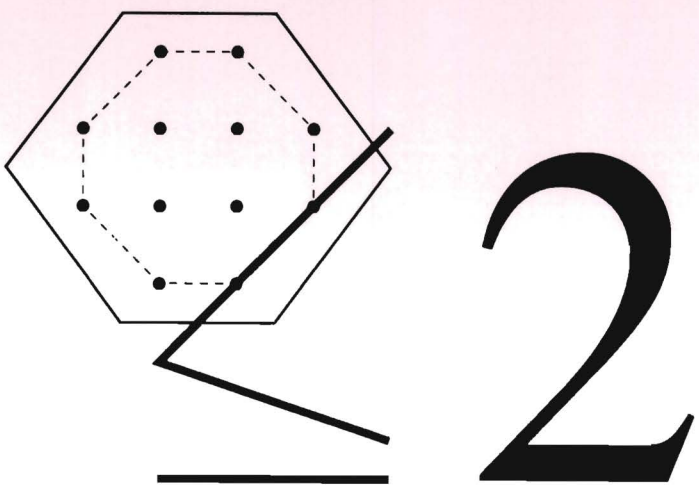
LP-based solution methods for single-machine scheduling problems

LP-based solution methods

for

single-machine scheduling problems



Marjan van den Akker

LP-based solution methods for single-machine scheduling problems

LP-based solution methods for single-machine scheduling problems

PROEFSCHRIFT

ter verkrijging van de graad van doctor aan de Technische Universiteit Eindhoven, op gezag van de Rector Magnificus, prof.dr. J.H. van Lint, voor een commissie aangewezen door het College van Dekanen in het openbaar te verdedigen op woensdag 21 december 1994 om 16.00 uur

door

JANNA MAGRIETJE VAN DEN AKKER

geboren te Gouda

Dit proefschrift is goedgekeurd door de promotoren

prof.dr. J.K. Lenstra
en
prof.dr. M.W.P. Savelsbergh

Copromotor: dr.ir. C.A.J. Hurkens

Ter herinnering aan mijn vader,
voor mijn moeder en mijn broer.

Acknowledgements

The process of writing this thesis has been completed successfully thanks to the help of a number of people.

In the first place, I want to thank my supervisors Jan Karel Lenstra and Martin Savelsbergh. Jan Karel was an excellent manager and gave many valuable comments on my manuscripts. Martin helped to ask the right question at the right moment. I want to thank him for the many stimulating discussions we had and for his help in writing this thesis.

I feel much indebted to Stan van Hoesel. While having coffee, we generated many good ideas concerning valid inequalities. I am grateful to Cor Hurkens, especially for a very useful remark concerning the characterization and for all his help with the computer. I thank my roommate Cleola van Eyl for always being there to solve practical problems and Ann Vandevelde for helping me to draw pictures with the computer. Furthermore, I want to thank Frits Spieksma for sending me unpublished notes and Bob Bixby for solving some large test instances.

I want to thank Jack Koolen for his friendship and for the many humorous discussions.

Finally, I want to thank Han for being involved in so many ways.

Marjan van den Akker

Contents

Chapter 1

Introduction

1.1 Scheduling

Imagine one of the following situations. In a factory products have to be manufactured by different machines, where each machine can handle at most one product at a time. Tasks assigned to a multiprogrammed computer system compete for processing on a timesharing basis. In a school lessons have to be assigned to teachers such that each lesson is given by a teacher capable of teaching that lesson. Planes requesting to use an airport runway are assigned places in a queue. In each of these situations a number of tasks has to be processed using one or more resources with limited capacity and availability, such as machines, tools, or employees. Usually, one wants to find an allocation of the tasks to the resources such that the workload of any resource does not exceed its capacity and the costs are minimal. Many of such problems can be modeled as *scheduling problems*, which are problems that concern the allocation of *jobs* to *machines* of limited capacity and availability. For instance, in the school example the jobs are the lessons and the machines are the teachers. Motivated and stimulated by the practical relevance, scheduling has become an important area of operations research.

We consider *nonpreemptive single-machine scheduling* problems. The usual setting for such a problem is as follows. A set of n jobs has to be scheduled on a single machine. Each job j ($j = 1, \ldots, n$) must be processed without interruption during a period of length 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 completion times such that the capacity and availability constraints are met and a given objective function is minimized. It is possible that *precedence relations* have been specified, i.e., for each job j ($j = 1, \ldots, n$) there is a set of jobs that have to precede job j and a set of jobs that have to succeed job j in any feasible schedule. It is further possible that each job is only available for processing during a prespecified period of time; job j may have a *release date* r_j, at which it becomes available, and a *deadline* $\tilde{d}_j$, by which it must be completed. In addition, it may have a positive weight w_j, which expresses its importance with

respect to the other jobs, and a *due date* d_j, which indicates the time by which it ideally should be completed. A due date differs from a deadline in the sense that a deadline is 'hard', whereas a due date may be exceeded at a certain cost. The weights and due dates are typically used to define the objective function. Some well-known objective functions are the weighted sum of the completion times and the weighted sum of the tardinesses; the tardiness T_j of job j is equal to the amount of time by which it exceeds its due date, if it is completed after that time, and 0 otherwise, i.e., $T_j = \max\{0, C_j - d_j\}$, where C_j denotes the completion time of job j. In the sequel, scheduling problems will be specified in terms of a three-field classification $\alpha|\beta|\gamma$, where

- α describes the machine environment: for a single-machine scheduling problem we have $\alpha = 1$;
- β describes the job characteristics: *prec* indicates that precedence relations have been specified, *sepa* that the precedence relations are series-parallel, $p_j = p$ that all processing times are equal to p, and r_j and $\bar{d}_j$ that release dates and deadlines have been specified, respectively;
- γ represents the objective function.

Note that other possibilities for α and β exist. We restricted ourselves to mentioning those values that occur in this thesis. As a test problem for the methods that we study in this thesis, we use $1|r_j|\sum w_jC_j$. This problem is $\mathcal{NP}$-hard, even if $w_j = 1$ for all j [Lenstra, Rinnooy Kan, and Brucker 1977]. An example of an instance with four jobs is given Table 1.1.

j	1	2	3	4
r_j	1	3	7	8
p_j	3	2	2	3
w_j	1	2	1	2

Table 1.1: An instance of $1|r_j|\sum w_jC_j$.

Feasible schedules can be represented through a *Gantt chart*. In a Gantt chart the horizontal axis represents time, and for each job a rectangle on this axis indicates the time interval during which the job is processed by the machine. For our example the Gantt chart in Figure 1.1 represents a feasible schedule with $\sum_{j=1}^{n} w_jC_j = 49$.

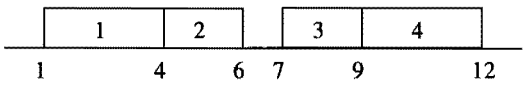


Figure 1.1: A Gantt Chart.

1.2 Complexity

The scheduling problems that are considered in this thesis fall in the area of *combinatorial optimization.* Combinatorial optimization involves problems in which we have to choose an optimal solution from a finite number of potential solutions. We consider a solution to be optimal if it has minimum objective value. Many combinatorial optimization problems, including some well-known single-machine scheduling problems, are $\mathcal{NP}$-hard. This means that no algorithm is known that solves each instance of the problem to optimality in time polynomial with respect to the size of the instance, and that it is considered very unlikely that such an algorithm exists. Hence, if we wish to solve the problem to optimality, then we should allow for algorithms that may take exponential time. An example of such an approach is *branch-and-bound*, which implicitly enumerates the set of all feasible solutions. Branch-and-bound proceeds by splitting the set of feasible solutions into subsets (this is the branching part). For each generated subset a lower bound on the value of the best solution is computed. If this lower bound exceeds the value of a known feasible solution, then we can skip the subset without further inspection (this is the bounding part). In order to be able to apply branch-and-bound effectively, it is of the utmost importance to find good lower bounds that can be easily computed.

1.3 Integer linear programming

Combinatorial optimization problems can be formulated as integer linear programs, which implies that integer linear programming is $\mathcal{NP}$-hard. If we relax the integrality conditions, then we obtain a linear programming problem, which is called the LP-relaxation; the optimal value of this problem is a lower bound on the optimal value of the original problem. Since there are algorithms that solve any instance of linear programming to optimality in polynomial time (Khachiyan [1979], Karmarkar [1984]), this lower bound can be computed efficiently. Solving LP-relaxations can hence be used as a lower bounding strategy in a branch-and-bound algorithm.

For nonpreemptive single-machine scheduling problems different mixed-integer programming formulations have been proposed; we give a survey in the next chapter. In this thesis, we consider a *time-indexed formulation* that has been studied before by Sousa and Wolsey [1992]. This 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 $1, 2, \ldots, T$; each job must be completed by time T. We introduce a binary variable x_{jt} for each job j ($j = 1, \ldots, n$) and time period t ($t = 1, \ldots, T - p_j + 1$), which equals 1 if job j is started in period t and 0 otherwise. The formulation is as follows:

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

subject to

$$\sum_{t=1}^{T-p_j+1} x_{jt} = 1 \qquad (j = 1, \ldots, n), \tag{1.1}$$

$$\sum_{j=1}^{n} \sum_{s=t-p_j+1}^{t} x_{js} \leq 1 \qquad (t = 1, \ldots, T), \tag{1.2}$$

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

Constraints (1.1) state that each job has to be processed exactly once, and constraints (1.2) state that the machine handles at most one job during any time period. In terms of this formulation, the schedule in Figure 1 is given by $x_{12} = 1$, $x_{25} = 1$, $x_{38} = 1$, $x_{4,10} = 1$.

1.4 Polyhedral combinatorics

Even though the lower bounds obtained by the LP-relaxation may be quite strong, we need better bounds to solve large combinatorial optimization problems. This strengthening of the lower bounds can be achieved by the use of *polyhedral combinatorics*. In this section, we briefly discuss the main ideas of polyhedral combinatorics; we refer to Schrijver [1986] and Nemhauser and Wolsey [1988] for a more elaborate description.

We start with some definitions. These definitions will be clarified by the example depicted in Figure 1.2. A *polyhedron* is defined as a set of points that satisfy a finite number of linear inequalities. We only consider *rational polyhedra*, i.e., polyhedra that can be described by inequalities with integral coefficients. If a polyhedron is bounded, then we call it a *polytope*. We define the convex hull of a set of vectors $x_1, \ldots, x_k$ as the set of points y that can be written as a *convex combination* of these vectors, i.e., $y = \sum_{j=1}^{k} \alpha_j x_j$ for some combination of nonnegative values α_j with $\sum_{j=1}^{k} \alpha_j = 1$. Minkowsky [1896] showed that any polytope is the convex hull of a finite number of extreme points, where an *extreme point* is defined as a point in the polyhedron that cannot be written as a convex combination of two other points in the polyhedron. Conversely, Weyl [1935] showed that the convex hull of a finite number of points is a polytope. The polytope in Figure 1.2 is described by the inequalities $x_1 \geq 0$, $x_1 + x_2 \geq 1$, $5x_1 - 2x_2 \geq 0$, $2x_1 + 6x_2 \leq 17$, and $4x_1 + x_2 \leq 12$. Its extreme points are $(1, 0)$, $(\frac{2}{7}, \frac{5}{7})$, $(1, \frac{5}{2})$, $(\frac{5}{2}, 2)$, and $(3, 0)$.

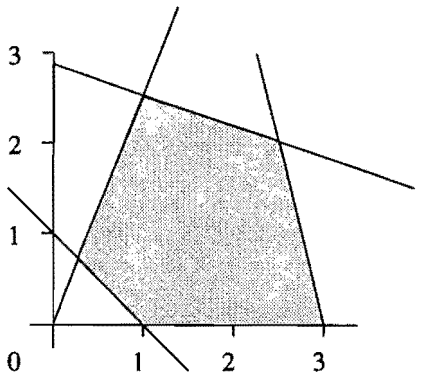


Figure 1.2: A polytope.

We define the affine hull of the set of vectors $x_1, \ldots, x_k$ as the set of points y that can be written as an *affine combination* of these vectors, i.e., $y = \sum_{j=1}^{k} \alpha_j x_j$ for some combination of values α_j such that $\sum_{j=1}^{k} \alpha_j = 1$. Note that the affine hull of two different points is the line through these points, whereas the convex hull is equal to the line-segment between these two points. A set $x_1, \ldots, x_k$ of vectors is called *affinely independent* if none of these vectors is in the affine hull of the other $k-1$ vectors; this is the case if and only if the system of equations $\sum_{j=1}^{n} \alpha_j x_j = 0$ and $\sum_{j=1}^{n} \alpha_j = 0$ has $\alpha_j = 0$ $(j = 1, \ldots, k)$ as a unique solution. Observe that the maximum number of affinely independent points in a k-dimensional space is $(k+1)$. The *dimension* of a polyhedron is the dimension of the smallest affine space that contains the polyhedron. A polyhedron P has dimension k if the maximum number of affinely independent points in P is $(k+1)$. We have that for a k-dimensional polyhedron P in $\mathcal{R}^n$ there are exactly $(n-k)$ linearly independent equations that are satisfied by all points in P. A polyhedron in $\mathcal{R}^n$ that has dimension n is called *full-dimensional*. In order to show that the dimension of a polyhedron P in $\mathcal{R}^n$ equals k, we have to determine $(n-k)$ linearly independent equations satisfied by all elements of P and $(k+1)$ affinely independent points in P; the first part shows that k is an upper bound on the dimension of P, whereas the second part shows that k is a lower bound. Instead of determining $(k+1)$ affinely independent points, one can also determine k linearly independent *direction vectors* in P, where a direction vector in P is a vector that can be written as $x - y$ for some $x, y \in P$. As the affine independence of the vectors x_j $(j = 1, \ldots k+1)$ is equivalent to the linear independence of the vectors $x_j - x_1$ $(j = 2, \ldots, k+1)$, the determination of $(k+1)$ affinely independent points boils down to the determination of k linearly independent direction vectors with a fixed endpoint. In many situations it is easier

to point out linearly independent direction vectors. The polytope in Figure 1.2 is a polytope in $\mathcal{R}^2$; its dimension is hence at most 2. Since the vectors $(1,0)$, $(3,0)$, and $(\frac{2}{7}, \frac{5}{7})$ are affinely independent, its dimension equals 2, i.e., it is full-dimensional.

Consider an integer linear programming problem given by $\min\{cx \mid x \in S\}$, where $S = \{x \mid Ax \leq b, x \geq 0, x \text{ integral }\}$ with A and b integral. The LP-relaxation is given by $\min\{cx \mid Ax \leq b, x \geq 0\}$. By definition, the set of feasible solutions of the LP-relaxation, denoted by P_{LP}, is a polyhedron. It is easy to see that the optimal value of the integer programming problem $\min\{cx \mid x \in S\}$ satisfies

$$\min\{cx \mid x \in S\} = \min\{cx \mid x \in \text{ conv}(S)\}.$$

Using results of Giles and Pulleyblank [1979], one can show that the convex hull of S is also a rational polyhedron (see Nemhauser and Wolsey [1988]). This polyhedron is clearly contained in P_{LP}. Note that, if we consider a problem in terms of binary variables, then S is finite and the convex hull of S hence is a polytope. In Figure 1.3 the convex hull of the integral points in the polytope of Figure 1.2 is indicated by a dashed quadrangle.

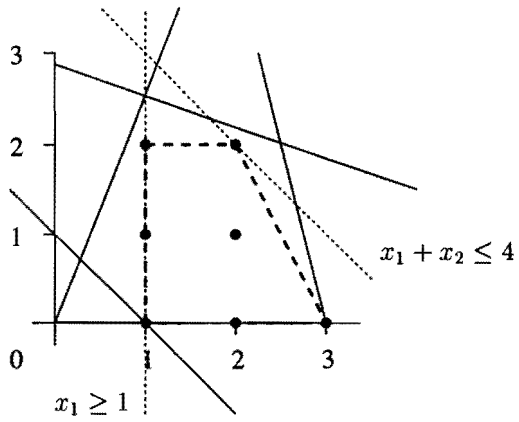


Figure 1.3: The convex hull of the set of integral solutions.

Suppose that we have a linear description of the convex hull of S. Since, by definition, each extreme point of the convex hull is integral, we can solve the integer linear programming problem to optimality through the simplex method, which is known to find an extreme point. In general, however, it is hard to find a linear description of the convex hull of S, which could be expected, since the integer linear programming problem is $\mathcal{NP}$-hard. Therefore, we try to find a partial description of

the convex hull of S. By studying the polyhedral structure, we derive inequalities that cut off parts of the set of feasible solutions to the LP-relaxation, but that do not eliminate any integral solution. Since such inequalities are satisfied by all points in the convex hull of S, they are called *valid inequalities* for the convex hull of S. Usually, different classes of valid inequalities can be found. These classes may contain many inequalities; some well-known classes consist of an exponential number of inequalities.

If $\pi x \leq \pi_0$ is a valid inequality for a polyhedron P, then the intersection of P and the hyperplane $\pi x = \pi_0$ is called the *face* of the polyhedron P induced by $\pi x \leq \pi_0$. Observe that a face of a polyhedron is a polyhedron too. We are particularly interested in *facet inducing inequalities*; these are valid inequalities that induce a face of dimension $\dim(P) - 1$; such a face is called a *facet*. The set of facet inducing inequalities of a polyhedron constitutes the set of inequalities that are necessary in the description of the polyhedron. Let $\pi x \leq \pi_0$ be a valid inequality, and let F denote the face induced by this inequality. If $\pi x \leq \pi_0$ is not satisfied at equality by all elements of the polyhedron, which is trivial to show in most situations, then the dimension of F is clearly at most $\dim(P)-1$. One can then show that $\pi x \leq \pi_0$ is facet inducing by determining $\dim(P)$ affinely independent vectors in F or by pointing out $\dim(P) - 1$ linearly independent direction vectors in F.

The inequality $x_1 + x_2 \leq 4$ is valid for the convex hull of the integral points in the polytope in Figure 1.3 and cuts off part of the set of fractional solutions. The face induced by this inequality is the point $(2, 2)$; this inequality is hence not facet inducing. The inequality $x_1 \geq 1$ is facet inducing, since the face induced by this inequality is the line segment between the points $(1, 0)$ and $(1, 2)$.

1.5 Branch-and-cut

In the previous section we have seen that, in order to find a tight lower bound on the optimal value of an integer linear programming problem, we must derive classes of valid inequalities that provide a partial linear description of the convex hull of the set of feasible solutions. Since in many situations these classes contain an exponential number of inequalities, it is impossible to compute this lower bound by including these inequalities in the formulation and solving the resulting linear programming problem. Furthermore, only the constraints that are close to the optimal solution are important. Therefore, to solve an integer linear program, we proceed in the following way. We start by solving the LP-relaxation. If this yields an integral solution, then we are done. It can be proved that for some polynomially solvable problems there exist integer linear programming formulations for which the solution to the LP-relaxation is always integral; an example is given in Section 3.2. In general, however, the solution is likely to be fractional. In this case, we

select from the set of known valid inequalities one or more inequalities that are violated by the current fractional solution. Adding these inequalities to our linear programming problem yields a new problem to which the current fractional solution is no longer feasible. We solve this new problem and, in case of a fractional solution, add violated inequalities again. This process is repeated until either an integral solution is found, or until we cannot find any valid inequality that is violated by the current fractional solution. This procedure is called a *cutting plane algorithm*, since the added inequalities cut off the current fractional solution. The process of finding violated inequalities for a given fractional solution is called *separation*.

If we do not find an integral solution by just adding cutting planes, then we proceed by applying branch-and-bound. In case of a formulation that uses binary variables, one possible branching strategy is to fix the value of a fractional variable at either 0 or 1. Obviously, there are many other possible branching strategies. It is well known that the choice of a branching strategy has great influence on the performance of a branch-and-bound algorithm. Unfortunately, it is difficult to indicate beforehand which branching strategy will perform best. Often, however, it pays off to take the specific structure of the problem into account. A branching strategy partitions the original problem into smaller subproblems that are characterized by one or more additional constraints; for example, if we fix the value of a variable x at 0, then the additional constraint is $x = 0$. For each subproblem we compute a lower bound by applying the cutting plane algorithm to the current formulation including the additional constraint. For this reason, such a branch-and-bound algorithm is called a *branch-and-cut* algorithm. Since we can discard any subproblem with a lower bound that is greater than or equal to the value of the best known solution, we are also interested in good integral solutions. These may be found by applying some heuristic, for instance, by transforming a fractional solution into an integral one.

1.6 Column generation

Computational experiments indicate that the time-indexed formulation yields good lower bounds. A limitation to the applicability of these lower bounds in a branch-and-cut algorithm is formed by the considerable amount of time needed to compute them, i.e., the time needed to solve LP-relaxations. This is mainly due to the large number of constraints in the formulation. Empirical findings indicate that the typical number of iterations needed by the simplex method increases proportionally with the number of rows and only very slowly with the number of columns (see for example Dantzig [1963]); it is sometimes said that for a fixed number of rows the number of iterations is proportional to the logarithm of the number of columns (see Chvátal [1983]). The time-indexed formulation contains $n + T$ constraints and approximately nT variables, where $T \geq \sum_{j=1}^{n} p_j$. Hence, especially in case of

instances with large processing times, we may expect large running times when the linear programming problems are solved through the simplex method.

To reduce the number of constraints we apply *Dantzig-Wolfe decomposition.* This decomposition proceeds as follows. The set of constraints $Ax \leq b$ is partitioned into two subsets, say $A^{(1)}x \leq b^{(1)}$ and $A^{(2)}x \leq b^{(2)}$. We assume that the polyhedron described by the constraints $A^{(2)}x \leq b^{(2)}$ is bounded; the boundedness assumption is not necessary and is made purely for simplicity of exposition. Any point satisfying these constraints can hence be written as a convex combination of extreme points, say $x = \sum_{k=1}^{K} \lambda_k x^k$, with $\sum_{k=1}^{K} \lambda_k = 1$ and $\lambda_k \geq 0$ $(k = 1, \ldots, K)$. If we substitute this representation in the objective function and in the constraint set $A^{(1)}x \leq b^{(1)}$, then we obtain a problem in the variables λ_k, which is called the *master problem* and is given by:

$$\min \sum_{k=1}^{K} (cx^k)\lambda_k$$

subject to

$$\sum_{k=1}^{K} (A^{(1)}x^k)\lambda_k \leq b^{(1)}, \tag{1.3}$$

$$\sum_{k=1}^{K} \lambda_k = 1, \tag{1.4}$$

$$\lambda_k \geq 0 \qquad (k = 1, \ldots, K).$$

Observe that by this reformulation the number of constraints has been decreased. However, as the number of variables equals the number of extreme points of the polytope $A^{(2)}x \leq b^{(2)}$, it may have been increased enormously. This does not really matter, though, since the application of Dantzig-Wolfe decomposition enables us to solve the master problem by *column generation.*

The column generation technique has been developed to solve linear programs with a large number of variables. A column generation algorithm always works with a *restricted* problem in the sense that only a subset of the variables is taken into account; the variables outside the subset are implicitly fixed at zero. Suppose that we have selected a subset of the columns that yields a feasible restricted problem. From the theory of linear programming it is known that, after solving this restricted problem to optimality, each included variable has nonnegative *reduced cost.* The reduced cost of the variable λ_k is given by

$$cx^k - \sum_{i=1}^{m} (A^{(1)}x^k)_i u_i - v,$$

where u_i denotes the dual variable of the ith constraint, and v denotes the dual variable of the convexity constraint. If each variable that is not included in the restricted problem also has nonnegative reduced cost, then we have found an optimal solution to the complete problem. If there exist variables with negative reduced cost, then we have to choose one or more of these variables to be included in the restricted problem. The main idea behind column generation is that the occurrence of variables with negative reduced cost is not checked by enumerating all variables but by solving an optimization problem. This optimization problem is called the *pricing problem* and is defined as the problem of finding the variable with minimal reduced cost. This means that we have to find the extreme point x_k of the polytope described by the constraints $A^{(2)}x \leq b^{(2)}$ for which $cx^k - \sum_{i=1}^{m}(A^{(1)}x^k)_i u_i - v$ is minimal. To solve this problem, we use the structure imposed by the constraints $A^{(2)}x \leq b^{(2)}$, i.e., we consider 'solutions' to the original problem that are feasible with respect to only a subset of the constraints. To apply column generation effectively, it is important to find a good method for solving the pricing problem.

If the column generation algorithm does not find an integral solution, then we proceed by applying branch-and-bound. A branch-and-bound algorithm in which the lower bounds are computed by solving LP-relaxations through column generation is called a *branch-and-price* algorithm. Observe that in any subproblem that arises from branching, only columns may be added that correspond to extreme points of the polytope $A^{(2)}x \leq b^{(2)}$ that satisfy the constraints defining the subproblem. This leads to a pricing problem with additional constraints that may complicate the problem. Therefore, it is very important to choose a branching strategy that does not deteriorate the structure of the pricing problem too much.

Up to now little attention has been paid to the combination of column and row generation, i.e., the combination of column generation and the addition of valid inequalities. If constraints are added to the master problem, then the dual variables corresponding to these constraints will occur in the reduced cost, and will hence play a role in the pricing problem, which may seriously complicate this problem. In most situations column generation is applied to a problem formulation obtained through Dantzig-Wolfe decomposition. Two kinds of valid inequalities can hence be distinguished: valid inequalities that are obtained from the original formulation and are translated to the reformulation, and valid inequalities that are obtained directly from the reformulation. In Chapter 5, we show that in many cases valid inequalities of the first kind can easily be handled, whereas valid inequalities of the second kind seem harder to deal with.

1.7 Outline of the thesis

The main goal of our research is to provide basic tools for the development of improved LP-based branch-and-bound algorithms for single-machine scheduling

problems. For these problems several mixed-integer programming formulations have been proposed. None of these formulations clearly dominates all others. In Chapter 2, we discuss these different formulations and their respective advantages and disadvantages.

In this thesis, we consider the time-indexed formulation that has been studied by Sousa and Wolsey [1992]. In Chapter 3, we investigate the polyhedral structure of this formulation. We first show that for some special objective functions the LP-relaxation of the formulation solves the problem. The main results presented in this chapter are complete characterizations of all facet inducing inequalities with right-hand side 1 or 2. This chapter is based on Van den Akker, Van Hoesel, and Savelsbergh [1993].

In Chapter 4, we first derive separation algorithms for the classes of inequalities that were identified in Chapter 3. After that, we discuss the development and implementation of a branch-and-cut algorithm for $1|r_j|\sum w_jC_j$ and give computational results. This chapter is based on Van den Akker, Hurkens, and Savelsbergh [1994A].

Due to the fact that the time-indexed formulation contains a large number of constraints and variables, the major part of the computation time of the branch-and-cut algorithm presented in Chapter 4 is spent on solving linear programming problems. It may hence be worthwhile to solve these problems by column generation. This technique is discussed in Chapter 5. We first show how to solve the LP-relaxation. Then we discuss how to combine column generation and the addition of the facet inducing inequalities that were identified in Chapter 3. We also consider the combination of row and column generation in a more general context. We further study the possibility of applying branch-and-bound. Finally, we discuss the application of so-called key formulations, We also give some computational results. This chapter is based on Van den Akker, Hurkens, and Savelsbergh [1994B].

Chapter 2

Mixed-integer programming formulations for single-machine scheduling problems

2.1 Introduction

Recently developed polyhedral methods have yielded substantial progress in solving important $\mathcal{NP}$-hard combinatorial optimization problems. A well-known example is the traveling salesman problem [Padberg and Rinaldi, 1991]. Relatively few papers, however, have been written on polyhedral methods for scheduling problems. Investigation and development of such methods is important, because for many $\mathcal{NP}$-hard scheduling problems it is difficult to obtain tight lower bounds. Lower bounds can be used in a branch-and-bound algorithm or may help to determine the quality of solutions obtained through some heuristic. Moreover, the lower bounds obtained by a polyhedral method can often be used to derive good solutions. Most of the research conducted on polyhedral methods for scheduling problems deals with single-machine scheduling problems. For these problems several mixed-integer programming formulations in terms of different kinds of variables have been proposed. None of these formulations clearly dominates all others. In this chapter, we discuss these formulations and their respective advantages and disadvantages.

2.2 Formulations based on completion times

We describe formulations in which the variables represent the *completion times* C_j of jobs $j = 1, \ldots, n$. The problem that we use to illustrate this formulation is $1||\sum w_j C_j$. Smith [1956] showed that this problem is solved in polynomial time by scheduling the jobs in order of nonincreasing ratio w_j/p_j. The problem can be

formulated as follows:

$$\min \sum_{j=1}^{n} w_j C_j$$

subject to

$$C_j \geq p_j \qquad (j = 1, \ldots, n), \tag{2.1}$$

$$C_j - C_i \geq p_j \vee C_i - C_j \geq p_i \qquad (i, j = 1, \ldots, n, i < j). \tag{2.2}$$

Constraints (2.1) state that no job can start before time zero. The disjunctive constraints (2.2) model the machine availability constraints. They state that no two jobs can overlap in their execution. Observe that this formulation is not an integer linear programming formulation; it contains disjunctive constraints instead of integrality constraints. This formulation has been studied by Queyranne [1986]. He shows that a complete description of the convex hull of the set of feasible solutions is given by the inequalities

$$\sum_{j \in S} p_j C_j \geq \sum_{i,j \in S: i \leq j} p_i p_j \qquad (S \subseteq \{1, \ldots, n\}). \tag{2.3}$$

This description contains an exponential number of inequalities. However, there exists an $O(n \log n)$ separation algorithm for these inequalities.

Queyranne and Wang [1991A] incorporate precedence constraints in the above formulation. Let A denote the set of ordered pairs of jobs between which a precedence relation has been specified: $(i, j) \in A$ means that job i has to be processed before job j. These precedence constraints can be included in the formulation by replacing constraint (2.2) for any pair of jobs $(i, j) \in A$ by

$$C_j - C_i \geq p_j.$$

They present three classes of valid inequalities. These classes are associated with parallel decomposition, series decomposition, and Z-induced subgraphs, where a Z-induced subgraph is a subgraph on four points, say v_1, v_2, v_3, v_4 with edges (v_1, v_2), (v_3, v_4), and (v_3, v_2). The first class consists of the inequalities (2.3). They show under which conditions the inequalities in these classes are facet inducing. They consider the special case of series-parallel precedence constraints, i.e., $1|sepa|\sum w_j C_j$. Lawler [1978] showed that this problem is polynomially solvable, and that the general problem $1|prec|\sum w_j C_j$ is $\mathcal{NP}$-hard. The last result was also obtained by Lenstra and Rinnooy Kan [1978]. Queyranne and Wang show that, in case the precedence constraints are series-parallel, the facet-inducing inequalities in the first two classes provide a complete description of the convex hull of the set of feasible solutions. Note that in this case the precedence graph does not contain Z-induced subgraphs and hence the third class of inequalities does not apply. They present a

computational study on the problem with general precedence constraints in a subsequent report (Queyranne and Wang [1991B]). In their cutting plane algorithm, they consider inequalities associated with parallel decomposition and a subset of the class of inequalities associated with series decomposition. For all these inequalities separation can be performed in polynomial time. In all their test problems the gap between the lower bound obtained by the cutting plane algorithm and the upper bound provided by a feasible solution is less than 1 percent. To find good feasible solutions two heuristics are used; one of these heuristics is based on the optimal solution of the LP-relaxation.

Von Arnim and Schrader [1993] also consider this formulation for the problem $1|prec|\sum w_jC_j$. They study the special case in which the precedence constraints are P_4-sparse, which means that in the precedence graph any subset of five jobs contains at most one Z-induced subgraph. Note that this case includes the case of series-parallel precedence constraints. They give three types of valid inequalities and show that these inequalities completely describe the convex hull of the set of feasible solutions. Precedence constraints are also studied by Von Arnim and Schulz [1994]. For general precedence constraints, they present a description of the geometric structure of the convex hull of the set of feasible solutions that depends on the series decomposition of the precedence graph and prove a dimension formula. They further show which of the inequalities identified by Von Arnim and Schrader [1993] are facet inducing and give a minimal description of the convex hull of the set of feasible solutions for the special case of P_4-sparse precedence constraints.

In his pioneering work on scheduling polyhedra, Balas [1985] investigates completion time based formulations for the job shop scheduling problem. As a special case, he considers the problem $1|r_j|\sum w_jC_j$ and derives some facet inducing inequalities for this problem.

The main advantage of this formulation is that it contains only n variables. This implies that instances with a large number of jobs or with large processing times can be handled by this formulation in the sense that these instances do not yield extremely large linear programs. A disadvantage is that only problems with a linear objective function in the completion times can be modeled in a straightforward way. In other situations, new variables and constraints have to be introduced. If the objective is to minimize the makespan, then we have to represent the makespan by a new variable $C_{\max}$ and add the constraints $C_{\max} \geq C_j$ $(j = 1, \ldots, n)$. If the objective is to minimize total weighted tardiness, where the tardiness T_j of job j is defined as $\max\{0, C_j - d_j\}$ for a given due date d_j, then this can be modeled by adding for each job j a nonnegative variable T_j denoting the tardiness of job j and adding the constraints $T_j \geq C_j - d_j$ $(j = 1, \ldots, n)$. Observe that, if extra variables are added, then the results concerning complete descriptions no longer hold.

2.3 Formulations based on start times and sequence determining variables

The formulation discussed in the previous section uses disjunctive constraints to model the choice of starting job i before or after job j. In this section, we discuss formulations in which *sequence determining variables* are used to model these two cases. They contain variables δ_{ij} $(i, j = 1, \ldots, n, i \neq j)$ such that $\delta_{ij} = 1$ if job i is processed before job j, and $\delta_{ij} = 0$ otherwise. Furthermore, they contain variables t_j $(j = 1, \ldots, n)$ that denote the start times of the jobs. Observe that, since the start time t_j and the completion time C_j of job j differ by the given processing time p_j, the formulation in the previous section can be stated in terms of start times as well. Following the existing papers on this formulation, we presented it in terms of completion times.

Two formulations based on start times and sequence determining variables have been proposed for the problem $1||\sum w_j C_j$. In the first formulation F_1 each disjunctive constraint is modeled by two constraints that contain a large constant; this is called the big M method. This formulation is given by:

$$\min \sum_{j=1}^{n} w_j t_j$$

subject to

$$t_j - t_i \geq p_i - M(1 - \delta_{ij}) \qquad (i, j = 1, \ldots, n, i \neq j), \tag{2.4}$$

$$\delta_{ij} + \delta_{ji} = 1 \qquad (i, j = 1, \ldots, n, i < j), \tag{2.5}$$

$$\delta_{ij} \in \{0, 1\} \qquad (i, j = 1, \ldots, n, i \neq j),$$

$$t_j \geq 0 \qquad (j = 1, \ldots, n),$$

where M should be at least $\sum_{j=1}^{n} p_j$. The second formulation F_2 is given by:

$$\min \sum_{j=1}^{n} w_j t_j$$

subject to

$$t_j \geq \sum_{i:i \neq j} p_i \delta_{ij} \qquad (j = 1, \ldots, n), \tag{2.6}$$

$$\delta_{ij} + \delta_{ji} = 1 \qquad (i, j = 1, \ldots, n, i < j), \tag{2.7}$$

$$\delta_{ij} + \delta_{jk} + \delta_{kj} \leq 2 \qquad (i, j, k = 1, \ldots, n, i \neq j \neq k), \tag{2.8}$$

$$\delta_{ij} \in \{0, 1\} \qquad (i, j = 1, \ldots, n).$$

Constraints (2.8) state that the order of the jobs defined by the variables δ_{ij} is acyclic. In formulation F_1 this is imposed by the constraints (2.4). Observe that the number of constraints in F_1 is of order n^2, whereas the number of constraints in F_2 is of order n^3. Therefore, it may be worthwhile to generate the constraints (2.8) as cutting planes. An important disadvantage of formulation F_1 is that the big M makes the LP-relaxation very weak: $\delta_{ij} = \frac{1}{2}$ for all i, j and $t_j = 0$ for all j is a feasible solution. Therefore, this formulation is only used as a theoretical concept and not for solving real problems.

For the problem $1||\sum w_j C_j$, Wolsey [1985] shows that the projection into the t-space, i.e., the subspace spanned by the variables t_j, of the polytope described by the constraints

$$t_j \geq \sum_{i:i \neq j} p_i \delta_{ij} \qquad (j = 1, \ldots, n),$$

$$\delta_{ij} + \delta_{ji} = 1 \qquad (i, j = 1, \ldots, n, i < j),$$

$$\delta_{ij} \geq 0 \qquad (i, j = 1, \ldots, n),$$

is the convex hull of the set of feasible schedules. In the previous section, we have seen that Queyranne [1986] directly describes this polytope in the t-space. Peters [1988] showed that for this problem constraints (2.8) can be omitted from the LP-relaxation of F_2 without changing its optimal value. Nemhauser and Savelsbergh [1992] show that the LP-relaxation of F_2 gives the optimal value. Hence, this also holds if the constraints (2.8) are omitted. Note that the last two results are implied by the results presented by Wolsey [1985]; they were however established independently. Nemhauser and Savelsbergh further show that the constraints in F_2 give a complete description of the convex hull of the set of feasible solutions in the space spanned by the variables t_j and δ_{ij}. They also show that the inequalities derived in Queyranne [1986] can be written as nonnegative linear combinations of the inequalities in the above linear program.

Wolsey [1990] investigates the (t, δ)-formulations for $1|prec|\sum w_j C_j$, i.e., the problem of minimizing the weighted sum of the completion times subject to precedence constraints. Precedence constraints can easily be included in the formulation. Again, let A denote the set of pairs (i, j) such that job i has to precede job j. The constraints $\delta_{ij} + \delta_{ji} = 1$ are replaced by $\delta_{ij} = 1$ and $\delta_{ji} = 0$ for each pair $(i, j) \in A$. Define S_j and P_j as the set of successors and predecessors of job j, respectively, i.e., $S_j = \{k|(j, k) \in A\}$ and $P_j = \{k|(k, j) \in A\}$. Wolsey presents the following two classes of valid inequalities:

$$t_j - t_i \geq p_i + \sum_{k \in S_i \cap P_j} p_k + \sum_{k \in S_i \setminus P_j} p_k \delta_{kj} + \sum_{k \in P_j \setminus S_i} p_k \delta_{ik} + \sum_{k \in N \setminus (S_i \cup P_j)} p_k (\delta_{kj} - \delta_{ki}) \qquad ((i,j) \in A), \tag{2.9}$$

and

$$t_j - t_i \geq p_i + \sum_{k \in S_i \cap P_j} p_k + \sum_{k \in S_i \setminus P_j} p_k \delta_{kj} + \sum_{k \in P_j \setminus S_i} p_k \delta_{ik} \qquad ((i,j) \in A). \tag{2.10}$$

He shows that all inequalities derived by Queyranne and Wang [1991A] can be written as nonnegative linear combinations of inequalities in these classes and inequalities in formulation F_2. As Queyranne and Wang found a complete description of the convex hull of the set of feasible solutions in the special case of series-parallel precedence constraints, the inequalities above must in this special case also lead to a complete description in the t-space. Define Q as the set containing all nonnegative (t, δ) that satisfy

$$t_j \geq \sum_{i : i \neq j} \delta_{ij} p_i \qquad (j = 1, \ldots, n),$$

$$\delta_{ij} + \delta_{ji} = 1 \qquad ((i,j) \notin A),$$

$$\delta_{ij} = 1 \wedge \delta_{ji} = 0 \qquad ((i,j) \in A),$$

$$t_j - t_i \geq p_i + \sum_{k \in S_i \cap P_j} p_k + \sum_{k \in S_i \setminus P_j} p_k \delta_{kj} + \sum_{k \in P_j \setminus S_i} p_k \delta_{ik} \qquad ((i,j) \in A),$$

$$\delta_{ij} \geq 0 \qquad ((i,j) \notin A).$$

Wolsey shows that the projection of Q into the t-space gives the convex hull of the set of feasible schedules. He further shows that the optimal value of the LP-relaxation of F_2 is greater than or equal to $\min\{wt \mid (t, \delta) \in Q\}$. However, since Q is obtained from the LP-relaxation of F_2 by replacing the constraints (2.8) with the inequalities (2.10), the number of constraints in Q is of order n^2, whereas in the LP-relaxation of F_2 the number of constraints is of order n^3, which is a disadvantage of F_2.

