Université catholique de Louvain

Faculté des Sciences Appliquées

Integer Programming-based Decomposition Approaches for Solving Machine Scheduling Problems

Thèse présentée en vue de l'obtention du grade
de Docteur en Sciences Appliquées par

Ruslan SADYKOV

Promoteurs:	Y. Nesterov
	L.A. Wolsey
Jury:	V. Wertz (Président)
	Ph. Baptiste
	Y. Crama
	Y. Pochet

Acknowledgements

I would like to thank many people without whom I would not have reached this stage in my life.

If there exist an ideal supervisor, Laurence Wolsey is for sure one. His expertise in the field and particularly his willingness to share his knowledge with others helped me a lot during last four years. His goodwill and support were indispensable for my work.

I owe a lot to Yurii Nesterov. Thanks to him I was able to come to CORE. His readiness to help in every aspect of the life facilitated significantly my stay in Louvain-la-Neuve.

I also thank Alexander Lazarev, my first supervisor in Russia. He introduced me to the European Operations Research community and inspired me to pursue scientific research.

I am grateful to Philippe Baptiste for our scientific collaboration. One of his ideas has grown into one of the chapters of this thesis. His continuing support allows me to further my research next year in France.

I thank the members of the jury and especially Yves Pochet for their valuable comments and suggestions on an earlier version of the thesis.

I want to say many kind words about CORE. People here create a very friendly and productive atmosphere. When you are at CORE, you feel like being at home. I have enjoyed the very good company of Francois, Peter, Michel, Robert, Quentin, Mathieu, Alexander, Hamish, Kent, Barat, Jean-Francois, Elina, Alexey, Diego, Andreas and many others. Thank you all. I would like to mention the professional administrative assistance of Mady. I appreciate her help very much.

Finally, I would like to express my gratitude to my parents and Leysan. What you have done for me is invaluable. Your support was essential for me. I love you all.

Contents

Chapter 1

Introduction

This thesis deals with scheduling problems. A standard definition of scheduling is the allocation of scarce resources to tasks. Usually one does not need just any allocation but the best one, according to some specified criterion. Today, in almost any scope of people's activity, one can find processes related to scheduling. We now give several examples of such processes.

- In a school, one needs to assign teachers as well as allocate rooms to lessons. The teacher assigned to a lesson should be qualified to conduct it. Also the room allocated to a lesson should be large enough to admit the group of pupils taking this lesson.
- In an airline company, the crews must be allocated to the flights carried out by the company. Of course, each flight needs suitable pilots, stewards, etc.
- The operating system in a computer is required to allocate the active processes to the processor on a timesharing basis. Also each process should receive enough memory.
- In a factory, a set of parts should be cut on various machines. Each machine cuts different parts with different speeds and can process only one part at a time. One needs to assign parts to machines and determine the order in which parts are cut on each machine. Additionally, the work should be finished as soon as possible.

The last example falls into the class of *machine scheduling* problems. In these problems one works with a specific kind of resources — *machines*. Each machine is available from time zero onwards and has capacity 1, i.e. can perform only one task at a time. Tasks are called *jobs* in machine scheduling. In principle, each job can be divided into several operations. However, in this thesis, we consider only cases in which every job consists of one operation. We also limit our attention to the case of non-preemptive jobs. Once the

execution of a non-preemptive job is started on a certain machine, it cannot be interrupted until the whole job has been processed.

In this thesis, we consider two one-machine and two multi-machine scheduling problems. We propose different approaches to solve these problems to optimality. The main "ingredient" we use is Integer Programming (IP). As IP itself is not very efficient in solving machine scheduling problems, we combine it with other methods through decomposition of the problems. These methods include Constraint Programming techniques and specialized scheduling algorithms.

In the following sections, we briefly introduce the machine scheduling theory, Integer Programming with the accent on decomposition methods we use, Constraint Programming techniques for machine scheduling. We conclude this chapter with the outline of the thesis.

1.1 Machine Scheduling

In machine scheduling, each job j from the given set $N = \{1, \ldots, n\}$ is characterized by certain parameters. The main parameter is a positive processing time p_j, i.e. the length of the job. Additionally, job j can have the following parameters.

- Release date r_j, the time when the job becomes available for processing.
- Due date d_j, the time by which ideally the job should be completed.
- Deadline $\bar{d}_j$, the time by which the job must be completed.
- Weight w_j, the value of the job.

The processing time and the weight of a job can vary depending on the machine the job is processed on. For a set J of jobs we will denote

$$r_J = \min_{j \in J} r_j, \quad p_J = \sum_{j \in J} p_j, \quad d_J = \max_{j \in J} d_j, \quad \bar{d}_J = \max_{j \in J} \bar{d}_j.$$

In any machine scheduling problem, the aim is to find a *schedule* which is optimal according to some criterion. Before addressing criteria, let us present some notation. The starting time of job j is denoted by S_j, and its completion time is denoted by C_j. The *lateness* L_j of job j is the difference between its completion time and its due date: $L_j = C_j - d_j$. The *tardiness* T_j of job j is the positive part of its lateness: $T_j = \max\{0, C_j - d_j\}$. Job j is *late* if its completion time exceeds its due date, otherwise job j is *on-time*. The sign of the lateness of job j is denoted by U_j, $U_j = 1$ if j is late and $U_j = 0$ if j is on-time.

The classical criteria in machine scheduling are:

- $C_{\max}$ — minimizing the maximum completion time (makespan),

- $L_{\max}$ — minimizing the maximum lateness,
- $\sum(w_j)C_j$ — minimizing the (weighted) sum of completion times,
- $\sum(w_j)T_j$ — minimizing the total (weighted) tardiness,
- $\sum(w_j)U_j$ — minimizing the (weighted) sum of late jobs.

Note that all the classical criteria are *regular*, i.e. they are nondecreasing with respect to the completion times of jobs. A regular optimality criteria allows us to consider only *early* schedules. In an early schedule, the processing of every job on any machine starts as early as possible: either at the completion time of the previous job assigned to this machine or at the release date of the job. Obviously, for any non-early schedule there exists an early schedule which is not worse according to a regular criteria. In the thesis, all schedules are assumed to be early.

A nice property of an early schedule is that it can be uniquely represented by an assignment of jobs to machines and permutations of the jobs assigned to the same machine. For one-machine problems, we will not distinguish between a permutation of jobs in N and the corresponding early schedule. As $\Pi(N)$ we denote the set of all the schedules containing all the jobs in set N.

Due to Graham et al. [38], there exists an established notation for machine scheduling problems. In this notation, problems are specified in terms of the three-field classification $\alpha \mid \beta \mid \gamma$.

- α specifies the number and the type of machines available. $\alpha = 1$ means that one machine is present. $\alpha = P$ signifies that several *identical* machines are available, i.e. the processing time of a job does not depend on the machine the job is processed on. $\alpha = R$ means that we have several *unrelated* machines, i.e. the processing time of a job is different depending on which machine the job is processed.
- β describes additional constraints of the problem. For example, r_j reveals the presence of different release dates for jobs; $p_j = p$ tells that the processing times of all the jobs are equal; *prec* signifies the presence of precedence relation between some jobs.
- γ represents the optimality criterion of the problem.

For example, the notation $1 \mid r_j \mid \sum w_j T_j$ specifies the problem in which only one machine is available, jobs have different release dates, the criterion is minimizing the weighted total tardiness. Note that the objective function $\sum w_j T_j$ assumes the presence of weights and due dates for the jobs.

Note that fields α and β can have values different from those given above, and also γ is not limited to classical criteria. We have not presented other values and criteria as the corresponding problems are not covered in the thesis.

One of the main research directions in machine scheduling is determining the complexity of problems. Complexity results for scheduling problems are

collected by Brucker and Knust [18]. As it can be seen from these results, a majority of machine scheduling problems are NP-hard. However, one needs to tackle such problems in practice. There exist three main approaches to solve NP-hard scheduling problems.

- *Approximation algorithms.* These are polynomial algorithms producing approximate solutions whose quality can be estimated a priori. One estimates the deviation of the approximate solution from the optimal one either on average or in the worst case. This deviation is called the *error*. An error can be relative (the ratio between the values of approximate and optimal solutions) of absolute (the difference between these values). An algorithm is *ρ-approximation* if it always produces a solution whose relative error does not exceed ρ. For some NP-hard problems, there exists a PTAS (*polynomial time approximation scheme*). For any fixed $\epsilon > 0$, a PTAS allows one to build an $1 + \epsilon$-approximation polynomial algorithm. For an NP-hard problem for which a PTAS does not exist, an important question is establishing the minimum value ρ such that there exists a ρ-approximation polynomial algorithm for this problem.

- *Heuristic algorithms.* These algorithms produce heuristic solutions and do not give any estimates of their quality. The main advantage of these algorithms is that they allow one to obtain a good solution quite fast. Although, an evident drawback is impossibility to determine how good is this solution. The most extended class of such algorithms are so-called "local search" or metaheuristic algorithms.

- *Enumeration algorithms.* These algorithms are not polynomial. They are aimed to produce an optimal solution of the problems. However, one cannot guarantee that such a solution will be found in a certain time. If an optimal solution is not found, an enumeration algorithm can usually give some solution and estimate its quality. The main drawback of such algorithms is that it is impossible to estimate a priory the quality of the solution obtained within a certain running time. The most extended class of enumeration algorithms are branch-and-bound algorithms.

  For some NP-hard machine scheduling problems, there exist very efficient enumeration algorithms, for example the algorithm by Carlier [20] for the problem $1 \mid r_j \mid L_{\max}$ and the algorithm by Szwarc et al. [81] for the problem $1 \mid\mid \sum T_j$.

In the thesis, we will concentrate on devising enumeration algorithms for certain machine scheduling problems.

For details about machine scheduling and scheduling in general, the reader can refer to books by Brucker [17] and Leung [56].

1.2 Integer Programming

Integer Programming is a method to formulate and solve many combinatorial optimization problems including machine scheduling problems. *Mixed Integer Program (MIP)* stands for a problem of minimizing or maximizing a linear objective function of many variables subject to linear constraints and integrality restrictions on a part of variables. A MIP can be formally written as:

$$\min \quad cx + hy, \tag{1.1}$$

$$Ax + Gy \leq b, \tag{1.2}$$

$$x \geq 0, \quad y \geq 0, \quad x \text{ is integer}, \tag{1.3}$$

where c is an n-dimensional vector, h is an q-dimensional vector, b is an m-dimensional vector, A is an m by n matrix, G is an m by q matrix, x is an n-dimensional vector of variables, and y is an q-dimensional vector of variables.

Example 1.1. We formulate the scheduling problem $R \mid d_j = d \mid \sum w_j U_j$ as a MIP. Given the set of jobs $N = \{1, \ldots, n\}$ and the set of machines $M = \{1, \ldots, m\}$, we introduce binary variables x_j^i, $i \in M, j \in N$. Variable x_j^i takes value 1 if job j is assigned to machine i, otherwise $x_j^i = 0$. Let p_j^i be the processing time of job j on machine i. Taking into account the fact that late jobs can be executed arbitrarily late without increasing the cost, we do not assign late jobs to any machine. Using this observation, we can write the following formulation.

$$\min \quad \sum_{j \in N} w_j \cdot \left(1 - \sum_{i \in M} x_j^i\right) \tag{1.4}$$

$$\sum_{i \in M} x_j^i \leq 1, \quad j \in N, \tag{1.5}$$

$$\sum_{j \in N} p_j^i x_j^i \leq d, \quad i \in M, \tag{1.6}$$

$$x_j^i \in \{0, 1\}, \quad i \in M, j \in N. \tag{1.7}$$

The objective function (1.4) represents the criterion $\sum w_j U_j$. The constraints (1.5) ensure that each job is assigned to at most one machine. The constraints (1.6) guarantee that every assigned job is on-time, i.e., for each machine $i \in M$, the sum of processing times of jobs assigned to i does not exceed the common due date d.

The usual way to solve MIP problems in practice is by *branch-and-bound*. As it can be seen from the name, this method relies on *branching* and *bounding*. Branching represents the idea of dividing a problem into a series of easier/smaller problems. Once all the easier problems are solved, we can use the information obtained to find the solution of the original problem.

A typical way to branch when solving a MIP problem is to reduce the set of possible values for variables which we want to be integer. For example, one can divide a MIP problem with an integer variable x into two subproblems by adding constraint $x \leq k$ into the first subproblem and constraint $x \geq k + 1$ into the second one, where k is an integer. Once we have divided the problem into easier subproblems, we solve each subproblem and choose the best solution. When the set of possible values of each integer variable is finite, we can perform complete enumeration. For this, the problem is divided into all possible subproblems in which each integer variable is fixed to some value. Such subproblems have only continuous variables and can be solved by Linear Programming methods. Unfortunately, complete enumeration is not suitable for any problem of practical size. The huge number of subproblems makes this approach impossible.

To avoid complete enumeration, the number of subproblems to be solved needs to be reduced. For this, we evaluate whether a group of subproblems is worth considering or not. One way to do it is by bounding, i.e. by computing a lower bound (for a minimization problem) on the value of the best solution of a group of subproblems. If the lower bound is greater than or equal to the value of the best solution known, then the group of subproblems can be excluded from consideration, as no subproblem from the group can have a better solution value. When solving a MIP by branch-and-bound, the lower bound is computed by solving the *Linear Programming (LP) relaxation* of the subproblem. The LP relaxation is the problem obtained by removing the integrality constraints on the variables.

Obviously, the bigger the lower bound is, the better. In order to improve the bound, one can add constraints called *cutting planes* to the formulation. Cutting planes (cuts) are inequalities that are redundant for the MIP but not redundant for its LP relaxation. Therefore adding cuts does not change the solution of the MIP but can improve the lower bounds. The branch-and-bound method which uses cuts is also called *branch-and-cut*.

We now give a description of the branch-and-cut method (for a minimization problem). The method maintains an enumeration tree in which all the subproblems are stored. Each subproblem is associated with a node in the tree. Nodes with subproblems which haven't been considered are called *active nodes*. The tree is initialized with the original MIP. Then an iterative procedure is applied. Each iteration has the following steps.

1. *Node selection.* An active node is chosen and removed from the list of active nodes. This choice has an impact on the performance of the method, so it should be made wisely. Unfortunately, there is no universally best rule for node selection.

2. *Bounding.* The LP relaxation of the current subproblem (associated with the current node) is solved. If the relaxation is infeasible then the iteration is terminated (this situation is called *pruning by infeasibility*). Otherwise, if the value of the solution of the relaxation is greater than

or equal to the value of the best known solution, then the iteration is terminated (*pruning by bound*). Otherwise, if in the LP solution all the integer variables take integer values, then the solution is feasible for the original MIP and better than the best known solution. In this case the best known solution is updated and the iteration is terminated (*pruning by optimality*).

3. *Cut generation.* Here we search for constraints that are violated by the solution of the relaxation but satisfied by any integer solution of the current subproblem. If such constraints are found, they are added to the subproblem formulation and one returns to step 2.

4. *Variable selection.* At this step, some variables which need to be integer are fractional in the solution of the relaxation. One such variable, say x, is chosen. This is another choice which has an influence on the performance of the method.

5. *Branching.* Suppose that the selected variable x has value $\bar{x}$ in the solution of the relaxation. It is known that $\bar{x}$ is fractional. Two descendant nodes of the current node are added to the tree and to the list of active nodes. With the first node the current subproblem with additional constraint $x \leq \lfloor \bar{x} \rfloor$ is associated. With the second node the current subproblem with additional constraint $x \geq \lceil \bar{x} \rceil$ is associated. Here $\lfloor \bar{x} \rfloor$ and $\lceil \bar{x} \rceil$ define, respectively, rounding down and rounding up of x to the nearest integer.

The reader can find more information about formulating and solving Mixed Integer Problems in the books by Bertsimas and Weismantel [13], Schrijver [76], Wolsey [87]. There are several Mixed Integer Programming software systems now available for solving MIPs using branch-and-bound and branch-and-cut. These systems allow one to concentrate on modelling and algorithmic issues, rather than implementation. For details, see the work by Atamturk and Savelsbergh [5].

1.3 Dantzig-Wolfe Decomposition

Just like most combinatorial problems, different scheduling problems have a particular structure. This structure can be exploited in order to speed up the solution of problems. For example, in certain multi-machine scheduling problems, once jobs have been assigned to machines, each machine can be considered separately. This means that algorithms for the corresponding one-machine problem can be applied.

One way to exploit structure is by decomposition of the problem. In 1960, Dantzig and Wolfe [29] have proposed an approach to decompose Linear Programs. Later on, it has been extended to MIPs [10, 71, 84]. We now

present this approach. Then we illustrate Dantzig-Wolfe decomposition on a MIP formulation of a scheduling problem.

Consider the formulation (1.1)-(1.3) and divide the constraints (1.2) into two sets:

$$\begin{aligned} \min \quad & cx + hy, && (1.8)\\ & A^{(1)}x + G^{(1)}y \le b^{(1)}, && (1.9)\\ & A^{(2)}x + G^{(2)}y \le b^{(2)}, && (1.10)\\ & x \in \mathbb{Z}^n_+, \quad y \ge 0, && (1.11) \end{aligned}$$

where $A^{(1)} \in \mathbb{R}^{m^1 \times n}$, $G^{(1)} \in \mathbb{R}^{m^1 \times q}$, $b^{(1)} \in \mathbb{R}^{m^1}$, $A^{(2)} \in \mathbb{R}^{m^2 \times n}$, $G^{(2)} \in \mathbb{R}^{m^2 \times q}$, $b^{(2)} \in \mathbb{R}^{m^2}$. Suppose that set

$$X = \left\{ (x, y) \in \mathbb{Z}^n_+ \times \mathbb{R}^q_+ : \; A^{(2)}x + G^{(2)}y \le b^{(2)} \right\}$$

is bounded and non-empty. Let $\{(x^t, y^t)\}$, $t \in T$, be the set of extreme points of $conv(X)$. Then we can represent set X in the following way:

$$X = \left\{ (x, y) \in \mathbb{Z}^n_+ \times \mathbb{R}^q_+ : \; (x, y) = \sum_{t \in T} \lambda_t (x^t, y^t), \sum_{t \in T} \lambda_t = 1, \lambda_t \ge 0 \; \forall t \in T \right\}.$$

Now substituting for x and y in the problem (1.8)-(1.11) leads to an equivalent *Master Problem*:

$$\begin{aligned} \min \quad & \sum_{t \in T} (c^t x^t + h^t y^t) \lambda_t, \\ & \sum_{t \in T} (A^{(1)} x^t + G^{(1)} y^t) \lambda_t \le b^{(1)}, \\ & \sum_{t \in T} \lambda_t = 1, \quad \lambda_t \ge 0, \; t \in T, \\ & \sum_{t \in T} x^t \lambda_t \in \mathbb{Z}^q_+. \end{aligned}$$

A difficulty arises when we want to solve the master problem by branch-and-bound. It is impractical to solve the LP relaxation of the master problem directly due to the huge number of variables. Instead, a modification of the simplex algorithm is applied — the *column generation* procedure. This procedure works with a subset of variables (columns) and generates missing variables only when they are needed. On each iteration of the column generation algorithm, the LP relaxation of the master problem with a restricted number of columns is solved. Let $\bar{\lambda} \in \mathbb{R}^{|T|}_+$ and $(\nu, \mu) \in \mathbb{R}^{m^1} \times \mathbb{R}$ be an optimal primal solution and the corresponding optimal dual solution of this restricted linear program. Linear programming theory tells us that the relaxation of the master problem is solved if the reduced cost

$$(c^t - \nu A^{(1)})x^t + (h^t - \nu G^{(1)})y^t - \mu$$

for each variable λ_t, $t \in T$, is nonnegative. To check whether all the variables λ_t have nonnegative reduced costs, the *pricing problem*

$$\begin{aligned} v(\nu,\mu) = \quad \min \quad & (c - \nu A^{(1)})x + (h - \nu G^{(1)})y - \mu \\ & A^{(2)}x + G^{(2)}y \le b^{(2)}, \\ & x \in \mathbb{Z}^n_+, \quad y \ge 0. \end{aligned}$$

is solved. Let $(x^{t'}, y^{t'})$ be an optimal solution of the pricing problem. If $v(\nu,\mu) \ge 0$, then $\bar{\lambda}$ is an optimal solution for the LP relaxation of the master problem. Otherwise, the reduced cost of variable $\lambda_{t'}$ is negative. In this case, $\lambda_{t'}$ is added to the restricted LP relaxation, and the next iteration begins.

If the solution of the LP relaxation of the master problem is not integer, branching is performed as in branch-and-bound. Combining column generation and branching results in the *branch-and-price* method.

There are two cases, in which Dantzig-Wolfe decomposition of a MIP can be advantageous. First, when the pricing problem breaks up into a set of independent problems which can be solved separately. Secondly, when the pricing problem can be solved efficiently by a specialized method.

Example 1.1. (continued) We now reformulate the problem $R \mid d_j = d \mid \sum w_j U_j$ using Dantzig-Wolfe decomposition of the formulation (1.4)-(1.7). We consider the constraints (1.5) as the constraints (1.9), and the constraints (1.6) — as the constraints (1.10). Then

$$X = \left\{ x \in \{0,1\}^{m \times n} : \sum_{j \in N} p^i_j x^i_j \le d, i \in M \right\}.$$

Here we can exploit the structure of the problem and decompose set X into sets X^i, $i \in M$, where $X^i = \{x^i \in \{0,1\}^n : \sum_{j \in N} p^i_j x^i_j \le d\}$. For each element $x^{it} \in X^i$, $t \in T^i$, we introduce a variable λ^i_t. In our case, $\lambda^i_t = 1$ if the subset $\{j \in N : x^{it}_j = 1\}$ of jobs is assigned to machine i. Using variables λ we reformulate the problem as follows.

$$\begin{aligned} \min \quad & \sum_{j \in N} w_j \cdot \left(1 - \sum_{i \in M} \sum_{t \in T^i} x^{it}_j \lambda^i_t\right) \\ & \sum_{i \in M} \sum_{t \in T^i} x^{it}_j \lambda^i_t \le 1, \quad j \in N, \quad (\nu_j) \\ & \sum_{t \in T^i} \lambda^i_t \le 1, \quad i \in M, \quad (\mu^i) \\ & \lambda^i_t \in \{0,1\}, \quad i \in M, t \in T^i. \end{aligned}$$

The LP relaxation of this formulation is solved by the column generation procedure. Let $(\nu,\mu) \in \mathbb{R}^n \times \mathbb{R}^m$ be an optimal dual solution of the LP

relaxation with restricted number of columns, on some iteration. To check if there are variables of the reformulation with a negative reduced cost, we need to solve the pricing problem. The reduced cost of variable λ_t^i, $t \in T^i$, $i \in M$, is equal to $\sum_{j \in N}(-w_j - \nu_j)x_j^{it} - \mu^i$. The pricing problem here decomposes into m subproblems, one for each machine $i \in M$. The subproblem for machine i checks if there are variables λ_t^i, associated with machine i, with a negative reduced cost, i.e. if $v^i(\nu, \mu^i) < 0$, where

$$v^i(\nu, \mu^i) = \min \left\{ \sum_{j \in N}(-w_j - \nu_j)x_j^i - \mu^i : \sum_{i \in N} p_j^i x_j^i \leq d,\ x^i \in \{0,1\}^n \right\}.$$

Each subproblem is a knapsack problem which is NP-hard in the ordinary sense. However, there exist algorithms which are very fast in practice. [45].

1.4 Generic Benders Decomposition

In this section we present another decomposition approach for solving MIPs. Consider an optimization problem formulated as

$$\begin{aligned} \min \quad & cx + hy + vz, && (1.12)\\ & A^{(1)}x + G^{(1)}y \leq b^{(1)}, && (1.13)\\ & A^{(2)}x + G^{(2)}y + Vz \leq b^{(2)}, && (1.14)\\ & x \in \mathbb{Z}_+^n, \quad y \in \mathbb{R}_+^q, \quad z \in Z \subseteq \mathbb{R}^s, && (1.15) \end{aligned}$$

where $V \in \mathbb{R}^{m^2 \times s}$, z is an s-dimensional vector of variables. Let Y be the set of points $(x, y) \in \mathbb{Z}_+^n \times \mathbb{R}_+^q$ for which one can find $z \in Z$ such that (x, y, z) is a feasible solution of the problem (1.12)-(1.15). Formally,

$$\begin{aligned} & (x, y) \in Y \subseteq \mathbb{Z}_+^n \times \mathbb{R}_+^q \Leftrightarrow \\ & \exists z \in Z :\ A^{(1)}x + G^{(1)}y \leq b^{(1)},\ A^{(2)}x + G^{(2)}y + Vz \leq b^{(2)}. \end{aligned}$$

We also define the function

$$\phi(x, y) = \min \left\{ vz :\ Vz \leq b^{(2)} - A^{(2)}x - G^{(2)}y,\ z \in Z \right\}, \quad (x, y) \in Y.$$

Then the problem (1.12)-(1.15) can be reformulated as

$$\min \left\{ cx + hy + \eta :\ \eta = \phi(x, y),\ (x, y) \in Y \right\}. \tag{1.16}$$

Let $\{g_k(x, y)\}_{k \in K}$ be a finite set of linear functions which are lower bounds on the function $\phi(x, y)$ such that

$$\forall (x, y) \in Y : \begin{cases} \forall k \in K : & g_k(x, y) \leq \phi(x, y), \\ \exists k' \in K : & g_{k'}(x, y) = \phi(x, y), \end{cases}$$

and let $\{f_l(x,y)\}_{l\in L}$ be a finite set of linear functions defining the set Y:

$$Y = \{f_l(x,y) \le 0, l \in L\}.$$

In the cases encountered in the thesis, Z is either a polyhedron or a discrete set, so we can guarantee that the sets K and L of functions $g_k(x,y)$ and $f_l(x,y)$ exist and they are finite. The problem (1.16) can be further reformulated as a MIP

$$\begin{aligned}
\min \quad & cx + hy + \eta, && (1.17)\\
& A^{(1)}x + G^{(1)}y \le b^{(1)}, && (1.18)\\
& g_k(x,y) \le \eta, \quad k \in K, && (1.19)\\
& f_l(x,y) \le 0, \quad l \in L, && (1.20)\\
& x \in \mathbb{Z}^n_+, \quad y \in \mathbb{R}^q_+. && (1.21)
\end{aligned}$$

The problem (1.17)-(1.21) is solved by a modification of the branch-and-bound method. The modification is motivated by the huge number of constraints (1.19) and (1.20). Only a part of them is included in the formulation in the beginning. Other constraints are added only when they are needed. We now give a description of the modified branch-and-bound method.

- Subsets $K^1 \subseteq K$ and $L^1 \subseteq L$ of constraints are found. K^1 and L^1 can be empty.
- The *master problem*

  $$\min\left\{(1.17) :\ g_k(x,y) \le \eta, k \in K^1,\ f_l(x,y) \le 0, l \in L^1,\ (1.21)\right\} \quad (1.22)$$

  is solved by branch-and-bound (branch-and-cut).
- During the solution of the master problem, whenever an integer solution $(\bar{x},\bar{y},\bar{\eta}) \in \mathbb{Z}^n_+ \times \mathbb{R}^q_+ \times \mathbb{R}$ is found at some node of the enumeration tree, we need to check if point $(\bar{x},\bar{y})$ is really in Y and whether $\bar{\eta} = \phi(\bar{x},\bar{y})$. We do this in the following way.
  - By solving the *slave problem*

    $$\min\left\{vz :\ Vz \le b^{(2)} - A^{(2)}\bar{x} - G^{(2)}\bar{y},\ z \in Z\right\}.$$

    the value $\phi(\bar{x},\bar{y})$ is found. Let $\bar{z}$ be the solution of the slave problem.
  - If the slave problem is infeasible, then $(\bar{x},\bar{y}) \notin Y$. This means that some of the constraints (1.20) are violated. Then a constraint $f_{l'}(x,y) \le 0$, $l' \in L$, is found such that $f_{l'}(\bar{x},\bar{y}) > 0$. This constraint is added to the master problem as an *infeasibility cut*, and the current node of the master problem's enumeration tree is resolved. Note that the cut is global, i.e. valid throughout the tree.

– If the slave problem is feasible and $\bar{\eta} = v\bar{z}$, the solution $(\bar{x}, \bar{y}, \bar{\eta})$ is feasible for the problem (1.17)-(1.21), as

$$\bar{\eta} = v\bar{z} = \phi(\bar{x}, \bar{y}) \geq g_k(\bar{x}, \bar{y}), \forall k \in K.$$

In this case, $(\bar{x}, \bar{y}, \bar{z})$ is the best known solution for the original problem (1.12)-(1.15). This solution is stored, and the current node of the master problem's enumeration tree is pruned.

– If the slave problem is feasible and $\bar{\eta} < v\bar{z}$, the solution $(\bar{x}, \bar{y}, \bar{\eta})$ is not feasible for the problem (1.17)-(1.21), as

$$\exists k' \in K : \; g_{k'}(\bar{x}, \bar{y}) = \phi(\bar{x}, \bar{y}) = v\bar{z} > \bar{\eta}.$$

Then the constraint $g_{k'}(x, y) \leq \eta$ is added to the master problem as an *optimality cut*, and the current node of the master problem's enumeration tree is resolved. This cut is also global.

The method presented has one difficulty. There is no universal algorithm for generating optimality and infeasibility cuts. One has to derive such an algorithm for every particular problem or a class of problems. However, for some special cases, it is known how to generate cuts algorithmically.

When the set Z can be described by linear constraints, i.e. the slave problem is a Linear Problem, this approach reduces to the classical Benders decomposition [12]. In this case, the slave problems are solved by the simplex method, and cuts are generated using the dual information.

The case, in which the master is a pure Binary Problem and $v = 0$, has been considered by Jain and Grossmann [41] (related work is done in [25]). Here the slave problem is a feasibility problem. The infeasibility cut for a binary solution $\bar{x}$ of the master problem is

$$\sum_{i:\, \bar{x}_i = 1} (1 - x_i) + \sum_{i:\, \bar{x}_i = 0} x_i \geq 1. \tag{1.23}$$

The cut (1.23) has received the name *"no-good" cut* in the literature, as it cuts off only one infeasible solution, and it is rather weak. The modified branch-and-bound method presented above for the case in which $v = 0$ has received the name *branch-and-check* by Thorsteinsson [82].

Further generalizations of the classical Benders decomposition are considered by Hooker and Ottosson [44] and by Sen and Sherali [77]. The former paper deals with the case when the both master and slave problems are pure Binary Problems. In the latter paper the authors have studied the case when the both master and slave problems are arbitrary MIPs.

1.5 Constraint Programming

Constraint Programming (CP) is a method for formulating and solving combinatorial problems. The main difference between this method and Integer

Programming is that CP concentrates in the first place on the feasibility of a solution to a problem, whereas IP — on the optimality.

When the CP method is applied, the problem is formulated as a *Constraint Satisfaction Problem (CSP)* or a series of CSPs. A CSP consists of

- a finite set of variables x_j, $j \in N = \{1, \ldots, n\}$,
- *domains* D_j (sets of possible values) of variables, $j \in N$,
- a finite set of constraints $(x_1, \ldots, x_n) \in \Delta_i$, $i \in M$, over the set of variables,

and can be formally written as

$$\text{find} \quad x_j, \quad j \in N, \tag{1.24}$$
$$(x_1, \ldots, x_n) \in \Delta_i, \quad i \in M, \tag{1.25}$$
$$x_j \in D_j, \quad j \in N. \tag{1.26}$$

Each constraint (1.25) specifies combinations of values allowed for the variables. For each variable, we need to find (or assign) a value from its domain, such that all the constraints are satisfied. The solution of a CSP is an appropriate assignment of values to variables.

To solve a CSP, we also perform enumeration. Unlike bounding in branch-and-bound, here we use *constraint propagation* to reduce the enumeration. Constraint propagation can be described as the use of the problem's constraints in an active way to reduce the domains of variables. A basic technique to do constraint propagation is maintaining *arc-consistency* [86]. A constraint is said to be "arc-consistent" if for every variable $j \in N$ and every value $v_j \in D_j$, there exist values for other variables from their domains such that the constraint is satisfied. Formally, the constraint propagation technique, which maintains arc-consistency for constraint $(x_1, \ldots, x_n) \in \Delta$ can be described as follows.

$$\forall j \in N, \forall v_j \in D_j :$$
$$\Big\{(v_1, \ldots, v_{j-1}, v_{j+1}, \ldots, v_n) \in D_1 \times \ldots \times D_{j-1} \times D_{j+1} \times \ldots \times D_n :$$
$$(v_1, \ldots, v_n) \in \Delta\Big\} = \emptyset \Rightarrow D_j := D_j \setminus \{v_j\}.$$

For numerical CSPs, the domains of variables often are represented by intervals $[lb(x_j), ub(x_j)]$, $\forall j \in N$. Here $lb(x_j)$ and $ub(x_j)$ are, respectively, the smallest and the largest values variable x_j can take. A usual way to propagate constraints in numeric CSPs, is maintaining *arc-B-consistency* [57], i.e. arc-consistency restricted to the bounds of the domains. Formally, the constraint propagation technique, which maintains arc-B-consistency for con-

straint $(x_1, \ldots, x_n) \in \Delta$, can be described as follows.

$$\forall j \in N :$$
$$\Big(\underline{v}_j = \max\Big\{v_j \in D_j \ : \exists v_i \in D_i, i \in N \setminus \{j\},\ (v_1, \ldots, v_n) \in \Delta\Big\} > lb(x_j) \text{ or}$$
$$\overline{v}_j = \min\Big\{v_j \in D_j \ : \exists v_i \in D_i, i \in N \setminus \{j\},\ (v_1, \ldots, v_n) \in \Delta\Big\} < ub(x_j)\Big) \Rightarrow$$
$$D_j := [\underline{v}_j, \overline{v}_j].$$

Algorithms that perform constraint propagation are called *filtering algorithms*.

Example 1.2. To give an idea of the notion of constraint propagation, we consider a small problem:

$$\text{find}\Big\{x, y \in \mathbb{Z} :\ y < x,\ x < 5,\ y \geq 4,\ x \in (-\infty, 5),\ y \in [4, +\infty)\ \Big\}.$$

Using constraint $y < x$, the domain of variable x can be reduced to $x \in (4, 5)$, $x \in \mathbb{Z}$, which is empty. A contradiction is detected, thus the problem has no solution.

An important aspect of Constraint Programming is the *locality principle*. This principle says that the propagation of each constraint should be performed independently of other constraints of the problem. Observance of this principle leads to the design of separate filtering algorithms for each constraint. Therefore, it is possible to build and solve a new CP model of a problem using separate "bricks" (constraints with their filtering algorithms) without inventing a new algorithm. Some algorithms significantly improve the propagation by considering a set of constraints from a global point of view. Such an algorithm and the corresponding set of constraints form a *global constraint*.

Usually filtering algorithms alone are not sufficient to solve a CSP, i.e. they cannot either reduce the domains of variables to singletons or detect a contradiction. When this happens, branching is applied, as in branch-and-bound. We now outline the method for solving constraint satisfaction problems. It also works with an enumeration tree initialized with the original CSP. Each iteration has the following steps.

1. *Node selection.* If there are no active nodes, the original problem has no solution, and the algorithm is terminated. Otherwise, an active node is chosen and removed from the list of open nodes.

2. *Constraint propagation.* Filtering algorithms for the constraints of the problem are executed one after another. If, as a result, the domain of some variable becomes empty, the current subproblem has no solution, and the iteration is terminated. Otherwise, if all the domains become singletons, a solution is found, and the algorithm is terminated. Otherwise, if there were some domain reductions, the constraint propagation step is repeated.

3. *Variable selection.* At this step, there exist variables whose domains are not singletons. One such variable is chosen, say x_j.

4. *Branching.* There are two main ways to branch in CP. First, domain D_j of variable x_j is divided into two domains D'_j and D''_j. Then two descendant nodes are created, which inherit the current subproblem with one exception: at one node variable x_j changes its domain to D'_j, at another node to D''_j. Alternatively, when domain D_j is finite, it can be divided into singletons. In this case, the number of descendant nodes equals the number of values in D_j.

As you can see, the enumeration in Constraint Programming resembles the enumeration in Integer Programming. We now briefly compare these two methods in this respect. In both IP and CP, node the node and variable selection strategies play an important role in the efficiency of the enumeration. The branching is also similar in some sense. When branching, we limit the possible set of values for a variable either explicitly or by adding constraints to the problem. The main difference is the way the enumeration is reduced. In CP, it is done using constraint propagation, whereas in IP, bounding and cut generation are applied.

Now we show how to formulate the feasibility variant of a one-machine non-preemptive scheduling problem as a CSP. The feasibility problem is the following. Set $N = \{1, \ldots, n\}$ of jobs should be processed on a machine which can handle one job at a time. Each job $j \in N$ has a processing time p_j, a release dates r_j and a deadline $\bar{d}_j$. The aim is to find a feasible schedule of jobs in N, i.e. a schedule in which job execution is not interrupted and release dates and deadlines of jobs are respected.

For each job $j \in N$ we introduce variable $start_j$. It represents the starting time of job j. Domain D_j of variable $start_j$ is an interval defined by the parameters of job j: $D_j = [r_j, \bar{d}_j - p_j]$ (see Figure 1.1). Domains represent the constraint that jobs should be processed (started and completed) within their *time windows* $[r_j, \bar{d}_j]$.

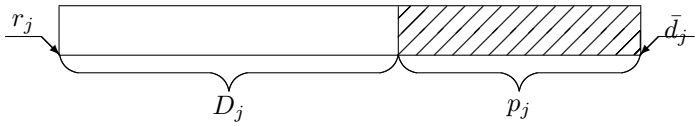


Figure 1.1: Job j and domain D_j of variable $start_j$

The model also contains one global constraint called `disjunctive`. This constraint guarantees that the machine processes at most one job at a time and there is no preemptions in job processing. `disjunctive` is represented by simple constraints in the following way

$$(start_i + p_i \leq start_j) \vee (start_j + p_j \leq start_i), \quad \forall i, j \in N.$$

Now we can give the CP formulation of the problem:

$$\begin{aligned} \text{find} \quad & start_j, \quad j \in N, \\ & \texttt{disjunctive}(start_1, \ldots, start_n, p_1, \ldots, p_n), \\ & start_j \in [r_j, \bar{d}_j - p_j], \quad j \in N. \end{aligned}$$

For the constraint `disjunctive`, several filtering techniques exist in the literature, see [7] for a review. As an example we now consider the *"Edge-Finding"* filtering technique, mainly because it is used later in the thesis.

