A time-indexed LP-based approach for min-sum job-shop problems

Giuseppe Lancia, Franca Rinaldi and Paolo Serafini
University of Udine, Dept. of Mathematics and Computer Science, Via delle Scienze 206, Udine, Italy,
{*giuseppe.lancia,franca.rinaldi,paolo.serafini*}*@uniud.it*

Abstract: In this paper we propose two time-indexed IP formulations for job-shop scheduling problems with a min-sum objective. The first model has variables associated to job scheduling patterns. The exponential number of variables calls for a column generation scheme which is carried out by a dynamic programming procedure. The second model is of network flow type with side constraints. This model can be strengthened by adding cutting inequalities of clique type. It turns out that the two models are equivalent, since the dual of the second formulation is equivalent to the compact dual of the first model. However, they require significantly different solution approaches and may behave differently in terms of computing time and memory usage. Good upper bounds are found by a heuristic procedure that randomly generates schedules from fractional solutions. These features allow for an effective pruning of the branch-and-bound tree and narrowing the gap between lower and upper bounds. However, the size of both models is critically affected by the time-indexed formulation which may heavily slow down the computation.

*Keywords:*job-shop, flow-shop, total weighted completion time, time-indexed formulation.

1 Introduction

The job-shop problem has been extensively studied in the last three decades especially in the formulation of makespan minimization. Although there are good heuristic procedures based on local search (for instance, the celebrated shifting bottleneck procedure [1] for the makespan minimization is a local search with a very large neighborhood), the performance of exact procedures has never been satisfying. The hardness of the job-shop problem, even with respect to other NP-hard problems, has been soon recognized [18] and it is widely believed that large-size instances cannot be solved exactly unless a huge amount of computing time is used [8].

The job-shop problem we consider has the objective of minimizing the total job cost, i.e., the sum of the penalties on the job completion times. We do not make any assumption on the type of function measuring the penalty, and earliness/tardiness penalties can be easily accommodated as well as release dates and deadlines. As remarked above, much of the research on job-shop problems has been devoted to the makespan minimization (see the surveys [5, 17, 7, 24] and the papers cited therein) and a relatively smaller number of papers has investigated the version of minimizing the total cost [10, 25, 13]. Furthermore only few papers have considered non regular objective functions, i.e., functions non monotonic with respect to completion times [11, 12, 3].

We introduce two time-indexed formulations, i.e., formulations in which the time is discretized into a finite number of time slots and constraints and variables are possibly associated with each time slot.

The first formulation, already presented in [19] in a preliminary form, is an ILP model with column generation, in which each column is a scheduling pattern for all operations of a job. This way the combinatorial structure implicit in the operation sequences for each job is already embedded in the matrix columns. The machine constraints are taken care of by typical ILP inequalities and there are also assignment constraints to impose the selection of exactly one pattern for each job. The fact that part of the combinatorial structure is embedded in the ILP matrix, a feature typical of ILP formulations with column generation, provides a reliable lower bound to the LP relaxation.

The second formulation, first presented in [20], is a network flow model in which flows are associated to schedules of single operations. The network structure embeds the operation sequence constraints for each job. The machine constraints are added to the network flow constraints.

The integrality relaxations of the two models have the same value. This fact can be proven directly, and it is also due to the fact that the second model can be derived from the first one by exploiting duality. Indeed, the dual of the first model can be compactly rewritten and the dual of this compact formulation is nothing but the network flow model.

The second model allows for adding cutting inequalities derived by an equivalent stable set problem with clique inequalities. A large number of clique inequalities are already embedded into the two models, and this explains why the gap between the lower bound and the optimal value is not large. The violation of a class of other clique inequalities can be easily identified. Clearly, adding violated inequalities makes the lower bound stronger, but slows down the computation and one should consider the trade-off between these two aspects.

Furthermore, from each fractional solution of both models we derive feasible schedules by a randomized procedure followed by a local search. This heuristic tool is essential in having quickly good solutions (most of the times it finds the optimal solution or a (very) good upper bound in the first levels of the branch-and-bound tree).

All these features have a positive effect in narrowing the gap between lower and upper bounds to the optimal solution and therefore in pruning the nodes of the branch-and-bound tree. However, both models suffer from the large size of the LP models due to the time-indexed formulation. If the time unit is small in comparison to the time horizon the size of the problems that can be solved is greatly reduced.

In this paper we do not claim to be able to solve exactly large-size instances. Indeed, we support the general belief that large-size job-shop instances defy exact solution in reasonable computing time [8]. However, we also think that small steps towards understanding the job-shop structure followed by small improvements in computing time have their value. Maybe in the future the problem will be successfully tackled by exploiting a large number of different tools. For instance, the recently developed techniques, known as matheuristics, call for complex recipes in which smaller subproblems are solved exactly by means of mathematical programming tools (see [21] for an updated and comprehensive survey). The job-shop could be tackled by these techniques within which exact models like the one proposed in this paper could prove very useful.

Among other promising approaches for the solution of job-shop scheduling problems we recall constraint programming [6, 4] and a number of heuristic procedures, such as local search metaheuristics [27, 23, 31], or hybrid procedures that combine local search and constraint programming approaches [2]. The heuristic procedures are essential in solving large-size instances, but they are also useful within exact techniques. Indeed, as already remarked, finding good incumbents as soon as possible is mandatory and this can be achieved by exploiting ideas from heuristics.

The models presented in this paper allow to solve exactly in a few seconds instances of size up to 5 jobs and 5 machines and in a few minutes instances with unit processing times of up to 10 jobs and 10 machines. Larger size instances require a much larger computing time to close the gap between upper and lower bounds. However, we may stop the computation at an early stage with a small gap in the hope that the upper bound is the optimum. We provide computational experiments which show the behavior of the approach. However, we point out that our code is just an academic prototype. It is possible that by profiling and optimizing our implementation, using more powerful computers and/or faster LP engines, the computing times can be considerably reduced.

We stress the fact that our models can deal with weighted earliness-tardiness penalties in the same way they deal with the sum of completion times. This is a positive feature typical of time-indexed formulations. We also note that both models can be immediately extended to deal with some variants of the job-shop, like the no-wait job-shop scheduling. In this case the problem size is even smaller.

The positive aspects of time-indexed formulations for scheduling problems have been pointed out in [29]. The first time-indexed formulation is due to [15] for a one machine problem. The idea of using column generation for scheduling problems has been already exploited (see [28, 29]). In [28] a single machine problem is studied and a time-indexed formulation is turned into a more compact model by exploiting the structure of single machine problems. In [29], a problem with identical machines and several jobs is investigated, and variables are associated to single machine scheduling patterns. In the above papers the computational burden of a time-indexed formulation can be successfully avoided. However, for more complex cases with many operations and machines, like the job-shop problem, there does not seem to be an easy way to escape from the many constraints of a time-indexed formulation.

The idea of using Lagrangian relaxation to split job precedences and machine constraints has been successfully used in [3] for the job-shop problem with earliness and tardiness costs. Actually, there are strong similarities between our model and one of the two models presented in [3], although the starting points of the two approaches and the solution techniques are quite different.

The paper is organized as follows. In Section 2 we define the type of job-shop problem we deal with. In Section 3 we present the integer LP model based on column generation. In Section 4 we introduce the compact network flow model. In Section 5 we investigate some cutting inequalities derived by a particular class of violated clique inequalities. In Section 6 we provide a small example to illustrate the cutting inequalities. In Section 7 we describe the heuristic procedure to generate schedules starting from fractional solutions of both models. In Section 8 we present some computational results. Finally, some conclusions follow in Section 9.

2 The job-shop problem

In the job-shop problem, a set M of m machines and a set J of n jobs are given. Every job $j \in J$ consists of a given sequence of $n(j)$ operations $O_1^j \to \cdots \to O_k^j \to \cdots \to O_{n(j)}^j$ and each operation O_k^j has to be processed without preemption on the machine $\mu(j,k)$ with a known processing time $q(j,k) > 0$. A *feasible schedule* of the jobs in J is a set of completion times $t(j,k)$ associated to each operation O_k^j such that: (i) the job precedence relations of the operations are respected and (ii) operations associated to the same machine do not overlap in time. It is not excluded that a machine can process more than one operation for the same job.

We assume that, for each job $j \in J$ and $k := 1, \ldots, n(j)$, a function

$$f_{jk} : R \to R \cup \{+\infty\}, \qquad t(j,k) \mapsto f_{jk}(t(j,k)) \tag{1}$$

is defined assigning a penalty to each operation completion time. The function $f_{jk}(t)$ takes on value $+\infty$ when its argument t is an infeasible completion time for operation O_k^j (clearly, $+\infty$ can be replaced by a suitably large number in an implementation, in fact any value larger than an upper bound to the optimum).

The cost of a feasible schedule is defined as the following separable objective function

$$\sum_{j \in J} \sum_{k=1}^{n(j)} f_{jk}(t(j,k)). \tag{2}$$

The functions f_{jk} can model release dates r_{jk} and deadlines d_{jk} for each operation by setting $f_{jk}(t) = +\infty$ for $t < r_{jk} + q(j,k)$ and $t > d_{jk}$, respectively. Fixed idle times of the machines can be dealt with by similar techniques.

