

A time-indexed formulation for single-machine scheduling problems : branch-and-cut

Citation for published version (APA):

Akker, van den, J. M., Hurkens, C. A. J., & Savelsbergh, M. W. P. (1995). *A time-indexed formulation for single-machine scheduling problems : branch-and-cut*. (Memorandum COSOR; Vol. 9524). Technische Universiteit Eindhoven.

Document status and date:

Published: 01/01/1995

Document Version:

Publisher's PDF, also known as Version of Record (includes final page, issue and volume numbers)

Please check the document version of this publication:

- A submitted manuscript is the version of the article upon submission and before peer-review. There can be important differences between the submitted version and the official published version of record. People interested in the research are advised to contact the author for the final version of the publication, or visit the DOI to the publisher's website.
- The final author version and the galley proof are versions of the publication after peer review.
- The final published version features the final layout of the paper including the volume, issue and page numbers.

[Link to publication](#)

General rights

Copyright and moral rights for the publications made accessible in the public portal are retained by the authors and/or other copyright owners and it is a condition of accessing publications that users recognise and abide by the legal requirements associated with these rights.

- Users may download and print one copy of any publication from the public portal for the purpose of private study or research.
- You may not further distribute the material or use it for any profit-making activity or commercial gain
- You may freely distribute the URL identifying the publication in the public portal.

If the publication is distributed under the terms of Article 25fa of the Dutch Copyright Act, indicated by the "Taverne" license above, please follow below link for the End User Agreement:

www.tue.nl/taverne

Take down policy

If you believe that this document breaches copyright please contact us at:

openaccess@tue.nl

providing details and we will investigate your claim.

A Time-Indexed Formulation for Single-Machine Scheduling Problems: Branch-and-Cut

J.M. van den Akker

Center for Operations Research and Econometrics
Catholic University of Louvain
Voie du Roman Pays 34, 1348 Louvain-la-Neuve, Belgium

C.A.J. Hurkens

Department of Mathematics and Computing Science
Eindhoven University of Technology
P.O. Box 513, 5600 MB Eindhoven, The Netherlands

M.W.P. Savelsbergh

School of Industrial and Systems Engineering
Georgia Institute of Technology
Atlanta, GA 30332-0205, U.S.A.

Abstract

In Van den Akker, Van Hoesel, and Savelsbergh [1994], we have studied a time-indexed formulation for single-machine scheduling problems and have presented a complete characterization of all facet inducing inequalities with right-hand sides 1 and 2 of the convex hull of the monotone extension of the set of feasible schedules. In this paper, we discuss the development of a branch-and-cut algorithm based on these facet inducing inequalities. We describe separation algorithms for each class of these inequalities, and elaborate on various other important components of the branch-and-cut algorithm, such as branching strategies, cut generation schemes, and primal heuristics. We present our computational experiences with the algorithm for the problem of minimizing the total weighted completion time on a single machine subject to release dates.

Key words: scheduling, separation, branch-and-cut.

1 Introduction

Scheduling problems are concerned with the optimal allocation of resources to a set of tasks or activities over time. These resources are generally scarce so the allocation

inevitably gives rise to competition among tasks that are vying for their use. For an excellent survey of sequencing and scheduling see Lawler, Lenstra, Rinnooy Kan, and Shmoys [1993]. We restrict our attention to single-machine scheduling problems, i.e., problems in which there is only a single resource. Many single-machine scheduling problems are naturally formulated as integer programs with variables indexed by pairs (j, t) , where j denotes a job and t denotes a time period. These formulations are commonly referred to as *time-indexed* formulations and have been studied by several researchers (Sousa [1989], Sousa and Wolsey [1992], Crama and Spieksma [1993], Van den Akker, Van Hoesel, and Savelsbergh [1994], Van den Akker [1994]). For a survey of these research activities and of other formulations that have been proposed for single-machine scheduling problems we refer to Queyranne and Schulz [1994].

In Van den Akker, Van Hoesel, and Savelsbergh [1994], we have studied the polyhedral structure associated with a time-indexed formulation for single-machine scheduling problems and have presented a complete characterization of all facet inducing inequalities with right-hand sides 1 and 2 of the convex hull of the monotone extension of the set of feasible schedules. In this paper, we discuss the development of a branch-and-cut algorithm based on these facet inducing inequalities and we present our computational experiences with the algorithm for the problem of minimizing the total weighted completion times on a single-machine subject to release dates, i.e., $1|r_j|\sum w_j C_j$, which is known to be NP-hard (Lenstra, Rinnooy Kan, and Brucker [1977]). Developing a branch-and-cut algorithm involves a lot of engineering, especially when dealing with large linear programs and large numbers of cuts. We elaborate on several such engineering aspects and show that handling them properly is of crucial importance to the overall performance of the algorithm.

This paper is organized as follows. In the first part, we concentrate on the separation algorithms that have been developed for each class of facet inducing inequalities. In the second part, we discuss various other important components of a branch-and-cut algorithm; we investigate different branching strategies and cut generation schemes, and develop several primal heuristics. We have conducted various computational experiments and their results are reported throughout the second part.

2 A time-indexed formulation for single-machine scheduling problems

The usual setting for nonpreemptive single-machine scheduling problems is as follows. A set of n jobs has to be scheduled on a single machine. Each job j ($j = 1, \dots, n$) must be processed without interruption during a period p_j . The machine can handle no more than one job at a time and is continuously available from time zero onwards. We are asked to

find an optimal feasible schedule, that is, a set of start times such that the capacity and availability constraints are met and a given objective function is minimized.

A time-indexed formulation is based on time-discretization, i.e., time is divided into periods, where period t starts at time $t - 1$ and ends at time t ; the planning horizon is denoted by T , which means that we consider the time-periods $1, 2, \dots, T$. We consider the following time-indexed formulation for single-machine scheduling problems:

$$\min \sum_{j=1}^n \sum_{t=1}^{T-p_j+1} c_{jt} x_{jt}$$

subject to

$$\sum_{t=1}^{T-p_j+1} x_{jt} = 1 \quad (j = 1, \dots, n), \quad (1)$$

$$\sum_{j=1}^n \sum_{s=t-p_j+1}^t x_{js} \leq 1 \quad (t = 1, \dots, T), \quad (2)$$

$$x_{jt} \in \{0, 1\} \quad (j = 1, \dots, n; \ t = 1, \dots, T - p_j + 1),$$

where the binary variable x_{jt} for each job j ($j = 1, \dots, n$) and time period t ($t = 1, \dots, T - p_j + 1$) indicates whether job j starts in period t ($x_{jt} = 1$) or not ($x_{jt} = 0$). Constraints (1) state that each job has to be processed exactly once, and constraints (2) state that the machine handles at most one job during any time period.

In Van den Akker, Van Hoesel, and Savelsbergh [1994], we have given a complete characterization of all facet inducing inequalities with right-hand side 1 and 2 of the convex hull of the monotone extension of the set of feasible schedules. These facet inducing inequalities form the basis of the branch-and-cut algorithm discussed in this paper.

Before we can present these facet inducing inequalities with right-hand side 1 and 2, we have to introduce the following notation. The set of indices (j, t) of variables with nonzero coefficients in an inequality is denoted by V . The set of variables with nonzero coefficients in an inequality associated with job j defines a set of time periods $V_j = \{s | (j, s) \in V\}$. If job j is started in period $s \in V_j$, then we say that job j is started in V . With each set V_j we associate two values

$$l_j = \min\{s | s - p_j + 1 \in V_j\}$$

and

$$u_j = \max\{s | s \in V_j\}.$$

For convenience, let $l_j = \infty$ and $u_j = -\infty$ if $V_j = \emptyset$. Note that if $V_j \neq \emptyset$, then l_j is the first period in which job j can be finished if it is started in V , and that u_j is the last period in which job j can be started in V . Furthermore, let $l = \min\{l_j | j \in \{1, \dots, n\}\}$ and $u = \max\{u_j | j \in \{1, \dots, n\}\}$.

Below, we present four theorems defining the structure of all facet inducing inequalities with right-hand sides 1 and 2. For proofs of these results, we refer the reader to Van den Akker, Van Hoesel, and Savelsbergh [1994]. For presentational convenience, we use $x(S)$ to denote $\sum_{(j,s) \in S} x_{js}$ and denote valid inequalities with right-hand side 1 by $x(V) \leq 1$ and valid inequalities with right-hand side 2 by $x(V^1) + 2x(V^2) \leq 2$, where $V = V^1 \cup V^2$ and $V^1 \cap V^2 = \emptyset$.

