \mathcal{P}^i_{k+l,|\mathcal{F}_i|}) \geq ub, \tag{3}$$

*for some $i$, $k$ and $l$ with $1 \leq i \leq m$, $1 \leq k < |\mathcal{F}_i|$ and $1 \leq l \leq |\mathcal{F}_i| - k$, then we may introduce a setup job of rank $|\mathcal{F}_i| - l + 1$ that separates $\mathcal{P}^i_{1,k}$ from $\mathcal{P}^i_{k+l,|\mathcal{F}_i|}$.* □

## 2.3 Deriving unrelated setup jobs

The derivation of unrelated setup jobs, which have no precedence relations, proceeds by the premise that batches of different families cannot overlap in time. Suppose that $\bar{d}(\mathcal{P}^i_{1,k}) < r(\mathcal{P}^i_{k+l,|\mathcal{F}_i|})$ and $\bar{d}(\mathcal{P}^h_{1,a}) < r(\mathcal{P}^h_{a+b,|\mathcal{F}_h|})$ and the intervals $[\bar{d}(\mathcal{P}^i_{1,k}),\ r(\mathcal{P}^i_{k+l,|\mathcal{F}_i|})]$ and $[\bar{d}(\mathcal{P}^h_{1,a}),\ r(\mathcal{P}^h_{a+b,|\mathcal{F}_h|})]$ overlap in time; that is, there is a point in time $t$ such that $\bar{d}(\mathcal{P}^i_{1,k}) \leq t \leq r(\mathcal{P}^i_{k+l,|\mathcal{F}_i|})$ and $\bar{d}(\mathcal{P}^h_{1,a}) \leq t \leq r(\mathcal{P}^h_{a+b,|\mathcal{F}_h|})$, with at least one of these $\leq$ signs holding as a strict inequality. The conclusion must then be that we may at least separate either $\mathcal{P}^i_{1,k}$ and $\mathcal{P}^i_{k+l,|\mathcal{F}_i|}$, or $\mathcal{P}^h_{1,a}$ and $\mathcal{P}^h_{a+b,|\mathcal{F}_h|}$, since the machine can process no more than one batch at a time. We may therefore introduce a setup job, but it has no precedence relations, since we cannot associate the setup job with either family. For the same reason, these unrelated setup jobs are quite weak. They have rank 0, and their release and due dates are not very tight either.

**Theorem 5** *If there are two families $\mathcal{F}_i$ and $\mathcal{F}_h$ and indices $k$, $l$, $a$ and $b$ for which the time intervals $[\bar{d}(\mathcal{P}^i_{1,k}),\ r(\mathcal{P}^i_{k+l,|\mathcal{F}_i|})]$ and $[\bar{d}(\mathcal{P}^h_{1,a}),\ r(\mathcal{P}^h_{a+b,|\mathcal{F}_h|})]$ overlap, then we may introduce a setup job $J_s$ of rank 0 with*

$$\begin{aligned}
p_s &= \min\{s_i, s_h\}, \\
r_s &= \min\{r(\mathcal{P}^i_{k+1,|\mathcal{F}_i|}) - s_i,\ r(\mathcal{P}^h_{a+1,|\mathcal{F}_h|}) - s_h\}, \\
d_s &= \max\{\min_{J_j \in \mathcal{P}^i_{k+l,|\mathcal{F}_i|}} (d_j - p_j),\ \min_{J_j \in \mathcal{P}^h_{a+b,|\mathcal{F}_h|}} (d_j - p_j)\}.
\end{aligned}$$

□

Obviously, any number of families may be involved in this type of derivation, but the resulting setup jobs will then be even weaker.

# 3 Lower Bounds

In this section, we present first the preemptive lower bound for the $1|r_j, \textit{setup-prec}|L_{\max}$ problem. Then, we show how we compute the bound $lb(\mathcal{P}^i_{1,k}, \mathcal{P}^i_{k+l,|\mathcal{F}_i|})$ needed in condition (3) to derive setup jobs.