We consider the problem of finding a feasible schedule of minimum cost. As usual, we assume that all data are integer, being integer multiples of a given time unit. As a consequence, we may restrict our attention to integral completion times $t(j,k)$.

3 A column generation model

In a discrete-time framework, we denote by $[a,b]$ the set of integers $\{z \in Z : a \leq z \leq b\}$ and call it *time interval.* A *scheduling pattern* (or simply a pattern) p for the job $j \in J$ is a sequence of $n(j)$ time intervals $[s(j,k), t(j,k)]$, $1 \leq k \leq n(j)$ with $t(j,k-1) \leq s(j,k)$ for each $2 \leq k \leq n(j)$ and $t(j,k) - s(j,k) = q(j,k)$ for each k. A scheduling pattern defines for each operation k of a job its starting time $s(k)$ and its ending time $t(k)$. The cost of a pattern is given by

$$c(p) = \sum_{k=1}^{n(j)} f_{jk}(t(j,k)). \tag{3}$$

Note that a pattern is a particular schedule of the job operations. If p is a pattern for job j, we denote by $t(p)$ the completion time of the last operation of the pattern, i.e. $t(p) := t(n(j))$.

Let $\overline{T}$ be a value for the time horizon. We assume that $\overline{T}$ is sufficiently large to contain an optimal schedule. We comment on this assumption in the next section. Let us denote by P^j the set of patterns for job j with $t(p) \leq \overline{T}$.

We associate to each pattern p in P^j an $m\overline{T}$-dimensional $\{0,1\}$–vector a^p with m fields of length $\overline{T}$, one for each machine $h \in M$. The t-th entry of the h-th field $a^p_{h,t}$ is 1 if and only if the operation of the job which must be executed by machine h is processed in the time slot $[t-1,t]$. In other words, for each k and for each t, the component $a^p_{\mu(j,k),t}$ is 1 if and only if $s(j,k)+1 \leq t \leq t(j,k)$.

Now we associate a binary variable x_p to each pattern p of P^j, with the meaning that $x_p = 1$ if and only if the job j is scheduled according to the pattern p. Then the job-shop problem with total cost objective may be formulated as the following binary linear programming problem:

BP:

$$\min \quad \sum_{j \in J} \sum_{p \in P^j} c(p)\, x_p$$

$$\sum_{p \in P^j} x_p = 1 \qquad j \in J \tag{4}$$

$$\sum_{j \in J} \sum_{p \in P^j} a^p_{h,t}\, x_p \leq 1 \qquad h \in M, \quad t = 1, \ldots \overline{T} \tag{5}$$

$$x_p \in \{0,1\} \quad p \in P^j, \quad j \in J. \tag{6}$$

Constraints (4) state that each job has to be scheduled according to exactly one pattern and are thus called *assignment constraints.* Conditions (5) guarantee that each machine processes no more than one job at a time and are called *machine constraints.* Finally, we have the binary conditions (6) on the variables. The integrality relaxation of **BP**, where constrains (6) are replaced by $x_p \geq 0$ for each $p \in P^j$, $j \in J$, will be denoted as **RP** (note that the condition $x_p \leq 1$ is not needed, as it is implied by (4)). **BP** contains a constraint matrix having $|J| + m\overline{T}$ rows and an exponential number of columns, one for each feasible pattern of each job. In order to solve **RP**, one has to resort to a column generation approach.

3.1 The pricing procedure

Let us denote by $u(j)$ the dual variable corresponding to the j-th assignment constraint and by $v(h,t) \le 0$ the dual variable corresponding to machine h and the time slot $[t-1,t]$. Moreover for job j, operation $k \in \{1,\dots,n(j)\}$ and pattern $p \in P^j$, let us denote by $s(j,k,p)$ the starting time of operation k within pattern p. Then the dual of **RP** is the following problem:

DP:

$$\begin{aligned} \max \quad & \sum_{j\in J} u(j) + \sum_{h\in M}\sum_{t=1}^{\overline{T}} v(h,t) \\ & u(j) - \sum_{k=1}^{n(j)} \sum_{t=s(j,k,p)+1}^{s(j,k,p)+q(k,j)} (-v(\mu(j,k),t)) \le c(p) \qquad p \in P^j, \quad j \in J \\ & v(h,t) \le 0. \end{aligned} \tag{7}$$

Let $\hat{u}$ and $\hat{v}$ denote the optimal dual variables of a reduced **RP**, i.e., a subproblem of **RP** containing all the constraints and only the columns generated so far. Then checking dual feasibility (and hence primal and dual optimality) is carried out by solving the following n independent problems, one for each job j, and checking non-negativity of the result:

$$-\hat{u}(j) + \min_{p\in P^j} c(p) + \sum_{k=1}^{n(j)} \sum_{t=s(j,k,p)+1}^{s(j,k,p)+q(j,k)} (-\hat{v}(\mu(j,k),t)). \tag{8}$$

The pricing problem (8) can be solved for each job j by a forward dynamic programming procedure. We first note that, apart from the constant $\hat{u}(j)$, the reduced cost of a pattern $p \in P^j$ is given by its original cost $c(p)$ augmented by the sum of the dual variables associated to the time slots when the operations are processed according to p. So we may interpret each value $-\hat{v}(h,t)$ as an additional cost on the use of machine h in the time interval $[t-1,t]$. Let $\tilde{v}(k,t) := \sum_{\tau=t-q(k)+1}^{t}(-\hat{v}(\mu(k),\tau))$ (here and in the sequel we occasionally simplify the notation by dropping the dependence on j).

In order to solve (8), we define labels

$$V(k,t), \quad 0 \le k \le n(j), \quad \underline{q}(k) \le t \le \overline{T} - \overline{q}(k)$$

where

$$\underline{q}(k) := \sum_{h=1}^{k} q(h), \quad \overline{q}(k) := \sum_{h=k+1}^{n(j)} q(h).$$

Each $V(k,t)$ represents the minimum reduced cost of a pattern consisting of the first k operations and completing the k-th operation *within* t.

We conventionally set $V(0,t) := 0$ for each t and initialize the other values as $V(k,t) := +\infty$. Then the labels $V(k,t)$ can be recursively computed, for $k := 1,\dots,n(j)$ and $\underline{q}(k) \le t \le \overline{T} - \overline{q}(k)$, as

$$V(k,t) = \min\{V(k,t-1), V(k-1,t-q(k)) + \tilde{v}(k,t) + f_{jk}(t)\} \tag{9}$$

where the two terms in the above expression represent the minimum reduced cost of patterns which complete the k-th operation before t and exactly at time t, respectively (it is understood that for some values of k and t one of the two terms may be missing in (9)). As a consequence, for any time t, the minimum reduced cost of a pattern $p \in P^j$ with $t(p) \le t$ is given by $V(n(j),t) - \hat{u}(j)$.

From the computational point of view, it is clearly convenient to work with values of $\overline{T}$ as small as possible. On the other side, if the optimal solutions have completion times larger than $\overline{T}$, they cannot be produced if $\overline{T}$ is chosen too small. In order to be sure that optimal solutions are not lost with the chosen time horizon $\overline{T}$, the following result can be useful. Given the current optimal solution $\overline{x}$ of **RP**, let $T(\overline{x}) := \min\{\tau : \hat{v}(h,t) = 0 \text{ for each } h \in M, t > \tau\}$. Note that, by the complementarity conditions, $T(\overline{x}) \leq \max\{t(p) : x_p > 0\}$.

Proposition 1: *Let us assume that the functions $f_{jk}(t)$ are non decreasing for $t \geq T(\overline{x})$. If, for each $j \in J$, $T(\overline{x}) + \sum_{k=1}^{n(j)} q(k) \leq \overline{T}$, then $\overline{x}$ is an optimal solution of* **RP** *for any time horizon larger than $T(\overline{x})$.*

Proof: Let us extend the time horizon and consider a pattern $p \in P^j$ with $t(p) > \overline{T}$. By extending the time horizon the dual variables associated to times $t > \overline{T}$ are zero. Let $\tau(1), \ldots, \tau(n(j))$ be the completion times of the different operations of the pattern p. Denote by $0 \leq \overline{k} < n(j)$ the maximum index for which $\tau(\overline{k}) - q(\overline{k}) + 1 \leq T(\overline{x})$. Since $v(k,t) = 0$ for each $t > T(\overline{x})$ (so that $\tilde{v}(k, \tau(k)) = 0$ for $k > \overline{k}$) it follows that

$$\overline{c}(p) = -\hat{u}(j) + \sum_{k=1}^{\overline{k}} \tilde{v}(k, \tau(k)) + \sum_{k=1}^{\overline{k}} f_k(\tau(k)) + \sum_{k=\overline{k}+1}^{n(j)} f_k(\tau(k)).$$

In particular, because of the optimality of labels $V(k,t)$, we have

$$\overline{c}(p) \geq -\hat{u}(j) + V(\overline{k}, \tau(\overline{k})) + \sum_{k=\overline{k}+1}^{n(j)} f_k(\tau(k)).$$

Finally, the hypothesis on the functions f_k implies that

$$\overline{c}(p) \geq -\hat{u}(j) + V(\overline{k}, \tau(\overline{k})) + \sum_{k=\overline{k}+1}^{n(j)} f_k\Big(\tau(\overline{k}) + \sum_{i=\overline{k}+1}^{k} q(i)\Big).$$

Now the right hand side of the above expression corresponds to the reduced cost of a pattern $\hat{p}$ which completes the $\overline{k}$-th operation in $\tau(\overline{k})$ at a cost $V(\overline{k}, \tau(\overline{k}))$ and the following operations afterwards without idle times. Since $\hat{p}$ completes the last operation of job j within time $T(\overline{x}) + \sum_{k=\overline{k}}^{n(j)} q(k) \leq \overline{T}$, $\overline{c}(\hat{p}) \geq 0$ by hypothesis. Therefore $\overline{c}(p) \geq \overline{c}(\hat{p}) \geq 0$. ∎

The previous proposition has two different implications. On one side, we may reduce the time horizon to $\max_j T(\overline{x}) + \sum_{k=\overline{k}}^{n(j)} q(k)$ if the current time horizon $\overline{T}$ is larger than needed. This way the computing time is not heavily affected by the initial choice of $\overline{T}$. On the other side, it tells how to reset $\overline{T}$ if it has been underestimated.