"Edge-Finding" filtering algorithms reduce the domains of variables *start* (i.e. time windows of jobs) by deducing implicit precedence relations between jobs. Consider a job $k \in N$ and a subset $\Omega \subseteq N \setminus \{k\}$ of jobs. If the difference between the earliest time when jobs in $\Omega \cup \{k\}$ can be started and the latest time when jobs in Ω can be completed is strictly less than the total processing time of jobs in $\Omega \cup \{k\}$, then job k should be processed last among the jobs in $\Omega \cup \{k\}$. Otherwise, the job processed last among jobs in Ω would be completed after its deadline. As job k should be processed after all the jobs in Ω, its release date can be adjusted to the earliest time moment when all the jobs in Ω can be completed. Formally, the rule for adjusting release dates using the "Edge-Finding" technique can be written as

$$\begin{aligned} & \forall \Omega \in N, \forall k \in N \setminus \Omega : \ \bar{d}_\Omega - r_{\Omega \cup \{k\}} < p_\Omega + p_k \Rightarrow \Omega \rightarrow k \Rightarrow \\ & start_k \geq \max_{\emptyset \neq \Omega' \subseteq \Omega} \{r_{\Omega'} + p_{\Omega'}\} \Rightarrow r_k := \max_{\emptyset \neq \Omega' \subseteq \Omega} \{r_{\Omega'} + p_{\Omega'}\}, \end{aligned} \tag{1.27}$$

where "$\rightarrow$" means "precedes". The rule for adjusting deadlines is symmetrical:

$$\begin{aligned} & \forall \Omega \in N, \forall k \in N \setminus \Omega : \ \bar{d}_{\Omega \cup \{k\}} - r_\Omega < p_\Omega + p_k \Rightarrow k \rightarrow \Omega \Rightarrow \\ & start_k \leq \max_{\emptyset \neq \Omega' \subseteq \Omega} \{\bar{d}_{\Omega'} - p_{\Omega'}\} - p_k \Rightarrow \bar{d}_k := \max_{\emptyset \neq \Omega' \subseteq \Omega} \{\bar{d}_{\Omega'} - p_{\Omega'}\}. \end{aligned} \tag{1.28}$$

There are a priori $O(n \cdot 2^n)$ pairs (k, Ω) to consider. But there exist algorithms that perform all the time window adjustments (1.28) and (1.27) in $O(n^2)$ time [7].

More information about applying Constraint Programming to scheduling problems can be found in the book by Baptiste et al. [7].

As we have seen, both Constraint Programming and Integer Programming are intended to solve combinatorial problems. And they have complementary strengths in the following sense. IP is usually better suited for solving problems in which the coefficients of variables in the objective function vary considerably. On the other side, CP is typically the choice when solving problems with complicated constraints. When we have a problem which has both these characteristics, such as some scheduling problems, it natural to use the strengths of both methods. Thus, we need so-called *hybrid methods* which combine CP and IP. For a review of different hybrid methods, the reader can

refer to [42]. One approach for combining IP and CP is the (hybrid) branch-and-check method. In this method, IP is used to solve the master problem, and CP is applied to solve the slave feasibility problems.

However, in the thesis, we do not use CP explicitly. Typically, CP gives only a limited information about the problem under resolution. For a feasibility problem, the answer of a CP algorithm is either "feasible" or "not feasible". There is no answer *why* a problem is infeasible. But such information is very useful to speed up a hybrid method, and particularly a branch-and-check algorithm. Therefore, on the one hand, we use ideas from CP, such as "Edge-Finding" technique. On the other hand, we do not stick to the usual CP framework, and try to retrieve the information about the reasons of infeasibility.

1.6 Outline of the Thesis

The aim in this thesis is to develop efficient algorithms to solve some strongly NP-hard scheduling problems to optimality using a combination of ideas from IP, CP and scheduling theory. The main "ingredient" of our approaches is Integer Programming. Usually, a pure IP approach is not very efficient when solving machine scheduling problems. However, our intuition suggests that it should be used as a part of an algorithm when the objective function is "sophisticated". And decomposition methods become crucial for this approach.

The main idea on which a large part of our results is based is to separate the optimality and feasibility components of the problem and let different methods tackle these components. Then IP is "responsible" for optimization, whereas specific combinatorial algorithms, which heavily use the structure of the problem, either generate feasible solutions and send them to IP, or check the feasibility of solutions provided by IP.

In Chapter 2 of the thesis we consider the problem $1 \mid r_j \mid \sum w_j U_j$, the one-machine scheduling problem with unequal release dates and the objective of minimizing the sum of the weights of the late jobs. The problem is formulated as a MIP and decomposed using the generic Benders scheme. Then it is solved by branch-and-check.

We propose several problem-specific improvements of the general method. First, the master problem is amended by the addition of constraints which form a relaxation of the scheduling problem. These additional constraints do not have a big impact on the time to solve the master problem but greatly decrease the number of infeasible solutions produced by it. Secondly, the standard "no-good" infeasibility cuts are replaced by stronger cuts. This change significantly decreases the number of cuts needed to solve the problem. Two families of stronger cut are developed and algorithms are designed to generate these cuts. Computational experiments indicate that our algorithm is highly competitive with other approaches available in the literature for solving this scheduling problem.

A large part of Chapter 3 is devoted to the multi-machine assignment scheduling problem (MMASP), a generalization of the problem $1 \mid r_j \mid \sum w_j U_j$. Several unrelated machines are available to process a set of jobs. In contrast to the one-machine problem, each job does not have a cost, but has a profit which is obtained only if the job is completed on-time. The profit and processing time of each job depend on the machine on which this job is processed. The aim is to maximize the overall profit.

Our branch-and-check algorithm for the one-machine scheduling problem is extended to the MMASP. The extension is relatively simple and basically consists in adding assignment inequalities to the master problem. Additionally, we propose another approach for solving the MMASP based on Dantzig-Wolfe decomposition. A decomposed MIP formulation is solved using the column generation-based branch-and-price method. In the column generation algorithm, the pricing problem is exactly the problem $1 \mid r_j \mid \sum w_j U_j$ which is solved using the branch-and-check algorithm from Chapter 2. The branch-and-price algorithm proposed for the MMASP outperforms the branch-and-check algorithm and other approaches in the literature.

At the end of Chapter 3, we adopt the algorithms proposed for solving the MMASP to the problem $R \mid r_j \mid L_{\max}$. Finally, we consider the special case of these two multi-machine problems, when machines are identical. Ways to exploit symmetry are discussed.

In Chapter 4, we address the problem $1 \mid\mid \sum w_j T_j$. This problem is one of the most challenging one-machine problems and it has received a lot of attention in the literature over the last 40 years. For this problem, we propose an original compact MIP formulation based on a special partition of the time horizon into intervals. Each interval has a permutation of jobs associated with it. Jobs completed in the same interval are required to be processed according to the permutation associated with this interval, i.e the number of schedules we consider is reduced. The partition is carried out in such a way that this reduced set of schedules contains an optimal one. The MIP formulation proposed is then compared numerically with other MIP formulations available in the literature.

Next, a Danzig-Wolfe decomposition of our MIP is considered. We show that, while solving the LP relaxation of the restricted master problem by column generation, the pricing problem can be solved in a pseudopolynomial time by dynamic programming. Ways to strengthen the LP relaxation of the restricted master using cuts are also discussed.

We evaluate numerically the quality of lower bounds obtained by solving the LP relaxation of the master problem. There is a good trade-off between the quality of lower bounds and the time needed to obtain them.

In Chapter 5, we draw some conclusions and discuss possible directions to extend the research presented in the thesis.

Chapter 2

Minimizing the sum of the weights of the late jobs on a single machine

In this chapter we consider the scheduling problem of minimizing the weighted number of late jobs on a single machine ($1 \mid r_j \mid \sum w_j U_j$). A branch-and-check algorithm is proposed to solve this problem to optimality.

In Section 2.1, we introduce the problem and review the literature. A MIP formulation of the problem is given in Section 2.2. We decompose this formulation by following the generic Benders scheme presented in Section 1.4. Then, a relaxation of the formulation is proposed. We use the constraints of this relaxation to initialize the master problem. In Section 2.3, we turn to solving the feasibility slave problem and generating cuts. Two families of infeasibility cuts are suggested, which are stronger than standard "no-good" cuts. First, a modification of the algorithm by Carlier [20] is developed to solve the slave problem and derive a strengthened "no-good" cut, when the slave problem does not have a feasible solution. Secondly, we show how to generate cuts by using the "Edge-Finding" constraint propagation technique. Section 2.4 discusses different variants of the branch-and-check algorithm proposed. Finally in Section 2.5, we test our algorithm on a set of instances with up to 140 jobs. Also, a comparison is made with the approach by Péridy at al. [63] which is, to our knowledge, the best one reported in the literature.

The results presented in this chapter have been published in [72, 73].

2.1 Introduction

In this chapter, we study the scheduling problem $1 \mid r_j \mid \sum w_j U_j$. A set of jobs $N = \{1, \ldots, n\}$ has to be processed on a single machine. Only one job

can be processed at a time and preemptions are not allowed. Each job $j \in N$ has a release date r_j, a processing time p_j, a due date d_j and a weight w_j. The objective is to minimize the weighted number of late jobs, i.e. to find a schedule $\pi \in \Pi(N)$, for which the sum of the weights of the late jobs is minimized. In the scheduling literature, this criteria is usually called "the weighted number of late jobs".

The problem is $\mathcal{NP}$-hard in the strong sense even when all weights are the same $(1 \mid r_j \mid \sum U_j)$, as shown by Lenstra et al. [55]. The special case of the problem without release dates $(1 \mid\mid \sum w_j U_j)$ is $\mathcal{NP}$-hard in the ordinary sense. A pseudopolynomial algorithm for this case has been proposed by Lawler and Moore [53]. Later Lawler et al. [54] developed a pseudopolynomial algorithm for the case with a fixed number of identical machines $(Pm \mid\mid \sum w_j U_j)$.

Some special cases of the problem can be solved in polynomial time. Moore [60] showed that the case without release dates and identical weights $(1 \mid\mid \sum U_j)$ can be solved in $O(n \log n)$. The polynomiality of the case with release dates and identical weights when release and due dates are ordered in the same way $(r_i < r_j \Rightarrow d_i \leq d_j)$ has been proven by Kise et al. [47]. Also an $O(n \log n)$ algorithm for this case is given by Lawler in [52]. Another polynomial algorithm by Lawler [50] solves the weighted case without release times when processing times and due dates are ordered in the opposite way.

For the $1 \mid r_j \mid \sum U_j$ problem Baptiste et al. [9] provided an exact algorithm that solves 99% of the test instances with 100 jobs and 80% of the instances with 160 jobs in one hour.

Recently several approaches to solve the general case with release dates and weights $1 \mid r_j \mid \sum w_j U_j$ have appeared in the literature. Baptiste et al. [8] proposed a Constraint Programming-based approach, which can be also applied when there are several identical parallel machines $(P \mid r_j \mid \sum w_j U_j)$. An algorithm which solves a Lagrangean relaxation of an original MIP formulation in order to obtain upper and lower bounds for the problem was suggested by Dauzère-Pérès and Sevaux [30]. A branch-and-bound algorithm was proposed by Péridy, Pinson and Rivreau [63]. It uses a Lagrangean relaxation of the time-indexed MIP formulation to compute lower bounds. This algorithm appears to be the most efficient algorithm available to solve the problem to optimality. It has been tested on a set of instances with up to 100 jobs. To tackle larger instances, some genetic algorithms were proposed by Sevaux and Dauzère-Pérès [79]. In this chapter, we present an exact branch-and-check algorithm which improves on the results of Péridy et al.

2.2 Formulation and Decomposition

As we have already seen in Example 1.1, problems with the criterion $\sum(w_j)U_j$ have the important property that late jobs can be processed arbitrarily late without increasing the cost. This observation allows us to schedule only on-time jobs and leave late jobs unscheduled. Late jobs can be processed in any

order after all the on-time jobs.

We now give a standard MIP formulation of the problem. This formulation appeared previously in [41, 16] in a multi-machine context. Binary variable x_j, $j \in N$, takes value 1 if job j is scheduled on-time, and otherwise $x_j = 0$. Sequencing binary variable δ_{ij}, $i, j \in N$, takes value 1 if both jobs i and j are on-time and i precedes j, and otherwise $\delta_{ij} = 0$. Assume without lost of generality that $p_j \leq d_j - r_j \ \forall j \in N$. Continuous variable s_j, $j \in N$, equals the starting time of job j if j is on-time. If j is late, s_j takes an arbitrary value from interval $[r_j, d_j - p_j]$. The standard formulation (SM) for our single machine problem is as follows.

$$\textbf{(SM)} \qquad \begin{aligned}
\min \quad & \sum_{j \in N} w_j \cdot (1 - x_j) && (2.1)\\
& s_j \geq s_i + p_i - U(1 - \delta_{ij}), \quad i, j \in N, i \neq j, && (2.2)\\
& s_j \geq r_j, \quad j \in N, && (2.3)\\
& s_j + p_j \leq d_j, \quad j \in N, && (2.4)\\
& \delta_{ij} + \delta_{ji} \leq x_j, \quad i, j \in N, i \neq j, && (2.5)\\
& \delta_{ij} + \delta_{ji} \geq x_i + x_j - 1, \quad i, j \in N, i < j, && (2.6)\\
& x \in \{0,1\}^n, \quad \delta_{ij} \in \{0,1\}^{n \times n}. && (2.7)
\end{aligned}$$

Let $\Delta(r, p, d)$ denote the feasible region (2.2)-(2.7) for variables (x, δ, s). Note that the objective function is equivalent to

$$\max \quad \sum_{j \in N} w_j x_j.$$

The constraints (2.2) ensure that job i precedes job j if $\delta_{ij} = 1$. Here U is a big value and can be, for example, instantiated as the difference between the maximum due date and the minimum release date. Note that, in this case, if $\delta_{ij} = 0$, we have

$$s_i + p_i - s_j \leq U = d_{\max} - r_{\min} \Rightarrow$$

$$\Rightarrow \begin{cases} s_i \leq d_{\max} - r_{\min} + s_j - p_i, \text{ which is dominated by } s_i \leq d_i - p_i, \\ s_j \geq s_i + p_i - d_{\max} + r_{\min}, \text{ which is dominated by } s_j \geq r_j, \end{cases}$$

and (2.2) is dominated by constraints (2.3) and (2.4).

Constraints (2.3) and (2.4) force the on-time jobs to be processed inside their time windows. The constraints (2.5) and (2.6) relate the variables x and δ: the first group ensures that both sequencing variables δ_{ij} and δ_{ji} take value zero if job j is late; the second group guarantees that if jobs i and j are both on-time, then one must be processed before the other.

It was shown in [41] that the formulation (SM) cannot be solved efficiently by the standard branch-and-bound method. Instead, we propose to decompose and solve this MIP using the generic Benders approach. In the master problem, we leave the variables x that determine which jobs are on-time and

which jobs are late. The variables δ and s specifying a feasible schedule for on-time jobs are moved to the slave problem. As both δ and s do not have coefficients in the objective function, the slave problem is a feasibility problem. Thus, the decomposed MIP can be solved by branch-and-check.

The first step of a branch-and-check algorithm is the initialization of the master problem with a set of constraints $f_l(x) \leq 0$, $l \in L^1$. As can be seen from Section 1.4, these constraints should form a relaxation of the feasible region $\Delta(r, p, d)$ in the space of the variables x. Set L^1 can be empty but, as has been pointed out in [16, 82], a well selected initial relaxation usually plays an important role in the efficiency of the approach.

Now we present families of valid inequalities for $\Delta(r, p, d)$. We will use some of these inequalities to form the required relaxation.

A first obvious observation is that the sum of the processing times of all on-time jobs cannot exceed the difference between the maximum due date d_N and the minimum release date r_N, i.e. the length of the interval $[r_N, d_N]$. Thus, the inequality

$$\sum_{j \in N} p_j x_j \leq d_N - r_N \tag{2.8}$$

is valid for $\Delta(r, p, d)$. The next proposition generalizes the valid inequality (2.8) to any interval.

Proposition 2.1. *Consider an interval $[r', d']$, $0 \leq r' < d'$. Let $Q_{[r',d']}$ be the set of jobs that must be processed inside $[r', d']$ if there are on-time: $Q_{[r',d']} = \{j \in N : \ r_j \geq r', d_j \leq d'\}$. Then the inequality*

$$\sum_{j \in Q_{[r',d']}} p_j x_j \leq d' - r' \tag{2.9}$$

is valid for $\Delta(r, p, d)$.

Remark. Note that the formulation

$$\begin{aligned} \min \quad & \sum_{j \in N} w_j \cdot (1 - x_j) \\ & \sum_{l \in Q_{[r_i,d_j]}} p_l x_l \leq d_j - r_i, \quad i, j \in N, r_i < d_j, \\ & x \in \{0,1\}^n. \end{aligned}$$

is correct for the preemptive problem $1 \mid pmtn, r_j \mid \sum w_j U_j$, which is NP-hard in the ordinary sense. This observation follows from the result of Carlier [21] which says that there exists a feasible preemptive schedule if and only if for all $i, j \in N$, $r_i < d_j$, the sum of the processing times of the jobs in $Q_{[r_i,d_j]}$ is less than or equal to $d_j - r_i$.

The constraint (2.9) can be strengthened by taking into account not only the jobs that should be processed completely within an interval, but also the

jobs that should be at least partly processed. Let χ^+ denote the positive part of the value χ: $\chi^+ = \max\{\chi, 0\}$. For a job $j \in N$, let $\alpha_j(r') = (r' - r_j)^+$ and $\beta_j(d') = (d_j - d')^+$. Then, if job j is on-time, it should be processed in interval $[r', d']$ at least during the time

$$\epsilon_j(r', d') = \min\left\{d' - r', \Big(p_j - \alpha_j(r')\Big)^+, \Big(p_j - \beta_j(d')\Big)^+\right\}.$$

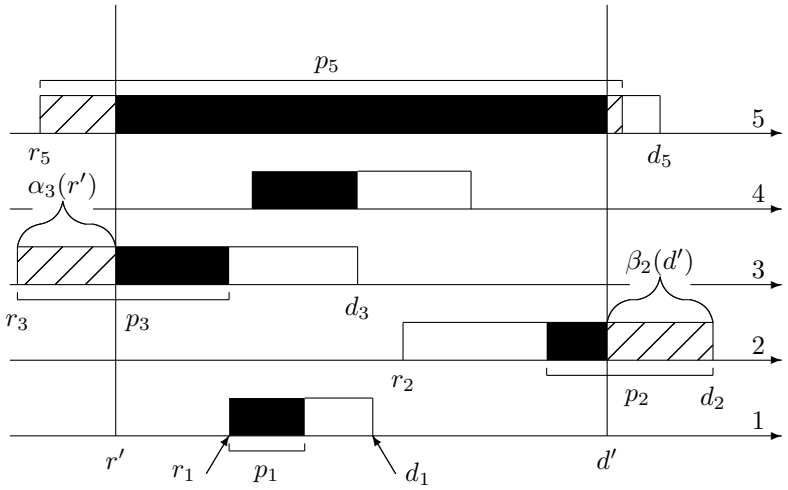


Figure 2.1: An illustration of jobs that should be at least partly processed in interval $[r', d']$.

To give some intuition to the to the reader, in Figure 2.1, we illustrate a time interval $[r', d']$ and the jobs that should be at least partly processed in it. For each job $j \in N$ its time window $[r_j, d_j]$ is shown as a rectangle. The length of the black area corresponds to the part of processing time p_j which should be processed within the interval $[r', d']$ when job j is on-time. The length of the hatched area corresponds to the part of the processing time which can be executed outside the interval. For example, in the Figure 2.1, $\alpha_1(r') = \beta_1(d') = \alpha_4(r') = \beta_4(d') = 0$ and $\epsilon_1(r', d') = p_1$, $\epsilon_4(r', d') = p_4$. Then, at most $\beta_2(d')$ units of processing time of job 2 can be executed outside the interval, and $\epsilon_2(r', d') = p_2 - \beta_2(d')$. Again at most the part $\alpha_3(r')$ of length of job 3 can be executed outside the interval, and $\epsilon_3(r', d') = p_3 - \alpha_3(d')$. Then, whenever job 5 starts, it should occupy whole interval if job 5 is on-time, therefore $\epsilon_5(r', d') = d' - r'$.

The sum $\sum_{j \in N} \epsilon_j(r', d')$ should not exceed the length of the interval. We prove it formally in the next proposition.

Proposition 2.2. *Consider an interval* $[r', d']$, $0 \leq r' < d'$. *Then the inequality*

$$\sum_{j \in N} \epsilon_j(r', d') \cdot x_j \leq d' - r' \tag{2.10}$$

is valid for $\Delta(r, p, d)$.

Proof. For each job $j \in N$, we show that at least the amount $\epsilon_j(r', d')$ of its total processing time p_j must lie inside the interval $[r', d']$.

If $r_j \geq r'$ or $\alpha_j(r') = 0$, the maximum amount that can be processed after d' is $(d_j - d')^+ = \max\{\alpha_j(r'), \beta_j(d')\}$. Hence the minimum processing time within the interval is $(p_j - \max\{\alpha_j(r'), \beta_j(d')\})^+ = \min\{(p_j - \alpha_j(r'))^+, (p_j - \beta_j(d'))^+\} \leq d' - r_j \leq d' - r'$, because we assume that $p_j \leq d_j - r_j$. The case when $d_j \leq d'$ or $\beta_j(d') = 0$ is similar.

Otherwise the interval $[r', d']$ lies strictly within the interval $[r_j, d_j]$, or equivalently $\alpha_j(r') > 0$ and $\beta_j(d') > 0$. Suppose that $\beta_j(d') \geq \alpha_j(r')$. Then if $p_j - \beta_j(d') < d' - r'$, the maximum amount of processing time of j that can lie outside the interval is $\beta_j(d')$. Thus the minimum processing time within the interval is $p_j - \beta_j(d') = p_j - \max\{\alpha_j(r'), \beta_j(d')\} = \min\{(p_j - \alpha_j(r'))^+, (p_j - \beta_j(d'))^+\}$. Alternatively, if $p_j - \beta_j(d') \geq d' - r'$, job j occupies the whole interval and uses $d' - r'$ of the processing time. □

Note that the inequalities (2.10) comply with the concept of energetic reasoning [35, 58] used in Constraint Programming. Adding these inequalities to a MIP model corresponds to performing energetic tests in CP, as in [7]. We now define the set $\mathcal{X}(r, p, d)$ of points which satisfy the constraints (2.10) for all possible intervals:

$$\mathcal{X}(r, p, d) = \{x \in \{0, 1\}^n : \ \epsilon(r', d') \cdot x \leq d' - r', \ \forall r' \geq 0, \forall d' > r'\}.$$

From Constraint Programming theory, it is known that performing the energetic tests for all possible intervals is equivalent to performing the energetic tests for a certain subset of intervals, whereas the remaining inequalities are dominated. In the next proposition, we formalize this property in the context of inequalities (2.10). First, let us define the sets O_1, O_2, $O(t)$, $t \in \mathbb{R}$:

$$\begin{aligned} O_1 &= \{r_j\}_{j \in N} \cup \{d_j - p_j\}_{j \in N} \cup \{r_j + p_j\}_{j \in N}, \\ O_2 &= \{d_j\}_{j \in N} \cup \{r_i + p_i\}_{j \in N} \cup \{d_i + p_i\}_{j \in N}, \\ O(t) &= \{r_j + d_j - t\}_{j \in N}. \end{aligned}$$

Proposition 2.3.

$$\begin{cases} \epsilon(r', d') \cdot x \leq d' - r', & \forall r' \geq 0, \forall d' > r', \\ x \in \mathbb{R}^n_+. \end{cases}$$

$$= \begin{cases} \epsilon(r', d') \cdot x \leq d' - r', & \forall r' \in O_1, \forall d' \in O_2, \\ \epsilon(r', d') \cdot x \leq d' - r', & \forall r' \in O_1, \forall d' \in O(r'), \\ \epsilon(r', d') \cdot x \leq d' - r', & \forall d' \in O_2, \forall r' \in O(d'), \\ x \in \mathbb{R}^n_+. \end{cases}$$

Proof. This proposition is equivalent to Proposition 19 in [7]. □

We can take $\mathcal{X}(r,p,d)$ as a relaxation of $\Delta(r,p,d)$ in the branch-and-check algorithm. However, preliminary numerical experiments showed that it is computationally advantageous to use only a subset of the constraints which form $\mathcal{X}(r,p,d)$. Thus, as a relaxation of $\Delta(r,p,d)$, we take the set

$$\bar{\mathcal{X}}(r,p,d) = \{x \in \{0,1\}^n : \ \epsilon(r_i,d_j)\cdot x \leq d_j - r_i, \ \forall i,j \in N, r_i < d_j\}.$$

We now outline the basic branch-and-check algorithm for the problem $1 \mid r_j \mid \sum w_j U_j$. The formulation (SM) can be rewritten in a compact form.

$$\begin{array}{ll} \max & wx \\ s.t. & x \in \bar{\mathcal{X}}(r,p,d), \\ & (x,\delta,s) \in \Delta(r,p,d). \end{array}$$

The master MIP

$$\max\{wx : \ x \in \bar{\mathcal{X}}(r,p,d)\}$$

is solved using the branch-and-cut method. Whenever an integer solution $\bar{x}$ is found at some node of the enumeration tree, the slave feasibility problem

$$\text{find}\{\delta, s : \ (\bar{x},\delta,s) \in \Delta(r,p,d)\} \tag{2.11}$$

is solved to check if there is a feasible schedule containing all the jobs in the set $\bar{J} = \{j \in N : \ \bar{x}_j = 1\}$. If (2.11) does not have a solution, an infeasibility cut is generated.

The slave problem can be solved by Constraint Programming after reformulating it as

$$\begin{array}{ll} \text{find} & s_j, \quad j \in \bar{J}, \\ & \texttt{disjunctive}(\{s_j\}_{j\in\bar{J}}, \{p_j\}_{j\in\bar{J}}), \\ & s_j \in [r_j, d_j - p_j], \quad j \in \bar{J}. \end{array}$$

For this problem, the infeasibility cut for this case suggested in the literature [41, 16] is the cut:

$$\sum_{j\in\bar{J}} x_i \leq \mid \bar{J} \mid -1. \tag{2.12}$$

This approach for solving the slave problem and generating cuts is not very efficient, as the cut (2.12) is not much better than the "no-good" cut (1.23). A very large number of cuts is needed to solve instances with more than 50 jobs, and therefore the basic branch-and-check algorithm presented does not terminate within a reasonable amount of time. In the following sections we present methods to generate stronger infeasibility cuts, which may help to solve bigger instances faster.

2.3 Generation of Infeasibility Cuts

In this section, we discuss ways to improve the basic branch-and-check algorithm given in Section 2.2. We propose alternative algorithms to check the solution of the master and to generate infeasibility cuts. In subsection 2.3.1, a modification of the Carlier algorithm [20] is suggested to check the feasibility of an integer solution of the master problem and to generate strengthened cuts of the type (2.12). In subsection 2.3.2, we present propagation-based inequalities that are a generalization of the interval inequalities (2.10). In subsection 2.3.3, a *separation algorithm* for a subclass of the propagation-based inequalities is proposed, i.e. an algorithm that checks whether an inequality in this family exists which is violated by a solution of the master MIP.

2.3.1 Modified Carlier Algorithm

In this subsection, we work with the set $\bar{J}$ of jobs obtained from an integer solution $\bar{x}$ of the master MIP. To check the feasibility of $\bar{J}$, we solve a more general problem.

We first introduce some notation. Consider a set Q of jobs. We call a schedule $\pi \in \Pi(Q)$ *feasible* with respect to constant L' if the maximum lateness of π is less than or equal to L', i.e., $L_{\max}(\pi) \leq L'$. Set Q of jobs is *feasible* with respect to L' if and only if there exists a feasible schedule $\pi \in \Pi(Q)$.

Obviously, all jobs in the set Q can be scheduled on-time only if there exists a schedule $\pi \in \Pi(Q)$ with nonpositive maximum lateness. Therefore, for a given vector $\bar{x}$, the slave problem has a solution if and only if the set $\bar{J} = \{j \in N : \bar{x}_j = 1\}$ of jobs is feasible with respect to the value 0. Feasibility of the set $\bar{J}$ can be verified by solving the problem of minimizing the maximum lateness with release dates on a single machine $1 \mid r_j \mid L_{\max}$, i.e., by finding a schedule $\pi^* \in \Pi(\bar{J})$ that minimizes the maximum lateness $L^*_{\max}$ and checking if $L^*_{\max} \leq 0$. To solve such a problem we can use, for example, the algorithm of Carlier [20] which is not polynomial but performs well in practice. Then, if the set $\bar{J}$ is infeasible, the cut (2.12) is added to the master problem.

To strengthen the cuts (2.12), we use the following idea. In many cases, in which the set $\bar{J}$ of jobs is infeasible, there exists a subset $Q \subset \bar{J}$ which is also infeasible. If it is possible to find such a subset Q, the cut

$$\sum_{j \in Q} x_j \leq \mid Q \mid -1 \tag{2.13}$$

is stronger than the cut (2.12). The inequality (2.13) is stronger, because, if a solution $\bar{x}$ violates (2.13), it also violates (2.12) but not vice versa, as $Q \subset \bar{J}$.

The strongest cut of the type (2.13) is based on the smallest possible infeasible subset Q of jobs. However, finding a minimum, or even a minimal infeasible subset of jobs seems to be a very difficult problem. Even checking

feasibility is $\mathcal{NP}$-hard in the strong sense. In addition only a limited time can be spent on finding an infeasible set Q of jobs since this computation is a subroutine which can be called many times. We therefore turn to a heuristic approach which tries to minimize the number of jobs in Q.

We suggest a modification of the Carlier algorithm [20] for solving the following one-machine feasibility problem. Given a set N of jobs with release dates, processing times, due dates and a constant L', we need to find a feasible schedule $\pi^* \in \Pi(N)$, or, else, an infeasible subset $Q \subseteq N$ of jobs.

The algorithm is of the branch-and-bound type. Triples of job parameters $\{r_j^{(\varphi)}, p_j, d_j^{(\varphi)}\}$, $j \in N$, are associated with each node φ of the search tree. Release dates and due dates of jobs can differ from one node to another. The parameters for the top node are given by the initial data.

At each node of the search tree, the Schrage schedule [75] π_σ is constructed using the following procedure. Jobs are scheduled one by one. Suppose the jobs from the subset $N' \subseteq N$ are already placed in the first $\mid N' \mid$ positions of the schedule. Let $C_{N'} = \max_{j \in N'} C_j(\pi_\sigma)$ be the maximum completion time of these jobs. In the next position in π_σ a job is placed that is unscheduled, available (with release date less or equal to $C_{N'}$) and with the smallest due date. If there are no available jobs, a job with the smallest due date from the set of unscheduled jobs with the smallest release date is placed next in the schedule. If the Schrage schedule is feasible, the algorithm is stopped and the set N is feasible.

The next theorem establishes some properties of infeasible Schrage schedules. The branching rule is based on these properties. The theorem uses the following lemma from [20].

Lemma 2.4. *For any subset $N' \subseteq N$,*

$$h(N') = \min_{j \in N'} r_j + \sum_{j \in N'} p_j - \max_{j \in N'} d_j = r_{N'} + p_{N'} - d_{N'}$$

is a lower bound on the maximum lateness of any schedule $\pi \in \Pi(N)$.

This lower bound is also used in the algorithm itself.

Theorem 2.5 is a modified version of the main theorem in [20]. The modification is caused by the presence of the constant L' and, as a consequence, the different choice of job a (see Theorem 2.5). In the theorem, jobs are renumbered in the order in which they are sequenced in the Schrage schedule. The two cases considered in the theorem are illustrated in Figures 2.2 and 2.3. Remember that $S_j(\pi)$ and $L_j(\pi)$ are, respectively, the starting time and the lateness of job $j \in N$ in schedule π.

Theorem 2.5. *Suppose that the Schrage schedule $\pi_\sigma \in \Pi(N)$ is infeasible with respect to constant L' and b is the smallest integer such that $L_b(\pi_\sigma) > L'$. Let a be the largest integer such that $a \leq b$ and*

$$S_a(\pi_\sigma) - \min_{a \leq j \leq b} r_j < L_b(\pi_\sigma) - L'. \tag{2.14}$$

Let $Q = \{a, \ldots, b\}$, then:

1. *if there is no job $c \in Q$ with $d_c > d_b$ then the set Q of jobs is infeasible with respect to constant L';*

2. *otherwise, let c be the last job in Q with $d_c > d_b$, and $J = \{c+1, \ldots, b\}$. Then, if there exists a feasible schedule $\pi' \in \Pi(N)$, then, in π', job c is processed either before all the jobs in J or after.*

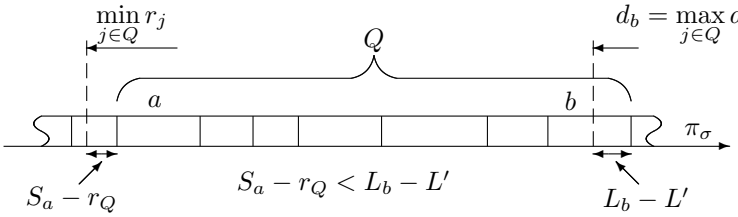


Figure 2.2: The infeasible Schrage schedule: case 1.

Proof. 1. Observe that there is no idle time between jobs from set Q in schedule π_σ. If there was idle time, this would imply the existence of job a', $a' > a$, which is scheduled immediately after the idle time in π_σ and satisfies condition (2.14), because the following would hold: $S_{a'}(\pi_\sigma) = r_{a'} = \min_{j\in\{a',\ldots,b\}} r_j$ (by the rules of constructing the Schrage schedule π_σ). This would contradict the condition that a is the largest such number. Notice, that job a always exists, since the first job in schedule π_σ satisfies (2.14).

So

$$L_b(\pi_\sigma) = S_a(\pi_\sigma) + \sum_{j\in Q} p_j - d_b. \tag{2.15}$$

Since there is no job $c \in Q$, such that $d_c > d_b$, then $d_b = \max_{j\in Q} d_j$. Using this fact and condition (2.14), (2.15) gives:

$$\min_{j\in Q} r_j + \sum_{j\in Q} p_j - \max_{j\in Q} d_j > L'. \tag{2.16}$$

By Lemma 2.4 the left-hand side of (2.16) is a lower bound on the maximum lateness for any schedule in $\Pi(Q)$. It follows that there is no schedule $\pi' \in \Pi(Q)$, such that $L_{\max}(\pi') \leq L'$, and therefore Q is infeasible with respect to L'.

2. Since $c \in Q$ is the last job in Q with $d_c > d_b$, $d_b = \max_{j\in J} d_j$. Since there is no idle time between jobs from Q in schedule π_σ, $L_b(\pi_\sigma) = S_c(\pi_\sigma) + p_c + \sum_{j\in J} p_j - d_b$. Notice that $S_c(\pi_\sigma) < \min_{j\in J} r_j$. Otherwise a job from J would be scheduled instead of job c in the Schrage schedule π_σ (as $d_c > d_b = \max_{j\in J} d_j$). If schedule $\pi' \in \Pi(N)$ is feasible and we schedule job

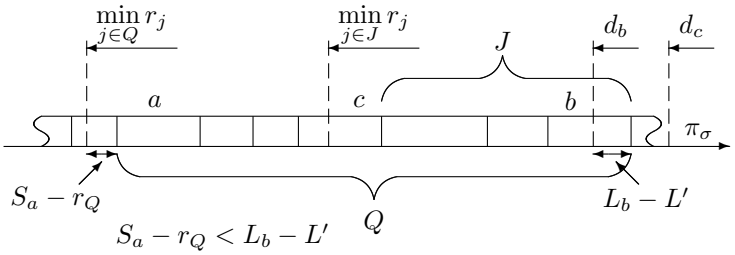


Figure 2.3: The infeasible Schrage schedule: case 2.

c inside J in π', the lateness of job $k \in J \subset N$, if scheduled last among jobs in $J \cup \{c\}$, is

$$L_k(\pi') \geq \min_{j\in J} r_j + p_c + \sum_{j\in J} p_j - d_k \geq L_b(\pi_\sigma) > L' \tag{2.17}$$

since $d_k \leq \max_{j\in J} d_j \leq d_b$. (2.17) contradicts the condition that $\max_{j\in N} L_j(\pi') \leq L'$. Therefore, in a feasible schedule π', job c is processed either before all the jobs in J or after. □

Theorem 2.5 shows that if, at a node of the search tree the Schrage schedule is infeasible and case 2 of Theorem 2.5 holds, two child nodes can be created. At the first node, job c is processed after all jobs in J, and at the second node, job c is processed before all jobs in J. Notice that, because of the way the Schrage schedule is constructed, it suffices to change the due date of job c to $r_c = r_J + p_J$ in the former case and to change the release date of job c to $d_c = d_b - p_J$ in the latter case.

Note that this branching rule guarantees that the algorithm terminates. Each time we branch, at least one precedence relation is added to both the child nodes. As there can be at most $n(n-1)/2$ precedence relations, the depth of the enumeration tree is at most $n(n-1)/2$. Therefore, the number of nodes is finite.

If case 1 of Theorem 2.5 holds, we have an infeasible subset Q of jobs. In this case, the parent node receives set Q from the current node, and the latter is pruned. Let us now consider a node of the search tree which is not a leaf node, i.e. a node at which case 1 of Theorem 2.5 holds. From its child nodes it receives two infeasible subsets of jobs. The following Theorem 2.6 shows how an infeasible subset of jobs is obtained for a non-leaf node. In the theorem, we assume that the constant L' is given.

Theorem 2.6. *Consider a node φ in the search tree of the algorithm. Suppose that the case 2 of Theorem 2.5 holds true. If $Q_a \subset N$ is an infeasible subset of jobs at the child node φ_a where c is scheduled before all the jobs in J, and $Q_b \subset N$ is an infeasible subset of jobs at the child node φ_b where c is scheduled after all the jobs in J, then:*

1. *if $c \notin Q_a$ then Q_a is infeasible at the node φ;*

2. *if $c \notin Q_b$ then Q_b is infeasible at the node φ;*

3. *if $c \in Q_a$ and $c \in Q_b$ then $Q = J \cup Q_a \cup Q_b$ is infeasible at the node φ.*

Proof. 1. If $c \notin Q_a$, then the parameters of the jobs in Q_a are the same at nodes φ and φ_a. Since at the node φ_a set Q_a is infeasible, it is also infeasible at the node φ.