Dyer and Wolsey [1990] compare different formulations for $1|r_j|\sum w_jC_j$. Among them is the following relaxation stated in terms of (t,δ)-variables:

$$\min \sum_{j=1}^{n} w_j t_j$$

subject to

$$\begin{aligned}
t_j &\geq \sum_{i:i\neq j} p_i\delta_{ij} && (j=1,\ldots,n),\\
\delta_{ij}+\delta_{ji} &= 1 && (i,j=1,\ldots,n, i<j),\\
t_j &\geq r_j && (j=1,\ldots,n),\\
\delta_{ij} &\geq 0 && (i,j=1,\ldots,n).
\end{aligned} \tag{2.11}$$

The above formulation still forms a relaxation of the original problem if the constraints $\delta_{ij}\in\{0,1\}$ are included. One of the reasons for this is that there are no constraints included to guarantee that the order in which the jobs are processed is acyclic. Furthermore, in the lower bounds on the starting time t_j of job j the release dates of the jobs that precede job j in the schedule are not taken into account. This may result in schedules in which different jobs overlap, which is demonstrated by the following 3-job example. Let $p_1=p_2=p_3=2$ and $r_1=r_2=r_3=2$. It is easy to see that $\delta_{12}=\delta_{23}=\delta_{31}=1$ and $t_1=t_2=t_3=2$ is feasible in the above formulation.

Nemhauser and Savelsbergh [1992] investigate the (t,δ)-formulation for the problem $1|r_j|\sum w_jC_j$. They do take release dates of preceding jobs into account by replacing constraints (2.11) in the above formulation by the following constraints:

$$\begin{aligned}
t_j \;\geq\; & r_i\delta_{ij} + \sum_{k<i,k\neq j} p_k(\delta_{ik}+\delta_{kj}-1) + \\
& \sum_{k\geq i,k\neq j} p_k\delta_{kj} \qquad (i,j=1,\ldots,n),
\end{aligned} \tag{2.12}$$

where the jobs are numbered such that $r_1\leq r_2\leq\ldots\leq r_n$. To limit the number of constraints, they do not include the constraints (2.8), but generate these inequalities as cutting planes. They identify five classes of valid inequalities and derive separation heuristics for these classes. These are embedded in a branch-and-cut algorithm. In their computational experiments the integrality gap, i.e., the difference between the value of the optimal solution and the solution to the initial linear programming relaxation, is less than 4 percent for 20-job problems and less than 3 percent for 30-job problems.

Peters [1988] studies these formulations for $1|\bar{d}_j|\ w_jC_j$, i.e., he incorporates deadlines in the formulation. He derives some classes of valid inequalities and shows

how to use dominance rules to fix some of the δ-variables. These results are used in the implementation of a branch-and-cut algorithm.

The formulations discussed in this section can be considered as extensions of the formulation discussed in the previous section. The above results indicate that the extended formulations are stronger than the formulation based on completion times; they contain more variables though. The above results further suggest that in this formulation fewer inequalities are needed to obtain a tight lower bound. Consider for example the case in which no release dates, deadlines, or precedence relations have been specified. The set of valid inequalities in terms of this formulation that provides a complete description of the convex hull of the set of feasible schedules is of polynomial size; in the formulation based on completion times an exponential number of inequalities is needed. The modeling of the objective functions proceeds in the same way as in case of a formulation based on completion times.

Observe that, if we restrict ourselves to the variables δ_{ij}, then a feasible solution to the scheduling problem defines a linear ordering. Therefore, valid inequalities for the linear ordering problem (see for example Grötschel, Jünger, and Reinelt [1985]) are also valid for the (t, δ)-formulation, and hence may also be used in a branch-and-cut algorithm based on this formulation. Müller [1994] studies the interval ordering polytope, which is an extension of the linear ordering polytope. He shows that the application of some known results on the interval ordering polytope to the linear ordering polytope may help to obtain better lower bounds for single-machine scheduling problems.

2.4 Time-indexed formulations

In this section, we introduce the *time-indexed formulation* that will be investigated in this thesis. This 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 all jobs have to be completed by time T. The formulation uses variables x_{jt}, where $x_{jt} = 1$ signals that job j is started in period t; $x_{jt} = 0$ otherwise. It is given by:

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

subject to

$$\sum_{t=1}^{T-p_j+1} x_{jt} = 1 \qquad (j = 1, \ldots, n), \tag{2.13}$$

$$\sum_{j=1}^{n} \sum_{s=t-p_j+1}^{t} x_{js} \leq 1 \quad (t = 1, \ldots, T), \tag{2.14}$$

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

Constraints (2.13) state that each job has to be processed exactly once; constraints (2.14) state that the machine handles at most one job during any time period.

Other variants of time-indexed formulations have been considered by Sousa [1989] and Dyer and Wolsey [1990]. They consider a formulation in terms of variables y_{jt}, where $y_{jt} = 1$ signals that job j is processed during period t; $y_{jt} = 0$ otherwise. Sousa also considers a formulation in terms of variables z_{jt}, where $z_{jt} = 1$ signals that job j has been started in or before period t; $z_{jt} = 0$ otherwise. He shows that these formulations are equivalent to the above formulation in the sense that the associated polyhedra have the same facial structure.

Sousa and Wolsey [1992] introduce three classes of valid inequalities for the time-indexed formulation in terms of the variables x_{jt}. The first class consists of inequalities with right-hand side 1, whereas the second and third class consist of inequalities with right-hand side $k \in \{2, \ldots, n\}$. They show that the inequalities in the first class are facet inducing. Their computational experiments reveal that the bounds obtained from the time-indexed formulation are strong compared to the bounds obtained from other mixed integer programming formulations.

Crama and Spieksma [1993] investigate the formulation in terms of variables x_{jt} for the special case of equal processing times. They exhibit a class of objective functions for which the optimal solution to the LP-relaxation is integral. Furthermore, they completely characterize all facet inducing inequalities with right-hand side 1 and present two other classes of facet inducing inequalities with right-hand side $k \in \{2, \ldots, n\}$. These facet inducing inequalities are embedded in a simple cutting plane algorithm. They tested this algorithm on problems with randomly generated cost coefficients c_{jt} and on the problem $1|p_j = p, r_j, \bar{d}_j| \sum w_j C_j$, where the release dates and deadlines may be violated at high cost. Their computational experiments indicate that all classes of inequalities effectively improve the lower bound provided by the LP-relaxation and that the cutting plane algorithm often finds an integral solution.

As mentioned in the previous section, Dyer and Wolsey [1990] compare different formulations for the problem $1|r_j| \sum w_j C_j$. They show that $Z_\delta \leq Z_Y \leq Z_X$, where Z_X and Z_Y are the optimal values of the LP-relaxations of the formulations in terms of the variables x_{jt} and y_{jt} respectively, and Z_δ is the optimal value of the relaxation stated in terms of the (t, δ)-variables that was presented in the previous section.

As the latter relaxation may be quite weak, the above result does not necessarily imply that the time-indexed formulation yields stronger lower bounds than the

formulations discussed in the previous two sections. Computational experiments, however, indicate that this formulation does yield strong lower bounds.

A disadvantage of the time-indexed formulation is that it contains a large number of constraints and a large number of variables. There are $n+T$ constraints and approximately nT variables. As T has to be at least $\sum_{j=1}^{n} p_j$, the formulation is pseudo-polynomial, since its size also depends on the processing times, whereas the formulations in the previous sections are polynomial, since their sizes only depend on the number of jobs. Therefore, to handle instances with a large number of jobs or large processing times we have to consider solution techniques specifically designed for large linear programs.

An advantage of this formulation is that it can be used to model many variants of the nonpreemptive single-machine scheduling problem. We can model all objective functions of the form $\sum_{j=1}^{n} f_j(t_j)$ by just setting $c_{jt} = f_j(t-1)$, whereas in the other formulations we can only deal with cost functions $f_j(t_j)$ that are linear. If the objective is to minimize the weighted sum of the tardinesses, then we set $c_{jt} = w_j(t+p_j-d_j-1)$ for $t+p_j-1 > d_j$ and $c_{jt} = 0$ for $t+p_j-1 \leq d_j$, and we do not have to resort to introducing new variables and constraints. If we want to minimize the makespan, then we have to add an extra variable and extra constraints, however. Release dates and deadlines are incorporated in the model by restricting the set of variables. If there are release dates r_j, then we discard the variables x_{jt} for $t = 1, \ldots, r_j$; this means that these variables are implicitly fixed at 0. Deadlines are incorporated in a similar way. Therefore, if release dates or deadlines are included, then the convex hull of the set of feasible solutions is a face of the original convex hull. This means that the set of facet inducing inequalities is contained in the set of facet inducing inequalities for the original problem. To include precedence constraints, we have to add extra constraints. This implies that it is more complicated to incorporate precedence constraints in this formulation than in the previous formulations.

2.5 A formulation based on positional start times and position determining variables

In this section, we discuss the formulation proposed by Lasserre and Queyranne [1992], which is based on *positional* start times and *position determining* variables. This model is inspired by a control theoretic view of scheduling decisions. In this view, a control is applied at discrete time instants specifying which job the machine will start processing. This formulation is based on variables t_k, where t_k denotes the start time of the kth job, and binary variables u_k^j indicating which job j is processed in the kth position. The set of feasible schedules is described by the following set of constraints:

$$\sum_{j=1}^{n} u_k^j = 1 \qquad (k = 1, \ldots, n), \tag{2.15}$$

$$\sum_{k=1}^{n} u_k^j = 1 \qquad (j = 1, \ldots, n), \tag{2.16}$$

$$t_k - t_{k-1} - \sum_{j=1}^{n} p_j u_{k-1}^j \geq 0 \qquad (k = 2, \ldots, n), \tag{2.17}$$

$$u_k^j \in \{0, 1\} \qquad (j, k = 1, \ldots, n),$$

where constraints (2.15) state that one job is chosen at each decision time, constraints (2.16) state that each job has to be processed exactly once, and constraints (2.17) state that no two jobs should overlap in their execution. Release dates r_j and deadlines $\bar{d}_j$ are included in the formulation by introducing the following constraints:

$$t_k - \sum_{j=1}^{n} r_j u_k^j \geq 0 \qquad (k = 1, \ldots, n), \tag{2.18}$$

$$-t_k + \sum_{j=1}^{n} (-p_j + \bar{d}_j) u_k^j \geq 0 \qquad (k = 1, \ldots, n). \tag{2.19}$$

Lasserre and Queyranne present a characterization of the set of feasible schedules in the subspace spanned by the variables u_k^j. For the problem $1||\sum w_j C_j$, they give complete descriptions of the projections of the convex hull of the set of feasible schedules into the subspace spanned by the positional start times, the subspace spanned by the positional completion times, and, for the special case of equal processing times, the subspace spanned by both these sets of variables.

In their computational experiments they compare this formulation and the time-indexed formulation on the basis of two problems. These problems are $1|r_j, \bar{d}_j|C_{\max}$, where $C_{\max}$ denotes the makespan, and $1|r_j|L_{\max}$, where $L_{\max}$ denotes the maximum lateness and the lateness L_j of job j is defined as the difference between its completion time and its due date d_j. In this formulation the makespan is given by $t_n + \sum_{j=1}^{n} u_n^j p_j$. The lateness of the kth job is given by $t_k + \sum_{j=1}^{n} u_k^j (p_j - d_j)$. We can model the problem of minimizing maximum lateness by choosing as objective the minimization of the variable L that denotes the maximum lateness and adding the constraints $L \geq t_k + \sum_{j=1}^{n} u_k^j (p_j - d_j)$ $(k = 1, \ldots, n)$. For all their test instances, they compute the lower bounds provided by the LP-relaxations of the above formulation and the time-indexed formulation. Their results indicate that the bounds are comparable and that for these problems no formulation is clearly superior to the other in terms of the quality of the LP-relaxation.

This formulation differs from the previous formulations in the sense that it allows for easy modeling of different types of problems. On the one hand, this formulation can also be used to formulate so-called *generic* scheduling problems. An example of such a problem is the positional due date problem studied by Hall [1986] and Hall, Sethi, and Sriskandarajah [1991]. This problem has the special feature that a number of deadlines has been specified at which a given number of jobs must be completed; only the number, and not the identity, of the completed jobs is important. As we have seen, the makespan given by $t_n + \sum_{j=1}^{n} p_j u_n^j$ is a linear function in this formulation, whereas it is nonlinear in the formulations in the previous sections. On the other hand, the start time of job j is given by $\sum_{k=1}^{n} u_k^j t_k$, which is nonlinear. Therefore, the weighted sum of the start times does not seem to admit a simple linear formulation in this model. We have seen that release dates and deadlines can be included in this formulation by the addition of extra constraints. To include precedence constraints we have to add $(n-1)$ extra constraints for each precedence relation: if job i has to be processed before job j, then this can be modeled by adding the constraints

$$u_k^j \leq \sum_{l=1}^{k-1} u_l^i \qquad (k = 2, \ldots, n).$$

The number of variables in this formulation is of the order n^2. Hence, it contains more variables than the formulation in terms of completion times and fewer variables than the time-indexed formulation. The formulation in terms of start times and sequence determining variables also contains approximately n^2 variables. For the test problems of Lasserre and Queyranne the strength of the LP-relaxation is comparable with the time-indexed formulation, which suggests that for these problems it is more beneficial to use this formulation; objective functions that are linear in the start or completion times of the jobs are however nonlinear in this formulation.

Chapter 3

Facets and characterizations

3.1 Introduction

We present new results for the time-indexed formulation of nonpreemptive single-machine scheduling problems studied by Sousa and Wolsey [1992]; see Section 2.4 for a brief description. Sousa and Wolsey introduce three classes of inequalities. The first class consists of inequalities with right-hand side 1; the second and third class of inequalities with right-hand side $k \in \{2, \ldots, n\}$. In their cutting plane algorithm, they use an exact separation method only for inequalities with right-hand side 1 and for inequalities with right-hand side 2 in the second class; they use a simple heuristic to identify violated inequalities in the third class. Their computational experiments revealed that the bounds obtained are strong compared to bounds obtained from other mixed integer programming formulations.

These promising computational results stimulated us to study the inequalities with right-hand side 1 or 2 more thoroughly. This has resulted in complete characterizations of all facet inducing inequalities with integral coefficients and right-hand side 1 or 2. Since only some of the classes of inequalities used in the computational experiments by Sousa and Wolsey are facet inducing, we expect that our results can be used to improve their cutting plane algorithm for single-machine scheduling problems. The development and implementation of a branch-and-cut algorithm based on the identified classes of inequalities will be discussed in Chapter 4 together with some computational results.

Recall that the time-indexed formulation studied by Sousa and Wolsey [1992] 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 all jobs have to be completed by time T. The formulation is as follows:

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

subject to

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

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

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

where $x_{jt} = 1$ if job j is started in period t and $x_{jt} = 0$ otherwise. This formulation can be used to model several single-machine scheduling problems by an appropriate choice of the objective coefficients and possibly a restriction of the set of variables. For instance, if the objective is to minimize the weighted sum of the starting times, then we choose coefficients $c_{jt} = w_j(t-1)$, where w_j denotes the weight of job j; if there are release dates r_j, then we discard the variables x_{jt} for $t = 1, \ldots, r_j$.

This chapter is organized as follows. In Section 3.2, we show that in some special cases the LP-relaxation of the time-indexed formulation solves the problem. These are the problems of minimizing the sum of the completion times and the weighted sum of the completion times. For the first one, we prove that the LP-relaxation always has an integral solution. For the second one, the LP-relaxation provides the optimal value of the problem; its optimal solution may be fractional, though. In Section 3.3, we present some basic concepts concerning the polyhedral structure of the time-indexed formulation. In Sections 3.4 and 3.5, we give complete characterizations of all facet inducing inequalities with right-hand side 1 and 2. We conclude this chapter with a brief discussion of related research in Section 3.6.

3.2 Special cases

3.2.1 Minimizing $\sum C_j$

We consider the problem of minimizing the sum of the completion times. This problem is solved by scheduling the jobs in order of nondecreasing processing times without idle time inserted; it is modeled by choosing $c_{jt} = t+p_j-1$ $(j = 1, \ldots, n, t = 1, \ldots, T - p_j + 1)$. We need the following preliminary lemma.

Lemma 3.1 *Suppose that nm jobs have to be scheduled on m parallel identical machines in such a way that $\sum_{j=1}^{nm} C_j$ is minimal. Then for any optimal schedule we have the following:*
(a) *Each machine processes n jobs.*
(b) *For each pair of jobs i and i', if job i is succeeded by fewer jobs on its machine than job i', then $p_i \geq p_{i'}$.*

Proof. (a) The main idea behind the proof is that if jobs $i_1, i_2, \ldots, i_k$ are scheduled on a machine in this order, then the sum of the completion times equals $(p_{i_1}) + (p_{i_1} + p_{i_2}) + \ldots + (p_{i_1} + p_{i_2} + \ldots + p_{i_k}) = \sum_{l=1}^{k}(k+1-l)p_{i_l}$, i.e., the processing time of the last job is counted once, that of the last but one job is counted twice, and so on. Suppose that there is a machine that contains more than n jobs. Then there clearly also is a machine processing less than n jobs. It follows from the above observation that the sum of the completion times is decreased by moving the job on the first position on the overloaded machine to the first position on the underloaded machine.

(b) Since job i has fewer successors on its machine than job i', p_i is counted not as often as $p_{i'}$ in $\sum_{j=1}^{nm} C_j$. To guarantee optimality, we need $p_i \geq p_{i'}$. □

Theorem 3.1 *If the objective function is the sum of the completion times, then the set of optimal solutions to the LP-relaxation of the time-indexed formulation is a face of the convex hull of the set of integral solutions.*

Proof. Consider any fractional optimal solution x^* of the LP-relaxation of the time-indexed formulation. Let L be the least common multiple of the denominators of the nonzero values in x^*.

We start with showing that x^* corresponds to a schedule on L machines that also minimizes the sum of the completion times. We divide the machine into L parallel pseudo-machines of capacity $\frac{1}{L}$ each, and each job j into L operations $O_{j1}, \ldots, O_{jL}$. Operation O_{jk} has processing time p_j and must be processed on one pseudo-machine without preemption. Operations belonging to the same job are allowed to be processed simultaneously. Each pseudo-machine can process one job at a time. We say that, if $x^*_{jt} = \frac{l}{L}$, then l operations belonging to job j are started in period t on l different pseudo-machines. In this way, x^* corresponds to a schedule S of the nL operations on the L parallel pseudo-machines. For example, if $n = 2, p_1 = 3, p_2 = 2$, and $x^*_{11} = \frac{1}{2}, x^*_{13} = \frac{1}{2}, x^*_{21} = \frac{1}{2}, x^*_{24} = \frac{1}{2}$, then $L = 2$ and S is the schedule depicted in Figure 3.1.

It is easy to see that the sum of the completion times of the operations O_{jk} in the schedule S equals $L \times \sum_{j=1}^{n} \sum_{t=1}^{T-p_j+1}(t + p_j - 1)x_{jt}$. Hence, the schedule S minimizes the sum of the completion times of the operations O_{jk}.

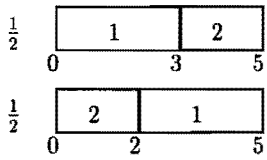


Figure 3.1: A schedule on two pseudo-machines.

We proceed by showing that x^* is a convex combination of integral solutions, which implies that x^* is in a face of the convex hul of the set of integral solutions. It is easy to see that this implies the theorem. Reindex the jobs such that $p_1 \leq p_2 \leq \ldots \leq p_n$. Consider the schedule S constructed above. By part (a) of Lemma 3.1, the schedule S is such that n operations are processed on each pseudo-machine. If $p_1 < p_2$, then part (b) of Lemma 3.1 states that all operations belonging to job 1 are in the first position on some pseudo-machine; all these operations are hence started in the first period, implying that $x^*_{11} = 1$. If, on the other hand, $p_1 = p_2 = \ldots = p_k < p_{k+1}$ for some $k > 1$, then we derive from part (b) of Lemma 3.1 that the operations corresponding to the jobs $1, 2, \ldots, k$ occupy the first k positions on each one of the pseudo-machines, which implies that all these operations are started in the periods $1, 1 + p_1, \ldots, 1 + (k-1)p_1$. For these k jobs, the scheduling problem boils down to assigning to these jobs a start time in one of these k periods. Hence, the vector $(x^*_{11}, x^*_{1,1+p_1}, \ldots, x^*_{1,1+(k-1)p_1}, x^*_{21}, \ldots, x^*_{2,1+(k-1)p_1}, \ldots x^*_{k,1+(k-1)p_1})$ is a fractional solution of this assignment problem. Since the constraint matrix of the assignment problem is totally unimodular, this vector can be written as a convex combination of integral solutions. Proceeding in this way, we find that x^* is a convex combination of integral solutions. □

Corollary 3.1 *If the objective function is the sum of the completion times, then the optimal solution to the LP-relaxation of the time-indexed formulation that is identified by the simplex method is integral.* □

3.2.2 Minimizing $\sum w_j C_j$

We now consider the problem of minimizing the weighted sum of the completion times. This problem is modeled by choosing coefficients $c_{jt} = w_j(t + p_j - 1)$. We assume that $w_j > 0$ for all $j \in \{1, \ldots, n\}$ and that the jobs are indexed such that $\frac{w_1}{p_1} \geq \frac{w_2}{p_2} \geq \ldots \geq \frac{w_n}{p_n}$. Smith [1956] showed that this problem is solved by scheduling the jobs in order of nonincreasing ratios w_j/p_j without idle time before the start time of any job. Because of the way the jobs are indexed, $1, 2, \ldots, n$ is an optimal sequence; this implies that job j is started in period $\tau_j = \sum_{i=1}^{j-1} p_i + 1$. For notational

convenience, we define $w_{-1} = +\infty$ and $w_{n+1} = 0$. Let Z_{LP} and Z_{IP} be the optimal value of the LP-relaxation and the integer programming problem, respectively.

Theorem 3.2 *If the objective function is* $\sum_{j=1}^{n} w_j C_j$, *then*
(1) $Z_{LP} = Z_{IP}$,
(2) *if* $\frac{w_{i-1}}{p_{i-1}} > \frac{w_i}{p_i} = \ldots = \frac{w_k}{p_k} > \frac{w_{k+1}}{p_{k+1}}$ *with* $i \leq k$, *then in any optimal solution to the LP-relaxation the jobs* $i, i+1, \ldots, k$ *are entirely processed during the periods* $\tau_i, \tau_i + 1, \ldots, \tau_{k+1} - 1$, *i.e., if* $x_{jt} > 0$ *with* $j \in \{i, i+1, \ldots, k\}$, *then* $\tau_i \leq t \leq \tau_{k+1} - p_j$.

Proof. (1) Observe that, by Smith's rule, the vector x^*, which is defined by

$$x^*_{jt} = \begin{cases} 1 & \text{if } t = \tau_j, \\ 0 & \text{otherwise,} \end{cases}$$

is an optimal solution to the integer programming problem and that the optimal value Z_{IP} is equal to $\sum_{j=1}^{n} w_j(\tau_j + p_j - 1)$. We will now show that x^* is also an optimal solution to the LP-relaxation by defining a feasible solution (u^*, v^*) to the dual of the LP-relaxation such that x^* and (u^*, v^*) satisfy the complementary slackness relations.

The dual D of the LP-relaxation is given by

$$\max \sum_{j=1}^{n} u_j + \sum_{t=1}^{T} v_t$$

subject to

$$u_j + \sum_{s=t}^{t+p_j-1} v_s \leq (t + p_j - 1)w_j \qquad (j = 1, ..., n, t = 1, ..., T - p_j + 1),$$

$$v_t \leq 0 \qquad (t = 1, \ldots, T).$$

We define (u^*, v^*) by

$$v^*_t = \begin{cases} -\sum_{i=j}^{n} w_i + k\frac{w_j}{p_j} & \text{if } t = \tau_j + k, \text{ with } j \in \{1, \ldots, n\}, \\ & k \in \{0, 1, \ldots, p_j\}, \\ 0 & \text{if } t \geq \sum_{j=1}^{n} p_j + 1, \end{cases}$$

$$u^*_j = (\tau_j + p_j - 1)w_j - \sum_{s=\tau_j}^{\tau_j+p_j-1} v^*_s \qquad (j = 1, 2, \ldots, n).$$

Note that $-v_t^*$ equals the total weight of the work that still has to be done at the beginning of period t if the jobs are processed in the order $1, 2, \ldots, n$. Note further that v_t^* is defined twice if $t = \tau_j$ and that both values coincide.

We first show that x^* and (u^*, v^*) satisfy the complementary slackness relations, i.e., for all t with $t = 1, \ldots, T$ we have

$$v_t^*(\sum_{j=1}^{n} \sum_{s=t-p_j+1}^{t} x_{js}^* - 1) = 0$$

and for all j, t with $j = 1, \ldots, n$, $t = 1, \ldots, T - p_j + 1$ we have

$$x_{jt}^*((t + p_j - 1)w_j - u_j^* - \sum_{s=t}^{t+p_j-1} v_s^*) = 0.$$

The second relation trivially holds, since x_{jt}^* is positive only if $t = \tau_j$, but then $(t + p_j - 1)w_j - u_j^* - \sum_{s=t}^{t+p_j-1} v_s^* = 0$ by definition. As to the first relation, note that, since x^* corresponds to a schedule without idle time, $\sum_{j=1}^{n} \sum_{s=t-p_j+1}^{t} x_{js}^* = 1$ for $1 \leq t \leq \sum_{j=1}^{n} p_j$. As $v_t^* = 0$ for all $t \geq \sum_{j=1}^{n} p_j + 1$ by definition, we have that the first relation holds.

What is left to show is that (u^*, v^*) is feasible. Clearly, $v_t^* \leq 0$ for all t. We have to show that for all j and for all t

$$V_j(t) \leq l_j(t),$$

where $V_j(t)$ and $l_j(t)$ are defined as $V_j(t) = \sum_{s=t}^{t+p_j-1} v_s^*$ and $l_j(t) = (t+p_j-1)w_j - u_j^*$. We prove that the above inequality holds by showing below that for each j the piecewise-linear function $\tilde{V}_j(t)$ connecting all values $V_j(t)$ is a concave function that coincides from time $t = \tau_j$ until time $t = \tau_j + 1$ with the line $\tilde{l}_j(t)$ connecting all values $l_j(t)$.

Consider any job j. Observe that for all t with $\tau_j \leq t < \tau_j + p_j = \tau_{j+1}$, we have $v_{t+1}^* - v_t^* = \frac{w_j}{p_j}$, and that for all t with $t \geq \tau_{n+1} = \sum_{j=1}^{n} p_j + 1$, we have $v_{t+1}^* - v_t^* = 0$. Since $\frac{w_1}{p_1} \geq \frac{w_2}{p_2} \geq \ldots \geq \frac{w_n}{p_n} > 0$, we have that the piece-wise linear function connecting the values v_t^* is concave. Hence, $\tilde{V}_j(t)$ is a concave function as well, since it is the sum of a finite number of concave functions. The line $\tilde{l}_j(t)$ has slope w_j and, by definition, $V_j(t) = l_j(t)$ for $t = \tau_j$. As to the slope of $\tilde{V}_j(t)$ between $t = \tau_j$ and $t = \tau_j + 1$, we have

$$\begin{aligned} V_j(\tau_j + 1) - V_j(\tau_j) &= v_{\tau_j+p_j}^* - v_{\tau_j}^* \\ &= (v_{\tau_j+p_j}^* - v_{\tau_j+p_j-1}^*) + (v_{\tau_j+p_j-1}^* - v_{\tau_j+p_j-2}^*) + \ldots + \\ &\quad (v_{\tau_j+1}^* - v_{\tau_j}^*) = p_j \frac{w_j}{p_j} = w_j, \end{aligned}$$

implying that $\tilde{l}_j(t)$ and $\tilde{V}_j(t)$ coincide from time τ_j until time $\tau_j + 1$.

We conclude that (u^*, v^*) is an optimal solution to D that satisfies the complementary slackness relations with x^*, which is an optimal solution to the integer programming problem and hence also to the LP-relaxation.

(2) Let x denote any optimal solution to the LP-relaxation. We know from linear programming theory that, since (u^*, v^*) is an optimal solution to D, x and (u^*, v^*) must satisfy the complementary slackness relations. Let $i \leq k$ be such that $\frac{w_{i-1}}{p_{i-1}} > \frac{w_i}{p_i} = \ldots = \frac{w_k}{p_k} > \frac{w_{k+1}}{p_{k+1}}$. By definition, we have $v^*_{\tau_i} - v^*_{\tau_i - 1} = \frac{w_{i-1}}{p_{i-1}}$, $v^*_{t+1} - v^*_t = \frac{w_i}{p_i}$ for all t with $\tau_i \leq t < \tau_{k+1}$, and $v^*_{\tau_{k+1}+1} - v^*_{\tau_{k+1}} = \frac{w_{k+1}}{p_{k+1}}$. Let j be any job from the job set $\{i, i+1, \ldots, k\}$. Since x and (u^*, v^*) satisfy the complementary slackness relations, it follows that if $x_{jt} > 0$, then $V_j(t) = l_j(t)$. By definition, $V_j(t) = l_j(t)$ for $t = \tau_j$. For all t with $\tau_i \leq t \leq \tau_{k+1} - p_j$, we have

$$V_j(t+1) - V_j(t) = v^*_{t+p_j} - v^*_t = p_j \frac{w_i}{p_i} = p_j \frac{w_j}{p_j} = w_j.$$

Hence, the points $V_j(t)$ $(\tau_i \leq t \leq \tau_{k+1} - p_j + 1)$ lie on a line with slope w_j, which implies that for all t with $\tau_i \leq t \leq \tau_{k+1} - p_j + 1$ we have

$$V_j(t) = l_j(t).$$

Furthermore,

$$\begin{aligned} V_j(\tau_i) - V_j(\tau_i - 1) &= v^*_{\tau_i + p_j - 1} - v^*_{\tau_i - 1} = (v^*_{\tau_i + p_j - 1} - v^*_{\tau_i}) + (v^*_{\tau_i} - v^*_{\tau_i - 1}) \\ &= (p_j - 1)\frac{w_i}{p_i} + \frac{w_{i-1}}{p_{i-1}} = (p_j - 1)\frac{w_j}{p_j} + \frac{w_{i-1}}{p_{i-1}} > w_j, \end{aligned}$$

and

$$V_j(\tau_{k+1} - p_j + 2) - V_j(\tau_{k+1} - p_j + 1) =$$

$$v^*_{\tau_{k+1}+1} - v^*_{\tau_{k+1} - p_j + 1} = (v^*_{\tau_{k+1}+1} - v^*_{\tau_{k+1}}) + (v^*_{\tau_{k+1}} - v^*_{\tau_{k+1} - p_j + 1}) =$$

$$\frac{w_{k+1}}{p_{k+1}} + (p_j - 1)\frac{w_i}{p_i} = \frac{w_{k+1}}{p_{k+1}} + (p_j - 1)\frac{w_j}{p_j} < w_j.$$

Since $\tilde{V}_j(t)$ is concave, it follows that for all t with $t < \tau_i$ or $t > \tau_{k+1} - p_j + 1$ we have

$$V_j(t) < l_j(t).$$

We conclude that, if $x_{jt} > 0$, then $\tau_i \leq t \leq \tau_{k+1} - p_j + 1$. Note that if $i = k = j$, i.e., the value of the ratio $\frac{w_j}{p_j}$ is unique, then we find that x_{jt} can only be positive for $t = \tau_j$ or $t = \tau_j + 1$.

We now show that $x_{jt} = 0$ for $t = \tau_{k+1} - p_j + 1$, which implies that in case the value of the ratio $\frac{w_j}{p_j}$ is unique, x_{jt} can only be positive for $t = \tau_j$.

Since x and (u^*, v^*) satisfy the complementary slackness relations, we have that $v_t^*(1 - \sum_{s=t-p_j+1}^{t} x_{js}) = 0$ for all t. Since $v_t^* < 0$ for all t with $1 \leq t \leq \sum_{j=1}^{n} p_j$, it must be that $\sum_{s=t-p_j+1}^{t} x_{js} = 1$ for all t with $1 \leq t \leq \sum_{j=1}^{n} p_j$. Hence, although x may be fractional, the machine has a workload equal to 1 in the time periods 1 until $\sum_{j=1}^{n} p_j$. Now, suppose that $\frac{w_1}{p_1} = \frac{w_2}{p_2} = \ldots = \frac{w_k}{p_k} > \frac{w_{k+1}}{p_{k+1}}$. The machine has workload 1 during the time periods 1 until $p_1 + \ldots + p_k = \tau_{k+1} - 1$. Furthermore, from the above it follows that the jobs $j \in \{1, 2, \ldots, k\}$ are the only jobs that may have $x_{jt} > 0$ for some t with $1 \leq t \leq \tau_{k+1} - 1$, i.e., the only jobs that may be started in time periods $1, \ldots, \tau_{k+1} - 1$. Hence, the jobs $1, 2, \ldots, k$ have been completed at the end of time period $\tau_{k+1} - 1$. Proceeding in this way we find that (2) holds. □

Corollary 3.2 *If the objective function is* $\sum_{j=1}^{n} w_j C_j$, *then*
(1) *if* $\frac{w_{j-1}}{p_{j-1}} > \frac{w_j}{p_j} > \frac{w_{j+1}}{p_{j+1}}$ *for some* $j \in \{1, 2, \ldots, n\}$, *then* $x_{j\tau_j} = 1$ *in any optimal solution to the LP-relaxation,*
(2) *if* $\frac{w_1}{p_1} > \frac{w_2}{p_2} > \ldots > \frac{w_n}{p_n}$, *then the optimal solution to the LP-relaxation is integral and unique.* □

Remark. Theorem 3.2 also holds if we do not restrict ourselves to positive weights w_j. Let l and m be such that $w_j > 0$ for $j \in \{1, 2, \ldots, l\}$, $w_j = 0$ for $j \in \{l + 1, \ldots, m - 1\}$, and $w_j < 0$ for $j \in \{m, \ldots, n\}$. We define $\tau_j = \sum_{i<j} p_i + 1$ for $j \in \{1, 2, \ldots, m - 1\}$ and $\tau_j = T - \sum_{i \geq j} p_i + 1$ for $j \in \{m, \ldots, n\}$. The proof proceeds along the same lines if we define v^* by

$$v_t^* = \begin{cases} -\sum_{i=j}^{l} w_i + k\frac{w_j}{p_j} & \text{if } t = \tau_j + k \text{ with } j \in \{1, 2, \ldots, l\}, \\ & k \in \{0, 1, \ldots, p_j - 1\}, \\ 0 & \text{if } \tau_{l+1} \leq t < \tau_m, \\ \sum_{i=m}^{j-1} w_i + k\frac{w_j}{p_j} & \text{if } t = \tau_j + k \text{ with } j \in \{m, \ldots, n\}, \\ & k \in \{0, 1, \ldots, p_j - 1\}. \end{cases}$$

In case of arbitrary weights, the first part of Corollary 3.2 still holds for all j with $w_j \neq 0$, but the second part of Corollary 3.2 has to be reformulated as follows: If $\frac{w_1}{p_1} > \frac{w_2}{p_2} > \ldots > \frac{w_n}{p_n}$, then the optimal solution to the LP-relaxation is integral and unique with the exception of the variables corresponding to a job with $w_j = 0$.

From the above, one might get the impression that the optimal value of the LP-relaxation is always optimal for any scheduling problem that can be modeled by this formulation and that is solvable in polynomial time. The following instance of

the problem $1|\bar{d}_j|\sum C_j$ shows that this is not the case. Smith [1956] showed that this problem is solved in $O(n \log n)$ time by the following backward scheduling rule: schedule in the last position the job with the largest processing time that is allowed to be completed there. Consider the following 3-job example with $T = 20$, $p_1 = 1, p_2 = 3$, and $p_3 = 10$, and $\bar{d}_1 = \bar{d}_2 = 20$ and $\bar{d}_3 = 13$. By the above rule, the optimal integral solution is $x_{11} = x_{2,12} = x_{32} = 1$. For this schedule we have $\sum_{j=1}^{n} C_j = \sum_{j=1}^{n} \sum_{t=1}^{T-p_j+1} (t + p_j - 1)x_{jt} = 26$. However, $x_{11} = x_{12} = x_{13} = \frac{1}{3}$, $x_{21} = \frac{1}{3}$, $x_{2,14} = \frac{2}{3}$, and $x_{34} = 1$ is a feasible solution to the LP-relaxation and has $\sum_{j=1}^{n} \sum_{t=1}^{T-p_j+1} (t + p_j - 1)x_{jt} = 22\frac{1}{3}$. This shows that the optimal value of the LP-relaxation can be strictly smaller than the optimal value of the original problem.

3.3 Basic properties of the polyhedral structure

In the time-indexed formulation, the convex hull of the set of feasible schedules is not full-dimensional. As it is often easier to study full-dimensional polyhedra, we consider the polyhedron P that is associated with an extended set of solutions and defined by

$$\sum_{t=1}^{T-p_j+1} x_{jt} \leq 1 \qquad (j = 1, \ldots, n), \tag{3.1}$$

$$\sum_{j=1}^{n} \sum_{s=t-p_j+1}^{t} x_{js} \leq 1 \qquad (t = 1, \ldots, T), \tag{3.2}$$

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

In this relaxation we no longer demand that each job has to be started. Although it may seem more natural to relax the equations into inequalities with sense greater-than-or-equal instead of less-than-or-equal, we have chosen for the latter option, since it has the advantage that the origin and the unit vectors are elements of the polyhedron, which is often convenient for dealing with affine independence. For instance, it is not hard to show that the inequalities $x_{js} \geq 0$ are facet inducing. Note that the collection of facet inducing inequalities for the polyhedron associated with the extended set of solutions includes the collection of facet inducing inequalities for the polyhedron associated with the original set of solutions. In this chapter, we say that any feasible solution to the above formulation is a feasible schedule, even though not all jobs have to be processed.

Before we present our analysis of the structure of facet inducing inequalities with right-hand side 1 or 2, we introduce some notation and definitions.

The index-set 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 $l = \min\{l_j|j \in \{1,\ldots,n\}\}$ and $u = \max\{u_j|j \in \{1,\ldots,n\}\}$.

We define an interval $[a,b]$ as the set of periods $\{a+1, a+2, \ldots, b\}$, i.e., the set of periods between time a and time b. If $a \geq b$, then $[a,b] = \emptyset$.

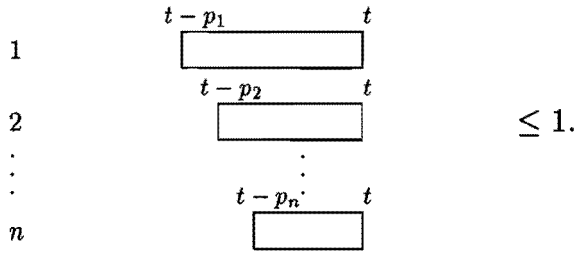
Lemma 3.2 *A facet inducing inequality $ax \leq b$ with integral coefficients a_{js} and integral right-hand side has either $b \geq 0$ and coefficients a_{js} in $\{0,1,\ldots,b\}$ or is a positive scalar multiple of $-x_{js} \leq 0$ for some (j,s) with $1 \leq s \leq T - p_j + 1$.*

Proof. Since the origin is an element of the polyhedron, any valid inequality $ax \leq b$ must have a nonnegative right-hand side. Since the schedule x defined by $x_{js} = 1$ and $x_{j's'} = 0$ for all $(j',s') \neq (j,s)$ is feasible for every (j,s), a valid inequality $ax \leq b$ does not have coefficients greater than b. We now show that a facet inducing inequality does not have negative coefficients, unless it is a positive scalar multiple of $-x_{js} \leq 0$. Let $ax \leq b$ be a valid inequality with $a_{js} < 0$. Let x be any feasible schedule with $x_{js} = 1$. If $x_{js} = 1$ is replaced by $x_{js} = 0$, then the schedule remains feasible and ax is increased. Therefore, any feasible schedule such that $ax = b$ satisfies $x_{js} = 0$. Since P is full-dimensional, it follows that the inequality $ax \leq b$ cannot be facet inducing, unless $ax = b$ is equivalent to $x_{js} = 0$, i.e., $ax \leq b$ is a positive scalar multiple of $-x_{js} \leq 0$. □

Observe that this lemma implies that the inequalities $x_{js} \geq 0$ are the only facet inducing inequalities with right-hand side 0. For presentational convenience, we use $x(S)$ to denote $\sum_{(j,s)\in S} x_{js}$. As a consequence of the previous lemma, valid inequalities with right-hand side 1 will be denoted by $x(V) \leq 1$ and valid inequalities with right-hand side 2 will be denoted by $x(V^1) + 2x(V^2) \leq 2$, where $V = V^1 \cup V^2$ and $V^1 \cap V^2 = \emptyset$. Furthermore, we define $V_j^2 = \{s \mid (j,s) \in V^2\}$.