3.2 The overall branch and price

Problem **BP** can be solved by a branch and price procedure based on its LP relaxation. We remark that, because of the column generation approach, the classical rule of branching on fractional components of the optimal solution of the relaxation cannot be used in this case. Indeed, while a branching constraint that fixes a variable x_p to 1 can be easily handled in the solution of the subproblems, a constraint $x_p = 0$ cannot be enforced. In particular, it could happen that a pattern fixed to 0 in a subproblem turns out to be the optimal solution of the pricing problem returned by the dynamic procedure. In order to overcome this difficulty, we have chosen to adopt the following branching rule.

At each node i of the tree search a time interval $[\sigma(j,k,i), \tau(j,k,i)]$ is defined for each job j and each operation k of job j. At the root node $\sigma(j,k,i) = 0$, $\tau(j,k,i) = \overline{T}$. The set of the feasible patterns is restricted to those elements in P^j which complete every operation $k = 1, \ldots, o(j)$ in $[\sigma(j,k,i), \tau(j,k,i)]$.

Since this restriction simply corresponds to assign release and due dates to the different operations, it may be modeled by updating the values of the penalty functions $f_{k,j}$ according with

$$f_{k,j}(t) = +\infty \;\; \forall t \notin [\sigma(j,k,i), \tau(j,k,i)].$$

This way the pricing procedure generates patterns of minimum reduced cost which are feasible in the node.

Let now $\overline{x}$ be an optimal fractional solution of the linear relaxation solved in the node i and define $R^j = \{p \in P^j : 0 < \overline{x}_p < 1\}$. Since $\overline{x}$ is fractional, the set $J(\overline{x}) = \{j \in J : |R^j| \geq 2\}$ is not empty. Take any job $j \in J(\overline{x})$. There exists at least one operation $k(j)$ whose completion time is different in two patterns of R^j. Let $t(p, k(j))$ be the completion time of the operation $k(j)$ for pattern p. Then $\hat{t} = \sum_{p \in R^j} t(p, k(j))\, x_p$ is the average completion time of the operation $k(j)$. We branch at node i by creating two subproblems i' and i''. We restrict the set of the feasible patterns of job j to those completing the operation $k(j)$ not later than $\hat{t}$ in one subproblem, and in the other subproblem strictly later than $\hat{t}$. These restrictions are accomplished by defining $\tau[\hat{j}, \hat{k}, i'] = \hat{t}$ and $\sigma[\hat{j}, \hat{k}, i''] = \hat{t} + 1$.

Before solving the relaxations corresponding to the new subproblems one has to eliminate from the formulation the patterns which are not any more feasible with respect to the last branching constraint.

4 A network flow model

We have found that a large number of columns need to be generated before reaching optimality in **RP** and this affects the computing times. In order to overcome this difficulty, we have reformulated **RP** in a compact way with size polynomial in the number of jobs and machines, and pseudopolynomial in the processing times. The new model is based on network flow properties.

We define a network $G = (N, E)$ as follows. The node set N of G is given by

$$N = \left\{(j,k,t) : j \in J,\, 0 \leq k \leq n(j),\, \underline{q}(j,k) \leq t \leq \overline{T} - \overline{q}(j,k)\right\},$$

where, as before, $\underline{q}(j,k) := \sum_{h=1}^{k} q(j,h)$ and $\overline{q}(j,k) := \sum_{h=k+1}^{n(j)} q(j,h)$. The nodes of G can be partitioned into levels $L(j,k)$, $j \in J$ and $0 \leq k \leq n(j)$ where each level $L(j,k)$, $k > 0$, contains a node (j,k,t) for each possible completion time t of the k-th operation of j and each level $L(j,0)$ contains a node $(j,0,t)$ for each possible starting time of the first operation.

In the network there are n source-sink pairs. The sources are the nodes $s_j := (j,0,0,)$ and the sinks are the nodes $d_j := (j, n(j), T)$.

The graph contains arcs of two types. The arcs of

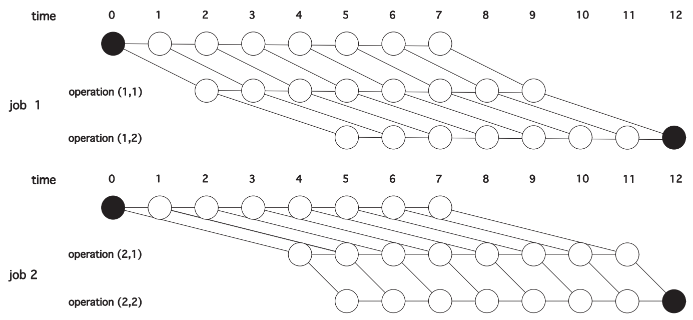
$$E_0 := \left\{((j,k,t-1),(j,k,t)) : j \in J,\, 0 \leq k \leq n(j),\, \underline{q}(j,k) < t \leq \overline{T} - \overline{q}(j,k)\right\}$$

are called *idle arcs* and connect nodes of the same level corresponding to consecutive time instants, while the arcs of

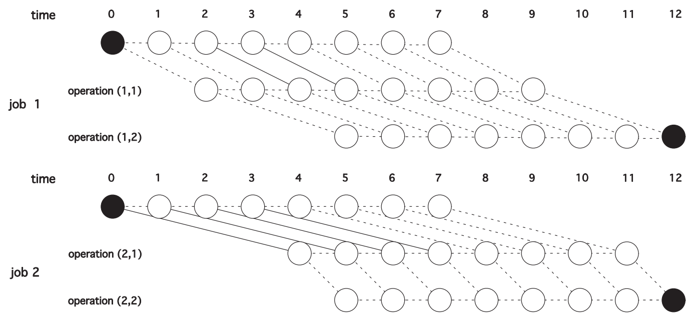
$$E_1 := \left\{((j,k-1,t-q(j,k)),(j,k,t)) : j \in J,\, 0 < k \leq n(j),\, \underline{q}(j,k) \leq t \leq \overline{T} - \overline{q}(j,k)\right\}$$

are called *active arcs* and connect nodes in consecutive levels $L(j,k-1)$ and $L(j,k)$ with time labels differing for the processing time of the operation (j,k). Let $E = E_0 \cup E_1$. The network G has n connected components, one for each job. Since each node (j,k,t) is the end node of at most one idle arc and at most one active arc, we denote the arcs in E_0 by $e^0(j,k,t)$ and the arcs in E_1 by $e^1(j,k,t)$, i.e. by simply using the same labels of the end node of the arcs. We associate to each arc a cost defined by

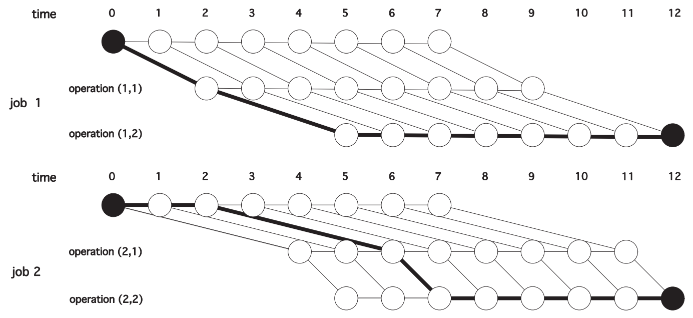
$$c(e^0(j,k,t)) = 0, \qquad c(e^1(j,k,t)) = f_{jk}(t).$$



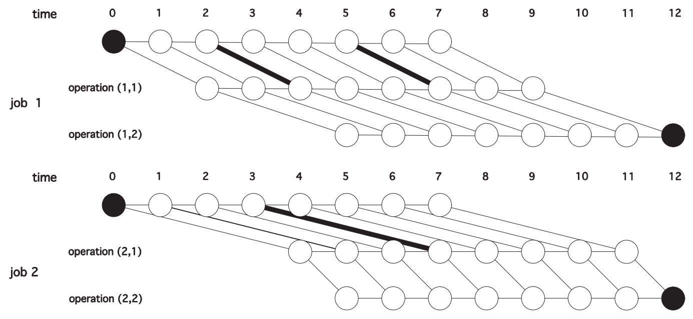
(a) **CRP** with two jobs, two operations per job and time horizon $\overline{T} = 12$