Theorem 1 *A facet inducing inequality $x(V) \leq 1$ has the following structure:*

$$\begin{aligned} V_1 &= [l - p_1, u], \\ V_j &= [u - p_j, l] \quad (j \in \{2, \dots, n\}), \end{aligned} \tag{3}$$

where $l = l_1 \leq u_1 = u$. \square

This theorem says that a facet inducing inequality with right-hand side 1 can be represented by the following diagram:

$$\begin{array}{ccc} & l - p_1 & u \\ 1 & \boxed{} & \\ & u - p_j & l \\ j \in \{2, \dots, n\} & \boxed{} & \leq 1. \end{array}$$

Next, we study the structure of valid inequalities with right-hand side 2. Consider a valid inequality $x(V^1) + 2x(V^2) \leq 2$. Clearly, at most two jobs can be started in $V = V^1 \cup V^2$. Let $j \in \{1, \dots, n\}$ and $s \in V_j$. It is easy to see that, if job j is started in period s , at least one of the following three statements is true.

1. It is impossible to start any job in V before job j , and at most one job can be started in V after job j .
2. There exists a job i with $i \neq j$ such that job i can be started in V before as well as after job j and any job j' with $j' \neq j, i$ cannot be started in V .
3. At most one job can be started in V before job j , and it is impossible to start any job in V after job j .

Therefore, we can write $V = L \cup M \cup U$, where $L \subseteq V$ is the set of variables for which statement (1) holds, $M \subseteq V$ is the set of variables for which statement (2) holds, and $U \subseteq V$ is the set of variables for which statement (3) holds. Analogously, we can write $V_j = L_j \cup M_j \cup U_j$.

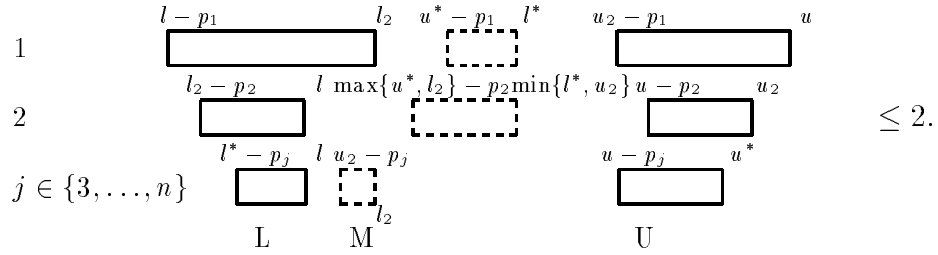
For both the sets L_j and the sets U_j , we can show that they have the same structure for all but two jobs. Therefore, to study the structure of valid inequalities with right-hand side 2, it suffices to consider three situations, based on the jobs with the deviant intervals L_j and U_j . The following three theorems describe the structure of the inequalities in each of the three situations.

Theorem 2 *A facet inducing inequality $x(V^1) + 2x(V^2) \leq 2$ with $l = l_1 < l_2 \leq l^*$, where $l^* = \min\{l_j \mid j \in \{3, \dots, n\}\}$, and $u = u_1 > u_2 \geq u^*$, where $u^* = \max\{u_j \mid j \in \{3, \dots, n\}\}$, has the following LMU-structure:*

$$\begin{aligned} L_1 &= [l - p_1, l_2], & M_1 &= [u^* - p_1, l^*] \setminus (L_1 \cup U_1), \\ L_2 &= [l_2 - p_2, l], & M_2 &= [\max\{u^*, l_2\} - p_2, \min\{l^*, u_2\}] \setminus (L_2 \cup U_2), \\ L_j &= [l^* - p_j, l], & M_j &= [u_2 - p_j, l_2] \setminus (L_j \cup U_j), \\ U_1 &= [u_2 - p_1, u], \\ U_2 &= [u - p_2, u_2], \\ U_j &= [u - p_j, u^*] \quad (j \in \{3, \dots, n\}), \end{aligned} \tag{4}$$

where $[u_2 - p_j, l] \subseteq L_j$ and $[u - p_j, l_2] \subseteq U_j$ for all $j \in \{3, \dots, n\}$. \square

This theorem says that a facet inducing inequality $x(V^1) + 2x(V^2) \leq 2$ with $l = l_1 < l_2 \leq l^*$ and $u = u_1 > u_2 \geq u^*$ can be represented by the following diagram:



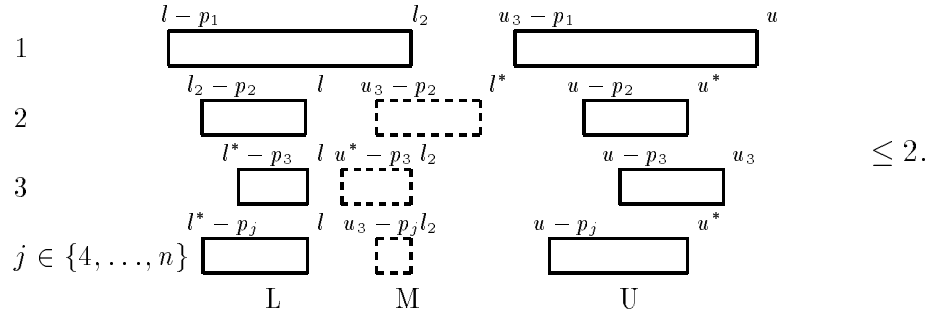
Theorem 3 *A facet inducing inequality $x(V^1) + 2x(V^2) \leq 2$ with $l = l_1 < l_2 \leq l^*$, where $l^* = \min\{l_j \mid j \in \{3, \dots, n\}\}$, $u = u_1 > u_3 \geq u^*$, where $u^* = \max\{u_j \mid j \in \{4, \dots, n\}\}$,*

and $l_j > l_2$ or $u_j < u_3$ for all $j \in \{2, \dots, n\}$ has the following LMU-structure:

$$\begin{aligned}
L_1 &= [l - p_1, l_2], & M_1 &= \emptyset, \\
L_2 &= [l_2 - p_2, l], & M_2 &= [u_3 - p_2, l^*] \setminus (L_2 \cup U_2), \\
L_3 &= [l^* - p_3, l], & M_3 &= [u^* - p_3, l_2] \setminus (L_3 \cup U_3), \\
L_j &= [l^* - p_j, l], & M_j &= [u_3 - p_j, l_2] \setminus (L_j \cup U_j), \\
U_1 &= [u_3 - p_1, u], \\
U_2 &= [u - p_2, u^*], \\
U_3 &= [u - p_3, u_3], \\
U_j &= [u - p_j, u^*] \quad (j \in \{4, \dots, n\}),
\end{aligned} \tag{5}$$

where $l^* \leq u^*$. \square

This theorem says that a facet inducing inequality $x(V^1) + 2x(V^2) \leq 2$ with $l = l_1 < l_2 \leq l^*$, $u = u_1 > u_3 \geq u^*$, and $l_j > l_2$ or $u_j < u_3$ for all $j \in \{2, \dots, n\}$ can be represented by the following diagram:

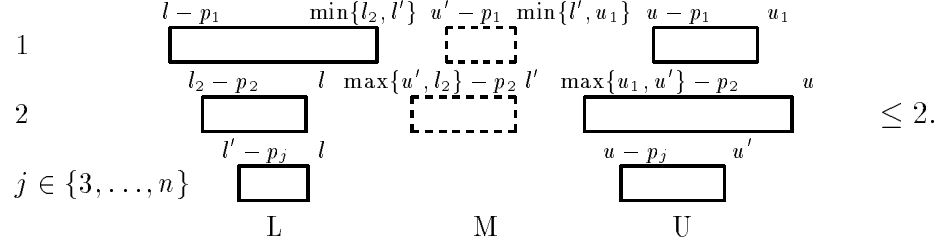


Theorem 4 A facet inducing inequality $x(V^1) + 2x(V^2) \leq 2$ with $l = l_1$ and $u = u_2$ has the following LMU-structure, where $l' = \min\{l_j \mid j \in \{3, \dots, n\}\}$ and $u' = \max\{u_j \mid j \in \{3, \dots, n\}\}$:

$$\begin{aligned}
L_1 &= [l - p_1, \min\{l_2, l'\}], & M_1 &= [u' - p_1, \min\{l', u_1\}] \setminus (L_1 \cup U_1), \\
L_2 &= [l_2 - p_2, l], & M_2 &= [\max\{u', l_2\} - p_2, l'] \setminus (L_2 \cup U_2), \\
L_j &= [l' - p_j, l], & M_j &= \emptyset, \\
U_1 &= [u - p_1, u_1], \\
U_2 &= [\max\{u_1, u'\} - p_2, u], \\
U_j &= [u - p_j, u'] \quad (j \in \{3, \dots, n\}),
\end{aligned} \tag{6}$$

where $[l' - p_2, l] \subseteq L_2$ and $[u - p_1, u'] \subseteq U_1$. \square

Note that l' and u' do not necessarily coincide with l^* and u^* because it is possible that $l_2 > l'$ and $u_1 < u'$. This theorem says that a facet inducing inequality $x(V^1) + 2x(V^2) \leq 2$ with $l = l_1$ and $u = u_2$ can be represented by the following diagram:



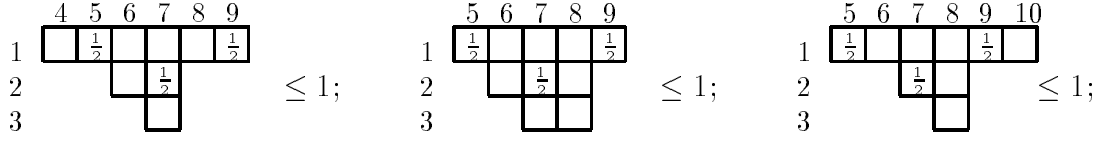
3 Separation

The separation algorithms for the four classes of facet inducing inequalities presented in the previous section are based on the enumeration of a small subset of all possible inequalities that may cut off the current fractional point.

