Solving an integrated job-shop problem with human resource constraints

Olivier Guyon, Pierre Lemaire, Eric Pinson, David Rivreau

Annals of Operations Research, Special Issue manuscript No.
(will be inserted by the editor)

Solving an integrated job-shop problem with human resource constraints

Olivier Guyon · Pierre Lemaire · Éric Pinson · David Rivreau

Received: date / Accepted: date

Abstract We propose two exact methods to solve an integrated employee-timetable and job-shop-scheduling problem. The problem is to find a minimum cost employee-timetable, where employees have different competences and work during shifts, so that the production, that corresponds to a job-shop with resource availability constraints, can be achieved. We introduce two new exact procedures: 1) a decomposition and cut generation approach and 2) a hybridization of a cut generation process with a branch and bound strategy. We also propose initial cuts that strongly improve these methods as well as a standard MIP approach. The computational performances of those methods on benchmark instances are compared to that of other methods from the literature.

Keywords Employee Timetabling Problem · Job-shop · Job-shop with resource availability constraints · Probing · Cut generation · Branch and Bound

Introduction

The purpose of a manufacturing factory is merely to produce some goods to meet some demand. Due to limited resources, an optimal production plan is usually hard to compute. The complexity lies mainly in two intertwined aspects: 1) a production schedule, that is an allocation of human and material resources to the different tasks (or jobs) that have to be processed, and 2) an employee timetable, that ensures that the human resources required by the production schedule are actually met.

Olivier Guyon
École des Mines de Saint-Étienne; Centre Microélectronique de Provence
880 avenue de Mimet 13541 Gardanne, France
E-mail: guyon@emse.fr

Pierre Lemaire
Grenoble-INP / UJF-Grenoble 1 / CNRS, G-SCOP UMR5272,
46 avenue Félix Viallet 38031 Grenoble cedex, France
E-mail: pierre.lemaire@grenoble-inp.fr

Éric Pinson · David Rivreau
PRES LUNAM Université /LISA EA4094 CNRS/Université Catholique de l'Ouest
3 place André-Leroy 49008 Angers, France
E-mail: {pinson,rivreau}@uco.fr

Except for special cases, each aspect is per se a difficult problem, as shows the huge literature on both scheduling problems (e.g., see Pinedo [Pin04] and Leung [Leu04]) and timetabling problems (e.g., see Ernst et al. [EJKS04] and Soumis et al. [SPR05] for states of the art). As a consequence, the resulting integrated problem has long been considered as too complex to be solved in practice and it is usually decomposed into an assignment part and a scheduling part, resulting in sub-optimal solutions.

Because of economical and financial pressure, higher performances must be sought for, through elaborate and adequate optimization procedures. That is why some recent efforts have been made to actually tackle the integrated problem, and this paper is one of them (indeed, this is the continuation of a previous work on a simpler problem [GLPR10] to try out the performances of cut generation processes for integrated problems).

In this paper, we propose two exact methods to solve an integrated employee-timetable and job-shob-scheduling problem. First, we provide a precise description of the problem we intend to solve, together with an integer linear formulation (Section 1), and we stress its links with the existing works (Section 2). Then, we introduce our two new procedures: a decomposition and cut generation approach (Section 3), and a hybridization of a cut generation process with a branch and bound strategy (Section 4). The computational performances of those methods on benchmark instances are compared to that of other methods from the literature (Section 5). Some conclusions are finally drawn in Section 6.

1 A model integrating an employee timetabling and a job-shop scheduling problems

1.1 Problem description

Our purpose is to solve two decision levels which interact in a global optimization process: a timetabling and a scheduling problem.

The job-shop problem consists in processing a set J of n jobs on a set K of m different machines. Each job i is made of a sequence of operations $O_{i1}, O_{i2}, \ldots, O_{im}$ which have to be scheduled according to a given order. Each operation O_{ij} has a processing time $p_{ij} \in \mathbb{N}$. It can be processed exactly by one of the m available machines. This performing machine is denoted by m_{ij}. For the sake of clarity, we denote by ρ_{ik} the processing time of job i on machine k. It is not allowed to preempt operations.

An operation can only be processed on machine k if an operator, able to use this resource, is available. We thus introduce a set E of μ operators where each operator e masters a subset K_e of machines. We assume that the operators work on a three-shift system. Consequently, the time horizon H we use is divided into a set S of σ consecutive and identical shifts s_0, s_1, …, $s_{\sigma-1}$. Each shift corresponds to a time period and has a fixed duration π (thus: $H = \sigma \cdot \pi$). Each employee e is assumed to be available on a subset of shifts S_e. The cost of its assignment to machine k during shift s is denoted by c_{eks}. Furthermore, regulation constraints impose that an operator must not work more than one shift among each triplet of consecutive shifts.

The objective of the problem is to find a minimum cost assignment of operators to both machines and shifts in such a way that a feasible operating sequence exists for each machine for a target makespan $C_{max} \leq H$ (maximum of the completion times of operations).

A small illustrative instance is provided by Example 1 (shown in Figure 1). It consists of 2 jobs, 2 machines, 3 employees and 3 shifts. The shift duration is $\pi = 8$ hours ($H = 24$). We want to schedule each operation before the end of the day, i.e., $C_{max} = 24$.

In Figure 1.a), the job data (machines and durations) are provided. Employee data are displayed in Figure 1.b). The costs to assign an employee to this machine are displayed for all shifts for which the employee is assumed to be available.

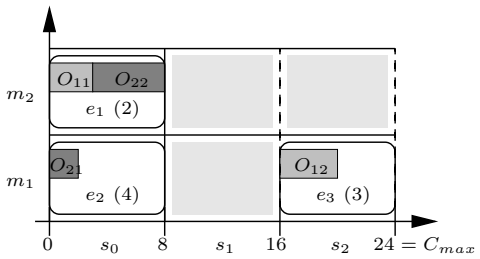
Figure 1.c) illustrates an optimal solution (with a total cost of 9) for Example 1. In this solution, a production plan with a makespan of 20, lower than C_{max}, is obtained. Employee e_1 (respectively e_2) is assigned to machine m_2 (resp. m_1) during s_0 in order to perform operations O_{11} and O_{22} (resp. O_{21}). During the last shift s_2, operation O_{12} is processed on m_1 by e_3.

i	m_{ij}	p_{ij}
1	2 1	3 4
2	1 2	2 5

a) Job data

e	S_e	K_e	c_{eks}
1	0 1 2	2	2 4 7
2	0 2	1	4 - 3
3	1 2	1	- 5 3
		2	- 6 8

b) Employee data



c) An optimal solution

Fig. 1 Example 1

1.2 An integer linear programming model

In this section, we propose an integer linear programming model $[P]$ associated with the problem at hand. For the sake of clarity, this mathematical formulation is split into four parts: the objective-function, the employee timetabling sub-problem, the job-shop sub-problem, and the coupling constraints.

Beforehand, we define the release date, or head (respectively the latency duration, or tail), denoted r_{ik} (respectively d_{ik}), associated with the processing of job i on machine k. These dates are simply induced by the conjunctive constraints associated with the job sequences. They can be computed in $O(n \cdot m)$ time using the following recurrence relations:

$$\begin{cases} r_{ik} = 0 & i = 1,\ldots,n \quad k = m_{i1} \\ r_{il} = r_{ik} + \rho_{ik} & i = 1,\ldots,n \quad j = 1,\ldots,m-1 \quad k = m_{ij} \quad l = m_{i(j+1)} \end{cases} \tag{1}$$

$$\begin{cases} d_{ik} = C_{max} & i = 1,\ldots,n \quad k = m_{im} \\ d_{ik} = d_{il} - \rho_{il} & i = 1,\ldots,n \quad j = 1,\ldots,m-1 \quad k = m_{ij} \quad l = m_{i(j+1)} \end{cases} \tag{2}$$

Let us now define the two groups of binary decision variables involved in the model:

- $x_{eks} = 1$ if and only if operator e is assigned to machine k during shift s, 0 otherwise
- $y_{ikt} = 1$ if and only if job i is processed on machine k at time t, 0 otherwise

Objective-function

$$[P] \quad \min \Theta = \sum_{e \in E} \sum_{k \in K_e} \sum_{s \in S_e} c_{eks} \cdot x_{eks} \tag{3}$$

Employee timetabling specific constraints

$$\sum_{k \notin K_e} \sum_{s=0}^{\sigma-1} x_{eks} = 0 \qquad e = 1, \ldots, \mu \tag{4}$$

$$\sum_{k \in K_e} \sum_{s \notin S_e} x_{eks} = 0 \qquad e = 1, \ldots, \mu \tag{5}$$

$$\sum_{k \in K_e} \left(x_{eks} + x_{ek(s+1)} + x_{ek(s+2)} \right) \leq 1 \quad e = 1, \ldots, \mu \quad s = 0, \ldots, \sigma - 3 \tag{6}$$

$$x_{eks} \in \{0,1\} \qquad e = 1, \ldots, \mu \quad k = 1, \ldots, m \quad s = 0, \ldots, \sigma - 1 \tag{7}$$