(b) arcs entering the bundle constraints for machine 1 and time 4



(c) optimal solution



(d) arcs corresponding to a clique in $\mathcal{G}^1$ for nodes $(1, 1, 4)$, $(1, 1, 7)$, $(2, 1, 7)$

Figure 1: The network flow model for two jobs and four operations

Let $\xi^0(j,k,t)$ and $\xi^1(j,k,t)$ be flow variables associated to the arcs $e^0(j,k,t)$ and $e^1(j,k,t)$ respectively. The idea is to send one flow unit from each source s_j to each sink d_j. If the flow is integral it gives rise to a path $s_j \to d_j$ which can be interpreted as a scheduling of the job j. Each arc $e^1(j,k,t)$ traversed by the flow corresponds to scheduling the operation k of job j with completion time t, i.e. $t(j,k) = \{t : \xi(e^1(j,k,t)) = 1\}$. In order to have a feasible flow for the machine constraints we have to introduce specific constraints, that we call *bundle constraints*. The resulting integer linear program is the following problem

CBP

$$\begin{array}{lll}
\min & \displaystyle\sum_{(j,k,t)} f_{jk}(t)\,\xi^1(j,k,t) & \\
& \xi^0(j,k,t)+\xi^1(j,k,t) = \xi^0(j,k,t+1)+\xi^1(j,k+1,t+q(j,k+1)) & \forall (j,k,t) \neq (j,0,0) = s_j \\
& & \qquad\neq (j,n(j),T) = d_j \\
& \xi^0(j,n(j),T)+\xi^1(j,n(j),T) = 1 & \forall j \\
& \displaystyle\sum_{(j,h,\tau)} \left\{\xi^1(j,h,\tau) \,:\, \mu(j,h) = h, t \le \tau \le t+q(j,h)-1 \right\} \le 1 & \forall (k,t) \\
& \xi^0(j,k,t), \xi^1(j,k,t) \in \{0,1\} & \forall (j,k,t)
\end{array} \tag{10}$$

where it is tacitly understood that some variables are missing if any label in the triple (j,k,t) is out of range. The first set of constraints takes care of flow conservation in all nodes except the sources and the sinks. The second set is the requirement of one flow unit arriving to each sink (and consequently starting from each source). The third set of constraints are the bundle constraints which impose at most one flow unit for each time t and for all operations requiring the same machine.

Let **CRP** be the integral relaxation of **CBP**. There is a close relationship between **CRP** and **RP**, as stated in the following proposition.

Proposition 2: **CRP** *and* **RP** *have the same optimal value and* **CBP** *and* **BP** *have the same optimal value.*

Proof: By the decomposition flow theorem, any solution ξ of **CRP** can be decomposed into a finite set of flows along source-sink paths. Since any path from s_j to d_j corresponds to a pattern $p \in P^j$ with the same cost, we may easily derive from ξ a solution x of **RP** with the same cost. This solution satisfies the assignment constraints (4) and the machine constraints (5) because ξ satisfies the flow constraints and the bundle constraints, respectively. The converse can be proven in a straightforward way. The second part of the statement follows easily. ∎

Actually the relationship between **CRP** and **RP** goes beyond the stated result. The problem (7), dual of problem **RP**, can be alternatively rewritten as a problem with a polynomial number of constraints by dropping the exponentially many constraints and substituting them with a set of linear constraints (after possibly introducing a polynomial number of new variables), which impose the optimality condition of the pricing dynamic programming procedure. This kind of reformulation is called compact [9, 22]. The dual of the compact reformulation problem is **CRP**.

As stated both **CRP** and **RP** yield the same lower bound and thus they exhibit the same strength in exploring the branch-and-bound tree. However, they behave differently. Whereas **RP** requires an ad hoc branch-and-price procedure, **CRP** can be solved by any integer linear programming solver. Moreover, taking care of branching constraints in **CRP** is simpler than in **RP** (see Section 3.2).

As an example of **CRP** consider an instance with two jobs and four operations. The first job consists of the operations (1,1) and (1,2) with processing times $q(1,1) = 2$, $q(1,2) = 3$ and the second job consists of the operations (2,1) and (2,2) with processing times $q(2,1) = 4$, $q(2,2) = 1$. Let us suppose that the operations (1,1) and (2,1) require one machine, while the operations (1,2) and (2,2) require a second machine. By fixing a time horizon $\overline{T} = 12$ we build the network in Figure 1(a), where all arcs are directed left-right and top-down. The black nodes are the sources and the sinks. In Figure 1(b) the non-dashed arcs are those present in the machine constraints associated to the first machine and time 4. In Figure 1(c) the optimal solution is shown for the objective function $f_{jk}(t) = t$, for all j,k.

5 Cutting inequalities

If the feasible flow for the network G is fractional we may try to find cutting inequalities. The cutting inequalities we are going to define are based on violated clique inequalities for an independent set problem in another graph $\mathcal{G}$.

The graph $\mathcal{G} = (\mathcal{N}, \mathcal{E})$ is defined as follows. Its nodes correspond to the arcs of E_1 in G. We may label the nodes of $\mathcal{G}$ as (j, k, t), exactly like the nodes of G for $k \geq 1$, since there is a one-to-one correspondence between these nodes and arcs in E_1. Therefore, each node $(j, k, t) \in \mathcal{N}$ corresponds to scheduling the k-th operation of job j with completion time t. We want to represent feasible schedules as subsets of $\mathcal{N}$. To this aim we define the edges $\mathcal{E}$ so that an independent set in $\mathcal{N}$ corresponds to a feasible schedule.

It is convenient to partition the nodes in $\mathcal{N}$ in two alternative ways: either by defining the node sets $\mathcal{N}_{jk} := \{(j', k', t) : j' = j,\ k' = k\}$, consisting of the nodes referring to the k-th operation of job j or by defining the node sets $\mathcal{N}^h = \{(j, k, t) : \mu(j, k) = h\}$, consisting of the nodes referring to the machine h.

The edges of $\mathcal{E}$ are built as follows. For each operation (j, k), all nodes in $\mathcal{N}_{jk}$ are linked into a clique (*operation clique*) whose edges are called *operation edges*. We denote these edge sets as $\mathcal{E}_{jk}$. These edges express the constraint that an operation cannot be scheduled twice (or more) at different times.

For any job j, we draw an edge between (j, k, t) and (j, k', t'), if $k < k'$ and $t' - \sum_{k<h\leq k'} q(j, h) < t$. These edges, denoted as $\mathcal{E}_j$, are called *precedence edges* and express the precedence condition among operations of the same job.

Let us fix an arbitrary real $0 < \varepsilon < 1$ and associate the real interval $[\,t - q(k) + \varepsilon\,,\, t\,]$ (*operation interval*) to each node (j, k, t). For each machine h, we draw an edge between (j, k, t) and (j', k', t') if $\mu(j, k) = \mu(j', k') = h$ and the intervals $[\,t - q(k) + \varepsilon\,,\, t\,]$ and $[\,t' - q(k') + \varepsilon\,,\, t'\,]$ have non empty intersection. We denote these edge sets as $\mathcal{E}^h$ (*machine edges*). These edges express the constraint that each machine can process at most one operation at a time. Note that some edges referring to the same operation can be both machine and operation edges.

Finally we define

$$\mathcal{E} := \bigcup_{\substack{1\leq j\leq n \\ 1\leq k\leq n(j)}} \mathcal{E}_{jk} \cup \bigcup_{1\leq j\leq n} \mathcal{E}_j \cup \bigcup_{1\leq h\leq m} \mathcal{E}^h.$$

Let us associate to each node $(j, k, t) \in \mathcal{N}$ the flow $\xi^1(j, k, t)$. Clearly, any 0-1 feasible flow for the network G corresponds to an independent set in $\mathcal{G}$ with $\sum_{j=1}^{n} n(j)$ nodes and viceversa. Therefore it must satisfy the clique inequality $\sum_{(j,k,t)\in\mathcal{K}} \xi^1(j, k, t) \leq 1$ for each clique $\mathcal{K}$ of $\mathcal{G}$.

However, fractional feasible flows for the network G could violate some clique inequalities. Our aim is to identify such inequalities. Let us first note that a fractional solution $\xi^1(j, k, t)$ cannot violate any clique inequality induced by a clique $\mathcal{K}$ in $\cup_{k=1,\ldots,n(j)}\mathcal{N}_{jk}$, i.e., whose nodes correspond to operations of the same job. Indeed, any flow for job j can be decomposed into a finite set of paths $s_j \to d_j$ in G. Now it is easy to see that different nodes of $\mathcal{K}$ correspond to arcs in G that, by construction, cannot belong to a same path. Hence the sum of the corresponding $\xi^1(j, k, t)$ values cannot exceed 1 by the flow constraints in (10). So these clique inequalities are already embedded in the constraints of **CRP**.