In the sequel, we shall often represent inequalities by diagrams. A diagram contains a line for each job. The blocks on the line associated with job j indicate

the time periods s for which x_{js} occurs in the inequality. For example, an inequality of the form (3.2) can be represented by the following diagram:



3.4 Facet inducing inequalities with right-hand side 1

The purpose of this section is twofold. First, we present new results that extend and complement the work of Sousa and Wolsey [1992]. Second, we familiarize the reader with our approach in deriving complete characterizations of classes of facet inducing inequalities.

Establishing complete characterizations of facet inducing inequalities proceeds in two phases. First, we derive necessary conditions in the form of various structural properties. Second, we show that these necessary conditions on the structure of facet inducing inequalities are also sufficient.

A valid inequality $x(V) \leq 1$ is called *maximal* if there does not exist a valid inequality $x(W) \leq 1$ with $V \subsetneq W$. The following lemma gives a general necessary condition and is frequently used in the proofs of structural properties.

Lemma 3.3 *A facet inducing inequality $x(V) \leq 1$ is maximal.* □

Property 3.1 *If $x(V) \leq 1$ is facet inducing, then the sets V_j are intervals, i.e., $V_j = [l_j - p_j, u_j]$.*

Proof. Let $j \in \{1, \ldots, n\}$ and assume $V_j \neq \emptyset$. By definition $l_j - p_j + 1$ is the smallest s such that $s \in V_j$ and u_j is the largest such value. Consider any s with $l_j - p_j + 1 < s < u_j$ and let job j be started in period s, i.e., $x_{js} = 1$.

Suppose $(i,t) \in V$ is such that $x_{it} = x_{js} = 1$ defines a feasible schedule. If $t < s$, i.e., job i is started before job j, then the schedule that we obtain by postponing the start of job j until period u_j is also feasible. This schedule does not satisfy $x(V) \leq 1$, which contradicts the validity of the inequality. Hence, no job can be started in V

before job j. Similarly, we obtain a contradiction if $t > s$, which implies that no job can be started in V after job j.

We conclude that choosing $x_{js} = 1$ prohibits any job from starting in V. Because of the maximality of $x(V) \leq 1$, we must have $(j, s) \in V$. □

Property 3.2 *Let $x(V) \leq 1$ be facet inducing.*
(a) *Assume $l = l_1 \leq l_2 = \min\{l_j | j \in \{2, \ldots, n\}\}$. Then $V_1 = [l - p_1, l_2]$ and $V_j = [l_j - p_j, l]$ for all $j \in \{2, \ldots, n\}$.*
(b) *Assume $u = u_1 \geq u_2 = \max\{u_j | j \in \{2, \ldots, n\}\}$. Then $V_1 = [u_2 - p_1, u]$ and $V_j = [u - p_j, u_j]$ for all $j \in \{2, \ldots, n\}$.*

Proof. (a) Let $x(V) \leq 1$ be facet inducing with $l = l_1 \leq l_2 = \min\{l_j \mid j \in \{2, \ldots, n\}\}$. Observe that Property 3.1 implies that V_1 is an interval and that by definition its lower bound equals $l - p_1$. We now show that the upper bound is equal to l_2. Since $x_{2,l_2-p_2+1} = 1$ and $x_{1s} = 1$ defines a feasible schedule for any $s > l_2$, we have that only one of these variables can occur in $x(V) \leq 1$; as by definition $(2, l_2 - p_2 + 1) \in V$, it follows that the upper bound of V_1 is at most l_2. Now, let $x_{1s} = 1$ for some $s \in [l - p_1, l_2]$. Reasoning as in the proof of Property 3.1 we can show that since $l - p_1 + 1 \in V_1$ it follows that no job can be started in V after job 1. As $s \leq l_2 = \min\{l_j | j \in \{2, \ldots, n\}\}$, it is impossible to start any job in V before job 1. From the maximality of $x(V) \leq 1$ we conclude that $V_1 = [l - p_1, l_2]$. Similar arguments can be applied to show that $V_j = [l_j - p_j, l]$ for all $j \in \{2, \ldots, n\}$.

The proof of (b) is similar to that of (a). □

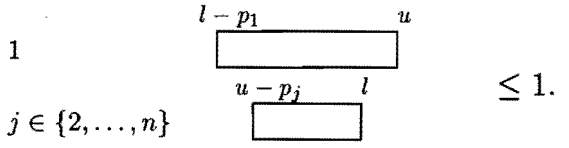
Observe that by Property 3.2(a) a facet inducing inequality $x(V) \leq 1$ with $l = l_1$ necessarily has $u_1 = u$. Consequently, Properties 3.2(a) and 3.2(b) can be combined to give the following theorem.

Theorem 3.3 *A facet inducing inequality $x(V) \leq 1$ has the following structure:*

$$\begin{aligned} V_1 &= [l - p_1, u], \\ V_j &= [u - p_j, l] \quad (j \in \{2, \ldots, n\}), \end{aligned} \tag{3.3}$$

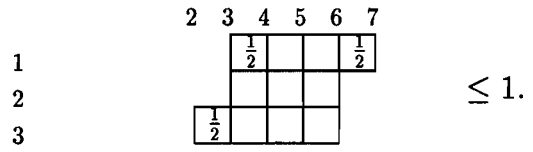
where $l = l_1 \leq u_1 = u$. □

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



Note that if $V_j = \emptyset$ for all $j \in \{2, \ldots, n\}$, $l = p_1$, and $u = T - p_1 + 1$, then the inequalities with structure (3.3) coincide with the inequalities (3.1); if $l = u$, then the inequalities with structure (3.3) coincide with the inequalities (3.2).

Example 3.1 Let $n = 3, p_1 = 3, p_2 = 4$, and $p_3 = 5$. The inequality with structure (3.3), $l = l_1 = 6$ and $u = u_1 = 7$ is given by the following diagram:



$\leq 1.$

Note that the fractional solution $x_{14} = x_{17} = x_{33} = \frac{1}{2}$ violates this inequality.

Sousa and Wolsey [1992] have shown that the given necessary conditions are also sufficient.

Theorem 3.4 *A valid inequality* $x(V) \leq 1$ *with structure* (3.3) *that is maximal is facet inducing.* □

Specific necessary and sufficient conditions for a valid inequality $x(V) \leq 1$ with structure (3.3) to be maximal are given by the following theorem. In the proof we use the following observation. If a valid inequality $x(V) \leq 1$ is maximal, then for any $(j, s) \notin V$ there must be a feasible schedule with $x_{js} = 1$ and $x(V) = 1$. We call such a schedule a *counterexample for* (j, s).

Theorem 3.5 *A valid inequality* $x(V) \leq 1$ *with structure* (3.3) *is maximal if and only if* $V_j \neq \emptyset$ *for some* $j \in \{2, \ldots, n\}$, *or* $x(V) \leq 1$ *is one of the inequalities* (3.1), *i.e.,* $l = p_1$ *and* $u = T - p_1 + 1$.

Proof. Let $x(V) \leq 1$ be a valid inequality with structure (3.3). Let $x(V) \leq 1$ be maximal and suppose that $V_j = \emptyset$ for all $j \in \{2, \ldots, n\}$. It is not hard to see that $x(V) \leq 1$ must be one of the inequalities (3.1), and hence $l = p_1$ and $u = T - p_1 + 1$. On the other hand, it is easy to see that if $V_j \neq \emptyset$ for some $j \in \{2, \ldots, n\}$, then there is a counterexample for any $(j, s) \notin V$. Hence $x(V) \leq 1$ is maximal. Analogously, if $x(V) \leq 1$ is one of the inequalities (3.1), then it is maximal. □

3.5 Facet inducing inequalities with right-hand side 2

In the previous section, we have derived a complete characterization of all facet inducing inequalities with right-hand side 1. We now derive a similar characterization of all facet inducing inequalities with right-hand side 2.

First, we study the structure of valid inequalities with right-hand side 2 and coefficients 0, 1, and 2. Consider a valid inequality $x(V^1) + 2x(V^2) \leq 2$. Clearly, at most two jobs can be started in V. Let $j \in \{1, ..., n\}$ 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.

(i) 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.

(ii) 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.

(iii) 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 (i) holds, $M \subseteq V$ is the set of variables for which statement (ii) holds, and $U \subseteq V$ is the set of variables for which statement (iii) holds. Analogously, we can write $V_j = L_j \cup M_j \cup U_j$. Note that each of the sets L_j, M_j, and U_j may be empty.

If job j is started in a period in V_j^2, then it is impossible to start any job in V before or after job j. It follows that $V_j^2 \subseteq L_j \cap U_j$ for all j, and hence $V^2 \subseteq L \cap U$. It is not hard to see that if $L_j \neq \emptyset$ and $U_j \neq \emptyset$, then the minimum of L_j is less than or equal to the minimum of U_j, and the maximum of L_j is less than or equal to the maximum of U_j. By definition $L_j \cap M_j = \emptyset$ and $M_j \cap U_j = \emptyset$. The set M consists of periods between the maximum of L_j and the minimum of U_j and hence M_j must be empty if $L_j \cap U_j \neq \emptyset$. By definition, of the sets L and U, $x(L) \leq 1$ and $x(U) \leq 1$ are valid inequalities.

We conclude that a valid inequality $x(V^1) + 2x(V^2) \leq 2$ can be represented by a collection of sets L_j, M_j, and U_j. To derive necessary conditions on the structure of facet inducing inequalities with right-hand side 2, we study this LMU-structure more closely.

A valid inequality $x(V^1) + 2x(V^2) \leq 2$ is called *nondecomposable* if it cannot be written as the sum of two valid inequalities $x(W) \leq 1$ and $x(W') \leq 1$. A valid

inequality $x(V^1) + 2x(V^2) \leq 2$ is called *maximal* if there does not exist a valid inequality $x(W^1) + 2x(W^2) \leq 2$ with $V \subseteq W$, $V^2 \subseteq W^2$, where at least one of the subsets is a proper subset. The following lemma yields a general necessary condition and will be used frequently to prove structural properties.

Lemma 3.4 *A facet inducing inequality $x(V^1)+2x(V^2) \leq 2$ is nondecomposable and maximal.* □

The remaining part of the analysis of the LMU-structure proceeds in two phases. In the first phase, we derive conditions on the structure of the sets L and U by considering them separately from the other sets. The structural properties thus derived reveal that we have to distinguish three situations when we consider the overall LMU-structure, based on how the sets L and U can be joined. In the second phase, we investigate each of these three situations and derive conditions on the structure of the set M.

Property 3.3 *If $x(V^1)+2x(V^2) \leq 2$ is facet inducing, then the sets L_j, M_j, and U_j are intervals.*

Proof. Let $j \in \{1, \ldots, n\}$ and assume $L_j \neq \emptyset$. By definition $l_j - p_j + 1$ is the smallest s such that $s \in L_j$. Let s_1 denote the largest such value. Consider any s with $l_j - p_j + 1 < s < s_1$ and let job j be started in period s, i.e., $x_{js} = 1$.

Reasoning as in the proof of Property 3.1, we can show that from $s_1 \in L_j$ it follows that no job can be started in V before job j. Suppose $(i_1, t_1), (i_2, t_2) \in V$ with $s < t_1$ and $s < t_2$ are such that $x_{js} = x_{i_1 t_1} = x_{i_2 t_2} = 1$ defines a feasible schedule. Then the schedule obtained by starting job j in period $l_j - p_j + 1$ instead of in period s is also feasible, which contradicts $l_j - p_j + 1 \in L_j$. Hence, at most one job can be started in V after job j.

We conclude that if $x_{js} = 1$, then no job can be started in V before job j and at most one job can be started in V after job j. Because of the maximality of $x(V^1)+2x(V^2) \leq 2$, we must have $s \in L_j$ and L_j hence is an interval. Analogously, the sets M_j and U_j are intervals. □

Consider a facet inducing inequality $x(V^1) + 2x(V^2) \leq 2$. We have seen that $V^2 \subseteq L \cap U$. Observe that if job j is started in $L_j \cap U_j$, then it is impossible to start any job in V before or after job j. Since $x(V^1) + 2x(V^2) \leq 2$ is maximal, this implies $V_j^2 = L_j \cap U_j$ for all j, i.e., $V^2 = L \cap U$.

Property 3.4 *Let $x(V^1) + 2x(V^2) \leq 2$ be facet inducing.*
(a) *Assume $l = l_1 \leq l_2 \leq \min\{l_j \mid j \in \{3, \ldots, n\}\}$. Then $L_1 = [l - p_1, l_2]$ and $L_j = [l_j - p_j, l]$ for all $j \in \{2, \ldots, n\}$. Furthermore, there exists a $j \in \{2, \ldots, n\}$ such that $L_j \neq \emptyset$.*

(b) *Assume* $u = u_1 \geq u_2 \geq \max\{u_j \mid j \in \{3,\ldots,n\}\}$. *Then* $U_1 = [u_2 - p_1, u]$ *and* $U_j = [u - p_j, u_j]$ *for all* $j \in \{2,\ldots,n\}$. *Furthermore, there exists a* $j \in \{2,\ldots,n\}$ *such that* $U_j \neq \emptyset$.

Proof. (a) Let $x(V^1) + 2x(V^2) \leq 2$ be facet inducing with $l = l_1 \leq l_2 \leq \min\{l_j \mid j \in \{3,\ldots,n\}\}$. Note that by definition $l - p_1 + 1 \in V_1$. Since the earliest possible completion time of a job started in V is l, we must have $l - p_1 + 1 \in L_1$. Reasoning as in the proof of Property 3.2, we find that L_j is an interval with lower bound equal to $l - p_1$ and with upper bound at most equal to l_2. Now, let $x_{1s} = 1$ for some $s \in [l - p_1, l_2]$. As in the proof of the Property 3.3, we can show that from $l - p_1 + 1 \in L_1$, it follows that at most one job can be started in V after job 1. Since $s \leq l_2$, it is impossible to start any job in V before job 1. Because of the maximality of $x(V^1) + 2x(V^2) \leq 2$, we conclude that $s \in L_j$ and hence $L_1 = [l - p_1, l_2]$. Similar arguments can be applied to show that $L_j = [l_j - p_j, l]$ for all $j \in \{2,\ldots,n\}$.

Now suppose $L_j = \emptyset$ for all $j \in \{2,\ldots,n\}$. We show that in this case $x(V^1) + 2x(V^2) \leq 2$ can be written as the sum of two valid inequalities with right-hand side 1, which contradicts the fact that $x(V^1) + 2x(V^2) \leq 2$ is facet inducing. Define $W = \{(1,s) \mid s \in L_1 \cap U_1\} \cup \{(j,s) \mid j \in \{2,\ldots,n\},\ s \in V_j\}$ and $W' = \{(1,s) \mid s \in V_1\}$. We first show that $\sum_{j=2}^{n} \sum_{s \in V_j} x_{js} \leq 1$ is a valid inequality. For all $j \in \{2,\ldots,n\}$ we have, since by assumption $L_j = \emptyset$, $l_j - p_j \geq l$, i.e., $s > l$ for all $s \in V_j$. Consequently, if $x_{j_1 s_1} = x_{j_2 s_2} = 1$ defines a feasible schedule such that $\sum_{j=2}^{n} \sum_{s \in V_j} x_{js} = 2$, then $x_{1,l-p_1+1} = x_{j_1 s_1} = x_{j_2 s_2} = 1$ also defines a feasible schedule, which contradicts the validity of $x(V^1) + 2x(V^2) \leq 2$. Hence, $\sum_{j=2}^{n} \sum_{s \in V_j} x_{js} \leq 1$ is a valid inequality and it easily follows that $x(W) \leq 1$ is also valid. Clearly, $x(W') \leq 1$ is a valid inequality and $x(W) + x(W') = x(V^1) + 2x(V^2)$. We conclude that $x(V^1) + 2x(V^2) \leq 2$ is not facet inducing. Hence, $L_j \neq \emptyset$ for some $j \in \{2,\ldots,n\}$.

The proof of (b) is similar to that of (a). □

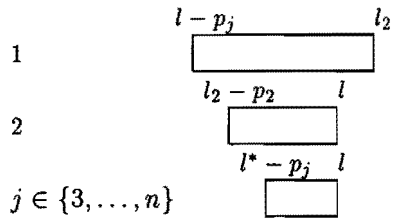
Like the proof of Theorem 3.5, many of the proofs of the properties and theorems presented in this section use counterexamples. If $x(V^1) + 2x(V^2) \leq 2$ is facet inducing, then, since $x(V^1) + 2x(V^2) \leq 2$ is maximal, for any $(j,s) \notin V$ there is a feasible schedule such that $x_{js} = 1$ and $x(V^1) + 2x(V^2) = 2$. We call such a schedule a *counterexample for* (j,s).

Property 3.5 *Let* $x(V^1) + 2x(V^2) \leq 2$ *be facet inducing.*
(a) *Assume* $l = l_1 \leq l_2 \leq l^*$, *where* $l^* = \min\{l_j \mid j \in \{3,\ldots,n\}\}$. *Then for all* $j \in \{3,\ldots,n\}$ *such that* $L_j \neq \emptyset$ *we have* $l_j = l^*$, *and for all* $j \in \{3,\ldots,n\}$ *such that* $L_j = \emptyset$ *we have* $l^* - p_j \geq l$, *i.e.,* $L_j = [l^* - p_j, l]$ *for all* $j \in \{3,\ldots,n\}$.
(b) *Assume* $u = u_1 \geq u_2 \geq u^*$, *where* $u^* = \max\{u_j \mid j \in \{3,\ldots,n\}\}$. *Then for all* $j \in \{3,\ldots,n\}$ *such that* $U_j \neq \emptyset$ *we have* $u_j = u^*$, *and for all* $j \in \{3,\ldots,n\}$ *such that* $U_j = \emptyset$ *we have* $u^* \leq u - p_j$, *i.e.,* $U_j = [u - p_j, u^*]$ *for all* $j \in \{3,\ldots,n\}$.

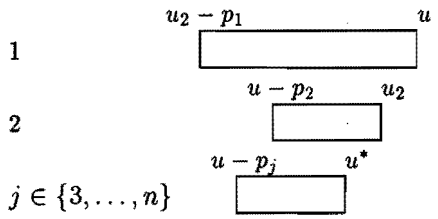
Proof. (a) Let $x(V^1) + 2x(V^2) \leq 2$ be facet inducing with $l = l_1 \leq l_2 \leq l^*$. By definition of l^* and Property 3.4, $L_j \subseteq [l^* - p_j, l]$ for all $j \in \{3, \ldots, n\}$. We assume without loss of generality that $l^* = l_3$. Suppose that $L_j \neq [l^* - p_j, l]$ for some $j \in \{4, \ldots, n\}$, say $L_4 \neq [l^* - p_4, l]$. Clearly, if $l^* - p_4 \geq l$, then $L_4 = \emptyset$ and hence $L_4 = [l^* - p_4, l]$. Consequently $l^* - p_4 < l$ and $l_4 > l^*$, i.e., $l^* - p_4 + 1 \notin V_4$. Since $x(V^1) + 2x(V^2) \leq 2$ is maximal, there is a counterexample for $(4, l^* - p_4 + 1)$. Let $x_{4,l^*-p_4+1} = x_{j_1 s_1} = x_{j_2 s_2} = 1$ define such a counterexample. Since $l^* - p_4 + 1 \leq l$, the jobs j_1 and j_2 are started after job 4. Clearly one of the jobs 1, 2, and 3 does not occur in $\{j_1, j_2\}$; suppose that it is job 3. It is now easy to see that $x_{3,l^*-p_3+1} = x_{j_1 s_1} = x_{j_2 s_2} = 1$ is a feasible schedule, which contradicts the validity of $x(V^1) + 2x(V^2) \leq 2$. If job 1 or job 2 does not occur in $\{j_1, j_2\}$ we obtain a contradiction in the same way.

The proof of (b) is similar to that of (a). □

Properties 3.4 and 3.5 state that if $x(V^1) + 2x(V^2) \leq 2$ is facet inducing and we assume $l = l_1 \leq l_2 \leq l^*$, then the set L can be represented by the following diagram:



Similarly, if we assume $u = u_1 \geq u_2 \geq u^*$, then the set U can be represented by the following diagram:



Observe that a facet inducing inequality with right-hand side 2 has at most three types of intervals L_j, each characterized by the definition of the first period of the interval, and at most three types of intervals U_j, each characterized by the definition of the last period of the interval. Stated slightly differently the intervals L_j have the same structure for all jobs but two. Similarly, the intervals U_j have the same structure for all but two jobs. It turns out that, when we study the overall

LMU-structure, it suffices to consider three situations, based on the jobs with the deviant intervals L_j and U_j:

(1a) $l = l_1 < l_2 \leq l^*$ and $u = u_1 > u_2 \geq u^*$, where $l^* = \min\{l_j \mid j \in \{3, \ldots, n\}\}$ and $u^* = \max\{u_j \mid j \in \{3, \ldots, n\}\}$;

(1b) $l = l_1 < l_2 \leq l^*$, $u = u_1 > u_3 \geq u^*$, and $l_j > l_2$ or $u_j < u_3$ for all $j \in \{2, \ldots, n\}$, where $l^* = \min\{l_j \mid j \in \{3, \ldots, n\}\}$ and $u^* = \max\{u_j \mid j \in \{2, 4, \ldots, n\}\}$;

(2) $l = l_1$ and $u = u_2$.

Before we investigate each of the three situations, we prove a property that applies to case 1.

Property 3.6 *If $x(V^1)+2x(V^2) \leq 2$ is facet inducing with $l = l_1 < l_2 = \min\{l_j \mid j \in \{2, \ldots, n\}\}$ and $u = u_1 > u_i = \max\{u_j \mid j \in \{2, \ldots, n\}\}$, then $l_2 < u_i$.*

Proof. Suppose that $l_2 \geq u_i$. We show that $x(V^1)+2x(V^2) \leq 2$ can be written as the sum of two valid inequalities with right-hand side 1, which contradicts the fact that $x(V^1)+2x(V^2) \leq 2$ is facet inducing. Let $W = \{(1, s) \mid s \in L_1 \cap U_1\} \cup \{(j, s) \mid j \in \{2, \ldots, n\},\ s \in V_j\}$ and $W' = \{(1, s) \mid s \in V_1\} \cup \{(j, s) \mid j \in \{2, \ldots, n\},\ s \in L_j \cap U_j\}$. Clearly $x(W) + x(W') = x(V^1) + 2x(V^2)$ and $x(W') \leq 1$ is a valid inequality. From $V_j \subseteq [l_j - p_j, u_j] \subseteq [l_2 - p_j, u_i]$ for all $j \in \{2, \ldots, n\}$ and $l_2 \geq u_i$, it easily follows that $\sum_{j=2}^{n} \sum_{s \in V_j} x_{js} \leq 1$ is a valid inequality and hence $x(W) \leq 1$ is also valid. □

3.5.1 Case (1a)

Observe that the conditions on l_j and u_j and Properties 3.4 and 3.5 completely determine the sets L and U. Therefore, all that remains to be investigated is the structure of the set M.

Property 3.7 *If $x(V^1) + 2x(V^2) \leq 2$ is facet inducing with $l = l_1 < l_2 \leq l^*$ and $u = u_1 > u_2 \geq u^*$, then we have that $M_1 = [u^* - p_1, l^*] \cap [l_2, u_2 - p_1]$, $M_2 = [u^* - p_2, l^*] \cap [l, u - p_2] \cap [l_2 - p_2, u_2]$, and $M_j = [u_2 - p_j, l_2] \cap [l, u - p_j]$ for $j \in \{3, \ldots, n\}$.*

Proof. Let $x(V^1) + 2x(V^2) \leq 2$ be facet inducing with $l = l_1 < l_2 \leq l^*$ and $u = u_1 > u_2 \geq u^*$. If job 1 is started in M_1, then, since $l_2 \leq l^*$ and $u_2 \geq u^*$, it should be possible to start job 2 in V before as well as after job 1, which implies that $M_1 \subseteq [l_2, u_2 - p_1]$. Furthermore, it should be impossible to start any job $j \in \{3, \ldots, n\}$ in

V and hence $M_1 \subseteq [u^* - p_1, l^*]$. We conclude that $M_1 \subseteq [u^* - p_1, l^*] \cap [l_2, u_2 - p_1]$. If job 1 is started in period $s \in [u^* - p_1, l^*] \cap [l_2, u_2 - p_1]$, then, since $s \in [l_2, u_2 - p_1]$, $[l_2, u_2 - p_1] \subset [l - p_1, u]$, and $L_2 \cap U_2 = [u - p_2, l]$, job 2 cannot be started in $L_2 \cap U_2$. Since $x(V^1) + 2x(V^2) \leq 2$ is maximal, it follows that $M_1 = [u^* - p_1, l^*] \cap [l_2, u_2 - p_1]$.

By definition $M_2 \subseteq [l_2 - p_2, u_2]$. If job 2 is started in M_2, then, since $l = l_1$ and $u = u_1$, it should be possible to start job 1 in V before as well as after job 2, which implies that $M_2 \subseteq [l, u - p_2]$. Furthermore, it should be impossible to start any job $j \in \{3, \ldots, n\}$ in V and hence $M_2 \subseteq [u^* - p_2, l^*]$. We conclude that $M_2 \subseteq [u^* - p_2, l^*] \cap [l, u - p_2] \cap [l_2 - p_2, u_2]$. If job 2 is started in period $s \in [u^* - p_2, l^*] \cap [l, u - p_2] \cap [l_2 - p_2, u_2]$, then, since $s \in [l_2 - p_2, u_2]$ and $L_1 \cap U_1 = [u_2 - p_1, l_2]$, job 1 cannot be started in $L_1 \cap U_1$. Since $x(V^1) + 2x(V^2) \leq 2$ is maximal, it follows that $M_2 = [u^* - p_2, l^*] \cap [l, u - p_2] \cap [l_2 - p_2, u_2]$.

Let $j \in \{3, \ldots, n\}$. If job j is started in M_j, then it should be possible to start job 1 in V before as well as after job j, which implies that $M_j \subseteq [l, u - p_j]$. Furthermore, it should be impossible to start any job $j' \in \{2, 3, \ldots, n\} \setminus \{j\}$ in V and hence $M_j \subseteq [u_2 - p_j, l_2]$. We conclude that $M_j \subseteq [u_2 - p_j, l_2] \cap [l, u - p_j]$. If job j is started in period $s \in [u_2 - p_j, l_2] \cap [l, u - p_j]$, then, since $s \in [u_2 - p_j, l_2]$, $L_1 \cap U_1 = [u_2 - p_1, l_2]$, and, by Property 3.6, $l_2 < u_2$, job 1 cannot be started in $L_1 \cap U_1$. Since $x(V^1) + 2x(V^2) \leq 2$ is maximal, it follows that $M_j = [u_2 - p_j, l_2] \cap [l, u - p_j]$.

Observe that by definition $M_k \subseteq [l_k - p_k, u_k]$ for all $k \in \{1, \ldots, n\}$ and that for all but $k = 2$ this condition is dominated by other conditions. □

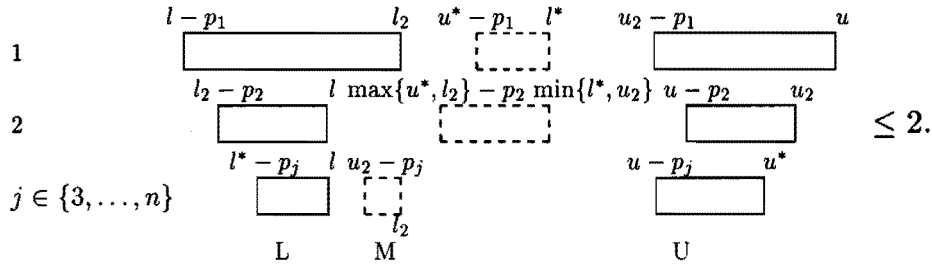
Properties 3.4, 3.5, and 3.7 completely determine the LMU-structure of a facet inducing inequality $x(V^1) + 2x(V^2) \leq 2$ with $l = l_1 < l_2 \leq l^*$ and $u = u_1 > u_2 \geq u^*$. However, in order to emphasize the inherent structure of the intervals M_j, we prefer to use a different representation of the set M. It is easy to show that, if $x(V^1) + 2x(V^2) \leq 2$ is facet inducing with $l = l_1 < l_2 \leq l^*$ and $u = u_1 > u_2 \geq u^*$, then for all $j \in \{3, \ldots, n\}$ we have $[u_2 - p_j, l] \subseteq L_j$ and $[u - p_j, l_2] \subseteq U_j$. We can use this observation to show that Properties 3.4, 3.5, and 3.7 can be combined to give the following theorem.

Theorem 3.6 *A facet inducing inequality $x(V^1) + 2x(V^2) \leq 2$ with $l = l_1 < l_2 \leq l^*$ and $u = u_1 > u_2 \geq u^*$ has the following LMU-structure:*

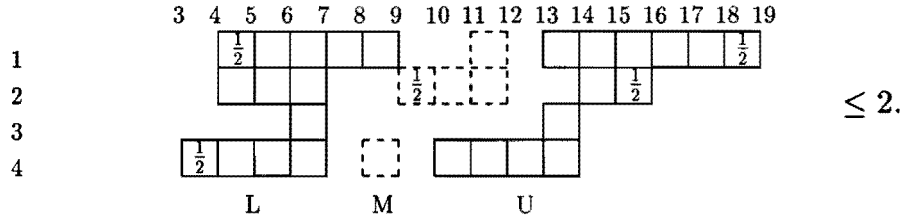
$$
\begin{array}{ll}
L_1 = [l - p_1, l_2], & M_1 = [u^* - p_1, l^*] \setminus (L_1 \cup U_1), \\
L_2 = [l_2 - p_2, l], & M_2 = [\max\{u^*, l_2\} - p_2, \min\{l^*, u_2\}] \setminus (L_2 \cup U_2), \\
L_j = [l^* - p_j, l], & M_j = [u_2 - p_j, l_2] \setminus (L_j \cup U_j), \\
 & \\
U_1 = [u_2 - p_1, u], & \\
U_2 = [u - p_2, u_2], & \\
U_j = [u - p_j, u^*] \quad (j \in \{3, \ldots, n\}), &
\end{array}
\tag{3.4}
$$

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

This theorem states 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:



Example 3.2 Let $n = 4$, $p_1 = 3$, $p_2 = 5$, $p_3 = 6$, and $p_4 = 9$. The inequality with LMU-structure (3.4) and $l = l_1 = 7$, $l_2 = 9$, $l^* = 12$, $u^* = 14$, $u_2 = 16$, and $u = u_1 = 19$ is given by the following diagram:



Note that the fractional solution $x_{15} = x_{1,19} = x_{2,10} = x_{2,16} = x_{4,4} = \frac{1}{2}$ violates this inequality. It is easy to check that this solution satisfies all inequalities with structure (3.3).

Sufficient conditions are given by the following theorem.

Theorem 3.7 *A valid inequality $x(V^1) + 2x(V^2) \leq 2$ with $l = l_1 < l_2 \leq l^*$ and $u = u_1 > u_2 \geq u^*$ and LMU-structure* (3.4) *that is nondecomposable and maximal is facet inducing.*

Proof. Let $x(V^1) + 2x(V^2) \leq 2$ be a valid inequality with $l = l_1 < l_2 \leq l^*$ and $u = u_1 > u_2 \geq u^*$ and LMU-structure (3.4) that is nondecomposable and maximal, and let $F = \{x \in P | x(V^1) + 2x(V^2) = 2\}$. We show that $\dim(F) = \dim(P) - 1$ by exhibiting $\dim(P) - 1$ linearly independent direction vectors in F; a direction vector in F is a vector $d = x - y$ with $x, y \in F$. For notational convenience, a direction vector will be specified by its nonzero components. We give three sets of direction vectors: unit vectors $d_{js} = 1$ for all $(j, s) \notin V$, $d_{js} = 1, vectorsd_{1,l-p_1+1} = d_{2u_2} = -1$

for all $(j,s) \in V^2$, and a set of $|V| - |V^2| - 1$ linearly independent direction vectors $d_{j_1 s_1} = 1, d_{j_2,s_2} = -1$ with $(j_1,s_1),(j_2,s_2) \in V \setminus V^2$. Together these give $\dim(P) - 1$ linearly independent direction vectors in F.

If $(j,s) \notin V$, then, since $x(V^1) + 2x(V^2) \leq 2$ is maximal, there is a counterexample for (j,s) defined by, say, $x_{js} = x_{j_1 s_1} = x_{j_2 s_2} = 1$. Clearly this schedule is an element of F. Note that the schedule $y_{j_1 s_1} = y_{j_2 s_2} = 1$ also is an element of F and hence $d = x - y$ yields the direction vector $d_{js} = 1$.

Note that for $(j,s) \in V^2$ the schedule defined by $x_{js} = 1$ is an element of F. Since $l < l_2$ and, by Property 3.6, $l_2 < u_2$, we have that $y_{1,l-p_1+1} = y_{2u_2} = 1$ defines a feasible schedule. This schedule also is an element of F and hence $d_{js} = 1, d_{1,l-p_1+1} = d_{2u_2} = -1$ is a direction vector in F for all $(j,s) \in V^2$.

We determine the $|V| - |V^2| - 1$ direction vectors $d_{j_1 s_1} = 1, d_{j_2 s_2} = -1$ with $(j_1,s_1),(j_2,s_2) \in V \setminus V^2$ in such a way that the undirected graph G whose vertices are the elements of $V \setminus V^2$ and whose edges are given by the pairs $\{(j_1,s_1),(j_2,s_2)\}$ corresponding to the determined direction vectors is a spanning tree. This implies that the determined direction vectors are linearly independent.

Observe that $d_{j_1 s_1} = 1, d_{j_2 s_2} = -1$ with $(j_1,s_1),(j_2,s_2) \in V \setminus V^2$ is a direction vector in F if there exists an index $(i,t) \in V \setminus V^2$ such that $x_{j_1 s_1} = x_{it} = 1$ and $y_{j_2 s_2} = y_{it} = 1$ both define feasible schedules. In this case, we say that $d_{j_1 s_1} = 1, d_{j_2 s_2} = -1$ is a *direction vector by* (i,t).

First, we determine direction vectors that correspond to edges in G within the sets $\{(j,s) \mid s \in (L_j \cup M_j) \setminus U_j\}$ and $\{(j,s) \mid s \in U_j \setminus L_j\}$. For $s-1, s \in L_1 \setminus U_1$, $d_{1,s-1} = -1, d_{1s} = 1$ is a direction vector by $(2,u_2)$. If $M_1 \neq \emptyset$, then $d_{1l_2} = -1, d_{1m} = 1$ is a direction vector by $(2,u_2)$, where m is the minimum of M_1, and for $s-1, s \in M_1, d_{1,s-1} = -1, d_{1s} = 1$ is a direction vector by $(2,u_2)$. Furthermore, for $s-1, s \in U_1 \setminus L_1, d_{1,s-1} = -1, d_{1s} = 1$ is a direction vector by $(2, l_2 - p_2 + 1)$. Now, let $j \in \{2,\ldots,n\}$. For $s-1, s \in L_j \setminus U_j$, $d_{j,s-1} = -1, d_{js} = 1$ is a direction vector by $(1,u)$. If $M_j \neq \emptyset$, then $d_{jl} = -1, d_{jm} = 1$ is a direction vector by $(1,u)$, where m is the minimum of M_j, and for $s-1, s \in M_j$, $d_{j,s-1} = -1, d_{js} = 1$ is a direction vector by $(1,u)$. Furthermore, for $s-1, s \in U_j \setminus L_j$, $d_{j,s-1} = -1, d_{js} = 1$ is a direction vector by $(1, l - p_1 + 1)$.

Second, we determine direction vectors that correspond to edges in G between sets $\{(j,s) \mid s \in (L_j \cup M_j) \setminus U_j\}$ belonging to different jobs and between sets $\{(j,s) \mid s \in U_j \setminus L_j\}$ belonging to different jobs. We define $W = \{(1,s) \mid s \in L_1 \cap U_1\} \cup \{(j,s) \mid j \in \{2,\ldots,n\}, s \in V_j\}$ and $W' = \{(1,s) \mid s \in V_j\} \cup \{(j,s) \mid j \in \{2,\ldots,n\}, s \in L_j \cap U_j\}$. Clearly $x(W) + x(W') = x(V^1) + 2x(V^2)$ and $x(W') \leq 1$ is a valid inequality. Since $x(V^1) + 2x(V^2) \leq 2$ is nondecomposable, there must be a feasible schedule such that $x(W) = 2$, i.e., $\sum_{j=2}^{n} \sum_{s \in V_j} x_{js} = 2$. Let $x_{j_1 s_1} = x_{j_2 s_2} = 1$ with $s_1 < s_2$ define such a schedule. It is easy to see that we may assume $s_1 = l_{j_1} - p_{j_1} + 1$ and $s_2 = u_{j_2}$. Since $l = l_1, y_{1,l-p_1+1} = y_{j_2 u_{j_2}} = 1$ also defines a feasible schedule, it follows that $d_{1,l-p_1+1} = -1, d_{j_1,l_{j_1}-p_{j_1}+1} = 1$ is a direction vector by (j_2,s_2). In the same way, since $u = u_1$, $y_{j_1,l_{j_1}-p_{j_1}+1} = y_{1u} = 1$ defines a

feasible schedule, it follows that $d_{1u} = -1, d_{j_2 u_{j_2}} = 1$ is a direction vector by (j_1, s_1). For $j \in \{2, \ldots, n\} \setminus \{j_1\}$ such that $L_j \cup M_j \neq \emptyset$, $d_{j_1, l_{j_1} - p_{j_1} + 1} = -1, d_{j, l_j - p_j + 1} = 1$ is a direction vector by $(1, u)$. Furthermore, for $j \in \{2, \ldots, n\} \setminus \{j_2\}$ such that $U_j \neq \emptyset$, $d_{j_2 u_{j_2}} = -1, d_{j u_j} = 1$ is a direction vector by $(1, l - p_1 + 1)$.

Finally, we determine a direction vector that corresponds to an edge in G between $L \cup M$ and U. Since $x(V^1) + 2x(V^2) \leq 2$ is nondecomposable and $x(U) \leq 1$ is a valid inequality, there exists a feasible schedule with $x(L) + x(M) = 2$. Let $x_{j_1 s_1} = x_{j_2 s_2} = 1$ define such a schedule. Since $l_1 = l$, we may assume without loss of generality that $j_1 = 1$. Since $s_2 \in L_{j_2} \cup M_{j_2}$, $y_{j_2 s_2} = y_{1u} = 1$ also is a feasible schedule. It follows that $d_{1 s_1} = -1, d_{1u} = 1$ is a direction vector by (j_2, s_2).

It is easy to see that the determined direction vectors form a spanning tree of G and we hence have determined $|V| - |V^2| - 1$ linearly independent direction vectors. □

Specific necessary and sufficient conditions for a valid inequality with $l = l_1 < l_2 \leq l^*$ and $u = u_1 > u_2 \geq u^*$ and LMU-structure (3.4) to be nondecomposable and maximal are given in the following two theorems.

Theorem 3.8 *A valid inequality $x(V^1) + 2x(V^2) \leq 2$ with $l = l_1 < l_2 \leq l^*$ and $u = u_1 > u_2 \geq u^*$ that has LMU-structure* (3.4) *is nondecomposable if and only if $M_j \neq \emptyset$ for some $j \in \{1, \ldots, n\}$, and $l^* < u_2$ or $l_2 < u^*$.*

Proof. Let $x(V^1) + 2x(V^2) \leq 2$ be a valid inequality with $l = l_1 < l_2 \leq l^*$ and $u = u_1 > u_2 \geq u^*$ that has LMU-structure (3.4).