2. The proof is similar to case 1.

3. The proof is by contradiction. Let $\pi \in \Pi(Q)$ be a feasible schedule. Since $c \in Q$ and $J \subset Q$, there are three possible ways of scheduling job c in π. First, c can be scheduled before J in π, but in this case $Q_a \subset Q$ is infeasible. Second, c can be scheduled after J in π, but in this case subset $Q_b \subset Q$ is infeasible. Therefore, c should be scheduled inside J in π, but this contradicts case 2 of Theorem 2.5. Thus, there is no feasible schedule $\pi \in \Pi(Q)$ and the set Q of jobs is infeasible at the node φ. □

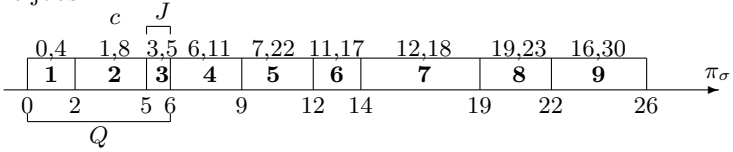
Remark. Suppose that a node φ of the search tree received subset Q_a from its first child node φ_a and $c \notin Q_a$. Then it is not necessary to explore the other child node φ_b, since by Theorem 2.6, it is already known that Q_a, and thus N, is infeasible at the node φ.

From the last two theorems it follows that, if the set N of jobs is infeasible, at all nodes of the search tree, we are able to determine an infeasible subset of jobs. Therefore, the top node gives an infeasible subset of jobs for the given instance.

The proposed algorithm uses a backtrack search strategy and can be implemented recursively. To find lower bounds, Lemma 2.4 is used. Notice that, if $h(Q) > L'$, then Q is infeasible with respect to constant L'. The modified Carlier algorithm is presented in Algorithm 2.1. It returns either a feasible schedule π^*, or an infeasible subset Q' of jobs.

Example 2.1. We now apply the modified Carlier algorithm to a 9-job instance. The parameters of the jobs are shown in Table 2.1. Constant L' is equal to 0. For each node φ we show an illustration of the Schrage schedule π_σ. Job numbers are indicated inside rectangles-jobs. For each job j, the two numbers shown above the job rectangle indicate its current release date $r_j^{(\varphi)}$ and its current due date $d_j^{(\varphi)}$. Numbers shown below are the completion times $C_j(\pi_\sigma)$ of the jobs.

Node 1.



Top node. $c = 2$ and $J = \{3\}$. The first child node 2 has $r_2^{(2)} = 4$.

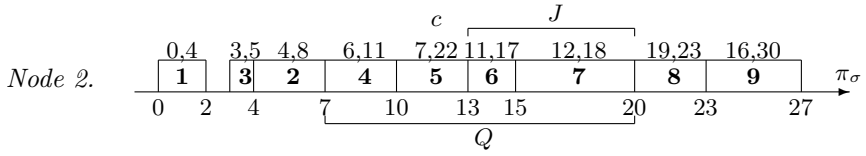
Algorithm 2.1 The modified Carlier algorithm

MOD_CARLIER(N, Q');
return Q'.

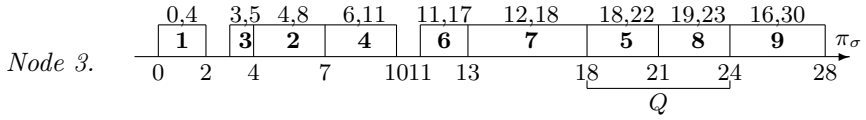
procedure MOD_CARLIER(N, Q')
construct the Schrage schedule $\pi_\sigma \in \Pi(N)$;
renumber jobs in the order they are sequenced in π_σ;
if $\max_{j \in N} L_j(\pi_\sigma) \leq L'$ **then**
  return π_σ; **end algorithm**;
end if
find the smallest number b such that $L_b(\pi_\sigma) > L'$;
find the largest number a, such that $a < b$ and
$S_a(\pi_\sigma) - \min_{j \in Q} r_j < L_b(\pi_\sigma) - L'$, where $Q = \{a, \ldots, b\}$;
if there is no job $c \in Q$ with $d_c > d_b$ **then**
  $Q' := Q$;
else
  find the largest number $c \in Q$ with $d_c > d_b$; $J := \{c+1, \ldots, b\}$;
  if $h(J) = r_J + p_J - d_b > L'$ **then**
    $Q' := J$;
  else if $h(J \cup \{c\}) = r_c + p_c + p_J - d_c > L'$ **then**
    $Q' := J \cup \{c\}$;
  else
    call **MOD_CARLIER**(N, Q_a), where $r_c = r_J + p_J$;
    if $c \in Q_a$ **then**
      call **MOD_CARLIER**(N, Q_b), where $d_c = d_b - p_J$;
    end if
    if $c \notin Q_a$ **then**
      $Q' := Q_a$;
    else if $c \notin Q_b$ **then**
      $Q' := Q_b$;
    else
      $Q' := J \cup Q_a \cup Q_b$;
    end if
  end if
end if
return Q'
end procedure

j	1	2	3	4	5	6	7	8	9
r_j	0	1	3	6	7	11	12	19	16
p_j	2	3	1	3	3	2	5	3	4
d_j	4	8	5	11	22	17	18	23	30

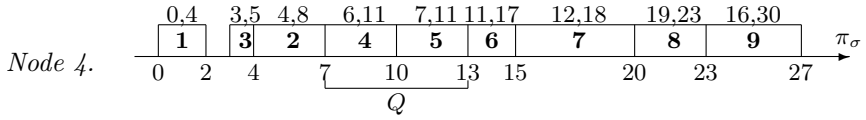
Table 2.1: The data for the 9-job example instance



$c = 5$, $J = \{6, 7\}$. The first child node 3 has $r_5^{(3)} = 18$. The second child node 4 has $d_5^{(4)} = 11$.



Since the Schrage schedule at node 3 does not have job c (Theorem 2.5, case 1), node 3 returns to node 2 $Q' := Q = \{5, 8\}$.



Since the Schrage schedule at node 4 does not have job c, node 4 returns to node 2 $Q' := Q = \{4, 5\}$.

Node 2 receives set $Q_a = \{5, 8\}$ from node 3, and it receives set $Q_b = \{4, 5\}$ from node 4. Since $c = 5 \in Q_a$ and $c = 5 \in Q_b$, node 2 returns to the top node 1 set $Q' := J \cup Q_a \cup Q_b = \{4, \ldots, 8\}$.

The top node receives from node 2 set $Q_a = \{4, \ldots, 8\}$. Since $c \notin Q_a$, the second child node is not explored. The top node returns set $Q' := Q_a = \{4, \ldots, 8\}$, therefore this set is infeasible for our example instance. In this case set Q' is also a *minimal* infeasible subset. In other words, removing any job from Q' makes it feasible. The search tree of the algorithm is depicted in Figure 2.4.

2.3.2 Propagation-Based Cuts

In this section, we present an alternative method for generating infeasibility cuts. The inequalities that form the relaxation $\bar{\mathcal{X}}(r, p, d)$ can be generalized in the following manner. These inequalities $\epsilon(r_i, d_j) \cdot x \leq d_j - r_i$ are associated with pairs of release and due dates (r_i, d_j), $i, j \in N$. By the propagation of the constraint `disjunctive`, we can tighten the time windows of jobs and obtain new values of the release and due dates. Note that the time windows can be tightened only when some other jobs are scheduled on-time. The idea is to use the constraints associated with the new values of the release and due dates. We will call these constraints *propagation-based*.

We now consider the case when the release date r_k of some job k can be increased to r_k^Ω by propagation if all the jobs in some set Ω are scheduled on-time. For instance, this can be done using the rule (1.27). The case of decreasing the due date d_k is completely symmetric and is not presented

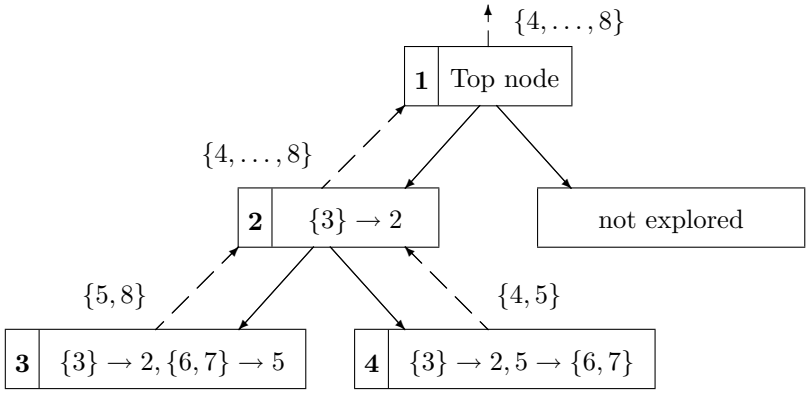


Figure 2.4: The search tree of the algorithm for the example 9-job instance

here. Also we do not consider the case when both release and due dates can be adjusted.

An example of a propagation-based inequality is illustrated in Figure 2.5. This inequality is associated with a set $\Omega \subseteq N$ of jobs and two jobs $k \in N \setminus \Omega$ and $j \in N$. In the figure, $\alpha_l(r_k^\Omega) = (r_k^\Omega - r_l)^+$ and $\beta_l(d_j) = (d_l - d_j)^+$. Again, the length of the black area corresponds to the part of the processing time p_l which must be processed within the interval $[r_k^\Omega, d_j]$ when job l and all jobs in Ω are on-time. The length of the hatched area corresponds to the part of processing time which can be executed outside the interval. When all the jobs in the set Ω are processed on-time, the total length of the filled areas of on-time jobs should not exceed the length of the interval.

Proposition 2.7 presents and justifies the propagation-based inequalities. Let $a_l(E)$ be the coefficient of variable x_l, $l \in N$, in inequality E in the left-hand side and let $b(E)$ be the value of the right-hand side. Then inequality E can be written as

$$\sum_{l \in N} a_l(E) x_l \leq b(E).$$

Let also $A_x(E)$ be the value of the left-hand side in inequality E for solution x, i.e.,

$$A_x(E) = \sum_{l \in N} a_l(E) x_l.$$

Proposition 2.7. *Consider a subset $\Omega \subset N$ of jobs and a job $k \in N \setminus \Omega$ such that, if all the jobs in Ω are on-time, then k cannot start before $r_k^\Omega > r_k$. Then, for each job $j \in N \setminus \Omega$ satisfying $d_j > r_k^\Omega$, the inequality*

$$\begin{aligned} \sum_{l \in N \setminus \Omega \setminus \{k\}} \epsilon_l(r_k^\Omega, d_j) \cdot x_l + (p_k - \beta_k(d_j))^+ x_k \leq \\ d_j - r_k^\Omega + (r_k^\Omega - r_k)(\mid \Omega \mid - \sum_{o \in \Omega} x_o) \end{aligned} \tag{2.18}$$

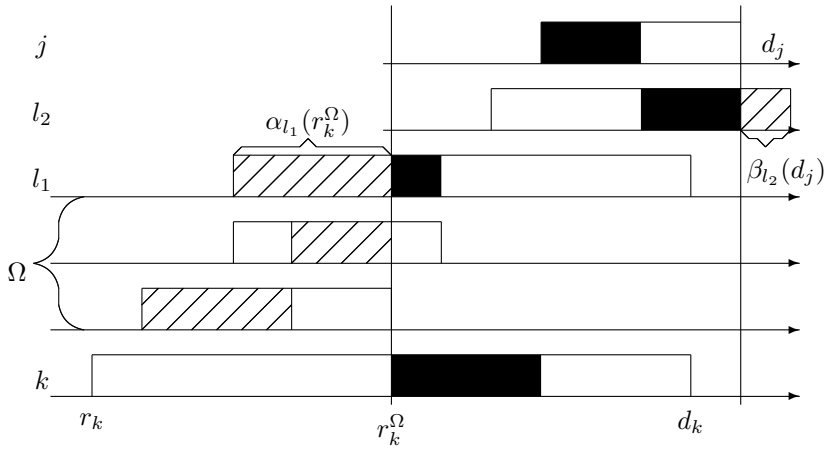


Figure 2.5: A propagation-based inequality

is valid for $\Delta(r, p, d)$.

Proof. Consider the inequality E of type (2.18) for set Ω, job k and some job $j \in N \setminus \Omega$ satisfying $d_j > r_k^\Omega$.

Consider a feasible schedule with $\sum_{o\in\Omega} x_o \leq \mid \Omega \mid -1$. Let E' be the inequality (2.10) associated with the interval $[r_k, d_j]$. Inequality E' is valid by Proposition 2.2, thus $\sum_{l\in N} a_l(E')x_l \leq b(E')$ for any feasible solution x. To prove this case we show that $b(E') \leq b(E)$ and $a_l(E) \leq a_l(E')$, $\forall l \in N$.

We have $b(E) \geq d_j - r_k = b(E')$. From $r_k^\Omega \geq r_k$, it follows that $\alpha_l(r_k^\Omega) \geq \alpha_l(r_k)$ and $a_l(E) \leq a_l(E')$, $\forall l \in N\setminus\Omega\setminus\{k\}$. Then, $a_l(E) = 0 \leq a_l(E')$, $\forall l \in \Omega$. Finally, $a_k(E) = a_k(E')$, as $\epsilon_k(r_k, d_j) = (p_k - \beta_k(d_j))^+$, and $a_l(E) \leq a_l(E')$, $\forall l \in N$.

Suppose now that $\sum_{o\in\Omega} x_o = \mid \Omega \mid$. Then the release date of job k can be set to r_k^Ω. Let E'' be the inequality (2.10) associated with interval $[r_k^\Omega, d_j]$. The idea of the proof is the same as in the previous case.

We have $b(E) = b(E'')$ and $a_l(E) = a_l(E'')$ $\forall l \in N \setminus \Omega$. Then $a_l(E) = 0 \leq a_l(E'')$ $\forall l \in \Omega$. Again, $a_l(E) \leq a_l(E'')$ $\forall l \in N$. □

Since the number of inequalities (2.18) can be exponential, we cannot include them all in the master problem. We propose to use these constraints as infeasibility cuts. Given an integer solution $\bar{x}$, the aim is to find an inequality (2.18), which is violated by $\bar{x}$. In other words, a separation algorithm is needed.

A straightforward idea for the separation algorithm for a given integer solution $\bar{x}$ is the following. We use a filtering algorithm, which performs adjustments of release dates of the jobs. Whenever the propagation algorithm adjusts the release date of some job $k \in N$ due to the presence of some set

$\Omega \subset N$, we check whether $\bar{x}$ violates an inequality (2.18) associated with the set Ω, job k and some job $j \in N \setminus \Omega$. There are $O(n)$ inequalities which have to be verified, since we need to consider all jobs in $N \setminus \Omega$. Verification of each inequality consists of $O(n)$ operations. As a result, the checking procedure has total complexity $O(n^2)$. Since the complexity of the most common filtering algorithms for the constraint `disjunctive` is $O(n^2)$ [7], the proposed straightforward separation algorithm has complexity $O(n^4)$.

2.3.3 Separation Algorithm for Edge-Finding-Based Cuts

We now introduce *Edge-Finding-based* inequalities. They form a subclass of the propagation-based inequalities. A constraint (2.18) is Edge-Finding-based if the propagation is carried out using the Edge-Finding technique. If the given integer solution $\bar{x}$ satisfies all the constraints which form the relaxation $\bar{\mathcal{X}}(r, p, d)$, it is possible to separate $\bar{x}$ with respect to the Edge-Finding-based inequalities in $O(n^3)$ time.

Here we present in detail the separation algorithm for the Edge-Finding-based inequalities. This separation algorithm is based on the filtering algorithm in [7, p.24], which adjusts release dates according to the Edge-Finding rule (1.27). The variant of the separation algorithm based on the adjustment of due dates is completely symmetric and is not covered here.

Assume that $\bar{x}$ satisfies constraints

$$\sum_{j \in N} \epsilon(r_i, d_j) \cdot x \leq d_j - r_i, \quad \forall i, j \in N, r_i < d_j, \tag{2.19}$$

which form the relaxation $\bar{\mathcal{X}}(r, p, d)$. The following three propositions describe cases when, if $\bar{x}$ violates an Edge-Finding-based inequality, then it must violate at least one inequality (2.19). Therefore, given the assumption, these three cases can be excluded from consideration.

Remember that $\bar{J} = \{j \in N : \bar{x}_j = 1\}$. Proposition 2.8 shows that, if an Edge-Finding-based inequality E is violated by $\bar{x}$, E can be associated only with a set $\Omega \subset \bar{J}$.

Proposition 2.8. *If a vector $\bar{x} \in \{0,1\}^n$ violates the Edge-Finding-based inequality E associated with $\Omega \subset N$, $k, j \in N \setminus \Omega$, and there exists a job $l \in \Omega$ such that $\bar{x}_l = 0$, then $\bar{x}$ also violates at least one inequality (2.19).*

Proof. Let E' be the inequality of type (2.19) that is associated with the interval $[r_k, d_j]$. Since E is violated by $\bar{x}$, we have $A_{\bar{x}}(E) > b(E)$. To prove the proposition we show that the right-hand side of inequality E is greater than or equal to the right-hand side of inequality E', and that the left-hand side of E is less then or equal to the left-hand side of inequality E'.

We have $\sum_{o \in \Omega} \bar{x}_o \leq \mid \Omega \mid -1$, thus $b(E) \geq d_j - r_k = b(E')$. From $r_k^\Omega > r_k$, it follows that $\alpha_l(r_k^\Omega) \geq \alpha_l(r_k)$ and $a_l(E) \leq a_l(E')\ \forall l \in N \setminus \Omega \setminus \{k\}$. Then $a_l(E) = 0 \leq a_l(E')\ \forall l \in \Omega$. Finally, $a_k(E) = a_k(E')$, and $A_{\bar{x}}(E) \leq A_{\bar{x}}(E')$. □

Proposition 2.9 shows that if an Edge-Finding-based inequality E is violated by $\bar{x}$, it must be associated with jobs $k, j \in N \setminus \Omega$, such that $d_j > d_k - p_k$.

Proposition 2.9. *If a vector $\bar{x} \in \{0,1\}^n$ violates the Edge-Finding-based inequality E associated with $\Omega \subset N$, $k, j \in N \setminus \Omega$, and $d_j \leq d_k - p_k$, then $\bar{x}$ also violates at least one inequality (2.19).*

Proof. If $\sum_{o \in \Omega} \bar{x}_o < |\ \Omega\ |$, the proof follows from Proposition 2.8. Let $\sum_{o \in \Omega} \bar{x}_o = |\ \Omega\ |$. We consider three cases illustrated in Figures 2.6-2.8.

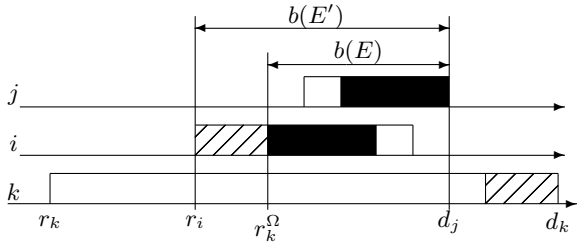


Figure 2.6: Proposition 2.9, case 1

1. There exists a job $i \in N \setminus \Omega$, with $r_i < r_k^\Omega$, $r_i + p_i > r_k^\Omega$, $d_i \leq d_j$, and $\bar{x}_i = 1$. Let E' be the inequality of type (2.19) associated with the interval $[r_i, d_j]$. We have $b(E') = b(E) + (r_k^\Omega - r_i)$. To prove case 1 we show that $A_{\bar{x}}(E')$ is bigger than $A_{\bar{x}}(E)$ by at least $r_k^\Omega - r_i$.

From $r_i < r_k^\Omega$, it follows that $\alpha_i(r_k) \leq \alpha_l(r_k^\Omega)$ and $a_l(E) \leq a_l(E')\ \forall l \in N \setminus \Omega \setminus \{k\}$. Then $a_k(E) = 0 = a_k(E')$, since $d_j \leq d_k - p_k$, and $a_l(E) = 0 \leq a_l(E')\ \forall l \in \Omega$. Moreover, from $d_i \leq d_j$, it follows that $a_i(E) = p_i - (r_k^\Omega - r_i)$ and $a_i(E') = p_i$. Therefore, $a_i(E) - a_i(E') = r_k^\Omega - r_i$ and $A_{\bar{x}}(E') - A_{\bar{x}}(E) \geq r_k^\Omega - r_i$.

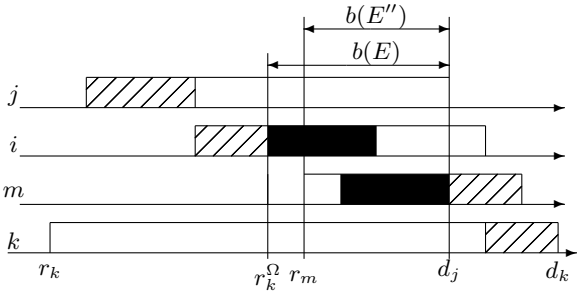


Figure 2.7: Proposition 2.9, case 2

2. Case 1 does not hold and there exists at most one job $i \in N \setminus \Omega$, with $r_i < r_k^\Omega$, $r_i + p_i > r_k^\Omega$, $d_i - p_i < d_j < d_i$, and $\bar{x}_i = 1$. Let $m \in N \setminus \Omega$ be the job with smallest release date, such that $r_m \geq r_k^\Omega$, $r_m < d_j$, $a_m(E) > 0$, and $\bar{x}_m = 1$. Such a job should exist, otherwise E would not be violated by $\bar{x}$,

since at most one job i with $\bar{x}_i = 1$ would have nonzero coefficient in E (and no coefficient can be bigger than $b(E)$). Let E'' be the inequality of type (2.19) associated with the interval $[r_m, d_j]$. We have $b(E'') = b(E) - (r_m - r_k^\Omega)$. To prove case 2 we show that $A_{\bar{x}}(E'')$ is smaller than $A_{\bar{x}}(E)$ by at most $r_m - r_k^\Omega$.

We have $r_l + p_l < r_k^\Omega$ or $r_l \geq r_m$ for all jobs in N except at most one job i, thus $a_l(E'') = a_l(E)\ \forall j \in N \setminus \{i\}$. From $\alpha_i(r_m) - \alpha_i(r_k^\Omega) = r_m - r_k^\Omega$, it follows that $a_i(E'') - a_i(E) \leq r_m - r_k^\Omega$ and $A_{\bar{x}}(E'') - A_{\bar{x}}(E) \leq r_m - r_k^\Omega$.

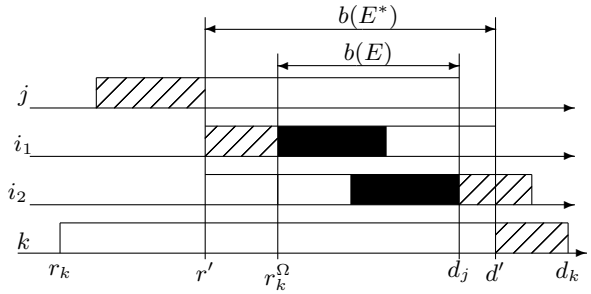


Figure 2.8: Proposition 2.9, case 3

3. Case 1 does not hold and there exist at least two jobs $i_1, i_2 \in N \setminus \Omega$, with $r_{i_t} < r_k^\Omega$, $r_{i_t} + p_{i_t} > r_k^\Omega$, $d_{i_t} - p_{i_t} < d_j$, and $\bar{x}_{i_t} = 1$ for $t = 1, 2$. As case 1 does not hold, we have $d_{i_1} > d_j$ and $d_{i_2} > d_j$. Without loss of generality let $d' = d_{i_1} \leq d_{i_2}$ and $r' = \max\{r_{i_1}, r_{i_2}\}$. Let E^* be the inequality of type (2.19), associated with the interval $[r', d']$. We have $b(E^*) = b(E) + (r_k^\Omega - r') + (d' - d_j)$. To prove case 3 we will show that $A_{\bar{x}}(E^*)$ is bigger than $A_{\bar{x}}(E)$ by at least $(r_k^\Omega - r') + (d' - d_j)$.

From $r' < r_k^\Omega$ and $d_j < d'$, it follows that $a_l(E^*) \geq a_l(E)\ \forall l \in N$. First let $r' = r_{i_2}$. Then $\beta_{i_1}(d_{i_1}) = 0$, $\alpha_{i_1}(r_k^\Omega) - \alpha_{i_1}(r_{i_2}) = r_k^\Omega - r'$, and $a_{i_1}(E^*) - a_{i_1}(E) \geq r_k^\Omega - r'$. Also, $\beta_{i_2}(d_{i_2}) = 0$, $\beta_{i_2}(d_j) - \beta_{i_2}(d_{i_1}) = d' - d_j$, and $a_{i_2}(E^*) - a_{i_2}(E) \geq d' - d_j$.

Now let $r' = r_{i_1}$. Then $a_{i_1}(E^*) = p_{i_1}$ and $a_{i_1}(E^*) - a_{i_1}(E) = \max\{r_k^\Omega - r', d' - d_j\}$. Also, $\alpha_{i_2}(r_k^\Omega) - \alpha_{i_2}(r_{i_1}) = r_k^\Omega - r'$, $\beta_{i_2}(d_j) - \beta_{i_2}(d_{i_1}) = d' - d_j$. Thus $a_{i_2}(E^*) - a_{i_2}(E) \geq \min\{r_k^\Omega - r', d' - d_j\}$. Therefore $A_{\bar{x}}(E^*) - A_{\bar{x}}(E) \geq (r_k^\Omega - r') + (d' - d_j)$. □

Let $e_Q = r_Q + p_Q$, for any set Q of jobs. Proposition 2.10 shows that if an Edge-Finding-based inequality E violates $\bar{x}$, then E can be associated only with jobs $k, j \in N \setminus \Omega$, such that there is no subset $J' \subseteq \bar{J}$ such that $d_{J'} \leq \max\{d_j, d_k\}$, $e_{J'} \geq r_k^\Omega$, $r_{J'} \leq r_k$, and $e_{J''} \leq e_{J'}$, $\forall J'' \subset J'$. Proposition 2.10 is illustrated in Figure 2.9.

Proposition 2.10. *If a vector $\bar{x} \in \{0,1\}^n$ violates the Edge-Finding-based inequality E associated with $\Omega \subset N$, $k, j \in N \setminus \Omega$, and there exists a subset $J' \subseteq \bar{J}$ such that $d_{J'} \leq \max\{d_j, d_k\}$, $e_{J'} \geq r_k^\Omega$, $r_{J'} \leq r_k$, and $e_{J''} \leq e_{J'}$, $\forall J'' \subset J'$, then $\bar{x}$ violates at least one inequality (2.19).*

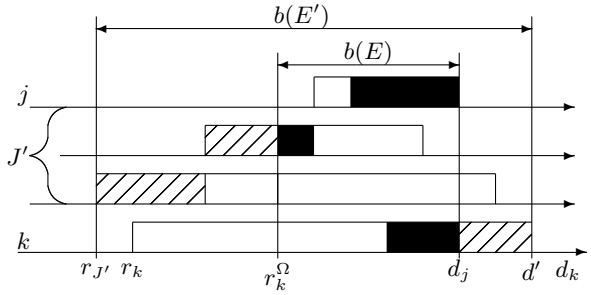


Figure 2.9: Proposition 2.10

Proof. If $\sum_{o\in\Omega}\bar{x}_o <\mid \Omega \mid$, the proof follows from Proposition 2.8. Let $\sum_{o\in\Omega}\bar{x}_o = \mid \Omega \mid$. Suppose $d_j > d_k - p_k$, since otherwise the proof follows from Proposition 2.9. Let $d' = \max\{d_k, d_j\}$.

Let E' be the inequality of type (2.19), associated with the interval $[r_{J'}, d']$. We have $b(E') = b(E) + (r_k^\Omega - r_{J'}) + (d' - d_j)$. To prove the proposition we show that $A_{\bar{x}}(E')$ is bigger than $A_{\bar{x}}(E)$ by at least $r_k^\Omega - r_{J'} + (d' - d_j)$.

From $d' \geq d_{J'}$ it follows that $a_l(E) \leq (p_l - \alpha_l(r_k^\Omega))^+$, $a_l(E') = p_l \ \forall l \in J'$. Also, $a_k(E') - a_k(E) = d' - d_j$. Then, from $a_l(E) \leq a_l(E') \ \forall l \in N$, we have $A_{\bar{x}}(E') - A_{\bar{x}}(E) \geq p_{J'} - \sum_{l\in J'}(p_l - \alpha_l(r_k^\Omega))^+ + (d' - d_j)$. Therefore, it suffices to show that

$$\sum_{l\in J'}(p_l - \alpha_l(r_k^\Omega))^+ \leq p_{J'} - (r_k^\Omega - r_{J'}). \tag{2.20}$$

The proof is by contradiction. Let $J'' = \{l \in J' : p_l - \alpha_l(r_k^\Omega) \geq 0\}$ and $r_{l'} = r_{J''}$. Assume that (2.20) does not hold. Then

$$e_{J'} = r_{J'} + p_{J'} < \sum_{l\in J''}(p_l - \alpha_l(r_k^\Omega)) + r_k^\Omega = p_{l'} - (r_k^\Omega - r_{l'})^+ + r_k^\Omega +$$

$$\sum_{l\in J''\setminus\{l'\}}(p_l - (r_k^\Omega - r_{l'})^+) \leq r_{l'} + p_{l'} + \sum_{l\in J''\setminus\{l'\}} p_l = r_{J''} + p_{J''} = e_{J''},$$

contradicting the assumption of the proposition. □

The separation algorithm for Edge-Finding based inequalities is presented in Algorithm 2.2. Remember that $\bar{x}$ satisfies all the constraints (2.19). For the algorithm we assume that the input set $\bar{J} = \{j \in N \ : \bar{x}_j = 1\}$ of jobs is ordered by nondecreasing release dates.

Note first that to check the condition (1.27) for every possible set Ω of jobs, it suffices to check only sets $\Omega_{[r_i,d_l]} = \{j \in \bar{J} : \ r_j \geq r_i, d_j \leq d_l\}$ for all $i, l \in \bar{J}$, $r_i \leq d_l$.

In the l-th outer iteration we consider two subsets $\Omega_\leq = \{i \in \bar{J} : d_i \leq d_l\}$ and $\Omega_> = \{i \in \bar{J} : d_i > d_l\}$. Let $C = \max_{\Omega'\subseteq\Omega_\leq}\{r_{\Omega'} + p_{\Omega'}\}$. Set Ω will be

Algorithm 2.2 The separation algorithm for Edge-Finding-based inequalities

for $l := 1$ to $\mid \bar{J} \mid$ **do**
  $\Omega_{\leq} := \{i \in \bar{J} : d_i \leq d_l\}$;
  $\Omega_{>} := \{i \in \bar{J} : d_i > d_l\}$;
  $P := p_{\Omega_{\leq}}$; $C := \max_{\Omega' \subseteq \Omega_{\leq}} \{r_{\Omega'} + p_{\Omega'}\}$;
  $\alpha_i(C) := (C - r_i)^+ \ \forall i \in \Omega_{>}$;
  $\bar{d} := \{\}$; $\bar{d} \leftarrow \{d_i; d'_i = d_i - \max\{p_i, \alpha_i(C)\}; d''_i := d_i - p_i\} \ \forall i \in \Omega_{>}$;
  sort $\bar{d}$; $H := -\infty$;
  for $k := 1$ to $\mid \bar{J} \mid$ **do**
    if $d_k \leq d_l$ **then**
      if $H < r_k + P$ **then**
        $h := k$; $H := r_k + P$;
      end if
      $P := P - p_k$
    else
      if $(r_k + p_k + P > d_l$ **or** $H + p_k > d_l)$ **and** $H < C$ **then**
        if $r_k + p_k + P > d_l$ **then**
          $\Omega := \{k + 1, \ldots, n\} \cap \Omega_{\leq}$;
        else
          $\Omega := \{h, \ldots, n\} \cap \Omega_{\leq}$;
        end if
        set d'_k to d_k in $\bar{d}$; sort $\bar{d}$;
        CHECK$(\Omega, k, C, \alpha(C), \bar{d})$
        set d'_k to $d_k - \max\{p_k, \alpha_k(C)\}$ in $\bar{d}$; sort $\bar{d}$;
      end if
    end if
  end for
end for

procedure CHECK$(\Omega, k, C, \alpha(C), \bar{d})$
$z := 0$; $s := \sum_{i \in \Omega_{>} \setminus \{k\}} (p_i - \alpha_i(C)) + p_k$; i is the maximum index in $\bar{d}$;
while $i > 1$ **and** $\bar{d}_i > d_k - p_k$ **do**
  if $s > \bar{d}_i - C$ **then**
    $\bar{x}$ violates the Edge-Finding-based inequality associated with set Ω and jobs k, j, $d_j = \bar{d}_i$; **end algorithm**;
  end if
  $i := i - 1$; $s := s - z \cdot (\bar{d}_{i+1} - \bar{d}_i)$;
  if $\bar{d}_i$ is some d'_j **then**
    $z := z + 1$;
  else if $\bar{d}_i$ is some d''_j **then**
    $z := z - 1$;
  end if
end while
end procedure

sought among the sets $\Omega_{\leq,i} = \{j \in \Omega_{\leq} : r_j \geq r_i\} = \Omega_{[r_i,d_l]}$, $\forall i \in N$, i.e for each job $k \in \Omega_{>}$ and each $i \in \Omega_{\leq}$, we will check whether

$$d_\Omega - r_{\Omega\cup\{k\}} < p_\Omega + p_k, \quad \text{where } \Omega = \Omega_{\leq,i}. \tag{2.21}$$

By the Edge-Finding rule (1.27), if (2.21) holds, then we can adjust the release date of job k to $r_k^\Omega = \max_{\Omega'\subset\Omega}\{r_{\Omega'} + p_{\Omega'}\}$. Note that r_k^Ω cannot be more than C, because $\Omega \subseteq \Omega_{\leq}$.

We will show now that, if (2.21) holds and $\bar{x}$ violates the inequality of type (2.18) associated with pair $(\Omega_{\leq,i}, k)$, then $r_k^\Omega = C$, i.e. $r_k^\Omega = C$ does not depend on k. This means that r_k^Ω can be computed outside the inner loop, which decreases the complexity of the separation algorithm. Note that the value of C can be computed in linear time by considering sets $\Omega_{\leq,i}$ for all $i \in \Omega_{\leq}$.

In the beginning of the k-th iteration of the inner loop in line 15, if $d_k > d_l$, we have:

$$P = p_{\Omega_{\leq,k}}, \qquad H = \max_{\Omega_{\leq,k}\subseteq\Omega'\subseteq\Omega_{\leq}}\{r_{\Omega'} + p_{\Omega'}\} = \max_{i:\, i<k}\{r_i + p_{\Omega_{\leq,i}}\}.$$

Let the maximum be achieved when $i = h$, i.e. $H = r_h + p_{\Omega_{\leq,h}} = e_{\Omega_{\leq,h}}$. Suppose that (2.21) holds and $H = C$. Remember that $r_k^\Omega \leq C$. Let $J' = \Omega_{\leq,h}$, then $d_{J'} < d_k$, $e_{J'} = H = C \geq r_k^\Omega$, $r_{J'} = r_h \leq r_k$ (as $h < k$), $e_{J''} \leq e_{J'} = C$ for all $J'' \subset J'$ (as $J'' \subset \Omega_{\leq}$). Therefore by Proposition 2.10, $\bar{x}$ should violate at least one inequality (2.19) — contradiction. Hence, if (2.21) holds, then $H < C$.

In line 15, we check the condition (2.21) that is equivalent to $r_k + p_k + P > d_l$ for $\Omega = \Omega_{\leq,k}$ and to $H + p_k > d_l$ for $\Omega = \Omega_{\leq,h}$. If one of these two inequalities holds, we obtain the set Ω in lines 17 ($\Omega := \Omega_{\leq,k}$) or 19 ($\Omega := \Omega_{\leq,h}$). Let $\Omega_{\leq,i} = \arg\max_{\Omega'\subseteq\Omega_{\leq}}\{e_{\Omega'}\}$. As $H < C$, we have $i > k$ and $\Omega_{\leq,i} \subseteq \Omega_{\leq,k} \subseteq \Omega_{\leq,h}$. Therefore, in both cases $r_k^\Omega = C$. See an illustration in Figure 2.10.

Having the pair (Ω, k), we need to check if there exists a violated Edge-Finding-based inequality associated with Ω, k and some job $j \in N$. Now we show how to do this in linear time.

We know before the inner loop in lines 8-26 that, if the "Edge-Finding" condition holds, then $r_k^\Omega = C$, $\forall k \in \Omega_{>}$. Thus, in any violated Edge-Finding-based inequality E, $a_i(E) = 0$ for all $i \in \Omega_{\leq}$ as $C - r_i > p_i$. Then, before the inner loop we know the values of $\alpha_i(r_k^\Omega)$ for all $i \in \Omega_{>}$, $i \neq k$. These values are stored in line 5 as $\alpha_i(C)$.

Let E' be the Edge-Finding-based inequality associated with Ω, some fixed job k and job $j \in \Omega_{>}$, $d_j = d_{\Omega_{>}}$. Then $a_i(E') = (p_i - \alpha_i(C))^+$, $\forall i \in \Omega_{>} \setminus \{k\}$. In line 30, we set $s := A_{\bar{x}}(E')$. If the right border d_j of the interval $[C, d_{\Omega_{>}}]$ is decreased, the coefficient of a variable x_i, $i \neq k$, remains the same as long as $\alpha_i(C) \geq \beta_i(d_j)$, i.e. while $d_j \geq d_i' := d_i - \max\{p_i, \alpha_i'\}$. Then the coefficient decreases until it becomes equal to zero when $d_j = d_i'' := d_i - p_i$. For the

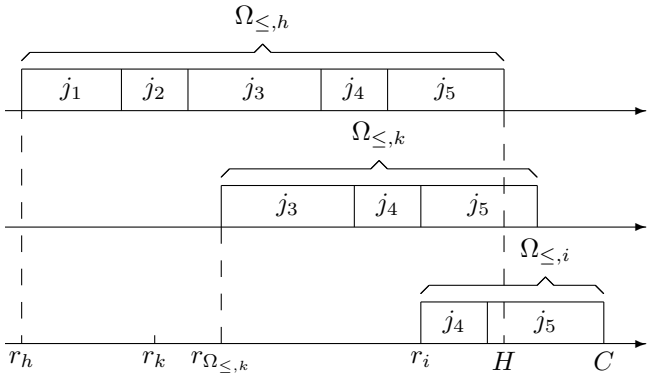


Figure 2.10: Illustration for the separation algorithm

variable x_k, $d'_k = d_k$. In line 6, for each job $i \in \Omega_>$ all three values $\{d_i, d'_i, d''_i\}$ are added to array $\bar{d}$ and $\bar{d}$ is sorted. In line 21, when job k is known, we change the value of d'_k and sort $\bar{d}$. Notice that in this case sorting takes only linear time, as only one value in $\bar{d}$ has been changed. In line 23, $\bar{d}$ is put into the initial state.