Job-shop specific constraints

$$\sum_{t=0}^{d_{ik}-\rho_{ik}} t \cdot y_{ikt} + \rho_{ik} \leq C_{max} \quad i = 1, \ldots, n \quad k = m_{im} \tag{8}$$

$$\sum_{t=r_{ik}}^{d_{ik}-\rho_{ik}} y_{ikt} = 1 \qquad i = 1, \ldots, n \quad k = 1, \ldots, m \tag{9}$$

$$\sum_{t=0}^{r_{ik}} y_{ikt} + \sum_{t=d_{ik}-\rho_{ik}+1}^{C_{max}} y_{ikt} = 0 \quad i = 1, \ldots, n \quad k = 1, \ldots, m \tag{10}$$

$$\sum_{u=r_{ik}+\rho_{ik}}^{t} y_{ilu} - \sum_{u=r_{ik}}^{t-\rho_{ik}} y_{iku} \leq 0 \quad i = 1, \ldots, n \quad j = 1, \ldots, m-1 \quad k = m_{ij} \quad l = m_{i(j+1)} \quad t = r_{ik} + \rho_{ik}, \ldots, d_{il} - \rho_{il} \tag{11}$$

$$\sum_{i=1}^{n} \sum_{u=\max(r_{ik}, t-\rho_{ik}+1)}^{\min(d_{ik}-\rho_{ik}, t)} y_{iku} \leq 1 \quad k = 1, \ldots, m \quad t = 0, \ldots, C_{max} \tag{12}$$

$$y_{ikt} \in \{0,1\} \qquad i = 1, \ldots, n \quad k = 1, \ldots, m \quad t = 0, \ldots, C_{max} \tag{13}$$

Coupling constraints

$$\sum_{e \in E} x_{eks} - \sum_{i=1}^{n} \sum_{u=\max(r_{ik}, t-\rho_{ik}+1)}^{\min(d_{ik}-\rho_{ik}, t)} y_{iku} \geq 0 \quad k = 1, \ldots, m \quad t = 0, \ldots, C_{max} \quad s = \lfloor t/\pi \rfloor \tag{14}$$

In this formulation, assignment variables x_{eks} are fixed to 0 if employee e does not master resource k (4) or is not available during shift s (5) [1]. Constraints (6) formalize the regulation rules stating that an employee must not work more than one shift among each triplet of consecutive shifts. Constraints (8) ensure that all jobs are completed prior to the target makespan C_{max}. Each operation has to be processed within its time window (9)-(10) and cannot start prior to the completion of its job predecessor (11). Furthermore, at most one operation can be processed on a given machine at each instant (12). Lastly, coupling constraints

[1] In our implementation, those variables are not created. Variables x_{eks} are thus created if and only if the employee e can work on machine k ($k \in K_e$) and is available on shift s ($s \in S_e$)

(14) ensure that an operation on a machine can be processed only if a qualified employee is assigned to the machine at hand.

This model is at the core of this work: both procedures of Sections 3 and 4 aim at solving it, using different decomposition strategies.

For the scheduling stage, one may remark that operators may be unavailable and/or not assigned to some machines for some shifts over the time horizon. Therefore, we have to take into account for each machine a collection Υ of time periods on which it cannot process any operation. Obviously, these periods depend on an instantiation $\bar{x}$ of decision variables x_{eks}, since $\bar{x}$ corresponds to a particular assignment of operators to both machines and shifts. This leads to the so called *job-shop scheduling problem with availability constraints*.

However the problem can be reduced to a classical job-shop scheduling problem by creating additional fictional jobs, as stated in the following proposition:

Proposition 1 *Let us consider a time slot $\upsilon = [\![\alpha_\upsilon, \beta_\upsilon]\!]$ corresponding to an unavailability period on machine k. To take into account this fixed inactivity period is equivalent to consider the additional job i_F composed of m operations with execution domain $[\![0, C_{max}]\!]$ and a null processing time except one of them (without loss of generality the first one) with execution domain $[\![\alpha_\upsilon, \beta_\upsilon]\!]$ and a duration equals to $\rho_{i_F k} = (\beta_\upsilon - \alpha_\upsilon)$.*

Proof Operations of i_F with an execution domain $[\![0, C_{max}]\!]$ and a null processing time can be processed at any moment. Scheduling i_F thus only depends on the scheduling of the only operation with a non-null processing duration, i.e., the one which has to be processed on k during υ. Let us denote by $O^k_{i_F}$ this specific operation. Because all other operations of i_F have a null duration, $O^k_{i_F}$ can be placed at any rank of the sequence of operations of i_F.

$O^k_{i_F}$ has a duration $(\beta_\upsilon - \alpha_\upsilon)$ and a processing domain $[\![\alpha_\upsilon, \beta_\upsilon]\!]$. Scheduling $O^k_{i_F}$ therefore implies that machine k is fully dedicated to $O^k_{i_F}$ during υ. No *real* job can indeed be processed on k during υ. Machine k is in fact not used during υ. This complete the proof.

In the following, we will denote by J_F the set of n_F fictional jobs associated with the fixed inactivity periods of Υ, with $n_F = \text{card}\{\Upsilon\}$. Given an assignment vector $\bar{x}$, the scheduling component associated with our problem consists in checking whether a feasible operating sequence associated with both real and fictional jobs $(J \cup J_F)$ exists for each machine and for the target makespan C_{max}.

2 Existing works

As it has already been mentioned, the integrated approach is usually considered as too complex to be solved in practice and, as a consequence, much of the literature deals either with production scheduling problems (e.g., see [Pin04, Leu04] for states of the art) or with timetabling problems (e.g., see [EJKS04, SPR05]). In what follows, the main works dealing with integrated problems and/or decomposition procedures are presented in Subsection 2.1.

Subsection 2.2 is dedicated to solution procedures for job-shop problems.

2.1 Integrated problems and decomposition procedures

Only few attempts exist that cope with an integrated solution method. To the best of our knowledge, the existing works either deal with different production systems and/or solve it using fundamentally different approaches.

In Daniels and Mazzola [DM94], flow-shop scheduling contexts are considered, and the authors assume that each operation must be processed by its machine parallel to a set of operators devoted to this operation during its entire processing time. Moreover, each employee can only be assigned to a subset of operations (because of skills considerations) and no shift partitioning of the time horizon is considered. Consequently, operators can be simply considered as additional parallel machines. They propose an enumerative approach of brand and bound type for solving this problem.

In the context of project scheduling, Alfares and Bailey [AB97] propose an integer linear programming model and dynamic programming based heuristics. Their problem is rather simple since the only constraints are precedences among operations, and that the volume of human resources on any given day is big enough for the operations of that day.

More recently, Chen [Che04] considers a parallel machine scheduling problem that involves job scheduling coupled to a resource allocation component; the processing times are inversely related to the amount of resources allocated and the objective is to minimize the total cost associated with both scheduling and resource allocation. A column generation based branch and bound is proposed for solving this problem.

Several other authors use decomposition procedures. Such procedures originate from Benders decomposition [Ben62]. In its basic version, it is a generic method for solving problems that contain groups of variables of different natures (e.g., MIPs with integer and continuous variables). The key idea is to assign some trial values to a group of variables and to find the best solution consistent with this particular instanciation. The major underlying idea relies on dual inference with respect to Lagrangian relaxation and Dantzig-Wolfe decomposition. Thus, the key element of this approach is the derivation of Benders cuts that exclude superfluous solutions. Classical Benders cuts are formulated by solving the dual of the subproblem obtained when the trial values are fixed, by exploiting a "nice" substructure in the global problem. Such an approach is commonly used for problems involving strategic as well as operational decisions.

During the last two decades, extensions of this powerful approach have been proposed: in particular, hybridization with constraint programming techniques or logic based Benders decompositions. One of the main pioneering works exploiting these ideas is due to Hooker [HO03,Hoo05,Hoo07] jointly with some other contributors. For instance, in [Hoo05,Hoo07], an hybrid method combining constraint programming techniques, mixed integer linear programming and logic based Benders decompositions is designed for solving a "planning and scheduling problem" in which tasks are to be assigned to facilities with some consumption considerations.

Similar solution procedures have been used by Artigues et al. [AGRV09] to solve a model linking a job-shop scheduling stage to an employee timetabling component. To the best of our knowledge, their model is the previously studied model that is the closest to the one developed in this paper. More precisely, a set A of additional activities to be completed by a set E of operators is introduced. The subset of activities each operator is able to perform is known. Each operation associated with the job-shop problem is then assumed to require, during its processing, a predefined number of these employees for each activity $a \in A$. As in our case, the time horizon is assumed to be partitioned in identical shifts. Clearly, the processing of a given operation O_{ij} during shift s induces a fixed requirement δ_{as} of operators devoted to the execution of each activity a on this particular time slot. Notice that this model allows an operator to work simultaneously on several machines. Moreover, an employee shift can cover more than one scheduling period. As a consequence, the authors allow the aggregation of activity demands for the operations processed during a given shift.

A lexicographic optimization problem where the makespan minimization is the primary objective whereas the employee cost minimization is the secondary objective is considered. The problem tackled in our study corresponds to the second stage of the problem described in [AGRV09].