First of all, however, we characterize the acyclic directed precedence graph $G$ induced by any set $\mathcal{S}$ of consistent setup jobs. We assume that $S$ contains for each family at least the initial setup job. Let $\mathcal{S}_i$ be the set of separating setup jobs induced by the jobs in $\mathcal{F}_i$. We have as vertex set $V = \mathcal{J} \cup \mathcal{S}$ and there is an arc $(J_j, J_k)$ if and only if $J_j \prec J_k$. If there is an arc $(J_j, J_k)$, then $J_j$ is an *immediate predecessor* of $J_k$ and $J_k$ is an *immediate successor* of $J_j$. If there is a path in $G$ from $J_j$ to $J_k$, then $J_j$ is a predecessor of $J_k$; $J_k$ is then a successor of $J_j$. There are no arcs between the unrelated setup jobs and the real jobs.

Let $G' = (V, A)$ be the transitive reduction of this graph, where $A$ denotes the remaining arc set. The release date of an initial setup job or a setup job derived by use of condition (3) may not be consistent with the precedence constraints; i.e., we may have that $r_s < r_j$ for some $J_j \prec J_s$. We therefore modify the release dates in the following way:

$$r_j \leftarrow \max\{r_j, \max_{J_k \prec J_j}(r_k + p_k)\} \text{ for all } J_j \in \mathcal{J} \cup \mathcal{S}.$$

This modification does not affect the optimal solution.

The graph $G'$ then has the following properties:

- It decomposes into $m$ arc-disjoint connected subgraphs, one for every family, on the one hand, and isolated vertices representing the unrelated setup jobs, on the other hand.
- For any arc $(J_j, J_k) \in A$, we have that $r_j + p_j \leq r_k$.
- For any arc $(J_j, J_k) \in A$, we have that if $J_j \in \mathcal{J}$, then $J_k \in \mathcal{S}$, and, conversely, if $J_j \in \mathcal{S}$, then $J_k \in \mathcal{J}$.
- Each job in $\mathcal{J}$ has at most one immediate successor and at most one immediate predecessor.
- There are $O(n)$ arcs; this means that the release date modification can be carried out in $O(n)$ time.

The following lemma stipulates a most agreeable property of $G'$.

**Lemma 1** *If $J_j \prec J_k$, then we have that $d_j \leq d_k - p_k$.*

**Proof**. Suppose $J_j$ is a setup job; $J_k$ is then a real job. Then, due to the way the separating setup jobs are specified, we have that $d_j \leq d_k - p_k$; see Theorem 1. Now, suppose $J_j$ is a real job; $J_k$ is then a separating setup job. Let $J_l \in \mathcal{J}$ be a successor of $J_k$ in $G'$ with $d_k = d_l - p_l$. Such a successor always exists; see Theorem 1. Due to the way the separating setup jobs are derived, we have that $\bar{d}_j < r_l - s_{f(l)}$. Since $r_l + p_l \leq d_l + ub$ and $p_k = s_{f(l)}$, we have that $d_j + ub - 1 < d_l - p_l - p_k + ub$, and hence that $d_j \leq d_k - p_k$. □

## 3.1 The preemptive bound

The $1|r_j, prec, pmtn|L_{\max}$ problem is solvable by Horn's rule (Horn [8]) after release and due date modification in $O(n^2)$ time. For the $1|r_j, setup\text{-}prec, pmtn|L_{\max}$ problem, the modification of the release dates takes $O(n)$ time only; modification of the due dates is not necessary, since we have that $d_j \leq d_k - p_k$ if $J_j \prec J_k$. Hence, we have the following result, the proof of which is included for sake of completeness.

**Theorem 6** *The $1|r_j, setup\text{-}prec, pmtn\,|L_{\max}$ is solvable in $O(n \log n)$ time by the following rule: At any time schedule an available job with the smallest due date.*

**Proof**. First of all, note that Horn's rule generates a feasible schedule for the problem $1|r_j, setup\text{-}prec, pmtn|L_{\max}$. This is true, since if $J_j \prec J_k$, then we have that $r_j + p_j \leq r_k$ and $d_j \leq d_k - p_k$; see the proof of Lemma 1.