Using array $\bar{d}$, the procedure CHECK can check whether there exists a violated inequality in linear time. It just decreases the right border of the interval and adjusts appropriately the sum s of coefficients of variables from $\Omega_>$. If for some right border d_j, the sum s is greater than the length of the interval $[C, d_j]$ then the desired inequality is found.

As the complexity of the procedure CHECK is $O(n)$, the overall complexity of the separation algorithm is $O(n^3)$.

2.4 Branch-and-Check Algorithm

In this subsection we discuss several variants of our branch-and-check algorithm for the problem $1 \mid r_j \mid \sum w_j U_j$. To recall the framework of the branch-and-check method, a flow chart is presented in Figure 2.11. We have contributed to this method on its two main stages, the initialization stage and the cut generation stage. In each variant of our branch-and-check algorithm, cuts are generated differently.

Remember that we have three ways to generate infeasibility cuts. The first usual approach consists in using standard cuts (2.12). To generate these cuts we only need the information whether solution $\bar{x}$ is feasible or not. In the experiments, to check feasibility, the instance of the problem $1 \mid r_j \mid L_{\max}$ with jobs in $\bar{J}$ was solved by Carlier algorithm [20]. We denote this standard approach "bas" (basic).

The second approach was suggested in subsection 2.3.1. Here we generate

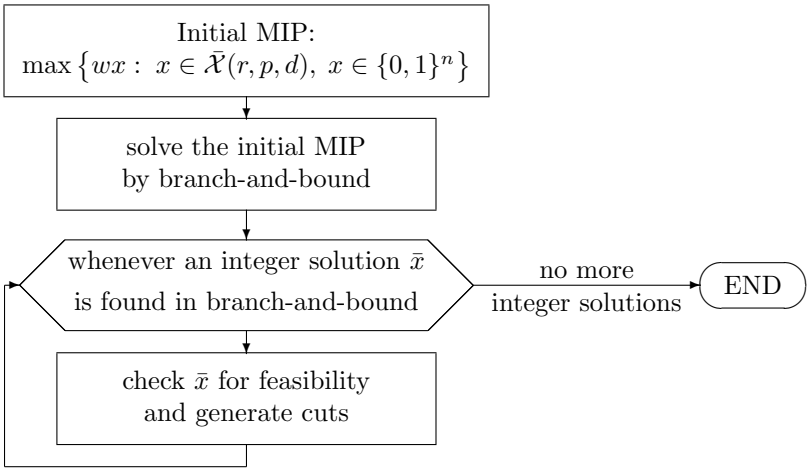


Figure 2.11: The flow chart of the branch-and-check method

the cuts (2.13) using the modified Carlier algorithm. Unfortunately, this algorithm is sometimes time consuming. Therefore, in the experiments, the search tree is truncated after 1000 nodes. The second approach is denoted as "str" (strengthened cuts).

The third approach was discussed in subsection 2.3.2. Here Edge-Finding-based inequalities are used as infeasibility cuts. To generate them, we apply the separation algorithm proposed in subsection 2.3.3. We denote this approach as "ef" (Edge-Finding-based cuts).

Notice that the separation algorithm does not tell us anything about the feasibility of a solution, if a violated Edge-Finding-based inequality is not found. Also, the modified Carlier algorithm cannot determine whether a solution is feasible or not, if it is interrupted after reaching the node limit. Therefore, in all variants of the algorithm we use the first approach to test infeasibility when the other approaches are unsuccessful.

We consider the following five variants of the cuts generation stage in the branch-and-check algorithm: "bas", "str+bas", "ef+str+bas", "ef+bas", "str+ef+bas". Flow charts for these variants are presented in Figure 2.12. For example, the last variant "str+ef+bas" signifies that first the modified Carlier algorithm is applied; then if it is interrupted, we use the separation algorithm; finally, if the latter cannot find a violated Edge-Finding-based inequality, the usual Carlier algorithm is used.

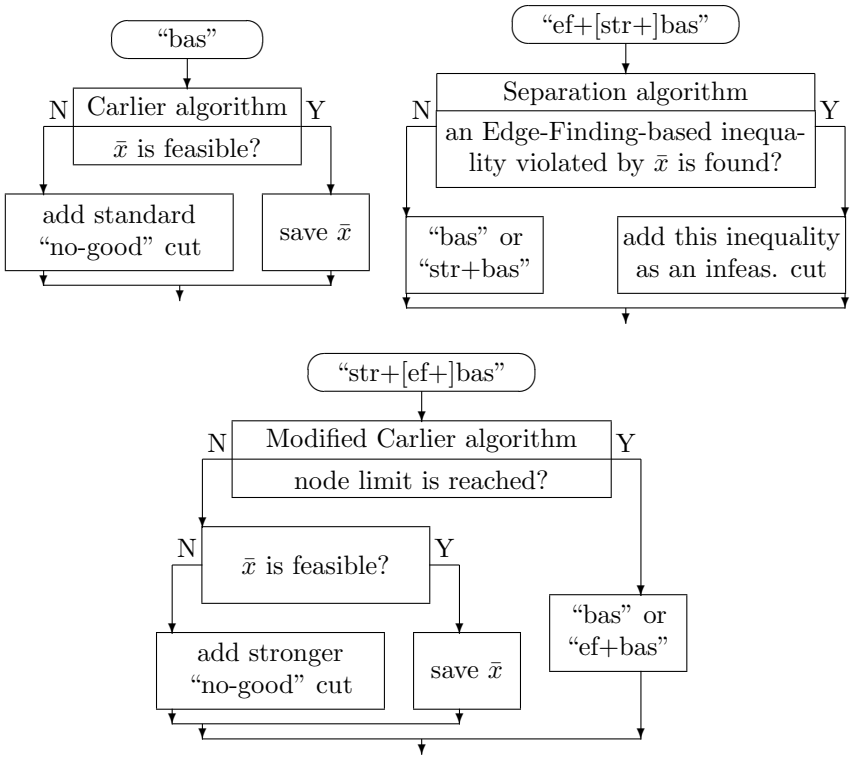


Figure 2.12: Variants of the cuts generation stage in the branch-and-check algorithm

2.5 Numerical Experiments

In this section we report on the results of a computational study of the proposed branch-and-check algorithm. We experiment with the five variants of the algorithm presented in Section 2.4.

For the experiments we used three test sets. The first test set contains the instances from [8]. The data for these instances was derived based on four parameters:

- processing times were uniformly generated in the interval $[p_{\min}, p_{\max}]$;
- weights were uniformly generated in the interval $[1, w_{\max}]$;
- release dates were generated from the normal distribution with parameters $(0, \sigma)$ (negative values generated are discarded), where σ depends on the *load* of the machine (the higher is load, the smaller is σ), see [8] for more details; the load equals the ratio of the sum of all processing

times and the difference between the maximum due date and minimum release date $d_N - r_N$;

- margins ($m_j = d_j - r_j - p_j$) were uniformly generated in the interval $[0, m_{\max}]$.

The values of the parameters were selected from each of the following values:

$(p_{\min}, p_{\max})$	(0,100), (25,75)
$m_{\max}$	50, 200, 350, 500, 650
load	1.0, 1.6, 2.2
$w_{\max}$	1, 10, 100.

For each quadruple of the parameters and for each $n \in \{10, 20, \ldots, 100\}$, there is one instance in the test set. In total there are 90 instances for each dimension. We denote this test set by "b". Instances from it with particular dimension n are denoted by "bn".

The second test set is the instances from [30]. For these instances processing times were uniformly generated in the interval $[0, 100]$. Given the number of jobs n, the remaining data is derived according to three parameters:

- weights were uniformly generated in the interval $[1, w_{\max}]$;
- release dates were uniformly generated in the interval $[0, K_1 n]$;
- due dates were uniformly generated in the interval $[r_j + p_j, r_j + p_j + K_2 n]$.

The parameters used were:

$w_{\max}$	10, 99
K_1	1, 5, 10, 20
K_2	1, 5, 10, 20.

For each triple of the parameters and for each $n \in \{20, 40, \ldots, 140\}$, there are ten instances in the test set. In total there are 320 instances for each dimension. We denote this test set by "s". Instances from it with particular dimension n are denoted by "sn".

The third set of instances have been generated in a similar manner to that of the second test set. The only difference is that the weights are dependent on the processing times. Each w_j, $j \in N$, was generated uniformly in the interval $[\max\{1, p_j - \delta\}, \min\{99, p_j + \delta\}]$, where δ denotes a degree of dependence. We used two values for the parameter δ: 10 and 25. For the parameters K_1 and K_2 the same values 1, 5, 10, 20 as in the previous case were used. For each triple of parameters and for each $n \in \{80, 100, 120\}$ there are ten instances in the test set. In total there are 160 instances for each dimension and each value of δ. We denote this test set by "d". Instances with particular dimension n and δ are denoted by "dδ-n".

In the experiments, we were interested in the following statistics.

P_{1h} — the percentage of instances solved to optimality in one hour.

All other statistics were evaluated only for the instances that were solved within the time limit.

T_{av}, T_{max} — average and maximum time, needed to solve an instance to optimality (in seconds).

N_{av}, C_{av} — average number of nodes and infeasibility cuts in the IP search tree.

$PMCE_{av}$ - average efficiency of the modified Carlier algorithm, i.e. the average percentage difference between the cardinality of the input set $\bar{J}$ of jobs and the cardinality of the infeasible subset $Q \subseteq \bar{J}$ found by the algorithm:

$$PMCE = \frac{\mid \bar{J} \mid - \mid Q \mid}{\mid \bar{J} \mid} \cdot 100\%.$$

We took into account only the cases when the modified Carlier algorithm found an infeasible subset within the node limit.

$PMCL$ — percentage of cases in which the modified Carlier algorithm reached the node limit.

Computational experiments have been carried out on a computer with a 2 GHz Pentium IV processor and 512 Mb RAM. The algorithm have been implemented in the *Mosel* modelling and optimization language [26] with *XPress-MP* (version 15.20) as the IP solver.

	"bas"		"ef+bas"	
Test	P_{1h}	T_{av}	P_{1h}	T_{av}
s40	99.7%	2.5	100%	0.9
s60	98.8%	16.6	99.7%	6.0
s80	95.3%	50.3	98.8%	15.2
s100	-	-	99.4%	88.5
b40	100%	0.8	100%	0.8
b50	98.9%	5.5	100%	3.0
b60	96.7%	38.5	98.9%	5.9
b70	94.4%	79.4	98.9%	27.7
b80	97.8%	91.1	98.9%	25.9
b90	87.8%	137.2	96.7%	41.5

Table 2.2: Results for the first two variants of the algorithm

In Tables 2.2 and 2.3 we present results for the five variants of the algorithm. "-" means, that the corresponding experiment has not been performed. We did not test the variant "str+ef+bas" for the instances "b" because, when

	"ef+str+bas"		"str+bas"		"str+ef+bas"	
Test	P_{1h}	T_{av}	P_{1h}	T_{av}	P_{1h}	T_{av}
s40	100%	0.6	100%	0.6	100%	0.6
s60	100%	3.3	100%	3.5	100%	3.4
s80	100%	15.3	100%	15.4	100%	11.8
s100	99.7%	46.0	99.7%	28.6	100%	33.0
s120	97.8%	132.4	99.7%	115.7	99.7%	109.0
s140	92.5%	127.2	95.9%	189.8	96.3%	183.6
b40	100%	0.6	100%	0.5	-	-
b50	100%	1.7	100%	1.5	-	-
b60	100%	3.3	100%	3.5	-	-
b70	100%	7.0	100%	6.7	-	-
b80	100%	14.5	100%	49.4	-	-
b90	98.9%	36.8	98.9%	26.3	-	-
b100	100%	104.2	100%	91.6	-	-

Table 2.3: Results for the last three variants of the algorithm

solving these instances by the variant "str+bas", the modified Carlier algorithm very rarely reached the node limit. This fact implies that we have the same results for the "str+ef+bas" variant. Taking this observation into account, we conclude that the variant "str+ef+bas" is the best on average for the test instances "s" and "b". The only exception is one of the "b80" instances, which is solved much faster by the "ef+str+bas" variant. Because of this instance, the average statistics for the test set "b80" and the variants "str+bas" and "str+ef+bas" are worse.

If we consider the efficiency of the cuts (2.13) and Edge-Finding-based inequalities (2.18) separately, then from the results in Tables 2.2 it follows that the former are more efficient.

	Algorithm from [63]			Algorithm "str+ef+bas"		
Test	P_{1h}	T_{av}	T_{max}	P_{1h}	T_{av}	T_{max}
b50	100%	15.3	118	100%	1.5	46
b60	100%	49.4	602	100%	3.5	48
b70	100%	97.0	2693	100%	6.7	117
b80	100%	140.6	3081	100%	49.4	3318
b90	93.3%	203.4	2375	98.9%	26.3	311
b100	93.3%	378.8	3568	100%	91.6	1512
s40	100%	8.3	147	100%	0.6	7
s60	99.7%	67.6	1982	100%	3.4	264
s80	98.1%	218.6	2237	100%	11.8	514
s100	94.1%	551.2	3548	100%	33.0	1110

Table 2.4: Comparison of the results

In the next experiment, we compared the results obtained with the best variant "str+ef+bas" of our algorithm with the results of the algorithm presented in the paper by Péridy et al. [63]. The authors of this paper have kindly supplied us with the code of their algorithm. Both algorithms have been tested on the instances "s" and "b" with up to 100 jobs. Note that exactly the same instances were used for both algorithms. The results presented in Table 2.4 reveal that our algorithm outperforms the one from [63]. Note that the latter is the best, to our knowledge, algorithm for the problem $1 \mid r_j \mid \sum w_j U_j$ available in the literature.

	"bas"		"str+ef+bas"			
Test	N_{av}	C_{av}	N_{av}	C_{av}	$PMCE_{\text{av}}$	$PMCL$
b50	396.6	85.6	219.9	10.8	42.9%	0.3%
b60	768.0	184.5	345.8	13.8	46.7%	2.7%
b70	1307.1	273.9	527.5	16.2	57.6%	0.1%
b80	1917.9	252.6	1840.3	85.1	42.1%	0.2%
b90	2358.6	274.4	1300.7	20.3	51.6%	0.4%
b100	-	-	3657.9	60.8	51.5%	0.1%
s40	188.0	30.5	85.3	4.1	34.9%	0.1%
s60	400.1	80.7	267.5	10.3	42.0%	1.0%
s80	1019.3	159.5	501.0	16.3	39.1%	2.2%
s100	-	-	849.8	12.0	48.9%	1.6%
s120	-	-	1907.0	17.3	44.3%	11.7%
s140	-	-	2580.0	21.7	42.2%	9.3%

Table 2.5: Impact of stronger cuts

In Table 2.5 we present additional parameters for the variants "bas" and "str+ef+bas" of the algorithm. It appears that in comparison with "bas", the average number of nodes in "str+ef+bas" is significantly lower. Moreover, on average the number of the strengthened cuts (2.13) generated in "str+ef+bas" is an order of magnitude less, than the number of the cuts (2.12) used in "bas". So the efficiency of the stronger cuts proposed here is quite high. Statistics for the parameter $PMCE_{\text{av}}$ partly explain this efficiency. The strengthened cuts are based on infeasible subsets of jobs where the cardinality is less by 40-50% on average than the cardinality of the initial infeasible sets, on which the standard cuts are based. Notice that the algorithm to derive the strengthened cuts is sufficiently fast, difficulties occur only with the biggest "s" instances, for which the algorithm did not terminate before the node limit in about 10% of the cases. In these cases sometimes we can still add generalized tightening inequalities in the variant "str+ef+bas" in contrast to the variant "str+bas", where only the standard cuts (2.12) are generated. This seems to be the reason why the former variant is superior to the latter.

Results for the test set "d" are shown in Table 2.6. As expected, instances in which the weights and processing times are dependent are harder to solve.

Test	P_{1h}	T_{av}	N_{av}	C_{av}
s80	100%	11.8	501.0	16.3
s100	100%	33.0	849.8	12.0
s120	99.7%	109.0	1907.0	17.3
d25-80	99.4%	38.7	2055.6	36.6
d25-100	96.3%	162.8	4728.1	29.3
d25-120	92.5%	163.0	3486.6	44.9
d10-80	96.9%	64.6	3675.5	47.6
d10-100	95.0%	187.2	6045.8	69.8
d10-120	86.9%	186.2	4327.7	59.0

Table 2.6: Results for the "dependent" instances

The stronger the dependence, the smaller the percentage of instances solved within the time limit.

Finally, we divided the sets of test instances "s" into subsets, depending on parameter K_2. The larger this parameter is, the larger is the average margin between due date d_j and the earliest completion time $r_j + p_j$ of jobs. Larger margins give larger average time windows of jobs. Looking at the results in Table 2.7, the efficiency of the algorithm appears to be highly dependent on the type of instances. The instances with the small time windows are harder to solve. Many more nodes and cuts are needed to solve such instances. However on the other hand, $PMCE_{av}$ statistics are about two times larger than for other instances. This means that the modified Carlier algorithm is particularly efficient for this type of instances. Moreover, this algorithm never reaches the node limit. So the stronger cuts are especially useful for instances with small time windows.

Test	P_{1h}	T_{av}	N_{av}	C_{av}	$PMCE_{av}$	$PMCL$
$K_2 = 1$ (the smallest average time window)						
s140	90.0%	654.7	9237.8	61.7	70.7%	0.0%
s120	100%	342.0	6135.1	40.9	71.5%	0.0%
s100	100%	86.0	2372.4	27.4	72.8%	0.0%
s80	100%	12.9	513.6	12.4	72.1%	0.0%
s60	100%	2.3	104.5	4.4	69.3%	0.0%
$K_2 = 5$						
s140	97.5%	77.5	1026.5	21.1	22.8%	23.0%
s120	98.8%	48.7	842.5	20.6	25.1%	21.2%
s100	100%	16.0	540.1	11.4	36.3%	2.1%
s80	100%	17.4	1040.1	37.6	39.3%	0.0%
s60	100%	3.0	322.1	13.8	36.0%	0.1%
$K_2 = 10$						
s140	97.5%	18.3	338.8	3.8	25.9%	6.3%
s120	100%	22.6	414.9	6.2	15.3%	17.6%
s100	100%	10.4	321.5	8.2	18.4%	3.1%
s80	100%	7.0	295.1	11.4	19.6%	1.1%
s60	100%	5.2	467.5	16.9	47.3%	0.6%
$K_2 = 20$ (the largest average time window)						
s140	100%	24.0	288.0	3.9	14.4%	5.3%
s120	100%	21.8	222.4	1.8	15.6%	3.7%
s100	100%	19.6	165.1	1.1	22.9%	7.6%
s80	100%	10.1	155.4	3.8	12.0%	24.3%
s60	100%	3.4	176.1	6.0	27.5%	4.5%

Table 2.7: Results for subsets of the test instances with different time windows

Chapter 3

Multi-machine problems

In this chapter, we consider two multi-machine scheduling problems with release dates. The first problem is the multi-machine assignment scheduling problem, and the second one is the problem of minimizing the maximum tardiness on unrelated machines ($R \mid r_j \mid T_{\max}$).

We present the first problem and review the related literature in Section 3.1. In Section 3.2, we formulate this problem as a MIP. The MIP formulation is decomposed according to the generic Benders approach. Branch-and-check algorithms are derived for the problem in Section 3.3. In Section 3.4, we apply another approach for solving the problem — Dantzing-Wolfe decomposition. Then branch-and-price algorithms which are based on this decomposition are presented. In Section 3.5, we modify the branch-and-check and branch-and-price algorithms to solve the problem $R \mid r_j \mid T_{\max}$. Section 3.6 presents ways to exploit the symmetry in the proposed algorithms in the case when machines are identical. Finally in Section 3.7, the results of the numerical experiments with the different algorithms are presented and discussed.

Part of the results presented in this chapter have been published in [73].

3.1 Multi-Machine Assignment Scheduling Problem

Now we introduce the multi-machine assignment scheduling problem (MMASP). A set $N = \{1, \ldots, n\}$ of jobs have to be processed on a set $M = \{1, \ldots, m\}$ of machines. Any job can be processed on any machine, each machine can process only one job at a time, and preemptions are not allowed. Each job $j \in N$ has a release date r_j and a due date d_j. If job $j \in N$ is assigned to machine $k \in M$, the processing time of j equals p_j^k. If job j is assigned to machine k and j is on-time, it generates a profit w_j^k. The objective is to find a schedule which maximizes the total profit.

The MMASP is a generalization of the scheduling problem $R \mid r_j \mid \sum w_j U_j$. The difference is that the profit of an on-time job in the MMASP depends on the machine the job is processed on, whereas in the latter problem the profit does not depend on the machine. Introduction of the machine-dependent profit may seem to be unrealistic, but it makes sense for the following problem which can be reformulated as the MMASP.

Again, a set N of non-preemptive jobs have to be processed on a set M of machines which can handle only one job at a time. Each job $j \in N$ has a release date r_j and a deadline $\bar{d}_j$. Job j also has machine-dependent processing times p_j^k, $k \in M$ and costs c_j^k. The objective is to process all the jobs within their time windows and minimize the total processing cost. We can reformulate this problem as the MMASP by changing deadlines to due dates ($d_j := \bar{d}_j$, $j \in N$) and converting costs to profits in the following manner:

$$w_j^k := C - c_j^k,\ j \in N,\ i \in M, \qquad \text{where } C = \sum_{j \in N} \max_{k \in M} c_j^k + 1.$$

Such a choice of C guarantees that the profit of any schedule in which all jobs are on-time is larger than the profit of any schedule in which at least one job is late. Therefore, if there exists a feasible schedule for the problem of minimizing total cost, any optimal schedule of the MMASP is also feasible and optimal for the problem of minimizing total cost as all jobs are processed on-time and maximum total profit corresponds to minimum cost.

In the literature, the MMASP has been used to illustrate the effectiveness of IP/CP hybrid methods. Jain and Grossmann [41] have suggested an iterative version of the generic Benders decomposition approach. In their algorithm, the master problem is solved by branch-and bound, and the solution is checked for feasibility by CP. If the solution is infeasible, a cut of the type (2.12) is generated and added to the master which is solved again from the scratch. Later, a more efficient branch-and-check algorithm has been proposed by Thorsteinsson [82]. To our knowledge, the best results so far have been obtained by Bockmayr and Pisaruk [16]. The main difference between their algorithm and that of Thorsteinsson is that they check not only integer solutions of the master problem but also fractional ones for feasibility. In this chapter, we improve on the results of Bockmayr and Pisaruk by following two directions. First, we modify the branch-and-check algorithm of Thorsteinsson by adding constraints to the master problem formulation. These constraints are similar to those that form the relaxation $\bar{\mathcal{X}}(r, p, d)$ proposed in Section 2.2. Secondly, we reformulate the problem and devise branch-and-price algorithms to solve it.

3.2 MIP Formulations

In this section, we give a MIP formulation of the problem. This formulation appeared previously in [41, 16]. Assignment binary variable x_j^k, $j \in N$, $k \in M$, takes value 1 if job j is processed on-time on machine k, and otherwise $x_j^k = 0$. Sequencing binary variable δ_{ij}, $i, j \in N$, takes value 1 if both jobs i and j are processed on-time on the same machine and i precedes j, and otherwise $\delta_{ij} = 0$. Continuous variable s_j, $j \in N$, equals the start time of job j if j is on-time. Continuous variable e_j, $j \in N$, equals the completion time of job j if j is on-time. If job j is late, s_j and e_j take any values from interval $[r_j, d_j]$. Using variables x, δ, s and e, we can write the formulation of (MM).

(MM)

$$\max \quad \sum_{k \in M} \sum_{j \in N} w_j^k x_j^k \tag{3.1}$$

$$s.t. \quad \sum_{k \in M} x_j^k = 1, \quad \forall j \in N, \tag{3.2}$$

$$\sum_{j \in N} p_j^k x_j^k \leq \max_{j \in N} d_j - \min_{j \in N} r_j, \quad \forall k \in M, \tag{3.3}$$

$$s_j + \sum_{k \in M} p_j^k x_j^k - e_j = 0, \quad \forall j \in N, \tag{3.4}$$

$$s_j \geq e_i - U(1 - \delta_{ij}), \quad \forall i, j \in N, i \neq j, \tag{3.5}$$

$$r_j \leq s_j, \quad \forall j \in N, \tag{3.6}$$

$$d_j \geq e_j, \quad \forall j \in N, \tag{3.7}$$

$$\delta_{ij} + \delta_{ji} \leq 1, \quad \forall i, j \in N, i < j, \tag{3.8}$$

$$\delta_{ij} + \delta_{ji} \geq x_i^k + x_j^k - 1, \quad \forall i, j \in N, i < j, \quad \forall k \in M, \tag{3.9}$$

$$x_i^k + x_j^l + \delta_{ij} + \delta_{ji} \leq 2, \quad \forall i, j \in N, i < j, \quad \forall k, l \in M, k \neq l, \tag{3.10}$$

$$x_j^k \in \{0,1\}^{m \times n}, \quad \delta_{ij} \in \{0,1\}. \tag{3.11}$$

Here the equalities (3.2) enforce the assignment of each job to exactly one machine. The inequalities (3.3) state that the total processing time of all the jobs that are assigned to machine m should be less than the difference between the latest due date and the earliest release date. These inequalities are similar to inequalities (2.8) which are valid for the single machine formulation (SM). The constraints (3.4) relate the start and completion times. The sequencing constraints (3.5) ensure that processing of job j begins after processing of job i if $\delta_{ij} = 1$. Here U is a big value and can be, for example, instantiated as the difference between the maximum due date and the minimum release date. In the same manner, as for the formulation (SM), one can show that, if $\delta_{ij} = 0$, constraints (3.5) are dominated by the constraints (3.6) and (3.7), which enforce the release dates and due dates. The constraints (3.8) insure

that only one of two jobs i, j is processed before the other. The constraints (3.9) and (3.10) relate x and δ variables: the first group ensures that if jobs i and j are assigned to machine k, then one must be processed before the other; the second group guarantees that the sequencing variables δ_{ij} and δ_{ji} are both zero if jobs i and j are assigned to different machines.

The formulation (MM) can be strengthened by adding valid inequalities in a similar way as it has been done for the formulation (SM) in Section 2.2. Let $\Theta(r, p, d)$ denotes the feasible region (3.4)-(3.11) for variables (x, δ, s, e). Remember that, for a job $j \in N$, $\alpha_j(r') = (r' - r_j)^+$ and $\beta_j(d') = (d_j - d')^+$. Let

$$\epsilon_j^k(r', d') = \min\left\{d' - r', \left(p_j^k - \alpha_j(r')\right)^+, \left(p_j^k - \beta_j(d')\right)^+\right\}.$$

By the same reasoning as for the constraints (2.10), the following statement is true. If job $j \in N$ is processed on-time on machine $k \in M$, it should be processed in interval $[r', d']$ at least during the time $\epsilon_j^k(r', d')$. So, we get the following proposition.

Proposition 3.1. *Consider an interval $[r', d']$, $0 \leq r' < d'$, and a machine $k \in M$. Then the inequality*

$$\sum_{j \in N} \epsilon_j^k(r', d') \cdot x_j^k \leq d' - r' \tag{3.12}$$

is valid for $\Theta(r, p, d)$.

Let $\bar{\mathcal{X}}^k(r, p^k, d)$ denotes the feasible region defined by the set of the constraints (3.12) for machine $k \in M$ and for all intervals $[r', d']$ based on the release and due dates of jobs, i.e. $r' = r_i$, $d' = d_j$, $i, j \in N$:

$$\bar{\mathcal{X}}^k(r, p^k, d) = \left\{x^k \in \{0,1\}^n : \ \epsilon^k(r_i, d_j) \cdot x^k \leq d_j - r_i, \ \forall i, j \in N, r_i < d_j\right\}.$$

Now we can write the formulation (MMS) which is a strengthening of formulation (MM).

$$\textbf{(MMS)} \qquad \max \quad \sum_{k \in M} \sum_{j \in N} w_j^k x_j^k \tag{3.13}$$

$$s.t. \quad \sum_{k \in M} x_j^k = 1, \quad \forall j \in N, \tag{3.14}$$

$$x^k \in \bar{\mathcal{X}}^k(r, p^k, d), \quad \forall k \in M, \tag{3.15}$$

$$(x, \delta, s, e) \in \Theta(r, p, d). \tag{3.16}$$

The formulations (MM) and (MMS) can be solved by the standard MIP branch-and-cut method. However, it is more efficient to decompose and solve them using a branch-and-check algorithms which are similar to the one proposed for the problem $1 \mid r_j \mid \sum w_j U_j$ in Chapter 2. Such algorithms are presented in the next section.

3.3 Branch-and-Check Algorithms

We now decompose the formulation (MM) by following the generic Benders decomposition scheme. The master problem is

$$\max \quad \sum_{k\in M}\sum_{j\in N} w_j^k x_j^k \tag{3.17}$$

$$s.t. \quad \sum_{k\in M} x_j^k = 1, \quad \forall j \in N, \tag{3.18}$$

$$x^k \in \bar{\mathcal{X}}^k(r, p^k, d), \quad \forall k \in M. \tag{3.19}$$

Let $\bar{x}$ be an integer solution at a node of the master and $\bar{J}^k = \{j \in N : \bar{x}_j^k = 1\}$, $k \in M$. Then the feasibility slave problem can be decomposed into m subproblems, one for each machine. The subproblem for machine $k \in M$ is formulated as

$$\begin{aligned} \text{find} \quad & s_j, \quad j \in \bar{J}^k \\ s.t \quad & \texttt{disjunctive}\left(\{s_j\}_{j\in\bar{J}^k}, \{p_j^k\}_{j\in\bar{J}^k}\right), \\ & s_j \in [r_j, d_j - p_j^k], \quad j \in \bar{J}^k. \end{aligned}$$

For each machine $k \in M$, we check the feasibility of the master problem solution $\bar{x}^k$ in a similar way as in the "str+ef+bas" variant of the branch-and-check algorithm presented in Section 2.4. First, the modified Carlier algorithm is executed. If an infeasible set of jobs $Q^k \subseteq \bar{J}^k$ is found, the cut

$$\sum_{j\in Q^k} x_j^k \leq |\, Q^k \,| - 1$$

is generated. Otherwise, if in the modified Carlier algorithm, the node limit is reached, the separation algorithm for the Edge-Finding-based cuts is applied. If such a cut is not found, the Carlier algorithm is executed and the cut

$$\sum_{j\in\bar{J}^k} x_j^k \leq |\, \bar{J}^k \,| - 1 \tag{3.20}$$

is generated when $\bar{x}^k$ is infeasible. If $\bar{x}^k$ is feasible for all machines $k \in M$, it is stored and the solution of the master problem is continued.

We call the branch-and-check algorithm presented just above as **BCh-MMS**. We call another branch-and-check algorithm as **BCh-MM**. In the latter the constraints (3.19) are replaced by the constraints (3.3) and the slave subproblems are solved by Constraint Programming. If solution $\bar{x}^k$, $k \in M$, is infeasible, the cut (3.20) is generated. Note that the algorithm **BCh-MM** resembles the algorithm proposed by Bockmayr and Pisaruk [16]. The only difference is that, in the latter algorithm, infeasibility cuts are generated for both integer and fractional solutions.

3.4 Branch-and-Price Algorithms

Given the structure of MMASP, it is very natural to think of a column-generation approach. The use of Dantzig-Wolfe decomposition to solve multi-machine scheduling problems is not new. For example, this approach was used by Chen and Powell [24] in tackling a problem similar to MMASP, but with identical release dates.

We now reformulate the MMASP. Let Ω^k denote the set of all feasible partial schedules on machine $k \in M$. Let w_ω be the total profit of partial schedule $\omega \in \Omega^k$, $k \in M$. For each job $j \in N$, let $a_{j\omega} = 1$ if partial schedule ω contains job j, and otherwise $a_{j\omega} = 0$. Binary variable λ_ω, $\omega \in \Omega^k$, $k \in M$, takes value 1 if partial schedule ω is included in the solution. Now we can formulate the MMASP as the master problem (MP).

$$\textbf{(MP)} \qquad \max \quad \sum_{k\in M}\sum_{\omega\in\Omega^k} w_\omega \lambda_\omega \tag{3.21}$$

$$s.t. \quad \sum_{k\in M}\sum_{\omega\in\Omega^k} a_{j\omega}\lambda_\omega \le 1, \quad \forall j \in N, \tag{3.22}$$

$$\sum_{\omega\in\Omega^k} \lambda_\omega \le 1, \quad \forall k \in M, \tag{3.23}$$

$$\lambda_\omega \in \{0,1\}, \quad \forall \omega \in \Omega^k, \forall k \in M. \tag{3.24}$$

The constraints (3.22) specify that each job belongs to at most one partial schedule and the constraints (3.23) guarantee that each machine is occupied by at most one partial schedule.

In the formulation (MP), the number of variables is exponential. Therefore, to solve the LP relaxation of the master, the column generation procedure is applied. Remember that we iterate between a solution of the LP restriction (RMP) of the master problem involving only a subset of the columns, and a solution of the pricing problem. In our case, the pricing problem breaks up into m subproblems, one for each machine. Let $(\theta, \mu) \in \mathbb{R}^{n+m}$ be an optimal dual solution of the LP relaxation of (RMP). To check if there are variables with a positive reduced cost which are not included to (RMP), we solve the pricing problem. The reduced cost of variable λ_ω, $\omega \in \Omega^k$, is equal to $\sum_{j\in N}(w_j^k - \theta_j)a_{j\omega} - \mu_k$. The pricing problem decomposes into m subproblems (SP^k), one for each machine $k \in M$:

$$\textbf{(SP}^k\textbf{)} \qquad \max \quad \sum_{j\in N}(w_j^k - \theta_j)x_j - \mu_k \tag{3.25}$$

$$s.t. \quad (x, \delta, s) \in \Delta(r, p^k, d). \tag{3.26}$$

It is easy to see that each pricing subproblem is exactly the one-machine problem $1 \mid r_j \mid \sum w_j U_j$. It can be solved using the branch-and-check algorithms presented in Chapter 2 as well as by the standard MIP branch-and-cut

method.

As the LP solution of (MP) may not be integer, it is necessary to embed the solution of the LP relaxation of (MP) within a branch-and-bound tree, giving a branch-and-price algorithm. We now discuss branching strategies for such an algorithm. Note that branching on the variables λ is not usually the right choice as this produces an unbalanced search tree and complicates the pricing problem at the nodes inside the tree. Instead, we propose to branch on the variables x of the formulation (MM). By definition, the relation of the variables x of the formulation (MM) and the variables λ of the formulation (MP) is

$$x_j^k = \sum_{\omega \in \Omega^k} a_{j\omega} \lambda_\omega, \quad \forall j \in N, \forall k \in M. \tag{3.27}$$

The next proposition shows that it is possible to branch on the the variables x. The result is probably known, but we include it for completeness.

Proposition 3.2. *Let $\bar{\lambda}$ be an LP solution of the formulation (MP) and $\bar{x}$ be the corresponding solution of the formulation (MM). Then $\bar{x}$ is fractional if and only if $\bar{\lambda}$ is fractional.*

Proof. From (3.27) it follows that if $\bar{\lambda}$ is integer then $\bar{x}$ is integer. Now, to prove the proposition, we show by contradiction that if $\bar{\lambda}$ is fractional then $\bar{x}$ is fractional.

Suppose that $\bar{\lambda}$ is fractional and $\bar{x}$ is integer. Consider the partial schedule $\omega' \in \Omega^{k'}$ with the smallest number of jobs among partial schedules in $\Omega^{k'}$ for which the corresponding variable is fractional: $0 < \lambda_{\omega'} < 1$. Consider a job i' which belongs to partial schedule ω', i.e. $a_{i'\omega'} = 1$. By (3.27) and $\lambda_{\omega'} > 0$ we have $\bar{x}_{i'}^{k'} > 0$, and by integrality of $\bar{x}$, $\bar{x}_{i'}^{k'} = 1$. Define $\Omega' = \{\omega \in \Omega^{k'} : \bar{\lambda}_\omega > 0, a_{i'\omega} = 1\}$. Partial schedule ω belongs to Ω' if ω is processed on machine k', contains job i' and participates in the solution ($\bar{\lambda}_\omega > 0$).

As $\omega' \in \Omega'$, $\bar{\lambda}_{\omega'} < 1$ and $\sum_{\omega \in \Omega'} \bar{\lambda}_\omega = \bar{x}_{i'}^{k'} = 1$, set Ω' contains another partial schedule ω''. ω' and ω'' do not coincide and ω'' contains at least as many jobs as ω'. Therefore there exists a job i'' which belongs to ω'' and does not belong to ω'. Similar to Ω' we define $\Omega'' = \{\omega \in \Omega^{k'} : \bar{\lambda}_\omega > 0, a_{i''\omega} = 1\}$. Again we have $\sum_{\omega \in \Omega''} \bar{\lambda}_\omega = \bar{x}_{i''}^{k'} = 1$.

As $\Omega' \cup \Omega'' \subseteq \Omega^{k'}$ and $\omega \notin \Omega''$,

$$\sum_{\omega \in \Omega^{k'}} \bar{\lambda}_\omega \geq \sum_{\omega \in \Omega' \cup \Omega''} \bar{\lambda}_\omega \geq \sum_{\omega \in \Omega''} \bar{\lambda}_\omega + \bar{\lambda}_{\omega'} = 1 + \bar{\lambda}_{\omega'} > 1,$$

but this contradicts the constraint (3.23) for $k = k'$. Therefore, if $\bar{\lambda}$ is fractional, then $\bar{x}$ is fractional. □

So, when the LP solution of the formulation (MP) is fractional, there exists a variable $x_{j'}^{k'}$ which also fractional. To branch, we set $x_{j'}^{k'} = 1$ at the first descendant node and we set $x_{j'}^{k'} = 0$ at the second one. This means

that, at the first node, all partial schedules or columns for machine k' must contain job j' and, at the second node, they must not contain j'. Such constraints can be easily satisfied when solving the pricing subproblems. For the branching variable selection, we use a standard "most-fractional-variable" strategy, choosing to branch on the variable which value is the closest to 0.5. The node selection strategy used is best bound.

Suppose that a variable $x_{j'}^{k'}$ is chosen for branching at some node of the branch-and-price enumeration tree. Then two descendant nodes are created, one with the additional constraint $x_{j'}^{k'} = 1$ and the other with $x_{j'}^{k'} = 0$. The (RMP) at the descendant nodes is then initialized by taking those columns of the parent node that satisfy the additional constraint.