To illustrate the underlying idea consider the class of facet inducing inequalities with right-hand side 1. Let \tilde{x} be the current LP solution and let F be a subset of variables with $\tilde{x}_{jt} > 0$ for all $(j, t) \in F$ and $\tilde{x}(F) > 1$. Any facet inducing inequality $x(V) \leq 1$ with $F \subseteq V$ cuts off \tilde{x} . Obviously, there may be many facet inducing inequalities that cut off \tilde{x} , but it suffices for the separation algorithm to identify only one of them. Our separation algorithm exploits this observation and restricts the search for a violated inequality to a subset of all facet inducing inequalities. The subset of facet inducing inequalities is chosen such that if there exists a violated inequality, then there exists a violated inequality in this subset.

Recall that each facet inducing inequality $x(V) \leq 1$ is completely determined by a job k , which w.l.o.g. is called job 1, and values l and u . Our separation algorithm restricts the search for a violated inequality to the subset of facet inducing inequalities covering F for which $u - l$ is minimal; minimal in the sense that there does not exist a facet inducing inequality $x(V') \leq 1$ with $F \subseteq V'$ and $u' - l' < u - l$, i.e., $u - l$ cannot be decreased without removing nonzero variables from the inequality.

Consider a three-job problem with $p_1 = 4$, $p_2 = 4$, and $p_3 = 3$. The LP solution $x_{15} = x_{19} = x_{27} = x_{2,11} = \frac{1}{2}$, $x_{31} = 1$ violates the three inequalities with structure (3) given by the diagrams below



Only the middle diagram represents a facet inducing inequality that covers F and is minimal with respect to $u - l$.

It can be shown that a facet inducing inequality $x(V) \leq 1$ that covers F is minimal with respect to $u - l$ if $\tilde{x}_{1, l-p_1+1} > 0$ and $\tilde{x}_{1u} > 0$. We refer to this condition as the *positive subset condition*. As a consequence of the positive subset condition, all potential violated minimal facet inducing inequalities $x(V) \leq 1$ can be enumerated in time polynomial in the number of fractional variables in the current LP solution, whereas the total number of facet inducing inequalities with right-hand side 1 is polynomial in the planning horizon T .

The development of separation algorithms for facet inducing inequalities with right-hand side 2 are also based on the identification of positive subset conditions.

This section is organized as follows. First, we consider facet inducing inequalities with right-hand side 1. Second, we consider facet inducing inequalities with right-hand side 2. For each type of inequality, we will study the structure of a minimal violated facet inducing inequality and identify a positive subset condition. We then proceed by presenting efficient separation algorithms that can be derived from the results obtained. In the sequel, \tilde{x} denotes the current LP-solution. As we start with the LP-relaxation of the original formulation, \tilde{x} satisfies (1) and (2).

3.1 A separation algorithm for facet inducing inequalities with right-hand side 1

To identify violated facet inducing inequalities with right-hand side 1, we have to identify violated inequalities with structure (3).

Lemma 1 *If \tilde{x} violates a facet inducing inequality $x(V) \leq 1$, then $\tilde{x}_{js} < 1$ for all $(j, s) \in V$.*

Proof. Let $x(V) \leq 1$ be facet inducing. Let $\tilde{x}_{js} = 1$ for some $(j, s) \in V$. Since \tilde{x} satisfies (1), $\tilde{x}_{js'} = 0$ for any $s' \neq s$. Because of the validity of $x(V) \leq 1$, any job j' that is started in V overlaps job j during some time-period. Hence, if $\tilde{x}_{j's'} > 0$ for some $(j', s') \in V$ with $j' \neq j$, then the workload of the machine is greater than 1 during some time period. Since inequalities (2) state that during any time period the workload of the machine is at most 1, it follows that $\tilde{x}_{j's'} = 0$ for all $(j', s') \in V$ with $j' \neq j$. We conclude that then

$\tilde{x}(V) = 1$, i.e., \tilde{x} does not violate $x(V) \leq 1$. \square

The following lemma shows that the separation can be restricted to the identification of inequalities satisfying the positive subset condition which states that $\tilde{x}_{1,l-p_1+1} > 0$ and $\tilde{x}_{1u} > 0$. By this condition $u - l$ is minimal in the sense that it cannot be decreased without removing nonzero variables from the inequality.

Lemma 2 *If \tilde{x} violates a facet inducing inequality $x(V) \leq 1$, then we may assume that $\tilde{x}_{1,l-p_1+1} > 0$ and $\tilde{x}_{1u} > 0$.*

Proof. Let \tilde{x} violate a facet inducing inequality $x(V) \leq 1$. Since \tilde{x} satisfies (2), we must have $l < u$. Suppose $\tilde{x}_{1,l-p_1+1} = 0$. If we increase l by 1, then the variable $x_{1,l-p_1+1}$ is removed from $x(V) \leq 1$ and the variables $x_{j,l+1}$ with $j \neq 1$ such that $u - p_j < l + 1$ are added to this inequality. In this way we obtain another facet inducing inequality. Since in the original inequality $\tilde{x}_{1,l-p_1+1} = 0$, \tilde{x} also violates the new inequality. We conclude that if $\tilde{x}_{1,l-p_1+1} = 0$, then we obtain another violated inequality by decreasing $u - l$. We may hence assume $\tilde{x}_{1,l-p_1+1} > 0$. In the same way we can show if $\tilde{x}_{1u} = 0$, then we obtain another violated inequality by decreasing u , i.e., by decreasing $u - l$. We may hence also assume that $\tilde{x}_{1u} > 0$. \square

Since the current LP-solution \tilde{x} satisfies the equations (1), for a violated inequality $x(V) \leq 1$ we must have $V_j \neq \emptyset$ for some $j \in \{2, \dots, n\}$ and hence $u - \max\{p_j \mid j \in \{2, \dots, n\}\} < l$. A facet inducing inequality $x(V) \leq 1$ is determined by a job j and time periods l and u . It is easy to see that the number of such inequalities is of order nTp_{\max} , where p_{\max} denotes the maximal processing time. However, the number of inequalities satisfying the positive subset condition is bounded by the square of the number of fractional variables in the current LP-solution and hence the number of inequalities that have to be checked by the separation algorithm is bounded by this number. The resulting separation algorithm is as follows.

SepRHS1($\tilde{\mathbf{x}}$)

```

begin
  for all jobs  $j \in \{1, \dots, n\}$  do
    for all  $l$  such that  $0 < \tilde{x}_{j,l-p_j+1} < 1$  do
      for all  $u$  such that  $l < u < l + \max\{p_i \mid i \neq j\}$  and  $0 < \tilde{x}_{ju} < 1$  do
        if  $\sum_{s \in [l-p_j, u]} \tilde{x}_{js} + \sum_{i \neq j} \sum_{s \in [u-p_i, l]} \tilde{x}_{is} > 1$ 
          then violated inequality identified;
end.
```

3.2 A separation algorithm for facet inducing inequalities with right-hand side 2

Facet inducing inequalities with right-hand side 2 are inequalities with structure (4), (5), or (6). Because of the complexity of the necessary conditions for an inequality with one of these structures to be facet inducing (see Van den Akker [1994]), the separation algorithm is not restricted to facet inducing inequalities but considers all nondecomposable inequalities with one of these structures, where an inequality $x(V^1) + 2x(V^2) \leq 2$ is called nondecomposable if it cannot be written as the sum of two valid inequalities with right-hand side 1. As we have done in the previous subsection, we will study the characteristics of violated inequalities satisfying the positive subset condition. The study proceeds in three phases and will be presented in the form of several lemmas. For reasons of brevity, we have omitted the proofs of these lemmas. The interested reader is referred to Van den Akker [1994].

Lemma 3 *If \tilde{x} violates the inequality $x(V^1) + 2x(V^2) \leq 2$ and satisfies all valid inequalities $x(W) \leq 1$ with $W \subseteq V$, then $\tilde{x}_{js} < 1$ for all $(j, s) \in V$.*

Recall that the positive subset condition implies that some function of the parameters is minimal in the sense that it cannot be decreased without removing nonzero variables from the inequality. As the positive subset condition is derived in three phases, the minimization of this function also proceeds in three phases. The following lemma shows that the positive subset conditions that are derived in the first phase imply that $u - l$ is minimal. In the first phase, we do not consider the classes of inequalities separately. If \tilde{x} violates some inequality $x(V^1) + 2x(V^2) \leq 2$ for which $u - l$ is not minimal, then the violated inequality that can be obtained from it by minimizing $u - l$ may be of a different type. For example, if \tilde{x} violates an inequality with structure (4) for which $u - l$ is not minimal, then by minimizing $u - l$ we may obtain an inequality with structure (6).