Let us now consider cliques in the graph $\mathcal{G}^h := (\mathcal{N}^h, \mathcal{E}^h)$ for machine h. By construction, $\mathcal{G}^h$ is an interval graph and thus all maximal cliques in $\mathcal{G}^h$ are of the form

$$\mathcal{N}^h(\tau) := \{(j,k,t) \ : \ \mu(j,k) = h \ , \ t - q(j,k) + \varepsilon \le \tau \le t\}$$

for some (real) τ. Therefore the bundle constraints for **CRP** are clique inequalities for the interval graph $\mathcal{G}^h$ and all maximal cliques in $\mathcal{G}^h$ are present in the bundle constraints. Some bundle constraints are redundant because some cliques are not maximal (in Figure 1(b) the inequalities for machine 1 and times $t \in \{1, 9, 10, 11\}$ are not maximal).

The fact that the bundle constraints are clique inequalities and the operation cliques and precedence cliques are implicitly taken care of by the flow constraints makes **CRP** (and **RP** as well) an effective model with a good lower bound. However, a fractional solution may violate a clique inequality of $\mathcal{G}$ for cliques containing edges both in $\mathcal{E}^h$ and in $\mathcal{E}_{jk} \cup \mathcal{E}_j$. As a simple example of a clique of this type, consider two nodes (j,k,t), (j,k,t'), with $t' - t > q(j,k)$, together with a third node (j'', k'', t''), with $t'' - q(j'', k'') < t$, and $t'' > t' - q(j,k)$. These three nodes form a clique which is neither an operation clique nor a machine clique. See Figure 1(d).

In general, there exist many maximal cliques with edges both in $\mathcal{E}^h$ and in $\mathcal{E}_{jk} \cup \mathcal{E}_j$. In order to describe them we define, for each h, the subgraph $\overline{\mathcal{G}}^h \subset \mathcal{G}$ induced by the nodes in $\mathcal{N}^h$. Under the common assumption that different operations of a same job are performed by different machines, nodes of $\mathcal{N}^h$ cannot be linked by precedence edges. Therefore $\overline{\mathcal{G}}^h := (\mathcal{N}^h, \bigcup_{\mu(j,k)=h} \mathcal{E}_{jk} \cup \mathcal{E}^h)$.

The rest of the section is devoted to characterize the maximal cliques of $\overline{\mathcal{G}}^h$ that can be violated by a fractional flow in G. Let us first give an informal description of the main result. Differently from $\mathcal{G}^h$, $\overline{\mathcal{G}}^h$ is not an interval graph. Having added edges, it is possible that a subset $\mathcal{K}$ of nodes that is not a clique in $\mathcal{G}^h$ becomes a clique in $\overline{\mathcal{G}}^h$. The structure of this clique is quite interesting. The extra edges which turn this set into a clique come all from the operations edges of one specific operation and not from edges of different operations. This result makes the identification of cliques of this type easy.

The main result needs the following lemma.

Lemma 3: *Given four intervals $a = [a_1, a_2]$, $b = [b_1, b_2]$, $c = [c_1, c_2]$, $d = [d_1, d_2]$, such that a and b are disjoint and c and d are disjoint, the corresponding interval graph cannot have all four edges (a,c), (a,d), (b,c), (b,d).*

Proof: Assume $a_2 < b_1$ and $c_2 < d_1$. Assuming that the four edges exist implies $b_1 \le c_2$ and $d_1 \le a_2$, so that

$$a_2 < b_1 \le c_2 < d_1 \le a_2.$$

▮

Proposition 4: *Let $\mathcal{K}$ be a clique in $\overline{\mathcal{G}}^h$ but not in $\mathcal{G}^h$. Let $\mathcal{K}^1, \ldots, \mathcal{K}^p$ be a partition of $\mathcal{K}$ according to the operations. Then exactly one $\mathcal{K}^i$ is not a clique in $\mathcal{G}^h$.*

Proof: Since nodes in different sets $\mathcal{K}^i$ are all joined in $\overline{\mathcal{G}}^h$ by edges of $\mathcal{E}_h$ and $\mathcal{K}$ is not a clique in $\mathcal{G}^h$, at least one $\mathcal{K}^i$ is not a clique in $\mathcal{G}^h$. If there are two sets $\mathcal{K}^i$ and $\mathcal{K}^j$ which are not cliques in $\mathcal{G}^h$, then there exist two nodes in $\mathcal{K}^i$ not joined by an edge, i.e. their intervals are disjoint. Similarly for $\mathcal{K}^j$. However all edges between nodes of $\mathcal{K}^i$ and $\mathcal{K}^j$, which are machine edges, must be present and this leads to a contradiction with Lemma 3. So it is impossible that two (or more) sets $\mathcal{K}^i$ are not cliques in $\mathcal{G}^h$. ▮

As a consequence of Proposition 4 we have that any clique $\mathcal{K}$ of $\overline{\mathcal{G}}^h$ which is not a clique in $\mathcal{G}^h$ is a union of a clique of $\mathcal{G}^h$ and an operation clique. This property suggests a simple procedure to list all maximal cliques in $\overline{\mathcal{G}}^h$. For a better understanding of the procedure the reader can also refer to the example in the next section.

For each machine h and for each operation $(\overline{j}, \overline{k})$ such that $\mu(\overline{j}, \overline{k}) = h$ we look for cliques in $\overline{\mathcal{G}}^h$ that may be obtained by adding an operation clique in $\mathcal{N}_{\overline{jk}}$ to a clique in $\mathcal{N}^h$ not containing nodes of $\mathcal{N}_{\overline{jk}}$. To this aim we need to define a set of cliques based on intervals. For each interval $[a, b]$ let $\mathcal{Q}(\overline{j}, \overline{k}, a, b)$ be the clique containing all nodes in $\mathcal{N}^h \setminus \mathcal{N}_{\overline{jk}}$ associated to operation intervals containing $[a, b]$, i.e.

$$\begin{aligned}\mathcal{Q}(\overline{j}, \overline{k}, a, b) = &\ \big\{(j,k,t) \in \mathcal{N}^h : (j,k) \neq (\overline{j}, \overline{k}),\ t - q(j,k) + \varepsilon \leq a \leq b \leq t\big\} = \\ &\ \big\{(j,k,t) \in \mathcal{N}^h : (j,k) \neq (\overline{j}, \overline{k}),\ b \leq t \leq a + q(j,k) - \varepsilon\big\}\ .\end{aligned}$$

Now, for each pair of nodes $(\overline{j}, \overline{k}, t')$, $(\overline{j}, \overline{k}, t'')$ such that $t' \leq t'' - q(\overline{j}, \overline{k})$ (so that they are not connected by a machine edge) we consider the interval $[t' - 1 + \varepsilon, t'' - q(\overline{j}, \overline{k}) + 1]$. Then the set of nodes

$$\mathcal{Q}(t' - 1 + \varepsilon, t'' - q(\overline{j}, \overline{k}) + 1) \cup \big\{(\overline{j}, \overline{k}, t')\big\} \cup \big\{(\overline{j}, \overline{k}, t'')\big\}$$

is a clique, but not of machine type nor of operation type. We expand this clique into a maximal one by adding nodes $(\overline{j}, \overline{k}, t)$ such that $t' < t < t''$. Explicitly, given t' and t'', this maximal clique can be expressed as

$$\mathcal{K}(\overline{j}, \overline{k}, t', t'') = \bigcup_{\substack{(j,k) \neq (\overline{j}, \overline{k}) \\ \mu(j,k) = h}} \big\{(j,k,t) : t'' - q(\overline{j}, \overline{k}) + 1 \leq t \leq t' - 1 + q(j,k)\big\} \cup \big\{(\overline{j}, \overline{k}, t) : t' \leq t \leq t''\big\}\,.$$

We are going to prove that all the maximal cliques in $\overline{\mathcal{G}}^h$ that are not machine cliques nor operation cliques can be listed among the cliques $\mathcal{K}(\overline{j}, \overline{k}, t', t'')$.

Proposition 5: *Any maximal clique in $\overline{\mathcal{G}}^h$ that is not a clique in $\mathcal{G}^h$ is of the form $\mathcal{K}(\overline{j}, \overline{k}, t', t'')$ for some operation $(\overline{j}, \overline{k})$ and times t', t''.*

Proof: Let $\mathcal{K}$ be a maximal clique in $\overline{\mathcal{G}}^h$. According to Proposition 4 its nodes can be split into an operation clique $\mathcal{K}_1$, for a particular operation $(\overline{j}, \overline{k})$ whose edges are not all machine edges, and a machine clique $\mathcal{K}_2$ containing nodes corresponding to the other operations of the same machine. Let t' and t'' be the minimum and maximum time indices, respectively, appearing in nodes of $\mathcal{K}_1$. Then $\mathcal{K}_1 \subset \big\{(\overline{j}, \overline{k}, t) : t' \leq t \leq t''\big\}$. The intersection of all intervals corresponding to nodes of $\mathcal{K}_2$ can be expressed as $[a + \varepsilon, b]$ with a and b integers and $\mathcal{K}_2 \subset \mathcal{Q}(\overline{j}, \overline{k}, a + \varepsilon, b)$. Since $\mathcal{K}$ is a clique and all the nodes of $\mathcal{K}_1$ correspond to intervals of the same length, in order to have intersection between each interval corresponding to a node of $\mathcal{K}_1$ and $[a + \varepsilon, b]$, it must be

$$t \geq a + \varepsilon \qquad \text{and} \qquad t - q(\overline{j}, \overline{k}) + \varepsilon \leq b \qquad \text{for each} \quad t' \leq t \leq t''$$

that is

$$a + \varepsilon \leq t' \qquad \text{and} \qquad b \geq t'' - q(\overline{j}, \overline{k}) + \epsilon.$$