After generating and solving (RMP) by LP at each node, we then solve (RMP) with its present set of columns as an IP, with the integrality constraints (3.24). Typically this restricted IP can be solved fairly rapidly, and provides a good feasible solution without going deep into the search tree. In addition, if the optimal value of this restricted IP is the integer round-up of the optimal value of (RMP), the node can be immediately pruned.

Depending on how the pricing subproblems are solved, we consider three variants of the branch-and-price algorithm proposed here. In the first variant **BP-SM**, we solve the formulations (SP^k), $k \in M$, using the standard MIP branch-and-cut method. In the second variant **BP-SMS**, each formulation (SP^k), $k \in M$, is amended by constraints $x^k \in \bar{\mathcal{X}}^k(r, p^k, d)$ and solved again by the standard MIP branch-and-cut method. The last variant **BP-BCh** uses the variant "str+ef+bas" of the branch-and-check algorithm presented in Section 2.4 to solve the pricing subproblems.

3.5 Minimizing the Maximum Tardiness on Unrelated Machines

First, we define the problem. A set $N = \{1, \ldots, n\}$ of non-preemptive jobs have to be processed on a set $M = \{1, \ldots, m\}$ of machines which can handle only one job at a time. Each job $j \in N$ has a release date r_j and a due date d_j. If job $j \in N$ is assigned to machine $k \in M$, the processing time of j equals p_j^k. The objective is find a schedule which minimizes the maximum tardiness $T_{\max} = \max_{j \in N}\{C_j - d_j, 0\}$. In the standard scheduling notation, the problem is denoted as $R \mid r_j \mid T_{\max}$.

Note that the criterion $T_{\max}$ is equivalent to the criterion $L_{\max}$. For a schedule π, if $L_{\max}(\pi) \geq 0$ then, by definition, $T_{\max}(\pi) = L_{\max}(\pi)$. To guarantee the condition $L_{\max}(\pi) \geq 0$ for any schedule, it suffices to decrease the due dates of all jobs by the value LB, where LB is a lower bound on the minimum maximum lateness. Decreasing the due dates of all jobs by an identical constant increases the lateness of any schedule by the same constant and does not change the set of optimal schedules of the problem.

Multi-machine scheduling problems with the maximum lateness criteria have not received as much attention in the literature as their one-machine analogue, the problem $1 \mid r_j \mid L_{\max}$. Some work has been done for the problem $P \mid r_j \mid L_{\max}$ with identical machines. Studies of lower bounds for this problem have been performed by Haouari and Gharbi [39, 40] and by Vandevelde et al. [83]. Exact branch-and-bound methods have been suggested by Carlier [22] and Gharbi and Haouari [37]. Lancia [48] has proposed an exact algorithm for the problem with two unrelated machines. To the best of our knowledge, the problem $R \mid r_j \mid L_{\max}$ with an arbitrary number of unrelated machines has not been studied in the literature.

In this section we will modify the algorithms we proposed for the MMASP for solving the maximum tardiness problem. Subsection 3.5.1 deals with the branch-and-check algorithm and subsection 3.5.2 concerns the branch-and-price algorithm.

3.5.1 Branch-and-Check Algorithm

We begin by modifying the formulation (MM) to the formulation (MMT) for the problem $R \mid r_j \mid T_{\max}$. Continuous variable t equals the maximum tardiness of the schedule.

$$\textbf{(MMT)} \qquad \begin{aligned} \min \quad & t && (3.28)\\ s.t. \quad & \sum_{k \in M} x_j^k = 1, \quad \forall j \in N, && (3.29)\\ & e_j - d_j \leq t, \quad \forall j \in N, && (3.30)\\ & (3.4) - (3.6), (3.8) - (3.11), && (3.31)\\ & t \geq 0. && (3.32) \end{aligned}$$

Here the constraints (3.30) guarantee that the tardiness of any job does not exceed the maximum tardiness. Let $\Theta^{\top}(r, p, d)$ denotes the feasible region (3.30)-(3.32) for variables (x, δ, s, e, t). Note that, for a fixed $t' \geq 0$, $\Theta^{\top}(r, p, d)$ is equivalent to $\Theta(r, p, d + t')$.

The formulation (MMT) can be strengthened by adding modified constraints (3.12), as it is shown in the next proposition.

Proposition 3.3. *Consider an interval $[r', d']$, $0 \leq r' < d'$, and a machine $k \in M$. Then the inequality*

$$\sum_{j \in N} \epsilon_j^k(r', d') \cdot x_j^k \leq d' + t - r' \tag{3.33}$$

is valid for $\Theta^{\top}(r, p, d)$.

Proof. We fix variable t to an arbitrary value $t' \geq 0$. By Proposition 3.1, inequality $\epsilon^k(r', d'+t')x^k \leq d'+t'-r'$ is valid for $\Theta(r, p, d+t')$. As $\epsilon_j^k(r', d') \leq \epsilon_j^k(r', d' + t')$, $\forall j \in N$, inequality $\epsilon^k(r', d')x^k \leq d' + t' - r'$ is also valid for

$\Theta(r, p, d + t')$. Therefore the inequality (3.33) is valid for $\Theta^{\top}(r, p, d)$ for any value $t' \geq 0$. □

Remark. When we do not have the constraint $t \geq 0$, i.e. when we consider the problem $R \mid r_j \mid L_{\max}$, the constraint (3.33) is not valid any more, as inequality $\epsilon^k(r', d') \leq \epsilon^k(r', d' + t')$ does not hold true for $t' < 0$.

We now give an outline of the branch-and-check algorithm for the problem. Let

$$\bar{\mathcal{X}}_t^k(r, p^k, d) =$$
$$\left\{(x^k, t) \in \{0,1\}^n \times \mathbb{R}_+ : \ \epsilon^k(r_i, d_j)x^k \leq d_j + t - r_i, \ \forall i, j \in N, r_i < d_j\right\}.$$

Then the master problem is

$$\begin{array}{ll} \min & t \\ s.t. & \displaystyle\sum_{k \in M} x_j^k = 1, \quad \forall j \in N, \\ & (x^k, t) \in \bar{\mathcal{X}}_t^k(r, p^k, d), \quad \forall k \in M. \end{array}$$

Let $(\bar{x}, \bar{t})$ is an integer solution of the master at some node of the search tree and $\bar{J}^k = \{j \in N : \bar{x}_j^k = 1\}$, $k \in M$. Again, the feasibility slave problem can be decomposed into m subproblems. The subproblem for machine $k \in M$ is formulated as

$$\begin{array}{ll} \text{find} & s_j, \quad j \in \bar{J}^k \\ s.t & \texttt{disjunctive}\left(\{s_j\}_{j \in \bar{J}^k}, \{p_j^k\}_{j \in \bar{J}^k}\right), \\ & s_j \in [r_j, d_j + \bar{t} - p_j^k], \quad j \in \bar{J}^k. \end{array}$$

To solve it, we solve the problem $1 \mid r_j \mid L_{\max}$ for set $\bar{J}^k$ of jobs with initial due dates using the Carlier algorithm. Let L_k^* be the value of an optimal solution. If $L_k^* > \bar{t}$ then the solution of the master is infeasible and we should generate an infeasibility cut. For this, we use the modified Carlier algorithm for set $\bar{J}^k$ of jobs and the constant $L' = L_k^* - 1$. As there is no schedule with the maximum lateness $L_k^* - 1$, we obtain an infeasible set of jobs $Q^k \subseteq \bar{J}^k$ and the infeasibility cut is

$$t \geq L_k^* - \sum_{j \in Q^k} (1 - x_j^k) \cdot (L_k^* - \bar{t}). \tag{3.34}$$

It is easy to see that such an infeasibility cut is valid. If all the jobs in set Q^k are assigned to machine k then (3.34) reduces to $t \geq L_k^*$ which is valid as, by the modified Carlier algorithm, any schedule of jobs in Q^k cannot have maximum tardiness equal to or less than $L_k^* - 1$. If at least one job in Q^k is not assigned to machine k, we have constraint $t \geq t'$, where t' is the right-hand side of (3.34) and $t' \leq \bar{t}$. This constraint is valid as $\bar{t}$ is a lower bound on the optimal maximum tardiness.

If the modified Carlier algorithm reaches the node limit, we interrupt it and set $Q^k = \bar{J}^k$ in the cut (3.34). If $\bar{x}^k$ is feasible for all machines $k \in M$, it is stored and the the solution of the master problem is continued.

3.5.2 Branch-and-Price Algorithm

We now apply Dantzig-Wolfe reformulation to (MMT). Let Ω^k denote the set of all partial schedules on machine $k \in M$. Let t_ω be the maximum tardiness of partial schedule $\omega \in \Omega^k$, $k \in M$. Again, $a_{j\omega} = 1$ if job $j \in N$ belongs to partial schedule ω, and λ_ω is a binary variable, $\omega \in \Omega^k$, $k \in M$. Continuous variable $t_{\max}$ represents the maximum tardiness. The master problem is the following.

(MPT)

$$\min \quad t_{\max} \tag{3.35}$$

$$s.t. \quad \sum_{k \in M} \sum_{\omega \in \Omega^k} a_{j\omega} \lambda_\omega = 1, \quad \forall j \in N, \tag{3.36}$$

$$\sum_{\omega \in \Omega^k} t_\omega \lambda_\omega \leq t_{\max}, \quad \forall k \in M, \tag{3.37}$$

$$\sum_{\omega \in \Omega^k} \lambda_\omega \leq 1, \quad \forall k \in M, \tag{3.38}$$

$$\lambda_\omega \in \{0,1\}, \quad \forall \omega \in \Omega^k, \forall k \in M. \tag{3.39}$$

The constraints (3.37) specify that the tardiness of each partial schedule does not exceed the maximum tardiness.

Let $(\theta, \nu, \mu) \in \mathbb{R}^{n+m+m}$ be an optimal dual solution of the LP relaxation of (MPT). Then the reduced cost of variable λ_ω, $\omega \in \Omega^k$, is equal to $-\sum_{j \in N} \theta_j a_{j\omega} - \nu^k t_\omega - \mu^k$. To check if there are variables with a negative reduced cost, we need to solve the pricing problem, which decomposes here into m subproblems (SPT^k), one for each machine $k \in M$:

(SPT^k)

$$\min \quad -\sum_{j \in N} \theta_j x_j - \nu^k t - \mu_k \tag{3.40}$$

$$s.t. \quad (x, \delta, s, t) \in \Delta^\top(r, p^k, d), \tag{3.41}$$

where $\Delta^\top(r, p^k, d)$ denotes the feasible region $\Delta(r, p^k, d)$ in which constraints $s_j + p_j^k \leq d_j$, $j \in N$, are replaced by constraints $s_j + p_j^k \leq d_j + t$, $j \in N$. In (SPT^k), binary variables x determine a values of the column with the minimum reduced cost. Each pricing subproblem can be denoted as $1 \mid r_j \mid \sum w_j U_j + w_0 T_{\max}$. It can be solved using the branch-and-check algorithm presented in subsection 3.5.1.

When the LP solution of (MP) is not integer, we branch on the variables x, as in the branch-and-price algorithm for the MMASP in Section 3.4. The node selection strategy and the branching variable strategy are also the same.

3.6 The Case with Identical Machines

In this section, we turn to a special case of the MMASP and the problem $P \mid r_j \mid T_{\max}$. In this case the machines are identical. This means that processing times and profits of any job are now the same regardless of the machine on which the job is processed. So, we have $p_j^k = p_j$, $w_j^k = w_j$, $j \in N$, $k \in M$.

In subsection 3.6.1, we will discuss how to use the symmetry to decrease the solving time in the branch-and-check algorithms. Subsection 3.6.2 describes how to break symmetry in Dantzig-Wolfe reformulations.

3.6.1 Symmetry Breaking in Branch-and-Check

It can be noticed that, if the machines are identical, the formulations (MM) and (MMT) have many symmetric solutions. Consider a solution of one of these formulations. Let ω_k, $k \in M$, be the partial schedule of jobs processed on machine k in this solution. Consider now a permutation of machines $\tau = (\tau_1, \ldots, \tau_m)$. The modified solution in which each partial schedule ω_k is processed on machine τ_k is feasible, as the processing times of jobs do not change, and has the same objective value as the initial solution, as the profits of jobs (the maximum tardiness of partial schedules) do not change. As the number of all permutations is $m!$, there are $m!$ solutions in each symmetry group.

In order to consider only one solution in each symmetry group, we propose to use a modified branching procedure when solving the master problem in the branch-and-check algorithm. Note that a general branching procedure for MIPs with large symmetry group has been proposed by Margot [59]. We now present a modified branching procedure, which is equivalent to the Margot's approach for the case with identical machines. This procedure is simpler and easy to understand and implement.

At a node of the search tree, we call a variable x_j^k, $j \in N$, $k \in M$, *fixed* if it has been set to 1 or 0 by the branching decision at some ascendant node. Let M_f^φ be the set of "empty" machines at node φ, i.e. for which there are no variables fixed to 1. The modified branching procedure consists in the following. Suppose, at some node φ of the search tree, variable $x_j^{k'}$ is chosen to branch on. As usual, we set $x_j^{k'}$ to 1 at the first child node φ_1 (and we set variables x_j^k, $k \in M$, $k \neq k'$, to zero, as $\sum_{k \in M} x_j^k \leq 1$). We set also $M_f^{\varphi_1} := M_f^\varphi \setminus \{k\}$. Setting variables at the second child node φ_2 can change. If k' is in M_f, we set all the variables x_j^k, $k \in M_f$, related to job j and machines in M_f to zero. In the words, if we have tried to assign job j to some "empty" machine, it is useless to assign j to another "empty" machine, as all the machines are identical. If $k' \notin M_f$, as usual, we set $x_j^{k'}$ to 0 at the second child node. We set also $M_f^{\varphi_2} := M_f^\varphi$. The modified branching procedure is given formally in Algorithm 3.1. Example of application of this procedure is

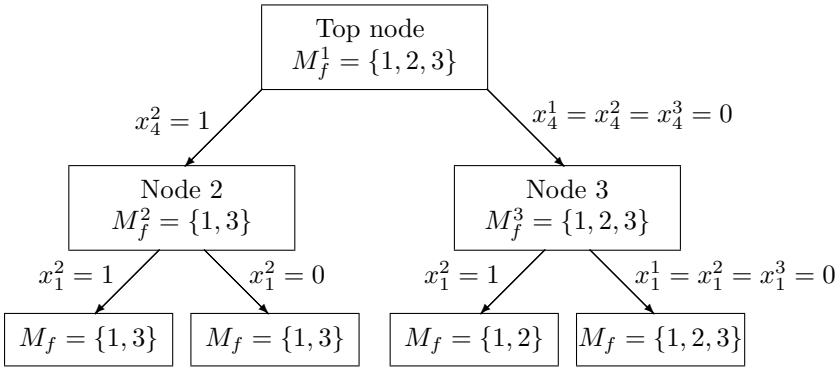


Figure 3.1: Application of the modified branching procedure for an instance with 3 machines

shown in Figure 3.1.

Algorithm 3.1 The modified branching procedure for breaking the symmetry

1: M_f^{φ} is calculated at the parent node ($M_f^{\varphi} := M$ if it is the top node)
2: choose a variable $x_j^{k'}$ to branch on;
3: at the first child node φ_1, set $x_j^{k'} = 1$, $x_j^k = 0$, $\forall k \in M$, $k \neq k'$;
4: $M_f^{\varphi_1} := M_f \setminus \{k\}$.
5: **if** $k' \in M_f$ **then**
6: at the second child node, set $x_j^k = 0$, $\forall k \in M_f$;
7: **else**
8: at the second child node, set $x_j^{k'} = 0$;
9: **end if**
10: $M_f^{\varphi_2} := M_f$.

In the following two propositions, we prove that, when the modified branching is applied, we obtain an optimal solution of the formulations (MM) and (MMT) and consider at most one solution in each symmetry group.

Proposition 3.4. *Suppose all the machines $k \in M$ are identical. Then the modified branching procedure given in Algorithm 3.1 does not prune all the optimal solutions of the formulations (MM) and (MMT).*

Proof. First, let $k' \notin M_f$. Then the modified branching does not differ from the usual branching.

To prove the proposition in the case where $k' \in M_f$, we prove that if there is an optimal solution $\tilde{x}$ pruned in Algorithm 3.1, there is also an optimal solution situated in the subtree rooted at the first child node. This case

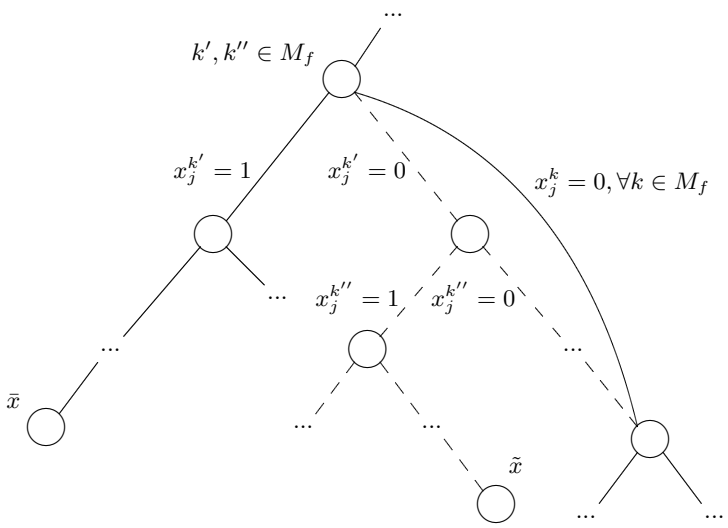


Figure 3.2: Illustration for the Proposition 3.4

is illustrated in Figure 3.2. There, dashed lines denote branching decisions which are considered when the usual branching is applied and not considered when the modified branching procedure is applied. $\tilde{x}$ can be pruned only if $\tilde{x}_j^{k''} = 1$, $k'' \in M_f$, $k'' \neq k'$. Let $\tilde{\omega}_k$ be the partial schedule processed on machine $k \in M$ in solution $\tilde{x}$. Consider the solution $\bar{x}$ in which partial schedule $\tilde{\omega}_{k'}$ is processed on machine k'', $\tilde{\omega}_{k''}$ is processed on k', and each partial schedule $\tilde{\omega}_k$, $k \in M$, $k \neq k'$, $k \neq k''$, is processed on machine k. As the machines k' and k'' are identical, solution $\bar{x}$ has the same objective value as $\tilde{x}$ and also optimal.

As neither variables $x_i^{k'}$ nor variables $x_i^{k''}$, $i \in N$, are fixed to 1 at the current node, solution $\bar{x}$ is situated in the subtree rooted at the first child node. □

Proposition 3.5. *If the modified branching is applied, then at most one solution of the formulations (MM) and (MMT) is considered in each symmetry group.*

Proof. We will show that, for any pair of solutions in the same symmetry group, at least one solution is pruned. This will prove the proposition.

Consider any full enumeration tree and two solutions $\bar{x}$ and $\tilde{x}$ which are in the same symmetry group. Let φ be the node in the tree such that $\bar{x}$ is situated in the subtree rooted at the first child node of φ, and $\tilde{x}$ is situated in the subtree rooted at the second child node of φ. Without loss of generality suppose that the first branching decision is $x_j^{k'} = 1$, i.e. job j is processed

on machine k' in $\bar{x}$. Let j be processed on machine k'' in $\tilde{x}$. We know that $k' \neq k''$. Then, as $\bar{x}$ and $\tilde{x}$ are in the same symmetry group, a job processed is on machine k' in $\bar{x}$ if and only if it is processed on machine k'' in $\tilde{x}$.

There are no variables $x_i^{k'}$ or $x_i^{k''}$, $i \in N$, fixed to 1 at node φ, as otherwise $\bar{x}$ and $\tilde{x}$ would not be in the same symmetry group (we would have $\bar{x}_i^{k'} = \tilde{x}_i^{k'} = 1$ or $\bar{x}_i^{k''} = \tilde{x}_i^{k''} = 1$ for some $i \in N$). Then, both k and k' are in M_f at node φ. Therefore, the second branching decision includes $x_j^{k''} = 0$, and the solution $\tilde{x}$ is pruned. □

Remark. The same idea also works for the case in which we have different families of identical machines. It suffices to slightly adjust Algorithm 3.1. In this case, we find set M_f and check condition $k' \in M_f$ for each family of machines.

3.6.2 Symmetry Breaking in Branch-and-Price

In this subsection, we will show how to modify the Dantzig-Wolfe reformulation (MP) to break the symmetry. The case with the formulation (MPT) is analogous.

As the machines are identical, we will not distinguish between the sets of partial schedules Ω^k, $k \in M$. Now we have one set Ω of all feasible partial schedules. The master formulation (MP) is then becomes

$$\textbf{(MPP)} \qquad \begin{aligned} \max \quad & \sum_{\omega \in \Omega} w_\omega \lambda_\omega && (3.42)\\ s.t. \quad & \sum_{\omega \in \Omega} a_{j\omega} \lambda_\omega \leq 1, \quad \forall j \in N, && (3.43)\\ & \sum_{\omega \in \Omega} \lambda_\omega \leq m, && (3.44)\\ & \lambda_\omega \in \{0,1\}, \quad \forall \omega \in \Omega. && (3.45) \end{aligned}$$

The constraint (3.44) says that we can choose at most m partial schedules, as there are m machines available.

Let $(\theta, \mu) \in \mathbb{R}^n \times \mathbb{R}$ be an optimal dual solution of the (MPP). The pricing problem here is

$$\max \Big\{ \sum_{j \in N} (w_j - \theta_j) x_j - \mu : \ (x, \delta, s) \in \Delta(r, p, d) \Big\}.$$

We can derive the same branch-and-price algorithm for the formulation (MPP) as the for the formulation (MP). However, one component of this algorithm should be modified, as we do not have the relation (3.27) between variables x and λ anymore. So, the branching strategy must be changed.

Notice that the master (MPP) takes the form of a set partitioning problem. For such problems, a branching scheme was developed by Ryan and

Foster [71]. They showed that, for any fractional solution $\bar{\lambda}$, there exists a pair of items (i, j) (in our case, jobs) such that $\bar{\lambda}$ can be separated using the disjunction

$$\sum_{\omega: a_{i\omega}=a_{j\omega}=1} \lambda_\omega \leq 0 \quad \text{or} \quad \sum_{\omega: a_{i\omega}=a_{j\omega}=1} \lambda_\omega \geq 1.$$

These constraints can be directly implemented in the pricing problem of the descendant nodes. At the first child node, as no column that have both $x_i = 1$ and $x_j = 1$ can be used, we add constraint $x_i + x_j \leq 1$ to the pricing problem. At the second child node, as jobs i and j are entirely covered by columns $\omega \in \Omega$ with $a_{i\omega} = a_{j\omega} = 1$, the pricing problem is augmented by constraint $x_i = x_j$.

Adding linear constraints to the pricing problem does not change the solution method for it, as the pricing problem is solved using the branch-and-check or branch-and-cut method. Note that a generic branching scheme which does not change the structure of the pricing problem in branch-and-price algorithms, has been suggested by Vanderbeck [85]. In this scheme, one uses the same algorithm for solving the pricing problem both at the root node and inside the search tree. However, at a node inside the search tree, the pricing algorithm can be called several times.

3.7 Numerical Experiments

In this section we present results of numerical experiments for the algorithms proposed above in Chapter 3. For reminder, the formulations for the branch-and-cut and branch-and-check algorithms are shown in Table 3.1, and the formulations for the pricing problem of the different branch-and-cut algorithms are shown in Table 3.2. All experiments were carried out on a PC with a Pentium 4 2GHz processor and 512 Mb RAM. The algorithms were implemented in the *Mosel* modelling and optimization language [26], using *XPress-MP* (version 14.21) as the MIP solver, and *CHIP* version 5.4.3 as the CP solver. It should be emphasized that both implementations are completely written in *Mosel*, and neither has been optimized in any way.

In subsection 3.7.1, results for seven algorithm for the MMASP are shown. Subsection 3.7.2 presents results for two algorithms for the problem $R \mid r_j \mid T_{\max}$.

3.7.1 The MMASP

The first nine instances of the MMASP problem are taken from [16]. The names of these instances are of the form "$m - n[\gamma]$", where m is the number of machines, n is the number of jobs, and γ denotes a character to distinguish instances of the same size. Note that these instances are with the total cost objective which needs to be minimized. A way to convert these instances to

MM	**MMS**
max wx *s.t.* $\sum_{k\in M} x_j^k = 1, \quad \forall j \in N,$ $(x, \delta, s, e) \in \Theta(r, p, d).$	max wx *s.t.* $\sum_{k\in M} x_j^k = 1, \quad \forall j \in N,$ $x^k \in \bar{\mathcal{X}}^k(r, p^k, d), \quad \forall k \in M,$ $(x, \delta, s, e) \in \Theta(r, p, d).$
BCh-MM	**BCh-MMS**
max wx *s.t.* $\sum_{k\in M} x_j^k = 1, \quad \forall j \in N,$ infeasibility cuts $x \in \{0,1\}^{n\times m}.$	max wx *s.t.* $\sum_{k\in M} x_j^k = 1, \quad \forall j \in N,$ $x^k \in \bar{\mathcal{X}}^k(r, p^k, d), \quad \forall k \in M,$ infeasibility cuts $x \in \{0,1\}^{n\times m}.$

Table 3.1: Formulations for different algorithms to solve the MMASP

BP-SM	**BP-SMS**	**BP-BCh**
max wx *s.t.* $(x, \delta, s) \in \Delta(r, p, d).$	max wx *s.t.* $x \in \bar{\mathcal{X}}(r, p, d),$ $(x, \delta, s) \in \Delta(r, p, d).$	max wx *s.t.* $x \in \bar{\mathcal{X}}(r, p, d),$ infeasibility cuts $x \in \{0,1\}^n.$

Table 3.2: Formulations for the pricing problem in different branch-and-price algorithms to solve the MMASP

the MMASP instances was shown in Section 3.1. The time limit to solve these instances was set to 1 hour.

In the experiments, for each instance, we were interested in the following statistics.

Obj — The value of the best solution found.

LB — The value of the best lower bound found.

Time — The solution time in seconds.

LP (XLP) — The value of the LP relaxation bound (after adding standard XPress-MP cuts). For the branch-and-check and the branch-and-price algorithms, this bound is given by the solution of the LP relaxation of the master problem at the top node.

Nds — The number of nodes generated in the search tree.

Test	Obj	LB	Time	XLP	Obj	LB	Time	XLP
		MM				**MMS**		
3-12a	101	101.0	65.0	98.0	101	101.0	6.8	99.6
3-12b	104	104.0	61.1	99.1	104	104.0	0.5	104.0
5-15a	115	115.0	388.6	113.0	115	115.0	16.6	114.4
5-15b	129	129.0	277.3	121.0	129	129.0	3.2	128.9
5-20a	159[1]	156.0	>1h	156.0	158	158.0	83.6	157.6
5-20b	143[1]	135.6	>1h	135.1	139	139.0	1066.5	137.4
6-24	238[1]	224.0	>1h	223.7	227	227.0	2405.5	226.3
7-30	- [2]	205.6	>1h	205.5	- [2]	211.4	>1h	210.8
8-34	- [2]	242.6	>1h	242.6	- [2]	249.9	>1h	249.9
Test		**BCh-MM**				**BCh-MMS**		
3-12a	101	101.0	0.3	97.9	101	101.0	0.4	99.9
3-12b	104	104.0	2.3	98.4	104	104.0	0.3	104.0
5-15a	115	115.0	0.4	112.7	115	115.0	0.6	114.5
5-15b	129	129.0	4.3	121.1	129	129.0	0.6	129.0
5-20a	158	158.0	6.1	154.9	158	158.0	1.3	157.7
5-20b	139	139.0	287.7	134.3	139	139.0	2.7	137.6
6-24	227[1]	225.1	>1h	222.7	227	227.0	3.7	226.3
7-30	215[1]	207.7	>1h	203.6	213	215.0	16.7	211.0
8-34	253[1]	243.6	>1h	241.6	252[1]	250.9	>1h	250.1

[1] Optimality of the solution is not proven
[2] A feasible solution is not found

Table 3.3: Branch-and-cut and branch-and-check algorithms for the MMASP

Cts — The number of infeasibility cuts generated in the branch-and-check algorithm.

Its - The total number of iterations in the branch-and-price algorithm.

Results for the first group of instances are shown in Table 3.3 and Table 3.4. The algorithms **MM** and **MMS** consist in just solving the formulations (MM) and (MMS) by the MIP solver. Experimental results showed that the scheduling constraints $(x, \delta, s, e) \in \Theta(r, p, d)$ do not improve the LP bound at all, whereas the constraints $x^k \in \bar{\mathcal{X}}^k(r, p^k, d)$, $k \in M$, improve the LP bound significantly. So, in the results obtained, the LP values (which are not reported in Table 3.3) satisfy

$$LP_{\text{BCh}-\text{MM}} = LP_{\text{MM}} \leq LP_{\text{MMS}}$$

and that

$$LP_{\text{BCh}-\text{MM}} \leq LP_{\text{BCh}-\text{MMS}} = LP_{\text{MMS}}.$$

	BP-SM			BP-SMS		BP-BCh		
Test	Obj	LB	Time	Obj	Time	Obj	Time	LP
3-12a	101	*	36.9	101	8.3	101	2.6	100.5
3-12b	104	*	51.7	104	17.5	104	2.3	104.0
5-15a	115	*	32.9	115	20.2	115	5.3	114.5
5-15b	129	*	52.7	129	17.1	129	5.3	129.0
5-20a	158	*	245.5	158	61.8	158	11.2	157.8
5-20b	139	*	475.5	139	83.4	139	18.2	138.5
6-24	232[1]	226.5	>1h	227	794.5	227	36.8	226.5
7-30	-[3]	-	>1h	213	444.0	213	57.8	212.2
8-34	-[3]	-	>1h	252	2857.2	252	90.9	251.3

[1] Optimality of the solution is not proven
[3] The LP relaxation of the master is not solved

Table 3.4: Branch-and-price algorithms

Remember that the formulation (MMS) differs from (MM) only by the presence of inequalities (3.15). Therefore LP_{MMS} cannot be less than LP_{MM}, and $LP_{\mathrm{BCh-MMS}}$ cannot be less than $LP_{\mathrm{BCh-MM}}$. Note that the XLP values behave similarly as they are typically less than one more than the LP values.

The best earlier results that we know of are those of Bockmayr and Pisaruk [16]. The algorithm **BCh-MM** is essentially the algorithm that they proposed. Our results for this algorithm resemble theirs in that they managed to solve the first six instances, but could not solve the last three within one hour.

We observe that the LP bound obtained by column generation in the branch-and-price algorithms (Table 3.4) is always better than the LP bounds from the direct MIP formulations (Table 3.3). As expected, column generation LP bounds are very tight. In addition, solving the restricted master problem as an IP at each node produces very good integer solutions quickly, as seen by the very small number of tree nodes (see Table 3.5).

It can be seen that that the algorithms **MM** and **BP-SM** have difficulties when solving both small and larger instances and they are not competitive with the others. Algorithms **MMS** and **BCh-MM** can solve small instances quite fast but run into trouble when solving larger instances. The reasons are the large size of the formulation for the former and the large number of infeasibility cuts generated for the latter.

It can be seen that the algorithm **BCh-MMS** is the fastest to solve all the instances except the biggest one "8-34". Algorithms **BP-SMS** and **BP-BCh** are the only two which solved all the instances within the time limit. The latter is much faster than the former. Overall, the algorithms **BCh-MMS** and **BP-BCh** with the strengthened MIP formulation clearly dominate the

others.

To compare these two algorithms further, we then generated some larger instances. For each instance with m machines and $n = \eta m$ jobs, some parameters are first defined randomly and uniformly: $bmc_k \in [6, 12]$ (base machine cost), $bmt_k \in [bmc_k - 2, bmc_k + 2]$ (base machine processing time), $k \in M$, $bjc_j \in [6, 12]$ (base job cost), $bjt_j \in [bjc_j - 2, bjc_j + 2]$ (base job processing time), $j \in N$. Costs and processing times of jobs are distributed uniformly in the following intervals: $c_j^k \in [round(bmc_k/2 + bjc_j/2) - 3, round(bmc_k/2 + bjc_j/2) + 3]$, $p_j^k \in [round(bmt_k/2 + bjt_j/2) - 3, round(bmt_k/2 + bjt_j/2) + 3]$, $k \in M$, $j \in N$. Then release dates and deadlines are generated with uniform distributions in the following intervals: $r_j \in [0, 10]$, $d'_j \in [\beta - 10, \beta + 10]$, $d_j = \max\{d'_j, r_j + \max_{k \in M}\{p_j^k\}\}$, $j \in N$, where $\beta = \sum_{k \in M, j \in N}(p_j^k)\frac{\theta}{m^2}$. Here θ is the "freedom" parameter; smaller θ means tighter deadlines. All the data generated are integer.

We generated five instances for each triple of parameters: (m, η, θ), where $m \in \{7, 8, 9\}$, $\eta \in \{3, 4, 5, 6\}$, and $\theta \in \{0.5, 0.6, 0.8, 1\}$. In Table 3.5, we present results for only those instances, which have at least one feasible solution (all the jobs can be scheduled inside their time windows) and for which the solution by the algorithm **BCh-MMS** took more than 100 seconds. Names of all the instances from the second group have the form "$m - n - \theta - \kappa$", where κ denotes the κ-th instance with the same parameters "$m - n - \theta$". Details are given for the nine instances from the first set, as well as the 25 newly generated instances.

We observe that, among 25 new instances, 7 instances are not solved by the branch-and-check algorithm, whereas the branch-and-price algorithm solved all the instances. Moreover, only 4 instances from 25 were solved faster by the algorithm **BCh-MMS**. We can conclude that the branch-and-price algorithm is more efficient than the branch-and-check algorithm for the larger instances.

Note, that the number of infeasibility cuts generated in the branch-and-check algorithm is quite small. So the quality of the relaxation (3.15) is good in the sense that the master problem (MP) produces a small number of infeasible integer solutions. The main difficulty with the branch-and-check algorithm seems to be the master problem (MP) itself. It cannot be efficiently solved by a state-of-the-art MIP solver. One of the possible directions to overcome this difficulty could be deriving new families of cuts which cut off fractional solutions.

3.7.2 The Maximum Tardiness Problem

The instances of the problem $R \mid r_j \mid T_{\max}$ were obtained by converting the MMASP instances from the first test set. In order to insure positiveness of the optimal value of the maximum tardiness the due dates of jobs were reduced by 10 (for the instance "6-24", by 15). The time limit to solve these instances was set to 10 minutes.

	Branch-and-check					Branch-and-price				
Test	*Obj*	Time	Nds	Cts	*XLP*	*Obj*	Time	Nds	Its	*LP*
3-12a	101	**0.4**	61	0	99.9	101	2.6	1	26	100.5
3-12b	104	**0.3**	1	0	104.0	104	2.3	1	21	104.0
5-15a	115	**0.6**	62	0	114.5	115	5.3	3	32	114.5
5-15b	129	**0.6**	1	0	129.0	129	5.3	3	35	129.0
5-20a	158	**1.3**	77	0	157.7	158	11.2	3	51	157.8
5-20b	139	**2.7**	392	8	137.5	139	18.2	5	45	138.5
6-24	227	**3.7**	416	0	226.3	227	36.8	2	44	226.5
7-30	213	**16.7**	516	10	211.0	213	57.8	6	83	212.2
8-34	252[1]	>1h	221168	0	250.1	252	**90.9**	10	76	251.3
	250.9									
7-35-0.6-1	270	507.8	28930	10	261.3	270	**214.5**	19	163	266.3
7-35-0.6-2	236	227.1	16462	10	231.8	236	**79.9**	8	79	234.4
7-35-0.6-5	306[1]	>1h	213424	5	294.9	306	**154.8**	13	107	302.1
	303.8									
8-32-0.6-3	278	107.3	5230	3	269.1	278	**38.2**	3	41	276.3
8-32-0.6-4	277	500.5	28350	0	267.9	277	**168.8**	37	176	273.4
8-32-0.6-5	243	915.0	47335	9	235.6	243	**73.8**	9	68	240.4
8-40-0.6-1	282	145.3	5491	16	279.7	282	**47.6**	1	42	282.0
8-40-0.6-2	354[1]	>1h	95637	23	330.3	**344**	**166.7**	1	53	343.4
	340.1									
8-40-0.6-3	288	1964.8	84367	13	280.4	288	**295.4**	17	136	286.3
8-40-0.8-1	271	182.5	6442	12	268.5	271	**130.8**	8	90	270.4
8-48-0.6-1	373[1]	>1h	51822	2	356.7	**363**	**596.5**	20	185	361.2
	359.9									
8-48-0.6-4	391	1078.8	25267	2	386.1	391	**554.3**	13	166	389.9
8-48-0.6-5	441	**343.5**	7687	2	432.5	441	354.0	9	121	438.7
8-48-0.8-3	322	**164.0**	3808	0	320.5	322	336.6	12	150	321.4
9-36-0.6-4	312	1290.0	43844	2	299.9	312	**195.6**	19	120	306.1
9-36-0.6-5	248	214.1	6943	7	243.1	248	**91.1**	6	62	247.0
9-36-0.8-1	228	**125.5**	5062	4	225.3	228	183.6	21	140	226.6
9-45-0.6-2	345[1]	>1h	55214	15	329.4	**339**	**364.2**	15	123	336.7
	335.5									
9-54-0.5-3	-[2]	>1h	35219	0	481.7	**504**	**677.3**	11	147	496.4
	495.1									
9-54-0.6-1	435	861.4	11803	0	430.5	435	**781.6**	11	121	432.9
9-54-0.6-2	452	676.4	8466	2	446.9	452	**241.8**	1	60	452.0
9-54-0.6-3	-[2]	>1h	32069	1	410.4	**425**	**1553.2**	33	253	420.1
	417.1									
9-54-0.6-4	-[2]	>1h	28387	0	435.3	**445**	**1952.8**	57	436	440.8
	440.9									
9-54-0.6-5	437	**643.0**	11406	1	431.9	437	952.8	14	164	435.9
9-54-0.8-4	405	1313.0	17652	1	402.7	405	**1193.7**	11	147	403.7

[1] Optimality of the solution is not proven, the best bound is just underneath
[2] A feasible solution is not found, the best bound is just underneath

Table 3.5: The algorithms **BCh-MMS** and **BP-BCh** for the MMASP : further comparison

Test	Obj.	LB	Time	Nds	Cts	XLP
3-12a	7	7.00	0.17	106	0	7.00
3-12b	10	10.00	0.28	157	0	9.00
5-15a	1	1.00	0.44	92	6	1.00
5-15b	7	7.00	4.38	2855	36	3.00
5-20a	4	4.00	5.38	1875	2	3.00
5-20b	5	5.00	16.63	4977	11	4.00
6-24	2[1]	1.00	>300	108800	3	1.00
7-30	8[1]	6.00	>300	15600	34	6.00
8-34	7[1]	6.00	>300	15300	1	6.00

[1] Optimality of the solution is not proven

Table 3.6: Branch-and-check algorithm for the problem $R \mid r_j \mid T_{\max}$

The results for the branch-and-check algorithm are presented in Table 3.6, and for the branch-and-price algorithm — in Table 3.7. Note that the values LB and XLP for the branch-and-check algorithm are integer as the variable t was set to be integer.