Lemma 4 *If \tilde{x} violates the inequality $x(V^1) + 2x(V^2) \leq 2$ with structure (4), (5), or (6) then we may assume that one of the following holds:*

- (a) $x(V^1) + 2x(V^2) \leq 2$ has structure (4) or (5), $\tilde{x}_{1,l-p_1+1} > 0$, and $\tilde{x}_{1u} > 0$;
- (b) $x(V^1) + 2x(V^2) \leq 2$ has structure (6), $\tilde{x}_{1,l-p_1+1} > 0$, and $\tilde{x}_{2u} > 0$.

In the second phase, we distinguish between inequalities with structure (4) or (5) and inequalities with structure (6). This means that if there exists a violated inequality with minimal $u - l$ that has structure (4) or (5), then the separation algorithm will identify a violated inequality with one of these structures and the given l and u . The same is true for inequalities with structure (6).

For $l_2 = l^*$ or $u_2 = u^*$ an inequality with structure (4) also has structure (5). The following lemma shows that it is beneficial for the separation algorithm to check some of the inequalities with structure (4) while it is considering inequalities with structure (5). Therefore, in the separation algorithm these inequalities are also seen as inequalities with structure (5), i.e., we allow structure (5) with $l_j = l_2$ and $u_j = u_3$ for some $j \in \{2, \dots, n\}$. The following lemma shows that for a violated inequality with structure (4) or (5) the positive subset condition implies that $u_{(2)} - l_2$ is minimal, where $u_{(2)}$ equals u_2 , if the inequality has structure (4) and equals u_3 if the inequality has structure (5).

Lemma 5 *If \tilde{x} violates an inequality $x(V^1) + 2x(V^2) \leq 2$ with structure (4) or (5), then we may assume that one of the following two holds:*

- (a) $x(V^1) + 2x(V^2) \leq 2$ has structure (4), $\tilde{x}_{2, l_2 - p_2 + 1} > 0$, and $\tilde{x}_{2u_2} > 0$;
- (b) $x(V^1) + 2x(V^2) \leq 2$ has structure (5) possibly with $l_j = l_2$ and $u_j = u_3$ for some $j \in \{2, \dots, n\}$, $\tilde{x}_{2, l_2 - p_2 + 1} > 0$, and $\tilde{x}_{3u_3} > 0$.

Consider an inequality with structure (6) and suppose that $l_2 > l'$. It is easy to see that if we set $l_2 = l'$, then the only part of the inequality that may be changed is the interval M_2 . Therefore, the inequality remains valid. Therefore, we assume $l_2 \leq l'$, although by this assumption we may drop the condition that $l_2 = \min\{s | s - p_2 + 1 \in V_2\}$. Analogously, we assume $u_1 \geq u'$.

The following lemma shows that for a violated inequality with structure (6) the positive subset condition implies that $(l_2 - l)^+ + (u - u_1)^+$ is minimal, where $x^+ = \max\{x, 0\}$. The expression $(l_2 - l)^+$ stems from the fact that since, by definition, $l_2 \geq l$, l_2 cannot be decreased if $l_2 = l$. Analogously, we find the expression $(u - u_1)^+$.

Lemma 6 *If \tilde{x} violates $x(V^1) + 2x(V^2) \leq 2$ with structure (6), then we may assume that*

- (a) if $l_2 > l$, then $\tilde{x}_{1l_2} > 0$;
- (b) if $u_1 < u$, then $\tilde{x}_{2, u_1 - p_1 + 1} > 0$. \square

In the third phase, each of the classes of inequalities is considered separately. The following lemma shows that for inequalities with structure (4) separation can be restricted to violated inequalities for which $(l^* - l_2)^+ + (u_2 - u^*)^+$ is minimal, where the expressions $(l^* - l_2)^+$ and $(u_2 - u^*)^+$ stem from the conditions on the parameters stating that $l_2 \leq l^*$ and $u_2 \geq u^*$.

Lemma 7 *If \tilde{x} violates $x(V^1) + 2x(V^2) \leq 2$ with structure (4), then we may assume that*

- (a) if $l^* > l_2$, then either $\tilde{x}_{1l^*} > 0$, $M_1 \neq \emptyset$, and l^* is the maximum of M_1 , or $\tilde{x}_{2l^*} > 0$, $M_2 \neq \emptyset$, and l^* is the maximum of M_2 ;

(b) if $u^* < u_2$, then either $\tilde{x}_{1u^*-p_1+1} > 0$, $M_1 \neq \emptyset$, and $u^* - p_1 + 1$ is the minimum of M_1 , or $\tilde{x}_{2u^*-p_2+1} > 0$, $M_2 \neq \emptyset$, and $u^* - p_2 + 1$ is the minimum of M_2 . \square

Note that for an inequality $x(V^1) + 2x(V^2) \leq 2$ with structure (4) with $l_2 < l^*$ we have that $M_1 \neq \emptyset$ and l^* is the maximum of M_1 if and only if $u^* - p_1 < l^* \leq u_2 - p_1$. The other conditions in the above lemma can be rewritten in a similar way. For inequalities with structures (5) and (6), we can derive similar lemmas which show that separation can be restricted to violated inequalities for which respectively $(l^* - l_2)^+ + (u_3 - u^*)^+$ and $(l' - l_2)^+ + (u_1 - u')^+$ are minimal.

Based on the previous lemmas, we can derive a separation algorithm for inequalities $x(V^1) + 2x(V^2) \leq 2$ with structure (4), (5), or (6). As for facet inducing inequalities with right-hand side 1, the algorithm is based on enumeration of fractional variables in the current solution. We will only present the algorithm that identifies violated inequalities with structure (4). The identification of violated inequalities with other structures proceeds in a similar way. Furthermore, we will not describe the algorithm in complete detail, but only provide an overview.

In the description of the separation algorithm $I_{2a}(j_1, j_2, l, l_2, l^*, u^*, u_2, u)$ denotes the value of the left-hand side of the inequality with structure (4) for \tilde{x} with job j_1 as job 1, job j_2 as job 2, and the numbers l, l_2, l^*, u^*, u_2 and u as indicated. The number of such inequalities is $O(n^2 T^4 p_{\max}^2)$. If the current solution satisfies all valid inequalities with right-hand side 1, then, by Lemma 3, the number of inequalities that are checked by the separation algorithm is bounded by the sixth power of the number of fractional variables in the current solution. The separation algorithm is as follows.

SepRHS2a(\tilde{x})

begin

```

for  $j_1$  and  $l$  such that  $\tilde{x}_{j_1, l-p_{j_1}+1} > 0$  and such that there exists a  $u$  with  $\tilde{x}_{j_1, u} > 0$  and
 $u > l + p_{\min}$  do
  for  $j_2$  and  $l_2$  such that  $\tilde{x}_{j_2, l_{j_2}-p_{j_2}+1} > 0$ ,  $j_2 \neq j_1$ ,  $l_2 > l$ , and
   $l_2 < l + \max\{p_j \mid j \neq j_1\}$  do
    for  $u_2$  such that  $\tilde{x}_{j_2, u_2} > 0$ ,  $u_2 > l_2$ ,  $U_{j_2} \neq \emptyset$  if  $L_{j_2} = \emptyset$ , and such that there exists a  $u$ 
    with  $\tilde{x}_{j_1, u} > 0$ ,  $u - \max\{p_j \mid j \neq j_1\} < u_2 < u$ , and  $u > l + p_{\min}$  do
      for  $u$  such that  $\tilde{x}_{j_1, u} > 0$ ,  $u - \max\{p_j \mid j \neq j_1\} < u_2 < u$ , and
       $u > l + p_{\min}$  do
        if  $l_2 - p_{j_2} \geq l \{L_{j_2} = \emptyset\}$ 
          then
            SepRHS2aL2empty( $\tilde{x}, j_1, j_2, l, l_2, u_2, u$ );
          else
            if  $u_2 \leq u - p_{j_2} \{U_{j_2} = \emptyset\}$ 
              then
                SepRHS2aU2empty( $\tilde{x}, j_1, j_2, l, l_2, u_2, u$ );