Suppose that $x(V^1) + 2x(V^2) \leq 2$ is nondecomposable. Since $x(L) \leq 1$ and $x(U) \leq 1$ are valid inequalities, we must have $M_j \neq \emptyset$ for some $j \in \{1, \ldots, n\}$. Suppose $l^* \geq u_2$ and $l_2 \geq u^*$. Let $W = \{(1, s) \mid s \in L_1 \cap U_1\} \cup \{(j, s) \mid j \in \{2, \ldots, n\}, s \in V_j\}$ and $W' = \{(1, s) \mid s \in V_1\} \cup \{(j, s) \mid j \in \{2, \ldots, n\}, s \in L_j \cap U_j\}$. Clearly $x(W) + x(W') = x(V^1) + 2x(V^2)$ and $x(W') \leq 1$ is valid. Observe that $V_2 \subseteq [l_2 - p_2, u_2]$ and, for $j \in \{3, \ldots, n\}$, $V_j \subseteq [l^* - p_j, u^*] \subseteq [u_2 - p_j, l_2]$. Since, by Property 3.6, $l_2 < u_2$, it follows that $\sum_{j=2}^{n} \sum_{s \in V_j} x_{js} \leq 1$ is valid and hence $x(W) \leq 1$ is also valid, which yields a contradiction. This implies that $l^* < u_2$ or $l_2 < u^*$.

Suppose that $M_j \neq \emptyset$ for some $j \in \{1, \ldots, n\}$, and $l^* < u_2$ or $l_2 < u^*$. Let W and W' be such that $x(W) + x(W') = x(V^1) + 2x(V^2)$ and $x(W) \leq 1$ and $x(W') \leq 1$ are valid inequalities. We assume without loss of generality that $(1, l - p_1 + 1) \in W$. Note that $x_{1l-p_1+1} = x_{2u_2} = 1$ is a feasible schedule. The validity of $x(W) \leq 1$ and $(1, l - p_1 + 1) \in W$ imply that $(2, u_2) \in W'$. We conclude that, since $l < u_2$ and $(1, l - p_1 + 1) \in W$, we have $(2, u_2) \in W'$. We show that $(1, u) \in W'$ and $(2, l_2 - p_2 + 1) \in W$. By assumption either $l^* < u_2$ or $l_2 < u^*$. Suppose $l^* < u_2$. Note that in this case $U_2 \neq \emptyset$ and $L_j \neq \emptyset$ for some $j \in \{3, \ldots, n\}$. Since $l^* < u_2$ and $(2, u_2) \in W'$, we have $(j, l^* - p_j + 1) \in W$ for all $j \in \{3, \ldots, n\}$ such that

$L_j \neq \emptyset$. Furthermore, since $l^* < u_2 < u$, it follows that $(1, u) \in W'$, and since $l_2 < u_2 < u$, it follows that $(2, l_2 - p_2 + 1) \in W$. Analogously, $(2, l_2 - p_2 + 1) \in W$ and $(1, u) \in W'$ if $l_2 < u^*$.

By assumption $M_j \neq \emptyset$ for some $j \in \{1, \ldots, n\}$. Suppose that $M_j \neq \emptyset$ for some $j \in \{2, \ldots, n\}$. If job j is started in M_j, then job 1 can be started in V before or after job j. If $s \in M_j$, then, since $x_{1,l-p_1+1} = x_{js} = 1$ is a feasible schedule and $(1, l-p_1+1) \in W$, it follows that $(j, s) \in W'$. We find that $x_{js} = x_{1u} = 1$ is a feasible schedule such that $x(W') = 2$, which yields a contradiction. We conclude that from $(1, l - p_1 + 1) \in W$, $(1, u) \in W'$ and $M_j \neq \emptyset$, it follows that $x(V^1) + 2x(V^2) \leq 2$ is nondecomposable.

Suppose that $M_1 \neq \emptyset$. If job 1 is started in M_1 then job 2 can be started in V before as well as after job 1. As in the previous case, from $(2, l_2 - p_2 + 1) \in W$, $(2, u_2) \in W'$, and $M_1 \neq \emptyset$, it follows that $x(V^1) + 2x(V^2) \leq 2$ is nondecomposable. □

Theorem 3.9 *A valid inequality $x(V^1) + 2x(V^2) \leq 2$ with $l = l_1 < l_2 \leq l^*$ and $u = u_1 > u_2 \geq u^*$, that has LMU-structure (3.4) is maximal if and only if each of the following eight hold:*

(1) *If $u^* < l_2$ and $l < l_2 - p_2$, then $L_1 \cap U_1 \neq \emptyset$;*

(2) *If $u_2 < l^*$ and $u_2 < u - p_2$, then $L_1 \cap U_1 \neq \emptyset$;*

One of the following holds:
(3a) $L_j \cap U_j \neq \emptyset$ *for some* $j \in \{2, \ldots, n\}$;
(3b) $l \leq u^* - p_2$ *and* $L_2 \neq \emptyset$ *and* $U_j \neq \emptyset$ *for some* $j \in \{3, \ldots, n\}$;
(3c) $l \leq u_2 - \min\{p_j \mid j \in \{3, \ldots, n\},\ L_j \neq \emptyset\}$ *and* $U_2 \neq \emptyset$ *and* $L_j \neq \emptyset$ *for some* $j \in \{3, \ldots, n\}$;

One of the following holds:
(4a) $L_j \cap U_j \neq \emptyset$ *for some* $j \in \{2, \ldots, n\}$;
(4b) $l^* \leq u - p_2$ *and* $U_2 \neq \emptyset$ *and* $L_j \neq \emptyset$ *for some* $j \in \{3, \ldots, n\}$;
(4c) $l_2 \leq u - \min\{p_j \mid j \in \{3, \ldots, n\},\ U_j \neq \emptyset\}$ *and* $L_2 \neq \emptyset$ *and* $U_j \neq \emptyset$ *for some* $j \in \{3, \ldots, n\}$;

One of the following holds:
(5a) $L_1 \cap U_1 \neq \emptyset$;
(5b) $\min\{l_2, l + p_2\} \leq u^* - p_1$ *and* $U_j \neq \emptyset$ *for some* $j \in \{3, \ldots, n\}$;
(5c) $\min\{l_2, l + p_2\} \leq u - \min\{p_j \mid j \in \{3, \ldots, n\},\ M_j \neq \emptyset\}$ *and* $M_j \neq \emptyset$ *for some* $j \in \{3, \ldots, n\}$;

One of the following holds:
(6a) $L_1 \cap U_1 \neq \emptyset$;
(6b) $l^* \leq \max\{u_2, u - p_2\} - p_1$ *and* $L_j \neq \emptyset$ *for some* $j \in \{3, \ldots, n\}$;
(6c) $l \leq \max\{u_2, u - p_2\} - \min\{p_j \mid j \in \{3, \ldots, n\},\ M_j \neq \emptyset\}$ *and* $M_j \neq \emptyset$ *for some* $j \in \{3, \ldots, n\}$;

For all $j \in \{3,\ldots,n\}$, *one of the following holds:*
(7a) $\min\{l^*, l+p_j\} \leq u_2 - p_1$ *and* $M_1 \neq \emptyset$;
(7b) $\min\{l^*, l+p_j\} \leq u - p_2$ *and* $M_2 \neq \emptyset$;
(7c) $\min\{l^*, l+p_j\} \leq l_2$;

For all $j \in \{3,\ldots,n\}$, *one of the following holds:*
(8a) $l_2 \leq \max\{u^*, u-p_j\} - p_1$ *and* $M_1 \neq \emptyset$;
(8b) $l \leq \max\{u^*, u-p_j\} - p_2$ *and* $M_2 \neq \emptyset$;
(8c) $u_2 \leq \max\{u^*, u-p_j\}$.

Proof. Let $x(V^1) + 2x(V^2) \leq 2$ be a valid inequality with $l = l_1 < l_2 \leq l^*$ and $u = u_1 > u_2 \geq u^*$ that has LMU-structure (3.4). Observe that $x(V^1)+2x(V^2) \leq 2$ is maximal if and only if it is impossible to extend any of the intervals L_j, M_j and U_j.

First, we show that M_1 cannot be extended. If $l_2 < u^* - p_1$, i.e., $[l_2, u^* - p_1] \neq \emptyset$, then $x_{2,l_2-p_2+1} = x_{1s} = x_{ju^*} = 1$ defines a counterexample for $(1,s)$ for all $s \in [l_2, u^* - p_1]$, where $j \in \{3,\ldots,n\}$ is such that $u_j = u^*$. If $l^* < u_2 - p_1$, i.e., $[l^*, u_2 - p_1] \neq \emptyset$, then $x_{j,l^*-p_j+1} = x_{1s} = x_{2u_2} = 1$ defines a counterexample for $(1,s)$ for $s \in [l^*, u_2 - p_1]$, where $j \in \{3,\ldots,n\}$ is such that $l_j = l^*$. Hence, M_1 cannot be extended. In the same way we can show that the intervals M_j with $j \in \{3,\ldots,n\}$ cannot be extended.

We show that M_2 cannot be extended if and only if (1) and (2) hold. Suppose that (1) does not hold, i.e., $u^* < l_2$, $l < l_2 - p_2$, and $L_1 \cap U_1 = \emptyset$. Clearly $(2, l_2 - p_2) \notin V$. Let job 2 be started in period $l_2 - p_2$. Since $l^* > l_2$, it is impossible to start a job $j \in \{3,\ldots,n\}$ in V before job 2. Since $u^* < l_2$, it is impossible to start a job $j \in \{3,\ldots,n\}$ in V after job 2. Clearly, job 1 cannot be started in $L_1 \cap U_1$, and we find that M_2 can be extended by $l_2 - p_2$. In the same way, if $u_2 < l^*$, $u_2 < u - p_2$, and $L_1 \cap U_1 = \emptyset$, then M_2 can be extended by $u_2 + 1$.

Now, suppose that (1) and (2) hold. Suppose $u^* \geq l_2$, i.e., $[l, \max\{u^*, l_2\} - p_2] = [l, u^* - p_2]$. If $l < u^* - p_2$, then $x_{1,l-p_1+1} = x_{2s} = x_{ju^*} = 1$ defines a counterexample for $(2,s)$ for all $s \in [l, u^* - p_2]$, where $j \in \{3,\ldots,n\}$ is such that $u_j = u^*$. Suppose that $u^* < l_2$, i.e., $[l, \max\{u^*, l_2\} - p_2] = [l, l_2 - p_2]$. If $l < l_2 - p_2$, then, by (1), $L_1 \cap U_1 \neq \emptyset$ and $x_{2s} = x_{1l_2} = 1$ defines a counterexample for $(2,s)$ for all $s \in [l, l_2 - p_2]$. Hence, there is a counterexample for $(2,s)$ for all $s \in [l, \max\{u^*, l_2\} - p_2]$. In the same way, we can show that there is a counterexample for $(2,s)$ for all $s \in [\min\{l^*, u_2\}, u - p_2]$. We conclude that M_2 cannot be extended.

Clearly, the upper bound of L_1 cannot be increased. The lower bound of L_1 cannot be decreased if and only if there is a counterexample for $(1, l - p_1)$. Such a counterexample is given by a feasible schedule $x_{1,l-p_1} = x_{js} = 1$ with $(j,s) \in V^2$ or by a feasible schedule $x_{1,l-p_1} = x_{j_1 s_1} = x_{j_2 s_2} = 1$ with $(j_1, s_1), (j_2, s_2) \in V \setminus V^2$. Observe that if $j \in \{2,\ldots,n\}$ is such that $L_j \cap U_j \neq \emptyset$, then $x_{1,l-p_1} = x_{jl} = 1$ defines a counterexample for $(1, l - p_1)$. We find that there is a counterexample of the first type if and only if (3a) holds. Consider a counterexample of the second

type. Since $u_2 \geq \max\{u_j \mid j \in \{3,\ldots,n\}\}$, we may assume that job 2 occurs in this counterexample, i.e., the counterexample contains job 2 and a job $j_1 \in \{3,\ldots,n\}$. Suppose that job 2 is started before job j_1. It is easy to see that job 2 is started in L_2. So $L_2 \neq \emptyset$ and job 2 is started in period l. Furthermore, job j_1 is started in U_{j_1}. Hence, $l \leq u^* - p_2$ and $U_{j_1} \neq \emptyset$. We find that there is a counterexample of the second type such that job 2 is started before job j_1 if and only if (3b) holds. If job 2 is started after job j_1, then job j_1 is started in period l and job 2 in U_2. Job j_1 may be chosen such that $p_{j_1} = \min\{p_j \mid j \in \{3,\ldots,n\},\ L_j \neq \emptyset\}$. We find that there is a counterexample of the second type such that job 2 is started after job j_1 if and only if (3c) holds. We conclude that the lower bound of L_1 cannot be decreased if and only if (3) holds. Analogously, U_1 cannot be extended if and only if (4) holds.

Clearly, L_2 cannot be extended if and only if there is a counterexample for $(2, l_2 - p_2)$ if $L_2 \neq \emptyset$, and a counterexample for $(2, l)$ if $L_2 = \emptyset$, i.e., if and only if there is a counterexample for $(2, y - p_2)$, where $y = \min\{l_2, l + p_2\}$. The proof that there is such a counterexample if and only if (5) holds, is similar to the proof that there is a counterexample for $(1, l - p_1)$ if and only if (3) holds. Analogously, U_2 cannot be extended if and only if (6) holds.

Let $j \in \{3,\ldots,n\}$. It is easy to see that L_j cannot be extended if and only if there is a counterexample for $(j, y - p_j)$, where $y = \min\{l^*, l + p_j\}$. Suppose $y \leq l_2$. If $L_1 \cap U_1 \neq \emptyset$, then $x_{j,y-p_j} = x_{1l_2} = 1$ defines a counterexample for $(j, y - p_j)$. If $L_1 \cap U_1 = \emptyset$, i.e., $l_2 \leq u_2 - p_1$, then $x_{j,y-p_j} = x_{1l_2} = x_{2u_2}$ defines such a counterexample. Hence, if $y \leq l_2$, then there is a counterexample for $(j, y - p_j)$. Now, suppose $y > l_2$. Since $x(U) \leq 1$, in any counterexample at least one job is started in $L \cup M$. If $x_{j,y-p_j} = 1$, then job 1 and 2 are the only jobs that can be started in $L \cup M$. It is now easy to see that a counterexample for $(j, y - p_j)$ contains job 1 and job 2, and we find that there is such a counterexample if and only if (7a) or (7b) holds. Analogously, the intervals U_j with $j \in \{3,\ldots,n\}$ cannot be extended if and only if (8) holds. □

3.5.2 Case (1b)

As in case (1a), the conditions on l_j and u_j and Properties 3.4 and 3.5 completely determine the sets L and U. From these properties we can easily derive that, if $l_2 = l^*$ and $u_3 = u^*$, then $L_i \neq \emptyset$ and $U_i \neq \emptyset$ for all i such that $p_i = \max\{p_j \mid j \in \{2,\ldots,n\}\}$. But then $l_i = l_2$ and $u_i = u_3$, and we are in case (1a). We conclude that $l_2 < l^*$ or $u_3 > u^*$. All that remains to be investigated is the structure of the set M.

Property 3.8 *If $x(V^1) + 2x(V^2) \leq 2$ is facet inducing 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 $j \in \{2,\ldots,n\}$, then $M_1 = \emptyset$, $M_2 = [u_3 - p_2, l^*] \cap [l, u - p_2] \cap [l_2 - p_2, u^*]$, $M_3 = [u^* - p_3, l_2] \cap [l, u - p_3] \cap [l^* - p_3, u_3]$, and $M_j = [u_3 - p_j, l_2] \cap [l, u - p_j]$ for $j \in \{4,\ldots,n\}$.*

Proof. Let $x(V^1)+2x(V^2) \leq 2$ be facet inducing 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 $j \in \{2,\ldots,n\}$. If job 1 is started in V and it is possible to start a job in V before job 1, then, since $l_2 = \min\{l_j \mid j \in \{2,\ldots,n\}\}$, job 2 can be started in V before job 1. If it is possible to start a job in V after job 1, then, since $u_3 = \max\{u_j \mid j \in \{2,\ldots,n\}\}$, job 3 can be started after job 1. This implies that $M_1 = \emptyset$.

By definition $M_2 \subseteq [l_2 - p_2, u^*]$. If job 2 is started in M_2, then, since $l = l_1$ and $u = u_1$, it should be possible to start job 1 in V before as well as after job 2, which implies that $M_2 \subseteq [l, u - p_2]$. Furthermore, it should be impossible to start any job $j \in \{3,\ldots,n\}$ in V and hence $M_2 \subseteq [u_3 - p_2, l^*]$. We conclude that $M_2 \subseteq [u_3 - p_2, l^*] \cap [l, u - p_2] \cap [l_2 - p_2, u^*]$. If job 2 is started in period $s \in [u_3-p_2, l^*]\cap[l, u-p_2]\cap[l_2-p_2, u^*]$, then, since $s \in [l_2-p_2, u^*]$, $L_1 \cap U_1 = [u_3-p_1, l_2]$, and $u_3 \geq u^*$, job 1 cannot be started in $L_1 \cap U_1$. Since $x(V^1)+2x(V^2) \leq 2$ is maximal, it follows that $M_2 = [u_3 - p_2, l^*] \cap [l, u - p_2] \cap [l_2 - p_2, u^*]$. Analogously, $M_3 = [u^* - p_3, l_2] \cap [l, u - p_3] \cap [l^* - p_3, u_3]$ and $M_j = [u_3 - p_j, l_2] \cap [l, u - p_j]$ for $j \in \{4,\ldots,n\}$. □

Properties 3.4, 3.5, and 3.8 determine the LMU-structure of 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 $j \in \{2,\ldots,n\}$. As in case (1a), we prefer to use a different representation of the set M in order to emphasize the inherent structure of the intervals M_j. It turns out 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 $j \in \{2,\ldots,n\}$ has the following property, which restricts the class of inequalities determined by Properties 3.4, 3.5, and 3.8 and leads to a simpler form of the intervals M_j.

Property 3.9 *If $x(V^1) + 2x(V^2) \leq 2$ is facet inducing 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 $j \in \{2,\ldots,n\}$, then $l^* \leq u^*$.*

Proof. Let $x(V^1) + 2x(V^2) \leq 2$ be facet inducing 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 $j \in \{2,\ldots,n\}$. To be able to prove that $l^* \leq u^*$, we first show that $[u_3-p_2, \min\{l^*, u_3\}] \subseteq V_2$ and $[\max\{u^*, l_2\}-p_3, l_2] \subseteq V_3$. It is easy to see that if job 2 is started in $[u_3 - p_2, \min\{l^*, u_3\}]$, then it is impossible to start any job $j \in \{3,\ldots,n\}$ in V and job 1 cannot be started in $L_1 \cap U_1$. Since $x(V^1) + 2x(V^2) \leq 2$ is maximal, it follows that $[u_3 - p_2, \min\{l^*, u_3\}] \subseteq V_2$. Analogously, $[\max\{u^*, l_2\} - p_3, l_2] \subseteq V_3$. Since, by assumption, $l_j > l_2$ or $u_j < u_3$ for all $j \in \{2,\ldots,n\}$, we have $l_3 > l_2$ and $u_2 < u_3$. From $[u_3 - p_2, \min\{l^*, u_3\}] \subseteq V_2$ and $u_2 < u_3$, we conclude that $l^* < u_3$. Analogously, $u^* > l_2$. From $l^* < u_3$ and $u^* > l_2$, it follows that $l^* < u^*$ if $l_2 = l^*$ or $u_3 = u^*$. We still have to show that $l^* \leq u^*$ if $l_2 < l^*$ and $u_3 > u^*$.

Suppose $l_2 < l^*$ and $u_3 > u^*$. We show that $[u - p_j, l^*] \subseteq U_j$ for all $j \in \{2, 4,\ldots,n\}$. Let $j \in \{2, 4,\ldots,n\}$, and let job j be started in $[u - p_j, l^*]$. Clearly, any job $i \in \{3,\ldots,n\} \setminus \{j\}$ cannot be started before job j. If job 2 is started before

job j, then, since $M_2 \subseteq [u_3 - p_j, l^*]$ and $l^* < u_3$, job 2 is not started in M_2, and job 2 is hence started in L_2. It is now easy to see that at most one job can be started in V before job j. Since $L_1 \cap U_1 = [u_3 - p_1, l_2]$ and $l_2 < l^* < u_3$, job 1 cannot be started in $L_1 \cap U_1$. Because of the maximality of $x(V^1) + 2x(V^2) \leq 2$, we conclude that $[u - p_j, l^*] \subseteq U_j$. Observe that from $l_2 < u^*$ and Property 3.8 follows that $U_j \neq \emptyset$ for some $j \in \{2, 4, \ldots, n\}$ or $M_2 \neq \emptyset$. If $U_j \neq \emptyset$ for some $j \in \{2, 4, \ldots, n\}$, then, since $[u - p_j, l^*] \subseteq U_j$, $l^* \leq u^*$. If $M_2 \neq \emptyset$, then since, by Property 3.8, $M_2 = [u_3 - p_2, l^*] \cap [l, u - p_2] \cap [l_2 - p_2, u^*]$, we must have $u_3 - p_2 < l^*$. It is easy to see that if job 2 is started in $[u_3 - p_2, l^*] \cap [l, u - p_2]$, then job 1 is the only job that can be started before as well as after job 2 and job 1 cannot be started in $L_1 \cap U_1$. Since $x(V^1) + 2x(V^2) \leq 2$ is maximal, this implies $M_2 = [u_3 - p_2, l^*] \cap [l, u - p_2]$ and we conclude that $l^* \leq u^*$. □

It is not hard to see that Properties 3.4, 3.5, 3.8, and 3.9 can be combined to the following theorem.

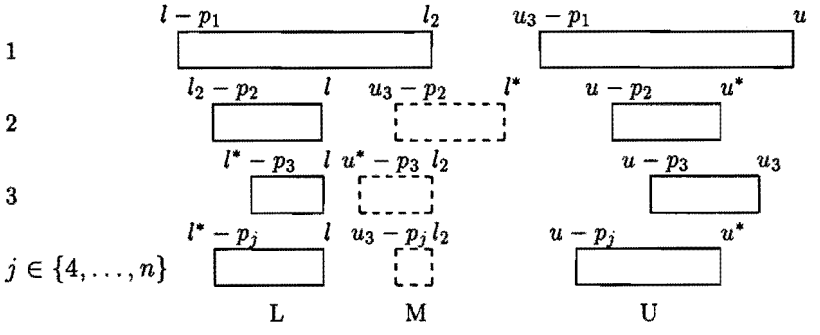
Theorem 3.10 *A facet inducing inequality $x(V^1) + 2x(V^2) \leq 2$ with $l = l_1 < l_2 \leq l^*$, $u = u_1 > u_3 \geq u^*$, and $l_j > l_2$ or $u_j < u_3$ for all $j \in \{2, \ldots, n\}$ has the following LMU-structure:*

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

$$
\begin{aligned}
U_1 &= [u_3 - p_1, u], \\
U_2 &= [u - p_2, u^*], \\
U_3 &= [u - p_3, u_3], \\
U_j &= [u - p_j, u^*] \quad (j \in \{4, \ldots, n\}),
\end{aligned}
\tag{3.5}
$$

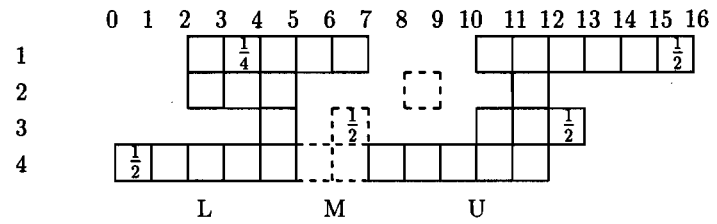
where $l^* \leq u^*$. □

This theorem states 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 $j \in \{2, \ldots, n\}$ can be represented by the following diagram:



$\leq 2.$

Example 3.3 Let $n = 4$, $p_1 = 3$, $p_2 = 5$, $p_3 = 6$, and $p_4 = 9$. The inequality with LMU-structure (3.5) and $l = l_1 = 5$, $l_2 = 7$, $l^* = 9$, $u^* = 12$, $u_3 = 13$, and $u = u_1 = 16$ is given by the following diagram:



$\leq 2.$

Note that the fractional solution $x_{1,16} = x_{37} = x_{3,13} = x_{41} = \frac{1}{2}$ and $x_{14} = \frac{1}{4}$ violates this inequality. It is easy to check that this solution satisfies all inequalities with structure (3.3).

Sufficient conditions are given in the following theorem.

Theorem 3.11 *A valid 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* $j \in \{2, \ldots, n\}$ *and LMU-structure* (3.5) *that is nondecomposable and maximal is facet inducing.* □

The proof of this theorem is similar to that of Theorem 3.7.

Specific necessary and sufficient conditions for a valid inequality given by $x(V^1) + 2x(V^2) \leq 2$ 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 $j \in \{2, \ldots, n\}$, and LMU-structure (3.5) to be nondecomposable and maximal are given in the following two theorems.

Theorem 3.12 *A valid 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* $j \in \{2, \ldots, n\}$, *that has LMU-structure* (3.5) *is nondecomposable if and only if* $M_j \neq \emptyset$ *for some* $j \in \{2, \ldots, n\}$.

Proof. Let $x(V^1) + 2x(V^2) \leq 2$ be a valid inequality 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 $j \in \{2, \ldots, n\}$, that has LMU-structure (3.5). If $x(V^1) + 2x(V^2) \leq 2$ is nondecomposable, then, since $x(L) \leq 1$ and $x(U) \leq 1$ are valid inequalities, $M_j \neq \emptyset$ for some $j \in \{2, \ldots, n\}$.

Suppose $M_j \neq \emptyset$ for some $j \in \{2, \ldots, n\}$. Let W and W' be such that $x(W) + x(W') = x(V^1) + 2x(V^2)$, and $x(W) \leq 1$ and $x(W') \leq 1$ are valid. We assume without loss of generality that $(1, l - p_1 + 1) \in W$. Since $l < l_2 < u_3$, $x_{1,l-p_1+1} = x_{3u_3} = 1$ is a feasible schedule. From $(1, l - p_1 + 1) \in W$ and $x(W) \leq 1$ it follows that $(3, u_3) \in W'$. In the same way, since $l_2 < u_3$ and $(3, u_3) \in W'$, it follows that

$(2, l_2 - p_2 + 1) \in W$ and, since $l_2 < u$, it follows that $(1, u) \in W'$. Now, let $j \in \{2, \ldots, n\}$ be such that $M_j \neq \emptyset$. If job j is started in M_j, then job 1 can be started in V before as well as after job j. If $s \in M_j$, then, since $x_{1,l-p_1+1} = x_{js} = 1$ is a feasible schedule and $(1, l - p_1 + 1) \in W$, we have $(j, s) \in W'$. We find that $x_{js} = x_{1u} = 1$ is a feasible schedule such that $x(W') = 2$, which yields a contradiction. We conclude that from $(1, l - p_1 + 1) \in W$, $(1, u) \in W'$, and $M_j \neq \emptyset$, it follows that $x(V^1) + 2x(V^2) \leq 2$ is nondecomposable. □

Theorem 3.13 *A valid 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 $j \in \{2, \ldots, n\}$, that has LMU-structure (3.5) is maximal if and only if each of the following six hold:*

One of the following holds:
(1a) $L_j \cap U_j \neq \emptyset$ *for some* $j \in \{2, \ldots, n\}$;
(1b) $l \leq u^* - p_3$, $L_3 \neq \emptyset$, *and* $U_j \neq \emptyset$ *for some* $j \in \{2, 4, \ldots, n\}$;
(1c) $l \leq u_3 - \min\{p_j \mid j \in \{2, 4, \ldots, n\},\ L_j \neq \emptyset\}$;

One of the following holds:
(2a) $L_j \cap U_j \neq \emptyset$ *for some* $j \in \{2, \ldots, n\}$;
(2b) $l^* \leq u - p_2$, $U_2 \neq \emptyset$, *and* $L_j \neq \emptyset$ *for some* $j \in \{3, \ldots, n\}$;
(2c) $l_2 \leq u - \min\{p_j \mid j \in \{3, \ldots, n\},\ U_j \neq \emptyset\}$;

One of the following holds:
(3a) $\min\{l^*, l + p_3\} \leq u - p_2$ *and* $M_2 \neq \emptyset$;
(3b) $\min\{l^*, l + p_3\} \leq l_2$ *and* $L_1 \cap U_1 \neq \emptyset$;
(3c) $\min\{l^*, l + p_3\} \leq l_2$ *and* $\min\{l^*, l + p_3\} \leq u^* - p_1$;
(3d) $\min\{l^*, l + p_3\} \leq l_2$, $\min\{l^*, l + p_3\} \leq u - \min\{p_j \mid j \in \{4, \ldots, n\},\ M_j \neq \emptyset\}$, *and* $M_j \neq \emptyset$ *for some* $j \in \{4, \ldots, n\}$;

One of the following holds:
(4a) $l \leq \max\{u^*, u - p_2\} - p_3$ *and* $M_3 \neq \emptyset$;
(4b) $u_3 \leq \max\{u^*, u - p_2\}$ *and* $L_1 \cap U_1 \neq \emptyset$;
(4c) $u_3 \leq \max\{u^*, u - p_2\}$ *and* $l^* \leq \max\{u^*, u - p_2\} - p_1$;
(4d) $u_3 \leq \max\{u^*, u - p_2\}$, $l \leq \max\{u^*, u - p_2\} - \min\{p_j \mid j \in \{4, \ldots, n\}, M_j \neq \emptyset\}$, *and* $M_j \neq \emptyset$ *for some* $j \in \{4, \ldots, n\}$;

For all $j \in \{4, \ldots, n\}$, one of the following holds:
(5a) $\min\{l^*, l + p_j\} \leq u - p_2$ *and* $M_2 \neq \emptyset$;
(5b) $\min\{l^*, l + p_j\} \leq l_2$;

For all $j \in \{4, \ldots, n\}$, one of the following holds:
(6a) $l \leq \max\{u^*, u - p_j\} - p_3$ *and* $M_3 \neq \emptyset$;
(6b) $u_3 \leq \max\{u^*, u - p_j\}$.

Proof. Let $x(V^1) + 2x(V^2) \leq 2$ be a valid inequality 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 $j \in \{2, \ldots, n\}$, that has LMU-structure (3.5). Observe that $x(V^1) + 2x(V^2) \leq 2$ is maximal if and only if it is impossible to extend any of the intervals L_j, M_j, and U_j.

In the same way as in the proof of Theorem 3.9 it can be shown that the intervals M_j cannot be extended.

We show that L_2 cannot be extended. Note that since $l_2 < u_3$, we have $L_2 \neq \emptyset$. It is now easy to see that it suffices to show that there is a counterexample for $(2, l_2 - p_2)$. If $L_1 \cap U_1 \neq \emptyset$, then $x_{2,l_2-p_2} = x_{1l_2} = 1$ defines a counterexample for $(2, l_2 - p_2)$. If $L_1 \cap U_1 = \emptyset$, i.e., $l_2 \leq u_3 - p_1$, then $x_{2,l_2-p_2} = x_{1l_2} = x_{3u_3} = 1$ defines a counterexample for $(2, l_2 - p_2)$. Hence L_2 cannot be extended. Analogously, U_3 cannot be extended.

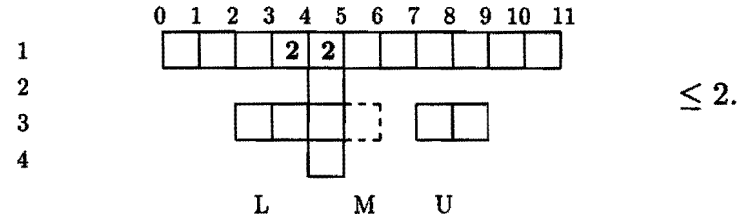
Clearly, L_1 cannot be extended if and only if there is a counterexample for $(1, l - p_1)$. If such a counterexample is given by the schedule $x_{1,l-p_1} = x_{j_1 s_1} = x_{j_2 s_2} = 1$ with $(j_1, s_1), (j_2, s_2) \in V \setminus V^2$, then, since $u_3 = \max\{u_j \mid j \in \{2, \ldots, n\}\}$, we may assume that job 3 occurs in this counterexample. We can use this observation to show that L_1 cannot be extended if and only if (1) holds. Analogously, U_1 cannot be extended if and only if (2) holds.

It is easy to see that L_3 cannot be extended if and only if there is a counterexample for $(3, y - p_3)$, where $y = \min\{l^*, l + p_3\}$. Since $u_1 = u$, we may assume that such a counterexample contains job 1. Suppose $y > l_2$. Since $x(U) \leq 1$, in any counterexample at least one job is started in $L \cup M$. If $x_{3,y-p_3} = 1$, then job 2 is the only job that can be started in $L \cup M$. Hence, in a counterexample for $(3, y - p_3)$ job 2 is started in M_2 and job 1 is started in U_1. Such a counterexample exists if and only if (3a) holds. If $y \leq l_2$ and $x_{3,y-p_3} = 1$, then job 1 and any job $j \in \{2, 4, \ldots, n\}$ with $M_j \neq \emptyset$ may be started in $L \cup M$. It is now not hard to see that there is a counterexample for $(3, y - p_3)$ if and only if (3) holds. Analogously, U_2 cannot be extended if and only if (4) holds.

Let $j \in \{4, \ldots, n\}$. It is easy to see that L_j cannot be extended if and only if there is a counterexample for $(j, y - p_j)$, where $y = \min\{l^*, l + p_j\}$. Suppose $y \leq l_2$. If $L_1 \cap U_1 \neq \emptyset$, then $x_{j,y-p_j} = x_{1l_2} = 1$ defines a counterexample for $(j, y - p_j)$. If $L_1 \cap U_1 = \emptyset$, i.e., $l_2 \leq u_3 - p_1$, then $x_{j,y-p_j} = x_{1l_2} = x_{3u_3} = 1$ defines such a counterexample. Hence, if $y \leq l_2$, then there is a counterexample for $(j, y - p_j)$. Now, suppose $y > l_2$. Since $x(U) \leq 1$, in any counterexample at least one job is started in $L \cup M$. If $x_{j,y-p_j} = 1$, then job 2 is the only job that can be started in $L \cup M$. It is now easy to see that a counterexample for $(j, y - p_j)$ contains job 1 and job 2, and we find that there is such a counterexample if and only if (5a) holds. We conclude that the intervals L_j with $j \in \{4, \ldots, n\}$ cannot be extended if and only if (5) holds. Analogously, the intervals U_j with $j \in \{4, \ldots, n\}$ cannot be extended if and only if (6) holds. □

Remark. It may seem more natural to define case (1a) as $l = l_1 < l_2 < l^*$ and $u = u_1 > u_2 > u^*$, and case (1b) as $l = l_1 < l_2 \leq l^*$ and $u = u_1 > u_3 \geq u^*$. Since under this definition Property 3.9 does not hold, we prefer the given definition.

This is illustrated by the following example. Let $n = 4$, $p_1 = 5$, $p_2 = 2$, $p_3 = 4$, and $p_4 = 2$. The following diagram represents a facet inducing inequality with structure (3.5) and $l = l_1 = u^* = 5$, $l_2 = l^* = 6$, $u_3 = 9$, and $u = u_1 = 11$, i.e., $u^* = l < l^*$. Note that $(1,4), (1,5) \in L \cap U$, i.e., x_{14} and x_{15} have coefficient 2.



3.5.3 Case (2)

Observe that in this case the conditions on l_j and u_j and Properties 3.4 and 3.5 do not completely determine the sets L and U. It turns out to be beneficial to introduce a notion slightly different from that of l^* and u^*, namely $l' = \min\{l_j \mid j \in \{3, \ldots, n\}\}$ and $u' = \max\{u_j \mid j \in \{3, \ldots, n\}\}$. Note that it is possible that $l_2 > l'$ or $u_1 < u'$, i.e., l' and u' do not necessarily coincide with l^* and u^* as defined in Property 3.5. We can however prove the following property that is similar to Property 3.5.

Property 3.10 *Let $x(V^1)+2x(V^2) \leq 2$ be facet inducing with $l = l_1$ and $u = u_2$.*
(a) *For all $j \in \{3, \ldots, n\}$ such that $L_j \neq \emptyset$, we have $l_j = l'$, and for all $j \in \{3, \ldots, n\}$ such that $L_j = \emptyset$, we have $l' - p_j \geq l$, i.e., $L_j = [l' - p_j, l]$ for all $j \in \{3, \ldots, n\}$.*
(b) *For all $j \in \{3, \ldots, n\}$ such that $U_j \neq \emptyset$, we have $u_j = u'$, and for all $j \in \{3, \ldots, n\}$ such that $U_j = \emptyset$, we have $u' \leq u - p_j$, i.e., $U_j = [u - p_j, u']$ for all $j \in \{3, \ldots, n\}$.* □

Proof. (a) By Property 3.4, $L_j \subseteq [l' - p_j, l]$ for all $j \in \{3, \ldots, n\}$. Assume without loss of generality that $l' = l_3$. Suppose that $L_j \neq [l' - p_j, l]$ for some $j \in \{4, \ldots, n\}$, say $L_4 \neq [l' - p_4, l]$. Clearly, if $l' - p_4 \geq l$, then $L_4 = \emptyset$ and hence $L_4 = [l' - p_4, l]$. Consequently, $l' - p_4 < l$ and $l_4 > l'$, i.e., $l' - p_4 + 1 \notin V_4$. Since $x(V^1)+2x(V^2) \leq 2$ is maximal, there is a counterexample for $(4, l' - p_4 + 1)$. Let $(j_1, s_1), (j_2, s_2) \in V$ be such that $x_{4,l'-p_4+1} = x_{j_1 s_1} = x_{j_2 s_2} = 1$ is a feasible schedule. Since $l' - p_4 + 1 \leq l$, the jobs j_1 and j_2 are started after job 4. Assume that job j_1 is started before job j_2. If job 2 does not occur in $\{j_1, j_2\}$, then, since $u_2 = u$, job j_2 may be replaced by job 2. So we may assume that job 2 occurs in $\{j_1, j_2\}$. It follows that one of the

jobs 1 and 3 does not occur in $\{j_1, j_2\}$; suppose this is job 3. It is now easy to see that $x_{3,l'-p_3+1} = x_{j_1 s_1} = x_{j_2 s_2} = 1$ is a feasible schedule. This schedule violates the inequality, which yields a contradiction. If job 1 does not occur in $\{j_1, j_2\}$, then we find a contradiction in the same way.

The proof of (b) is similar to that of (a). □

We next investigate the structure of the set M.

Property 3.11 *If $x(V^1) + 2x(V^2) \leq 2$ is facet inducing with $l = l_1$ and $u = u_2$, then $M_1 = [u' - p_1, l'] \cap [\min\{l_2, l'\}, u - p_1] \cap [l - p_1, u_1]$, $M_2 = [u' - p_2, l'] \cap [l, \max\{u_1, u'\} - p_2] \cap [l_2 - p_2, u]$, and $M_j = \emptyset$ for $j \in \{3, \ldots, n\}$.* □

Proof. Let $x(V^1) + 2x(V^2) \leq 2$ be facet inducing with $l = l_1$ and $u = u_2$. To determine M_1 we consider two cases: $l_2 \geq l'$ and $l_2 < l'$. Suppose $l_2 \geq l'$. Let job 1 be started in V. If it is possible to start a job in V after job 1, then, since $u_2 = u$, job 2 can be started in V after job 1. If it is possible to start a job in V before job 1, then, since $l' \leq l_2$, there exists a $j \in \{3, \ldots, n\}$ such that job j can be started in V before job 1. It easily follows that $M_1 = \emptyset$. Suppose $l_2 < l'$. By definition $M_1 \subseteq [l - p_1, u_1]$. If job 1 is started in M_1, then it should be possible to start job 2 in V before as well as after job 1, which implies that $M_1 \subseteq [\min\{l_2, l'\}, u - p_1]$. Furthermore, it should be impossible to start any job $j \in \{3, \ldots, n\}$ in V, and hence $M_1 \subseteq [u' - p_1, l']$. We conclude that $M_1 \subseteq [u' - p_1, l'] \cap [\min\{l_2, l'\}, u - p_1] \cap [l - p_1, u_1]$. If job 1 is started in period $s \in [u' - p_1, l'] \cap [\min\{l_2, l'\}, u - p_1] \cap [l - p_1, u_1]$, then, since $s \in [l - p_1, u_1]$ and $L_2 \cap U_2 = [\max\{u_1, u'\} - p_2, l]$, job 2 cannot be started in $L_2 \cap U_2$. Hence, if $l_2 < l'$, then $M_1 = [u' - p_1, l'] \cap [\min\{l_2, l'\}, u - p_1] \cap [l - p_1, u_1]$. Note that the intersection of these three intervals is empty if $l_2 \geq l'$. We conclude that $M_1 = [u' - p_1, l'] \cap [\min\{l_2, l'\}, u - p_1] \cap [l - p_1, u_1]$. Analogously $M_2 = [u' - p_2, l'] \cap [l, \max\{u_1, u'\} - p_2] \cap [l_2 - p_2, u]$.