Let now $\pi$ be the schedule produced by Horn's rule, and let $\sigma$ be any optimal schedule. We shall prove that we can transform $\sigma$ into $\pi$ by rescheduling jobs while preserving feasibility and optimality. Compare $\sigma$ with $\pi$ from time 0 onwards, and let $t$ be the first time at which the schedules start to differ: Suppose $J_j$ is scheduled between time $t$ and $t_1$ in $\sigma$ and $J_k$ is scheduled between time $t$ and $t_2$ in $\pi$. Let $\tau = \min\{t_1, t_2\}$. Find time $s > \tau$ that designates the smallest interval $[t, s]$ in $\sigma$ in which $J_k$ is processed for exactly $\tau - t \leq p_k$ units of time. Let $\mathcal{A}$ be the set of successors of $J_j$ in $G$ that are scheduled between $t$ and $s$ in $\sigma$. We then have that

$$d_j \leq d_l \quad \text{for all } J_l \in \mathcal{A}.$$

Also, since $J_k$ is scheduled at time $t$ in $\pi$, not $J_j$, we have that

$$d_k \leq d_j,$$

Hence, the following transformation of $\sigma$ retains both feasibility and optimality:

- Remove all portions of $J_j$, $J_k$ and the jobs in $\mathcal{A}$ between time $t$ and $s$, but leave the other jobs intact.
- Schedule $J_k$ in the time slot $[t, \tau]$.
- Schedule $J_j$ and the jobs in $\mathcal{A}$ in the remaining available time slots between $\tau$ and $s$ in the same order as before.

Now let $t \leftarrow \tau$, and repeat the argument till we reach the end of the schedule; both schedules are then identical. □

This rule can evidently be implemented in $O(n \log n)$ time, since there are $n$ real and no more than $n$ setup jobs, there are $O(n)$ preemptions, and the release and due dates of the available jobs need only to be maintained in a partial order.

### 3.2 Computing the bound $lb(\mathcal{P}^i_{1,k}, \mathcal{P}^i_{k+l,|\mathcal{F}_i|})$

The bound $lb(\mathcal{P}^i_{1,k}, \mathcal{P}^i_{k+l,|\mathcal{F}_i|})$, needed for condition (3), is a lower bound for scheduling some unspecified job in $\mathcal{P}^i_{1,k}$ in the same batch, say, $B$, as some unspecified job in $\mathcal{P}^i_{k+l,|\mathcal{F}_i|}$ ($i = 1, \ldots, m$, $1 \leq k < |\mathcal{F}_i|$, $0 \leq l \leq |\mathcal{F}_i| - 1$). We assume that some separating and unrelated setup jobs already have been derived and that the setup job that may be induced by this bound is consistent with them.

If we decide to schedule some job from $\mathcal{P}^i_{1,k}$ and some job from $\mathcal{P}^i_{k+l,|\mathcal{F}_i|}$ in the same batch, say, $B$, then $B$ spans at least the interval $T = [t_1, t_2]$, where

$$t_1 = \max_{J_j \in \mathcal{P}^i_{1,k}} (\bar{d}_j - p_j),$$

and

$$t_2 = \min_{J_j \in \mathcal{P}^i_{k+l,|\mathcal{F}_i|}} (r_j + p_j);$$

see Observation 1. We assume that $t_2 > t_1$. If not, then we let $lb(\mathcal{P}^i_{1,k}, \mathcal{P}^i_{k+l,|\mathcal{F}_i|}) = -\infty$.

Let $I'$ be any instance of the $1|r_j, \textit{setup-prec}|L_{\max}$ problem with the condition that we schedule those unspecified jobs in the same batch. To compute a lower bound, we construct an instance $I''$ with the additional constraint that the machine is not available for processing during the interval $T = [t_1, t_2]$. We initialize $I'' = I'$ and then remove all jobs $\mathcal{J}_j \in \mathcal{F}_i \cup \mathcal{S}_i$ from $I''$ for which the time intervals $[t_1, t_2]$ and $[r_j, \bar{d}_j]$ overlap; we do this to ensure that $L^*_{\max}(I'')$ is a valid lower bound on $L^*_{\max}(I')$.

Moreover, we try to derive more separating setup jobs for each family other than $\mathcal{F}_i$. If the machine is not available during the period $T = [t_1, t_2]$, then any two jobs $J_j$ and $J_k$ cannot be in the same batch if $r_j > t_1 - p_j$ and $\bar{d}_k < t_2 - p_k$; after all, $J_j$ must then be processed after period $T$ and $J_k$ before period $T$. So, if $\mathcal{C}_h = \{J_j \in \mathcal{F}_h \mid \bar{d}_j < t_2 + p_j\}$ and $\mathcal{D}_h = \{J_j \in \mathcal{F}_h \mid r_j > t_1 - p_j\}$ and $\mathcal{C}_h \neq \emptyset$ and $\mathcal{D}_h \neq \emptyset$, then we may add a setup job $J_s$ to $I''$ that separates the sets $\mathcal{C}_h$ and $\mathcal{D}_h$ for any family $\mathcal{F}_h \neq \mathcal{F}_i$, if this setup job is consistent with the other setup jobs.