```

```

    else  $\{L_{j_2} \neq \emptyset \text{ and } U_{j_2} \neq \emptyset\}$ 
      SepRHS2anonempty( $\tilde{x}, j_1, j_2, l, l_2, u_2, u$ );
    end.

SepRHS2aL2empty( $\tilde{x}, j_1, j_2, l, l_2, u_2, u$ )
begin
   $u^* = l_2$ ;
  for  $l^* = l_2$  and  $l^* > l_2$  such that condition (a) of Lemma 7 is satisfied do
    if  $I_{2a}(j_1, j_2, l, l_2, l^*, u^*, u_2, u) > 2$ 
      then violated inequality identified;
    end.

SepRHS2aU2empty( $\tilde{x}, j_1, j_2, l, l_2, u_2, u$ )
begin
   $l^* = u_2$ ;
  for  $u^* = u_2$  and  $u^* < u_2$  such that condition (b) of Lemma 7
    is satisfied do
    if  $I_{2a}(j_1, j_2, l, l_2, l^*, u^*, u_2, u) > 2$ 
      then violated inequality identified;
    end.

SepRHS2anonempty( $\tilde{x}, j_1, j_2, l, l_2, u_2, u$ )
begin
   $l^* = l_2$ ;  $\{M_{j_1} = \emptyset\}$ 
  for  $u^* = u_2$  and  $u^* < u_2$  such that  $\tilde{x}_{j_2, u^* - p - j_2 + 1} > 0$ ,  $M_{j_2} \neq \emptyset$ , and
     $u^* - p_{j_2} + 1$  is the minimum of  $M_{j_2}$  do
    if  $I_{2a}(j_1, j_2, l, l_2, l^*, u^*, u_2, u) > 2$ 
      then violated inequality identified;
     $u^* = u_2$ ;  $\{M_{j_1} = \emptyset\}$ 
    for  $l^* > l_2$  such that  $\tilde{x}_{j_2, l^*} > 0$ ,  $M_{j_2} \neq \emptyset$ , and  $l^*$  is the maximum of  $M_{j_2}$  do
      if  $I_{2a}(j_1, j_2, l, l_2, l^*, u^*, u_2, u) > 2$ 
        then violated inequality identified;
      for  $l^* > l_2$  and  $u^* < u_2$  such that the conditions from
        Lemma 7 are satisfied do
        if  $I_{2a}(j_1, j_2, l, l_2, l^*, u^*, u_2, u) > 2$ 
          then violated inequality identified;
        end.
    end.

```

4 A branch-and-cut algorithm for $1|r_j|\sum w_j C_j$

Based on the separation algorithms derived in the previous section, we have developed a branch-and-cut algorithm for the problem of minimizing the sum of the weighted completion times on a single machine subject to release dates, i.e., $1|r_j|\sum w_j C_j$. In

this section, we describe the main components of the branch-and-cut algorithm and we report on its performance. We investigate the quality of the lower bounds obtained by adding violated inequalities, we discuss possible branching strategies and cut generation schemes, and we develop several primal heuristics. Finally, we present the computational results of the two variants that perform best.

The branch-and-cut algorithms have been implemented using MINTO, a Mixed Integer Optimizer (Nemhauser, Savelsbergh, and Sigismondi [1994]). MINTO is a software system that solves mixed-integer linear programs by a branch-and-bound algorithm with linear relaxations. It also provides automatic constraint classification, preprocessing, primal heuristics, and constraint generation. Moreover, the user can enrich the basic algorithm by providing a variety of specialized application functions that can customize MINTO to achieve maximum efficiency for a problem class. Our computational experiments have been conducted with MINTO 2.0/CPLEX 3.0 and have been run on a IBM RS/6000 model 590. In all our experiments MINTO's preprocessing, primal heuristics, and cut generation have been deactivated.

For our computational experiments, we have used sets of 20 randomly generated instances with uniformly distributed parameters; the weights are in $[1, 10]$, the release dates are in $[0, \frac{1}{2} \sum_{j=1}^n p_j]$, and the processing times are in $[1, p_{\max}]$. We consider sets of 20-job instances with p_{\max} equal to 5, 10, and 20, respectively, and sets of 30-job instances with p_{\max} equal to 5 and 10, respectively. Recall that the number of constraints is $n + T$ and the number of variables is approximately nT . Since $T \geq \sum_{j=1}^n p_j$, the size of the linear program increases when the number of jobs increases as well as when the processing times increase. For the 30-job problems we did not consider $p_{\max} = 20$, since the memory requirements were too large.

4.1 Quality of the lower bounds

The goal of our first experiments was to evaluate the quality of the lower bounds obtained by just solving the LP-relaxation, by solving the LP relaxation in combination with facet inducing inequalities with right-hand side 1, and by solving the LP-relaxation in combination with facet inducing inequalities with right-hand side 1 and 2. The results for one hundred instances, twenty in each of the sets, are summarized in Table 1. Let Z_{LB} denote a lower bound on the optimal value Z_{IP} of the integer program. The gap G_{LB} corresponding to this lower bound is defined by

$$G_{LB} = \frac{Z_{IP} - Z_{LB}}{Z_{IP}} \times 100\%.$$

Note that this gap is expressed as a percentage. In Table 1, we report for each set of twenty instances corresponding to the same combination (n, p_{\max}) the following numbers:

- G_{LP}^{av} and G_{LP}^{max} : the average gap after solving the LP-relaxation and the maximum of these gaps;
- G_1^{av} and G_1^{max} : the average gap after the addition of cuts with right-hand side 1 and the maximum of these gaps;
- G_2^{av} and G_2^{max} : the average gap after the addition of cuts with right-hand side 1 and 2 and the maximum of these gaps.

	LP		1		2	
(n, p_{max})	G_{LP}^{av}	G_{LP}^{max}	G_1^{av}	G_1^{max}	G_2^{av}	G_2^{max}
(20, 5)	0.379	1.346	0.157	1.228	0.058	0.572
(20,10)	0.64	1.959	0.233	1.337	0.054	0.407
(20,20)	0.507	1.657	0.126	0.966	0.047	0.385
(30, 5)	0.390	1.309	0.179	0.664	0.121	0.599
(30,10)	0.478	1.099	0.121	0.934	0.096	0.592

Table 1. Quality of the bounds.

The results in Table 1 do not reflect the fact that many instances were solved to optimality just by adding cuts. Table 2 provides statistics on the frequency with which optimal solutions were found. More precisely, we report:

- n_{LP} : the number of instances for which the optimal solution of the LP-relaxation was integral;
- n_1 : the total number of instances that were solved to optimality after the addition of cuts with right-hand side 1;
- n_2 : the total number of instances that were solved to optimality after the addition of cuts with right-hand side 1 and 2.

These results show that the bounds obtained are excellent, even the initial linear relaxation is always within two percent of the optimum, and that both classes of inequalities are effective in reducing the integrality gap. Table 1 indicates that for most of the instances the addition of cuts with right-hand side 1 closes at least half of the integrality gap and that addition of cuts with right-hand side 2 reduces this gap even further. From Table 2 we conclude that the addition of cuts with right-hand side 2 significantly increases the number of instances that are solved without branching.

(n, p_{\max})	n_{LP}	n_1	n_2
(20, 5)	5	12	18
(20,10)	0	6	16
(20,20)	4	13	17
(30, 5)	5	6	8
(30,10)	0	5	9

Table 2. Number of instances that were solved to optimality.

4.2 Branching strategies

When the addition of cutting plane fails to solve the problem, we resort to branch-and-bound. In this section, we discuss three branching strategies that can be used in the branch-and-cut algorithm and we evaluate their performance. The first two strategies are general branching strategies, in the sense that they can be used for any 0-1 integer program, whereas the third strategy exploits the structure of feasible schedules.

General branching strategies for 0-1 integer programs are based on fixing variables, either a single variable (variable dichotomy) or a set of variables (GUB dichotomy). We have explored both strategies. In the first, we branch on the fractional variable x_{jt} closest to 0.5. We set $x_{jt} = 1$ on one branch, i.e., we force job j to start in time period t , and $x_{jt} = 0$ on the other branch, i.e., we prevent job j from being started in time period t . In case of ties, we select the variable with the smallest t . In the second, we branch on the assignment constraint $\sum_{1 \leq t \leq T-p_j+1} x_{jt} = 1$ for the job j that covers the largest time interval, i.e., the job j for which the difference between the first and last period with positive x_{jt} is maximum. We set $\sum_{1 \leq t \leq \lfloor t^* \rfloor} x_{jt} = 0$ on one branch, i.e, we force job j to start not later than $\lfloor t^* \rfloor$, and $\sum_{\lfloor t^* \rfloor < t \leq T-p_j+1} x_{jt} = 0$ on the other branch, i.e., we force job j to start before $\lfloor t^* \rfloor + 1$, where we have chosen t^* to be equal to $\sum_{1 \leq t \leq T-p_j+1} (t-1)x_{jt}$, the mean start time suggested by the current LP solution. The second branching scheme has the advantage that it divides the search space more evenly, which is a desirable characteristic of a branching strategy.