Test	Obj	LB	Time	Nds	Its	XLP
3-12a	7	7.00	8.40	6	71	6.41
3-12b	10	10.00	45.41	55	379	8.79
5-15a	1	1.00	6.63	1	38	0.27
5-15b	7[1]	4.25	>300	215	934	4.22
5-20a	4[1]	2.74	>300	117	990	2.60
5-20b	5	5.00	23.20	1	71	4.14
6-24	2	2.00	245.58	17	131	1.16
7-30	10[1]	6.03	>300	24	269	5.97
8-34	14[1]	5.48	>300	7	115	5.48

[1] Optimality of the solution is not proven

Table 3.7: Branch-and-price algorithm for the problem $R \mid r_j \mid T_{\max}$

Overall, the branch-and-check algorithm performs better, as it solved more instances to optimality. However, it did not solve the instance "6-24", whereas the branch-and-price algorithm did. However the latter did not solve more instances and the gap between the best found solution and the best found lower bound is higher in general.

It seems that the branch-and-price algorithm "loses" relative to the branch-and-check algorithm here because the former did not keep the high quality of LP bounds it has when solving the MMASP. Moreover, the LP bound progresses slowly during the tree search. This results sometimes in the large

number of nodes needed to solve the problem, as in the case with the instances "5-15b" and "5-20a". Another observation is that the column generation procedure needs more iterations to converge. Therefore, the time for solving one node increases for the maximum tardiness problem in comparison with the MMASP.

In the branch-and-check algorithm, more infeasibility cuts are generated when solving the maximum tardiness problem than the MMASP. This is due to the fact that the valid inequalities (3.33) are not as strong as (3.12). The larger is the value of variable t, the less strong are the valid inequalities. But still the number of cuts generated is not large for the test instances. The main difficulty with the branch-and-check algorithm is still the quality of the LP relaxation of the master.

Finally, it should be noted that, to the best of our knowledge, these two algorithms are the first approaches proposed in the literature for the problem $R \mid r_j \mid T_{\max}$.

Chapter 4

Minimizing the total weighted tardiness on a single machine

In this chapter we consider the scheduling problem of minimizing the total weighted tardiness on a single machine ($1 \mid\mid \sum w_j T_j$). New MIP formulations are proposed for this problem. We will analyze the quality of lower bounds which can be obtained by solving the LP relaxation of the proposed formulations. Branch-and-cut and branch-and-price algorithms based on these formulations will be also presented.

In Section 4.1, we define the problem and present a short review of the related literature. A time-indexed formulation for the problem is presented in Section 4.2. We will dwell on the drawback of this formulation and ways to overcome it. In Section 4.3, appropriate partitions of the time horizon into intervals are introduced. An algorithm to obtain such a partition is suggested. We make use of this partition in Section 4.4 and propose the interval-indexed MIP formulation for the problem. In Section 4.5, the Dantzig-Wolfe interval-indexed reformulation is presented. We consider a column generation algorithm to solve the LP relaxation of this reformulation. Ways to strengthen the Dantzig-Wolfe reformulation and the Branch-and-Price algorithm are also discussed. In Section 4.6, the results of the numerical study of the MIP formulations proposed are presented.

4.1 Introduction

In this chapter, we study the scheduling problem $1 \mid\mid \sum w_j T_j$. A set of jobs $N = \{1, \ldots, n\}$ has to be processed on a single machine. Only one job can be processed at a time and preemptions are not allowed. Each job $j \in N$ has

a processing time p_j, a due date d_j and a weight w_j. We suppose that all the processing times and all the due dates are integer. The objective is to minimize the total weighted tardiness, i.e. to find a schedule $\pi \in \Pi(N)$, for which the value $\sum_{j \in N} w_j T_j(\pi)$ if minimized.

The problem is $\mathcal{NP}$-hard in the strong sense as shown by Lenstra et al. [55] and Lawler [51]. In the same paper, Lawler proposed a pseudopolynomial algorithm for the problem $1 \mid\mid \sum T_j$ without weights. The latter problem was proved to be NP-hard in the ordinary sense by Du and Leung [31].

The problem to minimize the total weighted tardiness is very well studied due to its theoretical as well as practical interest. Several works have been devoted to dominance rules which help to reduce the search space when solving the problem to optimality. The best known dominance rules were proposed by Emmons [34] for the problem $1 \mid\mid \sum T_j$ and were extended to the case with weights by Rinnooy Kan et al. [69]. Other dominance rules appeared in the works by Akturk and Yildirim [4] and Rachamadugu [70].

Branch-and-bound algorithms for the problem were proposed in the works by Rinnooy Kan et al. [69], Picard and Queyranne [64], Potts and Van Wassenhove [66], Babu et al. [6], Bigras et al. [14], Pan and Shi [62]. The last three algorithms are based on the time-indexed formulation of the single machine scheduling problems. We will review this formulation in the next section. It seems that currently the best algorithms to solve optimally 50-job instances are those by Potts and Van Wassenhove and Babu et al. The algorithm of Pan and Shi is the only one which is able to solve all the 100-job instances from the standard test set (available from the OR-Library [11]).

To solve larger instances, local search heuristics were proposed by Crauwels et al. [28] and Congram et al. [27]. Surveys on the single machine problems with the total weighted tardiness criterion can be found in [1, 78].

4.2 Time-indexed formulation

One of the ways to formulate a single-machine scheduling problem as a MIP is to use a so-called "time-indexed" formulation. We now give the standard time-indexed formulation for the single-machine problem with an arbitrary cost function. We assume that all the processing times of jobs are integers. Let T be the length of the time horizon, i.e. the maximum possible completion time of a job in any schedule. Binary variable x_{jt}, $j \in N$, $t \in [1, T]$, takes value 1 if job j is started at time moment t, and otherwise $x_{jt} = 0$. Let c_{jt} be the cost of scheduling job j at time moment t. Then, the standard time-indexed formulation (TI) is as follows

$$\min \quad \sum_{t=1}^{T} c_{jt} x_{jt} \tag{4.1}$$

(TI)

$$s.t. \quad \sum_{t=1}^{T-p_j+1} x_{jt} = 1, \quad j \in N, \tag{4.2}$$

$$\sum_{j \in N} \sum_{s=\max\{1,t-p_j+1\}}^{t} x_{js} \leq 1, \quad t \in [1,T], \tag{4.3}$$

$$x_{jt} \in \{0,1\}, \quad j \in N,\ t \in [1,T]. \tag{4.4}$$

The constraints (4.2) state that each job starts exactly once. The constraints (4.3) guarantee that, at each time moment, only one job is processed. The problem $1 \mid\mid \sum w_j T_j$ can be modelled using the formulation (TI) by setting $c_{jt} = w_j \cdot \max\{t + p_j - 1 - d_j, 0\}$.

The time-indexed formulation is known for more than 40 years. It was used, for example, in the works by Bowman [19], Pritsker et al. [67], Redwine and Wismer [68]. The polyhedral study of this formulation was conducted by Dyer and Wolsey [32], Sousa and Wolsey [80], Akker et al. [2]. The main advantage of this formulation is that, by solving its LP relaxation, one can obtain a very strong lower bound on the optimal solution value. Another obvious advantage is the possibility to model single-machine non-preemptive problem with any objective function. Unfortunately, the formulation (TI) has one big drawback. For practical instances with a large number of jobs and large processing times, the size of the formulation becomes so large that it is difficult to solve even its LP relaxation in a reasonable time.

To overcome this difficulty, van den Akker et al. [3] proposed to solve a Dantig-Wolfe reformulation (DWTI) of (TI) instead of formulation (TI) itself. They showed that, for instances with large processing times, the LP relaxation of (DWTI) can be solved much faster than the LP relaxation of (TI). However, instances with more than 30 jobs remained intractable.

Bigras et al. [14] extended the applicability of the Dantzing-Wolfe reformulation of (TI) by decomposing the time horizon into several intervals. They showed that this allows one to significantly decrease the solution time for the LP relaxation of (DWTI). Although the quality of the lower bounds decreases after the decomposition, they still remain quite tight.

Babu et al. [6] applied Lagrangian decomposition to the formulation (TI). They introduced duplicate binary variables y_{jt}, $j \in N$, $t \in [1,T]$, and replaced the constraints (4.3) by the following set of constraints

$$\sum_{s=\max\{1,t-p_j+1\}}^{t} x_{js} = y_{jt}, \quad j \in N, t \in [1,T], \tag{4.5}$$

$$\sum_{j \in N} y_{jt} \leq 1, \quad t \in [1,T], \tag{4.6}$$

$$\sum_{t=1}^{T} y_{jt} = p_j, \quad j \in N. \tag{4.7}$$

The variables y_{jt} can be viewed as the number of units of job $j \in N$ processed at time moment $t \in [1,T]$. To obtain a Lagrangian relaxation, the authors relaxed the constraints (4.5). Babu et al. claimed that, by solving this Lagrangian relaxation, for the total weighted tardiness problem, one can obtain lower bounds of a better quality and in an order of magnitude less time than by solving the LP relaxation of (TI). A better quality of lower bounds is achieved by using the structure of the total weighted tardiness problem when solving the Lagrangian relaxation. For instance, one can forbid to schedule a job just after another one, if after interchanging of these jobs, the cost decreases. Our experiments showed that indeed the Lagrangian relaxation lower bound is better than the time-indexed LP relaxation lower bound for some instances. However, in our experiments, the former lower bound was worse on average than the latter one (see Table 4.6).

The use of formulation (TI) is even more problematic when one tries to solve one-machine scheduling problems to optimality. In this case, it is necessary to solve the LP relaxation of (TI) at each node of the search tree. Pan and Shi [62] proposed a method in which the LP relaxation of the time-indexed formulation has to be solved only at the top node. At other nodes of the tree, to obtain a lower bound, one needs to find a solution of the LP relaxation of the transportation formulation by Lawler [49] which can be solved much faster. The difference in quality between the lower bounds obtained by solving the LP relaxation of (TI) and lower bounds obtained using the method of Pan and Shi increases while going deep into the search tree. Nevertheless, this method allows them to solve to optimality difficult 100-jobs instances of the problem $1 \mid\mid \sum w_j T_j$.

In this chapter, we will decompose the time horizon into intervals and use the structure of the total weighted tardiness problem to modify the time-indexed formulation. This modification allows us to obtain lower bounds for the problem with a good trade-off between the time needed to compute them and their quality.

There are differences between our approach and the approach of Bigras et al. [14]. In the latter, the time horizon is partitioned into a certain number of intervals uniformly, whereas we use the structure of the problem to partition the time horizon in a special way. Another difference is that, in our approach, the number of intervals cannot be determined a priory, whereas in the the approach of Bigras et al., one can specify the number of intervals.

4.3 Appropriate partitions of the time horizon

The approach we develop here is based on the idea of defining a small number of intervals such that, given which jobs are completed in which intervals, we know their optimal ordering. This will allow us to develop a MIP formulation in which the binary variables represent the assignment of jobs to intervals. The major advantage of this technique is that the resulting MIP, in which time-indexed variables are replaced by interval-indexed variables, is small and its LP relaxation can be solved fast.

We will show how it is possible to define

i) a set M of "appropriate" intervals $\{I_1, \ldots, I_m\}$,

ii) permutations σ_u, one for each interval $u \in M$, such that if jobs in a set Q are completed within interval I_u, σ_u tells us the order in which they should be processed.

Once these are defined, an assignment of jobs to intervals corresponds to a schedule, and there is an assignment of jobs which corresponds to an optimal schedule.

Example 4.1. We now show an example of the correspondence between an assignment of jobs to intervals and a schedule. We are given 5 jobs. Suppose we defined the partition of the time horizon into 3 intervals $\{I_1, I_2, I_3\}$ and the set σ of permutations of jobs, $\sigma = \{\sigma_1, \sigma_2, \sigma_3\}$, $\sigma_1 = (3, 5, 4, 1, 2)$, $\sigma_2 = (5, 3, 1, 2, 4)$, $\sigma_3 = (3, 5, 1, 4, 2)$. Let jobs 1 and 4 be assigned to interval I_1 and jobs 2, 3 and 5 be assigned to interval I_3. This assignment and the corresponding schedule are depicted in Figure 4.1.

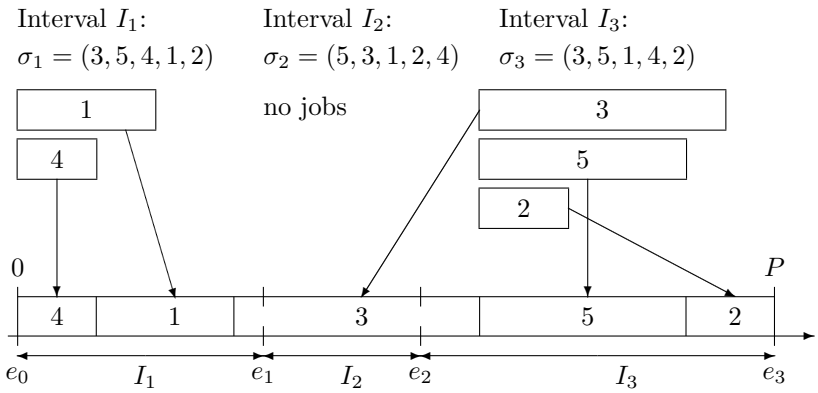


Figure 4.1: Assignment of jobs to intervals and the corresponding schedule

We now make the presentation more formal.

As the total weighted tardiness is a regular criterion and all jobs are available at the time moment zero, we have the following property. Let P be the sum of all processing times: $P = \sum_{j \in N} p_j$.

Proposition 4.1. *There exists an optimal schedule π without idle times and the completion time of the last job in π equals P.*

From now on we assume that the jobs are indexed by nondecreasing due dates. Consider a *partition* $D(e)$, $e \in \mathbb{R}_+^m$, of the time horizon into a set M of intervals. $D(e) = \{I_1, \ldots, I_m\}$, where $I_u = [e_{u-1} + 1, e_u]$ with $e_0 = 0$ and $e_m = P$.

Definition 4.1. Partition $D(e)$ is *based on due dates* if every due date d_j, $j \in N$, equals some e_u, $u \in M$, i.e. $\{d_1, \ldots, d_n\} \subseteq \{e_u\}_{u \in M}$.

We say that job j is *assigned* to interval I_u in schedule π if it is completed in this interval: $C_j(\pi) \in I_u$. Job j is on-time in interval I_u if $d_j \geq e_u$, and j is late in I_u if $d_j \leq e_{u-1}$. Let $Q_u(\pi)$ be the set of jobs completed in interval I_u in schedule π. Note that, if $D(e)$ is a partition based on due dates, job $j \in N$ is either on-time or late in interval I_u, $u \in M$.

With $m =\mid M \mid$, we define a set $\sigma = \{\sigma_1, \ldots, \sigma_m\}$ of m permutations of $\{1, 2, \ldots, n\}$.

Definition 4.2. Given a partition $D(e) = \{I_1, \ldots, I_m\}$, σ is an *appropriate* set of permutations for $D(e)$ if there exists an optimal schedule π^* in which, for any interval I_u, $u \in M$, and any two jobs $i, j \in Q_u(\pi)$, job i is sequenced before job j when $\sigma_u(i) < \sigma_u(j)$.

We now give examples of partitions and appropriate sets of permutations for them.

Example 4.2. Consider the problem $1 \mid\mid \sum w_j C_j$, a special case of $1 \mid\mid \sum w_j T_j$ ($d_j = 0$, $\forall j \in N$). In this case, there is one single interval $I_1 = [1, P]$ and the permutation $\sigma_1 = \{j_1, \ldots, j_n\}$ corresponding to Smith's rule ($p_{j_{k-1}}/w_{j_{k-1}} \leq p_{j_k}/w_{j_k}$, $1 < k \leq n$) is appropriate for I_1.

Example 4.3. Consider the problem $1 \mid p_j = p \mid \sum w_j T_j$ in which all processing times are equal to p. For a partition based on due dates, the following set of permutations σ is appropriate. In σ_u, $u \in M$, we put first late jobs in I_u ordered according to Smith's rule, then we put on-time jobs in I_u in any order. We now show that there is an optimal schedule in which jobs are processed according to σ. Consider an optimal schedule π. For each $u \in M$, we put jobs in $Q_u(\pi)$ according to σ_u by necessary interchanges of adjacent jobs. As all processing times are equal, every job is now completed within the same interval as before interchanges. Therefore, after interchanges, late jobs remain late and on-time jobs remain on-time, and the cost of the schedule does not increase.

Example 4.4. Consider the partition $D(e) = \{I_u = [u, u]\}_{u \in \{1,2,\ldots,P\}}$ for the problem $1 \mid\mid \sum w_j T_j$. Any set $\{\sigma_1, \sigma_2, \ldots, \sigma_P\}$ of permutations is appropriate for this partition, as at most one job can be assigned to an interval.

It is possible to obtain an appropriate set of permutations for any partition of the time horizon. One just needs to take an optimal schedule π^* and put jobs in all permutations according to the order in which they are processed in π^*. But this approach is meaningless as we need to solve the problem first. Here we are interested in partitions for which it is possible to compute an appropriate set of permutations efficiently.

Definition 4.3. A partition $D(e)$ is *appropriate* if it is possible to compute an appropriate set of permutations for it in a polynomial time.

Now we will show how one can obtain an appropriate partition of the time horizon into a relatively small number of intervals. From now on we concentrate on partitions which are based on due dates. Let $\bar{\sigma}$ be a set of permutations for such a partition. We say that $\bar{\sigma}$ satisfies the *WSPT+LPT rule* (Weighted Shortest Processing Time + Longest Processing Time) if, in every permutation $\bar{\sigma}_u$, $u \in M$, first there are late jobs in I_u ordered according to Smith's rule and then there are on-time jobs I_u ordered according to LPT rule. Additionally, late jobs in I_u with the same WSPT ratio are also ordered according to LPT rule. Formally, $\bar{\sigma}$ satisfies WSPT+LPT rule if the following conditions hold for each pair of jobs $i, j \in N$ and each $u \in M$.

- If job i is late in I_u and job j is on-time in I_u then $\bar{\sigma}_u(i) < \bar{\sigma}_u(j)$.
- If jobs i and j are both on-time in I_u and $p_i > p_j$ then $\bar{\sigma}_u(i) < \bar{\sigma}_u(j)$.
- If jobs i and j are both late in I_u and $p_i/w_i < p_j/w_j$ then $\bar{\sigma}_u(i) < \bar{\sigma}_u(j)$.
- If jobs i and j are both late in I_u and $p_i/w_i = p_j/w_j$ and $p_i > p_j$ then $\bar{\sigma}_u(i) < \bar{\sigma}_u(j)$.

Intuitively, a set of permutations satisfying the WSPT+LPT rule should be appropriate for a partition based on due dates. But, in the next example, we show that this is not true.

Example 4.5. Consider the 3-job instance shown in Table 4.1 and the partition $D(e)$, $e = 5, 9, 16$. There is only one optimal schedule $\pi^* = (2, 1, 3)$, see Figure 4.2. In π^*, all the jobs are completed in interval $I_3 = [10, 16]$, but the permutation $\bar{\sigma}_3$ satisfying the WSPT+LPT rule is $(1, 3, 2)$.

Actually, a set of permutations $\bar{\sigma}$ satisfying the WSPT+LPT rule is "almost" appropriate. "Almost" means that there is always an optimal schedule in which, for each $u \in M$, all the jobs completed in I_u, except for maybe one job f, are processed according to $\bar{\sigma}_u$. This exception can occur only if job f is late in I_u and is completed first in I_u.

j	1	2	3
p_j	4	10	2
d_j	9	5	9
w_j	2	3	0.8

Table 4.1: The data for Example 4.5

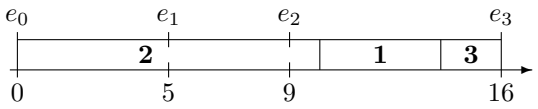


Figure 4.2: A partition based on due dates and the optimal schedule for Example 4.5

The next lemma proves that a set of permutations satisfying the WSPT+LPT rule is "almost" appropriate and gives conditions when the exception described above occurs. We denote $C_{ij}(\pi) = \max\{C_i(\pi), C_j(\pi)\}$.

Lemma 4.2. *Let $D(e)$ be a partition based on due dates and $\bar{\sigma}$ be a set of permutations for $D(e)$ satisfying the WSPT+LPT rule. Then there exists an optimal schedule π such that, for each $u \in M$,*

- *either jobs in $Q_u(\pi)$ are processed according to $\bar{\sigma}_u$,*
- *or there are two late jobs $f, s \in Q_u(\pi)$ processed first and second among jobs in $Q_u(\pi)$ such that $p_f/w_f \geq p_s/w_s$,*

$$C_s(\pi) \leq d_s + \frac{p_s w_f}{w_s} \tag{4.8}$$

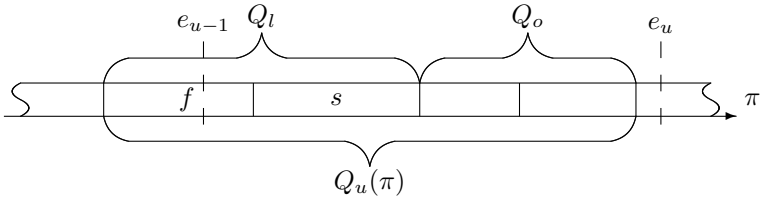
and jobs in $Q_u(\pi) \setminus \{f\}$ are processed according to $\bar{\sigma}_u$.

Proof. Let π be an optimal schedule and, suppose that for some interval $u \in M$, jobs in $Q_u(\pi)$ are not processed according to the conditions of this lemma. We now reschedule jobs in $Q_u(\pi)$ according to the conditions of the lemma while keeping π optimal.

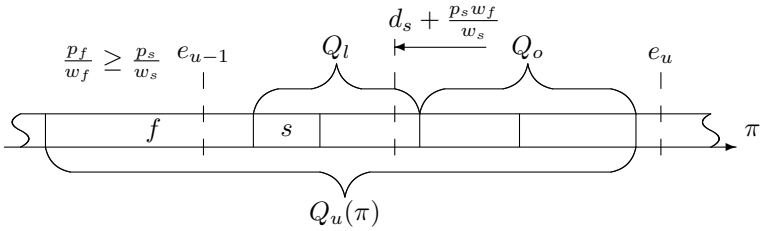
First, let subset Q_o include all the on-time jobs in $Q_u(\pi)$. All jobs in Q_o are processed last among jobs in $Q_u(\pi)$. Otherwise, by putting a job in Q_o after a late job in $Q_u(\pi)$, we would decrease the cost of π which is impossible as π is optimal. We can reschedule jobs in Q_o according to permutation $\bar{\sigma}_u$ without increasing the cost.

We now consider two cases.

1. If there is at most one late job in $Q_u(\pi)$ or, for two late jobs $f, s \in Q_u(\pi)$ processed first and second in I_u, we have $p_f/w_f < p_s/w_s$ or (4.8) is false, then let subset Q_l include all the late jobs in $Q_u(\pi)$.



2. Otherwise, let subset Q_l include all the late jobs in $Q_u(\pi) \setminus \{f\}$. Note that, in this case, we have $p_f/w_f \geq p_s/w_s$ and (4.8) is true.



Consider a pair of neighboring jobs $i, j \in Q_l$, $p_i/w_i < p_j/w_j$. Now we prove by contradiction that i precedes j in π. Suppose that j precedes i in π. Consider schedule π_{ij} obtained from π by interchanging i and j. We compare the costs of π and π_{ij}.

$$\begin{aligned}
&\sum_{k\in N} w_k T_k(\pi_{ij}) - \sum_{k\in N} w_k T_k(\pi) = \\
&w_i T_i(\pi_{ij}) + w_j T_j(\pi_{ij}) - w_i T_i(\pi) - w_j T_j(\pi) = \\
&w_i \cdot \max\{C_{ij}(\pi_{ij}) - p_j - d_i, 0\} + w_j(C_{ij}(\pi_{ij}) - d_j) - \\
&w_i(C_{ij}(\pi) - d_i) - w_j(C_{ij}(\pi) - p_i - d_j) = \\
&w_i \cdot \max\{C_{ij}(\pi_{ij}) - p_j - d_i, 0\} - w_i(C_{ij}(\pi) - d_i) + w_j p_i \qquad (4.9)
\end{aligned}$$

as $C_{ij}(\pi_{ij}) = C_{ij}(\pi)$, i, j are both late in π and j is late in π_{ij}. If $C_{ij}(\pi_{ij}) - p_j - d_i \geq 0$ then

$$(4.9) = -w_i p_j + w_j p_i < 0. \qquad (4.10)$$

Otherwise $C_{ij}(\pi_{ij}) - p_j - d_i < 0$. Then job i is on-time in π_{ij} and $i \notin Q_u(\pi_{ij})$. This can happen only when jobs j, i are processed first and second in I_u in π, and we are in case 1. Therefore, (4.8) is false, i.e. $C_{ij}(\pi) > d_i + (p_i w_j)/w_i$, and

$$(4.9) = -w_i C_{ij}(\pi) + w_i d_i + w_j p_i < -w_i d_i - p_i w_j + w_i d_i + w_j p_i = 0, \qquad (4.11)$$

We have obtained that the cost of π_{ij} is smaller than the cost of π. This contradicts the optimality of π. So, we have proved that i precedes j in π.

We have shown that the order of jobs in Q_l can differ from permutation $\bar{\sigma}_u$ only for neighboring jobs $i, j \in Q_l$, such that $p_i/w_i = p_j/w_j$. In this case, we can interchange these jobs without increasing the cost.

After necessary interchanges of jobs, we have rescheduled jobs in $Q_o \cup Q_l$ according to permutation $\bar{\sigma}_u$. Note that, if we interchange jobs, the longer job is always put before the shorter one, as interchanged jobs are ordered according to LPT rule in $\bar{\sigma}_u$. Therefore, jobs in $Q_u(\pi)$ remain assigned to interval I_u.

Case 1. Here $Q_u(\pi) = Q_o \cup Q_l$. Therefore, jobs in $Q_u(\pi)$ are now processed according to $\bar{\sigma}_u$, and the rescheduling is done.

Case 2. Here $Q_o \cup Q_l = Q_u(\pi) \setminus \{f\}$. Therefore, jobs in $Q_u(\pi) \setminus \{f\}$ are processed according to $\bar{\sigma}_u$. Let job s' be processed second among jobs in $Q_u(\pi)$. We know that $p_f/w_f \geq p_{s'}/w_{s'}$. If (4.8) is true for f and s', then the rescheduling is done. Otherwise, we have moved to case 1 and, as it has been shown above, there is a possibility to convert schedule π into an optimal schedule as desired.

So, for each interval $u \in M$, it is possible to reschedule jobs in $Q_u(\pi)$ according to the condition of the lemma while keeping jobs assigned to I_u. Repeated interval by interval, it is possible to convert π into another optimal schedule satisfying the conditions of the lemma. □

Using Lemma 4.2, in the next theorem, we will determine partitions for which is it possible to easily derive an appropriate set of permutations from a set satisfying the WSPT+LPT rule.

Theorem 4.3. *A partition $D(e)$ based on due dates is appropriate if, for each $u \in M$ and each pair of jobs $(i, j) \in N$ with $p_i/w_i \leq p_j/w_j$, at least one of the following two conditions is true.*

$$e_u \leq e_{u-1} + p_j, \tag{4.12}$$

$$e_{u-1} \geq d_i + \left\lceil \frac{w_j p_i}{w_i} \right\rceil - p_i. \tag{4.13}$$

Proof. Let $\bar{\sigma}$ be a set of permutations for $D(e)$, and $\bar{\sigma}$ satisfies the WSPT+LPT rule. For each $u \in M$, we convert permutation $\bar{\sigma}_u$ into σ_u by moving all "long" jobs $j \in N$, satisfying (4.12), to the beginning of the permutation:

$$\bar{\sigma}_u = (i_1, \ldots, j_1, \ldots, j_k, \ldots, i_n) \rightarrow (j_1, \ldots, j_k, i_1, \ldots, i_n) = \sigma_u.$$

Obviously, σ can be obtained in polynomial time. To prove the theorem, it remains to show that σ is appropriate for partition $D(e)$.

Let π be an optimal schedule satisfying the conditions of Lemma 4.2. Such a schedule exists, as $D(e)$ is a partition based on due dates and $\bar{\sigma}$ satisfies the WSPT+LPT rule. We will now show that, for all $u \in M$, jobs in $Q_u(\pi)$ are processed according to permutation σ_u. This will conclude the proof.

Consider an interval $u \in M$. We have two cases. In the first case, jobs in $Q_u(\pi)$ are processed according to permutation $\bar{\sigma}_u$. There can be only one job j in $Q_u(\pi)$ satisfying (4.12). Such a job j is processed first among jobs in $Q_u(\pi)$, because it is started before the beginning of interval I_u. Therefore, jobs in $Q_u(\pi)$ are processed also according to permutation σ_u.

In the second case, jobs in $Q_u(\pi)$ are not processed according to permutation $\bar{\sigma}_u$. Therefore, by Lemma 4.2, there are two late jobs $f, s \in Q_u(\pi)$ processed first and second in I_u such that $p_f/w_f \geq p_s/w_s$ and (4.8) is true. Then (4.13) is false for the pair of jobs (s, f). Otherwise, we would have

$$C_s(\pi) - p_s \overset{(4.8)}{\leq} d_s + \frac{w_f p_s}{w_s} - p_s \leq d_s + \left\lceil \frac{w_f p_s}{w_s} \right\rceil - p_s \overset{(4.13)}{\leq} e_{u-1},$$

which would mean that $f \notin Q_u(\pi)$. As (4.13) is false for pair of jobs (s, t), (4.12) should be true for job f. For all jobs in $Q_u(\pi) \setminus \{f\}$, (4.12) is false, as they are not processed first in $Q_u(\pi)$. So, job f precedes all jobs in $Q_u(\pi)\setminus\{f\}$ in σ_u. By Lemma 4.2, jobs in $Q_u(\pi) \setminus \{f\}$ are processed according to $\bar{\sigma}_u$, and therefore according to σ_u. We can conclude that jobs in $Q_u(\pi)$ are processed according to permutation σ_u. □

We now present an algorithm for finding an appropriate partition which satisfies the conditions of Theorem 4.3 and has the smallest number of intervals. The idea of the algorithm is the following. First, we take the partition based on due dates with the smallest number of intervals. Then, by further dividing some intervals, this partition is converted to the partition satisfying the conditions of Theorem 4.3 with the smallest number of intervals.

In the first stage of the algorithm, we obtain the partition based on due dates that has the smallest number of intervals. This partition $D(e)$ is unique. In $D(e)$, each value e_u, $u \in M$, corresponds to a due date: $\{e_u\}_{u\in M} = \{d_j\}_{j\in N, d_j \leq P}$.

We say that a pair of jobs $(i, j) \in N$ with $p_i/w_i \leq p_j/w_j$, is *special* for interval I_u, $u \in M$, if both conditions (4.12) and (4.13) are false. A partition satisfying the conditions of Theorem 4.3 does not have special pairs for any interval.

In the second stage of the algorithm, we further partition some intervals. To give an idea on how we do that, suppose first that there is only one special pair (i, j) for I_u. Let

$$H_{ij} = \left[d_i + \left\lceil \frac{w_j p_i}{w_i} \right\rceil - p_i,\ e_{u-1} + p_j \right] = [h^l_{ij}, h^r_{ij}].$$

In order to "get rid of" the special pair, interval I_u is divided into intervals $I^{(1)}_u = [e_{u-1}, t_1]$ and $I^{(2)}_u = [t_1 + 1, e_u]$, where $t_1 \in H_{ij}$. Then we have $t_1 \leq e_{u-1} + p_j$, $t_1 \geq h^l_{ij}$, and pair (i, j) is not special neither for $I^{(1)}_u$ nor for $I^{(2)}_u$.

We now consider the general case, in which interval I_u has a set B of special pairs of jobs (i_b, j_b), $b \in B$. Then we search for a partition of I_u into a set $K = \{1, \ldots, k\}$ of intervals $I^{(v)}_u = [t_{v-1} + 1, t_v]$, $v \in K$, $t_0 = e_{u-1}$, $t_k = e_u$. New intervals should not have special pairs of jobs, i.e. we should have

$$t_v \leq t_{v-1} + p_{j_b} \ \text{ or } \ t_{v-1} \geq h^l_{i_b j_b}, \quad \forall v \in K, \quad \forall b \in B,$$

i.e. for each special pair (i_b, j_b), interval $H_{i_b j_b}$ should contain at least one point t_v, $v \in K$. Moreover, we need to minimize the number k of new intervals.

In Figure 4.3, we depict an example of an optimal partition of interval I_u which has 6 special pairs of jobs. Notice that, after determining t_1, intervals $H_{i_2 j_2}$, $H_{i_3 j_3}$, $H_{i_5 j_5}$ and $H_{i_6 j_6}$ uncovered by t_1 increase by $t_1 - e_{u-1}$.

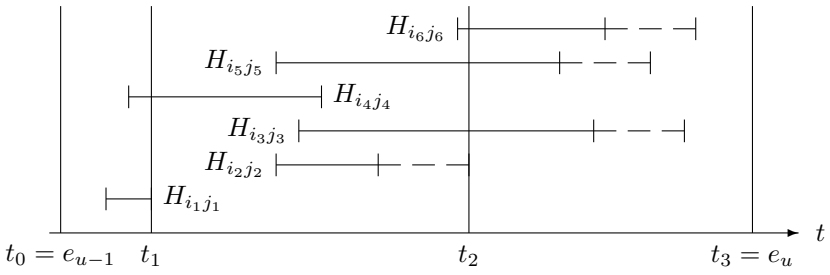


Figure 4.3: A partition of I_u minimizing k ($k = 3$)

An algorithm which partitions an interval containing special pairs of jobs and minimizes k is quite simple. We start with the empty set B' of covered intervals $H_{i_b j_b}$. In each iteration k, we choose the interval $H_{i_{b'} j_{b'}}$, $b' \in B \setminus B'$, with the smallest right border and set $t_k = h^r_{i_{b'} j_{b'}}$. Then we add intervals covered by t_k to set B'. Next, right borders of intervals in $b \in B \setminus B'$ are increased by $t_k - t_{k-1}$, and we pass to the next iteration. We continue until set $B \setminus B'$ becomes empty. This algorithm is formally presented in Algorithm 4.1.

Proposition 4.4. *Algorithm 4.1 is correct.*

Proof. Consider the interval $H_{i_{b'} j_{b'}}$ with the smallest right border $h^r_{i_{b'} j_{b'}}$. One of the time moments t' should be contained in interval $H_{i_{b'} j_{b'}}$. Then the optimal number of time moments equals the sum of one and the optimal number of time moments of the subproblem having set $B \setminus B'$ of intervals, where B' is the set of intervals containing t'. To prove the proposition it suffices to show that, if $t' = h^r_{i_{b'} j_{b'}}$, the optimal solution of the subproblem is minimum among the subproblems for all possible $t' \in H_{i_{b'} j_{b'}}$. This is easy to see, as for such t', the cardinality of set $B \setminus B'$ is minimum, and intervals in $B \setminus B'$ contain the corresponding intervals in subproblems for other possible values of t'. □

This algorithm is an extension of the algorithm by Finke et al. [36] for the minimum clique cover problem on an interval graph. In an interval graph, each vertex represents an interval. Two vertices are adjacent if the corresponding intervals intersect. A clique in a graph is a set of pairwise adjacent vertices. In an interval graph, every time moment t corresponds to a clique, i.e. a set of intervals which contain t. In the minimum clique cover problem, one needs

Algorithm 4.1 Algorithm for partitioning interval I_u which have set B of special pairs of jobs (i_b, j_b), $b \in B$

1: $B' := \emptyset$; $t_0 := e_{u-1}$; $k := 0$
2: **for all** $b \in B$ **do**
3: $\quad h^r_{i_b j_b} := t_0 + p_{j_b}$
4: $\quad h^l_{i_b j_b} := d_{i_b} + \left\lceil \frac{w_{j_b} p_{i_b}}{w_{i_b}} \right\rceil - p_{i_b}$;
5: **end for**
6: **while** $B' \neq B$ **do**
7: $\quad b' := \arg\min_{b \in B' \setminus B} \{h^r_{i_b j_b}\}$
8: $\quad k := k + 1$; $t_k := h^r_{i_{b'} j_{b'}}$
9: $\quad$ **for all** $b \in B' \setminus B$ **do**
10: $\quad\quad$ **if** $t_k \in [h^l_{i_b j_b}, h^r_{i_b j_b}]$ **then**
11: $\quad\quad\quad B' := B' \cup \{b\}$
12: $\quad\quad$ **else**
13: $\quad\quad\quad h^r_{i_b j_b} := h^r_{i_b j_b} + t_k - t_{k-1}$
14: $\quad\quad$ **end if**
15: $\quad$ **end for**
16: **end while**

to find the minimum number of cliques such that every vertex is contained in at least one clique. In an interval graph, this is equivalent to the problem to find the minimum number of time moments such that every interval contains at least one. Our problem to partition an interval with special pairs of jobs is a generalization of the latter problem in the sense that, when a time moment is chosen, some intervals increase.