As for the solution procedure, Artigues et al., unlike we, heavily rely on constraint programming. Indeed, two constraint programming based formulations, in association with linear programming relaxations, are proposed. The first (direct) constraint programming formulation models both job-shop scheduling and employee timetabling components by means of global constraints. A linear programming relaxation based on the employee timetabling part is used for search tree pruning and reduced cost reduction is proposed. The second constraint programming formulation relies on a decomposition involving the domains of the starting time variables, and leading to the definition of demand intervals associated with each activity $a \in A$. A new linear programming relaxation, including the scheduling component, is introduced by means of additional binary variables. Energetic reasoning feasibility checking techniques can thus be exploited. Both constraint programming formulations are solved using a backtrack search strategy.

In [GLPR10], Guyon et al. investigate the integration of an employee timetabling and a production scheduling problems. At the first level, they manage a classical employee timetabling problem whereas at the second level, they aim at supplying a feasible production schedule for a set of interruptible tasks with qualification requirements and time-windows. They propose two exact methods to solve the resulting problem. The former is based on a Benders decomposition while the latter relies on a specific decomposition and a cut generation process. Although this second approach has some basic similarities with the first one presented in this paper, most of the key components (cut structure, master and slave problems, ...) differ. In particular, the scheduling stage in [GLPR10] (scheduling on parallel machines with preemption) can be reduced to a polynomially solvable flow problem, while in the present work the scheduling stage (a job-shop) is NP-complete and notoriously hard to solve in practice.

2.2 Solution procedures for job-shop problems

As it is mentioned above, the scheduling stage is a job-shop problem, which is an NP-complete problem. Hence an important issue is how to solve it efficiently. For this particular sub-problem, we rely on the existing litterature.

More precisely, we want to solve a job-shop with availability constraints. This is NP-hard since job-shop without unavailability periods is already strongly NP-hard [RK76,GJ79]. Due to their evident utility for applications, scheduling problems with availability constraints are specifically adressed in the scheduling literature [BFHS88, DM94, Lee96, SS98, Sch00, Agg04a, GH05, WSC05, LS08, MCZ10].

In particular, Mauguière et al. [MBB05] show that 5 categories of the job-shop scheduling problem with availability constraints have to be distinguished. The type of problems we consider in this paper corresponds to the class denoted $JP2$ which refers to the job-shop problem with *non-crossable* unavailability periods and *non-resumable* operations. In [Agg02], Aggoune defines a method (based on the search of a shortest path problem) to solve up to optimality problems of $JP2$ with only two jobs. He also proposes an exact branch and bound method (that exploits the classical disjunctive graph representation of the job-shop problem [RS64]) for any problem of $JP2$ [Agg04b]. Other effective attempts [MBB05,

Zri05] to solve the job-shop scheduling problem with availability constraints clearly show the growing interest of researchers for it.

As said before, the scheduling part of our problem can be seen as a classical job-shop problem with fictional jobs. Since the concern of our paper was not to investigate that specific sub-problem but mainly to deal with the integrated problem, we chose to use the dedicated job-shop solver of [Riv99] as a black box. In brief, this solver is a classical branch and bound process where the key point is the use of the immediate selections on disjunctions [Car75], the immediate selections on ascendant/descendant sets [CP89, CP90] and the shaving procedure [CP94, MS96] as elimination rules. A survey on these elimination rules can be found in [CPPR04]. Since the aim of the solver is either to find a solution within a given C_{max} or to prove that there is no such solution in a given amount of CPU time, in each node we try to build a feasible solution using serial scheduling based on the current deadlines of operations. If this procedure fails, we relax the precedence constraints on jobs and we build a feasible solution on each machine (using [Car82]). These solutions provide us with earliest starting times and latest starting times for each operation. Clearly, most of the time, these dates do not fulfill the precedence relations constraints of jobs (otherwise we have a feasible solution and we are done). So in order to identify the greatest violation, the earliest starting times and latest starting times are amended through the precedence relations of each job by a forward and backward process. This gives us an evaluation of violation for each job: we select the job with the lowest compliance level and then we develop two nodes by halving the time window of one of its operations.

From a functional point of view, the solver is given a CPU time limit, so it can fail to prove that there is a solution or not (job-shop is not a particularly easy scheduling problem). We can also select to use the shaving procedure or not: indeed, if for hard problems this elimination procedure is essential, for easy problem it induces a CPU over-consumption that may be sometimes avoided.

3 A decomposition and cut generation approach

3.1 Problem decomposition

Due to the intrinsic two-level decision structure of $[P]$, it seems quite natural to investigate decomposition methods based on the splitting of $[P]$ into two interacting sub-problems:

- A master problem corresponding to the employee timetabling component of $[P]$: $[ETP]$
- A slave or satellite problem corresponding to the scheduling component of $[P]$: $[JobShop]$

This approach clearly relies on the relaxation of the coupling constraints (14).

3.2 Master problem

The master problem $[ETP]$ is an employee timetabling problem that attempts to find a minimal cost assignment of operators to both machines and shifts, where coupling constraints binding operators and machines are relaxed. This problem, relying only on the decision variables x_{eks}, is stated as follows:

$$[ETP] : \min \Theta = \sum_{e \in E} \sum_{k \in K_e} \sum_{s \in S_e} c_{eks} \cdot x_{eks}$$
$$Cut$$
$$(4),(5),(6),(7)$$

where *Cut* denotes a set of valid inequalities that are iteratively added to the model. They invalidate solutions of $[ETP]$ which do not lead to a feasible solution for the scheduling component of $[P]$.

3.3 Slave problem

Let us denote by $\bar{x}$ an optimal solution to the current master problem $[ETP]$. Clearly, $\bar{x}$ is a particular assignment of operators to both machines and shifts. Moreover, $\bar{x}$ is a feasible solution for the global problem $[P]$ if it satisfies the scheduling component $[JobShop]$. To perform this test, we first define aggregate information based on $\bar{x}$ in the following way :

$$\bar{z}_{ks} = \min(\sum_{e \in E} \bar{x}_{eks}, 1) \quad k = 1, \ldots, m \quad s = 0, \ldots, \sigma - 1 \tag{15}$$

Clearly, $\bar{z}_{ks} = 1$ if at least one operator is assigned to machine k during shift s, 0 otherwise (the machine k is not used). Each machine requires an employee for its use. Therefore, machine k can be used during shift s if and only if $\bar{z}_{ks} = 1$. Considering those entries, checking whether or not $[JobShop]$ is feasible for a given assignment $\bar{x}$ can be performed by solving the following decision problem:

$[JobShop(\bar{z})]$: *Is it possible to find an assignment of decision variables satisfying:*

$$\sum_{i=1}^{n} \sum_{t=\max(r_{ik}, \bar{\upsilon} \cdot \pi - \rho_{ik})}^{\min(C_{max}, (\bar{\upsilon}+1) \cdot \pi)} y_{ikt} = 0 \;\; k = 1, \ldots, m \quad \bar{\upsilon} \in \bar{S}_k \tag{16}$$
$$(8),(9),(10),(11),(12),(13)$$

where $\bar{S}_k = \{s \in S | \bar{z}_{ks} = 0\}$ denotes the set of time slots during which machine k cannot be used with respect to employee resources defined by $\bar{z}$.

$[JobShop(\bar{z})]$ is a job-shop scheduling problem with fixed inactivity periods. It can be reduced to a classical job-shop scheduling problem (see Proposition 1, Section 1.2). The only difference with the job-shop specific constraints presented in Section 1.2 relies on constraints (16). These constraints prevent, with respect to $\bar{z}$, the use of any machine during a shift for which no operator has been assigned.

3.4 Cut generation process

Let us denote by $\bar{z}$ the vector determined, according to (15), by a feasible solution $\bar{x}$ of $[ETP]$. If $[JobShop(\bar{z})]$ is unfeasible, it implies that jobs cannot be processed with the available resources defined by $\bar{z}$. Consequently, $[JobShop(\bar{z})]$ fails when $\bar{x}$ does not lead to a feasible solution for problem $[P]$. $\bar{x}$ must therefore be discarded from the set of feasible solutions.

Trivially, $\bar{z}$ (and thus $\bar{x}$) is unfeasible because at least one machine among those who are temporarily not used should be available for executing a job. A valid cut translating this simple finding in mathematical terms is:

$$\sum_{e \in E} \sum_{k \in K_e} \sum_{s \in S_e} \bar{\beta}_{ks} \cdot x_{eks} \geq 1 \tag{17}$$

where $\bar{\beta}_{ks} = 1$ if $\bar{z}_{ks} = 0$, 0 otherwise.

This inequality should be added to the pool *Cut* of the master problem $[ETP]$. Since there is a finite number of eligible assignments of employees to both machines and shifts, the process stops in a finite number of steps.

3.5 Initializing the pools of valid inequalities

3.5.1 Introduction

In order to speed up the convergence of the process in such cut generation approaches, it is often interesting to initialize the pool of cuts with a subset of inequalities allowing to (partially) recover the feasibility when solving the master problem. In this section, we propose three different valid inequalities leading to such an initialization of the set *Cut* of $[ETP]$.

3.5.2 Probing cuts

Basically, probing refers to an elimination technique used for enhancing MIP solvers [Sav94]. The underlying idea is to fix a decision variable to one of its bounds, and to measure the consequence in terms of logical implications. If fixing this variable leads to an unfeasibility, it clearly implies that no solution with the current variable value exists. In some cases, some variables (namely binary variables) can thus be definitively fixed.