The above branching strategies specify how the current set of feasible solutions is to be divided into smaller subsets. They do not specify how the subproblem to be solved next is to be selected. We have considered two selection strategies: depth-first search and best-bound search. Depth-first search is usually applied to get (hopefully good) feasible solutions fast; experience shows that feasible solutions are more likely to be found deep in the tree than at nodes near the root. Having a good feasible solution is necessary to be able to prune nodes and thus to reduce the size of the branch-and-bound tree. Best-

bound search is motivated by the observation that to prove optimality the node with the best bound has to be evaluated, so it may as well be explored first. Computational experiments have revealed that the branching strategies based on variable and GUB dichotomy work best with best-bound search of the tree.

The positional branching strategy, which is the traditional branching strategy used in combinatorial algorithms, exploits the structure of feasible schedules and fixes jobs at a certain position in the schedule. At level d in the branch-and-bound tree the jobs in positions $1, \dots, d-1$ have already been fixed and some job j is fixed at position d . Fixing a job j in position d is accomplished by fixing its start time at the maximum of its release date and the completion time of the $(d-1)$ th job. Note that this can be done because the objective function is nondecreasing in the completion times of the jobs. As a dominance rule, we do not allow a job to be fixed in position d if its release date is so large that it is possible to complete some other job that has not yet been fixed before this release date. The subproblems at level d are defined by fixing the jobs that have not yet been fixed at position d . The order in which these subproblems are selected is determined on the basis of the mean start times suggested by the current LP solution, i.e., the jobs are put in nondecreasing order of $\sum_{1 \leq t \leq T-p_j+1} tx_{jt}$. This strategy works best in combination with depth-first search of the tree.

In Tables 3a and 3b, we compare the performance of the different branching strategies for the two sets of 30-job instances with $p_{\max} = 5$ and $p_{\max} = 10$. Since the majority of the 20-job instances were solved to optimality in the root node, we do not report results for these instances. In the first three rows of the tables, we report on the number of nodes in the branch-and-bound tree: the average number (n^{av}), the maximum number (n^{max}), and the standard deviation (σ_n). In the last three rows of the table, we report on the computation time. Several observations can be made based on these results.

(30,5)	positional branching	GUB dichotomy	variable dichotomy
n^{av}	52.30	6.60	489.90
n^{max}	255	29	7545
σ_n	66.02	7.63	1641.01
t^{av}	7.98	6.08	213.12
t^{max}	20.31	23.99	3307.94
σ_t	5.50	4.55	716.77

Table 3a. Performance of the different branching strategies for $n = 30$ and $p_{\max} = 5$.

(30,10)	positional branching	GUB dichotomy	variable dichotomy
n^{av}	26.57	19.35	169.05
n^{max}	286	247	2661
σ_n	56.62	52.86	578.68
t^{av}	23.27	53.15	477.38
t^{max}	384.63	691.51	7269.59
σ_t	60.10	146.96	1585.95

Table 3b. Performance of the different branching strategies for $n = 30$ and $p_{\max} = 10$.

First, the branching strategy based on variable dichotomy is clearly inferior to the other two. Second, GUB branching requires fewer nodes than positional branching. However, evaluating fewer nodes does not translate into faster solution times. There are two factors that, in our opinion, contribute to this phenomenon. Positional branching fixes many more variables which reduces the size of the linear programs that have to be solved. In addition, in a depth-first search strategy consecutive linear programs differ only slightly. Consequently the basis of last solved linear program is a good starting basis for the current linear program. In a best-bound search strategy consecutive linear programs are likely to differ considerably. Consequently, the basis of the last solved linear program does not provide a good starting basis for the current linear program. Furthermore, since many cuts are generated during the solution process, the basis associated with the basis associated with the last linear program solved in the parent node does not provide a good starting basis either. A final observation is that there is a high variance in complexity among the instances. With GUB branching all but one instance are solved in fewer than 20 nodes and less than 60 seconds; the one difficult instance solves in a little less than 250 nodes and 700 seconds. To verify whether this is typical behavior, we tested GUB branching and positional branching on an extended set of 40 instance with $p_{\max} = 10$. The results for this extended set of instances can be found in Table 4 and show a similar pattern.

An advantage of the branching strategy based on GUB dichotomy is that it can be applied for all objective functions $\sum_{j=1}^n \sum_{t=1}^{T-p_j+1} c_{jt} x_{jt}$, whereas the positional branching strategy is based on the assumption that it is most favorable to start a job as early as possible and can only be applied if the objective function is nondecreasing in the completion times of the jobs.

On the other hand, with the positional branching strategy the total number of nodes

(30,10)	positional branching	GUB dichotomy
n^{av}	133.38	29.05
n^{max}	2108	573
σ_n	370.06	95.92
t^{av}	83.49	59.98
t^{max}	638.79	691.51
σ_t	158.91	142.37

Table 4. Performance of the different branching strategies for $n = 30$ and $p_{\max} = 10$.

in the branch-and-bound tree only depends on the number of jobs, whereas for the branching strategy based on GUB dichotomy the number of nodes depends on the number of jobs as well as on the planning horizon; hence on the size of the processing times. This suggests that positional branching may perform better for instances with large processing times.

4.3 Cut generation schemes

In this subsection, we study the influence of different cut generation schemes on the performance of the branch-and-cut algorithm. Cut generation schemes try to find the proper balance between the expected increase in performance due to stronger bounds that result from the addition of cuts and the expected decrease in performance due to the effort required to identify violated cuts and to solve larger and more difficult linear programs. Cut generation schemes specify, among other things, when we try to identify violated inequalities, which of the identified violated inequalities are added, and when inactive inequalities are deleted.

The experiments of the previous section showed that 30-job instances with $p_{\max} = 5$ are relatively easy, in the sense that their solution requires very few nodes, and that a large sample of 30-job instances with $p_{\max} = 10$ is necessary to be able to draw reliable conclusions. Therefore, the remaining experiments have been conducted with a set of 40 randomly generated 30-job instances with $p_{\max} = 10$.

We have investigated various possible cut generation schemes that specify choices related to which classes of cuts to use and when to use them.

R12T12: At all nodes, add cuts with right-hand side 1 and 2.

R12T1: At the root node, add cuts with right-hand side 1 and 2; in all other nodes, add

cuts with right-hand side 1.

R12: At the root node, add cuts with right-hand side 1 and 2; in all other nodes, do not add cuts.

R1T1: At all nodes, add cuts with right-hand side 1.

The performance of these variants is shown in Table 5a and 5b. We report the performance of these variants with positional branching as well as with GUB branching. Again, n^{av} and n^{max} denote the average and maximum number of nodes, and t^{av} and t^{max} denote the average and maximum computation time.

	<i>R12T12</i>	<i>R12T1</i>	<i>R12</i>	<i>R1T1</i>
n^{av}	133.38	220.90	377.07	422.07
n^{max}	2108	3677	7504	4764
t^{av}	83.49	71.25	75.62	64.01
t^{max}	638.79	528.46	816.73	595.59

Table 5a. Cut generation schemes with positional branching.

	<i>R12T12</i>	<i>R12T1</i>	<i>R12</i>	<i>R1T1</i>
n^{av}	29.05	67.42	107.70	89.97
n^{max}	573	1981	3443	1595
t^{av}	59.98	69.54	66.09	59.81
t^{max}	691.51	889.87	991.02	846.20

Table 5b. Cut generation schemes with GUB branching.

Several observations can be made based on these results. First, the cut generation scheme *R12T12*, i.e., generating cuts with right-hand side 1 and 2, clearly results in the fewest number of evaluated nodes. However, evaluating fewer nodes does not translate into faster solution times. For positional branching, cut generation scheme *R1T1*, i.e., generating only cuts with right-hand side 1, is much faster than *R12T12* even though it generates considerably more nodes, and for GUB branching cut generation scheme *R1T1* is about as fast as *R12T12* although it generates more nodes. This is probably due to the fact that the linear programs that result if cuts with right-hand side 2 are

added are more difficult because they are denser than cuts with right-hand side 1. So far the two best variants of the algorithm are positional branching with cut generation scheme *R1T1* and GUB branching with cut generation scheme *R12T12*. We prefer cut generation scheme *R12T12* over *R1T1* for GUB branching because it seems to be more robust in the sense that the maximum number of evaluated nodes and the maximum computation time over all instances are the smallest. For the remainder, we will restrict our computational experiments to these two versions.

The cut generation schemes discussed above specify choices related to which classes of cuts to use and when to use them. We have also considered cut generation schemes that try to improve the performance by limiting the number of violated inequalities that will be added to the active linear program. In fact, such a cut generation scheme has been active during all previous experiments. When MINTO processes a node, it monitors the changes in the value of the LP solutions from iteration to iteration. If it detects that the total change in the value of the LP solution in the last three iterations is less than 0.5 percent, i.e., 0.005 times the value of the current LP solution, it forces MINTO to branch. This feature is incorporated in MINTO to handle the ‘tailing-off’ effect exhibited by many cutting plane algorithms. Tailing-off refers to a situation in which for several consecutive iterations violated inequalities are identified and added to the current linear program, but the objective function value only marginally improves. In such situations, it may be more advantageous to branch than to continue generating cuts.