The formal procedure for obtaining an appropriate partition of the time horizon is presented in Algorithm 4.2. As the input parameter this procedure has the set N of jobs. Note that the number of intervals obtained by the procedure does not exceed $n^2/2 + n$, i.e. the sum of all the pairs of jobs and the maximum number of different due dates.

j	1	2	3	4
p_j	3	13	4	8
d_j	10	10	6	25
w_j	2	8	4	10

Table 4.2: The data for Example 4.6

Example 4.6. Consider the 4-job instance, shown in Table 4.2. The partition based on the due dates with the smallest number of intervals is $D'(e)$ where $e = \{6, 10, 25, 28\}$. Intervals I_1, I_2, I_4 do not have special pairs. Interval I_3 has one special pair $(1, 2)$, as $H_{1,2} = [19, 23] \subset [11, 25] = I_3$ (see Figure 4.4). By dividing interval I_3 into two intervals $I_3^{(1)} = [11, 23]$ and $I_2^{(2)} = [24, 25]$, we ob-

Algorithm 4.2 The procedure **Find_Appropriate_Division**(N)

1: $\bar{d}_0 := e_0 := 0$; $P := \sum_{j \in N} p_j$; $m := 0$; $d_{n+1} := P$
2: **for** $j := 1$ to $n+1$ **do**
3: $\quad \bar{d}_j := \min\{d_j, P\}$
4: $\quad$ **if** $\bar{d}_j > \bar{d}_{j-1}$ **then**
5: $\quad\quad m := m+1$; $e_m := \bar{d}_j$
6: $\quad$ **end if**
7: **end for**
8: $u := 1$
9: **while** $u \leq m$ **do**
10: $\quad B := \emptyset$; $k := 1$
11: $\quad$ **for all** $(i,j) \in N$ **do**
12: $\quad\quad$ **if** $H_{ij} \subseteq [e_{u-1}+1, e_u - 1]$ **then**
13: $\quad\quad\quad B := B \cup \{H_{ij}\}$
14: $\quad\quad$ **end if**
15: $\quad$ **end for**
16: $\quad$ **if** $B \neq \emptyset$ **then**
17: $\quad\quad$ run Algorithm 4.1 and obtain $(t_1, \ldots, t_k)$
18: $\quad\quad m := m + k - 1$
19: $\quad\quad$ **for** $v := m$ **downto** u **do**
20: $\quad\quad\quad e_v := e_{v-k+1}$
21: $\quad\quad$ **end for**
22: $\quad\quad$ **for** $v := 1$ **downto** $k-1$ **do**
23: $\quad\quad\quad e_{v+u-1} := t_v$
24: $\quad\quad$ **end for**
25: $\quad$ **end if**
26: $\quad u := u + k$
27: **end while**
28: **return** $\{e_u\}$, $u \in 1, \ldots, m$.

tain division $D(e)$ where $e = \{6, 10, 23, 25, 28\}$. The set $\bar{\sigma}$ of permutations satisfying the WSPT+LPT rule is the following. $\bar{\sigma}_1 = (4,3,2,1)$, $\bar{\sigma}_2 = (3,2,4,1)$, $\bar{\sigma}_3 = (3,1,2,4)$, $\bar{\sigma}_4 = (3,1,2,4)$, $\bar{\sigma}_5 = (4,3,1,2)$. Now an appropriate set σ of permutations is built according to Theorem 4.3: $\sigma_1 = (4,2,3,1)$, $\sigma_2 = (2,4,3,1)$, $\sigma_3 = (3,1,2,4)$, $\sigma_4 = (3,1,2,4)$, $\sigma_5 = (4,3,1,2)$.

4.4 Interval-indexed MIP formulation

In the previous section, we have shown how to obtain an appropriate partition of the time horizon. Here we present a MIP formulation which is based on such a partition. The main decision variables are *assignment* variables. They determine to which interval each job is assigned. Using this assignment and an appropriate set of permutations, the corresponding schedule is constructed.

Figure 4.4: A partition based on due dates for Example 4.6

Note that an assignment can be infeasible, for instance, when the sum of processing times of jobs assigned to the first u intervals exceeds e_u. From the definition of an appropriate set of permutations we know that there exists an optimal schedule which corresponds to an assignment.

Consider a partition $D(e) = \{I_1, \ldots, I_m\}$ based on due dates and an appropriate set σ of permutations for it. Let B_j^u and A_j^u, $j \in N$, $u \in M$, be the sets of jobs which come, respectively, before and after job j in permutation σ_u. We also denote τ_j^u as the minimum tardiness of job j if it is assigned to interval u:

$$\tau_j^u = \begin{cases} e_{u-1} + 1 - d_j, & j \text{ is late in } I_u, \\ 0, & j \text{ is on-time in } I_u. \end{cases} \tag{4.14}$$

The assignment binary variable z_j^u, $j \in N$, $u \in M$, takes value 1 if job j is completed in interval I_u or earlier, and otherwise $z_j^u = 0$. For each $j \in N$, we set $z_j^0 = 0$ and $z_j^m = 1$. The integer variable $\tilde{T}_j$, $j \in N$, denotes the difference between the tardiness T_j of job j and the value τ_j^u, where u is the index of the interval to which job j is assigned. Note that, when job j is assigned to interval I_u, we have $z_j^u - z_j^{u-1} = 1$. Therefore,

$$\tilde{T}_j = T_j - \sum_{u \in M} \tau_j^u (z_j^u - z_j^{u-1}). \tag{4.15}$$

We now give the interval-indexed formulation.

$$\min \quad \sum_{j \in N} w_j \cdot \left(\tilde{T}_j + \sum_{u \in M} \tau_j^u (z_j^u - z_j^{u-1}) \right) \tag{4.16}$$

$$s.t. \quad \sum_{j \in N} p_j z_j^u \le e_u, \quad \forall u \in M, \tag{4.17}$$

$$z_j^{u-1} \le z_j^u, \quad \forall j \in N,\ \forall u \in M, \tag{4.18}$$

(IIF)

$$\tilde{T}_j \ge p_j z_j^u + \sum_{i \in A_j^u} p_i z_i^{u-1} + \sum_{i \in B_j^u} p_i z_i^u - (e_{u-1} + 1) - (1 - z_j^u + z_j^{u-1}) \cdot \min\{e_u - e_{u-1} - 1, \sum_{i \in B_j^u} p_j - 1\}, \quad \forall j \in N,\ \forall u \in M,\ d_j \le e_{u-1}, \tag{4.19}$$

$$z_j^u \in \{0, 1\}, \quad \forall j \in N, \forall u \in M, \tag{4.20}$$

$$\tilde{T}_j \in \mathbb{Z}_+, \quad \forall j \in N. \tag{4.21}$$

The objective function (4.16) follows from (4.15). The constraints (4.17) guarantee that schedule is feasible, i.e. the sum of the processing times of jobs assigned to the first u intervals is not more than e_u, the total length of these intervals. The inequalities (4.18) ensure that the values taken by z are consistent with the meaning of these variable. The constraints (4.19) relate variables z and $\tilde{T}$. Note that these constraints are defined only when $d_j \leq e_{u-1}$, in other words, when job j is late in interval I_u. In Proposition 4.5, the validity of these constraints is proved.

We have also tried an analogous formulation where the binary variables x_j^u (job j is assigned to interval I_u or not) instead of variables y_j^u, but the formulation (IIF) was much more efficient.

We now make some definitions. Let set B_j, $j \in N$, include job j and jobs which precede j in the schedule corresponding to the solution. Let also $\underline{T}_j^u$ be the value of the right-hand side of the constraint (4.19) for given j and u.

Proposition 4.5. *If the set σ of permutations is appropriate then the formulation (IIF) is correct.*

Proof. To prove the proposition, for each $j \in N$ and for each $u \in M$ such that jobs j is late in I_u, we show the following.

1. If job j is assigned to interval I_u, then $\underline{T}_j^u = \sum_{i \in B_j} p_i - (e_{u-1} + 1)$.
2. If job j is not assigned to interval I_u, then $\underline{T}_j^u \leq 0$.

This would imply (4.15) and the validity of the constraints (4.19).

Case 1. Suppose job j is assigned to interval I_u. Then $z_j^u - z_j^{u-1} = 1$ and

$$\begin{aligned} \underline{T}_j^u &= p_j z_j^u + \sum_{i \in A_j^u} p_i z_i^{u-1} + \sum_{i \in B_j^u} p_i z_i^u - (e_{u-1} + 1) = \\ &= \sum_{i \in N} p_i z_i^{u-1} + \sum_{i \in B_j^u \cup \{j\}} p_i \cdot (z_i^u - z_i^{u-1}) - (e_{u-1} + 1) = \\ &= \sum_{i \in B_j} p_i - (e_{u-1} + 1). \end{aligned} \tag{4.22}$$

Case 2. Suppose job j is not assigned to interval I_u. Then $z_j^u - z_j^{u-1} = 0$ and

$$\underline{T}_j^u = p_j z_j^u + \sum_{i \in A_j^u} p_i z_i^{u-1} + \sum_{i \in B_j^u} p_i z_i^u - \min\{e_u, \sum_{i \in B_j^u} p_j + e_{u-1}\}.$$

If $e_u < \sum_{i \in B_j^u} p_j + e_{u-1}$ then

$$\underline{T}_j^u \leq \sum_{i \in N} p_i z_i^u - e_u \overset{(4.17)}{\leq} 0,$$

otherwise

$$\begin{aligned}\underline{T}_j^u &= p_j z_j^u + \sum_{i\in A_j^u} p_i z_i^{u-1} + \sum_{i\in B_j^u} p_i z_i^u - \sum_{i\in B_j^u} p_j - e_{u-1} \leq \\ &\leq \sum_{i\in A_j^u\cup\{j\}} p_i z_i^{u-1} - e_{u-1} \overset{(4.17)}{\leq} 0.\end{aligned}$$

□

Now we mention some ways to tighten the formulation (IIF). First, global precedence relations can be determined using dominance rules. If job i precedes job j in an optimal schedule ($i \to j$) then the following inequalities can be added:

$$z_i^u \geq z_j^u, \quad \forall u \in M. \tag{4.23}$$

Then, sometimes we can reduce the domains of the jobs. Remember that domain D_j of job $j \in N$ is the set of possible values for the completion time of j, initially $D_j = [p_j, P]$. Domains can be reduced based on dominance rules. Let $\mathcal{B}_j$ and $\mathcal{A}_j$ be the sets of jobs that should be processed before and after job j by dominance rules. Then job j cannot be started before all the jobs in $\mathcal{B}_j$ are finished and should be completed before all the jobs in $\mathcal{A}_j$ are started:

$$D_j := D_j \Big\backslash \left(\left[p_j, \sum_{i\in\mathcal{B}_j} p_i + p_j - 1 \right] \bigcup \left[P - \sum_{i\in\mathcal{A}_j} p_i + 1, P \right] \right).$$

Constraint propagation can also be used to reduce domains of jobs.

Using the domains of jobs, we can add the following constraints:

$$\begin{aligned} &z_j^u - z_j^{u-1} = 0, \quad I^u \cap D_j = \emptyset, j \in N, u \in M, \\ &z_j^u - z_j^{u-1} = 1, \quad D_j \subseteq I_u, j \in N, u \in M. \end{aligned} \tag{4.24}$$

Note that dominance rules should be used carefully, as some of them may be inconsistent with the set σ of permutations used for building the formulation proposed here. In other worlds, there might be no optimal schedule where jobs are processed according to σ and, at the same time, satisfying the dominance rules.

However, dominance rules can be safely used to restrict the set of intervals to which a job can be assigned. In other words, if a complete interval is dominated, it can be excluded from the domain of a job, but a part of an interval cannot be eliminated from the domain. Then there exists an optimal schedule where jobs are processed according to σ and within their domains. We show this now.

Let σ be an appropriate set of permutations obtained according to Theorem 4.3. Suppose there is an optimal schedule π which satisfies dominance rules and jobs in $Q_u(\pi)$, $u \in M$, are not processed according to σ_u. Under

the agreement, interval I_u is completely included in the domain of each job in $Q_u(\pi)$. As in Lemma 4.2, it is possible to reschedule jobs in $Q_u(\pi)$ according to the condition of this lemma without increasing the cost while keeping jobs assigned to I_u, i.e. keeping jobs within their domains. By Theorem 4.3, jobs in $Q_u(\pi)$ are now processed according to σ_u.

For example, it is safe to use the constraints (4.23) and (4.24), as they operate only with whole intervals and not with their parts.

Consider now a vector $v = (v_1, \ldots, v_n)$ of interval indices, $v_j \in \{0, 1, \ldots, m\}$, $j \in N$. For a schedule $\pi \in \Pi(N)$, let $G_v(\pi)$ be the set of jobs such that $j \in G_v(\pi)$ if and only if j is assigned to one of the first v_j intervals in π. Let also n_v be the maximum number of jobs in all sets $G_v(\pi)$, $\pi \in \Pi(N)$. Formally,

$$G_v(\pi) = \{j \in N : \ C_j(\pi) \leq e_{v_j}\} \ \text{ and } \ n_v = \max_{\pi \in \Pi(N)} \mid G_v(\pi) \mid .$$

n_v can be found by solving an instance of the problem $1 \mid\mid \sum U_j$ in $O(n \log n)$ time [60], in which we set $d_j = e_{v_j}$, $j \in N$. Using the numbers n_v, $v \in \{0, 1, \ldots, m\}^n$, we can derive the following valid inequalities:

$$\sum_{j \in N} z_j^{v_j} \leq n_v.$$

4.5 Non-compact formulation

The quality of the lower bounds produced by the LP relaxation of (IIF) does not allow one to solve the interval-indexed formulation by branch-and-cut.

In this section, we give another MIP formulation of the problem. The LP relaxation of this formulation produces lower bounds of a better quality. Consider again a partition $D(e)$ based on due dates and an appropriate set σ of permutations for $D(e)$.

Let the set Ω_u, $u \in M$, include all partial schedules starting not later than e_{u-1} where jobs are completed in interval I_u and processed according to permutation σ_u. Each set Ω_u, $u \in M \setminus \{m\}$, also includes empty partial schedules, i.e. the schedules containing no jobs. For each job $j \in N$, $a_{j\omega} = 1$ if partial schedule ω contains job j, otherwise $a_{j\omega} = 0$. Let c_ω, $\omega \in \Omega_u$, be the nonnegative difference between e_{u-1} and the starting time of ω. Let w_ω, $\omega \in \Omega_u$, be the total cost of ω: $w_\omega = \sum_{j \in N} a_{j\omega} w_j T_j(\omega)$. The sum of processing times of all jobs included in ω we denote p_ω: $p_\omega = \sum_{j \in N} a_{j\omega} p_j$.

In Figure 4.5 an example of constructing a full schedule from partial schedules is depicted. The time horizon is partitioned into 4 intervals. Partial schedule $\omega_1 = (4)$ belongs to set Ω_1, $\omega_2 = (3, 1)$ belongs to Ω_2, empty partial schedule ω_3 belongs to Ω_3, $\omega_4 = (5, 2)$ belongs to Ω_4. c values of the partial schedules are also shown.

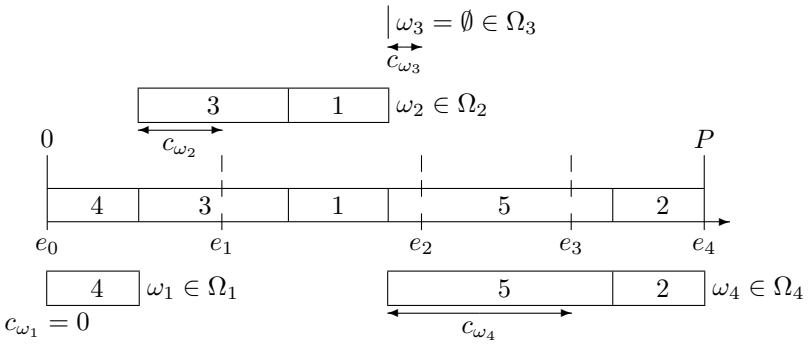


Figure 4.5: Full schedule and its partial schedules

The binary variable λ_ω equals 1 if the solution includes partial schedule ω, and otherwise $\lambda_\omega = 0$. Using these variables, the problem can be formulated as the following master problem.

(IIMP)

$$\min \quad \sum_{u \in M} \sum_{\omega \in \Omega_u} w_\omega \lambda_\omega \tag{4.25}$$

$$s.t. \quad \sum_{u \in M} \sum_{\omega \in \Omega_u} a_{j\omega} \lambda_\omega = 1, \quad \forall j \in N, \tag{4.26}$$

$$e_{u-1} - \sum_{\omega \in \Omega_u} c_\omega \lambda_\omega + \sum_{\omega \in \Omega_u} p_\omega \lambda_\omega = e_u - \sum_{\omega \in \Omega_{u+1}} c_\omega \lambda_\omega,$$
$$1 \leq u < m, \tag{4.27}$$

$$\sum_{\omega \in \Omega_u} \lambda_\omega = 1, \quad \forall u \in M \tag{4.28}$$

$$\sum_{\omega \in \Omega_u} a_{j\omega} \lambda_\omega \in \{0, 1\}, \quad \forall j \in N, \forall u \in M. \tag{4.29}$$

The constraints (4.26) specify that each job is contained in exactly one partial schedule. The constraints (4.27) guarantee that the difference between the starting times of partial schedules of intervals I_{u+1} and I_u equals to the sum of processing times of jobs in the partial schedule of I_u. In other words, a partial schedule should start at the time moment when the previous partial schedule is completed. Finally, the constraints (4.28) state that only one partial schedule should be selected for each interval. Note that $c_\omega = 0, \omega \in \Omega_1$, as a full schedule should start at the time moment 0.

As shown in Section 1.3, the LP relaxation (LPMP) of formulation (IIMP) can be solved by column generation. Remember that jobs in each partial schedule are assigned to the same interval. Therefore, the pricing problem here decomposes into subproblems for each interval.

4.5.1 Solving the pricing problem

When solving (LPMP) by column generation, we deal with the LP relaxation (LPRMP) of the master problem (IIMP) with a restricted set of columns. Let $(\theta, \eta, \mu) \in \mathbb{R}^{n+m+m}$ denote optimal dual solution of (LPRMP). The reduced cost of variable λ_ω, $\omega \in \Omega_u$, is

$$\sum_{j:\, a_{j\omega}=1} w_j T_j(\omega) - \sum_{j \in N} \theta_j a_{j\omega} - \eta_u(p_\omega - c_\omega) - \eta_{u-1} c_\omega - \mu_u.$$

To check if there are columns with a negative reduced cost, we need to solve the pricing problem which decomposes here into m subproblems, one for each interval I_u, $u \in M$.

Now we formulate the pricing subproblem (PS_u) which finds a partial schedule $\omega \in \Omega_u$ with the minimum reduced cost. Binary variable x_j takes value 1 if job $j \in N$ is included in the partial schedule ($a_{j\omega} = 1$), and otherwise $x_j = 0$. Continuous variable c equals c_ω, i.e. the difference between the end of interval I_{u-1} and the starting time of the partial schedule. Integer variable $\tilde{T}_j$, $j \in N$, equals $T_j(\omega) - \tau_j^u$ if $a_{j\omega} = 1$. If $a_{j\omega} = 1$ then $\tilde{T}_j = 0$. Note that $T_j(\omega) = \tilde{T}_j + \tau_j^u x_j$ and $p_\omega = \sum_{j \in N} p_j x_j$. Now we can formulate the following MIP.

$$\min \quad \sum_{j \in N} (w_j \tau_j^u - \theta_j - p_j \eta_u) x_j + \sum_{j \in N} w_j \tilde{T}_j + (\eta_u - \eta_{u-1}) c \tag{4.30}$$

$$s.t. \quad \tilde{T}_j \geq -c + \sum_{i \in B_j^u} p_i x_i + p_j x_j - (1 - x_j) \cdot (e_u - e_{u-1} - 1),$$

$$\forall j \in N, d_j \leq e_{u-1}, \tag{4.31}$$

($\mathbf{PS_u}$)

$$-c + \sum_{j \in N} p_j x_j \leq e_u - e_{u-1}, \tag{4.32}$$

$$-c + \sum_{i \in B_j^u \cup \{j\}} p_i x_i \geq 1, \quad \forall j \in N, \tag{4.33}$$

$$c \geq 0, \tag{4.34}$$

$$x_j \in \{0, 1\}, \quad \forall j \in N, \tag{4.35}$$

$$\tilde{T}_j \in \mathbb{Z}_+, \quad \forall j \in N.. \tag{4.36}$$

The constraints (4.31) correspond to the constraints (4.19). The constraints (4.32) and (4.33) guarantee that all jobs in the partial schedule are completed in interval I_u. When solving the subproblems, we do not consider empty partial schedules. These schedules are included in the initial set of columns of (LPRMP).

We show now that the problem (PS_u) always has an optimal solution which corresponds to a special kind of partial schedules which we call "shifted".

Definition 4.4. A partial schedule $\omega \in \Omega_u$, $u \in M$, is *left-shifted*, if c_ω equals the processing time $p_\omega^{[1]}$ of the first job in ω minus 1. A partial schedule ω is

right-shifted if

$$c_\omega = \max\{p_\omega - (e_u - e_{u-1}), 0\}.$$

The meaning of such partial schedules is that if we "shift" to the right a right-shifted schedule or to the left a left-shifted one, then the partial schedule obtained does not belong to Ω^u any more. Examples of shifted and non-shifted schedules are depicted in Figure 4.6.

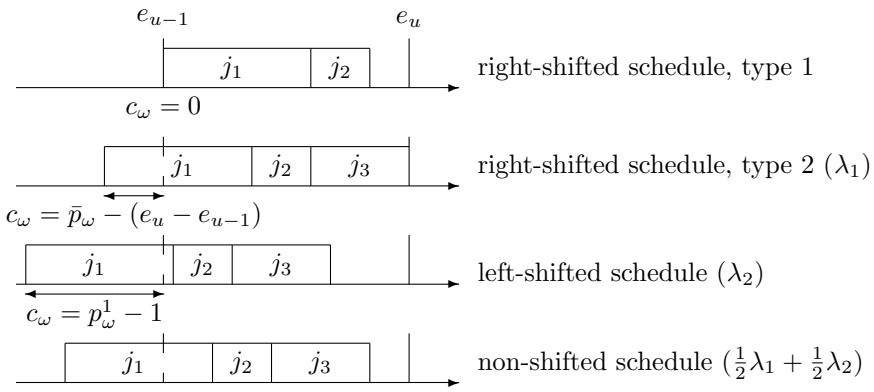


Figure 4.6: Shifted and non-shifted partial schedules

Proposition 4.6. *There exists an optimal either left-shifted or right-shifted partial schedule for subproblem (PS_u), $u \in M$.*

Proof. Let $(x^*, c^*, \tilde{T}^*)$ be an optimal solution (PS_u). We now fix variables x to x^*. Then the objective (4.30) reduces to $\sum_{j \in N} w_j \tilde{T}_j + (\eta_u - \eta_{u-1})c$. The domain of c, or the set of possible values of c, is now an interval, given by the constraints (4.32), (4.33) and (4.34). As the objective function is linear, it reaches its minimum on one of extreme values of the domain of c, as it is shown in Figure 4.7. It is easy to check that these extreme values correspond to shifted partial schedules. □

So, when solving the formulation (LPRMP) by column generation, we deal only with columns which correspond to shifted partial schedules. Any partial schedule $\omega \in \Omega_u$, $u \in M$, can be obtained as a linear combination of columns of (LPRMP). Let $\omega_l \in \Omega_u$ and $\omega_r \in \Omega_u$ be the left-shifted and right-shifted schedules which include the same set of jobs as the schedule ω. Then, for some $\alpha \in [0, 1]$, we have

$$\begin{aligned} w_\omega &= \alpha w_{\omega_l} + (1-\alpha) w_{\omega_r}, \\ c_\omega &= \alpha c_{\omega_l} + (1-\alpha) c_{\omega_r}. \end{aligned}$$

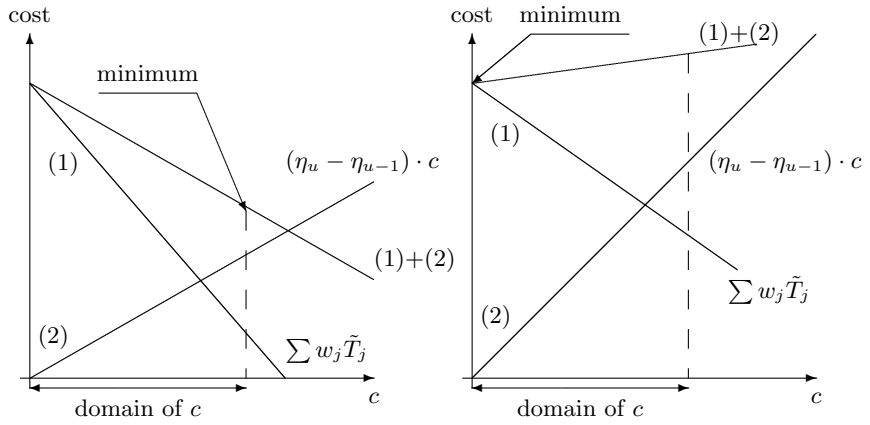


Figure 4.7: Two cases of the minimization of the objective (4.30) when the variables x are fixed

The formulation (PS_u), $u \in M$, can be solved using the standard branch-and-cut method. However, it is possible to use the structure of the pricing subproblem and solve it more efficiently by dynamic programming. If job $j \in N$ is included in the partial schedule and completed at time moment $t \in I_u$, then j contributes the cost C^u_{jt} to the objective (4.30), where

$$C^u_{jt} = w_j \cdot \max\{t - d_j, 0\} - \theta_j - p_j \eta_u + (\eta_u - \eta_{u-1}) \cdot \max\{e_{u-1} - t + p_j, 0\}.$$

The contribution of the variable c is included in the cost of job $j \in N$ when j is the first in the partial schedule, i.e. when $t - p_j \leq e_{u-1}$. In this case $c = e_{u-1} - t + p_j$ and the last term in the formula for C^u_{jt} equals $(\eta_u - \eta_{u-1})c$. When job j is not the first, the last term equals zero.

Let $Z^u_j(t)$ be the minimum reduced cost for partial schedules in Ω_u containing jobs in $B^u_j \cup \{j\}$ and terminating at time moment $t \in I_u$. Let also b_j be the predecessor of job $j \in N$ in permutation σ_u. If $B^u_j = \emptyset$ then $b_j = 0$. We set $Z^u_0(t) := \infty$, $u \in M$, $t \in I_u$. As we know that jobs in the partial schedule are processed according to permutation σ_u, we can build the following recursion:

$$Z^u_j(t) = \begin{cases} \min\{Z^u_{b_j}(t), Z^u_{b_j}(t - p_j) + C^u_{jt}\}, & t - p_j \geq e_{u-1} + 1, \\ \min\{Z^u_{b_j}(t), C^u_{jt}\}, & t - p_j \leq e_{u-1}. \end{cases} \tag{4.37}$$

Then the value of optimal solution of (PS^u) is

$$Z^u_{j^*}(t^*) = \min_{j \in N, t \in I^u} Z^u_j(t).$$

By working backwards, starting from (j^*, t^*), we can retrieve the optimal

values of the variables x and c. It is easy to see that the total time for solving all pricing subproblems by dynamic programming is $O(nP)$.

Note that additional constraints which state that a job should or should not be completed in interval $u \in M$ can be respected easily when solving (PS_u) by dynamic programming. One just needs to suitably modify the recursion (4.37).

4.5.2 Strengthening the formulation

In this subsection, we present three ways to strengthen the formulation (IIMP). Namely, we will use dominance rules, reduced costs and cutting planes.

As shown in Section 4.4, dominance rules for the problem $1 \mid\mid \sum w_j T_j$ can be used to reduce the domains of jobs. If an interval I_u, $u \in M$, is excluded completely from the domain of a job $j \in N$, we can add the constraint stating that j should not be completed in I_u. Such a constraint is not included explicitly to (IIMP). It is taken into account when we solve the pricing subproblem (PS_u).

Similar constraints can be added based on reduced costs. Suppose the solution of the LP relaxation of (IIMP) is LB and UB is an upper bound on the optimal solution of the problem. For each job $j \in N$ and each interval I_u, $u \in M$, we solve the pricing subproblem (PS_u) with the additional constraint $x_j = 1$ forcing j to be assigned to I_u. Let ζ be the value of an optimal solution of this subproblem. If $LB + \zeta \geq UB$ then there is no solution in which j is assigned to I_u with a value less than UB. Therefore, we can add the constraint stating that j should not be completed in I_u. Again, this constraint is taken into account only when we solve the pricing subproblem (PS_u).

A standard way to strengthen a MIP formulation is to use valid inequalities in the form of cutting planes. We now show how to generate cover cuts for the formulation (IIMP). We will convert cover inequalities for the formulation (IIMP) into cover inequalities for (IIMP). The relation between the variables λ of (IIMP) and the variables z of (IIF) is the following:

$$z_j^u = \sum_{1 \leq v \leq u} \sum_{\omega \in \Omega_v, a_{j\omega}=1} \lambda_\omega. \tag{4.38}$$

Set C_u of jobs, $u \in N$, is a *cover* for the first u intervals if $\sum_{j \in C_u} p_j > e_u$. Note that all jobs in C_u cannot be assigned to first u intervals. Then, for a cover C_u, the inequality

$$\sum_{j \in C_u} z_j^u \leq \mid C_u \mid -1, \tag{4.39}$$

is valid for (IIF). The cover inequality (4.39) has the equivalent inequality for the variables λ:

$$\sum_{j \in C_u} \sum_{1 \leq v \leq u} \sum_{\omega \in \Omega_v, a_{j\omega}=1} \lambda_\omega \leq \mid C_u \mid -1. \tag{4.40}$$

The inequality (4.40) is valid for the formulation (IIMP).

Now we present a separation algorithm for inequalities (4.40). Consider a solution λ^* of the formulation (IIMP). We transform λ^* into $z^* \in \mathbb{R}^{n \times m}$ using the formula (4.38). To separate the cover inequalities (4.39), we solve the following problem [87]:

$$\zeta = \min \left\{ \sum_{j \in N} (1 - z_j^{u*}) y_j : \sum_{j \in N} p_j y_j \geq e_u + 1,\ y \in \{0,1\}^n \right\} \tag{4.41}$$

If $\zeta < 1$ then $C_u = \{j : y_j^* = 1\}$ is a cover, and the inequality (4.40) for this cover violates λ^*.

Valid inequalities can be tightened using a so-called *lifting* procedure. To lift cover inequalities, a heuristic from [61] is used. Let α_j be the coefficient of variable z_j^u, $j \in N$, in a cover inequality for cover C^u. We define also $L_\alpha = \{j \in N :\ \alpha_j > 0\}$. The coefficient β_i of variable z_i^u, $i \in N \setminus L_\alpha$, which we add to this inequality, equals $\mid C_u \mid -1 - \zeta_i$, where

$$\zeta_i = \max \left\{ \sum_{j \in L_\alpha} \alpha_j y_j : \sum_{j \in L_\alpha} p_j y_j \leq e_u - p_i,\ y \in \{0,1\}^{|L_\alpha|} \right\}. \tag{4.42}$$

Note that the problem (4.42) can be solved in polynomial time by dynamic programming, as coefficients α_j, $j \in N$, are integer and $\alpha_j <\mid C_u \mid \leq n$.

Similarly to cover inequalities, we will now define reverse cover inequalities. Set RC_u of jobs, $u \in N$, is a *reverse cover* for the last $m - u$ intervals if

$$\sum_{j \in RC_u} p_j \geq P - e_u - \max_{j \in RC_u} p_j.$$

All jobs in RC_u cannot be be assigned to the last $m - u$ intervals, as at least one job in RC_u would be completed in interval I_u or earlier. Therefore, for a reverse cover RC_u, the inequality

$$\sum_{j \in RC_u} (1 - z_j^u) \leq \mid RC_u \mid -1, \tag{4.43}$$

is valid for (IIF). The reverse cover inequality (4.43) has the equivalent inequality for the variables λ:

$$\sum_{j \in RC_u} \sum_{u < v \leq m} \sum_{\substack{\omega \in \Omega_v, \\ a_{j\omega} = 1}} \lambda_\omega \leq \mid RC_u \mid -1. \tag{4.44}$$

Now we present a separation algorithm for inequalities (4.44). Consider a solution λ^* of the formulation (IIMP) such that the corresponding solution

z^* is fractional. The separation problem for the reverse cover inequalities can be formulated in the following way:

$$\zeta = \min \left\{ \sum_{j \in N} z_j^{u*} y_j : \sum_{j \in N} p_j y_j \geq P - e_u + \max_{\substack{j \in N, \\ y_j = 1}} \{p_j\},\ y \in \{0,1\}^n \right\}. \quad (4.45)$$

Again, if $\zeta < 1$ then $RC_u = \{j \in N :\ y_j^* = 1\}$ is a reverse cover. Note that, to get the solution of the problem (4.45), we can solve n following knapsack problems, one for each $k \in N$,

$$\zeta^{(k)} = \min \left\{ \sum_{\substack{j \neq k, \\ p_j \leq p_k}} z_j^{u*} y_j + z_k^{u*} : \sum_{\substack{j \in N, \\ p_j \leq p_k}} p_j y_j \geq P - e_u,\ y \in \{0,1\}^n \right\}, \quad (4.46)$$

and put $\zeta = \min_{k \in N} \zeta^{(k)}$.

To lift reverse cover inequalities, a similar heuristic to that for cover inequalities is used. Let α_j^u is the coefficient of variable z_j^u in a reverse cover inequality based on reverse cover RC_u, $L_\alpha = \{j \in N :\ \alpha_j > 0\}$. The coefficient β_i of variable z_i^u, $i \in N \setminus L_\alpha$, which we add to this inequality, equals $\mid RC_u \mid -1 - \zeta_i$, where

$$\zeta_i = \max \left\{ \sum_{j \in L_\alpha} \alpha_j y_j : \sum_{j \in L_\alpha} p_j y_j \leq P - e_u - 1 - p_i + \max_{\substack{j \in N, \\ y_j = 1}} \{p_j, p_i\},\ y \in \{0,1\}^n \right\}.$$

The solution to the latter problem can be found by solving several knapsack problems, one for each $k \in L_\alpha$,

$$\begin{aligned} \zeta_i^{(k)} = \quad & \max \quad \sum_{\substack{j \in L_\alpha, \\ p_j \leq p_k}} \alpha_j y_j + \alpha_k \\ & s.t. \quad \sum_{\substack{j \in L_\alpha, \\ p_j \leq p_k}} p_j y_j \leq P - e_u - 1 - p_i + \max\{p_k, p_i\}, \\ & \qquad y \in \{0,1\}^{|L_\alpha|}, \end{aligned}$$

and putting $\zeta_i = \max_{k \in L_\alpha} \zeta_i^{(k)}$.

Note that adding cover or reverse cover inequalities to the formulation (IIMP) does not change the structure of the pricing subproblems. One should just change appropriately the coefficients for variables x_j, $j \in N$, in (4.30) by taking into account optimal values for dual variables associates with cover inequalities added.

Numerical experiments showed that it is advantageous to include both variables z and λ in the formulation (IIMP) along with the constraints (4.38).

In this case, when we add cuts (4.39) and (4.43) over variables z, the solution time for the column+cut generation algorithm is less than in the case, in which only λ variables are used and the cuts (4.40) and (4.44) are generated.

4.5.3 Implementation issues

The first question arising, when one wants to implement a column generation algorithm, is the generation of the initial set of columns. Here this initial set is based on a heuristic solution represented by schedule π'. Note that jobs in π' should be processed according to the set σ of permutations. For each interval I^u, $u \in M$, such that $Q_u^(\pi') \neq \emptyset$, we generate one or two columns. The first column corresponds to right-shifted partial schedule $\omega_r \in \Omega_u$ containing all jobs in $Q_u^(\pi')$. The second column corresponds to left-shifted partial schedule $\omega_l \in \Omega_u$ containing the same set of jobs. Only one column is generated for intervals I_1 and I_m, as sets Ω_1 and Ω_m contain only right-shifted partial schedules. Additionally, for each interval I_u, $u \in M \setminus \{m\}$, we generate columns which correspond to empty partial schedules $\omega_l', \omega_r' \in \Omega_u$. For this, we set $w_{\omega_l'} = w_{\omega_r'} := 0$, $a_{j\omega_l'} = a_{j\omega_r'} := 0$, $\forall j \in N$, $c_{\omega_r'} := 0$, $c_{\omega_l'} := e_{u-1}$.

Now we discuss some ways to speed up the column generation algorithm. As has been shown before, in each iteration, we need to solve m subproblems. So, for each interval I_u, $u \in M$, we obtain a column in Ω_u with the minimum reduced cost. Then, all such columns with a negative reduced cost can be added to the master problem formulation, not just the best one.

Then, we know that the set of all columns of the formulation (IIMP) can be restricted to the set of columns corresponding to shifted partial schedules. When such a column $\omega \in \Omega_u$ is added to the master problem formulation, we can add also its "pair", i.e. the column $\omega' \in \Omega^u$ containing the same set of jobs as ω, but "shifted" to the other side.

4.5.4 Branch-and-Price algorithm

Consider the solution λ^* of the LP relaxation of the formulation (IIMP). λ^* can be converted to z^* using the formula (4.38). If z^* is integer then the problem is solved. Remember that the optimal schedule found is represented by a set of partial schedules. Any partial schedule is a linear combination of the corresponding left-shifted and right-shifted schedules. The variables λ represent only shifted partial schedules. Therefore, λ^* can be fractional, even when z^* is integer.

When z^* is fractional, the problem is still unsolved and the value of the optimal solution of (LPMP) is a lower bound. In this case, one needs to branch in order to solve the problem. We branch on a job $j \in N$, for which vector z_j^* is fractional. Let I_u, $u \in M$, be an interval such that z_j^u is fractional. Then, at the first descendant node, we impose the condition $z_j^u = 1$ meaning that job j should be completed not later than e_u. At the second descendant node, we we impose the condition $z_j^u = 0$ meaning that job j should be completed

not earlier than $e_u + 1$. As it has been discussed before, these conditions do not need to be included explicitly in the formulation (IIMP). They just state that a certain job cannot be assigned to certain intervals. Thus, this job is not taken into account when solving the pricing subproblems for these intervals. To speed up the resolution of the formulation (LPMP) at the descendant nodes, we can pass there the columns satisfying the branching conditions.

At each node of the search tree, a heuristic feasible solution can be obtained if z^* is fractional. For this we use the notion of an $\alpha-$*point* [74]. The $\alpha-$point, of job $j \in N$, $0 < \alpha_j \leq 1$, is defined to be the first interval $u(\alpha_j) \in M$, in which an α_j fraction of job j has been completed, according to z^*:

$$u(\alpha_j) = \min\{u \in M : \ z_j^{u*} \geq \alpha_j\}.$$

The heuristic schedule is obtained by ordering the jobs according to their $\alpha-$points. If two jobs $i, j \in N$ have the same $\alpha-$point, they are scheduled according to permutation σ_v, where $v = u(\alpha_i) = u(\alpha_j)$.

In our implementation, at each node of the search tree, 20 heuristic schedules are built, and the best one is chosen. The first 10 schedules π_i, $i \in \{1, \ldots, 10\}$, are obtained using $\alpha_j = 1/i$, $j \in N$. To build a schedule from the last ten, the values α_j, $j \in N$, are chosen randomly. The cost of the best heuristic schedule gives us an upper bound.

4.6 Numerical experiments

In this section, we present results of an experimental study of the formulations (IIF) and (IIMP). We have experimentally compared the formulation (IIF) with other compact MIP formulations of the problem. We have also tested the quality of lower bounds obtained by solving the LP relaxation of the formulation (IIMP). Preliminary results for the Branch-and-Price algorithm are also presented.