In our approach, this idea is exploited to identify shifts during which some machines must be used. For this purpose, we specify for each machine k the list Υ^k of shifts during which k cannot be used. The exhaustive list of shifts where there is a machine which cannot be used is denoted by $\Upsilon = \cup_{k=1}^{m} \Upsilon^k$. Notice that according to proposition 1 (see Section 1.2), each inactivity shift s on machine k can be modeled by an additional fictional job. Clearly, modeling this inactivity by a fictional job is equivalent to set $\bar{z}_{ks} = 0$. If the resulting job-shop scheduling problem (with fixed inactivity periods) is unfeasible, then at least one inactivity pair $(\bar{k}, \bar{s})$ involved in Υ must be removed, i.e., the associated machine $\bar{k}$ must be used during shift $\bar{s}$. That leads to the following so-called probing cut:

$$\sum_{e \in E} \sum_{k \in K_e} \sum_{s \in S_e} \alpha_{ks} \cdot x_{eks} \geq 1 \tag{18}$$

where $\alpha_{ks} = 1$ if $s \in \Upsilon^k$, 0 otherwise.

In our experiments, and for obvious computational considerations, we focused only on configurations (machine k, shift s) of Υ corresponding to singletons (e.g. (k_1, s_3)) or pairs of couples (e.g., $(k_1, s_3) \wedge (k_2, s_0)$). Notice that if a probing cut is generated for a given singleton, this implies that machine k has to be used during shift s. In the following, such singleton will be refered to as *fixed by probing*.

3.5.3 Minimal number of working shifts per machine

The valid initial inequalities presented in this section are based on the computing of a lower bound LB_k for each machine k representing the minimum number of worked shifts which are required to ensure the feasibility of $[P]$. The corresponding cut simply makes sure that machine k will be used during at least LB_k shifts:

$$\sum_{e \in E | k \in K_e} \sum_{s \in S_e} x_{eks} \geq LB_k \quad k = 1, \ldots, m \tag{19}$$

LB_k relies on the computing of an upper bound $\sigma' \leq \sigma$ on the maximal number of shifts during which machine k can be used. By symmetry, LB_k can hence be defined as follows: $LB_k = (\sigma - \sigma')$. It is computed by Algorithm 1.

Algorithm 1 Computing of LB_k

$\sigma' \leftarrow \max(LB_k^1, LB_k^2)$
unfeasible $\leftarrow$ false
repeat
  $O_F \leftarrow \sigma'$ fictional operations which have to be processed on k for a duration of π
  $[JobShop_F] \leftarrow$ job-shop problem that aims to schedule each job $i \in J$ and each operation of O_F
  if $[JobShop_F]$ is unfeasible **then**
    unfeasible $\leftarrow$ true
  else
    $\sigma' \leftarrow \sigma' + 1$
  end if
until (unfeasible) $\vee$ $(\sigma' > \sigma)$
$LB_k \leftarrow \sigma - \sigma'$

At the beginning of Algorithm 1, we use two auxiliary lower bounds: LB_k^1 and LB_k^2. Indeed, computing LB_k from scratch can be time-consuming and those auxiliary bounds greatly improve the performances. They are defined as follows:

– LB_k^1 is the number of pairs (machine k, shift s) *fixed by probing* (see 3.5.2):

$$LB_k^1 = card\{s \in S | (k,s) \text{ is fixed by probing}\}$$

– LB_k^2 is based on the sum of processing times of operations which have to be scheduled on machine k:

$$LB_k^2 = \left\lceil \frac{\sum_{j \in J} \rho_{ik}}{\pi} \right\rceil$$

3.5.4 Preventing multiple assignments

The two groups of cuts defined above do not prevent from generating solutions assigning multiple operators to the same pair (machine, shift). Such solutions are clearly sub-optimal since assignment costs c_{eks} are strictly positive and a machine needs only one operator to be handled. This third collection of initial cuts ensures that at most one operator is assigned to any pair (machine, shift) :

$$\sum_{e \in E | (k \in K_e) \wedge (s \in S_e)} x_{eks} \leq 1 \quad k = 1, \ldots, m \quad s = 0, \ldots, \sigma - 1 \tag{20}$$

Even if they could seem redundant, according to our experiments, these additional cuts are quite efficient in solving the master problem $[ETP]$.

3.5.5 Algorithm

The overall process underlying the cut generation approach presented in this section is summarized in Algorithm 2:

Algorithm 2 Exact cut generation process

```
LB ← 0
repeat
  x̄ ← optimum of [ETP], with a cost: Θ_x̄
  if x̄ is defined then
    LB ← Θ_x̄
    ∀(k,s) ∈ {K × S}   z̄_ks ← min(Σ_{e∈E} x̄_eks, 1)
    feasible ← solve[JobShop(z̄)]
    if ¬ feasible then
      add cut (17) to the pool Cut of [ETP]
    end if
  end if
until (feasible) ∨ (x̄ is not defined)
return LB
```

4 An hybridization of a cut generation process with a branch and bound strategy

4.1 Introduction

In this section, we present a second exact solution strategy. Two classic approaches for tackling hard combinatorial optimization problems are hybridized: branch and bound and cut generation process. Like in the decomposition and cut generation process (see Section 3), we also decompose the problem $[P]$ into two independent sub-problems. The method's main particularity relies on the fact that the global process control is ensured by a vector $\bar{z}$ of *indicatrices* which decides whether a machine k is available for processing jobs during a given shift or not. $\bar{z}$ is instantiated by an enumerative process.

The underlying global process is explained in Section 4.2. The main components associated with the branch and bound approach are detailed in sub-Sections 4.2.1 to 4.2.4, while implementation issues and the overall algorithm is presented in sub-Section 4.2.5.

4.2 The global process

Let us define, for any couple (k,s) of machine/shift, an *indicatrice* representing the availability of machine k during shift s (implying that at least one operator is assigned to k during s):

$$\forall (k,s) \in \{K \times S\} \quad \bar{z}_{ks} = \begin{cases} -1 \text{ if machine } k \text{ is not constrained during shift } s \\ \ \ 0 \text{ if machine } k \text{ must not be used during shift } s \\ +1 \text{ if machine } k \text{ must be used during shift } s \end{cases}$$

In this alternative approach, the vector $\bar{z}$ is exploited to guide the global process, and its instantiation is performed by means of an enumerative approach of branch and bound type, as described in Sections 4.2.1 to 4.2.4.

Thus, each node of the search tree corresponds to a partial availability planning on the set of machines M. A given vector $\bar{z}$ leads to two sub-problems $[ETP(\bar{z})]$ and $[JobShop(\bar{z})]$ derived from the decomposition principle exploited previously. The employee timetabling sub-problem $[ETP(\bar{z})]$ can be formulated as follows:

$$
\begin{aligned}
[ETP(\bar{z})] : \min \Theta_{\bar{z}} = & \sum_{e \in E} \sum_{k \in K_e} \sum_{s \in S_e} c_{eks} \cdot x_{eks} \\
& \sum_{e \in E} x_{eks} = \bar{z}_{ks} \qquad \forall (k,s) \in \{\{M \times S\} \,|\bar{z}_{ks} \neq -1\} \qquad (21) \\
& \textit{Cut} \\
& (4),(5),(6),(7)
\end{aligned}
$$

where *Cut* denotes a set of valid inequalities (17), already defined in Section 3, that are iteratively added to the model.

Clearly, $[ETP(\bar{z})]$ aims at finding a minimal cost assignment of operators to both machines and shifts, the solution space being limited to assignments x consistent with constraints (21). $[JobShop(\bar{z})]$ is exactly the same as in Section 3.3.

In our approach, $[JobShop(\bar{z})]$ is first solved. If it is feasible, the linear relaxation of $[ETP(\bar{z})]$ is then solved up to optimality thanks to a LP solver. The corresponding optimal solution clearly gives a lower bound $LB(\bar{z})$ associated with the current node $(\bar{z})$.

Let us denote by UB the value of the incumbent solution (best solution found so far). If either 1) $LB(\bar{z}) > UB$, 2) the LP-relaxation of $[ETP(\bar{z})]$ has no feasible solution or 3) $[JobShop(\bar{z})]$ is unfeasible, then the current distribution $\bar{z}$ is not consistent. The associated node in the enumeration search tree is discarded and a backtracking occurs.

Otherwise, the collection of implication rules detailed in Section 4.2.3 is performed, as well as a consistency test consisting in forcing any non constrained pair (machine k, shift s) ($\bar{z}_{ks} = -1$) to be inactive ($\bar{z}_{ks} = 0$), and solving the resulting sub-problems $[ETP(\bar{z})]$ and $[JobShop(\bar{z})]$. If the solution $\bar{y}$ of $[JobShop(\bar{z})]$ is once again feasible, then the optimal solution $(\bar{x})$ of $[ETP(\bar{z})]$, if defined, yields a feasible solution $(\bar{x},\bar{y})$ of value $\Theta_{\bar{z}}$ to the global problem $[P]$. If $\Theta_{\bar{z}} < UB$, then an improved solution to $[P]$ has been exhibited: it becomes the new incumbent solution, and UB is updated. Otherwise, a cut (17) is added to the pool *Cut* in problem $[ETP]$. Notice that inequalities (17) are valid for any node of the search tree.