Let $j \in \{3, \ldots, n\}$. If job j is started in V and it is possible to start a job in V before job j, then, since $l_1 = l$, job 1 can be started before job j. If it is possible to start a job in V after job j, then, since $u_2 = u$, job 2 can be started after job j. It follows that $M_j = \emptyset$. □

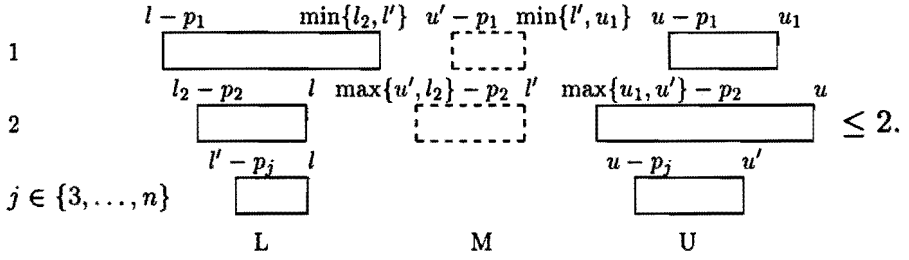
Properties 3.4, 3.10, and 3.11 completely determine the LMU-structure of a facet inducing inequality $x(V^1) + 2x(V^2) \leq 2$ with $l = l_1$ and $u = u_2$. As in the previous two cases, we prefer to use a different representation of the set M, in order to emphasize the inherent structure of the intervals M_j. It is easy to show that if $x(V^1) + 2x(V^2) \leq 2$ is facet inducing with $l = l_1$ and $u = u_2$, then $[l' - p_2, l] \subseteq L_2$ and $[u - p_1, u'] \subseteq U_1$. It is now not hard to see that Properties 3.4, 3.10, and 3.11 can be combined to give the following theorem.

Theorem 3.14 *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:*

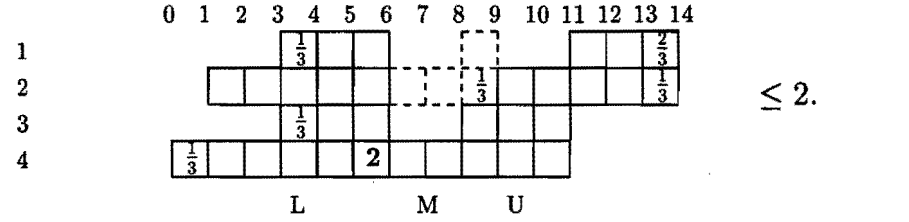
$$
\begin{array}{ll}
L_1 = [l - p_1, \min\{l_2, l'\}], & M_1 = [u' - p_1, \min\{l', u_1\}] \setminus (L_1 \cup U_1), \\
L_2 = [l_2 - p_2, l], & M_2 = [\max\{u', l_2\} - p_2, l'] \setminus (L_2 \cup U_2), \\
L_j = [l' - p_j, l], & M_j = \emptyset, \\
 & \\
U_1 = [u - p_1, u_1], & \\
U_2 = [\max\{u_1, u'\} - p_2, u], & \qquad (3.6) \\
U_j = [u - p_j, u'] \qquad (j \in \{3, \ldots, n\}), &
\end{array}
$$

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

This theorem states 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:



Example 3.4 Let $n = 4$, $p_1 = 3$, $p_2 = 5$, $p_3 = 6$, and $p_4 = 9$. The inequality with LMU-structure (3.6) and $l = l_1 = 6$, $l_2 = 6$, $l^* = 9$, $u^* = 11$, $u_1 = 6$, and $u_2 = u = 14$ is given by the following diagram:



Note that $(4, 6) \in L \cap U$, i.e., x_{46} has coefficient 2. The fractional solution $x_{14} = x_{29} = x_{2,14} = x_{34} = x_{41} = \frac{1}{3}$ and $x_{1,14} = \frac{2}{3}$ violates this inequality. It is easy to check that this solution satisfies all inequalities with structure (3.3).

Sufficient conditions are given by the following theorem.

Theorem 3.15 *A valid inequality* $x(V^1) + 2x(V^2) \leq 2$ *with* $l = l_1$ *and* $u = u_2$ *and LMU-structure* (3.6) *that is nondecomposable and maximal is facet inducing.*

Proof. The proof proceeds along the same lines as that of Theorem 3.7. The first and second set of direction vectors can be determined as in the proof of Theorem 3.7. We consider the third set of direction vectors, i.e., we determine the $|V| - |V^2| - 1$ direction vectors $d_{j_1 s_1} = 1, d_{j_2 s_2} = -1$ with $(j_1, s_1), (j_2, s_2) \in V \setminus V^2$ in such a way that the undirected graph G whose vertices are the elements of $V \setminus V^2$ and whose edges are given by the pairs $\{(j_1, s_1), (j_2, s_2)\}$ corresponding to the determined direction vectors is a spanning tree.

First, we determine direction vectors that correspond to edges in G within the sets $\{(j,s) \mid s \in (L_j \cup M_j) \setminus U_j\}$ and $\{(j,s) \mid s \in U_j \setminus L_j\}$. For $s-1, s \in L_1 \setminus U_1$, $d_{1,s-1} = -1, d_{1s} = 1$ is a direction vector by $(2,u)$. If $M_1 \neq \emptyset$, then $d_{1,\min\{l_2,l'\}} = -1, d_{1m} = 1$ is a direction vector by $(2,u)$, where m is the minimum of M_1; for $s-1, s \in M_1$, $d_{1,s-1} = -1, d_{1s} = 1$ is also a direction vector by $(2,u)$. Let $s-1, s \in U_1 \setminus L_1$. Note that $s-1 > \min\{l_2, l'\}$. If $l_2 \leq l'$, then $d_{1,s-1} = -1, d_{1s} = 1$ is a direction vector by $(2, l_2 - p_2 + 1)$. If $l_2 > l'$, then $d_{1,s-1} = -1, d_{1s} = 1$ is a direction vector by $(j, l' - p_j + 1)$, where $j \in \{3, \ldots, n\}$ is such that $l_j = l'$. In the same way we find that for $s-1, s \in L_2 \setminus U_2$, $d_{2,s-1} = -1, d_{2s} = 1$ is a direction vector by $(1, u_1)$ if $u_1 \geq u'$ and by (j, u') if $u_1 < u'$, where $j \in \{3, \ldots, n\}$ is such that $u_j = u'$. Observe that, if job 2 is started in M_2, then job 1 is the only job that can be started before or after job 2. We find that, if $L_2 \neq \emptyset$ and $M_2 \neq \emptyset$, then $d_{2l} = -1, d_{2m} = 1$ is a direction vector by $(1, u_1)$, where m is the minimum of M_2. For $s-1, s \in M_2$, $d_{2,s-1} = -1, d_{2s} = 1$ is also a direction vector by $(1, u_1)$. Furthermore, for $s-1, s \in U_2 \setminus L_2$, $d_{2,s-1} = -1, d_{2s} = 1$ is a direction vector by $(1, l - p_1 + 1)$. Now, let $j \in \{3, \ldots, n\}$. Note that $M_j = \emptyset$. Clearly, for $s-1, s \in L_j \setminus U_j$, $d_{j,s-1} = -1, d_{js} = 1$ is a direction vector by $(2,u)$, and for $s-1, s \in U_j \setminus L_j$, $d_{j,s-1} = -1, d_{js} = 1$ is a direction vector by $(1, l - p_1 + 1)$.

Second, we determine direction vectors that correspond to edges in G between the sets $\{(j,s) \mid s \in (L_j \cup M_j) \setminus U_j\}$ belonging to different jobs and between sets $\{(j,s) \mid s \in U_j \setminus L_j\}$ belonging to different jobs. It is easy to see that for $j \in \{3, \ldots, n\}$ such that $L_j \neq \emptyset$, $d_{1,l-p_1+1} = -1, d_{l'-p_j+1} = 1$ is a direction vector by $(2,u)$. For $j \in \{3, \ldots, n\}$ such that $U_j \neq \emptyset$, $d_{2u} = -1, d_{ju'} = 1$ is a direction vector by $(1, l-p_1+1)$. We still have to determine a direction vector that corresponds to an edge in G between $\{(2,s) \mid s \in (L_2 \cup M_2) \setminus U_2\}$ and one of the sets $\{(j,s) \mid s \in (L_j \cup M_j) \setminus U_j\}$ with $j \in \{1, 3, \ldots, n\}$ and a direction vector that corresponds to an edge in G between $\{(1,s) \mid s \in U_1 \setminus L_1\}$ and one of the sets $\{(j,s) \mid s \in U_j \setminus L_j\}$ with $j \in \{2, \ldots, n\}$. Observe that, since $x(V^1) + 2x(V^2) \leq 2$ is nondecomposable, we must have $(L_2 \cup M_2) \setminus U_2 \neq \emptyset$. We define $W = \{(1,s) \mid s \in V_1\} \cup \{(2,s) \mid s \in L_2 \cap U_2\} \cup \{(j,s) \mid j \in \{3, \ldots, n\}, s \in L_j\}$ and $W' = \{(1,s) \mid s \in L_1 \cap U_1\} \cup \{(2,s) \mid s \in V_2\} \cup \{(j,s) \mid j \in \{3, \ldots, n\}, s \in U_j\}$. Note that $x(W) + x(W') = x(V^1) + 2x(V^2)$. Since $x(V^1) + 2x(V^2) \leq 2$ is nondecomposable, there exists a feasible schedule such that $x(W) = 2$, or there exists a feasible schedule such that $x(W') = 2$. Suppose that there exists a feasible schedule such that $x(W) = 2$. It is easy to see that in such a schedule some job $j \in \{3, \ldots, n\}$, say job j_1, is started in L_j, and that job 1

is started after job j_1. It easily follows that $x_{j_1,l'-p_{j_1}+1} = x_{1u_1} = 1$ defines a feasible schedule. If $L_2 \neq \emptyset$, then, since $[l' - p_2, l] \subseteq L_2$, we have $l_2 \leq l'$. It follows that $y_{2,l_2-p_2+1} = y_{1u_1} = 1$ defines a feasible schedule. If job 2 is started in M_2, then job 1 can be started after job 2. Hence, if $M_2 \neq \emptyset$, then $y_{2,l_2-p_2+1} = y_{1u_1} = 1$ also defines a feasible schedule. We conclude that $d_{2,l_2-p_2+1} = -1, d_{j_1,l'-p_{j_1}+1} = 1$ is a direction vector by $(1, u_1)$. As $x_{j_1,l'-p_{j_1}+1} = x_{1u_1} = 1$ defines a feasible schedule, $y_{j_1,l'-p_{j_1}+1} = y_{2u} = 1$ clearly also defines a feasible schedule. We find that $d_{1u_1} = -1, d_{2u} = 1$ is a direction vector by $(j_1, l' - p_{j_1} + 1)$.

Suppose that there is a feasible schedule such that $x(W') = 2$. It is now not hard to see that $x_{2,l_2-p_2+1} = x_{ju'} = 1$ is a feasible schedule for some job $j \in \{3, \ldots, n\}$, say for job j_1, such that $U_j \neq \emptyset$. Similarly to the previous case, we find that $d_{2,l_2-p_2+1} = 1, d_{1,l-p_1+1} = -1$ is a direction vector by (j_1, u') and that $d_{j_1,u'} = 1, d_{1u_1} = -1$ is a direction vector by $(2, l_2 - p_2 + 1)$.

Finally, we determine a direction vector that corresponds to an edge in G between $L \cup M$ and U. Since $x(V^1) + 2x(V^2) \leq 2$ is nondecomposable and $x(U) \leq 1$ is a valid inequality, there is a feasible schedule such that $x(L) + x(M) = 2$. It is easy to see that in such a schedule job 1 and job 2 are started in $L \cup M$. Let $x_{1s_1} = x_{2s_2} = 1$ be such a schedule. Since $s_1 \in L_1 \cup M_1$, $y_{1s_1} = y_{2u} = 1$ is also a feasible schedule. It follows that $d_{2s_2} = 1, d_{2u} = -1$ is a direction vector by $(1, s_1)$.

It is easy to see that the determined direction vectors form a spanning tree of G and we have hence determined $|V| - |V^2| - 1$ linearly independent direction vectors. □

Specific necessary and sufficient conditions for a valid inequality $x(V^1)+2x(V^2) \leq 2$ with $l = l_1$ and $u = u_2$ that has LMU-structure (3.6) to be nondecomposable and maximal are given in the following two theorems.

Theorem 3.16 *A valid inequality $x(V^1)+2x(V^2) \leq 2$ with $l = l_1$ and $u = u_2$ that has LMU-structure* (3.6) *is nondecomposable if and only if $M_1 \neq \emptyset$ or $M_2 \neq \emptyset$, and $l' < u_1$ or $l_2 < u'$.*

Proof. Let $x(V^1) + 2x(V^2) \leq 2$ be a valid inequality with $l = l_1$ and $u = u_2$ that has LMU-structure (3.6).

Suppose that $x(V^1) + 2x(V^2) \leq 2$ is nondecomposable. Since $x(L) \leq 1$ and $x(U) \leq 1$ are valid inequalities, we must have $M_j \neq \emptyset$ for some $j \in \{1, \ldots, n\}$; since by definition of LMU-structure (3.6), $M_j = \emptyset$ for all $j \in \{3, \ldots, n\}$, it follows that $M_1 \neq \emptyset$ or $M_2 \neq \emptyset$. Suppose that $l' \geq u_1$ and $l_2 \geq u'$. We define $W = \{(1, s) \mid s \in V_1\} \cup \{(2, s) \mid s \in L_2 \cap U_2\} \cup \{(j, s) \mid j \in \{3, \ldots, n\}, s \in L_j\}$ and $W' = \{(1, s) \mid s \in L_1 \cap U_1\} \cup \{(2, s) \mid s \in V_2\} \cup \{(j, s) \mid j \in \{3, \ldots, n\}, s \in U_j\}$. Clearly $x(W) + x(W') = x(V^1) + 2x(V^2)$. Since $l' \geq u_1$, it follows that $\sum_{s \in V_1} x_{1s} + \sum_{j=3}^{n} \sum_{s \in L_j} x_{js} \leq 1$ is valid, and hence $x(W) \leq 1$ is valid. In the same

way, it follows from $l_2 \geq u'$ that $x(W') \leq 1$ is a valid inequality, which yields a contradiction. We conclude that $l' < u_1$ or $l_2 < u'$.

Suppose that $M_1 \neq \emptyset$ or $M_2 \neq \emptyset$, and $l' < u_1$ or $l_2 < u'$. Let W and W' be such that $x(W) + x(W') = x(V^1) + 2x(V^2)$ and $x(W) \leq 1$ and $x(W') \leq 1$ are valid inequalities. We assume without loss of generality that $(1, l - p_1 + 1) \in W$. Since $l < u$, $x_{1,l-p_1+1} = x_{2u} = 1$ is a feasible schedule. As $x(W) \leq 1$ is valid and $(1, l - p_1 + 1) \in W$, it follows that $(2, u) \in W'$. We conclude that, since $l < u$ and $(1, l - p_1 + 1) \in W$, we have $(2, u) \in W'$. If $L_j \neq \emptyset$ for some $j \in \{3, \ldots, n\}$, then $l' < u$. Since $l' < u$ and $(2, u) \in W'$, $(j, l' - p_j + 1) \in W$ for all $j \in \{3, \ldots, n\}$ such that $L_j \neq \emptyset$. In the same way, if $U_j \neq \emptyset$ for some $j \in \{3, \ldots, n\}$, then $l < u'$. Since $l < u'$ and $(1, l - p_1 + 1) \in W$, $(j, u') \in W'$ for all $j \in \{3, \ldots, n\}$ such that $U_j \neq \emptyset$. By assumption $M_1 \neq \emptyset$ or $M_2 \neq \emptyset$. We consider each of these two cases.

Suppose $M_1 \neq \emptyset$. To prove that $x(V^1) + 2x(V^2) \leq 2$ is nondecomposable, we first show that $(2, l_2 - p_2 + 1) \in W$. By assumption $l' < u_1$ or $l_2 < u'$. Suppose $l' < u_1$. Note that $U_1 \neq \emptyset$ and $L_j \neq \emptyset$ for some $j \in \{3, \ldots, n\}$. Since $l' < u_1$ and $(j, l' - p_j + 1) \in W$ for all $j \in \{3, \ldots, n\}$ such that $L_j \neq \emptyset$, it follows that $(1, u_1) \in W'$. Observe that if job 1 is started in M_1, then job 2 can be started in V before job 1. It follows that $x_{2,l_2-p_2+1} = x_{1s} = 1$ is a feasible schedule for all $s \in M_1$, and hence $x_{2,l_2-p_2+1} = x_{1u_1} = 1$ is a feasible schedule. Since $(1, u_1) \in W'$, we find that $(2, l_2 - p_2 + 1) \in W$. Now, suppose $l_2 < u'$. Note that $L_2 \neq \emptyset$ and $U_j \neq \emptyset$ for some $j \in \{3, \ldots, n\}$. Since $l_2 < u'$ and $(j, u') \in W'$ for all $j \in \{3, \ldots, n\}$ such that $U_j \neq \emptyset$, it also follows that $(2, l_2 - p_2 + 1) \in W$. If $s \in M_1$, then, as $x_{2,l_2-p_2+1} = x_{1s} = 1$ is a feasible schedule and $(2, l_2 - p_2 + 1) \in W$, we have $(1, s) \in W'$. We find that $x_{1s} = x_{2u} = 1$ is a feasible schedule such that $x(W') = 2$, which yields a contradiction. We conclude that from $(2, l_2 - p_2 + 1) \in W$, $(2, u) \in W'$, and $M_1 \neq \emptyset$, it follows that $x(V^1) + 2x(V^2) \leq 2$ is nondecomposable.

Analogously, if $M_2 \neq \emptyset$, then $(1, u_1) \in W'$, and from $(1, l - p_1 + 1) \in W$, $(1, u_1) \in W'$, and $M_2 \neq \emptyset$, it follows that $x(V^1) + 2x(V^2) \leq 2$ is nondecomposable. □

Theorem 3.17 *A valid inequality $x(V^1) + 2x(V^2) \leq 2$ with $l = l_1$ and $u = u_2$ that has LMU-structure* (3.6) *is maximal if and only if each of the following ten hold:*

(1) *If $u_1 < l'$ and $u_1 < u - p_1$, then $u_1 \geq u'$ and $L_2 \cap U_2 \neq \emptyset$;*

(2) *If $l_2 > u'$ and $l_2 - p_2 > l$, then $l_2 \leq l'$ and $L_1 \cap U_1 \neq \emptyset$;*

One of the following holds:
(3a) *$L_j \cap U_j \neq \emptyset$ for some $j \in \{2, \ldots, n\}$;*
(3b) *$l \leq u' - p_2$, $L_2 \neq \emptyset$, and $U_j \neq \emptyset$ for some $j \in \{3, \ldots, n\}$;*
(3c) *$l \leq u - \min\{p_j \mid j \in \{3, \ldots, n\}, L_j \neq \emptyset\}$ and $L_j \neq \emptyset$ for some $j \in \{3, \ldots, n\}$;*

One of the following holds:
(4a) $L_j \cap U_j \neq \emptyset$ *for some* $j \in \{1, 3, \ldots, n\}$;
(4b) $l' \leq u - p_1$ *and* $L_j \neq \emptyset$ *for some* $j \in \{3, \ldots, n\}$;
(4c) $l \leq u - \min\{p_j \mid j \in \{3, \ldots, n\}, U_j \neq \emptyset\}$ *and* $U_j \neq \emptyset$ *for some* $j \in \{3, \ldots, n\}$;

If $l_2 > l$, *then one of the following holds:*
(5a) $L_1 \cap U_1 \neq \emptyset$;
(5b) $\min\{l_2, l + p_2\} \leq u' - p_1$ *and* $U_j \neq \emptyset$ *for some* $j \in \{3, \ldots, n\}$;

If $l_2 = l$, *then one of the following holds:*
(6a) $L_j \cap U_j \neq \emptyset$ *for some* $j \in \{1, 3, \ldots, n\}$;
(6b) $l \leq u_1 - \min\{p_j \mid j \in \{3, \ldots, n\}, L_j \neq \emptyset\}$, $U_1 \neq \emptyset$, *and* $L_j \neq \emptyset$ *for some* $j \in \{3, \ldots, n\}$;
(6c) $l \leq u' - \min\{p_j \mid j \in \{1, 3, \ldots, n\}, L_j \neq \emptyset\}$ *and* $U_j \neq \emptyset$ *for some* $j \in \{3, \ldots, n\}$;

If $u_1 < u$, *then one of the following holds:*
(7a) $L_2 \cap U_2 \neq \emptyset$;
(7b) $l' \leq \max\{u_1, u - p_1\} - p_2$ *and* $U_j \neq \emptyset$ *for some* $j \in \{3, \ldots, n\}$;

If $u_1 = u$, *then one of the following holds:*
(8a) $L_j \cap U_j \neq \emptyset$ *for some* $j \in \{2, \ldots, n\}$;
(8b) $l_2 \leq u - \min\{p_j \mid j \in \{3, \ldots, n\}, U_j \neq \emptyset\}$, $L_2 \neq \emptyset$, *and* $U_j \neq \emptyset$ *for some* $j \in \{3, \ldots, n\}$;
(8c) $l' \leq u - \min\{p_j \mid j \in \{2, \ldots, n\}, U_j \neq \emptyset\}$ *and* $U_j \neq \emptyset$ *for some* $j \in \{3, \ldots, n\}$;

For all $j \in \{3, \ldots, n\}$, *one of the following holds:*
(9a) $\min\{l', l + p_j\} \leq l_2$;
(9b) $\min\{l', l + p_j\} \leq u - p_1$ *and* $M_1 \neq \emptyset$;
(9c) $\min\{l', l + p_j\} \leq u_1 - p_2$ *and* $M_2 \neq \emptyset$;

For all $j \in \{3, \ldots, n\}$, *one of the following holds:*
(10a) $u_1 \leq \max\{u', u - p_j\}$;
(10b) $l_2 \leq \max\{u', u - p_j\} - p_1$ *and* $M_1 \neq \emptyset$;
(10c) $l \leq \max\{u', u - p_j\} - p_2$ *and* $M_2 \neq \emptyset$.

Proof. Let $x(V^1) + 2x(V^2) \leq 2$ be a valid inequality with $l = l_1$ and $u = u_2$ that has LMU-structure (3.6).

We show that M_1 cannot be extended if and only if (1) holds. Suppose that (1) holds. Observe that, if $l_2 \geq l'$, then $M_1 = \emptyset$, and we have to show that there

is a counterexample for all $(1, s)$ with $s \in [l', u - p_1]$. If $l_2 < l'$, then we have to show that there is a counterexample for all $(1, s)$ with $s \in [l_2, u' - p_1]$ or $s \in [\min\{l', u_1\}, u - p_1]$. If $l' < u - p_1$, then $x_{j,l'-p_j+1} = x_{1s} = x_{2u} = 1$ defines a counterexample for all $(1, s)$ with $s \in [l', u - p_1]$, where $j \in \{3, \ldots, n\}$ is such that $l_j = l'$. If $l_2 \geq l'$, this implies that M_1 cannot be extended. Now, suppose $l_2 < l$. If $u_1 < l'$ and $u_1 < u - p_1$, then by (1) $u_1 - p_2 + 1 \in L_2 \cap U_2$ and we find that $x_{2,u_1-p_2+1} = x_{1s} = 1$ defines a counterexample for all $(1, s)$ with $s \in [u_1, u - p_1]$. It follows that there is a counterexample for all $(1, s)$ with $s \in [\min\{l', u_1\}, u - p_1]$. Clearly, if $l_2 < u' - p_1$, then $x_{l_2-p_2+1} = x_{1s} = x_{ju'} = 1$ defines a counterexample for all $(1, s)$ with $s \in [l_2, u' - p_1]$, where $j \in \{3, \ldots, n\}$ is such that $u_j = u'$. We conclude that M_1 cannot be extended. It is not hard to see that, if (1) does not hold, then M_1 can be extended by $\max\{u' - p_1, u_1\} + 1$. Hence M_1 cannot be extended if and only if (1) holds. Analogously, M_2 cannot be extended if and only if (2) holds.

In the same way as in the proof of Theorem 3.9, it can be shown that L_1 cannot be extended if and only if (3) holds. Analogously, U_2 cannot be extended if and only if (4) holds.

Clearly, L_2 cannot be extended if and only if there is a counterexample for $(2, y - p_2)$, where $y = \min\{l_2, l + p_2\}$. It is not hard to see that if $y > l$, i.e., $l_2 > l$, then any counterexample for $(2, y - p_2)$ contains job 1. Such a counterexample exists if and only if (5) holds. Now, suppose $y = l$, i.e., $l_2 = l$. We assume without loss of generality that $p_3 \geq p_4 \geq \ldots \geq p_n$. Note that by this assumption we have that if $L_j \neq \emptyset$ for some $j \in \{3, \ldots, n\}$, then $L_3 \neq \emptyset$, and if $U_j \neq \emptyset$ for some $j \in \{3, \ldots, n\}$, then $U_3 \neq \emptyset$. Clearly, there is a counterexample for $(2, l - p_2)$ defined by $x_{2,l-p_2} = x_{js} = 1$ with $(j, s) \in V^2$ if and only if (6a) holds. We show that there is a counterexample for $(2, l - p_2)$ defined by $x_{2,l-p_2} = x_{j_1 s_1} = x_{j_2 s_2} = 1$ with $(j_1, s_1), (j_2, s_2) \in V \setminus V^2$ if and only if (6b) and (6c) hold. If (6b) holds, then $x_{2,l-p_2} = x_{j_1 l} = x_{1u_1} = 1$ defines a counterexample for $(2, l - p_2)$, where $j_1 \in \{3, \ldots n\}$ is such that $L_{j_1} \neq \emptyset$ and $p_{j_1} = \min\{p_j \mid j \in \{3, \ldots, n\}, L_j \neq \emptyset\}$. Now, suppose (6c) holds. Note that $U_3 \neq \emptyset$. Let $j_1 \in \{1, 3, \ldots, n\}$ be such that $p_{j_1} = \min\{p_j \mid j \in \{1, 3, \ldots, n\}, L_j \neq \emptyset\}$. If $j_1 \neq 3$, then $x_{2,l-p_2} = x_{j_1 l} = x_{3u'} = 1$ defines a counterexample for $(2, l - p_2)$. If $j_1 = 3$, then, since $L_1 \neq \emptyset$ and $p_{j_1} = \min\{p_j \mid j \in \{1, 3, \ldots, n\}, L_j \neq \emptyset\}$, we have $p_1 \geq p_3$. Note that since $x(V^1) + 2x(V^2) \leq 2$ has LMU-structure (3.6), we have $[u - p_1, u'] \subseteq U_1$. Since $U_3 \neq \emptyset$ and $[u - p_1, u'] \subseteq U_1$, it follows that $U_1 \neq \emptyset$ and hence $u_1 \geq u'$. It follows that (6b) holds, and hence there is a counterexample for $(2, l - p_2)$. Now, let $x_{2,l-p_2} = x_{j_1 s_1} = x_{j_2 s_2} = 1$ with $(j_1, s_1), (j_2, s_2) \in V \setminus V^2$ and $s_1 < s_2$ define a counterexample for $(2, l - p_2)$. It is easy to see that if $j_2 = 1$, then (6b) holds, and if $j_2 \in \{3, \ldots, n\}$, then (6c) holds. We find that, if $y = l$, then L_2 cannot be extended if and only if (6) holds. We conclude that L_2 cannot be extended if and only if (5) and (6) hold. Analogously, U_1 cannot be extended if and only if (7) and (8) hold.

In the same way as in the proof of Theorem 3.9, it can be shown that the intervals L_j with $j \in \{3, \ldots, n\}$ cannot be extended if and only if (9) holds. Analogously,

the intervals U_j with $j \in \{3, \ldots, n\}$ cannot be extended if and only if (10) holds. □

3.6 Related research

As mentioned in the introduction, Sousa and Wolsey [1992] and Crama and Spieksma [1991] have also studied the time-indexed formulation of single machine scheduling problems. In this section, we briefly indicate the relation between their research and our research.

Sousa and Wolsey present three classes of valid inequalities. The first class consists of inequalities with right-hand side 1, and the second and third class consist of inequalities with right-hand side $k \in \{2, \ldots, n-1\}$. Each class of inequalities is derived by considering a set of jobs and a certain time period. The right-hand side of the resulting inequality is equal to the cardinality of the considered set of jobs.

They show that the inequalities in the first class, which is exactly the class of inequalities with structure (3.3), are all facet inducing. In Section 3, we have complemented this result by showing that all facet inducing inequalities with right-hand side 1 are in this class. With respect to the other two classes of valid inequalities we make the following observations. Any inequality in the second class that has right-hand side 2 can be lifted to an inequality with LMU-structure (3.4) if $p_{k_1} \neq p_{k_2}$ and to an inequality with LMU-structure (3.6) if $p_{k_1} = p_{k_2}$, where $\{k_1, k_2\}$ is the set of jobs considered. Any inequality in the third class that has right-hand side 2 can be written as the sum of two valid inequalities with right-hand side 1. For either of the two classes, Sousa and Wolsey give an example of a fractional solution that violates one of the inequalities in the class and for which they claim that it does not violate any valid inequality with right-hand side 1. We found that in both cases the latter claim is false.

Crama and Spieksma investigate the special case of equal processing times. They completely characterize all facet inducing inequalities with right-hand side 1 and present two other classes of facet inducing inequalities with right-hand side $k \in \{2, \ldots, n-1\}$.

Our characterization of all facet inducing inequalities with right-hand side 1 was found independently and generalizes their result. The inequalities in their second class that have right-hand side 2 are special cases of the inequalities with LMU-structure (3.6), and the inequalities in their third class that have right-hand side 2 are special cases of the inequalities with LMU-structure (3.4). In addition to the facet inducing inequalities reported in their paper, they have identified other classes of facet inducing inequalities with right-hand side 2 [Spieksma 1991].

Chapter 4

Separation and computation

4.1 Introduction

To obtain insight in the effectiveness of the classes of facet inducing inequalities discussed in the previous chapter, we have developed separation algorithms that identify violated inequalities in these classes. Based on these separation algorithms, we have developed a branch-and-cut algorithm for $1|r_j|\sum w_jC_j$, which is known to be $\mathcal{NP}$-hard. We investigate several branching strategies, row management policies, and primal heuristics that can be embedded in the branch-and-cut algorithm, and we show the influence of the choice of these strategies on the performance of the algorithm. We also compare the performance of our algorithm to the performance of algorithms presented in related work.

4.2 Separation

The separation algorithms are based on the following observation. If an inequality in one of the identified classes is violated, then there is also a violated inequality such that some specific subset of the variables that occur in the inequality is positive. We may hence assume that in a violated inequality these variables are positive. For example, if there is a violated inequality with right-hand side 1, i.e., with structure (3.3), then there is a violated inequality with $x_{1,l-p_1+1} > 0$ and $x_{1u} > 0$. It turns out that such a subset of positive variables completely determines the inequality. We can therefore restrict ourselves to enumerating all combinations of positive variables in the current LP-solution that constitute such a subset, and check the corresponding inequality for violation. The resulting enumeration algorithms are polynomial in the number of positive variables in the current LP-solution, while the number of facet inducing inequalities with right-hand side 1 or 2 is polynomial in the planning horizon T.

This section is organized as follows. We first consider facet inducing inequalities

with right-hand side 1, and after that facet inducing inequalities with right-hand side 2. In each case we show in the form of several lemmas which variables in a violated inequality may assumed to be positive. Then we describe the resulting separation algorithms. Recall that the time-indexed formulation is given by:

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

subject to

$$\sum_{t=1}^{T-p_j+1} x_{jt} = 1 \qquad (j = 1, \ldots, n), \tag{4.1}$$

$$\sum_{j=1}^{n} \sum_{s=t-p_j+1}^{t} x_{js} \leq 1 \qquad (t = 1, \ldots, T), \tag{4.2}$$

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

In the sequel, $\tilde{x}$ denotes the current LP-solution. As we start with the LP-relaxation of the original formulation, $\tilde{x}$ satisfies (4.1) and (4.2).

4.2.1 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 (3.3).

Lemma 4.1 *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 (4.1), $\tilde{x}_{js'} = 0$ 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 $x_{j's'} > 0$ 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 (4.2) 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 $x(V) \leq 1$. □

Lemma 4.2 *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 (4.2), $x(V) \leq 1$ is such that $l < u$. 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 $\tilde{x}_{1,l-p_1+1} = 0$, $\tilde{x}$ also violates the new inequality. We conclude that if $\tilde{x}_{1,l-p_1+1} = 0$, then by increasing l we obtain another violated inequality and we may hence assume $\tilde{x}_{1,l-p_1+1} > 0$. In the same way we can show that we may assume $\tilde{x}_{1u} > 0$. □

Since the current LP-solution $\tilde{x}$ satisfies inequalities (4.1), for a violated inequality $x(V) \leq 1$ we must have $V_j \neq \emptyset$ for some $j \in \{2, \ldots, n\}$ and hence $u - \max\{p_j \mid j \in \{2, \ldots, n\}\} < l$. A facet inducing inequality $x(V) \leq 1$ is determined by a job j and numbers l and u. The number of such inequalities is of order $nTp_{\max}$, where $p_{\max}$ denotes the maximum processing time. The previous lemma shows that the number of inequalities that we have to check is bounded by the square of the number of fractional variables in the current LP-solution. We find the following separation algorithm.

Sepa1($\tilde{\mathbf{x}}$)

begin
  for all jobs $j \in \{1, \ldots, n\}$ **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.

4.2.2 Separation algorithms for facet inducing inequalities with right-hand side 2

Facet inducing inequalities with right-hand side 2 are inequalities with structure (3.4), (3.5), or (3.6). We first show which variables may assumed to be positive in a violated inequality with one of these structures. If a facet inducing inequality $x(V^1) + 2x(V^2) \leq 2$ is violated and the set of positive variables yields an inequality that is nondecomposable, then this inequality will be identified by the separation algorithm. Because of the complexity of the necessary and sufficient conditions for inequalities with structure (3.4), (3.5), or (3.6) to be maximal, we do not guarantee that the identified inequality is maximal; it may hence not be facet inducing. If the set of positive variables yields an inequality that is the sum of two valid inequalities

with right-hand side 1, then no violated inequality will be identified, but there clearly exists a violated inequality with right-hand side 1.

Lemma 4.3 *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 $(j,s) \in V$.*

Proof. Let $\tilde{x}$ violate $x(V^1)+2x(V^2) \leq 2$ and satisfy all valid inequalities $x(W) \leq 1$ with $W \subseteq V$. We show that if $\tilde{x}_{js} = 1$ for some $(j,s) \in V$, then $\tilde{x}(V^1) + 2\tilde{x}(V^2) \leq 2$, i.e., $\tilde{x}$ does not violate $x(V^1) + 2x(V^2) \leq 2$. Note that any inequality $x(V^1) + 2x(V^2) \leq 2$ can be represented by sets L, M and U.

If $\tilde{x}_{js} = 1$ for some (j,s) such that $s \in M_j$, then, by definition of the set M, there is a job i such that job i is the only job that can be started before or after job j and hence $\tilde{x}_{j's'} = 0$ for all $(j',s') \in V$ with $j' \in \{1,2,\ldots,n\} \setminus \{i,j\}$. It is now easy to see that $\tilde{x}$ does not violate $x(V^1) + 2x(V^2) \leq 2$.

Suppose that $\tilde{x}_{js} = 1$ for some (j,s) such that $s \in L_j$. Clearly, $\tilde{x}_{js'} = 0$ for all $s \neq s'$. By definition of the set L, $\tilde{x}_{j's'} = 0$ for all $(j',s') \neq (j,s)$ with $s' \in L_{j'}$. Define $J_M(j)$ as the set of jobs i such that if job i is started in M_i, then job j is the only job that can be started before or after job j. Since $\tilde{x}_{js} = 1$, $\tilde{x}_{j's'} = 0$ for all (j',s') with $s' \in M_{j'}$ and $j' \notin J_M(j)$. To show that $\tilde{x}(V^1)+2\tilde{x}(V^2) \leq 2$ we still have to show that $\sum_{i \in J_M(j)} \sum_{s \in M_i} \tilde{x}_{is} + \sum_{i \neq j} \sum_{s \in U_i} \tilde{x}_{is} \leq 1$. By definition of the set U, $\sum_{i \neq j} \sum_{s \in U_i} x_{is} \leq 1$ is a valid inequality. Furthermore, if job $i \in J_M(j)$ is started in M_i, then any job $j' \neq j$ cannot be started after job i and hence $\sum_{i \neq j} \sum_{s \in U_i} x_{is} = 0$. It is now easy to see that $\sum_{i \in J_M(j)} \sum_{s \in M_i} x_{is} + \sum_{i \neq j} \sum_{s \in U_i} x_{is} \leq 1$ is a valid inequality. By assumption $\tilde{x}$ satisfies this inequality and we conclude that $\tilde{x}(V^1)+2\tilde{x}(V^2) \leq 2$, i.e., $\tilde{x}$ does not violate $x(V^1) + 2x(V^2) \leq 2$.

Analogously, $\tilde{x}$ does not violate $x(V^1) + 2x(V^2) \leq 2$, if $\tilde{x}_{js} = 1$ for some (j,s) with $s \in U_j$. □

Lemma 4.4 *If $\tilde{x}$ violates the inequality $x(V^1)+2x(V^2) \leq 2$ with structure* (3.4), (3.5), *or* (3.6) *then we may assume that one of the following holds:*
(i) *$x(V^1) + 2x(V^2) \leq 2$ has structure* (3.4) *or* (3.5), *$\tilde{x}_{1,l-p_1+1} > 0$, and $\tilde{x}_{1u} > 0$;*
(ii) *$x(V^1) + 2x(V^2) \leq 2$ has structure* (3.6), *$\tilde{x}_{1,l-p_1+1} > 0$, and $\tilde{x}_{2u} > 0$.*

Proof. Let $\tilde{x}$ violate $x(V^1) + 2x(V^2) \leq 2$ with structure (3.4) or (3.5). Suppose that $\tilde{x}_{1,l-p_1+1} = 0$. We show that by increasing l we obtain another violated inequality. If $l_2 > l+1$ and we increase l by 1, then the variable $x_{1,l-p_1+1}$ is removed from the inequality and some of the variables $x_{j,l+1}$ with $j \in \{2,\ldots,n\}$ are added to the inequality. We obtain another violated inequality with the same structure. Suppose that $l_2 = l+1$. If $x(V^1)+2x(V^2) \leq 2$ has structure (3.4) and we increase l by 1 in $x(V^1)+2x(V^2) \leq 2$, then the new inequality has $l = l_2$ and by interchanging

jobs 1 and 2 we find that this inequality has structure (3.6). If $x(V^1) + 2x(V^2) \leq 2$ has structure (3.5) and we increase l by 1, then the new inequality has $M_j = \emptyset$ for $j \in \{3, \ldots, n\}$. It is not hard to see that, since $M_3 = \emptyset$, the inequality can be lifted by increasing u^* to u_3. In this way we also obtain an inequality with structure (3.6). As in the proof of Lemma 4.2, we conclude that we may assume that either $\tilde{x}_{1,l-p_1+1} > 0$, or that there exists a violated inequality with structure (3.6). In the first case we analogously find that there exists a violated inequality such that also $\tilde{x}_{1u} > 0$, i.e., we may assume that (i) holds, or that there exists a violated inequality with structure (3.6).