Sinc $a + \varepsilon \leq t'$ implies $a + \varepsilon \leq t' - 1 + \varepsilon$ and $b \geq t'' - q(\overline{j}, \overline{k}) + \epsilon$ implies $b \geq t'' - q(\overline{j}, \overline{k}) + 1$ it follows

$$\mathcal{Q}(\overline{j}, \overline{k}, a + \varepsilon, b) \subset \mathcal{Q}(\overline{j}, \overline{k}, t' - 1 + \varepsilon, t'' - q(\overline{j}, \overline{k}) + 1).$$

Therefore

$$\mathcal{K} = \mathcal{K}_1 \cup \mathcal{K}_2 \subset \big\{(\overline{j}, \overline{k}, t) : t' \leq t \leq t''\big\} \cup \mathcal{Q}(\overline{j}, \overline{k}, t' - 1 + \varepsilon, t'' - q(\overline{j}, \overline{k}) + 1) = \mathcal{K}(\overline{j}, \overline{k}, t', t'')$$

and, by the maximality of the clique $\mathcal{K}$, $\mathcal{K} = \mathcal{K}(\overline{j}, \overline{k}, t', t'')$. ∎

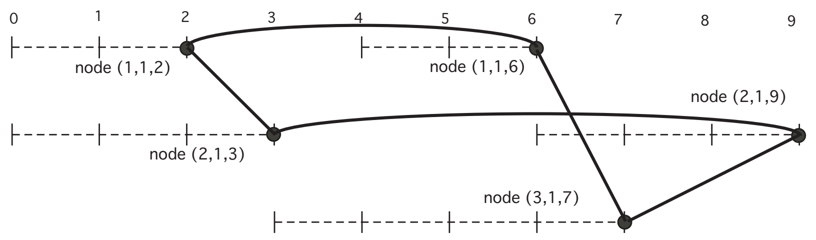


Figure 2: An odd hole generated by three operations

Since we are interested in listing only those cliques whose corresponding inequality is violated by the current solution, it makes sense to consider only cliques generated by pairs (t', t'') with $t' + q(\overline{j}, \overline{k}) \leq t''$ such that $\xi^1(\overline{j}, \overline{k}, t') > 0$ and $\xi^1(\overline{j}, \overline{k}, t'') > 0$. Indeed, as it is easy to verify, these cliques include all the cliques corresponding to inequalities with maximum violation. This speeds-up the computation and makes the procedure quite effective in finding cutting inequalities.

In $\mathcal{G}$ there are also odd holes with five or more nodes (a clique of three nodes is also a particular odd hole). See Figure 2 for an example of an odd hole generated by three operations of processing times 2, 3 and 4 respectively. We recall that for an odd hole $\mathcal{H}$, an odd hole inequality is defined by

$$\sum_{(j,k,t)\in\mathcal{H}} \xi^1(j,k,t) \leq \frac{|\mathcal{H}|-1}{2}$$

and is facet defining for the independent set polyhedron of $\mathcal{G}$. A fractional solution can violate some odd hole inequalities and it can be useful to detect and add them in order to improve the lower bound.

Odd holes can be generated by the procedure described in [16]. In detail, given a fractional solution $\overline{\xi}^1$ of **CRP**, we consider the subgraph $\mathcal{G}(\overline{\xi}^1)$ induced by the nodes of $\mathcal{G}$ corresponding to strictly positive components of $\overline{\xi}^1$ and associate to each edge $((j,k,t),(j',k',t'))$ of $\mathcal{G}(\overline{\xi}^1)$ the cost $1-\overline{\xi}^1(j,k,t)-\overline{\xi}^1(j',k',t')$. Then the odd hole inequality induced by $\mathcal{H}$ is satisfied if and only if $\mathcal{H}$ has cost not smaller than 1. So the separation problem with respect to the odd hole inequalities may be solved by finding the odd hole of minimum cost in the subgraph $\mathcal{G}(\overline{\xi}^1)$ and this task may be accomplished in polynomial time.

All these inequalities can be used to strengthen the lower bound also for **RP**. Since any fractional solution for **RP** can be immediately translated into a fractional solution or **CRP**, the valid inequality for the flow variables in **CRP** becomes a valid inequality for the pattern variables in **RP**. However, a non trivial complication arises for problem **RP**. One should take care of the new constraints in generating new columns. It is possible to deal with this problem by modifying the pricing dynamic procedure. We do not deal with this issue in this paper.

6 An example of cutting inequalities

As a small example let us consider an instance with 4 jobs, 4 operations per job and 4 machines whose data are:

job\op.	1	2	3	4
1	2	5	3	2
2	4	1	4	3
3	3	6	2	1
4	5	2	2	3

processing times $q(j,k)$

job\op.	1	2	3	4
1	1	2	3	4
2	4	2	1	3
3	1	2	3	4
4	3	4	1	2

machines $\mu(j,k)$

Solving **CRP** with time horizon $T = 24$ we get a fractional solution of value 60.75. Before starting the branch-and-bound procedure we try to strengthen the lower bound by adding clique inequalities. We illustrate the procedure described in the previous section by referring to the machine 2. We show in Figure 3 the subgraph $\mathcal{G}^h$ for the machine 2. Each node is labeled as (j,k,t) with (j,k) written at the left of each node row and t written on the top of each column. The black nodes are those with $\xi^1(j,k,t) > 0$ (the variable value is written near the node) and only edges of $\mathcal{E}^h$ connecting black nodes are shown. The number in brackets at the right are the operation processing times. The graph $\overline{\mathcal{G}}^h$ is obtained by adding all edges between nodes on the same row.

Suppose we want to identify violated cliques in $\overline{\mathcal{G}}^2$. Then we have to consider in turn the operations $(1,2)$, $(2,2)$, $(3,2)$ and $(4,4)$ and build the interval graphs $\mathcal{G}^2(\bar{j},\bar{k})$. Let us illustrate the case $(\bar{j},\bar{k}) = (4,4)$. As stated in the previous section, we consider only pairs of nodes $(4,4,t')$ and $(4,4,t'')$ such that the associated operation intervals are disjoint, i.e., $t'+3 \le t''$, and $\xi^1(4,4,t') > 0$, $\xi^1(4,4,t'') > 0$.

There are three pairs to be processed, namely $(t',t'') = (12,16)$, $(t',t'') = (12,17)$ and $(t',t'') = (14,17)$. For the pair $(t',t'') = (12,16)$, by applying the formula $[t'-1+\varepsilon, t''-q(\bar{j},\bar{k})+1]$, we derive the interval $[11+\varepsilon, 14]$ that must be contained in the operation interval of each node of the clique associated to an operation different from $(4,4)$. From this interval we define the clique

$$\mathcal{Q}(4,4,11+\varepsilon,14) = \{(1,2,14),(1,2,15),(1,2,16),(3,2,14),(3,2,15),(3,2,16),(3,2,17)\}\,.$$

All these nodes are associated to null values of ξ^1 and so we may discard this clique because it cannot produce a violated inequality. For the pair $(t',t'') = (12,17)$ we derive the interval $[11+\varepsilon,15]$. Since $\mathcal{Q}(4,4,11+\varepsilon,15) \subset \mathcal{Q}(4,4,11+\varepsilon,14)$ we may discard this clique as well. For the pair $(t',t'') = (14,17)$ we derive the interval $[13+\varepsilon,15]$, from which we define the clique

$$\mathcal{Q}(4,4,13+\varepsilon,15) = \{(1,2,15),(1,2,16),(1,2,17),(1,2,18),(3,2,15),(3,2,16),(3,2,17),(3,2,18),(3,2,19)\}$$

so that

$$\begin{aligned}\mathcal{K}(4,4,14,17) = \{&(1,2,15),(1,2,16),(1,2,17),(1,2,18),\\ &(3,2,15),(3,2,16),(3,2,17),(3,2,18),(3,2,19),(4,4,14),(4,4,15),(4,4,16),(4,4,17)\}.\end{aligned}$$

This is a violated inequality since $\sum_{(j,k,t)\in\mathcal{K}(4,4,14,17)} \xi^1(j,k,t) = 1.25 > 1$. We repeat the procedure for the other operations. For $(\bar{j},\bar{k}) = (1,2)$ there is no new clique, because the nodes $(1,2,7)$ and $(1,2,11)$ are already linked by a machine edge. For $(\bar{j},\bar{k}) = (2,2)$ we identify the cliques producing violated inequalities:

$$\begin{aligned}\mathcal{K}(2,2,5,6) = \{&(1,2,6),(1,2,7),(1,2,8),(1,2,9),(2,2,5),(2,2,6),\\ &(3,2,6),(3,2,7),(3,2,8),(3,2,9),(3,2,10),(4,4,6),(4,4,7)\}\\ \mathcal{K}(2,2,8,9) = \{&(1,2,9),(1,2,10),(1,2,11),(1,2,12),(2,2,8),(2,2,9),\\ &(3,2,9),(3,2,10),(3,2,11),(3,2,12),(3,2,13),(4,4,9),(4,4,10)\}\end{aligned}$$

and for $(\bar{j},\bar{k}) = (3,2)$ we identify the clique producing a violated inequality:

$$\begin{aligned}\mathcal{K}(3,2,12,19) = \{&(1,2,14),(1,2,15),(1,2,16),(1,2,17),\\ &(3,2,13),(3,2,14),(3,2,15),(3,2,16),(3,2,17),(3,2,18),(3,2,19),(4,4,14),(4,4,15)\}.\end{aligned}$$

We identify three more violated inequalities by processing the other machine graphs. After introducing these seven inequalities, we obtain an integral solution of value 61. We show in Figure 4 the gantt diagram of this schedule (the numbers in the boxes refer to the machines).