Section 4.2.5 provides the detailed implementation issues related to this procedure.

4.2.1 Branching strategy

The enumerative strategy we use is a binary branching scheme. To expand the current active node, a pair (machine k, shift s) is selected by the means of a branching heuristic. Two child nodes are then created, corresponding respectively to $\bar{z}_{ks} = 0$ (machine k must not be used during shift s) and $\bar{z}_{ks} = 1$ (machine k must be used during shift s).

In our implementation, the node associated to $\bar{z}_{ks} = 0$ is systematically explored first. The reason for this choice is that prohibiting machine activity leads to a more constrained problem, and thus to a much more reduced search tree size.

The branching heuristic we use selects the pair of machine/shift $(\bar{k},\bar{s})$ defined as follows:

– $\bar{k}$ is the machine $k \in K$ with the minimal difference between the number of available shifts and the lower bound LB_k of the number of worked shifts on k (see 3.5.3):

$$\bar{k} = \text{argmax}_{k \in K | \exists s \in S, \bar{z}_{ks} = -1} \left(\text{card} \left\{ (k,s) \in \{K \times S\} \, | \bar{z}_{ks} \in \{-1,1\} \right\} - LB_k \right)$$

– $\bar{s}$ is the most *tardive* shift such that no decision has been taken on pair $(\bar{k}, \bar{s})$:

$$\bar{s} = \text{argmax} \left(s \in S | \bar{z}_{\bar{k}s} = -1 \right)$$

Several other branching heuristics have been investigated. The one presented above has been the most efficient in the experiments.

4.2.2 Lower bound

As mentioned before (see 4.2), $[ETP(\bar{z})]$ aims at finding a minimal cost assignment of operators to both machines and shifts consistent with the current distribution $\bar{z}$. Clearly, an optimal solution to problem $[ETP(\bar{z})]$ gives a valid lower bound associated to the related node in the search tree. For obvious computational reasons, we only solve the linear relaxation of $[ETP(\bar{z})]$ for each node of the search tree.

4.2.3 Implication rules

In order to reduce the search space and thus to speed up our process, two implication rules have been defined.

Probing based implication rule The first implication rule is based on the probing strategy detailed in Section 3.5.2. It aims at checking if successive decisions about the absence of work for some pairs (machine, shift) are not too restrictive. Such restrictions could indeed prevent any descendant of the branching node from leading to a feasible complete solution. For this purpose, each non constrained pair (machine, shift) in the current distribution $\bar{z}$ (i.e., $\bar{z}_{ks} = -1$) is in turn forced to inactivity ($\bar{z}_{ks} = 0$). The resulting sub-problem $[JobShop(\bar{z})]$ is solved, and if unfeasibility is detected, we can set $\bar{z}_{ks}$ to 1, i.e., any subsequent distribution in the sub tree $\bar{z}$ satisfies that machine k is used.

Implication rule based on the minimal number of working shifts per machine This implication rule ensures that, for each machine $k \in K$, the lower bound LB_k on the minimal number of working shifts (see 3.5.3) is verified. This logical rule can be expressed as follows:

$$\left(\text{card} \left\{ s \in S | \bar{z}_{ks} \neq 0 \right\} = LB_k \right) \Rightarrow \left(\bar{z}_{ks} = 1 \quad \forall s \in \left\{ S | \bar{z}_{ks} = -1 \right\} \right)$$

4.2.4 Consistency checking

As mentioned in Section 4.2, we perform a consistency test for each *non pruned* node of the search tree. This test aims at checking if the partial distribution $\bar{z}$ associated with the current branching node is already a complete feasible solution for the problem.

To do so, we first get a complete instantiation of $\bar{z}$ by setting $\bar{z}_{ks} = 0$ for any pair (machine k, shift s) that the current partial distribution $\bar{z}$ does not constraint (i.e., $\bar{z}_{ks} = -1$). For the resulting complete distribution $\bar{z}$, we check the feasibility of both $[ETP(\bar{z})]$ and $[JobShop(\bar{z})]$. If these two sub-problems are feasible, the associated optimal solutions yield a feasible solution for the global problem $[P]$. They can hence be compared to the best incumbent solution. Otherwise, adding the cut (17) to the pool *Cut* of valid inequalities in problem $[ETP]$ prevents us from generating such a non consistent distribution by enumeration.

4.2.5 Algorithm

The hybrid *branch and bound / cut generation* method described above is summarized in Algorithms 3 to 8. Algorithm 3 is the main algorithm; it refers to algorithms 4 to 8 described hereafter. In our implementation, we used the two following specific data structures:

- P: a typical stack which is used to store the nodes of the search tree in a Last In, First Out (LIFO) order
- γ: a node of the search tree. Each node γ is characterized by 5 attributes:
  - $k \in M$: the machine of the selected branching variable used to construct γ
  - $s \in S$: the shift of the selected branching variable used to construct γ
  - *explored* $\in \{$*true, false*$\}$. If *true*, the node has already been explored and its two children have been (if necessary) generated. If *false*, it is the first time the node is met.
  - *value* $\in \{-1, 0, 1\}$: fixed value for the branching variable used to construct γ
  - *decisions*: list of the decisions (due to the branching variable selection and the implications rules) related to node γ. By construction, each descendant node of γ respects all these decisions.

Algorithm 3 Hybrid Branch and Bound and cut generation procedure

$UB \leftarrow +\infty$
$\forall k \in M \; \forall s \in S \; \bar{z}_{ks} \leftarrow -1$
$P \leftarrow \{\text{Node (k=0, s=0, value=-1, explored=false, decisions=}\emptyset\text{)}\}$
repeat
  $\gamma \leftarrow$ node at the top of P
  if $\neg$ γ.explored **then**
    γ.explored $\leftarrow$ true
    $\bar{z}_{\gamma.k\gamma.s} = \gamma.value$
    if $[JobShop(\bar{z})]$ is not unfeasible **then**
      imply$(\gamma, \bar{z})$
      f $\leftarrow$ evaluate$(\gamma, \bar{z})$
      if $(f \neq -1) \wedge (f < UB)$ **then**
        $\forall (k,s) \in \{K \times S\} \quad (\bar{z}_{ks})' \leftarrow \bar{z}_{ks}$
        $\forall (k,s) \in \{K \times S | \bar{z}_{ks} = -1\} \quad \bar{z}_{ks} \leftarrow 0$
        $\bar{x} \leftarrow$ optimal solution of $[ETP(\bar{z})]$; cost: $\Theta_{\bar{z}}$
        if $(\bar{x}$ exists$) \wedge (\Theta_{\bar{z}} < UB) \wedge ([JobShop(\bar{z})]$ is feasible$)$ **then**
          $UB \leftarrow \Theta_{\bar{z}}$
        else
          add cut (17) to the set *Cut* of $[ETP]$
        end if
        $\forall (k,s) \in \{K \times S\} \quad \bar{z}_{ks} \leftarrow (\bar{z}_{ks})'$
        if $\neg$ leaf$(\bar{z})$ **then**
          branch$(P, \bar{z})$
        else
          remove$(P, \bar{z})$
        end if
      else
        remove$(P, \bar{z})$
      end if
    else
      remove$(P, \bar{z})$
    end if
  else
    remove$(P, \bar{z})$
  end if
until $P = \emptyset$
return UB

Algorithm 4 Implication rules: imply($\gamma, \bar{z}$)

if $\gamma.value = 0$ **then**
  if card $\{s \in S | \bar{z}_{\gamma.ks} \neq 0\} = LB_{\gamma.k}$ **then**
    for all $s \in S$ **do**
      if $\bar{z}_{\gamma.ks} = -1$ **then**
        $\bar{z}_{\gamma.ks} \leftarrow 1$
        $\gamma.decisions \leftarrow \gamma.decisions \cup \{\bar{z}_{\gamma.ks} = 1\}$
      end if
    end for
  end if
  for all $s \in S$ **do**
    if $\bar{z}_{\gamma.ks} = -1$ **then**
      $\bar{z}_{\gamma.ks} \leftarrow 0$
      if $[JobShop(\bar{z})]$ is unfeasible **then**
        $\bar{z}_{\gamma.ks} \leftarrow 1$
        $\gamma.decisions \leftarrow \gamma.decisions \cup \{\bar{z}_{\gamma.ks} = 1\}$
      else
        $\bar{z}_{\gamma.ks} \leftarrow -1$
      end if
    end if
  end for
end if

Algorithm 5 Evaluation: evaluate($\gamma, \bar{z}$)

result $\leftarrow$ -1
$\bar{x} \leftarrow$ optimal solution of the LP-relaxation of $[ETP(\bar{z})]$; cost: LP
if $\bar{x}$ exists **then**
  result $\leftarrow$ LP
end if
return result

Algorithm 6 Branching: branch($P, \bar{z}$)

$\bar{k} \leftarrow \text{argmax}_{k \in K | \exists s \in S, \bar{z}_{ks} = -1} (\text{card}\{(k,s) \in \{K \times S\} | \bar{z}_{ks} \in \{-1,1\}\} - LB_k)$
$\bar{s} \leftarrow \text{argmax}(s \in S | \bar{z}_{\bar{k}\bar{s}} = -1)$
$P \leftarrow P \cup \text{Node}\left(\text{k=}\bar{k}, \text{ s=}\bar{s}, \text{ value=1, decisions=}\{\bar{z}_{\bar{k}\bar{s}} = 1\}\right)$
$P \leftarrow P \cup \text{Node}\left(\text{k=}\bar{k}, \text{ s=}\bar{s}, \text{ value=0, decisions=}\{\bar{z}_{\bar{k}\bar{s}} = 0\}\right)$

Algorithm 7 Removing: remove($P, \bar{z}$)

$\forall d \in \gamma.decisions \quad \bar{z}_{d.kd.s} \leftarrow -1$
$P \leftarrow P \setminus \gamma$

Algorithm 8 Leaf: leaf($\bar{z}$)

for all $(k,s) \in \{K \times S\}$ **do**
  if $\bar{z}_{ks} = -1$ **then**
    return false
  end if
end for
return true

5 Computational experiments

In this section, we first describe the instances used as a test bed and the methods used as comparison (section 5.1). We then briefly evaluate the interest of the initial inequalities (section 5.2). Finally, we investigate the experimental behavior of the methods proposed in this paper (section 5.3).