Now, let $\tilde{x}$ violate $x(V^1) + 2x(V^2) \leq 2$ with structure (3.6). As in the previous case, we can show that we may assume $\tilde{x}_{1,l-p_1+1} > 0$, or $l_2 = l$ and $\tilde{x}_{2,l-p_2+1} > 0$, since otherwise l can be increased. Analogously, we may assume $\tilde{x}_{2u} > 0$, or $u_1 = u$ and $\tilde{x}_{1u} > 0$. If $\tilde{x}_{1,l-p_1+1} > 0$ and $\tilde{x}_{2u} > 0$, then (ii) holds.

Suppose that $\tilde{x}_{1,l-p_1+1} = 0$. We show that we may assume that (i) or (ii) holds. By the above we may assume $l_2 = l$ and $\tilde{x}_{2,l-p_2+1} > 0$. If $u_1 = u$ and $\tilde{x}_{1u} > 0$, then (ii) holds with jobs 1 and 2 interchanged. Suppose that $u_1 < u$ or $\tilde{x}_{1u_1} = 0$; hence, we may assume $\tilde{x}_{2u} > 0$. If we interchange jobs 1 and 2, then we find an inequality with structure (3.4) or (3.5) with $l = l_2$ and possibly $u = u_2$. Note that in the new inequality $\tilde{x}_{1,l-p_1+1} > 0$ and $\tilde{x}_{1u} > 0$. Since in this inequality $\tilde{x}_{2,l_2-p_2+1} = 0$, we obtain another violated inequality by increasing l_2. Similarly, if in the new inequality $u = u_2$, then we obtain another violated inequality by decreasing u_2. We may hence assume that (i) holds. We conclude that if $\tilde{x}_{1,l-p_1+1} = 0$, then we may assume that either (ii) holds with jobs 1 and 2 interchanged, or that (i) holds. Analogously, we find that we may assume that either (i) or (ii) holds if $\tilde{x}_{2u} = 0$. □

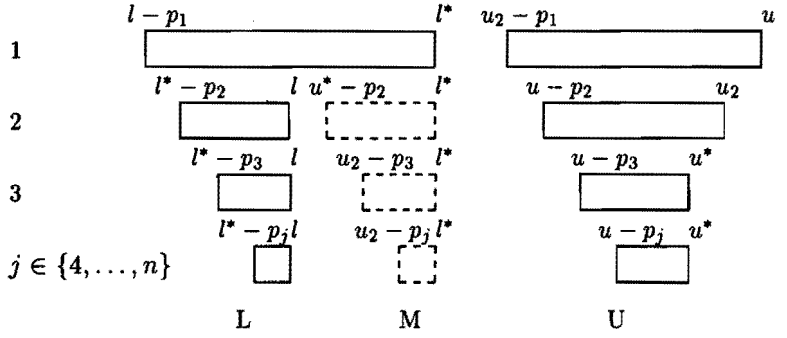
Note that for an inequality $x(V^1)+2x(V^2) \leq 2$ with structure (3.4), $[u_2-p_j, l] \subseteq L_j$ and $M_j = [u_2 - p_j, l_2] \setminus (L_j \cup U_j)$ for all $j \in \{3, \ldots, n\}$. This implies that $l^* \leq u_2$, unless $L_j = \emptyset$ and $M_j = \emptyset$ for all $j \in \{3, \ldots, n\}$. In the latter case $u_2 - p_j \geq l$ for all $j \in \{3, \ldots, n\}$, and it is easy to see that the structure of the inequality does not change if we define $l^* = u_2$. We may hence assume $l^* \leq u_2$, although under this assumption $l^* = \min\{l_j \mid j \in \{3, \ldots, n\}\}$ may not hold. In the separation algorithm we assume $l^* \leq u_2$, and similarly we assume $l_2 \leq u^*$. It is not hard to see that these assumptions imply that $l_2 = u^*$ if $L_2 = \emptyset$ and $u_2 = l^*$ if $U_2 = \emptyset$.

Observe that for $l_2 = l^*$ or $u_2 = u^*$ an inequality with structure (3.4) also has structure (3.5). The following lemma shows that it is beneficial for the separation algorithm to check some of the inequalities with structure (3.5) while it is considering inequalities with structure (3.5). Therefore, in the separation algorithm these inequalities are also seen as inequalities with structure (3.5), i.e., we allow structure (3.5) with $l_j = l_2$ and $u_j = u_3$ for some $j \in \{2, \ldots, n\}$.

Lemma 4.5 *If $\tilde{x}$ violates an inequality $x(V^1) + 2x(V^2) \leq 2$ with structure* (3.4) *or* (3.5)*, then we may assume that one of the following two holds:*

(i) $x(V^1) + 2x(V^2) \leq 2$ *has structure* (3.4), $\tilde{x}_{2,l_2-p_2+1} > 0$, *and* $\tilde{x}_{2u_2} > 0$;
(ii) $x(V^1) + 2x(V^2) \leq 2$ *has structure* (3.5) *possibly with* $l_j = l_2$ *and* $u_j = u_3$ *for some* $j \in \{2, \ldots, n\}$, $\tilde{x}_{2,l_2-p_2+1} > 0$, *and* $\tilde{x}_{3u_3} > 0$.

Proof. Suppose that $\tilde{x}$ violates the inequality $x(V^1)+2x(V^2) \leq 2$ with structure (3.4); suppose further that $\tilde{x}_{2,l_2-p_2+1} = 0$. We first show that we may assume $l_2 = l^*$. Let $l_2 < l^*$. Note that $l_2 \leq u^*$. If $l_2 < u^*$, then we obtain another violated inequality by increasing l_2. If $l_2 = u^*$, then we obtain such an inequality by increasing l_2 and u^* simultaneously. Hence, we may assume $l_2 = l^*$, i.e., $x(V^1) + 2x(V^2) \leq 2$ can be represented by the following diagram:



$\leq 2.$

Observe that since $l^* = l_2 < u_2$ and $M_j = [u_2 - p_j, l_2] \setminus (L_j \cup U_j)$ for $j \in \{3, \ldots, n\}$, we must have $L_j \neq \emptyset$ for some $j \in \{3, \ldots, n\}$. We may assume $\tilde{x}_{j,l^*-p_j+1} > 0$ for some $j \in \{3, \ldots, n\}$, since otherwise we obtain another violated inequality by increasing l^*; w.l.o.g. we assume $\tilde{x}_{3,l^*-p_3+1} > 0$. Note that by assumption $l^* = l_2 \leq u^*$. It is easy to see that $x(V^1) + 2x(V^2) \leq 2$ also has structure (3.5) with job 3 as job 2. If $u_2 > u^*$, then we may assume $\tilde{x}_{2u_2} > 0$, and we find that (ii) holds with jobs 2 and 3 interchanged. If $u^* = u_2$, then by Property 3.6, $l^* = l_2 < u_2 = u^*$, and it is not hard to see that we may assume $\tilde{x}_{ju^*} > 0$ for some $j \in \{2, \ldots, n\}$. If $j = 3$, then (i) holds with job 3 and as job 2. If $j \neq 3$, then $\tilde{x}$ violates an inequality with structure (3.5), possibly with $l_j = l_2$, and $u_j = u_3$ for some $j \in \{2, \ldots, n\}$ and we find that (ii) holds. We conclude that if $\tilde{x}_{2,l_2-p_2+1} = 0$, then we may assume that (i) holds with job 3 as job 2 or (ii) holds. Analogously, we can show that we may assume that (i) or (ii) holds if $\tilde{x}_{2u_2} > 0$.

Let $\tilde{x}$ violate $x(V^1) + 2x(V^2) \leq 2$ with structure (3.5). We have that we may assume $\tilde{x}_{l_2-p_j+1} > 0$ for some j such that $l_j = l_2$, since otherwise we obtain another violated inequality by increasing l_2. If $l_2 < l^*$, then we must have $j = 2$. Note that, if $l_2 = l^*$, then the sets V_j with $j \in \{2, 4, \ldots, n\}$ all have the same structure. Furthermore, by definition of case (1b), we have $l_3 > l_2$. It now easily follows that we may assume $j = 2$. Analogously, we may assume $\tilde{x}_{3u_3} > 0$. We conclude that we may assume that (ii) holds. □

The proofs of the following lemmas proceed along the same lines as those of the previous ones.

Lemma 4.6 *If $\tilde{x}$ violates $x(V^1) + 2x(V^2) \leq 2$ with structure* (3.6)*, then we may assume that*
(a) *if $l_2 > l$, then $\tilde{x}_{1l_2} > 0$;*
(b) *if $u_1 < u$, then $\tilde{x}_{2,u_1-p_1+1} > 0$.* □

Lemma 4.7 *If $\tilde{x}$ violates $x(V^1) + 2x(V^2) \leq 2$ with structure* (3.4)*, 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.* □

Note that for an inequality $x(V^1) + 2x(V^2) \leq 2$ with structure (3.4) 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.

Lemma 4.8 *If $\tilde{x}$ violates $x(V^1) + 2x(V^2) \leq 2$ with structure* (3.5)*, then we may assume that*
(a) *if $l^* > l_2$, then $\tilde{x}_{2l^*} > 0$, $M_2 \neq \emptyset$, and l^* is the maximum of M_2;*
(b) *if $u^* < u_3$, then $\tilde{x}_{3u^*-p_3+1} > 0$, $M_3 \neq \emptyset$, and $u^* - p_3 + 1$ is the minimum of M_3.* □

Lemma 4.9 *If $\tilde{x}$ violates $x(V^1) + 2x(V^2) \leq 2$ with structure* (3.6)*, 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 $u' < u_1$, 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.* □

Based on the previous lemmas, one can derive separation algorithms for inequalities $x(V^1) + 2x(V^2) \leq 2$ with structure (3.4), (3.5), or (3.6). We do not describe these algorithms in full detail; we give the main lines of the separation algorithm for inequalities with structure (3.4).

The algorithm identifies inequalities that are nondecomposable. Let $x(V^1) + 2x(V^2) \leq 2$ with structure(3.4) be nondecomposable. Since $x(V^1) + 2x(V^2) \leq 2$ has $M \neq \emptyset$, we must have $u > l + p_{\min}$, where $p_{\min}$ is the minimum of the processing times. Furthermore, it is not hard to see that $L_j \neq \emptyset$ for some $j \in \{2, \ldots, n\}$

and $U_j \neq \emptyset$ for some $j \in \{2, \ldots, n\}$; it follows that $l_2 < l + \max\{p_j \mid j \neq 1\}$ and $u_2 > u - \max\{p_j \mid j \neq 1\}$. Note that if $L_2 = \emptyset$ and $U_2 = \emptyset$, then $u^* = l_2$ and $l^* = u_2$. It is now easy to see that, since, by Property 3.6, $l_2 < u_2$, $x(V^1) + 2x(V^2) \leq 2$ is the sum of two valid inequalities $x(W) \leq 1$ and $x(W') \leq 1$, where $W = \{(1,s) \mid s \in L_1 \cap U_1\} \cup \{(j,s) \mid j \in \{2, \ldots, n\},\ s \in V_j\}$ and $W' = \{(1,s) \mid s \in V_1\} \cup \{(j,s) \mid j \in \{2, \ldots, n\},\ s \in L_j \cap U_j\}$. Hence $L_2 \neq \emptyset$ or $U_2 \neq \emptyset$.

In the separation algorithm $I_{1a}(j_1, j_2, l, l_2, l^*, u^*, u_2, u)$ denotes the left-hand side of the inequality with structure (3.4) 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 $O(n^2 T^4 p_{\max}^2)$. The number of inequalities that are checked by the separation algorithm is bounded by the sixth power of the number of positive variables in the current LP-solution. The separation algorithm is as follows.

Sepa21a($\tilde{x}$)
begin
for j_1 and l such that $\tilde{x}_{j_1, l - p_{j_1} + 1} > 0$ and there exists a u such that $\tilde{x}_{j_1,u}$ 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 there exists a 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**
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
Sepa21aL2emp($\tilde{x}, j_1, j_2, l, l_2, u_2, u$);
else
if $u_2 \leq u - p_{j_2}$ $\{U_{j_2} = \emptyset\}$
then
Sepa21aU2emp($\tilde{x}, j_1, j_2, l, l_2, u_2, u$);
else $\{L_{j_2} \neq \emptyset$ and $U_{j_2} \neq \emptyset\}$
Sepa21anonemp($\tilde{x}, j_1, j_2, l, l_2, u_2, u$);
end.

Sepa21aL2emp($\tilde{x}, j_1, j_2, l, l_2, u_2, u$)
begin
$u^* = l_2$;
for $l^* = l_2$ and $l^* > l_2$ such that condition (a) of Lemma 4.7 is satisfied **do**
if $I_{1a}(j_1, j_2, l, l_2, l^*, u^*, u_2, u) > 2$
then violated inequality identified;
end.

Sepa21aU2emp$(\tilde{x}, j_1, j_2, l, l_2, u_2, u)$
begin
$l^* = u_2$;
for $u^* = u_2$ and $u^* < u_2$ such that condition (b) of Lemma 4.7
is satisfied **do**
if $I_{1a}(j_1, j_2, l, l_2, l^*, u^*, u_2, u) > 2$
then violated inequality identified;
end.

Sepa21anonemp$(\tilde{x}, j_1, j_2, l, l_2, u_2, u)$
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 $I_{1a}(j_1, j_2, l, l_2, l^*, u^*, u_2, u) > 2$
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 $I_{1a}(j_1, j_2, l, l_2, l^*, u^*, u_2, u) > 2$
then violated inequality identified;
for $l^* > l_2$ and $u^* < u_2$ such that the conditions from
Lemma 4.7 are satisfied **do**
if $I_{1a}(j_1, j_2, l, l_2, l^*, u^*, u_2, u) > 2$
then violated inequality identified;
end.

4.3 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 $1|r_j|\sum w_j C_j$. In this section we describe this branch-and-cut algorithm and report on its performance. In Subsection 4.3.1, we discuss the performance of the cutting plane algorithm, i.e., we study the quality of the lower bounds obtained by adding violated inequalities to the LP-relaxation. To apply a branch-and-cut algorithm, we have to specify a branching strategy, a row management policy, and what primal heuristic is used. Different choices lead to different variants of the branch-and-cut algorithm. In Subsections 4.3.2, 4.3.3, and 4.3.4, we discuss several branching strategies, row management policies, and primal heuristics, and we present the results of the computational experiments that we have conducted to empirically test the performance of the several variants of the algorithm. In Subsection 4.3.5, we present the computational results of two variants that seem to dominate the other variants of the branch-and-cut algorithm. In Subsection 4.3.6, we compare the performance of our algorithm to those of the

algorithms presented by Sousa and Wolsey [1992] and Crama and Spieksma [1993] and to 'standard' branch-and-bound algorithms.

We report results for five sets of twenty 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 three sets of 20-job instances with $p_{\max}$ equal to 5, 10, and 20, respectively, and two 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 $p_{\max} = 20$, as in this case the linear programs turned out to be too large for our branch-and-cut algorithm.

We have embedded all variants of the algorithm in MINTO. MINTO, a Mixed INTeger Optimizer [Nemhauser, Savelsbergh, and Sigismondi, 1994], 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 1.6/CPLEX 2.1 and have been run on a SUN Sparcstation.

4.3.1 Quality of the lower bounds

The goal of our first experiments is to test the quality of the lower bounds that are obtained by the cutting plane algorithm. The results for the one hundred test instances are summarized in Tables 4.1 and 4.2. Let Z_{LB} denote a lower bound on the optimal value Z_{IP} of the integer programming problem. 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 4.1, 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 4.1: Quality of the bounds.

We observed that many instances were solved to optimality by the cutting plane algorithm. Table 4.2 provides some more statistics on the frequency with which this occurs. 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.

$(n, p_{\max})$	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 4.2: Number of instances that were solved to optimality.

The results show that both classes of inequalities are effective in reducing the integrality gap. Table 4.1 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 4.2 we conclude that the addition of cuts with right-hand side 2 significantly increases the number of instances that are solved without branching.

4.3.2 Branching strategies

If we have not found an integral solution after the addition of cutting planes to the LP-relaxation, then we have to branch to find an integral solution. In this subsection, we discuss several branching strategies that can be used in the branch-and-cut algorithm and test their influence on the performance of the algorithm. In our computational experiments, the lower bounds are computed by adding all cuts with right-hand side 1 and 2 to the LP-relaxation; all primal heuristics that will be discussed in Section 4.3.4 are applied.

To apply a branch-and-bound algorithm, we not only have to specify how to partition the set of feasible solutions, but also how to select the next subproblem to be considered. We consider two selection strategies: *depth-first* and *best-bound*. Depth-first search selects among the subproblems that have not yet been evaluated the one that was created last. This means that we select a subproblem at the deepest level in the tree. Depth-first search is applied in the hope to find a good feasible solution fast. This feasible solution gives an upper bound, which can be used to fathom nodes, i.e., to skip subproblems from further consideration. Best-bound search selects the subproblem with the best lower bound. In case of minimization this means that it selects the subproblem with the smallest lower bound. Best-bound search is motivated by the observations that the subproblem with the best lower bound has to be evaluated anyway and that it is more likely to contain the optimal solution than any other node.

We tested three kinds of branching strategies. The first kind of strategy can be used for 0-1 programming problems in general, the second one for 0-1 programming problems that contain constraints of the form $\sum x_j \leq 1$ or $\sum x_j = 1$, whereas the third one employs the structure of the scheduling problem.

The first kind of strategy is based on fixing the value of variables to either 0 or 1. In the first one of these strategies a variable with value closest to $\frac{1}{2}$ is fixed. The idea behind this strategy is that it fixes a variable of which it is hard to determine the value in an optimal solution. In the other strategy a fractional variable with value closest to 1 is fixed. The idea here is that the fixed variable will probably have value 1 in the optimal solution and that the node corresponding to value 0 can hence be fathomed quickly. It turns out that these two branching strategies perform best in combination with best-bound search.

The second kind of strategy performs so-called *special ordered set (SOS)* branching. Such strategies can be applied to 0-1 programming problems that contain constraints of the form $\sum x_j \leq 1$ or $\sum x_j = 1$. In our problem the constraints (4.1), which state that each job has to be started exactly once, are of the latter form. The subproblems are defined by the constraints $\sum_{t=1}^{t^*} x_{jt} = 0$ and $\sum_{t=t^*+1}^{T-p_j+1} x_{jt} = 0$ for some $j \in \{1, \ldots, n\}$ and $t^* \in \{1, \ldots, T - p_j + 1\}$. This branching strategy splits the interval in which a given job j is allowed to be started in the parts $[0, t^*]$ and $[t^*, T - p_j + 1]$. We consider two diffrerent ways to select j and t^*. The first one is to

choose the job j that covers the largest time interval, i.e., the job for which the difference between the first and the last period with positive x_{jt} is maximal. Then we choose t^* equal to the *mean start time* of job j, which is given by $\sum_{t=1}^{T-p_j+1}(t-1)x_{jt}$. The idea behind this strategy is that for the job that covers the largest time interval the current solution deviates most from a single start time. Splitting the interval at the mean start time leads to an even division of the solution space according to the current fractional solution. The second method is to choose j and t^* in such a way that $\sum_{t=1}^{t^*} x_{jt}$ is closest to $\frac{1}{2}$. This strategy forces job j to be started entirely in the time interval in which its 'earlier half' is started, or in the time interval in which its 'later half' is started. These branching strategies make more general decisions about the starting time of jobs than strategies that fix a single variable. They provide a more even division of the solution space and hence a more balanced tree. There is however no indication that a feasible solution will be found quickly. These strategies also perform best in combination with best-bound search.

The last branching strategy is based on fixing jobs at certain positions in the schedule. At level d in the branch-and-bound tree some job is fixed at position d. At this level the jobs in positions $1, \ldots, d-1$ have already been fixed. Note that, if a job j is fixed in position d in a schedule in which the jobs in positions $1, \ldots, d-1$ have already been fixed, then its starting time can be set equal to the maximum of its release date and the completion time of the $(d-1)$st job. Hence, we can fix one of the variables x_{jt} at 1. 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. This branching strategy is applied in combination with depth-first search. 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 an ordering of the jobs suggested by the fractional solution. The jobs are put in nondecreasing order of *virtual start times*. The virtual start times can be defined in different ways: equal to the mean start time, which is given by $\sum_{t=1}^{T-p_j+1}(t-1)x_{jt}$, equal to $t-1$, where t is the index for which x_{jt} is maximal, or equal to $t-1$, where t is the first time period for which x_{jt} is positive. It turns out that choosing the virtual start time equal to the mean start time yields the best performance. Therefore, we only consider this choice for the virtual start time in our branch-and-cut algorithm, and we do not report on computational experiments with the other possibilities. In Subsection 4.3.4, we shall discuss the use of virtual start times in primal heuristics.

In Tables 4.3(a,b) and 4.4 we report on the performance of the different branching strategies for the twelve instances with $n = 30$ and $p_{\max} = 5$ and the eleven instances with $n = 30$ and $p_{\max} = 10$ that were not solved to optimality by the cutting plane algorithm. Since the majority of the 20-job instances was solved to optimality without branching, we do not report results on these instances. The conclusions we draw from the 30-job instances also hold for the 20-job instances.

Table 4.3(a) shows for the instances with $n = 30$ and $p_{\max} = 5$ the performance

of the different branching strategies:

- *pos*: fixing jobs at certain positions using mean start times;
- *SOS-mst*: SOS-branching with j and t^* such that j is the job that covers the largest time interval and t^* is the mean start time of job j;
- *SOS-half*: SOS-branching with j and t^* such that $\sum_{t=1}^{t^*} x_{jt}$ is as close as possible to $\frac{1}{2}$;
- *half*: fixing the value of a variable closest to $\frac{1}{2}$;
- *max-frac*: fixing the value of the fractional variable with maximum value.

If the results are based on a subset of the instances, then the number between brackets indicates the cardinality of the subset. For the instances that are excluded, there was insufficient memory, except for *SOS-mst*(10), where an outlier was omitted. In the first three rows of both 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 second part of the table, we give the same information on the computation time (t^{av}, $t^{\max}$, and σ_t).

(30,5)	*pos*	*SOS-mst*	*SOS-half*(10)	*half*(11)	*max-frac*(11)
n^{av}	96	10	210	351	1388
$n^{\max}$	219	33	1081	1511	7049
σ_n	66	8	334	529	2223
t^{av}	94	60	1052	970	3751
$t^{\max}$	219	107	4401	4385	20526
σ_t	52	31	1615	1428	6228

Table 4.3(a): Performance of the different branching strategies for $n = 30$ and $p_{\max} = 5$.

From Table 4.3(a), we conclude that the branching strategies *half*, *max-frac*, and *SOS-half* display an erratic behavior. Strategy *half* performs better than strategy *max-frac*, and strategy *SOS-half* is dominated by *SOS-mst*. Therefore, we have decided not to include the strategies *max-frac* and *SOS-half* in our computations for the 30-job instances with $p_{\max} = 10$. The performance of the other strategies is reported in Table 4.3(b).

For the instances with $n = 30$ and $p_{\max} = 5$, Table 4.4 reports on the performance of the strategies *pos*, *SOS-mst*, and *half* in more detail. It shows for each instance the number of nodes and the computation time that was required to solve the instance to optimality. A '?' means that there was not enough memory.

(30,10)	*pos*	*SOS-mst*	*SOS-mst*(10)	*half*(10)
n^{av}	134	49	14	49
n^{max}	353	403	27	159
σ_n	122	112	7	54
t^{av}	691	1742	421	1352
t^{max}	2648	14952	1106	3854
σ_t	750	4189	330	1390

Table 4.3(b): Performance of the different branching strategies for $n = 30$ and $p_{max} = 10$.

	pos		*SOS-mst*		*half*	
problem	# nodes	time	# nodes	time	# nodes	time
R30.5.1	29	37	7	47	36	44
R30.5.2	18	93	3	85	3	88
R30.5.4	90	78	7	39	19	62
R30.5.5	116	86	8	27	1271	4385
R30.5.7	92	79	21	107	?	?
R30.5.11	6	27	3	26	3	18
R30.5.13	26	27	5	35	1511	2868
R30.5.14	126	106	33	107	229	447
R30.5.16	198	219	9	39	87	452
R30.5.17	219	126	7	22	3	39
R30.5.18	134	104	6	44	5	52
R30.5.20	92	142	15	82	689	2214

Table 4.4: Performance of the three best branching strategies for the instances with $n = 30$ and $p_{max} = 5$.

On our instances the strategy *SOS-mst* performs best. However, for $n = 30$ and $p_{max} = 10$, the average computation time that is required exceeds the computation time that is required by $half(10)$ and *pos*, due to one outlier. Recall that this outlier was omitted in column *SOS-mst*(10). It has also been omitted in column $half(10)$, because of lack of memory. This might indicate that the strategy *pos* is more robust than *SOS-mst*, i.e., the number of nodes that have to be evaluated does not vary as much as the number of nodes that have to be evaluated by *SOS-mst*.

An advantage of the strategy *SOS-mst* is that it can be applied for all objective functions $\sum_{j=1}^{n} \sum_{t=1}^{T-p_j+1} c_{jt}x_{jt}$. The strategy *pos* is based on the assumption that it is most favorable to start a job as early as possible, which implies that *pos* can only

be used in case the objective function is non-decreasing in the completion times of the jobs.

On the other hand, for the strategy *pos* the total number of nodes in the branch-and-bound tree only depends on the number of jobs n, whereas for *SOS-mst* this number also depends on the horizon T and hence on the size of the processing times. This suggests that *pos* may perform better for instances with large processing times.

Since the strategies *pos* and *SOS-mst* clearly dominate all other strategies, we from now on only consider these two strategies.

During the branch-and-cut algorithm it may happen that in a number of consecutive iterations violated inequalities are identified and added to the current LP-relaxation, but that the optimal value hardly changes; this effect is called *tailing off*. To prevent tailing off, we sometimes force a branching step in a node other than the root node. A branching step is forced whenever the value of the current LP-relaxation has improved less than 0.5 percent during the last three iterations. In many cases, this means that outside the root node a branching step is performed after three iterations. It turns out that this forced branching strongly reduces the computation time.

4.3.3 Row management

In this subsection, we study the influence of different row management policies on the performance of the branch-and-cut algorithm. *Row management* involves the way in which violated inequalities that have been identified by the separation algorithm are handled. The main issues are the following. Which of the identified inequalities are added to the formulation? Do we remove inequalities that do not seem to play a role anymore, i.e., inequalities whose slack variable has exceeded a certain value or whose dual variable is equal to 0? We have to deal with the following trade-off. On the one hand, the best lower bound is obtained by adding as many violated inequalities as possible. On the other hand, the time needed to solve the linear programming problems may increase significantly if we add many inequalities.

Our separation procedure works as follows. First, it is checked if there are violated inequalities with right-hand side 1. If no such inequalities are identified, then it is subsequently checked whether there are violated inequalities with structure (3.5), (3.4), or (3.6). If violated inequalities are identified in one of these classes, then all these inequalities are added to the formulation and the other classes are skipped from further inspection.

To investigate the effect of the addition of cutting planes on the performance of the branch-and-bound algorithm, we tested the following policies:

- *noinq*: do not add cuts;
- [1]: add only cuts with right-hand side 1;
- [1, 2]: add cuts with right-hand 1 and 2;
- *r*2: add cuts with right-hand side 1 and 2 in the root node, and add no cuts in the rest of the tree;
- *r*21: add cuts with right-hand side 1 and 2 in the root node, and add only cuts with right-hand side 1 in the rest of the tree.

The last two policies are motivated by the observation that it is important to find a good lower bound in the root node, since this bound can be used in all other nodes that are evaluated. Inequalities that are generated in the rest of the tree only play a role in a part of the tree, while adding them increases the time needed to solve the linear programs.

The performance of these policies is reported in the Tables 4.5(a-d) and 4.6. As in the previous section, we only report on results for the instances with 30 jobs. The conclusions that we draw also hold for the 20-job instances. In Tables 4.5(a-d), we report on the performance of these policies when combined with the branching strategies *pos* and *SOS-mst*. 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. Furthermore, $\#inq^{\text{av}}$ and $\#inq^{\max}$ denote the average and maximum number of inequalities that have been added. Again, if the results are not based on all instances, then the number between brackets indicates how many instances are included. In Table 4.5(d) the outlier that was omitted in Table 4.3(b) for the strategy *SOS-mst* has been excluded for all policies. This instance turns out to be very hard to solve with *SOS-mst*: for most policies of adding cutting planes, there was not enough memory.

(30, 5)	*noinq*	[1]	*r*2	*r*21	[1, 2]
n^{av}	751	225	225	167	96
$n^{\max}$	1975	492	564	344	219
t^{av}	204	87	105	94	94
$t^{\max}$	448	188	179	183	219
$\#inq^{\text{av}}$	0	55	112	123	170
$\#inq^{\max}$	0	118	440	440	440

Table 4.5(a): Comparing different policies of adding cuts for *pos* with $n = 30$ and $p_{\max} = 5$.

$(30,10)$	*noinq*	[1]	$r2(10)$	$r21$	$[1,2]$
n^{av}	2609	316	272	228	134
n^{max}	10406	1147	831	828	353
t^{av}	1788	457	461	536	691
t^{max}	7528	1277	1189	1549	2648
$\#inq^{av}$	0	170	292	402	601
$\#inq^{max}$	0	498	1007	1188	2071

Table 4.5(b): Comparing different policies of adding cuts for *pos* with $n = 30$ and $p_{max} = 10$.

$(30,5)$	*noinq*	[1]	$r2$	$r21$	$[1,2]$
n^{av}	96	42	16	10	10
n^{max}	383	339	41	21	33
t^{av}	174	145	56	51	60
t^{max}	868	1170	142	95	107
$\#inq^{av}$	0	84	112	126	135
$\#inq^{max}$	0	382	440	440	440

Table 4.5(c): Comparing different policies of adding cuts for *SOS-mst* with $n = 30$ and $p_{max} = 5$.

$(30,10)$	*noinq*(9)	[1](10)	$r2(10)$	$r21(10)$	$[1,2](10)$
n^{av}	328	55	28	17	14
n^{max}	1035	279	75	45	27
t^{av}	4991	1135	442	433	421
t^{max}	16346	3570	929	1229	1106
$\#inq^{av}$	0	192	292	327	341
$\#inq^{max}$	0	485	1007	1008	1008

Table 4.5(d): Comparing different policies of adding cuts for *SOS-mst* with $n = 30$ and $p_{max} = 10$.

As could be expected, the results presented in Tables 4.5(a-d) indicate that the number of nodes that have to be evaluated increases when only inequalities with right-hand side 1 are added, and increases even more when no inequalities are added. The addition of cutting planes clearly reduces the computation time.

In case of positional branching, the average computation time was minimal when only cuts with right-hand side 1 were added. However, for the instances with $p_{\max} = 5$ the difference between this strategy and $[1, 2]$, $r2$, and $r21$ was quite small. For the instances with $p_{\max} = 10$ the difference with $r2$ is also small, but the results on the latter policy are not based on all instances.

In case of branching strategy *SOS-mst* the policies in which inequalities with right-hand side 2 are involved provide minimal computation times, but none of these policies clearly dominates the others.

Overall, the branching strategy *SOS-mst* in combination with policies in which inequalities with right-hand side 2 are involved seems to perform best, although no variant is clearly dominant.

As we have seen in Table 4.2, including only cuts with right-hand side 1 instead of all cuts decreases the number of instances that are solved without branching. For a better comparison between [1] and $[1, 2]$, we report in Table 4.6 the average computation time per combination of n and $p_{\max}$ with branching strategies *pos* and *SOS-mst*. All instances that are not solved to optimality by adding cuts with right-hand side 1 to the LP-relaxation are included, including the ones with 20 jobs. We omit the outlier with $n = 30$ and $p_{\max} = 10$ from the regular average calculation and present it separately.

		pos		*SOS-mst*	
$(n, p_{\max})$	# instances	[1]	$[1, 2]$	[1]	$[1, 2]$
(20, 5)	8	12	12	20	13
(20,10)	14	38	52	69	46
(20,20)	7	166	134	107	136
(30, 5)	14	81	83	128	54
(30,10)	14	344	367	890	314
R30.10.16	1	1277	2648	?	14952

Table 4.6: Average computation times.

The results in Table 4.6 show that for the 20-job instances no strategy dominates the others. For the 30-job instances *SOS-mst* in combination with $[1, 2]$ seems to perform best, except for one nasty instance. For positional branching policy [1] performs slightly better than policy $[1, 2]$.

In the computational experiments in the next sections, we restrict ourselves for each branching strategy to one of the row management policies that performs best. We consider the combinations of branching strategy *SOS-mst* with policy $[1, 2]$ and of branching strategy *pos* with policy [1].

Instead of adding all inequalities that are identified by the separation algorithm, one can also add a restricted number of inequalities, say the m most violated inequalities, where m is some small number. It turned out that for most of our test problems the performance of the branch-and-cut algorithm hardly changes if we choose $m = 5$. This can be explained by the fact that the number of inequalities that are added per iteration is not too large; for most of the instances this number is less than 10. If we only add the most violated inequality, i.e., $m = 1$, then the number of linear programs that have to be solved and the number of nodes that have to be evaluated increase significantly. Since only a small number of inequalities are added, the linear programs can be solved more quickly, and therefore the computation time does not increase that much; for some instances it even decreases. For some of the instances with $n = 30$ and $p_{\max} = 10$, the branch-and-cut algorithm needs too much memory when all identified inequalities are added, whereas the problem is solvable when in each iteration only the most violated inequality is added.

To prevent the size of the linear programs from growing too much, we remove inequalities whose slack variable has exceeded a certain value. Every five iterations, we remove all inequalities with slack variables greater than 0.1. The deletion of inactive inequalities decreases the computation time of the branch-and-cut algorithm. Increasing the frequency of removing inequalities to for example every two iterations has hardly any influence on the performance of the algorithm.

4.3.4 Primal heuristics

In this subsection we describe the primal heuristics that we use in the branch-and-cut algorithm. The availability of good feasible solutions may reduce the number of nodes that have to be evaluated considerably, 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. Good feasible solutions can often be derived from fractional solutions that are optimal for the current LP-relaxation.

We have implemented four primal heuristics. The first heuristic is derived from Smith's rule, which states that the problem is solved by scheduling the jobs in order of nonincreasing w_j/p_j ratio in case no release dates or deadlines have been specified. This first heuristic schedules the jobs according to the following rule. The job in the first position is the job with the smallest release date. When a job is completed, select from all available jobs the one with the smallest w_j/p_j ratio to be scheduled next.

The other three heuristics make use of the values in the current fractional solution. They schedule the jobs according to the virtual orderings that were mentioned in Section 4.3.2. Recall that these orderings are based on virtual start times, which can be defined in three different ways. The virtual start time of job j can be defined by:

- the mean start time, which is defined as $\sum_{t=1}^{T-p_j+1}(t-1)x_{jt}$;
- the start time with maximum value, which is defined as $t-1$, where t is the index for which x_{jt} is maximum;
- the first start time with positive value, which is defined as $t-1$, where t is the first index for which x_{jt} is positive.

These three ways of defining the virtual start times provide three heuristics. In most situations, the ordering based on the mean start time provides the best feasible solution. It is however useful to apply all three methods, since they hardly take time. Note that these heuristics are applied every time that a linear program has been solved, whereas the first heuristic is applied only once.

In Table 4.7 we report for the twelve problems with $n = 30$ and $p_{\max} = 5$ that were not solved to optimality by adding cuts with right-hand side 2 to the LP-relaxation the value provided by the first heuristic (Z_{ratio}), the best value of the other three heuristics after the LP-relaxation has been solved ($Z_{\text{frac}}^{\text{lp}}$), the best value of these heuristics attained in the root node ($Z_{\text{frac}}^{\text{root}}$), and the optimal value (Z_{IP}).

problem	Z_{ratio}	$Z_{\text{frac}}^{\text{lp}}$	$Z_{\text{frac}}^{\text{root}}$	Z_{IP}
R30.5.1	6888	6996	6745	6727
R30.5.2	6869	6637	6533	6533
R30.5.4	5739	5540	5540	5540
R30.5.5	5201	5369	5201	5185
R30.5.7	4646	4682	4646	4620
R30.5.11	6246	6022	6016	5986
R30.5.13	5468	5508	5468	5434
R30.5.14	4163	4068	4050	4044
R30.5.16	4676	4611	4597	4597
R30.5.17	6419	6295	6285	6279
R30.5.18	4726	4731	4679	4606
R30.5.20	4721	4623	4616	4558

Table 4.7: Values obtained by the heuristics.

The computational results show that fractional solutions occurring during a branch-and-cut algorithm provide very good starting-points for obtaining primal solutions; the heuristics based on these fractional solutions provide better primal solutions than the first heuristic.

The effect of the use of the heuristics on the performance of the algorithm is presented in Tables 4.8(a,b). For the 30-job instances, we report results on the

variant of the branch-and-cut algorithm with branching strategy *SOS-mst* and addition of all cuts and the variant with branching strategy *pos* and addition of cuts with right-hand side 1. For both variants, we give the average and maximum number of nodes and the average and maximum computation time (n^{av}, n^{max}, t^{av}, and t^{max}) that were realized with and without the use of heuristics. For the second variant all instances that were not solved to optimality by adding only cuts with right-hand side 1 to the LP-relaxation are included. Again, if only a subset of the instances is included, then the number between brackets denotes the cardinality of the subset.

	SOS-mst, [1, 2]		*pos*, [1]	
(30, 5)	heur	no heur	heur	no heur
n^{av}	10	11	203	204
n^{max}	33	33	492	492
t^{av}	60	62	81	85
t^{max}	107	101	188	188

Table 4.8(a): Effect of the heuristics for the instances with $n = 30$ and $p_{max} = 5$.

	SOS-mst, [1, 2]		*pos*, [1]	
(30, 10)	heur	no heur	heur	no heur
n^{av}	14	19	277	301
n^{max}	27	53	1147	1166
t^{av}	421	610	406	444
t^{max}	1106	1966	1277	1272

Table 4.8(b): Effect of the heuristics for the instances with $n = 30$ and $p_{max} = 10$.

The effect of the heuristics differs per instance. Sometimes, the computation time increases a bit due to the time required to compute the primal solution. In other cases, the number of nodes and the computation time are reduced significantly. The computational results indicate that for the instances with $n = 30$ and $p_{max} = 10$ the use of the heuristics significantly reduces the number of nodes and the computation time.

4.3.5 Two variants in more detail

In the previous three sections, we described the development of a branch-and-cut algorithm for the problem $1|r_j|\sum w_jC_j$ and tested different variants of the

algorithm. For each of the branching strategies *pos* and *SOS-mst*, we report in detail on the performance of a variant of the branch-and-cut algorithm that seems to perform best. In Table 4.9 we report on the performance of the algorithm when inequalities with right-hand side 1 and 2 were added and the branching strategy *SOS-mst* was used; in Table 4.10 we consider the case of adding only inequalities with right-hand side 1 in combination with branching strategy *pos*. Table 4.9 shows for each instance that was not solved to optimality by the cutting plane algorithm the number of inequalities with right-hand side 1 and 2 that have been added (#inq1 and #inq2), the number of linear programs that have been solved (#lp), the number of nodes that have been evaluated (#nodes), and the computation time (time). Table 4.10 shows the same figures except for #inq2.

4.4 Related work

Sousa and Wolsey [1992] implemented a cutting plane algorithm based on their three classes of valid inequalities, which have already been discussed in Section 3.6. Their separation algorithms for inequalities with right-hand side 1 and for inequalities with right-hand side 2 in the second class were based on enumeration; they used a simple heuristic to identify violated inequalities in the third class, whereas inequalities with right-hand side $k > 2$ in the second class are left out of consideration. All twenty of their test instances were solved to optimality in the root node. Since they provided us with only four of these instances, which we solved in the root node as well, we were not able to compare both algorithms on a solid basis.