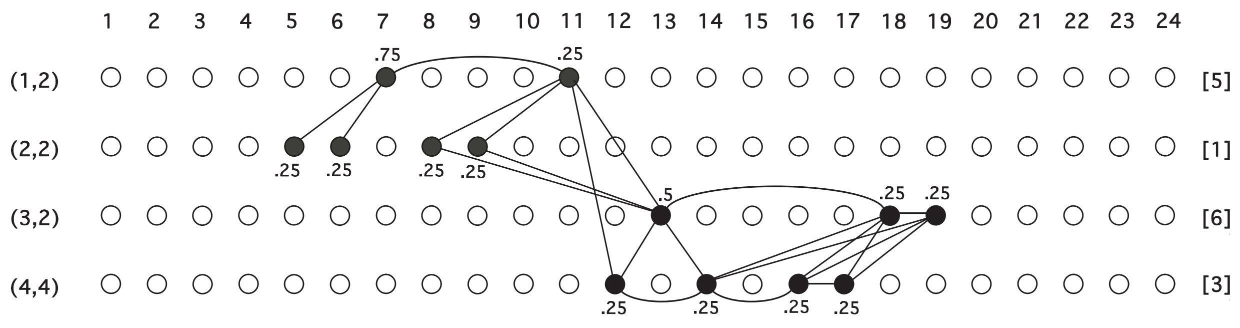


Figure 3: Machine edges in $\mathcal{G}^2$

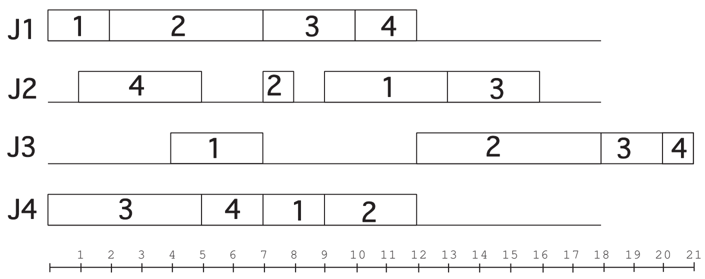


Figure 4: Optimal schedule

7 Computing a schedule from a fractional solution

Since the structure of a fractional solution embeds a lot of information on a possible integer solution, optimal fractional solutions can often be used to generate integer solutions by rounding procedures (see for instance [26]). In our case we have designed a randomized rounding procedure for both **RP** and **CRP**, which has turned to be crucial in having a viable branch-and-bound procedure. Indeed it takes too much time to obtain good integer solutions on some deep leaves of the branch-and-bound tree and therefore it is mandatory to have good incumbents available as soon as possible.

The rounding procedure works as follows. Given a solution of **RP**, for each job j we randomly select a pattern with probability proportional to the value of the corresponding variable. Given a solution of **CRP**, we recursively compute a path from s_j to d_j, for each job j, as follows: once a node (j, k, t) has been reached, the next node of the path is selected with probability proportional to the flow values of the two arcs exiting from (j, k, t). This is the same as decomposing the flow in paths and choosing randomly a path, i.e. a pattern, with probability proportional to its flow value.

After computing a scheduling pattern $[s(j, k), t(j, k)]$, $1 \leq k \leq n(j)$ for all jobs j, we may determine a permutation of the jobs on each machine m by sorting the intervals $[s(j, k), t(j, k)]$, $\mu(j, k) = m$, with respect to the $s(j, k)$ values. Let $[s(k), t(k)]$ and $[s(h), t(h)]$ be two consecutive intervals. If $t(h) > t(k)$ the machine precedence constraint between k and h is fixed as $k \to h$. If $t(h) \leq t(k)$ the two operations k and h are switched with probability $(t(k) - t(h))/(t(k) - t(h) + s(h) - s(k))$. The probabilities are fixed to 1/2 if $t(k) - t(h) = s(h) - s(k) = 0$.

Once the precedence graph is defined, if the functions $f(j, k)$ are regular (i.e. non decreasing functions of the completion times), the cost of the corresponding schedule can be computed as a function of the longest paths on the graph. When the functions $f(j, k)$ are not regular, as in the case of earliness costs, the cost of the schedule can be determined by an LP procedure.

Since the procedure is randomized, we can generate many solutions and apply to each one a local search heuristic based on the following moves. Let π^h be the ordering of the operations processed by machine h. For each π^h we define a neighborhood $N(\pi^h)$ as the set of permutations which may be obtained from π^h by interchanging the values $\pi^h(i)$ and $\pi^h(j)$ for some i, j, i.e., by interchanging the positions of a pair of operations (not necessarily successive) on one machine.

In our experience we have observed that this composite procedure provides good feasible solutions most of the times.

8 Computational results

We have solved **BP** by an ad hoc branch-and-price algorithm in C, using CPLEX 6.5 as the LP solver, on a Sun Workstation. The algorithm uses both column and row generation. After each column generation, it launches the local search procedure to obtain feasible solutions starting from the current fractional solution. Next, it checks the machine constraints by complete enumeration and adds all the violated machine inequalities to the current formulation.

We have solved **CBP** in two alternative ways. In the first way, which we may call *full mode*, we have used `glpk` as the LP solver together with a branch-and-cut framework developed by us in C on a 2 GB RAM 1.4 GHz Intel Core Duo CPU. The interaction with the framework involves launching the heuristic to get upper bounds and finding cutting clique inequalities to improve the lower bounds. More precisely, the heuristic described in Section 7 is launched after each LP computation. Starting from the current fractional solution, the procedure is called three times to generate feasible schedules on which the local search is applied. Adding clique inequalities is done as follows. After solving the first LP, we detect violated clique inequalities for each machine. All these violated inequalities are added to **CRP** at the same time and the problem is re-optimized. Since clique inequalities cut effectively mainly at the root node, we run three phases of clique inequality generation at this node and only one in the other nodes of the branch-and-bound tree.

In the second way, which we may call *batch mode*, **CBP** was solved as a stand-alone ILP problem by using CPLEX 11.210 on a 4 GB RAM 2.4GHz Intel Core Duo CPU. The mentioned framework is not able at the moment to interface with more recent versions of CPLEX and so we have to trade-off the possibility of launching heuristics and using effective cuts but with a slow solver (like `glpk` or CPLEX 6.5), with a faster solver (like CPLEX 11.210) but without the extra benefits of ad-hoc heuristics and cutting inequalities.

As a first test, we ran four sets of flow-shop instances with 5 and 10 jobs and 3 and 5 machines with objective function the sum of job completion times. Each set contains 10 instances with processing times randomly generated in the range [3, 10] according to [30]. The instances with 5 jobs have been solved via **CBP**, full mode, whereas the instances with 10 jobs have been solved both via **CBP**, full mode, and **BP**. We fixed a time limit of 900 seconds for both.

We report the computational results on the ten instances of each set in Tables 1–4. The column LB1 and LB2 report the lower bounds obtained at the root node before and after introducing the clique inequalities, respectively. The LB2 value is not reported in case the problem is already solved before introducing the clique inequalities. For instances with 5 jobs, the column UB provides the optimal value of the instances. For instances with 10 jobs, the columns UB_0, UB_1 and UB_2 provide the best upper bound found (by local search) during the solution of the root node for **BP** and the best upper bound found at the time limit solving **BP** and **CBP**, respectively. Usually, these upper bounds are found by the heuristic procedure which is much quicker in generating good integer solutions than the branch-and-bound procedure itself. Values with the star indicate that the instance has been solved to proved optimality. The column GAP refers to the ratio $(\text{UB} - \text{LB1})/\text{UB}$, expressed as a percentage, where UB is the best upper bound found. The next two columns report the number of the branch-and-bound tree nodes explored and CPU time in seconds for **CBP**.

We note that all instances with 5 jobs are solved to optimality in few seconds by both algorithms. Also, in the case of 10 jobs, in 15 minutes the exact procedures always produce feasible solutions of value at most

3% and 5% larger than the LP lower bound for the instances with 3 and 5 machines, respectively, and solve to optimality four out of ten instances with 3 machines.

As a second test, we applied **CBP**, batch mode, to the job-shop instances with earliness/tardiness costs, 2 machines, 10, 15 and 20 jobs and 5 machines, 10 jobs, proposed in [3]. These instances are divided into four classes depending on the due-date generation. This accounts for the terms 'tight' vs 'loose' and 'equal' vs 'tard' in the instance names. For details how the instances have been generated see [3]. The results of the test are reported in Table 5, where LB is the **CRP** value at the root node of the branch-and-bound tree (clearly without adding clique inequalities), UB is the best incumbent value (as before the star indicates that this value is proved optimal) found within a time limit of 1800 seconds, CPU is the cpu time in seconds, LB [3] and UB [3] are the lower bounds and upper bounds reported in [3] for the same instances, v^* is the optimal value of the instance and GAP refers to the ratio (UB $-$ LB)/UB, expressed as a percentage, where UB is the best known upper bound. We remark that the values v^* have been computed by applying **CBP**, batch mode, and allowing very large computing times. We report these values mainly for the sake of reference and gap evaluation.