5.1 Test bed and methods

To test the methods proposed above, we used the generator of instances of [AGRV09]. We have defined 6 categories of instances of increasing size. For each category, two sets of 25 "feasible" (respectively "unfeasible") instances have been generated. The parameters used as inputs by the instance generator are displayed in Table 1.

In Table 1, column μ refers to the total number of employees, of whom μ_{extra} are *extra employees*. Those extra employees ensure that the production requirements can always be mathematically met (as in [AGRV09]). However we make a distinction between the two types of instances. In the case of "feasible" instances, the production requirements can actually be met in pratice: virtually no extra employees are needed and, even though more expensive, they do not incur a prohibitive cost (they are typically outsourced personel or regular staff working overtime). In the case of "unfeasible" instances, it is not possible to meet production requirements without extra employees, and those have a prohibitive cost, making the instances unfeasible in practice (an extra employee has a cost of 1000, that dominates the sum of all other costs; hence, if a solution has a cost of e.g., 8028, that means that 8 extra employees are needed).

In Table 1, Θ^* is the best known solution for $[P]$. It is the optimum for 57.7% of all cases; otherwise it corresponds to the best solution found by either the hybrid branch and bound / cut generation method described in Section 4 or a MIP solver applied to the formalisation $[P]$ (within a CPU time limit of 2 hours for each method). For the 15 compatible instances picked out from [AGRV09], C_{max} is the optimal makespan of the job-shop problem. For the 285 other generated instances, C_{max} is computed as follows: 1) a heuristic makespan C^{heur}_{max} to the job-shop problem is found thanks to our job-shop solver (cf. Section 2.2), 2) the number δ of shifts is fixed to $\lceil C^{heur}_{max}/\pi \rceil$ and 3) C_{max} is fixed to $(\delta \cdot \pi)$.

We try to solve every instance with the two exact methods proposed above:

- `Cut`: the decomposition and cut generation approach described in Section 3 ;
- `HyBB`: the hybrid branch and bound / cut generation method described in Section 4.

As a basis for comparison, we also try to solve each instance with :

- `MIP`: a MIP solver applied to the formalisation $[P]$.
- `AGRV09`: a slightly modified version of the method proposed in [AGRV09]. Indeed, Artigues et al. solve a lexicographic optimization problem where the makespan minimization is the primary objective whereas the employee cost minimization is the secondary objective. We thus define `AGRV09` as the second stage, the minimal makespan value being directly fixed in a preprocessing stage.
- `Heur`: a heuristic solution. A heuristic solution to the job-shop problem is found thanks to our job-shop solver (cf. Section 2.2) and every pair (machine, shift) that is worked is fixed. The employees are then assigned optimaly using a MIP solver. (Note: a simpler heuristic, where employees are assigned greedily instead of optimaly, has also been tried

type	category	n	m	μ (incl. μ_{extra})	μ_{extra}	# max skills per employee	π	C_{max}	Θ^*
feasible	ejs6 $\times$ 4 $\times$ 25	6	4	25	10	2	8	$\in [37,56]$	$\in [17,33]$
	ejs6 $\times$ 6 $\times$ 25	6	6	25	10	4	10	$\in [48,70]$	$\in [24,73]$
	ejs8 $\times$ 8 $\times$ 40	8	8	40	20	4	8	$\in [44,64]$	$\in [70,106]$
	ejs8 $\times$ 8 $\times$ 50	8	8	50	20	4	10	$\in [60,80]$	$\in [40,64]$
	ejs10 $\times$ 10 $\times$ 40	10	10	40	20	4	8	$\in [72,80]$	$\in [174,286]$
	ejs10 $\times$ 10 $\times$ 50	10	10	50	20	4	10	$\in [80,100]$	$\in [81,133]$
unfeasible	ejs6 $\times$ 4 $\times$ 25	6	4	25	21	2	8	$\in [40,56]$	$\in [8009,12013]$
	ejs6 $\times$ 6 $\times$ 25	6	6	25	19	4	10	$\in [50,70]$	$\in [8028,18015]$
	ejs8 $\times$ 8 $\times$ 40	8	8	40	32	4	8	$\in [64,64]$	$\in [22040,32034]$
	ejs8 $\times$ 8 $\times$ 50	8	8	50	42	4	10	$\in [60,90]$	$\in [19040,28031]$
	ejs10 $\times$ 10 $\times$ 40	10	10	40	30	4	8	$\in [72,80]$	$\in [39056,50055]$
	ejs10 $\times$ 10 $\times$ 50	10	10	50	40	4	10	$\in [80,100]$	$\in [35056,47068]$

Table 1 Parameters of the 300 instances (25 instances per category)

out; it is barely faster than `Heur` and produces drastically worse solutions, and thus it has been discarded.)

CPU time has been limited to 5 minutes for the two categories of small instances ejs6 $\times$ 4 $\times$ 25 and ejs6 $\times$ 6 $\times$ 25. Is has been limited to 10 minutes for the four categories of big instances ejs8 $\times$ 8 $\times$ 40, ejs8 $\times$ 8 $\times$ 50, ejs10 $\times$ 10 $\times$ 40 and ejs10 $\times$ 10 $\times$ 50.

In the remaining (if it is not specified), the initial inequalities of Section 3.5 are added to the three methods `Cut`, `HysBB` and `MIP`. We did not experiment the interest of the initial cuts for `AGRV09`.

5.2 Initial inequalities

To evaluate the initial inequalities described in Section 3.5, we first compare in Table 2 the results of the LP-relaxation of $[P]$ with and without the initial inequalities. In this table, column LP/Θ^* gives the deviation between the optimum of the LP-relaxation of $[P]$ (LP) and the best known solution for $[P]$ (Θ^*).

category	linear relaxation (LP) of $[P]$		linear relaxation (LP) of $[P]$ with initial inequalities				
	LP/Θ^*	time	LP/Θ^*	# initial inequalities	preprocess time	LP time	total time
ejs6 $\times$ 4 $\times$ 25	83.3%	0.3s	98.1%	48.3	0.2s	0.2s	0.4s
ejs6 $\times$ 6 $\times$ 25	66.9%	1.0s	87.8%	69.3	1.1s	1.0s	2.1s
ejs8 $\times$ 8 $\times$ 40	65.8%	4.4s	87.7%	127.8	5.7s	3.9s	9.6s
ejs8 $\times$ 8 $\times$ 50	70.3%	8.7s	85.9%	123.4	10.3s	9.9s	20.2s
ejs10 $\times$ 10 $\times$ 40	62.4%	21.5s	82.3%	211.6	49.1s	22.3s	71.4s
ejs10 $\times$ 10 $\times$ 50	61.9%	44.4s	70.6%	194.5	74.7s	52.3s	127.0s
Total	68.4%	13.4s	85.4%	129.1	23.5s	14.9s	38.4s

Table 2 Impact of the initial inequalities on the LP-relaxation. Instances are not distinguished by type, since the results are almost identical in both cases.

For each type and each category of instances, the LP-relaxation is much better with the initial inequalities: the average ratio LP/Θ^* is indeed 17% higher when the initial inequal-

ities are used, and the extra computing time remains moderate (24*s* in average and about 128*s* for the worst case).

To further chalenge the initial inequalities, we compare the rate of success of the three exact methods (`Cut`, `HyBB`, `MIP`), depending on whether the initial inequalities are used (Table 3).

category	`Cut`		`HyBB`		`MIP`	
	without initial cuts	with initial cuts	without initial cuts	with initial cuts	without initial cuts	with initial cuts
ejs6 × 4 × 25	0.0%	98.0%	72.0%	100.0%	86.0%	100.0%
ejs6 × 6 × 25	0.0%	44.0%	28.0%	74.0%	46.0%	76.0%
ejs8 × 8 × 40	0.0%	20.0%	0.0%	52.0%	26.0%	60.0%
ejs8 × 8 × 50	0.0%	16.0%	0.0%	28.0%	12.0%	28.0%
ejs10 × 10 × 40	0.0%	12.0%	0.0%	18.0%	2.0%	16.0%
ejs10 × 10 × 50	0.0%	2.0%	0.0%	4.0%	0.0%	0.0%
Total	0.0%	32.0%	16.7%	46.0%	28.7%	46.7%

Table 3 Impact of the initial inequalities on the exact methods. Instances are not distinguished by type, since the results are almost identical in both cases.