Crama and Spieksma [1993] presented a branch-and-cut algorithm for the case that all processing times are equal. All of their separation algorithms are based on the enumeration of inequalities. They tested their algorithm on two classes of problems. The first one concerns problems in which the coefficients c_{jt} of the objective function are randomly generated. The second one concerns problems in which $c_{jt} = w_j(t - r_j)$ if $r_j \leq t \leq d_j$ and $c_{jt} = M$ otherwise, where M is some large integer. These are problems in which we have to minimize the weighted sum of the completion times subject to release dates and deadlines; the release dates and deadlines may however be violated at large cost. For the first class of problems, the quality of the lower bounds obtained by the addition of cutting planes is almost equal for our algorithm and theirs. For all the thirty instances in the second class their algorithm finds an integral solution without branching. For twenty-eight of these instances our algorithm finds the optimal value by the addition of cuts with right-hand 1 and 2 to the LP-relaxation. For some of these instances we still have to branch to find an integral solution, but we only have to evaluate a small number of nodes.

For the problem $1|r_j|\sum w_jC_j$, different 'standard' branch-and-bound algo-

rithms, i.e., branch-and-bound algorithms that are not based on linear programming relaxations, have been developed. An example is the algorithm presented by 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 have to be evaluated by their algorithm is much larger than the number of nodes that have to be evaluated by our algorithm, but 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 'standard' branch-and-bound algorithms are designed for one specific problem type.

problem	#inq1	#inq2	#lp	#nodes	time
R20.5.10	44	32	36	15	35
R20.5.18	16	4	28	17	16
R20.10.2	71	86	43	7	87
R20.10.4	165	284	77	7	167
R20.10.8	38	4	14	5	23
R20.10.13	52	4	22	7	40
R20.20.5	91	131	27	5	288
R20.20.8	75	15	21	3	160
R20.20.17	58	8	23	9	23
R30.5.1	39	36	36	7	43
R30.5.2	87	353	55	3	85
R30.5.4	30	75	32	7	39
R30.5.5	55	18	29	8	27
R30.5.7	73	39	67	21	107
R30.5.11	29	17	20	3	26
R30.5.13	25	8	19	5	35
R30.5.14	66	51	78	7	107
R30.5.16	90	176	42	9	93
R30.5.17	21	2	15	7	22
R30.5.18	77	61	31	6	44
R30.5.20	77	119	60	15	82
R30.10.1	95	67	28	3	104
R30.10.5	99	15	43	21	397
R30.10.7	107	16	23	7	104
R30.10.9	293	715	64	15	921
R30.10.10	68	17	38	15	113
R30.10.12	265	488	104	27	1106
R30.10.14	136	330	69	13	589
R30.10.15	189	287	51	7	384
R30.10.16	2084	2089	936	403	14952
R30.10.17	84	62	41	13	324
R30.10.20	71	4	28	17	189

Table 4.9: Performance of the branch-and-cut algorithm with *SOS-mst* and [1, 2].

problem	#inql	#lp	#nodes	time
R20.5.10	44	85	71	29
R20.5.18	14	69	58	19
R20.10.2	58	41	32	62
R20.10.4	116	63	42	74
R20.10.8	31	58	52	34
R20.10.13	41	60	56	50
R20.20.5	34	125	114	189
R20.20.8	72	23	11	119
R20.20.17	51	41	35	63
R30.5.1	55	226	204	73
R30.5.2	55	77	66	44
R30.5.4	24	143	134	56
R30.5.5	73	227	194	74
R30.5.7	53	297	273	93
R30.5.11	22	70	61	37
R30.5.13	26	45	38	25
R30.5.14	55	365	344	112
R30.5.16	90	207	188	93
R30.5.17	19	500	492	161
R30.5.18	118	532	486	188
R30.5.20	74	252	221	90
R30.10.1	95	82	64	120
R30.10.5	102	159	130	366
R30.10.7	125	165	144	322
R30.10.9	224	37	20	291
R30.10.10	67	386	362	349
R30.10.12	375	465	407	740
R30.10.14	200	923	837	991
R30.10.15	115	238	219	334
R30.10.16	498	1287	1147	1277
R30.10.17	24	82	77	109
R30.10.20	42	71	64	131

Table 4.10: Performance of the branch-and-cut algorithm with *pos* and [1].

Chapter 5

Column generation

5.1 Introduction

The main disadvantage of the time-indexed formulation is its size; there are $n + T$ constraints and approximately nT variables, where T is at least $\sum_{j=1}^{n} p_j$. As a consequence, for instances with many jobs or large processing times, the memory requirements and the solution times will be large. This was confirmed by our computational experiments. An analysis of the division of the total computation time over the various components of the branch-and-cut algorithm discussed in the previous chapter revealed that a major part of the time was spent on solving linear programs. Recall that the typical number of simplex iterations is proportional to the number of constraints and to the logarithm of the number of variables.

We explore two reformulations of the LP-relaxation that may alleviate some of the problems mentioned above. The first reformulation is relatively simple and straightforward and only reduces the number of nonzeros in the constraint matrix. Therefore, it primarily addresses the memory issue. The second reformulation is based on Dantzig-Wolfe decomposition and reduces the number of rows from $n+T$ to $n+1$ at the expense of many more variables. However, this does not really matter, since the reformulated problem can be solved by means of column generation. Consequently, it addresses both the memory issue and the solution time issue.

In Section 5.2, we discuss these reformulations in more detail and explain the column generation procedure. In Section 5.3, we investigate the possibility of adding cutting planes when the linear programs are solved by column generation. We first study the addition of the facet inducing inequalities that were identified in Chapter 3. After that, we study the combination of row and column generation in a more general context. In Section 5.4, we present some branching schemes that are compatible with the column generation procedure. Finally, in Section 5.5, we explore the possibilities of key formulations that may reduce the amount of computation time and memory required by the column generation algorithm.

5.2 Solving the LP-relaxation

Recall that the LP-relaxation of the time-indexed formulation is given by:

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

subject to

$$\sum_{t=1}^{T-p_j+1} x_{jt} = 1 \qquad (j = 1, \ldots, n), \tag{5.1}$$

$$\sum_{j=1}^{n} \sum_{s=t-p_j+1}^{t} x_{js} \leq 1 \qquad (t = 1, \ldots, T), \tag{5.2}$$

$$x_{jt} \geq 0 \qquad (j = 1, \ldots, n; \ t = 1, \ldots, T - p_j + 1).$$

Since the constraints $x_{jt} \leq 1$ are dominated by the constraints (5.2), they are omitted. In the constraint matrix of this formulation the column associated with variable x_{js} has a one in the row of constraint (5.1) corresponding to job j and in the rows of the constraints (5.2) with $t = s, \ldots, s + p_j - 1$. This column hence contains $p_j + 1$ ones, with p_j of them in consecutive positions. The following reformulation reduces the number of nonzeros in the constraint matrix to three per column. First, we transform the constraints (5.2) into equations by adding slack variables z_t $(t = 1, \ldots, T)$. Then, we subtract constraint (5.2) with $t - 1$ from constraint (5.2) with t for $t = 2, \ldots, T$. The reformulation is given by

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

subject to

$$\sum_{t=1}^{T-p_j+1} x_{jt} = 1 \qquad (j = 1, \ldots, n), \tag{5.3}$$

$$\sum_{j=1}^{n} x_{j1} + z_1 = 1, \tag{5.4}$$

$$\sum_{j=1}^{n} (x_{jt} - x_{jt-p_j}) + z_t - z_{t-1} = 0 \qquad (t = 2, \ldots, T), \tag{5.5}$$

$$x_{jt} \geq 0 \qquad (j = 1, \ldots, n; \ t = 1, \ldots, T - p_j + 1),$$

$$z_t \geq 0 \qquad (t = 1, \ldots, T).$$

In the sequel, we shall refer to this reformulation as the sparse formulation.

Much more gain, however, might be expected from the application of Dantzig-Wolfe decomposition (see Chapter 1). The constraints (5.1) constitute the set of constraints in the master problem; the constraints (5.2) and the constraints $x_{jt} \geq 0$ $(j = 1, \ldots, n,\ t = 1, \ldots, T - p_j + 1)$ describe the polytope P that is written as the convex hull of its extreme points. To obtain the reformulation explicitly, we have to determine the extreme points of P.

The polytope P is described by the system

$$\begin{pmatrix} \tilde{A}^{(2)} \\ -I \end{pmatrix} x \leq \begin{pmatrix} \mathbf{1} \\ \mathbf{0} \end{pmatrix},$$

where $\tilde{A}^{(2)}$ denotes the matrix representing the constraints (5.2) and I the identity matrix. The variable x_{js} occurs in the constraint (5.2) corresponding to t if and only if $s \leq t \leq s + p_j - 1$. This means that in the matrix $\tilde{A}^{(2)}$ the column corresponding to x_{js} has a one in the positions $s, \ldots, s + p_j - 1$, i.e., the ones are in consecutive positions. Therefore, $\tilde{A}^{(2)}$ is an *interval matrix*; such matrices are known to be totally unimodular (see for example Schrijver [1986]). This implies that constraint matrix $\begin{pmatrix} \tilde{A}^{(2)} \\ -I \end{pmatrix}$, which describes the polytope P, is totally unimodular. The extreme points of P are hence integral. As the constraints (5.1) are not included in the description of P, the extreme points of P form integral 'solutions' to the original problem that satisfy the constraints (5.2) but not necessarily the constraints (5.1). Since the latter constraints state that each job has to be started exactly once, the extreme points of P can be considered as schedules in which jobs do not have to be processed exactly once. In the sequel, such schedules will be called *pseudo-schedules.*

Let x^k $(k = 1, \ldots, K)$ be the extreme points of P; hence, any $x \in P$ can be written as $\sum_{k=1}^{K} \lambda_k x^k$ for some nonnegative values λ_k such that $\sum_{k=1}^{K} \lambda_k = 1$. The master problem is then as follows:

$$\min \sum_{k=1}^{K} \Big(\sum_{j=1}^{n} \sum_{t=1}^{T-p_j+1} c_{jt} x_{jt}^k \Big) \lambda_k$$

subject to

$$\sum_{k=1}^{K} \Big(\sum_{t=1}^{T-p_j+1} x_{jt}^k \Big) \lambda_k = 1 \qquad (j = 1, \ldots, n), \tag{5.6}$$

$$\sum_{k=1}^{K} \lambda_k = 1, \tag{5.7}$$

$$\lambda_k \geq 0 \qquad (k = 1, \ldots, K).$$

Observe that the jth element of the column corresponding to λ_k, i.e., $\sum_{t=1}^{T-p_j+1} x_{jt}^k$, is equal to the number of times that job j occurs in the pseudo-schedule x^k. This means that the column corresponding to the pseudo-schedule x^k only indicates how many times each job occurs in this schedule. The cost coefficient of the variable λ_k is equal to the cost of the pseudo-schedule x^k. Consider the following two-job example with $p_1 = 2$ and $p_2 = 3$. The variable λ_k corresponding to the pseudo-schedule $x_{11} = x_{13} = 1$ has cost coefficient $c_{11} + c_{13}$ and column $(2, 0, 1)^T$, where the one in the last entry stems from the convexity constraint (5.7).

By reformulating the problem in this way, the number of constraints is decreased from $n + T$ to $n + 1$. On the other hand, the number of variables is significantly increased. Fortunately, this does not matter, since the master problem can be solved through column generation. To apply column generation, we have to find an efficient way to determine a column with minimal reduced cost, i.e., to solve the *pricing problem*. The reduced cost of the variable λ_k is given by

$$\sum_{j=1}^{n} \sum_{t=1}^{T-p_j+1} c_{jt} x_{jt}^k - \sum_{j=1}^{n} \pi_j \Big(\sum_{t=1}^{T-p_j+1} x_{jt}^k \Big) - \alpha,$$

where π_j denotes the dual variable of constraint (5.6) for job j, and α denotes the dual variable of constraint (5.7). This can be rewritten as

$$\sum_{j=1}^{n} \sum_{t=1}^{T-p_j+1} (c_{jt} - \pi_j) x_{jt}^k - \alpha.$$

Recall that each extreme point x^k represents a pseudo-schedule in which jobs do not need to be processed exactly once. Such pseudo-schedules can be represented by paths in a network in the following way. The nodes of the network correspond to time periods $1, 2, \ldots, T + 1$. The arcs correspond to the use of the machine during these time periods; for each job j and each period s, with $s \leq T - p_j + 1$, there is an arc from s to $s + p_j$ that indicates that the machine processes job j from time s to time $s + p_j$; we say that this arc *belongs* to the variable x_{js}. Furthermore, for each time period t there is an *idle time* arc from t to $t + 1$ that indicates that the machine is idle in period t. In the sequel, we denote this network by N. The path corresponding to pseudo-schedule x^k is the path containing all arcs belonging to the variables with value 1 in x^k completed by idle time arcs. From now on we refer to this path as path k. Note that the correspondence between the extreme points x^k and paths in the network N can also be established by observing that the matrix $\tilde{A}^{(2)}$ is a network matrix corresponding to the network N. Figure 5.1 depicts this network for our 2-job example with $p_1 = 2, p_2 = 3$, and $T = 6$. The solution $x_{11} = x_{13} = 1$ corresponds to the path containing the arc from 1 to 3, the arc from 3 to 5, the arc from 5 to 6 and the arc from 6 to 7.

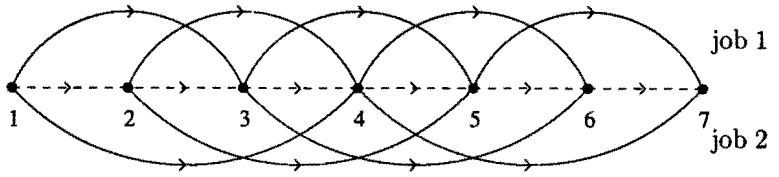


Figure 5.1: The network N for a 2-job example.

If we set the length of the arc belonging to x_{jt} equal to $c_{jt} - \pi_j$, for all j and t, and the length of all idle time arcs equal to 0 (see Figure 5.2), then the reduced cost of the variable λ_k equals the length of path k plus the dual variable α. Therefore, solving the pricing problem boils down to finding the shortest path in the network N with the lengths of the arcs defined as above. As the network is directed and acyclic, the shortest path problem can be solved in $O(nT)$ time by dynamic programming. We conclude that the pricing problem is solvable in $O(nT)$ time too.

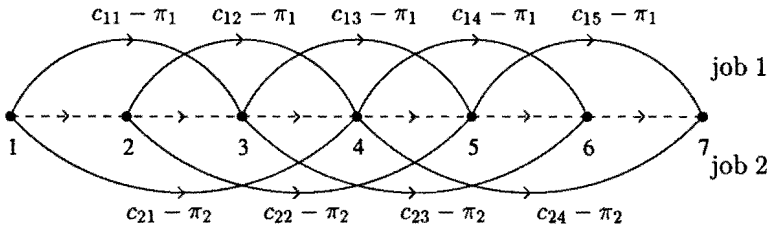


Figure 5.2: The network N for the 2-job example with cost on the arcs.

A slightly different way to view the pricing problem is the following. Note that the reduced cost of the variable λ_k is equal to $\sum_{j=1}^{n} \sum_{t=1}^{T-p_j+1} c_{jt} x_{jt}^k - \sum_{j=1}^{n} n_j^k \pi_j - \alpha$, where n_j^k denotes the number of times that job j occurs in the pseudo-schedule x^k, i.e., the number of arcs belonging to job j in path k. To achieve that the length of this path equals the reduced cost of the variable λ_k, we have to give a length c_{jt} to the arc corresponding to x_{jt}, which stems from the cost coefficient. Furthermore, as for each job j the dual variable π_j occurs in the reduced cost as many times as an arc belonging to job j occurs in the path, we have to subtract π_j from the length of each arc belonging to job j. We see that the dual variable of a constraint (5.6) occurs in the length of the arcs belonging to variables in this constraint.

Observe that the optimal solution of the master problem is given in terms of the variables λ_k and that the columns only indicate how many times each job occurs in the corresponding pseudo-schedule. Therefore, to derive the solution in terms of the original variables, we have to maintain the pseudo-schedules corresponding to the columns. Consider the following three-job example with $p_1 = 2, p_2 = 3, p_3 = 3$

and $T = 8$. Suppose that the solution in terms of the reformulation has $\lambda_k = \frac{1}{2}$ for the columns $(1, 2, 0, 1)^T$ and $(1, 0, 2, 1)^T$, where the first columns corresponds to the pseudo-schedule $x_{11} = x_{23} = x_{26} = 1$ and the second one to the pseudo-schedule $x_{31} = x_{34} = x_{17} = 1$. In terms of the original formulation this solution is given by $x_{11} = \frac{1}{2}, x_{17} = \frac{1}{2}, x_{23} = \frac{1}{2}, x_{26} = \frac{1}{2}, x_{31} = \frac{1}{2}$, and $x_{34} = \frac{1}{2}$.

It turns out that the optimal solution found by the column generation algorithm does not necessarily correspond to an extreme point in terms of the original formulation. The solution in the previous example yields such an extreme point. If, however, the pseudo-schedule corresponding to the second column is replaced by $x_{11} = x_{33} = x_{36} = 1$, then the solution in terms of the original formulation is $x_{11} = 1, x_{23} = \frac{1}{2}, x_{26} = \frac{1}{2}, x_{33} = \frac{1}{2}, x_{36} = \frac{1}{2}$, which is a convex combination of the feasible schedules $x_{11} = x_{23} = x_{36} = 1$ and $x_{11} = x_{26} = x_{33} = 1$.

We tested the performance of the column generation algorithm on the LP-relaxation of the time-indexed formulation for $1|r_j|\sum w_j C_j$. We report results for twelve sets of five randomly generated instances with uniformly distributed weights in $[1, 10]$ and uniformly distributed release dates in $[0, \frac{1}{2}\sum_{j=1}^{n} p_j]$. Half of the instances have twenty jobs, the others have thirty jobs. The processing times are in $[1, p_{\max}]$, where $p_{\max}$ equals 5, 10, 20, 30, 50, or 100. We have five instances for each combination of n and $p_{\max}$; these instances are called Rn.$\mathrm{p}_{\max}$.i, where i is the number of the instance. As in the previous chapter, our computational experiments have been conducted with MINTO 1.6/CPLEX 2.1 and have been run on a Sparc Workstation.

The computational results are given in Tables 5.1(a) and 5.1(b). These tables show the number of columns that have been generated (# col), the running time of the column generation algorithm (time cg), the time required to solve the LP-relaxation of the original formulation by the simplex method (time lp), and the time required to solve the LP-relaxation of the sparse formulation by the simplex method (time lp-sp). All running times are in seconds.

A '?' in the tables means that there was insufficient memory. These kind of problems mainly occurred for instances with $(n, p_{\max})$ equal to $(20, 100)$, $(30, 50)$, and $(30, 100)$. For these kinds of instances the size of the constraint matrix approximately equals 1500×25000, 1200×29000, and 2300×56000, respectively. The number of nonzeros is about 1,300,000, 740,000, and 2,900,000. The computational results indicate that the number of columns that are generated mainly depends on the number of jobs and hardly depends on the size of the processing times. It turns out that for problems with small processing times the simplex method is faster. For problems with larger processing times, i.e., with $p_{\max} \geq 20$, the column generation algorithm clearly outperforms the simplex method. For these large problems the simplex method requires a lot of memory as well as a large running time. Contrarily to what one might expect, the sparse formulation increases the running time of the simplex algorithm. This is due to the fact that for the sparse formulation the simplex method needs a lot of time to find a feasible basis, whereas in case of the

problem	# col	time cg	time lp	time lp-sp
R20.5.1	349	30	5	8
R20.5.2	350	28	5	8
R20.5.3	311	22	3	4
R20.5.4	360	28	4	4
R20.5.5	285	21	5	7
R20.10.1	384	46	11	17
R20.10.2	247	23	11	14
R20.10.3	228	19	12	16
R20.10.4	238	18	8	12
R20.10.5	293	28	10	13
R20.20.1	391	53	43	44
R20.20.2	278	31	45	58
R20.20.3	287	37	58	94
R20.20.4	266	29	29	70
R20.20.5	290	34	47	72
R20.30.1	299	41	65	149
R20.30.2	321	41	53	97
R20.30.3	354	49	64	125
R20.30.4	425	62	55	178
R20.30.5	257	33	69	131
R20.50.1	303	61	293	800
R20.50.2	278	48	128	288
R20.50.3	254	50	283	764
R20.50.4	259	45	262	307
R20.50.5	315	66	291	880
R20.100.1	304	102	?	3275
R20.100.2	375	116	?	4271
R20.100.3	356	127	?	5710
R20.100.4	452	150	?	3913
R20.100.5	306	103	?	4755

Table 5.1(a): Performance of the column generation algorithm and the simplex method for the 20-job instances.

problem	# col	time cg	time lp	time lp-sp
R30.5.1	747	142	8	25
R30.5.2	503	71	10	31
R30.5.3	535	81	11	30
R30.5.4	751	138	11	32
R30.5.5	763	132	8	16
R30.10.1	567	107	33	75
R30.10.2	699	140	20	80
R30.10.3	744	145	34	86
R30.10.4	635	120	29	70
R30.10.5	603	132	58	147
R30.20.1	815	249	327	675
R30.20.2	809	229	164	439
R30.20.3	548	129	219	549
R30.20.4	531	110	160	264
R30.20.5	521	118	173	565
R30.30.1	584	173	289	1150
R30.30.2	812	338	?	1951
R30.30.3	666	219	523	1114
R30.30.4	627	202	408	1284
R30.30.5	545	181	?	1612
R30.50.1	534	244	?	3868
R30.50.2	679	290	?	2135
R30.50.3	569	233	?	5249
R30.50.4	714	312	?	3786
R30.50.5	615	261	?	3954
R30.100.1	587	451	?	?
R30.100.2	639	450	?	?
R30.100.3	553	374	?	?
R30.100.4	714	589	?	?
R30.100.5	802	482	?	?

Table 5.1(b): Performance of the column generation algorithm and the simplex method for the 30-job instances.

original formulation the columns of the slack variables are unit-vectors, which can easily be included in the basis. The sparse formulation, however, enables us to solve larger problems.

To investigate the performance of the simplex and interior point method on the largest problems, Bixby [1994] solved some of the LP-relaxations using CPLEX 3.0 on a Sparc 10, which is faster than our workstation and also has more memory. His results are given in Table 5.2. The first three lines in this table are based on the original formulation. The last line on the sparse formulation. The '$*$' indicates that steepest edge pricing was applied. The results indicate that for instances with larger processing times our column generation algorithm is also faster than interior point methods.

problem	time primal simplex	time dual simplex	time interior point
R20.20.2	12.5	29.0	19.9
R20.50.2	42.5	172.2	95.3
R30.100.1	6471.0	?	2990.7
R30.100.1(SP)	1689.0*	?	793.0

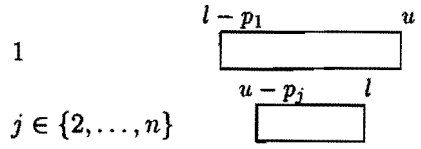
Table 5.2: Performance of simplex and interior point methods.

5.3 Combining row and column generation

In the previous chapter, we have demonstrated that cutting planes can be used to obtain very strong lower bounds from the time-indexed formulation. Since for instances with large processing times the LP-relaxation is solved much faster by column generation, it is natural to examine whether cut generation can be combined with column generation. The main difficulty is that the pricing problem can become much more complicated after the addition of extra constraints, since each constraint that is added to the master problem introduces a new dual variable. For a discussion of the complexity of combining row and column generation we refer to Barnhart et al. [1994].

In this section we show how column and row generation can be combined in our application. We discuss the combination of column generation with the addition of facet inducing inequalities with right hand side 1 or 2, which have been characterized in Chapter 3.

We first consider facet inducing inequalities with right hand side 1. Such inequalities are denoted by $x(V) \leq 1$, which is a short notation for $\sum_{(j,s)\in V} x_{js} \leq 1$. We proved that V is given by $\{(1,s) \mid s \in [l-p_1, u]\} \cup \{(j,s) \mid j \neq 1, s \in [u-p_j, l]\}$, for some l and u with $l < u$ and some special job, which for presentational convenience is assumed to be job 1. Such an inequality can be represented by the following diagram:



Suppose that we add such an inequality $x(V) \leq 1$ to the Dantzig-Wolfe master problem. In terms of the variables λ_k in the master problem it is given by

$$\sum_{k=1}^{K} (\sum_{(j,s) \in V} x_{js}^k) \lambda_k \leq 1.$$

We see that the coefficient of λ_k equals the number of variables in V in the pseudo-schedule x^k, i.e., the number of arcs in the corresponding path that belong to a variable in V. We denote this number by n_V^k; note that n_V^k is readily determined and that all the arcs belonging to a variable in V leave from a vertex corresponding to a period between $l - p_{\max}$ and u. It is hence not hard to express inequalities of this type in terms of the reformulation, which implies that these constraints can easily be added and that it is not hard to compute elements of newly generated columns corresponding to additional constraints. After a facet inducing inequality $x(V) \leq 1$ has been added, the master problem becomes:

$$\min \sum_{k=1}^{K} (\sum_{j=1}^{n} \sum_{t=1}^{T-p_j+1} c_{jt} x_{jt}^k) \lambda_k$$

subject to

$$\sum_{k=1}^{K} n_j^k \lambda_k = 1 \qquad (j = 1, \ldots, n),$$

$$\sum_{k=1}^{K} \lambda_k = 1,$$

$$\sum_{k=1}^{K} n_V^k \lambda_k \leq 1,$$

$$\lambda_k \geq 0 \qquad (k = 1, \ldots, K).$$

We denote the dual variable of the additional constraint by μ_1. The reduced cost of the variable λ_k is given by

$$\sum_{j=1}^{n} \sum_{t=1}^{T-p_j+1} c_{jt} x_{jt}^k - \sum_{j=1}^{n} n_j^k \pi_j - \alpha - n_V^k \mu_1.$$

It is easy to see that the pricing problem boils down to determining the shortest path in the network N, where the length of the arc belonging to the variable x_{js} equals $c_{js} - \pi_j - \mu_1$ if $(j,s) \in V$ and $c_{js} - \pi_j$ if $(j,s) \notin V$; the length of the idle time arcs is equal to zero. In fact, we solve the original pricing problem in which μ_1 has been subtracted from the length of the arcs belonging to variables in V. If already some constraints have been added, then the length of the arcs is adapted in the same way for each additional constraint. Hence, the addition of this kind of constraints does not change the structure of the pricing problem. We conclude that we can combine column generation with the addition of facet inducing inequalities with right hand side 1.

The addition of facet inducing inequalities with right hand side 2 proceeds in a similar way. As in Chapter 3, a valid inequality with right hand side 2 is denoted by $x(V^1)+2x(V^2) \leq 2$, which is a short notation for $\sum_{(j,s)\in V^1} x_{js}+2\sum_{(j,s)\in V^2} x_{js} \leq 2$, where $V^1 \cap V^2 = \emptyset$; recall that all facet inducing inequalities with right hand side 2 belong to the classes (3.4), (3.5), and (3.6), which have been described in Chapter 3. In terms of the variables λ_k in the master problem, such an inequality is given by

$$\sum_{k=1}^{K}(\sum_{(j,s)\in V^1} x_{js}^k + 2\sum_{(j,s)\in V^2} x_{js}^k)\lambda_k \leq 2.$$

The coefficient of λ_k equals the value of the left hand side of the original inequality obtained by substitution of the pseudo-schedule x^k; it hence equals the number of arcs in the corresponding path that belong to variables in V^1 plus two times the number of such arcs belonging to V^2. We denote these numbers by $n_{V^1}^k$ and $n_{V^2}^k$, respectively. Just like in the case of facet inducing inequalities with right hand side 1, these numbers can easily be computed, which implies that it is easy to express facet inducing inequalities with right hand side 2 in terms of the reformulation. Observe that if an inequality $x(V^1) + 2x(V^2) \leq 2$ has been added to the master problem, then the reduced cost is given by

$$\sum_{j=1}^{n}\sum_{t=1}^{T-p_j+1} c_{jt}x_{jt}^k - \sum_{j=1}^{n} n_j^k\pi_j - \alpha - (n_{V^1}^k + 2n_{V^2}^k)\mu_2,$$

where μ_2 is the dual variable corresponding to the additional constraint. We again have to solve the original pricing problem in which μ_2 has been subtracted from the length of the arcs belonging to a variable in V^1 and $2\mu_2$ from the length of the arcs belonging to a variable in V^2. Clearly, several facet inducing inequalities with right hand side 2 can be added in this way. The same holds for a combination of facet inducing inequalities with right hand sides 1 and 2.

Summarizing, we conclude that for each additional constraint the coefficient of λ_k is equal to the value of the left hand side of the constraint obtained when the pseudo-schedule x^k is substituted. Furthermore, the dual variables of additional

constraints do not change the structure of the pricing problem; they only change the coefficients of the objective function.

We show that this principle also holds in other contexts in which column generation based on a reformulation obtained through Dantzig-Wolfe decomposition is combined with the addition of constraints that are given in terms of the original formulation.

Consider the linear programming problem that is defined by $\min\{cx \mid A^{(1)}x \leq b^{(1)}, A^{(2)}x \leq b^{(2)}\}$, where $c \in \mathcal{R}^n$, $A^{(1)} \in \mathcal{R}^{m_1 \times n}$, and $b^{(1)} \in \mathcal{R}^{m_1}$, and where for presentational convenience, we assume that $A^{(2)}x \leq b^{(2)}$ describes a bounded set and hence a polytope. Let x^k $(k = 1, \ldots, K)$ be the extreme points of this polytope. The master problem obtained through Dantzig-Wolfe decomposition is as follows:

$$\min \sum_{k=1}^{K}(\sum_{j=1}^{n} c_j x_j^k)\lambda_k$$

subject to

$$\sum_{k=1}^{K}(\sum_{j=1}^{n} a_{ij}^{(1)} x_j^k)\lambda_k \leq b_i^{(1)} \qquad (i = 1, \ldots, m_1),$$

$$\sum_{k=1}^{k} \lambda_k = 1,$$

$$\lambda_k \geq 0 \qquad (k = 1, \ldots, K).$$

The reduced cost of the variable λ_k is equal to

$$\sum_{j=1}^{n} c_j x_j^k - \sum_{i=1}^{m_1} \pi_i (\sum_{j=1}^{n} a_{ij}^{(1)} x_j^k) - \alpha,$$

where π_i denotes the dual variable of the ith constraint and α the dual variable of the convexity constraint. The pricing problem can hence be written as

$$\min\{\sum_{j=1}^{n}(c_j - \sum_{i=1}^{m_1} \pi_i a_{ij}^{(1)})x_j^k - \alpha \mid k = 1, \ldots, K\}.$$

Theorem 5.1 *If the pricing problem is solved without using some special structure of the cost coefficients c_j, then the addition of a valid inequality $dx \leq d_0$ stated in terms of the original formulation does not complicate the pricing problem.*

Proof. In terms of the Dantzig-Wolfe reformulation the inequality $dx \leq d_0$ is given by

$$\sum_{k=1}^{K}(dx^k)\lambda_k \leq d_0.$$

The coefficient of λ_k is hence equal to the value of the left hand side of the inequality when the extreme point x^k is substituted. Observe that after the addition of this inequality the reduced cost is given by

$$\sum_{j=1}^{n} c_j x_j^k - \sum_{i=1}^{m_1} \pi_i (\sum_{j=1}^{n} a_{ij}^{(1)} x_j^k) - \alpha - \sigma (\sum_{j=1}^{n} d_j x_j^k),$$

where σ denotes the dual variable of the additional constraint. The pricing problem is hence given by

$$\min\{\sum_{j=1}^{n} (c_j - \sum_{i=1}^{m_1} \pi_i a_{ij}^{(1)} - \sigma d_j) x_j^k - \alpha \mid k = 1, \ldots, K\}.$$

Observe that the new pricing problem differs from the original pricing problem only in the objective coefficients. Hence, if we can solve the pricing problem without using some special structure of the objective coefficients, i.e., we can solve this problem for arbitrary c_j, then the addition of constraints does not complicate the pricing problem. □

If, on the other hand, some valid inequality $\sum_{k=1}^{K} g_k \lambda_k \leq g_0$ stated in terms of the reformulation has been added, then the pricing problem becomes $\min\{\sum_{j=1}^{n} (c_j - \sum_{i=1}^{m_1} \pi_i a_{ij}^{(1)}) x_j^k - \alpha - \sigma g_k \mid k = 1, \ldots, K\}$. This means that by the addition of this constraint some extra term g_k is included in the cost of each feasible solution x^k to the pricing problem. As there may be no obvious way to transform the cost g_k into costs on the variables x_j^k, the additional constraint can complicate the structure of the pricing problem. An example of this situation is the addition of clique constraints to the set partitioning formulation of the generalized assignment problem that was discussed by Savelsbergh [1993].

We implemented a cutting plane algorithm based on facet inducing inequalities with right hand side 1 in which the linear programs were solved by column generation. In our application it turns out to be beneficial to add only a small number of violated inequalities in each separation step. The reason is that in this way we limit the number of rows in the linear programs and the number of dual variables that are involved in the pricing problem. We report preliminary results for the problems Rn.$p_{\max}$.i with $n = 20$ and $p_{\max} = 5, 10, 20, 30, 50$, and $n = 30$ and $p_{\max} = 5, 10, 20, 30$. It appears that for larger instances there is not enough memory. We omit instances for which the LP-relaxation has an integral solution. The results concerning the case in which only the most violated inequality is added in each separation step are found in Tables 5.3(a) and 5.3(b). They show the number of columns that have been generated (# col), the number of inequalities that have been added (# inq cg), and the running time (time cg). They further show the number of added inequalities (# inq orf) and the running time (time orf) when the

original formulation was used. A '?' in the tables means that there is not enough memory.

problem	# col	# inq cg	time cg	# inq orf	time orf
R20.5.3	648	2	78	1	4
R20.5.5	473	1	46	1	5
R20.10.1	955	14	221	9	18
R20.10.2	852	9	169	10	18
R20.10.3	629	22	150	13	18
R20.10.4	1105	33	454	30	19
R20.10.5	766	3	117	5	13
R20.20.2	1404	21	636	21	87
R20.20.3	410	1	58	1	62
R20.20.4	1358	34	673	36	84
R20.20.5	641	7	127	7	63
R20.30.1	?	?	?	109	457
R20.30.2	693	3	125	3	72
R20.30.3	1119	16	392	12	101
R20.30.5	2336	31	1834	39	163
R20.50.1	1420	22	690	?	?
R20.50.2	950	14	309	?	?
R20.50.3	2596	44	3456	?	?
R20.50.4	2955	49	4443	?	?
R20.50.5	974	5	297	?	?

Table 5.3(a): Combining row and column generation for the 20-job problems.

These results indicate that in our current implementation the combination of row and column generation requires a lot of running time and memory. However, column generation enables us to solve larger problems, such as the instances with $n = 20, p_{\max} = 50$ and $n = 30, p_{\max} = 30$.

It turns out that the major part of the computation time is used for solving linear programming problems. Each time a column generation step is performed a linear program has to be solved. Since we add one column at a time this means that a linear program has to be solved for each generated column. As the size of these linear programs increases during the process, this may result in large running times.

Therefore, it might be interesting to delete previously generated columns whose reduced cost have a large positive value; it is very unlikely that such columns will enter the basis again. Besides the running time, this may also reduce the amount

problem	# col	# inq cg	time cg	# inq orf	time orf
R30.5.1	2804	16	2559	14	13
R30.5.2	1753	15	1091	12	16
R30.5.3	1330	11	653	7	14
R30.5.4	1378	10	691	6	13
R30.5.5	2324	12	1683	10	12
R30.10.1	3580	28	5464	32	70
R30.10.2	3936	11	5002	9	28
R30.10.3	3075	30	3903	25	60
R30.10.4	?	?	?	55	85
R30.10.5	3059	22	3876	21	100
R30.20.1	3468	31	5751	27	650
R30.20.2	4311	22	5819	22	248
R30.20.3	3065	33	5204	34	343
R30.20.4	?	?	?	64	351
R30.20.5	?	?	?	96	679
R30.30.1	4470	39	12762	?	?
R30.30.2	1990	17	3407	?	?
R30.30.3	4550	48	12711	?	?
R30.30.4	?	?	?	?	?
R30.30.5	?	?	?	?	?

Table 5.3(b): Combining row and column generation for the 30-job problems.

of memory that is required. This is due to the fact that we do not only have to include each column in the constraint matrix, but we also have to maintain the pseudo-schedules corresponding to each column.

It may also be worthwhile to investigate column generation schemes in which more than one column is added in each iteration. In this way fewer column generation steps have to be performed and hence fewer linear programs have to be solved. These issues are interesting topics for further research.

5.4 Branching

Recall that if we do not find an integral solution by the addition of cutting planes to the LP-relaxation, then we proceed by branching. In this section we investigate what kind of branching strategies can be combined with the column generation algorithm. We show that all branching strategies from the previous chapter can be applied. As in our application the addition of facet inducing inequalities with right hand side 1 or 2 does not change the structure of the pricing problem, these branching

strategies can also be applied in case of combined column and row generation, i.e., it is possible to combine a branch-and-price algorithm with the addition of facet inducing inequalities with right hand side 1 or 2.

In each node of the branch-and-bound tree we are only allowed to generate columns that satisfy the constraints defining the subproblem in that node. Some well-known branching strategies are based on variable fixing. Given a fractional variable λ_k, we split the problem into two subproblems that are defined by the constraints $\lambda_k = 0$ and $\lambda_k = 1$. It is then very likely that in the subproblem defined by $\lambda_k = 0$ the column corresponding to λ_k is the optimal solution to the pricing problem and will hence be generated again. This implies that this solution has to be excluded from the pricing problem. If more variables are fixed, then more solutions have to be excluded. It turns out that this branching strategy complicates the pricing problem too much.

This indicates that we have to choose a branching strategy that does not destroy the structure of the pricing problem. We show that in our application any branching strategy that fixes variables in the original formulation can be applied. Suppose that in a subproblem the variable x_{js} is fixed at zero. Then we are only allowed to generate columns that represent a path not containing the arc belonging to the variable x_{js}. This can be achieved by omitting this arc from the network N. Suppose, on the other hand, that this variable is fixed at one. Then all columns that are generated have to correspond to paths containing the arc belonging to x_{js}, i.e., the path determined by the pricing problem has to contain the arc from s to $s + p_j$ corresponding to job j. This means that the pricing problem decomposes into two subproblems. We have to determine the shortest path from 0 to s and the shortest path from $s + p_j$ to $T + 1$.

From the above, we conclude that the branching strategies that were discussed the previous chapter can be applied in combination with column generation. Especially, the strategy *pos* that fixes jobs at certain positions in the schedule can be applied, since it fixes variables at 1. SOS-branching fixes the sum of a set of variables at 0. Hence, it fixes a set of variables at 0. It follows that SOS-branching can be applied by omitting a set of arcs from the network N.

5.5 Key formulations

In this section we discuss so-called key formulations. The main idea behind *key formulations* is to reformulate the master problem by choosing a key extreme point and writing the columns corresponding to the other extreme points as their difference with the column corresponding to the key extreme point. Key formulations allow implicit handling of the convexity constraints, which reduces the number of constraints. They also reduce the number of nonzeros in the constraint matrix. As our master problem contains only one convexity constraint, we only gain in the

second respect. More benefit might be expected from *alternative key formulations.* The idea behind these formulations is to represent columns as combinations of elementary columns. The use of these formulations might reduce the number of columns that are used by the column generation algorithm. As we have not implemented the application of key formulations or alternative key formulations, we restrict ourselves to a theoretical description.

Most column generation algorithms are based on formulations that can be obtained through Dantzig-Wolfe decomposition. Dantzig and Van Slyke [1967] and Rosen [1964] present specialized solution procedures for linear programs that have the structure of a Dantzig-Wolfe master problem. Both these procedures use key formulations. The use of key formulations in column generation algorithms and some of its applications are discussed by Barnhart et al. [1994].