We have run the experiments on a set of test instances with the number of jobs varying from 10 to 50. These instances are generated as follows. For each job $j \in N$, an integer processing time p_j, an integer weight w_j, and an integer due date d_j are generated following uniform distributions in [1,100], [1,10] and $[P \cdot (1 - TF - RDD/2), P \cdot (1 - TF + RDD/2)]$, respectively. Here TF is a Tardiness Factor and RDD is the Range of Due Dates. TF and RDD take their values in $\{0.2, 0.4, 0.6, 0.8, 1\}$. Five problems are generated for each combination of TF and RDD parameters. In total, we have 125 problems for each number n of jobs. We have generated the instances for $n = \{10, 20, 30\}$. The instances for $n = \{40, 50\}$ were taken from OR-Library [11].

The experiments have been carried out on a computer with a 2 GHz Pentium IV processor and 512 Mb RAM.

4.6.1 Comparison of compact MIP formulations

We have experimentally compared next MIP formulations for the problem $1 \mid\mid \sum w_j T_j$: the interval-indexed formulation (IIF), the sparse time-indexed formulation, the linear ordering formulation, and the "positional" formulation. The last two formulation have not been yet presented, and we will do so now. These formulations are pretty standard and appeared, for example, in [46].

We now present the sparse reformulation of the time-indexed formulation (TI). This reformulation is due to Pan and Shi [62]. Remember that binary variable x_{jt}, $j \in N$, $t \in [1, P]$, takes value 1 if job j is started at time moment t, and otherwise $x_{jt} = 0$. A series of row transformations to constraints (4.3) of (TI) is applied. The equation for $t = 1$ is left unchanged. For $t = 2, \ldots, P$, we multiply both sides of the equation for $t - 1$ by -1 and then add it to the equation for t. This gives us the following sparse reformulation.

(TIS)

$$\min \quad \sum_{t=1}^{P} \max\{0, t + p_j - 1 - d_j\} x_{jt} \tag{4.47}$$

$$s.t. \quad \sum_{t=1}^{P-p_j+1} x_{jt} = 1, \quad j \in N, \tag{4.48}$$

$$\sum_{j \in N} x_{j1} = 1, \tag{4.49}$$

$$\sum_{j \in N} x_{jt} - \sum_{j \in N:\, p_j < t} x_{j,t-p_j} = 0, \quad t \in [2, P], \tag{4.50}$$

$$x_{jt} \in \{0, 1\}, \quad j \in N,\ t \in [1, P]. \tag{4.51}$$

Here the constraints (4.50) mean that, if a job is started at a time moment $t \in [2, P]$, then another job j should start at time moment $t - p_j$. The matrix of the formulation (TIS) is significantly less dense that the matrix of the formulation (TI). Therefore, (TIS) can be solved much faster by MIP solvers. Note that this formulation is not fully compact, as it contains a pseudopolynomial number of variables and constraints.

We now turn to the linear ordering formulation. Binary variable δ_{ij}, $i, j \in N$, takes value 1 if job i precedes job j, and otherwise $\delta_{ij} = 0$. Continuous variable T_j, $j \in N$, represents the tardiness of job j. Now we can write the linear ordering formulation.

$$\min \quad \sum_{j\in N} w_j T_j \tag{4.52}$$

$$s.t. \quad \delta_{ij} + \delta_{ji} \le 1, \quad i,j \in N,\ i<j, \tag{4.53}$$

(LO)

$$\delta_{ij} + \delta_{jk} + \delta_{ki} \le 2, \quad i,j,k \in N,\ i \ne j \ne k, \tag{4.54}$$

$$T_j \ge \sum_{i\in N, i\ne j} p_i \delta_{ij} + p_j - d_j, \quad j \in N, \tag{4.55}$$

$$T_j \ge 0, \quad j \in N, \tag{4.56}$$

$$\delta_{ij} \in \{0,1\}, \quad i,j \in N,\ i \ne j. \tag{4.57}$$

The constraints (4.53) state that, for every pair of jobs, one should precede the other. The constraints (4.54) guarantee that, for every triple of jobs $i,j,k \in N$, if i precedes j and j precedes k then i should precede k. The constraints (4.55) relate the variables δ and T.

Now we present the "positional" formulation. Assignment binary variable x_j^k, $j,k \in N$ takes value 1 if job j is assigned to position k in the schedule. Continuous variable γ_k, $k \in N$, equals to the completion time of the job at position k. Continuous variables T_j, $j \in N$, represents the tardiness of the job j. Then the formulation is

$$\min \quad \sum_{j\in N} w_j T_j \tag{4.58}$$

$$s.t. \quad \sum_{j\in N} x_j^k = 1, \quad k \in N, \tag{4.59}$$

$$\sum_{k\in N} x_j^k = 1, \quad j \in N, \tag{4.60}$$

(POS)

$$\gamma_k = \gamma_{k-1} + \sum_{j\in N} p_j x_j^k, \quad k \in N \setminus \{1\}, \tag{4.61}$$

$$\gamma_1 = \sum_{j\in N} p_j x_j^1, \tag{4.62}$$

$$T_j \ge \gamma_k - U \cdot (1 - x_j^k) - d_j, \quad j,k \in N, \tag{4.63}$$

$$T_j \ge 0, \quad j \in N, \tag{4.64}$$

$$x_i^k \in \{0,1\}, \quad i,k \in N. \tag{4.65}$$

The assignment constraints (4.59) and (4.60) state that a job can occupy exactly one position in the schedule, and a position can be occupied by exactly one job. The constraints (4.61) and (4.62) compute positional completion times. The constraints (4.55) relate the variables x, γ and T.

We have used the *Xpress-MP* MIP solver version 15.20 to solve the formulations (IIF), (TI), (LO), and (POS) by branch-and-cut. We have put a time limit of 10 minutes when solving the test instances. For all formulations, we have used the Emmons dominance rules in the preprocessing stage.

	(POS)		(TIS)		(LO)		(IIF)	
n	P_{10m}	T_{av}	P_{10m}	T_{av}	P_{10m}	T_{av}	P_{10m}	T_{av}
10	96.8%	29.4	100%	0.3	100%	0.1	100%	0.1
20	29.6%	50.7	99.2%	23.3	98.4%	2.2	100%	1.7
30	21.0%	7.7	80.0%	120.5	88.0%	30.9	82.4%	30.8
40	17.6%	0.2	51.2%	450.6	65.6%	37.9	64.8%	34.0
50	-	-	31.2%	272.1	47.2%	84.6	57.6%	56.5

Table 4.3: Comparison of four MIP formulations for the problem

In the experiments, for each number n of jobs, we were interested in the following statistics.

P_{10m} — percentage of instances solved to optimality within 10 minutes.

T_{av} — average time in seconds needed to solve an instance to optimality (only for instances solved to optimality).

Nd_{av} — average number of nodes in the search tree (only for instances solved to optimality).

Gap — average integrality gap, i.e. the average difference between the best found solution and the best found lower bound, percentage wise the best found solution (only for instances which were not solved to optimality and for which at least one feasible solution was found).

XLP — average difference between the optimal solution and the lower bound at the top node of the search tree after generating standard cuts, percentage wise the optimal solution (only for instances for which the LP relaxation was solved within the time limit).

The comparison results are presented in Table 4.3. As it can be seen, the formulations (LO) and (IIF) clearly outperform the others. It is also worth noticing that the formulation (TI) has the smallest XLP ratio, but the size of the formulation does not allow to use it even for small instances.

In Table 4.4, we present detailed results for the formulations (LO) and (IIF). The formulation (LO) is better when solving 30-jobs instances, as it can solve more instances within the time limit. However, the formulations (LO) and (IIF) have solved almost the same number of 40-jobs instances. Moreover, the formulation (IIF) has solved more 50-jobs instances. We also notice that the interval-indexed formulation is much tighter than the linear ordering formulation. The statistics XLP is more than 3 times smaller for (IIF) than for (LO). Also, the average integrality gap is much better for instances unsolved by (IIF) than for instances unsolved by (LO).

The formulation (LO) has $O(n^3)$ constraints. When the dimension of the problem increases, the size of this formulation quickly grows and becomes very

	(LO)				
n	$P_{10\mathrm{m}}$	T_{av}	Nd_{av}	XLP	Gap
10	100%	0.1	10.2	8.8%	0.0%
20	98.4%	2.2	323.1	18.9%	22.0%
30	88.0%	30.9	2380.5	23.0%	25.1%
40	65.6%	37.9	1038.7	23.6%	30.6%
50	47.2%	84.6	554.9	23.7%	30.5%
	(IIF)				
n	$P_{10\mathrm{m}}$	T_{av}	Nd_{av}	XLP	Gap
10	100%	0.1	14.1	4.7%	0.0%
20	100%	1.7	338.3	6.8%	0.0%
30	82.4%	30.8	5245.8	6.8%	3.0%
40	64.8%	34.0	2320.0	7.0%	3.6%
50	57.6%	56.5	2654.8	7.2%	3.9%

Table 4.4: Further comparison of the formulations (LO) and (IIF)

large. Therefore, the effectiveness of (LO) drops rapidly with the increase of the dimension. Results of Khowala et al. [46] show that, for larger n, the formulation (LO) becomes less and less efficient and "loses" even to the formulation (POS).

Theoretically, the formulation (IIF) also has $O(n^3)$ constraints, as an appropriate partition generated by Algorithm 4.2 has $O(n^2)$ intervals. However, in the experiments, the number of intervals was below $2n$ for all test instances. So, in practice, the size of the formulation (IIF) is $O(n^2)$. This suggests that, for larger dimensions, the size of (IIF) will remain reasonable, and the interval-indexed formulation will remain the best among four formulations considered here. However, note that the length of the time horizon has a big impact of the formulation (TI). When this length is small, the formulation (TI) is usually the best choice.

4.6.2 Non-compact formulation

In this subsection, we experimentally test the LP relaxation (LPMP) of the formulation (IIMP). We have used the *Mosel* modelling and programming language version 1.7 to implement the column generation algorithm. Note that the subroutines to solve the pricing problem by dynamic programming and the cut separation problem have been coded in the *Microsoft Visual C++* environment version 6.0. To implement the cut separation algorithm, the code of the ExpKnap algorithm by Pisinger [65] was used. In the column generation algorithm, to solve the LP relaxation (LPRMP) of the restricted master problem, the *XPress-MP* MIP solver version 15.20 has been used.

In the experiments, for each number n of jobs, we were interested in the following statistics.

$T_{\text{av}}^{\text{CG}}$ — average time in seconds needed to solve (LPMP).

T_{av} — average time in seconds needed to solve (LPMP) and generate all violated cover and reverse cover cuts.

Cl_{av} - average number of columns generated.

It_{av} - average number of iterations in the column generation algorithm.

Ct_{av} - average number of cuts generated.

In_{av} - average number of intervals.

LB^{CG} - - average difference between the column generation lower bound and the optimal solution, percentage wise the value of the optimal solution.

LB - average difference between the column+cut generation lower bound and the optimal solution, percentage wise the value of the optimal solution.

n	T_{av}	$T_{\text{av}}^{\text{CG}}$	Cl_{av}	It_{av}	Ct_{av}	In_{av}	LB	LB^{CG}
10	0.1	0.1	82.4	8.8	3.0	10.0	2.35%	4.02%
20	0.3	0.2	280.1	18.2	11.2	19.2	2.45%	3.89%
30	0.9	0.5	578.1	25.5	21.8	28.7	1.81%	2.86%
40	2.0	1.3	958.1	33.0	28.5	38.2	2.15%	3.05%
50	4.0	2.6	1436.8	41.0	38.9	47.6	1.50%	2.36%

Table 4.5: Results for the column generation algorithm

Results are presented in Table 4.5. As it can be seen, the LP relaxation (LPMP) can be solved relatively fast using the column generation algorithm. A useful observation is that the average number of iterations and the average number of cuts grow linearly when the number of jobs increases, i.e. the algorithm scales well. This explains the fact that the average time grows quite slowly.

Another important observation concerns the lower bounds. The cover and reverse cover cuts help significantly to decrease the average gap between the lower bound and the optimal solution. Moreover, the cut generation takes less time than the column generation. We conclude that it is worth to use cuts for this formulation.

We now compare our column+cut generation lower bound with the lower bound produced by the LP relaxation of the formulation (TIS) and the lower bound obtained by Babu et al. [6] using the Lagrangean relaxation of the time-indexed formulation. We have received the code of Babu et al. and have

run it on the same computer. To solve the LP relaxation of the formulation (TIS), we have used the *Xpress-MP* LP solver version 16.10 with the default settings. The comparison between three lower bounds is shown in Table 4.6.

	LP relaxation of (TIS)		Babu et al. [6]		Our bound	
n	Bound	Time	Bound	Time	Bound	Time
40	0.67%	278.7	0.75%	26.3	2.15%	2.0
50	0.73%	570.6	0.82%	49.1	1.50%	4.0

Table 4.6: Comparison of three lower bounds for the problem $1 \mid\mid \sum w_j T_j$

Our experiments showed that the Lagrangean relaxation lower bound is slightly worse than the lower bound of the LP relaxation of (TIS). But the former lower bound can be computed much faster. On the other hand, the Lagrangean relaxation lower bound is better than our lower bound. However, the calculation of our lower bound takes an order of magnitude less time.

We conclude that our column+cut generation lower bound has a good trade-off between the quality and the time needed to obtain it.

Chapter 5

Conclusions

We now review the main results presented in the thesis and discuss some directions and perspectives for future research.

In Chapter 2 we considered one of the classical one-machine non-preemptive scheduling problems, minimizing the weighted number of late jobs on a single machine with presence of different release dates. This problem can be seen as a selection of a feasible set of jobs with the maximum total weight. The feasibility of a set is determined by a possibility to schedule jobs in this set on a single machine without preemption and without violation of release and due dates. To solve this problem to optimality, we proposed a Branch-and-Check algorithm which is based on the Benders decomposition of a Mixed Integer Programming formulation. This decomposition breaks the problem into two natural stages, choosing a set of jobs and checking its feasibility. The first stage is tackled by Integer Programming, and in the second stage, we use specialized combinatorial scheduling algorithms and CP. The cooperation between two stages is carried out via infeasibility cuts which are generated in the second stage and used in the first stage.

The idea of such a decomposition of a scheduling problem is not new. It has been already suggested in the context of multi-machine problems, for example in [41, 16]. Our contribution to this approach consists in two ingredients. First, we suggested additional tightening knapsack-type inequalities for the MIP formulation of the first stage of the problem, namely choosing a set of jobs. Adding these inequalities allows us to significantly decrease the number of infeasible sets of jobs chosen in the first stage. Thus, the number of infeasibility cuts produced in the second stage also decreases leading to a smaller solution time for the whole algorithm.

Secondly, we proposed new algorithms for generating infeasibility cuts. The usual approach is to generate so-called "no-good" cuts which mean that all the jobs in an infeasible set cannot be chosen at the same time. Our first algorithm is based on the idea that, when a set of jobs is infeasible, it is possible usually to find a strict subset of this set of jobs that is also

infeasible. In this case, it is possible to generate the "no-good" cut for a subset of jobs rather that for the whole set. Such a strengthened "no-good" cut can replace many standard "no-good" cuts thus significantly decreasing the solution time. The idea for our second algorithm came from Constraint Programming, particulary from the "Edge-Finding" technique used in CP. This technique allows to find special pairs (Q, k), where Q is a set of jobs and k is a job. When jobs in $Q \cup \{k\}$ are processed on a single machine without preemption and without violation of release and due dates, the interval in which job k can be processed is restricted. Using such pairs (Q, k), it is possible to add additional tightening inequalities to the MIP formulation of the first stage of the problem. We use these inequalities as infeasibility cuts, because their number can be exponential.

The resulting Branch-and-Check algorithm which uses the two ingredients described above was able to solve all the standard test instances with up to 100 jobs within one hour on a computer with a 2 GHz processor and 512 Mb of RAM. We have also carried out a numerical comparison of this algorithm with another algorithm recently proposed in the literature. This comparison on a set of standard test instances showed the superiority of the Branch-and-Check algorithm.

An important open question that arises when analyzing the decomposition scheme on which the proposed Branch-and-Check algorithm is based concerns "no-good" infeasibility cuts. Is it possible to devise a practically reasonable algorithm which, given an infeasible set of jobs, is able to find a minimal infeasible subset of this set? Such a subset of jobs would allow us to generate the strongest inclusion wise "no-good" cut. A straightforward algorithm consists in taking away jobs from the given set one by one and checking each time whether the set becomes feasible or not. If it becomes feasible, we put the job back to the set. Such an approach requires solving a series of feasibility problems whose number equals the cardinality of the set of jobs. When this set is reasonably large, this straightforward approach is unlikely to perform well in practice.

In the Branch-and-Check algorithm proposed we did not use dominance rules for the problem. It is interesting to see the impact of these rules on the efficiency of the algorithm. Particulary, those rules might be useful, which can be expressed only with x variables. For instance, such a rule is "if job j is processed on-time then job i should be processed on-time".

Another research direction could be to look for other types of infeasibility cuts and tightening inequalities for the MIP formulation of the first stage of the problem. However, our opinion is that the most important line of research would be to study applicability of the decomposition scheme presented above to practical problems. In this context, the results in the next chapter of the thesis seem to be interesting.

In Chapter 3, we studied two multi-machine non-preemptive scheduling problems. The first one is the multi-machine assignment scheduling problem

and the second one is the problem of minimizing the maximum tardiness on unrelated machines. For both problems we proposed Branch-and-Check and Branch-and-Price algorithms.

The first multi-machine problem is a generalization of the one-machine problem considered in Chapter 2. For this multi-machine problem we propose a Branch-and-Check algorithm based on a similar decomposition scheme. The difference is that, in the first stage, we choose not just a set of jobs but an assignment of jobs to machines. In the second stage, for each machine, we check the feasibility of the set of jobs assigned to the machine. The MIP formulation for the first stage is amplified by the assignment constraints. The infeasibility cuts used are the same as in the one-machine case.

The alternative approach we proposed to solve the multi-machine assignment scheduling problem was the Dantzing-Wolfe reformulation. In it, each binary variable corresponds to a partial schedule on a certain machine. The Linear Programming relaxation of the Dantzing-Wolfe reformulation is solved by column generation. The pricing problem here decomposes into one-machine problems, one for each machine. Every one-machine problem is exactly the problem considered in Chapter 2. Thus, the Branch-and-Check algorithm for the one-machine problem can be used for solving the pricing problem. A Branch-and-Price algorithm for solving the Dantzing-Wolfe reformulation was finally presented.

A numerical comparison between our Branch-and-Check and Branch-and-Price algorithms and other algorithms existent in the literature was carried out on sets of publically available and newly generated instances. The Branch-and-Price algorithm solved all the test instances with up to 9 machines and 54 jobs. It was the best among the algorithms tested.

Then, our Branch-and-Check and Branch-and-Price algorithms were modified to tackle the problem to minimize the maximum tardiness on unrelated machines. The results of the experimental research for these modified algorithms were not so successful as in the case with the first multi-machine problem. However, to our knowledge, these two modified algorithm are the first approaches proposed for the the problem to minimize the maximum tardiness on unrelated machines.

Finally, in the chapter devoted to the multi-machine problems, we considered a special case, in which machines are identical. Methods to break symmetry were suggested for both algorithms. For the Branch-and-Check algorithm, we proposed an alternative branching strategy which allows us to consider only one solution in each symmetry group. For the Branch-and-Check algorithm, we showed how to reformulate the master problem in order to remove the symmetry. Also, branching strategies were discussed which are suitable for the non-symmetric reformulation.

Although the proposed algorithms are quite efficient in solving the multi-machine problems studied, there is still room for improvement. For the Branch-and-Check algorithms, the "bottleneck" is the IP formulation for the

first stage of the problem, the assignment of jobs to machines. This formulation includes the assignment constraints as well as knapsack constraints (additional tightening inequalities). This relates it to the Generalized Assignment Problem [23] which is quite hard to solve by state-of-the-art MIP solvers. Thus, an effort should be made to decrease the solution time for the first stage of the problem. One of approaches for this could be to derive families of valid inequalities which can be used as cuts when solving the MIP formulation. Another approach could be using dominance rules for the corresponding scheduling problem.

Again, an important question is the possibility of extending the proposed algorithms in order to solve practical problems. One practical generalization of the multi-machine problems studied here is the so-called *cumulative* case, in which machines can have capacities different from 1. In this case, machines act as resources, and jobs can require only a part of the machine to be processed on. This means that a machine can process several jobs in parallel. Algorithms which use the decomposition scheme similar to ours have been recently proposed in the literature for such problems, see for example a work by Hooker [43]. In these algorithms, the interaction between the assignment part of the problem and feasibility checking is carried out using standard "no-good" cuts. Thus, extending our algorithms for generating strengthened "no-good" cuts for the cumulative case would be of value.

In Chapter 4 we considered another classical non-preemptive one-machine scheduling problem, minimizing the total weighted tardiness. This important problem has been being studied in literature for more than 30 years. We contributed to the study by proposing two new MIP formulations for it.

The starting point of our study was the time-indexed MIP formulation. In it, each binary variable determines whether a job is completed at a certain time moment or not. It is known that the LP relaxation for this formulation provides tight lower bounds for the problem. However, the size of this formulation is big which limits a lot its applicability in practice.

To overcome the drawback of the time-indexed formulation, we proposed to partition the time horizon into a relatively small number of intervals and replace the time-indexed variables by interval-indexed variables which determine whether a job is completed in a certain interval or not. The partition is carried out in such a way that, given a set of jobs completed in the same interval, we know their optimal ordering. Such a partition was called *appropriate*. We proposed an algorithm for finding an appropriate partition of the time horizon into intervals. The number of intervals is guaranteed to be $O(n^2)$, but in practice, this number was always less than $2n$, where n is the number of jobs.

Using an appropriate partition of the time horizon, we presented the interval-indexed formulation for the problem. A numerical comparison between this formulation and other compact MIP formulations for the problem was carried out on a set of standard test instances. Experiments showed that

the proposed formulation has the smallest integrality gap, and it is the best compact MIP formulation to use for instances when the number of jobs is equal to 50 and more.

Then, we proposed a Dantzig-Wolfe interval-indexed reformulation of the problem. In it, each binary variable represents a partial schedule containing jobs completed in the same interval. The column generation algorithm for solving the LP relaxation of this reformulation was considered. We showed that the pricing problem can be solved by dynamic programming in pseudopolynomial time. Ways to improve the Dantzig-Wolfe reformulation using dominance rules and cover cuts were studied. Experimental research showed that the LP relaxation of the reformulation can be solved quite fast and generates fairly good lower bounds on the optimal solution of the problem.

We believe that future investigation is needed for both the time-indexed and the interval-indexed formulations and their LP relaxations. The interval-indexed relaxation can be solved much faster but it loses a lot to the time-indexed relaxation in terms of the quality of lower bounds. The impact of the number of intervals on the solution time and the the quality of lower bounds is not well studied here. Such a study can help in determining in how many intervals it is best to partition the time horizon.

A natural direction for the future research concerns the Branch-and-Price algorithm which can be built on the basis of the Dantzig-Wolfe interval-indexed reformulation proposed. It is interesting to compare such an algorithm with other column generation-based approaches [3, 14] for solving the problem. In [14], Bigras et al. have also exploited the idea of a partition of the time horizon into intervals. There, the partition does not use the structure of the problem. However, the uniform partition the authors use helps to speed up the column generation algorithm by allowing the decomposition of the pricing problem. We think it is worth to study the relation between the approach of Bigras et al. and our approach. By taking the best components from the both algorithms, it may be possible to improve their efficiency.

Bibliography

[1] T. Abdul-Razaq, C. Potts, L. Van Wassenhove. A survey of algorithms for the single machine total weighted tardiness scheduling problem. *Disc. Applied Math.* 26:235-253, 1990.

[2] J.M. van den Akker, C.A.J. Hurkens, M.W.P. Savelsbergh. A polyhedral approach to single-machine scheduling problems. *Math. Prog.* 85:541-572, 1999.

[3] J.M. van den Akker, C.P.M. van Hoesel, M.W.P. Savelsbergh. Time indexed formulations for single-machine scheduling problems: column generation. *INFORMS J. of Computing* 12:111-124, 2000.

[4] M.S. Arkturk, Yildririm M.B. A new dominance rule for the total weighted tardiness problem. *Production Planning and Control*, 10(2):138-149, 1999.

[5] A. Atamturk, M.W.P. Savelsbergh. Integer-Programming software systems. *Annals of Oper. Res.* 140:67-124, 2005.

[6] P. Babu, Péridy L., Pinson E. A branch and bound algorithm to minimize total weighted tardiness on a single processor. *Annals of Oper. Res.* 129:33-46, 2004.

[7] Ph. Baptiste, C. Le Pape, W. Nuijten. *Constraint-Based Scheduling: Applying Constraints Programming to Scheduling Problems.* Kluwer Academics Publishers, London, 2001.

[8] Ph. Baptiste, A. Jouglet, C. Le Pape, W. Nuijten. A constraint-based approach to minimize the weighted number of late jobs on parallel machines. Research Report 2000/288, Université Technologie de Compiègne, 2000.

[9] Ph. Baptiste, L. Peridy, E. Pinson. A branch and bound to minimize the number of late jobs on a single machine with release time constraints. *Eur. J. of Oper. Res.* 144(1):1-11, 2003.

[10] C. Barnhart, E.L. Johnson, G.L. Nemhauser, M.W.P. Savelsbergh, P.H. Vance. Branch-and-price: column generation for huge integer programs. *Oper. Res.* 46:316-329, 1998.

[11] J.E. Beasley. OR-Library, `http://people.brunel.ac.uk/~mastjjb/jeb/info.html`.

[12] J.F. Benders. Partitioning procedures for solving mixed-variables programming problems, *Numerische Mathematik*, 4:238-252, 1962.

[13] D. Bertsimas, R. Weismantel. *Optimization Over Integers.* Dynamic Ideas, Belmont, Massachusetts, 2005.

[14] L.-P. Bigras, M. Gamache, G. Savard. Time-indexed formulations and the total weighted tardiness problem. *Les Cahiers du GERAD*, G-2005-30, 2005.

[15] A. Bockmayr, Th. Kasper. Branch-and-Infer: a unifying framework for integer and finite domain constraint programming. *INFORMS J. on Computing* 10:287-300, 1998.

[16] A. Bockmayr, N. Pisaruk. Detecting infeasibility and generating cuts for MIP using CP. *Computers and Oper. Res.* 33(10):2777-2786, 2006.

[17] P. Brucker. *Scheduling Algorithms.* Springer, Heidelberg, 1995.

[18] P. Brucker, S. Knust. Complexity results for scheduling problems, `http://www.mathematik.uni-osnabrueck.de/research/OR/class/`.

[19] E.H. Bowman. The schedule-sequencing problem. *Oper. Res.* 7:621-624, 1959.

[20] J. Carlier. The one machine sequencing problem. *Eur. J. of Oper. Res.* 11:42-47, 1982.

[21] J. Carlier. Problème d'ordonnancements à contrantes de ressources: algorithms and complexité. Thèse d'Etat, Université Paris IV, 1984.

[22] J. Carlier. Scheduling jobs with release dates and tails on identical machines to minimize the makespan. *Eur. J. of Oper. Res.* 29:298-306, 1987.

[23] D.G. Cattrysse, L.N. Van Wassenhove. A survey of algorithms for the generalized assignment problem. *Eur. J. of Oper. Res.* 60:260-272, 1992.

[24] Z.-L. Chen, W.B. Powell. Solving parallel machine scheduling problems by colum generation. *INFORMS J. on Computing* 11:78-94, 1999.

[25] G. Codato, M. Fischetti. Combinatorial Benders' cuts for mixed-integer linear programming. To appear in *Oper. Res.* 54(4), 2006.

[26] Y. Colombani, T. Heipcke. Mosel: an extensible environment for modeling and programming solutions. In: Proceedings of the CP-AI-OR'02, 277-290. Le Croisic, France, 2002.

[27] R. Congram, C. Potts, S. Van de Velde. An iterated dynasearch algorithm for the single-machine weighted tardiness problem. *INFORMS J. on Computing* 14:52-67, 2002.

[28] A. Crauwels, C. Potts, L. Van Wassenhove. Local search heuristics for the single machine total weighted tardiness scheduling problem. *INFORMS J. on Computing* 10:341-350, 1998.

[29] G.B. Dantzig, P. Wolfe. Decomposition principle for linear programs. *Oper. Res.* 8:101-111, 1960.

[30] S. Dauzère-Pérès, M. Sevaux. Using Lagrangean relaxation to minimize the weighted number of late jobs on a single machine. *Naval Res. Logist.* 50(3):273-288, 2003.

[31] J. Du, J.Y.-T. Leung. Minimizing total tardiness on one machine is NP-hard. *Math. Oper. Res.* 15(3):483-495, 1990.

[32] M. Dyer, L.A. Wolsey. Formulating the single machine sequencing problem with release dates as mixed integer program. *Disc. Applied Math.* 26:255-270, 1990.

[33] A. Eremin, M. Wallace. Hybrid Benders decomposition algorithms in constraint logic programming. In: Proceedings of the CP'2001. *Lecture Notes on Computer Science* 2239:1-15. Springer, 2001.

[34] H. Emmons. One-machine sequencing to minimize certain functions of jobs tardiness. *Oper. Res.*, 17:701-715, 1969.

[35] P. Ersquirol, P. Lopez, C. Thuriot. Raisonnement temporel sous contraintes de ressource er problèmes d'ordonnoncement. *Revue d'Intelligence Artificielle* 5(3):7-32, 1991.

[36] G. Finke, V. Jost, M. Queyranne and A. Sebő. Batch processing with interval graph compatibilities between tasks. Cahier No. 108, Laboratoire Leibnitz, Grenoble, 2004.

[37] A. Gharbi, M. Haouari. Minimizing makespan on parallel machines subject to release dates and delivery times. *J. of Sched.* 5:329-355, 2002.

[38] R.L. Graham, E.L. Lawler, J.K. Lenstra, A.H.G Rinnooy Kan. Optimization and approximation in deterministic sequencing and scheduling: a survey. *Ann. Discrete Math.* 5:287-326, 1979.

[39] M. Haouari, A. Gharbi. An improved max-flow-based lower bound for minimizing maximum lateness on identical parallel machines. *Oper. Res. Letters* 31:49-52, 2003.

[40] M. Haouari, A. Gharbi. Lower bounds for scheduling on identical parallel machines with head and tails. *Annals of Oper. Res.* 129:187-204, 2004.

[41] V. Jain, I.E. Grossmann. Algorithms for hybrid MILP/CLP models for a class of optimization problems. *INFORMS J. on Computing* 13(4):258-276, 2001.

[42] J.N. Hooker. Logic, optimization, and constraint programming. *INFORMS J. on Computing* 14(4):295-321, 2002.

[43] J. N. Hooker. A hybrid method for planning and scheduling. *Constraints* 10:385-401, 2005.

[44] J.N. Hooker, G. Ottosson. Logic-based Benders decomposition. *Math. Prog.* 96:33-60, 2003.

[45] H. Kellerer, U.Pferschy, D. Pisinger. *Knapsack Problems.* Springer, Heidelberg, 2004.

[46] K. Khowala, A. Keha, J. Fowler. A comparison of different formulations for the non-preemptive single machine total weighted tardiness scheduling problem. In Proceedings of the 2nd Multidisciplinary International Conference on Scheduling : Theory and Applications (MISTA 2005), New York, 2005.

[47] H. Kise, T. Ibaraki, H. Mine. A solvable case of the one machine scheduling problem with ready and due times. *Oper. Res.* 26(1):121-126, 1978.

[48] G. Lancia. Scheduling jobs with release dates and tails on two unrelated parallel machines to minimize the makespan. *Eur. J. of Oper. Res.* 120:277-288, 2000.

[49] E.L. Lawler. On scheduling problems with deferral costs. *Manag. Sci.* 11(2):280-288, 1964.

[50] E.L. Lawler. Sequencing to minimize the weigthed number of tardy jobs. *RAIRO Oper. Res.* 10:27-33, 1976.

[51] E.L. Lawler. A pseudopolynomial algorithm for sequencing jobs to minimize total tardiness. *Annals of Disc. Math.* 1:331-342, 1977.

[52] E.L. Lawler. Knapsack-like scheduling problems, the Moore-Hodgson algorithm and the "tower od sets" property. *Math. and Comp. Modelling* 20(2):91-106, 1994.

[53] E.L. Lawler, J.M. Moore. A functional equation and its application to resource allocation and sequencing problems. *Manag. Sci.* 16:77-84, 1969.

[54] E.L. Lawler, J.K. Lenstra, A.G.H. Rinnooy Kan, D.B. Shmoys. Sequencing and scheduling: algorithms and complexity. In: Handbooks in Operations Research and Management Science, vol 4, *Logistics of Production and Inventory.* 455-522. North-Holland, Amsterdam, 1993.

[55] J.K. Lenstra, A.H.G. Rinnooy Kan, P. Brucker. Complexity of machine scheduling problems. *Annals of Discr. Math.* 1:343-362, 1977.

[56] J.Y-T. Leung (ed.) *Handbook of Scheduling: Algorithms, Models, and Performance Analysis.* CRC Press, Boca Raton, USA, 2004.

[57] O. Lhomme. Consistency techniques fo numeric CSPs. In: Proceedings of the IJCAI'93, Chambéry, France, 1993.

[58] P. Lopez, J. Erschler, P. Ersquirol. Ordonnoncement de tâches sous contraints : une approche énergétique. *RAIRO Automatique, Productique, Informatique Industrielle* 26(6):453-481, 1992.

[59] F. Margot. Exploiting orbits in symmetric ILP. *Math. Prog.* 98:3-21, 2003.

[60] J.M. Moore. An n job, one machine sequencing algorithm for minimizing the number of late jobs. *Manag. Sci.* 15(1):102-109, 1968.

[61] G.L. Nemhauser, L.A. Wolsey. *Integer and Combinatorial Optimization.* John Wiley & Sons Inc., New York, 1988.

[62] Y. Pan, L. Shi. On the equivalence of the max-min transportation lower bound and the time-indexed lower bound for single-machine scheduling problems. *Optimization Online*, N. 1100, 2005.

[63] L. Péridy, E. Pinson, D. Rivraux. Using short-term memory to minimize the weighted number of late jobs on a single machine. *Eur. J. of Oper. Res.* 148:591-603, 2003.

[64] J. Picard, M. Queyranne. The time-dependent traveling salesman problem and its application to the tardiness problem in one machine scheduling. *Oper. Res.* 26:86-110, 1978.

[65] D. Pisinger. An expanding-core algorithm for the exact 0-1 Knapsack Problem. *Eur. J. of Oper. Res.* 87:175-187, 1995.

[66] C.N. Potts, L.N. Van Wassenhove. A branch-and-bound algorithm for the weighted tardiness problem. *Oper. Res.* 33:363-377, 1985.

[67] A.A.B. Pritsker, L.J.Watters, P.M. Wolfe. Multiproject scheduling with limited resources : a zero-one programming approach. *Manag. Sci.* 16:93-108, 1969.

[68] C.N. Redwine, D.A. Wismer. A mixed integer programming model for scheduling orders in a steel mill. *J. of Optimization Theory and Applications* 14:305-318, 1974.

[69] A. Rinnooy Kan, Lageweg B., Lenstra J.K. Minimizing total cost in one-machine scheduling. *Oper. Res.*, 23:908-927, 1975.

[70] R.M.V. Rachamadugu. A note on the weigthed tardiness problem. *Oper. Res.*, 35:450-451, 1987.

[71] D.M. Ryan, B.A. Foster. An Integer Programming Approach to scheduling. A. Wren (ed.) *Computer Scheduling of Public Transport Urban Passenger Vehicle and Crew Scheduling.* North-Holland, Amsterdam, 269-280, 1981.

[72] R. Sadykov. A hybrid branch-and-cut algorithm for the one-machine scheduling problem. In: Proceedings of the CP-AI-OR'04. *Lecture Notes on Computer Science* 3011:409-414, Springer, 2004.

[73] R. Sadykov, L. Wolsey. Integer programming and constraint programming in solving a multi-machine assignment scheduling problem with deadlines and release dates. *INFORMS J. on Computing* 18(2):209-217, 2006.

[74] M.W.P. Savelsbergh, R.N. Uma, J. Wein. An experimental study of LP-based approximation algorithms for scheduling problems. *INFORMS J. on Computing* 17:123-136, 2005.

[75] L. Schrage. Solving resource-constrained network problems by implicit enumeration: non preemptive case. *Oper. Res.* 18:263-278, 1970.

[76] A. Schrijver. *Theory of Linear and Integer Programming.* John Wiley & Sons Inc., New York, 1998.

[77] S. Sen, H.D. Sherali. Decomposition with branch-and-cut approaches for two-stage stochastic mixed-integer programming. To appear in *Math. Prog.*

[78] T. Sen, J.M. Sulek, P. Dileepan. Static scheduling research to minimize weighed and unweighed tardiness: a state-of-the-art survey. *Int. J. Production Economics*, 83:1-12, 2003.

[79] M. Sevaux, S. Dauzère-Pérès. Genetic algorithms to minimize the weighted number of late jobs on a single machine. *Eur. J. of Oper. Res.* 151:296-306, 2003.

[80] J.P. Sousa, L.A. Wolsey. A time-indexed formulation of non-preemptive single-macine scheduling problems. *Math. Prog.* 54:353-367, 1992.

[81] W. Szwarc, A. Grosso, F. Della Croce. Algorithmic paradoxes of the single machine total tardiness problem. *J. of Sched.* 4:93-104, 2001.

[82] E.S. Thorsteinsson. Branch-and-check: a hybrid framework integrating mixed integer programming and constraint logic programming. In: Proceedings of the CP'2001. *Lecture Notes on Computer Science* 2239:16-30, Springer, 2001.

[83] A. Vandevelde, H. Hoogeveen, C. Hurkens, J.K. Lenstra. Lower bounds for the head-body-tail problem on parallel machines: a computational study of the multiprocessor flow shop. *INFORMS J. on Computing* 17(3):305-320, 2005.

[84] F. Vanderbeck. Decomposition and colomn generation for Integer Programs. Ph.D. thesis, Faculté des Sciences Appliquées, Université Catholique de Louvain, Louvain-la-Neuve, 1994.

[85] F. Vanderbeck. Branching in branch-and-price: a generic scheme. Working Paper U-05.14, Applied Mathematics, University Bordeaux 1, 2005.

[86] P. Van Hentenryck, A general arc-consistency algorithm and its specializations. *Artificial Intelligence* 57(3):291-321, 1992.

[87] L.A. Wolsey. *Integer Programming.* John Wiley & Sons Inc., New York, 1998.