It is very clear from these results that the initial inequalities proposed in Section 3.5 strongly improve all three methods. The improvements is particularly important for `Cut` and `HyBB`. Indeed, these two methods cannot solve instances larger than 6 jobs without initial inequalities. As a matter of fact, the initial cuts are an integrated part of the two exact methods `Cut` and `HyBB` proposed in this article and they are nicely complementary to the other components of those methods.

As stated before, the initial inequalities are added to each method described in the remaining (except for `AGRV09`).

5.3 Exact methods

In this section, we compare the resultats of the two exact methods we have proposed in this paper (`Cut` and `HyBB`) with other approaches (`MIP`, `AGRV09` and `Heur`).

Table 4 reports the performances of each method. Column *success* gives the percentage of instances solved to optimality within the time limit. Column *time* displays the average computation time. Column *gap* gives the average deviation to the best known solution Θ^* [2]. The statistical significance of those results is verified by an exhaustive computation of Kruskal-Wallis tests for every pair of methods and every measure; when relevant, p-values are provided in the following discussion.

Table 5 compares the successes of the different methods. This table can be read as a matrix a_{ij} where a_{ij} is the number of instances the method i solves to optimality within the time limit whereas the method j does not. a_{ii} gives the number of successes of the method i.

Furthermore, Table 6 provides a direct comparison between the two best performing methods: `HyBB` and `MIP`. In this table, *success* is the number of instances solved to optimality by both methods; *time(S)* is the CPU time in case of success (that is for the instances solved to optimality by both approaches); and *gap(F)* is the average deviation to the best known solution in case of failure (that is when at least one method fails).

[2] $gap = \min\left(1, \frac{|value-\Theta^*|}{\Theta^*}\right)$ if a solution *value* has been found, 1 otherwise

category	Cut			HyBB			MIP			AGRV09			Heur		
	success	time	gap	success	time	gap	success	time	gap	success	time	gap	success	time	gap
ejs6 $\times$ 4 $\times$ 25	100%	2s	0%	100%	1s	0%	100%	1s	0%	88%	53s	1%	8%	0s	17%
ejs6 $\times$ 6 $\times$ 25	60%	122s	5%	76%	86s	0%	84%	65s	0%	44%	182s	31%	0%	0s	49%
ejs8 $\times$ 8 $\times$ 40	36%	387s	7%	68%	234s	4%	68%	264s	2%	32%	478s	47%	0%	0s	35%
ejs8 $\times$ 8 $\times$ 50	16%	504s	10%	24%	461s	5%	28%	454s	2%	8%	568s	45%	0%	0s	24%
ejs10 $\times$ 10 $\times$ 40	8%	548s	17%	16%	529s	8%	20%	525s	43%	0%	601s	51%	0%	1s	28%
ejs10 $\times$ 10 $\times$ 50	4%	588s	28%	8%	571s	6%	0%	594s	92%	0%	601s	73%	0%	1s	28%
Total	37%	358s	11%	49%	314s	4%	50%	317s	23%	29%	414s	41%	1%	0s	30%

(a) “feasible” instances

category	Cut			HyBB			MIP			AGRV09			Heur		
	success	time	gap	success	time	gap	success	time	gap	success	time	gap	success	time	gap
ejs6 $\times$ 4 $\times$ 25	96%	27s	0%	100%	3s	0%	100%	4s	0%	0%	298s	26%	0%	0s	27%
ejs6 $\times$ 6 $\times$ 25	28%	239s	9%	72%	115s	2%	68%	131s	1%	0%	299s	42%	0%	0s	44%
ejs8 $\times$ 8 $\times$ 40	4%	576s	11%	36%	436s	4%	52%	367s	1%	0%	600s	31%	0%	0s	27%
ejs8 $\times$ 8 $\times$ 50	16%	510s	15%	32%	432s	9%	28%	463s	18%	0%	600s	34%	0%	0s	34%
ejs10 $\times$ 10 $\times$ 40	16%	514s	15%	20%	493s	2%	12%	543s	57%	0%	601s	25%	0%	1s	17%
ejs10 $\times$ 10 $\times$ 50	0%	600s	30%	0%	601s	2%	0%	600s	96%	0%	601s	33%	0%	1s	20%
Total	27%	411s	13%	43%	347s	3%	43%	351s	29%	0%	500s	32%	1%	0s	28%

(b) “unfeasible” instances

Table 4 Performance of each method (average values)

		dominated					dominated				
	$i \setminus j$	Cut	HyBB	MIP	AGRV09	Heur	Cut	HyBB	MIP	AGRV09	Heur
dominating	Cut	*56*	0	1	18	54	*40*	0	1	40	40
	HyBB	17	*73*	4	30	71	25	*65*	4	65	65
	MIP	20	6	*75*	33	73	26	4	*65*	65	65
	AGRV09	5	0	1	*43*	41	0	0	0	*0*	0
	Heur	0	0	0	0	*2*	0	0	0	0	*0*

(a) "feasible" instances (b) "unfeasible" instances

Table 5 Comparison of the number of instances solved by each method

category	success	HyBB		MIP	
		time(S)	gap(F)	time(S)	gap(F)
ejs6 $\times$ 4 $\times$ 25	25/25	1s	-	2s	-
ejs6 $\times$ 6 $\times$ 25	19/25	19s	2%	12s	2%
ejs8 $\times$ 8 $\times$ 40	15/25	39s	10%	89s	4%
ejs8 $\times$ 8 $\times$ 50	6/25	22s	6%	92s	3%
ejs10 $\times$ 10 $\times$ 40	4/25	195s	9%	478s	51%
ejs10 $\times$ 10 $\times$ 50	0/25	-	6%	-	92%
Total	69/150	27s	6%	47s	43%

(a) "feasible" instances

category	success	HyBB		MIP	
		time(S)	gap(F)	time(S)	gap(F)
ejs6 $\times$ 4 $\times$ 25	25/25	3s	-	4s	-
ejs6 $\times$ 6 $\times$ 25	17/25	40s	7%	52s	4%
ejs8 $\times$ 8 $\times$ 40	9/25	143s	6%	95s	1%
ejs8 $\times$ 8 $\times$ 50	7/25	61s	12%	111s	25%
ejs10 $\times$ 10 $\times$ 40	3/25	36s	2%	127s	64%
ejs10 $\times$ 10 $\times$ 50	0/25	-	2%	-	96%
Total	61/150	42s	5%	49s	48%

(b) "unfeasible" instances

Table 6 Comparison of HyBB and MIP in case of success or failure (average values)

From all those experiments, the first general conclusion is that HyBB is the best performing methods on any kind of instances. Then come MIP and Cut, then AGRV09 and the poorest method is Heur. Typically, HyBB solves to optimality more instances than the other methods (but MIP on small instances) and besides it solves them faster when it succeeds or provides tighter bounds when it fails.

From Table 4, it is clear that HyBB achieves better success and gap than Heur, and this is statistically significant (p-values $< 10^{-7}$). HyBB is also better than AGRV09 (all p-values $< 10^{-3}$). Compared to Cut, HyBB is more successful and provides tigher gap (p-values $<$ 0.05 for success, and $< 10^{-7}$ for gap); it also seems to be faster, but this is not statistically significant (p-value$>$ 0.6 on feasible instances). Furthermore, none of Cut, AGRV09 or Heur solves to optimality an instance that HyBB does not (see Table 5).

As a consequence, HyBB is only challenged by MIP for the title of best method. From Table 4, it is not clear whether HyBB has better success than MIP; indeed, the two methods cannot be distinguished on this criteria (p-value$>$ 0.8), nor on the computation time. The dominance of HyBB is however quite clear for the gap (p-values $<$ 0.05). Table 6 allows for a more detailed analysis. In case of success, HyBB is quicker (however, the statistical relevance is not so good). In case of failure, HyBB provides a significant better gap (p-values $<$ 0.003).

All in all, HyBB performs generaly better than MIP, with exceptions. It is not dominated for the success, nor for the computation times, and achieves better gaps. In particular, on hard instances that none of the methods solve, HyBB is clearly better: the gap is significantly lower, and does not explode when the size of the problem increases; moreover it must be noted that HyBB provides feasible solutions for all 300 instances, while MIP failed for 74 of them. Nevertheless, on some instances, MIP may outperform HyBB (e.g., for small instances).

Similar detailed analysis confirm 1) that HyBB also outperforms Cut, AGRV09 and Heur (all results are statistically significant); 2) that MIP outperforms Cut (except for big instances), AGRV09 and Heur; 3) that Cut outperforms AGRV09 and Heur.

Note that Heur, which is never very good, is very consistent in providing very quickly feasible solutions with a gap of about 30%. Such solutions are not good for small instances, but become competitive for the big ones. Actually, Heur outperforms every method but HyBB on the most challenging instances. This is essentially due to the fact that, within the given time, methods Cut, MIP and AGRV09 may fail to find a feasible solution for such instances.