The experiments carried out to evaluate the quality of the bounds, discussed in Section 4.1, revealed that it is impossible to predict the change in objective function value after the addition of violated inequalities. It frequently happened that the objective function hardly changed for several iterations before improving significantly. Consequently, it is very likely that MINTO, with default settings, would sometimes force branching too soon. To ensure the best possible bound at the root node, we have chosen to deactivate forced branching in the root node.

To evaluate the effect of different forcing strategies on the performance of the algorithms, we have investigated the following three strategies: no forced branching, no forced branching at the root node but forced branching at all other nodes, and forced branching throughout the tree. The results are shown in Tables 6a and 6b. The following observation can be made based on these results. First, the strategy that we adopted works well. Second, the tailing-off effect is much stronger when cuts with right-hand side 2 are used.

There are various other ways to limit the number of violated inequalities that will be added to the active linear program: limit the number of cuts that is added in a single round of cut generation, limit the number of rounds of cut generations per node evaluation, and limit the number of nodes at which cut generation takes place (this is sometimes referred to as the cut frequency). All these did not seem to have a significant

	no forcing	root forcing	forcing
n^{av}	27.92	29.05	56.12
n^{max}	419	573	1419
t^{av}	130.33	59.98	56.45
t^{max}	2396.48	691.51	804.85

Table 6a. Forcing strategies with GUB branching.

	no forcing	root forcing	forcing
n^{av}	432.70	422.07	409.80
n^{max}	5036	4746	4746
t^{av}	65.55	64.01	62.32
t^{max}	602.62	595.59	608.17

Table 6b. Forcing strategies with positional branching.

positive effect on the performance of the basic algorithm. In most cases, the performance of these variants was actually worse.

Finally, we have experimented with cut generation schemes in which inequalities are deleted when they have been inactive for a number of consecutive iterations, i.e., the dual variable associated with the inequality has been 0 for a number of consecutive iterations. Again, such a cut generation scheme has been in effect during all previous experiments. To control the size of the active linear program MINTO monitors the dual variables associated with all the inequalities that have been generated during the solution process and, if the value of a dual variable has been equal to zero for fifty consecutive iterations, MINTO will deactivate the corresponding inequality. Deactivating an inequality means deleting it from the active linear program and storing it in a *cut pool*. Every time the active linear program is solved and a new linear programming solution exists, the inequalities in the cut pool will be inspected to see if any of them are violated by the current solution. If so, these inequalities will be reactivated. Reactivating a inequality means adding it to the active formulation and deleting it from the cut pool. The cut pool has a fixed size and is maintained on a first-in-first-out basis, i.e., if the pool overflows the inequalities that have been in the pool the longest will be deleted. As soon as a cut is deleted from the cut pool it can never be reactivated again. Table 7 shows the effect of different thresholds for deletion (10, 50, 1000) for GUB branching.

A threshold for deletion specifies the number of consecutive iterations in which the dual variable associated with a constraint has to be equal to 0 before it is deactivated. Note that setting the threshold to 1000 for this application is equivalent to no cut deletion. We

	10	50	1000
n^{av}	28.82	29.05	30.42
n^{max}	573	573	595
t^{av}	75.98	59.98	80.06
t^{max}	901.98	691.51	1320.93

Table 7. Effect of different thresholds for cut deletion.

conclude that cut deletion greatly influences the computation time and that MINTO’s default setting is appropriate for our application. We have not performed the same experiment for positional branching, but it is very likely that our conclusions are also applicable to positional branching.

4.4 Primal heuristics

In this subsection, we describe the primal heuristics that have been incorporated in the branch-and-cut algorithm. The availability of good feasible solutions is important for various reasons. In case of depth-first search (which we do in case of positional branching) it may significantly reduce the number of nodes that has to be evaluated, since any node with a lower bound greater than or equal to the value of the best known solution can be skipped from further consideration. In case of best-bound search (which we do in case of GUB branching) it will not reduce the number of evaluated nodes by much, but it will reduce the set of unevaluated nodes that has to be kept, which is important for large integer programs because it reduces the chance of running out of memory. Furthermore, good feasible solutions are essential for effective reduced cost fixing.

We have implemented four primal heuristics. The first heuristic is derived from Smith’s rule (Smith [1956]). Smith’s rule solves $1||\sum w_j C_j$, i.e., the case without release dates. Smith’s rule states that $1||\sum w_j C_j$ is solved by scheduling the jobs in order of nondecreasing p_j/w_j ratio. Our first heuristic schedules the jobs according to the following rule: at each decision point schedule the available job with the smallest p_j/w_j ratio, where the first decision point is the smallest release date, and the k th decision point is either the completion time of the job scheduled in the $(k - 1)$ th position or, in case there are no jobs available at that time, the smallest release date among the unscheduled jobs.

The other three heuristics schedule the jobs according to some ordering based on the values of the current linear programming solution. We have used the following three orderings:

- schedule jobs in order of nondecreasing mean start time $\sum_{t=1}^{T-p_j+1} (t-1)x_{jt}$;
- schedule jobs in order of nondecreasing maximum start time $\argmax_t \{x_{jt}\}$;
- schedule jobs in order of nondecreasing first start time $\argmin_t \{x_{jt} > 0\}$.

In most situations, the ordering based on the mean start time provides the best feasible solution. However, since these heuristics take very little time we always apply all of them. Furthermore, note that these heuristics are applied every time that a linear program has been solved, whereas the first heuristic is applied only once.

Let z_{UB} denote an upper bound on the optimal value z_{IP} of the integer program. The gap G_{UB} corresponding to this upper bound is defined by

$$G_{UB} = \frac{z_{UB} - z_{IP}}{z_{IP}} \times 100\%$$

In Table 8a and 8b, we report for those 30-job instances that were not solved to optimality by the initial LP-relaxation the following numbers:

- G_{ratio}^{av} and G_{ratio}^{max} : the average gap for the first heuristic and the maximum of these gaps;
- G_{init}^{av} and G_{init}^{max} : the average gap for the best of the other three heuristics when applied to the solution of the initial LP relaxation and the maximum of these gaps;
- G_{root}^{av} and G_{root}^{max} : the average gap after the root node has been evaluated and the maximum of these gaps;

Observe that the gap after the root node has been evaluated may differ for the two variants we consider, since we do not generate cuts with right-hand side 2 with the positional branching scheme. The computational results show that the solutions to the

G_{ratio}^{av}	G_{ratio}^{max}	G_{init}^{av}	G_{init}^{max}	G_{root}^{av}	G_{root}^{max}
9.03	17.52	1.47	6.94	0.19	1.23

Table 8a. Performance of the primal heuristics when cuts with right-hand side 1 and 2 are added.

G_{ratio}^{av}	G_{ratio}^{max}	G_{init}^{av}	G_{init}^{max}	G_{root}^{av}	G_{root}^{max}
9.03	17.52	1.47	6.94	0.44	2.22

Table 8b. Performance of the primal heuristics when only cuts with right-hand side 1 are added.

LP-relaxations encountered during the solution process provide good starting-points for obtaining primal solutions; the heuristics based on these fractional solutions provide much better primal solutions than the first heuristic.

4.5 Overall performance and conclusions

Our final experiments aim at assessing the overall quality of the branch-and-cut algorithm that we have developed. We have tried to do this in three ways. First, we compare the performance of our branch-and-cut algorithm to that of a commercially available IP solver. Second, we compare the performance of our algorithm to the performance of other branch-and-cut algorithms that have been developed by other research groups for the same or related single-machine scheduling problems. Finally, we compare the performance of our branch-and-cut algorithm to the performance of combinatorial branch-and-bound algorithms for the same problem.

In Table 9, we present the results of the two variants of our algorithm that perform best, i.e., GUB branching plus cuts with right-hand-side 1 and 2 (GUB12), positional branching plus cuts with right-hand side 1 (POS1), as well as the results of the CPLEX mixed integer optimizer [CPLEX Optimization Inc. 1994]]. For each instance the optimal value (Z_{IP}) is given. For the variants GUB12 and POS1 we report on the number of nodes ($\#nds$), the computation time in seconds (time), the number of linear programs that had to be solved ($\#lps$), and the number of cuts that has been added ($\#cuts$). As CPLEX does not generate cuts, we only report on the number of nodes and the computation time. CPLEX was allowed to evaluate at most 50000 nodes. Consequently, it did not find the optimal value for all instances. Therefore, we give for each instance the value of the best solution that was found (Z_{CP}). Furthermore, '*****' indicates that CPLEX did not have enough memory. The results show that incorporating problem structure, in the form of cuts, primal heuristics, and branching strategies results in a faster and more robust algorithm.

Sousa and Wolsey [1992] developed a branch-and-cut algorithm using the time-indexed formulation, but with different classes of inequalities and different separation algorithms. We have only been able to compare our algorithms on a set of 4 instances.