We now compute the preemptive lower bound for $I''$ subject to the condition that the machine is not available during period $T$. We can easily cope with this condition by adding an independent dummy job $J_0$ to $I''$ with $r_0 = t_1$, $p_0 = t_2 - t_1$, and $d_0 = \min_{J_j \in \mathcal{J}} d_j - 1$. Horn's rule schedules $J_0$ then in period $T$, and we compute $lb(\mathcal{P}^i_{1,k}, \mathcal{P}^i_{k+l,|\mathcal{F}_i|})$ as $\max_{J_j \in I'' \setminus \{J_0\}} L_j$.

# 4 Implementation and computational experiments

## 4.1 Implementation

We have developed a branch-and-bound algorithm that uses a forward sequencing branching rule, in which a node at level $k$ ($k = 1, \ldots, n$) corresponds to an active partial schedule

consisting of $k$ jobs. A node at level $k$ has $n-k$ descendant nodes, one for each unscheduled job. We branch from the nodes in order of non-decreasing release dates of the jobs associated with the nodes.

In the root node of the tree, we run a two-phase randomized local-search algorithm to find a good initial upper bound *ub*; it uses simulated annealing first and then tries to improve the solution by tabu search. The neighborhood of a feasible sequence is in either phase defined as the set of sequences obtained by either relocating any single job, or swapping any two jobs in the sequence. In fact, both the simulated annealing and the tabu search subroutines are straightforward implementations of the basic principles, as outlined in for instance Van Laarhoven and Aarts [9] and Glover [6]. Also, we use several simple but effective dominance criteria to restrict the growth of the branch-and-bound tree. Given this upper bound, we iteratively derive as many and as strong as possible consistent setup jobs. Deriving setup jobs is computationally expensive; for this reason, it is carried out only in the root node of the branch-and-bound tree. In the nodes of the tree, it is too time-consuming to compute the preemptive bound, although it takes only $O(n \log n)$. We have extended Carlier's lower bound [4] for the $1|r_j|L_{\max}$ to deal with precedence relations and compute this bound instead; this bound requires only $O(n)$ time in each node. For a detailed description of our implementation, we refer to Schutten, Van de Velde, and Zijm [13].

## 4.2 Computational experiments

The performance of the branch-and-bound algorithm was evaluated for instances with up to 50 jobs. All parameters were randomly generated from discrete uniform distributions, except for the release times that come from a Poisson distribution. The processing times were drawn from the discrete uniform distribution $[1, 100]$, the number of families $m$ from the uniform distribution $[2, \lfloor n/5 \rfloor]$, and the family indices of the jobs from the uniform distribution $[1, m]$. Let $\bar{p}$ denote the average job processing time. In addition to $n$, there are four input parameters:

- $s$, defining the interval $[1, s \cdot \bar{p}]$ from which the setup times are uniformly drawn,
- $a$ and $k$, defining the mean inter arrival time $(\bar{p} + a \cdot \bar{s})/k$ of the Poisson distribution from which the release times are drawn, where $\bar{s}$ is the average setup time, and
- $d$, defining the interval $[r_j + p_j, r_j + p_j + d \cdot \bar{p}]$ from which $d_j$ is uniformly drawn.

We generated instances for $n = 30, 40, 50$, $s = 0.25, 0.50, 0.75$, $a = 0.25, 0.33, 0.5$, $k = 0.8, 0.9$ and $d = 2, 4, 6$. For each combination of $n$, $s$, $a$, $k$, and $d$, we generated 15 instances. Table 1 gives a summary of our computational results for varying values of $n$, the number of jobs, and $k$, determining the arrival intensity. Crudely speaking, we can say that $k$ determines the workload in the shop: The larger $k$, the higher the workload. We found that the performance of the branch-and-bound algorithm does not significantly vary with the other parameters. The column '*#opt*' gives the number of instances out of 405

for which the branch-and-bound algorithm found an optimal solution within one minute on a HP 9000/710 workstation. It shows that we virtually solve all instances with $n = 30$. The next two columns give averages only for the instances solved to optimality within one minute. The column '*#nodes*' gives the average number of nodes searched, and the column '*seconds*' gives the average computing time in seconds that the algorithm takes. The time for the preprocessing phase, i.e., for running the approximation algorithms and deriving the setup jobs, is not included here. The preprocessing phases typically takes about 2 to 4 seconds on the HP. Table 1 shows that the instances get more difficult with increasing number of jobs, as expected, and with increasing value of $k$. If the workload is high, i.e., if there are many jobs available at the same time for processing, then it is more difficult to derive setup jobs of high rank, and consequently, our lower bounds get less effective with increasing value of $k$. Table 1 also shows that the instances that we can solve within the time limit take little time on average. This suggests a considerable watershed between computationally easy and hard instances.