One should notice here that the comparison with [AGRV09] is not totally fair. Indeed, [AGRV09] solves more general problems than the methods described in this article. We remind the reader that [AGRV09] tackles a problem where each job can require more than one employee to be processed. It also addresses a problem with two hierarchical objectives: 1) minimizing the makespan and 2) minimizing the employee cost. Furthermore, AGRV09 is not competitive at all on unfeasible instances because of the very high costs of the *extra* employees. The method (dichotomy) has not been fitted for such test problems.

To conclude with these experiments, we point out that really large instances of the problem tackled in this paper are intractable with the current methods. Indeed one can observe that none of the methods experimented here can solve instances with more than 10 jobs and 10 machines; starting from 8 jobs and 8 machines, all methods have serious difficulties. This is not a surprise since the job-shop sub-problem itself is awfully difficult to solve, even with only 10 jobs and 10 machines.

6 Conclusion

In this paper we have proposed two different procedures to solve an integrated employee-timetable and job-shob-scheduling problem, and we have evaluated them through computational experiments. In the process, we have outlined initial cuts that significantly help the finding of a solution for our own methods, but also for a standard MIP approach.

Our first method, the decomposition and cut generation procedure (Cut), is rather disappointing. However, its mixed results can easily be explained: the generated cuts do not contain enough global information and tend to eliminate only the current solution. Stronger cuts would be necessary; however the design of such cuts appears to be quite difficult and unpredictable perspectives (a tighter collaboration with the job-shop solver) would probably be the key. This phenomenon illustrates the fact that integrated problems require ad-hoc, well-tuned procedures that are capable of using the very particularities of the problem.

The second method, the hybridization of a cut generation process with a branch and bound strategy (HyBB) provides very good results and greatly improves over the performances of the best methods of the literature.

As for future research directions, the relevance of the hybridization of a cut generation process with a branch and bound strategy should be proved on other integrated problems, e.g., that incorporate different scheduling and/or timetabling constraints. Furthermore, variants of this hybridization should be tried out to provide a fast initial convergence that would allow to quickly get good feasible solutions.

Acknowledgements

The authors wish to thank Christian Artigues for kindly giving us access to his code and test problems. Thanks are also due to the anonymous referees who provided useful comments and suggestions that led to improvements in the paper.

A Computing environment

All experiments have been carried out on a standard PC (Intel(R) Core(TM) i3 CPU M 370 @ 2.40GHz, 2.39 GHz, 3.42 Go RAM) running MS Windows XP. Our own procedures are implemented in Java. The software of [AGRV09] is the original one, kindly provided by the authors, and is written in C++. Mixed integer and/or continuous linear programs have been solved with Ilog Cplex 12.2; for constraint based scheduling, we used Ilog Solver 6.7 and Scheduler 6.7.

References

AB97. Hesham K. Alfares and James E. Bailey. Integrated project task and manpower scheduling. *IIE Transactions*, 29:711–717, 1997.

Agg02. Riad Aggoune. *Ordonnancement d'ateliers sous contraintes de disponibilité des machines*. PhD thesis, Université de Metz, 2002.

Agg04a. Riad Aggoune. Minimizing the makespan for the flow shop scheduling problem with availability constraints. *European Journal Of Operational Research*, 153:534–543, 2004.

Agg04b. Riad Aggoune. Une procédure par séparation et évaluation pour l'ordonnancement d'un job-shop sous contraintes de disponibilité. In *Proceedings of 5e Conférence Francophone de MOdélisation et SIMulation (MOSIM'04)*, Nantes - France, 2004.

AGRV09. Christian Artigues, Michel Gendreau, Louis-Marie Rousseau, and Adrien Vergnaud. Solving an integrated employee timetabling and job-shop scheduling problem via hybrid branch-and-bound. *Computers & Operations Research*, 36:2330–2340, 2009.

Ben62. J.F. Benders. Partitioning procedures for solving mixed-variables programming problems. *Numerische Mathematik 4*, 4:238–252, 1962.

BFHS88. Jacek Blazewicz, Gerd Finke, Reinhard Haupt, and Gunter Schmidt. New trends in machine scheduling. *European Journal of Operational Research*, 37(3):303–317, 1988.

Car75. Jacques Carlier. *Ordonnancement à contraintes disjonctives. Thèse de 3ème cycle*. PhD thesis, Paris VI, 1975.

Car82. Jacques Carlier. The one-machine sequencing problem. *European Journal Of Operational Research*, 11:42–47, 1982.

Che04. Z.-L. Chen. Simultaneous job scheduling and resource allocation on parallel machines. *Annals of Operations Research*, 129:135–153, 2004.

CP89. Jacques Carlier and Éric Pinson. An algorithm for solving the job-shop problem. *Management Science*, 35:164–176, 1989.

CP90. Jacques Carlier and Éric Pinson. A practical use of jackson's preemptive schedule for solving the job-shop problem. *Annals of Operations Research*, 26:169–287, 1990.

CP94. Jacques Carlier and Éric Pinson. Adjustement of heads and tails for the job-shop problem. *European Journal of Operational Research*, 78(2):146–161, 1994.

CPPR04. Jacques Carlier, Laurent Péridy, Éric Pinson, and David Rivreau. *Handbook of scheduling: algorithms, models and performance analysis*, chapter 4- Elimination rules for job-shop scheduling problem: overview and extensions. CRC Press, 2004.

DM94. Richard L. Daniels and Joseph B. Mazzola. Flow shop scheduling with resource flexibility. *Operations Research*, 42:504–522, 1994.

EJKS04. Andreas T. Ernst, Houyuan Jiang, Mohan Krishnamoorthy, and David Sier. Staff scheduling and rostering : a review of applications, methods and models. *European Journal of Operational Research*, 153:3 – 27, 2004.

GH05. Anis Gharbi and Mohamed Haouari. Optimal parallel machines scheduling with availability constraints. *Discrete Applied Mathematics*, 148:63 – 87, 2005.

GJ79. Michael R. Garey and David S. Johnson. *Computers and intractability: a guide to the theory of NP-Completeness*. W.H. Freeman, 1979.

GLPR10. Olivier Guyon, Pierre Lemaire, Éric Pinson, and David Rivreau. Cut generation for an integrated employee timetabling and production scheduling problem. *European Journal of Operational Research*, 201(2):557–567, 2010.

HO03. John N. Hooker and G. Ottosson. Logic based benders decomposition. *Mathematical Programming, series A*, 96:33–60, 2003.

Hoo05. John N. Hooker. A hybrid method for planning and scheduling. *Constraints*, 10:385–401, 2005.

Hoo07. John N. Hooker. Planning and scheduling by logic-based benders decomposition. *Operations Research*, 55:588–602, 2007.

Lee96. Chung-Yee Lee. Machine scheduling with an availability constraint. *Journal of Global Optimization*, 9:395–416, 1996.

Leu04. Joseph Y-T. Leung. *Handbook of scheduling: algorithms, models and performance analysis*. CRC Press, Boca Raton, FL, USA, 2004.

LS08. Lu-Wen Liao and Gwo-Ji Sheen. Parallel machine scheduling with machine availability and eligibility constraints. *European Journal of Operational Research*, 184:458–467, 2008.

MBB05. Philippe Mauguière, Jean-Charles Billaut, and Jean-Louis Bouquard. New single machine and job-shop scheduling problems with availability constraints. *Journal of scheduling*, 8:211–231, 2005.

MCZ10. Ying Ma, Chengbin Chu, and Chunrong Zuo. A survey of scheduling with deterministic machine availability constraints. *Computers & Industrial Engineering*, 58(2):199–211, 2010.

MS96. Paul Martin and David B. Shmoys. A new approach to computing optimal schedules for the job-shop scheduling problem. In *Proceedings of the 5th International IPCO conference*, pages 389–403, 1996.

Pin04. Michael Pinedo. *Scheduling: theory, algorithms and systems*. Prentice Hall, second edition, 2004.

Riv99. David Rivreau. *Problèmes d'ordonnancement disjonctifs: règles d'élimination et bornes inférieures*. PhD thesis, Université de Technologie de Compiègne, 1999.

RK76. Alexander H. G. Rinnooy Kan. *Machine scheduling problems: classification, complexity and computation*. Kluwer Academic Publishers, 1976.

RS64. Bernard Roy and Bernard Sussman. Les problèmes d'ordonnancement avec contraintes disjonctives, 1964.

Sav94. Martin W. P. Savelsbergh. Preprocessing and probing techniques for mixed integer programming problems. *INFORMS Journal on Computing*, 6(4):445–454, 1994.

Sch00. Günter Schmidt. Scheduling with limited machine availability. *European Journal of Operational Research*, 121:1–15, 2000.

SPR05. François Soumis, Gilles Pesant, and Louis-Marie Rousseau. *Gestion de production et ressources humaines*, chapter 4, Gestion des horaires et affectation du personnel. Presses Internationales Polytechnique, 2005.

SS98. Éric Sanlaville and Günter Schmidt. Machine scheduling with availability constraints. *Acta Informatica*, 35:795–811, 1998.

WSC05. Guoqing Wang, Hongyi Sun, and Chengbin Chu. Preemptive scheduling with availability constraints to minimize total weighted completion times. *Annals of Operations Research*, 133:183–192, 2005.

Zri05. Nozha Zribi. *Ordonnancement des job-shops flexibles sous contraintes de disponibilité des machines*. PhD thesis, Université des sciences et technologies de Lille, 2005.