Recall that the main idea of key formulations is to reformulate the master problem by choosing a *key* extreme point x^* and replacing the corresponding variable λ_{k^*} by $1 - \sum_{k \neq k^*} \lambda_k$. As in our case the extreme points represent paths, the key extreme point is called the *key path.* We obtain the following key formulation:

$$\min \sum_{k \neq k^*} \Big(\sum_{j=1}^{n} \sum_{t=1}^{T-p_j+1} c_{jt}(x_{jt}^k - x_{jt}^*) \Big) \lambda_k + \sum_{j=1}^{n} \sum_{t=1}^{T-p_j+1} c_{jt} x_{jt}^*$$

subject to

$$\sum_{k \neq k^*} \Big(\sum_{t=1}^{T-p_j+1} (x_{jt}^k - x_{jt}^*) \Big) \lambda_k = 1 - \sum_{t=1}^{T-p_j+1} x_{jt}^* \qquad (j = 1, \ldots, n), \tag{5.8}$$

$$\sum_{k \neq k^*} \lambda_k \leq 1, \tag{5.9}$$

$$\lambda_k \geq 0 \qquad (k = 1, \ldots, K, k \neq k^*).$$

The coefficient of λ_k in the constraint (5.8) corresponding to job j equals the difference between the number of times that job j occurs in the pseudo-schedule x^k and the number of times that this job occurs in the key path x^*. Constraint (5.9) states that the variable corresponding to the key path is nonnegative. The reduced cost of the variable λ_k is given by

$$\sum_{j=1}^{n} \sum_{t=1}^{T-p_j+1} c_{jt}(x_{jt}^k - x_{jt}^*) - \sum_{j=1}^{n} \pi_j \Big(\sum_{t=1}^{T-p_j+1} (x_{jt}^k - x_{jt}^*) \Big) - \alpha,$$

where π_j denotes the dual variable of constraint (5.8) for job j and α denotes the dual variable of constraint (5.9). This can be rewritten as

$$\sum_{j=1}^{n} \sum_{t=1}^{T-p_j+1} (c_{jt} - \pi_j) x_{jt}^k - \alpha - \sum_{j=1}^{n} \sum_{t=1}^{T-p_j+1} (c_{jt} - \pi_j) x_{jt}^*.$$

As the last two terms are independent of k, the pricing problem does not change by this reformulation. Note that although the key path is a feasible solution to this pricing problem, it does not correspond to a column in the reformulation. As this path is implicitly included in any restricted problem occurring during the column generation algorithm, the column corresponding to the key path will not be identified as a column with negative reduced cost.

Key formulations have mainly been used to be able to omit the convexity constraints, and thus to reduce the number of rows. The convexity constraints ensure the nonnegativity of the variables belonging to the key extreme points; these constraints can be handled implicitly by changing the key extreme point whenever the corresponding variable tends to become negative. Especially in case of problems with many subproblems, such as for example the multi-commodity flow problem in which the flow of each commodity is represented by a subset of the columns, the application of key formulations has been reported to yield a considerable reduction in computation time [Barnhart et al., 1991]. Since our problem contains only one convexity constraint, the advantage of applying key formulations is limited from this respect. We still have the advantage, however, that the reformulation provides us with a constraint matrix that contains fewer nonzero elements.

More benefit may be expected from the application of *alternative* key formulations. The idea behind the alternative key formulation is to represent columns as combinations of elementary columns. Barnhart et al. [1994] describe the application of alternative key formulations to the multi-commodity flow problem. Our problem is similar to this problem in the sense that in the column generation formulations of both problems the columns represent paths in a network. Therefore, we can apply key formulations in a way similar to theirs.

In the key formulation of our problem, each column corresponds to the difference of the key path and some other path in the network. It is easy to see that this difference boils down to a set of disjoint cycles. Any column in the key formulation can hence be written as the sum of a set of columns that all represent a single cycle. The alternative key formulation uses such columns; these are called simple columns. Since several single cycles can be combined with the key path to obtain a new path, we have to allow for solutions with $\sum_{k \neq k^*} \lambda_k > 1$, which means that the constraint (5.9) is no longer valid. We replace it by a set of constraints, one constraint for each arc on the key path to ensure that the arc is exchanged at most once. Note that in this way the number of constraints is increased. As in the case of a standard key formulation, however, the convexity constraints can be handled implicitly by changing the key path whenever one of the arcs on this path tends to be exchanged more than once. The main advantage that can be expected from the alternative key formulation approach is that fewer columns need to be included in the formulation. Computational advantages of this approach have been reported by Vance [1993].

Let the key path consist of the arcs $a(j_1, s_1), a(j_2, s_2), \ldots, a(j_L, s_L)$, where $a(j, s)$ denotes the arc belonging to the variable x_{js} if $j \in \{1, \ldots, n\}$ and the idle time arc of period s if $j = 0$. We assume without loss of generality that $s_1 < s_2 < \ldots < s_L$. Let x^i $(i \in I)$ be the set of pseudo-schedules that correspond to paths that differ from the key path by a single cycle, and let C_i $(i \in I)$ be the set of corresponding cycles. Clearly, $I \subseteq K$. The alternative key formulation is as follows:

$$\min \sum_{i \in I} (\sum_{j=1}^{n} \sum_{t=1}^{T-p_j+1} c_{jt}(x^i_{jt} - x^*_{jt}))\lambda_i + \sum_{j=1}^{n} \sum_{t=1}^{T-p_j+1} c_{jt} x^*_{jt}$$

subject to

$$\sum_{i \in I} (\sum_{t=1}^{T-p_j+1} (x^i_{jt} - x^*_{jt}))\lambda_i = 1 - \sum_{t=1}^{T-p_j+1} x^*_{jt} \quad (j = 1, \ldots, n), \tag{5.10}$$

$$\sum_{i : a(j_l, s_l) \in C_i} \lambda_i \leq 1 \quad (l = 1, \ldots, L), \tag{5.11}$$

$$\lambda_i \geq 0 \quad (i \in I).$$

Note that the constraints (5.10) state that each key path arc is exchanged at most once. In general, the change of the convexity constraints may weaken the LP-relaxation. If, for example, the convexity constraints in the alternative key formulation are given by

$$\begin{aligned} \lambda_1 + \lambda_2 &\leq 1, \\ \lambda_1 + \lambda_3 &\leq 1, \\ \lambda_2 + \lambda_3 &\leq 1, \end{aligned}$$

then the solution $\lambda_1 = \lambda_2 = \lambda_3 = \frac{1}{2}$ is feasible with respect to these constraints, but is excluded by the convexity constraint in the standard key formulation. In our application, however, this complication does not occur due to the total unimodularity of the matrix representing the convexity constraints. The total unimodularity follows from the fact that any single cycle contains a set of consecutive arcs of the key path; each column of the matrix hence has its ones in consecutive positions.

In the column generation framework we may generate columns that are combinations of simple columns. Each column that is generated, however, can be decomposed into simple columns; only the simple columns that are not already present in the restricted master problem are added. The reduced cost of a column equals the sum of the reduced costs of the simple columns of which it is composed. The reduced cost of the simple columns corresponding to λ_i is given by

$$\sum_{j=1}^{n} \sum_{t=1}^{T-p_j+1} c_{jt}(x^i_{jt} - x^*_{jt}) - \sum_{j=1}^{n} \pi_i \sum_{t=1}^{T-p_j+1} (x^i_{jt} - x^*_{jt}) - \sum_{l : a(j_l, s_l) \in C_i} \sigma_l, \tag{5.12}$$

where π_j denotes the dual variable of constraint (5.10) corresponding to job j and σ_l the dual of constraint (5.11) for key path arc $a(j_l, s_l)$. Now consider the reduced cost of a column that is the sum of several simple columns; suppose that this column corresponds to path k and that it is composed of the columns of the variables λ_i with $i \in I'$. The reduced cost of this column equals the sum over $i \in I'$ of the cost (5.12). As $x^k - x^* = \sum_{i \in I'}(x^i - x^*)$, this cost is given by

$$\sum_{j=1}^{n} \sum_{t=1}^{T-p_j+1} c_{jt}(x^k_{jt} - x^*_{jt}) - \sum_{j=1}^{n} \pi_i \sum_{t=1}^{T-p_j+1} (x^k_{jt} - x^*_{jt}) - \sum_{i \in I'} \sum_{l:a(j_l,s_l) \in C_i} \sigma_l.$$

This can be rewritten as

$$\sum_{j=1}^{n} \sum_{t=1}^{T-p_j+1} (c_{jt} - \pi_j) x^k_{jt} - \sum_{i \in I'} \sum_{l:a(j_l,s_l) \in C_i} \sigma_l - \sum_{j=1}^{n} \sum_{t=1}^{T-p_j+1} (c_{jt} - \pi_j) x^*_{jt}.$$

These reduced costs are equal to the reduced costs in the standard key formulation except for the term containing the dual variables of the convexity constraints: this term has become dependent of the path k. It equals the sum of the dual variables of the convexity constraints belonging to those key path arcs that are not in the path k. The pricing problem still boils down to finding the shortest path in the network N. To achieve that the length of a path is equal to the reduced cost of the corresponding column, the lengths of the arcs have to be set as follows. The length of the arcs on the key path is equal to the length in the original pricing problem. The length of the arcs outside the key path has to be set such that, for any path k, the sum of the dual variables of the convexity constraints belonging to the key path arcs not in the path k is included in the length of this path. This can be achieved by subtracting the sum of the dual variables corresponding to the part of the key path that is overlapped by an arc outside the key path from its length in the original pricing problem. To be more precise, for the arc $a(j, s)$ we define $S(j, s)$ as the time interval between its endpoints, i.e., $S(j, s) = [s, s + p_j]$ for an arc corresponding to job $j \in \{1, 2, \ldots, n\}$ and $S(j, s) = [s, s + 1]$ for idle time arcs. By $|S|$ we denote the number of time periods in the interval S. Let $a(j, s)$ be an arc outside the key path. By definition, it overlaps $|S(j, s)|$ time periods. For a key path arc $a(j_l, s_l)$ it overlaps $|S(j, s) \cap S(j_l, s_l)|$ periods that are also overlapped by this key path arc, and it hence overlaps a fraction $\frac{|S(j,s) \cap S(j_l,s_l)|}{|S(j_l,s_l)|}$ of this key path arc. Therefore, for each key path arc we subtract this fraction times the dual variable σ_l from the original length of the arc. The length of arc $a(j, s)$ belonging to job $j \in \{1, 2, \ldots, n\}$ has to be set equal to

- $c_{js} - \pi_j$ if $a(j, s)$ is on the key path;
- $c_{js} - \pi_j - \sum_{l=1}^{L} \frac{|S(j,s) \cap S(j_l,s_l)|}{|S(j_l,s_l)|} \sigma_l$ otherwise.

Analogously, the length of an idle time arc $a(j,s)$ has to be set equal to

- 0 if $a(j,s)$ is on the key path;
- $-\sum_{l=1}^{L} \frac{|S(j,s) \cap S(j_l,s_l)|}{|S(j_l,s_l)|} \sigma_l$ otherwise.

We conclude that for alternative key formulations the pricing problem still boils down to finding the shortest path in the network N; the costs on the arcs are however different from the costs in case of the original formulation. Computational experiments have to show whether the use of alternative key formulations improves the column generation algorithm.

Bibliography

[1] E. BALAS (1985). On the facial structure of scheduling polyhedra. *Mathematical Programming Study 24*, 179-218.

[2] C. BARNHART, C.A. HANE, E.L. JOHNSON, AND C.G. SIGISMONDI (1991). *An alternative formulation and solution strategy for multi-commodity network flow problems.* Report COC-9102, Georgia Institute of Technology, Atlanta.

[3] C. BARNHART, E.L. JOHNSON, G.L. NEMHAUSER, M.W.P. SAVELSBERGH, AND P.H. VANCE (1994). *Branch-and-price: column generation for solving huge integer programs.* Report COC-9403, Georgia Institute of Technology, Atlanta.

[4] 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.

[5] B. BIXBY (1994). Private communication.

[6] V. CHÁTAL (1983). *Linear programming.* W.H. Freeman and Company, New York.

[7] Y. CRAMA AND F.C.R. SPIEKSMA (1993). *Scheduling jobs of equal length: complexity and facets.* Report M 93-09, Faculty of Economics, University of Limburg, Maastricht.

[8] H. CROWDER, E.L. JOHNSON, AND M.W. PADBERG (1983). Solving large-scale zero-one linear programming problems. *Operations Research 31*, 803-834.

[9] G.B. DANTZIG (1963). *Linear Programming and Extensions.* Princeton University Press, Princeton.

[10] G.B. DANTZIG AND R.M. VAN SLYKE (1967). Generalized upper bounding techniques. *Journal of Computer System Science 1*, 213-266.

[11] M.E. DYER AND L.A. WOLSEY (1990). Formulating the single machine sequencing problem with release dates as a mixed integer program. *Discrete Applied Mathematics 26*, 255-270.

[12] A.M. GEOFFRION (1974). Lagrangean relaxation for integer programming. *Mathematical Programming Studies 2*, 82-114.

[13] M. GRÖTSCHEL, M. JÜNGER, AND G.REINELT (1985). Facets of the linear ordering polytope. *Mathematical Programming 33*, 43-60.

[14] R. GILES AND W.R. PULLEYBLANK (1979). Total dual integrality and integral polyhedra. *Linear Algebra and Its Applications 25*, 191-196.

[15] N.G. HALL (1986). Scheduling problems with generalized due dates. *IEEE Transactions 18*, 220-222.

[16] N.G. HALL, S.P. SETHI, AND C. SRISKANDARAJAH (1991). On the complexity of generalized due dates scheduling problems. *European Journal of Operations Research 51* 100-109.

[17] K.L. HOFFMAN AND M.W. PADBERG (1985). LP-based combinatorial problem solving. *Annals of Operations Research 4*, 145-194.

[18] N. KARMARKAR (1984). A new polynomial-time algorithm for linear programming. *Combinatorica 4*, 373-395.

[19] L.G. KHACHIYAN (1979). A polynomial algorithm in linear programming (in Russian). *Doklady Akademii Nauk SSSR 244*, 1093-1096. English translation: *Sviet Mathematics Doklady 20*, 191-194.

[20] J.B. LASSERRE AND M. QUEYRANNE (1992). Generic scheduling polyhedra and a new mixed-integer formulation for single-machine scheduling. E. BALAS, G. CORNUEJOLS, R. KANNAN (EDS.). *Integer programming and combinatorial optimization*, Proceedings of the IPCO-conference held at Carnegie Mellon University, University Printing and Publications, Carnegie Mellon University, Pittsburgh.

[21] E.L. LAWLER (1978). Sequencing jobs to minimize total weighted completion time subject to precedence constraints. *Annals of Discrete Mathematics 2*, 75-90.

[22] J.K. LENSTRA, A.H.G. RINNOOY KAN, AND P. BRUCKER (1977). Complexity of machine scheduling problems. *Annals of Discrete Mathematics 1*, 343-362.

[23] J.K. LENSTRA AND A.H.G RINNOOY KAN (1978). Complexity of scheduling under precedence constraints. *Operations Research 26*, 22-35.

[24] H. MINKOWSKI (1896). *Geometrie der Zahlen (Erste Lieferung)*. Teubner, Leipzig. Reprinted: Chelsea, New York, 1967.

[25] R. MÜLLER (1994). The linear ordering polytope as a basis for integer programs for machine scheduling. *Proceedings of the Fourth International Workshop on Project Management and Scheduling*, Leuven.

[26] G.L. NEMHAUSER AND M.W.P. SAVELSBERGH (1992). A cutting plane algorithm for the single machine scheduling problem with release times. M. AKGÜL, H. HAMACHER, AND S. TUFECKI (eds.). *Combinatorial Optimization: New Frontiers in the Theory and Practice*, NATO ASI Series F: Computer and Systems Sciences 82, Springer, Berlin, 63-84.

[27] G.L. NEMHAUSER, M.W.P SAVELSBERGH, AND G.C. SIGISMONDI (1994). MINTO, a Mixed INTeger Optimizer. *Operations Research Letters 15*, 47-58.

[28] G.L. NEMHAUSER AND L.A. WOLSEY (1988). *Integer and Combinatorial Optimization*. Wiley, New York.

[29] M.W. PADBERG AND G. RINALDI (1991). A branch-and-cut algorithm for the resolution of large-scale symmetric traveling salesman problems. *SIAM Review 33*, 60-100.

[30] R. PETERS (1988). L'ordonnancement sur une machine avec des constraints de délai. *Belgian Journal of Operations Research, Statistics and Computer Science 28*, 33-76.

[31] M. QUEYRANNE (1986). *Structure of a simple scheduling polyhedron*. Working paper, Faculty of Commerce, University of British Columbia, Vancouver.

[32] M. QUEYRANNE AND Y. WANG (1991A). Single machine scheduling polyhedra with precedence constraints. *Mathematics of Operations Research 16*, 1-20.

[33] M. QUEYRANNE AND Y. WANG (1991B). *Single machine scheduling: polyhedral computations*. Working paper, Faculty of Commerce, University of British Columbia, Vancouver.

[34] M.W.P. SAVELSBERGH (1993). *A Branch-and-price algorithm for the generalized assignment problem*. Report COC-93-02, Georgia Institute of Technology, Atlanta.

[35] J.B. ROSEN (1964). Primal partition programming for block diagonal matrices. *Numerische Mathematik 6*, 250-260.

[36] A. SCHRIJVER (1986). *Theory of Linear and Integer Programming.* Wiley, New York.

[37] W.E. SMITH (1956). Various optimizers for single-stage production. *Naval Research Logistics Quarterly 3*, 59-66.

[38] J.P. DE SOUSA (1989). *Time-indexed formulations of non-preemptive single-machine scheduling problems.* PhD-thesis, Catholic University of Louvain, Louvain-la-Neuve.

[39] 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.

[40] F.C.R. SPIEKSMA (1991). Private communication.

[41] P.H. VANCE (1993). *Crew scheduling, cutting stock, and column generation: solving huge integer programs.* PhD dissertation, School of Industrial and Systems Engineering, Georgia Institute of Technology, Atlanta, Georgia.

[42] J.M. VAN DEN AKKER, C.P.M. VAN HOESEL, AND M.W.P. SAVELSBERGH (1993). *Facet inducing inequalities for single-machine scheduling problems*, Memorandum COSOR 93-27, Eindhoven University of Technology, Eindhoven.

[43] J.M. VAN DEN AKKER, C.A.J. HURKENS, AND M.W.P. SAVELSBERGH (1994A). *A time-indexed formulation for single-machine scheduling problems part 2: separation and computation.* In preparation.

[44] J.M. VAN DEN AKKER, C.A.J. HURKENS, AND M.W.P. SAVELSBERGH (1994B). *Column generation for single-machine scheduling problems.* In preparation.

[45] A. VON ARNIM AND R SCHRADER (1993). *The permutahedron of P_4-sparse posets.* Report No. 93.143, Department of Applied Mathematics and Computer Science, University of Cologne, Cologne.

[46] A. VON ARNIM AND A. SCHULZ (1994). *Facets of the generalized permutahedron of a poset.* Preprint No. 386/1994, Technische Universität Berlin, Berlin.

[47] H. WEYL (1935). Elementare theorie der konvexen polyeder. *Helvetici 7*, 290-306. English translation: The elementary theory of convex polyhedra. *Contributions to the Theory of Games I* (H.W. Kuhn and A.W. Tucker, eds), Princeton University Press, Princeton, 1950.

[48] L.A. WOLSEY (1985). *Mixed integer programming formulations for production planning and scheduling problems.* Invited talk at the 12-th International Symposium on Mathamtical Programming, MIT, Cambridge.

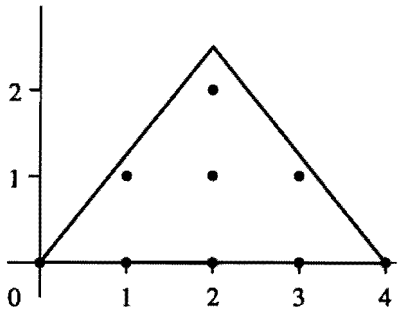
[49] L.A. WOLSEY (1990). Formulating single machine scheduling problems with precedence constraints. *Economic Decision Making: Games, Econometrics and Optimisation, Contributions in Honour of Jacques Dreze*, edited by J.J. Gabsewicz, J-F. Richard, and L.A. Wolsey, North-Holland, Amsterdam.

Samenvatting

Stelt u zich de volgende situatie voor. Een automonteur moet verschillende auto's repareren. Van elke auto weten we wanneer hij bij de garage wordt gebracht, wanneer hij gerepareerd moet zijn en hoeveel tijd de monteur nodig heeft om hem te repareren. De monteur kan maar één auto tegelijkertijd repareren. Het ziet er naar uit dat hij niet alle auto's op tijd gerepareerd kan hebben. We zoeken nu een werkschema voor de monteur waarbij het aantal auto's dat te laat klaar is zo klein mogelijk is.

Dit is een voorbeeld van een *machinevolgordeprobleem* met één machine, namelijk de monteur. Machinevolgordeproblemen betreffen het plannen van *taken* op *machines* met beperkte beschikbaarheid en capaciteit. Een *schema* geeft aan op welk tijdstip welke taak door welke machine wordt uitgevoerd. We zoeken een schema waarvoor de produktiekosten, die meestal berekend worden uit de voltooiingstijden van de taken, zo laag mogelijk zijn. De verscheidenheid aan eigenschappen van taken, machines en produktiekosten leidt tot een groot aantal verschillende machinevolgordeproblemen. Voor veel van deze problemen zijn geen snelle oplossingsmethoden bekend. Dit geldt zelfs voor problemen met maar één machine zoals het probleem in het bovenstaande voorbeeld.

In dit proefschrift bestuderen we oplossingsmethoden voor *één-machineproblemen*, d.w.z. machinevolgordeproblemen met één machine. Een één-machineprobleem kunnen we formuleren als een *geheeltallig lineair programmeringsprobleem*, d.w.z. een probleem waarbij de minimale waarde wordt gezocht van een gegeven lineaire doelstellingsfunctie als de oplossing moet voldoen aan een aantal lineaire beperkingen en de waarde van elke variabele bovendien geheeltallig moet zijn. Een eenvoudig voorbeeld van een dergelijk probleem is het volgende. Vind de oplossing (x, y) waarvoor $x - 10y$ minimaal is binnen de verzameling van punten (x, y) waarvoor geldt dat $x \geq 0$, $5x - 4y \geq 0$ en $5x + 4y \leq 20$ (dit zijn de lineaire beperkingen) en dat x en y geheeltallig zijn (dit zijn de geheeltalligheidseisen). De verzameling toegelaten oplossingen van dit probleem, d.w.z. de verzameling van alle (x, y) die aan de gestelde voorwaarden voldoen, is weergegeven in figuur 1. De verzameling van punten (x, y) die aan de lineaire beperkingen voldoen correspondeert met de dik getekende driehoek. Vanwege de geheeltalligheidseisen worden de toegelaten oplossingen nu gegeven door de stippen binnen en op de rand van de driehoek.

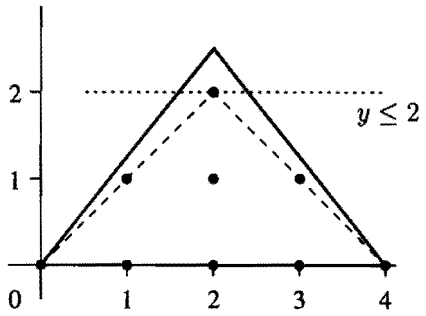


Figuur 1: De verzameling toegelaten oplossingen.

In hoofdstuk 2 bespreken we verschillende manieren om een één-machineprobleem te formuleren als een geheeltallig lineair programmeringsprobleem. De oplossingsmethoden in dit proefschrift zijn gebaseerd op een *tijdsgeïndexeerde formulering*. We gaan ervan uit dat het tijdvak dat we bekijken is verdeeld in T perioden en dat het uitvoeren van een taak altijd gestart wordt aan het begin van een periode. Voor elke taak j en periode t geeft de variabele x_{jt} aan of taak j gestart wordt aan het begin van periode t; x_{jt} is gelijk aan 1 indien dit het geval is en anders gelijk aan 0.

In hoofdstuk 3 bestuderen we de verzameling toegelaten oplossingen van het geheeltallig lineair programmeringsprobleem op basis van de tijdsgeïndexeerde formulering. Een geheeltallig lineair programmeringsprobleem is in het algemeen moeilijk oplosbaar. Als echter de geheeltalligheidseisen worden weggelaten, dan krijgen we een *lineair programmeringsprobleem*. Dit probleem wordt de *LP-relaxatie* genoemd. Een lineair programmeringsprobleem kan wel snel worden opgelost.

Door over te gaan op de LP-relaxatie wordt de verzameling toegelaten oplossingen uitgebreid. In het voorbeeld in figuur 1 is de oplossingsverzameling van de LP-relaxatie de volledige dik getekende driehoek. De oplossing van de LP-relaxatie is meestal dan ook niet geheeltallig. In geval van een niet-geheeltallige oplossing weten we dat alle geheeltallige oplossingen een waarde hebben die tenminste gelijk is aan de waarde van de gevonden oplossing; de waarde van de gevonden oplossing vormt een *ondergrens* op de waarde van de oplossing van het geheeltallig lineair programmeringsprobleem. Om deze ondergrens te verhogen en om dichter bij een geheeltallige oplossing te komen, voegen we aan de LP-relaxatie lineaire beperkingen toe die fractionele oplossingen van de LP-relaxatie uitsluiten, maar waar alle geheeltallige oplossingen wel aan voldoen. Deze beperkingen worden daarom *toegelaten ongelijkheden* genoemd. In het probleem uit figuur 1 is $y \leq 2$ een toegelaten ongelijkheid. In figuur 2 is te zien dat deze ongelijkheid een stuk van de dik getekende driehoek afsnijdt.



Figuur 2: Een toegelaten ongelijkheid en het convex omhulsel van de verzameling geheeltallige oplossingen.

We zijn in het bijzonder geïnteresseerd in toegelaten ongelijkheden die noodzakelijk zijn in de beschrijving van de *convex omhullende* van de verzameling van de geheeltallige oplossingen. Deze toegelaten ongelijkheden worden *facetten* genoemd. In figuur 2 is de convex omhullende van de verzameling van geheeltallige oplossingen van het probleem uit figuur 1 aangegeven met een stippellijn. Deze stippellijn valt op de x-as samen met de rand van de dik getekende driehoek. De facetten corresponderen met de zijkanten van de gestippelde driehoek. Als alle facetten aan de LP-relaxatie zijn toegevoegd, dan wordt gegarandeerd een geheeltallige oplossing gevonden. Voor een probleem dat niet snel oplosbaar is geldt echter dat het vinden van alle facetten erg moeilijk is.

In hoofdstuk 3 bestuderen we facetten voor één-machineproblemen. We beschouwen facetten met rechterkant 1 en 2, d.w.z. facetten van de vorm $\sum a_{jt}x_{jt} \leq 1$ en $\sum a_{jt}x_{jt} \leq 2$. Eerst bewijzen we een aantal structurele eigenschappen van deze facetten. Uit deze eigenschappen volgt dat alle facetten met rechterkant 1 bevat zijn in één klasse ongelijkheden. Omdat bekend is dat alle ongelijkheden in deze klasse ook werkelijk facetten definiëren, zijn hiermee alle facetten met rechterkant 1 gekarakteriseerd. Uit de eigenschappen van facetten met rechterkant 2 volgen drie klassen ongelijkheden die samen al deze facetten bevatten. De meeste ongelijkheden in deze klassen waren tot nu toe onbekend. We laten zien welke van de ongelijkheden in deze klassen facetten definiëren. Hiermee hebben we alle facetten met rechterkant 2 gekarakteriseerd.

In hoofdstuk 4 gaan we na hoe de in hoofdstuk 3 gekarakteriseerde ongelijkheden gebruikt kunnen worden om één-machineproblemen op te lossen. Het aantal facetten is zo groot dat het ondoenlijk is om ze allemaal aan de LP-relaxatie toe te voegen. Daarom gaan we als volgt te werk. We lossen eerst de LP-relaxatie op. Als de oplossing niet geheeltallig is, dan zoeken we binnen de verzameling gekarakteriseerde ongelijkheden naar ongelijkheden die geschonden worden door

deze oplossing, d.w.z. ongelijkheden waar deze oplossing niet aan voldoet. We voegen de geschonden ongelijkheden toe aan de LP-relaxatie. De oplossing die we hadden is niet meer toegelaten in het nieuwe probleem. We lossen het nieuwe probleem op en, als de oplossing niet geheeltallig is, voegen we opnieuw ongelijkheden toe. Dit proces wordt herhaald totdat we een geheeltallige oplossing bereiken of geen geschonden ongelijkheden meer kunnen vinden. Het vinden van ongelijkheden die door een niet-geheeltallige oplossing worden geschonden heet *separatie*. In het eerste deel van hoofdstuk 4 leiden we separatiemethoden af voor de in hoofdstuk 3 gekarakteriseerde ongelijkheden. Door het toevoegen van ongelijkheden wordt weliswaar niet altijd een geheeltallige oplossing gevonden, maar de ondergrens verkregen uit de LP-relaxatie wordt sterk verbeterd.

Als we geen geschonden ongelijkheden meer kunnen vinden en we hebben nog geen geheeltallige oplossing bereikt, dan passen we *branch-and-bound* toe. Bij branch-and-bound wordt het probleem gesplitst in deelproblemen door de verzameling toegelaten oplossingen te splitsen (dit heet branching). Voor elk deelprobleem berekenen we een ondergrens op de waarde van de oplossing van dit deelprobleem (dit heet bounding). Als er een toegelaten geheeltallige oplossing bekend is met een waarde kleiner dan of gelijk aan de ondergrens, dan hoeft het deelprobleem niet meer bekeken te worden, omdat dit deelprobeem geen oplossing kan hebben die beter is dan de beste bekende oplossing. In het branch-and-bound algoritme gebruiken we de ondergrens die wordt verkregen door het toevoegen van ongelijkheden aan de LP-relaxatie. Een branch-and-bound algoritme waarin de ondergrenzen op deze manier bepaald worden heet een *branch-and-cut* algoritme. Voor de kwaliteit van het branch-and-cut algoritme is het van belang op welke manier het probleem gesplitst wordt, in welke volgorde de deelproblemen worden bekeken, welke van de geschonden ongelijkheden worden toegevoegd en hoe goede toegelaten geheeltallige oplossingen worden bepaald. In het tweede deel van hoofdstuk 4 bespreken we verschillende mogelijkheden, hetgeen resulteert in verschillende varianten van het algoritme. We hebben deze verschillende varianten getest en presenteren de verkregen rekenresultaten.

Uit de rekenresultaten van hoofdstuk 4 blijkt dat de tijdsgeïndexeerde formulering goede ondergrenzen levert. Omdat het aantal beperkingen en variabelen in deze formulering erg groot is, moeten we grote lineaire programmeringsproblemen oplossen om de ondergrenzen te berekenen, hetgeen veel tijd kost. Daarom bestuderen we in hoofdstuk 5 een methode die speciaal ontworpen is om grote lineaire programmeringsproblemen op te lossen. Deze methode werkt als volgt. We passen eerst *Dantzig-Wolfe decompositie* toe. Hierdoor krijgen we een nieuwe formulering met minder beperkingen en meer variabelen. Het grote aantal variabelen is geen bezwaar, want we kunnen het verkregen probleem oplossen met *kolomgeneratie*. Bij kolomgeneratie beschouwen we aanvankelijk een beperkt probleem in de zin dat we slechts een gedeelte van de variabelen opnemen. Nadat we het beperkte probleem hebben opgelost, gaan we na of het nodig is om extra variabelen toe te voegen.

Dit wordt gedaan door het oplossen van het zogenaamde *pricing-probleem*. Indien nodig voegen we variabelen toe en herhalen de procedure. We laten zien dat het oplossen van het pricing-probleem voor onze formulering neerkomt op het bepalen van het kortste pad in een netwerk. Dit impliceert dat het pricing-probleem snel opgelost kan worden. Onze rekenresulaten laten zien dat we met kolomgeneratie de LP-relaxaties van grote problemen sneller op kunnen lossen. We bespreken ook het combineren van kolomgeneratie en het toevoegen van ongelijkheden en het combineren van kolomgeneratie en branch-and-bound. We geven rekenresultaten gebaseerd op een voorlopige implementatie van het combineren van kolomgeneratie en het toevoegen van ongelijkheden. Verder besteden we aandacht aan het herformuleren door middel van zogenaamde *key formulations*, waardoor het geheugengebruik van het kolomgeneratie-algoritme misschien gereduceerd kan worden.

Curriculum vitae

Marjan van den Akker was born on December 25, 1965 in Gouda, the Netherlands. She obtained her VWO-diploma at the Rythovius-college in Eersel in 1984.

In the same year she started to study Mathematics at Eindhoven University of Technology. During her studies she specialized in discrete mathematics, and from February 1989 to May 1989 she visited the University of Århus in Denmark. She graduated cum laude in June 1990 with a master's thesis on 'Distance-biregular graphs', which was written under the guidance of prof.dr. A.E. Brouwer.

After her graduation, she started as a PhD-student at Eindhoven University of Technology in the group of prof.dr. J.K. Lenstra. She attended the courses of the Landelijk Netwerk Mathematische Besliskunde, of which she obtained a diploma in 1993. The results of the research that she performed are described in this thesis. She is currently employed as a post-doc at the Center for Operations Research and Econometrics (CORE) in Louvain-la-Neuve.

Stellingen

behorende bij het proefschrift

LP-based solution methods for single-machine scheduling problems

door

JANNA MAGRIETJE VAN DEN AKKER

I

Stel dat een code $C \subseteq Q^n$ bestaat uit twee disjuncte deelcodes C_1 en C_2 waarvoor er gehele getallen r_1 en r_2 bestaan zodanig dat de bollen $B_r(\mathbf{c})$ met $r = r_i$ als $\mathbf{c} \in C_i$ $(i = 1, 2)$ disjunct zijn. Dan heet C met deelcodes C_1 en C_2 een (r_1, r_2)-*foutenverbeterende code*. Zo'n code heet *perfect* als de vereniging van alle bollen $B_r(\mathbf{c})$ met $r = r_i$ als $\mathbf{c} \in C_i$ $(i = 1, 2)$ gelijk is aan Q^n. De code heet *bipartiet* als $\min\{d(c, c') | c, c' \in C_i\} > 2r_i + 1$ $(i = 1, 2)$.

Cohen en Montaron [1979] hebben de volgende familie $(2, 1)$-foutenverbeterende codes afgeleid. Laat $\mathcal{P}$ een code met de parameters van een Preparata-code met lengte $n = 2^{2t} - 1$ zijn en $\mathcal{H}$ de vereniging van $\mathcal{P}$ en de woorden op afstand 3 van $\mathcal{P}$. Laten verder $\bar{\mathcal{P}}$ en $\bar{\mathcal{H}}$ de extensies van $\mathcal{P}$ en $\mathcal{H}$ zijn. Dan is $\bar{\mathcal{H}}$ met $C_1 = \bar{\mathcal{P}}$ en $C_2 = \bar{\mathcal{H}} \setminus \bar{\mathcal{P}}$ een $(2, 1)$-foutenverbeterende code.

Bipartiete perfecte $(r, 1)$-foutenverbeterende codes zijn nu als volgt gekarakteriseerd. Als C een bipartiete perfecte $(r, 1)$-foutenverbeterende code is, dan geldt dat $r = 2$ en dat C tot de bovenstaande familie behoort [Van den Akker, Koolen en Vaessens, 1990].

J.M. VAN DEN AKKER, J.H. KOOLEN, AND R.J.M. VAESSENS (1990). Perfect codes with distinct protective radii. *Discrete Mathematics 81*, 103-109, *Discrete Mathematics 89*, 325.

G. COHEN AND B. MONTARON (1979). Empilements parfaits de boules dans les espaces vectoriels binaires. *C.R. Acad. Sci. Paris 288.*

II

Een bipartiete graaf heet *afstands-biregulier* als alle punten in dezelfde puntklasse hetzelfde aantal buren hebben en als voor ieder tweetal punten x en y die gelegen zijn op afstand i van elkaar geldt dat het aantal buren van y gelegen op afstand $i-1$ van x en het aantal buren van y gelegen op afstand $i+1$ van x alleen afhankelijk zijn van i en de klasse van punten waartoe x behoort.

Er bestaat een afstands-bireguliere graaf met intersectie-array

$$\left\{ \begin{array}{ccccc} 28; & 1, & 4, & 6, & 28 \\ 10; & 1, & 2, & 12, & 10 \end{array} \right\}.$$

J.M. VAN DEN AKKER (1990). *Distance-biregular graphs.* Master's thesis, Department of Mathematics and Computing Science, Eindhoven University of Technology.

III

Alle facet-definiërende ongelijkheden met rechterkant 1 of 2 voor de tijdsgeïndexeerde formulering voor één-machineproblemen die bestudeerd wordt in dit proefschrift zijn gekarakteriseerd (zie hoofdstuk 3 van dit proefschrift).

IV

Ofschoon in de rekenexperimenten in hoofdstuk 4 van dit proefschrift de vertakkingsstrategie *SOS-mst*, die vertakt op basis van *special ordered sets*, voor de meeste instanties minder knopen en een lagere rekentijd oplevert dan de vertakkingsstrategie *pos*, die opdrachten op bepaalde posities in het schema vastlegt, mag verwacht worden dat voor probleeminstanties met grotere verwerkingstijden de vertakkingsstrategie *pos* het beste werkt.

V

Stel dat de LP-relaxatie van een geheeltallig lineair programmeringsprobleem wordt opgelost door het probleem eerst te herformuleren door middel van Dantzig-Wolfe-decompositie en het onstane probleem vervolgens op te lossen met kolomgeneratie. Stel bovendien dat bij het oplossen van het pricing-probleem geen gebruik wordt gemaakt van de structuur van de kostencoëfficiënten van het originele probleem. Dan kunnen aan de herformulering toegelaten ongelijkheden worden toegevoegd die afkomstig zijn uit de originele formulering zonder dat de oplossingsmethode van het pricing-probleem veranderd hoeft te worden (zie hoofdstuk 5 van dit proefschrift).

VI

Met de tijdsgeïndexeerde formulering, die bestudeerd wordt in dit proefschrift, kan men problemen met m parallelle machines modelleren door de rechterkant van de beperkingen $\sum_{j=1}^{n} \sum_{t-p_j<s\leq t} x_{js} \leq 1$, die de capaciteitsrestrictie van de machine weergeven, te vervangen door m. We nemen aan dat het aantal opdrachten groter is dan m. Om de convex omhullende van de verzameling toegelaten oplossingen volledig-dimensionaal te maken, vervangen we de voorwaarde dat elke opdracht precies één keer wordt gestart door de voorwaarde dat elke opdracht maximaal één keer wordt gestart. Nu geldt dat de ongelijkheden

$$\sum_{l-p_{j_1}<s\leq u} x_{j_1 s} + \sum_{j\neq j_1} \sum_{u-p_j<s\leq l} x_{js} \leq m,$$

waarbij j_1 een gegeven opdracht is en $l < u$, toegelaten zijn. De ongelijkheden waarvoor geldt dat $|\{j|j \neq j_1, p_j > u - l\}| \geq m$ definiëren facetten.

VII

Positieve actie, waarbij elke vrouwelijke sollicitant die aan de functie-eisen voldoet voorrang heeft boven elke mannelijke sollicitant, zal op den duur in het nadeel van vrouwen werken.

VIII

Mensen die klagen over de Nederlandse Spoorwegen hebben waarschijnlijk nog nooit met de Brabantse Buurtspoorwegen en Autodiensten (BBA) gereisd.

IX

Om het gebruik van de fiets binnen de stad te bevorderen, dient de overheid het door rood fietsen niet te zien als het overtreden van de wet maar als het vrijwillig nemen van een risico. Om dit risico te beperken, dient op een verkeerslicht voor auto's naast het rode licht ook het oranje licht aan te gaan voordat het verkeerslicht op groen springt; bovendien dient de kleur van het licht aan alle zijden zichtbaar te zijn.

X

In het dagelijks leven is een fractionele oplossing in de vorm van een compromis vaak te prefereren boven een geheeltallige oplossing die wordt aangenomen in een extreem punt.

XI

Het onbekend zijn van het beroep wiskundige bij het arbeidsbureau duidt weliswaar op waardering voor dit beroep, maar niet op een realistische inschatting van de huidige arbeidsmarkt.