| $n$ | $k$ | *#opt* | *#nodes* | *seconds* |
|---|---|---|---|---|
| 30 | 0.8 | 401 | 35,614 | 0.7 |
| 30 | 0.9 | 395 | 48,688 | 0.9 |
| 40 | 0.8 | 385 | 40,357 | 0.9 |
| 40 | 0.9 | 355 | 107,832 | 2.4 |
| 50 | 0.8 | 358 | 83,544 | 2.1 |
| 50 | 0.9 | 295 | 131,112 | 3.1 |

Table 1: Performance of the branch-and-bound algorithm.

Table 2 gives for varying $n$ and $k$ the results of the preprocessing step for those instances that were solved to optimality within one minute. The column '$lb1$' gives the average preemptive lower bound in the root node of the search tree without the addition of derived setup jobs. The column '$lb2$' gives this lower bound, but now with the addition of the setup jobs. The average value of the initial solution found by our approximation algorithm is found in the column '*ub*'. The average optimal solution value is given in the column '*opt*'. We see that the gap between the initial lower bound $lb1$ and the optimal solution value *opt* is approximately halved by the addition of the setup jobs. The average number of derived setup jobs is given in the column '*derived*', whereas the average number of setups in the optimal solution we found is given in the column '*setups*'. Note that in general there exist more than one optimal solution and each may have a different number of setups.

As far as possible, Table 3 gives the same information for those instances for which the algorithm failed to find an optimal solution within one minute. In comparison to Table 2, we have added the column $ub^*$, which gives the average value of the incumbent upper bound after one minute of computation time.

Our computational results did not reveal any relation between the difficulty of an instance and the choices of the parameters $a$, $s$, and $d$; the difficulty of an instance primarily

| $n$ | $k$ | $lb1$ | $lb2$ | $ub$ | $opt$ | $derived$ | $setups$ |
|---|---|---|---|---|---|---|---|
| 30 | 0.8 | 96.4 | 125.1 | 154.0 | 152.8 | 14.4 | 18.4 |
| 30 | 0.9 | 120.8 | 152.7 | 188.1 | 186.5 | 12.8 | 17.4 |
| 40 | 0.8 | 118.3 | 153.9 | 185.0 | 183.9 | 19.6 | 25.7 |
| 40 | 0.9 | 150.5 | 188.5 | 226.7 | 224.7 | 17.2 | 24.2 |
| 50 | 0.8 | 126.5 | 171.3 | 204.3 | 202.6 | 25.5 | 33.5 |
| 50 | 0.9 | 158.5 | 200.1 | 240.4 | 237.7 | 22.4 | 31.2 |

Table 2: Results of preprocessing: solvable instances.

| $n$ | $k$ | $lb1$ | $lb2$ | $ub$ | $ub^*$ | $derived$ |
|---|---|---|---|---|---|---|
| 30 | 0.8 | 342.2 | 409.2 | 468.0 | 468.0 | 8.8 |
| 30 | 0.9 | 375.1 | 433.1 | 522.3 | 520.5 | 7.6 |
| 40 | 0.8 | 292.4 | 374.2 | 471.7 | 470.1 | 13.2 |
| 40 | 0.9 | 334.5 | 425.0 | 517.3 | 515.8 | 12.8 |
| 50 | 0.8 | 284.4 | 377.5 | 471.6 | 468.9 | 19.1 |
| 50 | 0.9 | 332.8 | 430.4 | 533.5 | 532.1 | 17.8 |

Table 3: Results of preprocessing: hard instances.

depends on how close the release dates are to each other. Close release dates are most likely to occur in case of a high workload parameter $k$. The performance of the algorithm deteriorates in case of close release dates for two reasons. First, such release dates in combination with the almost agreeable due dates lead to a considerable lateness. This makes that $\bar{d}_j$, the latest possible completion time of job $J_j$, is relatively large; we then may expect to have few sets for which $\bar{d}(\mathcal{P}^i_{1,k}) < r(\mathcal{P}^i_{k+l,|\mathcal{F}_i|})$, and as a result, fewer setup jobs and thereby weaker lower bounds. Second, if the release dates are close to each other, then certain dominance criteria in our branch-and-bound algorithm are less effective. Indeed, Table 3 confirms our expectations: it shows that difficult instances have larger $L^*_{\max}$ and permit fewer setup jobs than the solvable instances.