		GUB12				POS1				CPLEX		
	Z_{IP}	#nds	time	#lps	#cuts	#nds	time	#lps	#cuts	#nds	time	Z_{CP}
1	9665	1	9.89	16	124	16	13.39	44	176	50000	4248	9668
2	7128	1	7.41	4	41	1	6.52	4	41	50000	2793	7128
3	9102	1	17.78	30	488	1	18.02	30	488	28534	2087	9102
4	9148	1	9.70	9	135	1	8.85	11	139	1391	105	9148
5	8885	15	25.35	33	98	36	23.42	63	133	*****		
6	9506	1	16.08	4	39	1	14.29	6	44	338	55	9506
7	10688	7	15.93	20	131	65	25.53	90	170	35029	2494	10688
8	9252	1	15.86	5	61	1	14.80	5	61	*****		
9	11350	17	53.05	73	821	4	76.27	75	1257	*****		
10	6495	9	12.60	27	86	286	52.51	337	130	203	22	6495
11	9980	1	11.26	21	148	1	10.42	21	148	358	49	9980
12	11081	15	38.91	84	616	141	93.09	221	900	*****		
13	8361	1	8.50	3	17	1	7.46	3	17	61	15	8361
14	10060	14	42.01	78	544	90	71.12	166	541	*****		
15	12139	7	32.97	47	433	126	60.47	173	447	50000	6936	1214
16	9739	247	691.51	607	2152	182	384.63	375	1727	*****		
17	10719	13	19.16	40	144	44	15.56	61	86	50000	4608	10719
18	11085	1	9.76	2	53	1	8.41	2	53	4	7	11085
19	9925	1	10.83	3	25	1	9.42	3	25	831	78	9925
20	10080	33	14.51	42	65	64	16.71	75	64	1557	113	10080
21	10512	1	13.01	13	71	1	12.24	13	71	193	23	10512
22	11518	7	23.70	32	207	700	553.89	886	803	*****		
23	9053	23	55.93	83	641	32	56.11	81	567	*****		
24	8627	1	11.01	6	61	1	10.29	6	61	*****		
25	9880	57	156.76	160	885	1010	638.79	1142	994	*****		
26	7690	1	10.59	10	48	1	9.64	10	48	*****		
27	12057	1	22.73	42	465	1	24.35	42	465	6658	561	12057
28	8710	11	25.86	56	348	194	169.10	289	832	50000	4594	8710
29	8986	1	8.91	10	33	1	7.93	11	39	42	7	8986
30	10330	1	7.55	2	16	1	7.37	2	16	1829	146	10330
31	10614	1	11.62	7	97	1	10.24	7	97	*****		
32	9782	1	6.90	4	35	1	6.03	4	35	*****		
33	11418	78	600.79	221	1313	12	205.43	70	803	*****		
34	6668	573	269.08	815	840	2108	564.08	2281	808	*****		
35	10527	3	25.74	59	405	14	31.78	60	418	*****		
36	5839	1	6.94	4	32	1	6.19	4	32	147	13	5839
37	7791	1	10.85	13	93	1	9.66	13	93	388	45	7791
38	8949	5	15.85	20	116	27	17.10	46	133	*****		
39	8674	5	13.97	19	164	159	27.16	171	141	4022	286	8674
40	11688	3	38.37	44	865	6	41.35	49	867	*****		

Table 9: Performance of the variants GUB12 and POS1 and of CPLEX

Each one is solved at the root node by both algorithms. Therefore, we cannot make any meaningful comparative statements. Crama and Spieksma [1993] developed a branch-and-cut algorithm for the case in which all processing times are equal. They tested their algorithm on two classes of problems. The first one has randomly generated objective coefficients c_{jt} . The second one has objective coefficients $c_{jt} = w_j(t - r_j)$ if $r_j \leq t \leq d_j$ and $c_{jt} = M$ otherwise, where M is some large integer; these instances model minimization of the weighted sum of the completion times subject to release dates and deadlines, where release dates and deadlines may be violated at large cost. For both problem classes the performance of the algorithms is comparable. However, their branch-and-cut algorithm incorporates classes of cuts that have been derived specifically for problems with equal processing times, whereas our algorithm does not.

For the problem $1|r_j|\sum w_j C_j$, several combinatorial branch-and-bound algorithms have been developed, i.e., branch-and-bound algorithms that are not based on linear programming relaxations. An example is the algorithm of Belouadah, Posner, and Potts [1992]. The lower bounds in their algorithm are based on job-splitting. It turns out that the number of nodes that has to be evaluated by their algorithm is much larger than the number of nodes that has to be evaluated by our algorithm, but that their algorithm requires much less computation time. This indicates that our lower bounds are better, but that we need much more time to compute them. This is due to the fact that we have to solve large linear programs. Recall that the size of the linear programs increases, when the processing times increase. Therefore, their algorithm can handle instances with larger processing times. However, our branch-and-cut algorithm can easily be applied to many types of scheduling problems with various objective functions, whereas these combinatorial branch-and-bound algorithms are typically designed for one specific problem type.

We conclude that the strength of the presented branch-and-cut algorithm is that it can be applied successfully to a wide range of single-machine scheduling problems, but that its weakness is the fact that in its current form it is limited to instances with a relatively small number of jobs and relatively small processing times, because otherwise the time to solve the linear programs becomes prohibitive. In a sequel paper (Van den Akker, Hurkens, and Savelsbergh [1995]), we will investigate column generation as a way of handling this weakness.

Finally, we like to make some general observations. First, the integrality gap, i.e., the difference between the optimal value of the integer program and the value of the linear programming relaxation, may not be a reliable indicator of the difficulty of the problem. The average integrality gap for the most difficult class of instances we considered ($n = 30, p_{\max} = 10$) is only 0.48 percent and the maximum gap is just 1.10 percent. However, most of the instances are very hard to solve. Second, developing a successful branch-and-cut algorithm involves much more than designing and implementing sepa-

ration algorithms. It also requires the development of appropriate branching schemes, primal heuristics, and cut generation schemes. The importance of the latter is often overlooked. For our algorithms the choice of the appropriate cut generation scheme was crucial to achieving good overall performance.

References

- H. BELOUADAH, M.E. POSNER, AND C.N. POTTS (1992). Scheduling with release dates on a single machine to minimize total weighted completion time. *Discrete Applied Mathematics* 36, 213-231.
- Y. CRAMA AND F.C.R. SPIEKSMAN (1993). *Scheduling jobs of equal length: complexity and facets*. Report M 93-09, Faculty of Economics, University of Limburg, Maastricht.
- CPLEX OPTIMIZATION, INC. (1994). *Using the CPLEX Callable Library, Version 3.0*.
- E.L. LAWLER, J.K. LENSTRA, AND A.H.G. RINNOOY KAN, AND D.B. SHMOYS (1993). Sequencing and scheduling: Algorithms and complexity, in: *Handbooks in Operations Research and Management Science, Vol. 4*, S.C. GRAVES ET AL. (ed.), North-Holland, Amsterdam, 445-522.
- J.K. LENSTRA, A.H.G. RINNOOY KAN, AND P. BRUCKER (1977). Complexity of machine scheduling problems. *Annals of Discrete Mathematics* 1, 343-362.
- G.L. NEMHAUSER, M.W.P. SAVELSBERGH, AND G.C. SIGISMONDI (1994). MINTO, a Mixed INTeger Optimizer. *Operations Research Letters* 15, 47-58.
- G.L. NEMHAUSER AND L.A. WOLSEY (1988). *Integer and Combinatorial Optimization*. Wiley, New York.
- M. QUEYRANNE AND A.S. SCHULZ (1994). *Polyhedral Approaches to Machine Scheduling* Preprint No. 408/1994, Department of Mathematics, Technical University of Berlin, Berlin.
- W.E. SMITH (1956). Various optimizers for single-stage production. *Naval Research Logistics Quarterly* 3, 59-66.
- J.P. DE SOUSA (1989). *Time-indexed formulations of non-preemptive single-machine scheduling problems*. PhD-thesis, Catholic University of Louvain, Louvain-la-Neuve.
- J.P. DE SOUSA AND L.A. WOLSEY (1992). A time-indexed formulation of non-preemptive single-machine scheduling problems. *Mathematical Programming* 54, 353-367.
- J.M. VAN DEN AKKER (1994). *LP-based solution methods for single-machine scheduling problems*. PhD-thesis, Eindhoven University of Technology.

J.M. VAN DEN AKKER, C.P.M. VAN HOESEL, AND M.W.P. SAVELSBERGH (1994). *A time-indexed formulation for single-machine scheduling problems: Characterization of facets*. In editorial process.

J.M. VAN DEN AKKER, C.A.J. HURKENS, AND M.W.P. SAVELSBERGH (1995). *Column generation for single-machine scheduling problems*. In preparation.