# 5 Conclusions

We have addressed a practical scheduling problem in manufacturing arising from the fundamental controversy between efficient production and due date performance. We have presented a branch-and-bound algorithm that solves instances of reasonable size to optimality. Our major contribution is a lower bounding strategy that proceeds by ignoring the setup times and replacing them by setup jobs. This strategy induces a relaxed problem with specific precedence constraints such that its preemptive version is solvable in

$O(n \log n)$ time. We are currently investigating to what extent this strategy is useful for other scheduling problems with family setup times, including the parallel machine scheduling problem.

The algorithm described in this paper is now included in JOBPLANNER, a commercial shop floor control system, which resulted from the cooperation between the Production and Operations Management Group of the University of Twente and a consultancy firm. It is presently operational at two companies that deal with major setup times, including a manufacturer of printed circuit boards (PCBs), where the production is organized as a reentrant flowshop. Each PCB has essentially the same routing with about 25 operations. The base material of a PBC is teflon or epoxy. JOBPLANNER has proved to be specifically useful for scheduling machines where large setup times occur when production is switched from using one base material to the other. In the past, the operators at the shop floor had little insight when to switch from one base material to the other – currently, the schedule is made at a higher level by the production manager. The company is enthousiastic about JOBPLANNER, because of the quality of the resulting schedules and the reduction in the effort to schedule the shop. Scheduling used to be a full-time job; now, it only takes one or two hours a day.

**Acknowledgement** The authors thank Johann Hurink and the referees for their constructive comments.

# References

[1] J. Adams, E. Balas, and D. Zawack. The shifting bottleneck procedure for job shop scheduling. *Management Science*, 34:391–401, 1988.

[2] J.D. Blackburn. *Time-Based Competition, The Next Battle Ground in American Manufacturing*. Richard D. Irwin, Homewood, Ill., 1991.

[3] J. Bruno and P. Downey. Complexity of task sequencing with deadlines, set-up times and changeover costs. *SIAM Journal on Computing*, 7:393–404, 1978.

[4] J. Carlier. The one-machine sequencing problem. *European Journal of Operational Research*, 11:42–47, 1982.

[5] W.E. Deming. *Quality, Productivity, and Competitive Position*. MIT Center for Advanced Engineering Study, Cambridge, Mass., 1982.

[6] F. Glover. Tabu search - Part I. *ORSA Journal on Computing*, 1:190–206, 1989.

[7] R.L. Graham, E.L. Lawler, J.K. Lenstra, and A.H.G. Rinnooy Kan. Optimization and approximation in deterministic sequencing and scheduling: A survey. *Annals of Discrete Mathematics*, 5:287–326, 1979.

[8] W.A. Horn. Some simple scheduling algorithms. *Naval Research Logistics Quaterly*, 21:177–185, 1974.

[9] P.J.M. Van Laarhoven and E.H.L. Aarts. *Simulated Annealing: Theory and Applications*. Reidel, Dordrecht, The Netherlands, 1987.

[10] J.K. Lenstra, A.H.G. Rinnooy Kan, and P. Brucker. Complexity of machine scheduling problems. *Annals of Discrete Mathematics*, 1:343–362, 1977.

[11] G.J. Meester and W.H.M. Zijm. Multi-resource scheduling for an FMC in discrete parts manufacturing. In M.M. Ahmad and W.G. Sullivan, editors, *Flexible Automation and Integrated Manufacturing*, pages 360–370. CRC Press Inc., Atlanta, 1993.

[12] C.N. Potts and L.N. van Wassenhove. Integrating scheduling with batching and lot-sizing: a review of algorithms and complexity. *Journal of the Operational Research Society*, 43:395–406, 1992.

[13] J.M.J. Schutten, S.L. van de Velde, and W.H.M. Zijm. Single-machine scheduling with release dates, due dates and family setup times. Technical Report LPOM-93-4, University of Twente, Department of Mechanical Engineering, 1993.