Note that a guaranteed optimal solution was found for all instances with 2 machines and 10 jobs, for all instances with 2 machines and 15 jobs but one, for three instances with 2 machines and 20 jobs and for two instances with 5 machines and 10 jobs. For the other instances the time limit of 1800 sec was reached and either we have available only an upper bound or CPLEX has been unable to find at least one feasible solution. The latter circumstance has happened in two instances with 5 machines and 10 jobs. These two instances plus an additional third instance for which we do not have the optimum, seem particularly difficult as indicated by the much larger gaps. On the other hand, as can be seen by comparing with the optimal values v^* in the last column, in five out of the twelve cases where the time limit has been reached without proving optimality, the upper bound is indeed the optimum.

We point out an inconsistency in the first row of Table 5. We have found an LB of 460 and a proved optimal value of 462 for this instance. However, [3] reports an upper bound of 453! Clearly there must be some data error, which is very difficult to be traced back at this point.

This is the only test where it has been possible to compare directly our model with another model proposed in the literature by using the same instances. The method proposed in [3] is one of the most effective methods. They solve also other larger instances, that we have not considered for problems of model size which, as remarked, is an intrinsic limit of time-indexed formulations. However, we consider a positive result for our model that in the instances in which we have made the comparison we can obtain almost always a better or an equal solution. We may also note that our model yields a stronger lower bound for the majority of instances.

As a third test, we have solved two sets of randomly generated job-shop instances with 10 jobs, 5 and 10 machines, respectively, and unit processing times with objective function the sum of the job completion times. We have solved problem **CBP**, batch mode. The results are reported in Table 6 and Table 7. Note that for such instances the clique inequalities are all induced by either operations cliques or by machine cliques and thus are already embedded in **CBP**, making the solution batch mode very effective. All the instances were solved to optimality in a short time and the relative gap (UB $-$ LB)/UB is at most 8% over all instances.

9 Conclusions

In this paper we have presented a time-indexed model for flow-shop and job-shop problems with a separable objective function. For each job, any type of objective function of the completion time can be modeled. The time-indexed model has two possible formulations. The first one is based on column generation and each pricing procedure is carried out by a dynamic programming algorithm. The second one is a compact reformulation of the first model and turns out to be a network flow model with additional cross-arcs constraints. This second formulation allows for adding cutting inequalities and for a simple branching scheme. The advantage of these formulations consists, as already observed for time-indexed models [29], in having good lower bounds for the branch-and-bound method. On the other hand the size of the model increases considerably especially if the time unit is small with respect to the scheduling horizon. This is a drawback which at the moment prevents solving exactly instances of medium-large size, which, however, are out of reach with respect to the current available methods [8].

The computational tests we have carried out are based on codes that have not been optimized. These are complex codes whose optimization can be envisaged only after promising computational tests. In our opinion the results are satisfactory and worth carrying out future work toward improving various implementation phases.

There are some issues we have not covered in this paper. For instance the model allows quite naturally to include some variants of the job-shop problem, like the no-wait flow-shop, in which the job operations must be executed consecutively, i.e. without idle times between them. In general constraints within each jobs can be embedded in the pricing procedure and most of the times can be handled by the dynamic programming algorithm. However, constraints for the machine operations, like presence of set-up times, cannot be dealt with because the model can handle only, separately, constraints for each instant of time.

Moreover, considering that instances with many machines cannot be solved exactly, we may nonetheless approach these instances in a matheuristic fashion ([21]) by solving exactly small subproblems with one or two machines. These subproblems can be solved exactly with the techniques developed in this paper.

References

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

[2] J.C. Back, T.K. Feng and J. Watson, “Combining Constraint Programming and Local Search for Job-Shop Scheduling”, to appear on *INFORMS Journal on Computing.*

[3] P. Baptiste, M. Flamini and F. Sourd, “Lagrangian bounds for just-in-time job-shop scheduling”, *Computers & Operations Research*, **35**, pp. 906–915, 2008.

[4] P. Baptiste, C. Le Pape and W. Nuijten, “Constraint-Based Scheduling”, Springer, Berlin, 2001.

[5] J. Błażewicz, W. Domschke and E. Pesch, “The job shop scheduling problem: Conventional and new solution techniques”, *European Journal of Operational Research*, **93**, pp. 1–33, 1996.

[6] S.C. Brailsford, C.N. Potts and B.M. Smith, “Constraint satisfaction problems: Algorithms and applications”, *European Journal of Operational Research*, **119**, pp. 557–581, 1999.

[7] P. Brucker, *Scheduling Algorithms*, Springer, Berlin, 5^{th} edition, 2007.

[8] P. Brucker, “The Job-Shop Problem: Old and New Challenges”, in *Proceedings of the MISTA Conference 2007*, pp. 15-22, P. Baptiste, G. Kendall, A. Munier-Kordon and F. Sourd eds, 2007.

[9] R. D. Carr and G. Lancia, “Compact vs Exponential-Size LP Relaxations”, *Operations Research Letters*, **30(1)**, pp. 57–65, 2002.

[10] S. Chakrabarti, C. Phillips, A. Schulz, D. Shmoys, C. Stein and J. Wein, “Improved scheduling algorithms for minsum criteria”, *Proceedings of the 23rd International Colloquium on Automata, Languages and Programming*, pp. 646–657, Springer, 1996.

[11] H. Chen, C. Chu and J.M. Proth, “An improvement of the Lagrangean relaxation approach for job shop scheduling: a dynamic programming method”, *IEEE Transactions on Robotics and Automation*, **14**, pp. 786–795, 1998.

[12] A. Chen and P. Luh, “An alternative framework to Lagrangian relaxation approach for job shop scheduling”, *European Journal of Operational Research*, **149**, pp. 499–512, 2003.

[13] F. Della Croce, M. Ghirardi and R. Tadei, “An improved branch-and-bound algorithm for the two machine total completion time flow shop problem”, *European Journal of Operational Research*, **139**, pp. 293–301, 2002.

[14] U. Dorndorf, E. Pesch and T. Phan-Huy, “Constraint propagation and problem decomposition: A preprocessing procedure for the job shop problem”, *Annals of Operations Research*, **115**, pp. 125–145, 2002.

[15] M.E. Dyer and L. A. Wolsey, "Formulating the single machine sequencing problem with release dates as a mixed integer program", *Discrete Applied Mathematics*, **26**, pp. 255–270, 1990.

[16] M. Grötschel, L. Lovász and A. Schrijver, *Geometric Algorithms and Combinatorial Optimization*, Springer-Verlag, Berlin, 1993.

[17] A.S. Jain and S. Meeran, "Deterministic job-shop scheduling: Past, present and future", *European Journal of Operational Research*, **113**, pp. 390–434, 1999.

[18] B.J. Lageweg, J.K. Lenstra and A.H.G. Rinnooy Kan, "Job-shop scheduling by implicit enumeration", *Management Science*, vol. **34**, pp. 441–450, 1977.

[19] G. Lancia, F. Rinaldi and P. Serafini, "A column generation approach to solve job-shop problems", *IFORS99 Conference*, Beijing, P.R. China, August 16-20, Final Program, p. 11, 1999.

[20] G. Lancia, F. Rinaldi and P. Serafini, "A compact optimization approach to solve job-shop problems", *Proceedings of the MISTA Conference 2007*, pp. 293–300, P. Baptiste, G. Kendall, A. Munier-Kordon and F. Sourd eds , 2007.

[21] V. Maniezzo, Th. Stützle and S. Voß eds, *Hybridizing Metaheuristics and Mathematical Programming*, Springer-Verlag, Berlin, 2010.

[22] K. Martin, "Using Separation Algorithms to Generate Mixed Integer Model Reformulations", *Operations Research Letters*, vol. **10**, pp. 119–128, 1991.

[23] E. Nowicki and C. Smutnicki, "An Advanced Tabu Search Algorithm for the Job Shop Problem", *Journal of Scheduling*, *8*, pp. 145 – 159, 2005.

[24] C.N. Potts and V.A. Strusevich, "Fifty years of scheduling: a survey of milestones", *Journal of the Operational Research Society*, **60**, pp. S41–S68, 2009.

[25] M. Queyranne and M. Sviridenko, "Approximation algorithms for shop scheduling problems with minsum objective", *Journal of Scheduling*, **5**, pp. 287–305, 2002.

[26] M.W.P. Savelsbergh, R.N. Uma and J. Wein, "An Experimental Study of LP-Based Approximation Algorithms for Scheduling Problems", *INFORMS Journal on Computing*,**17**, pp. 123–136, 2005.

[27] R.J.M. Vaessens, E.H.L. Aarts and J.K. Lenstra, "Job Shop Scheduling by Local Search", *INFORMS Journal on Computing*, **8**, pp. 302-317, 1996.

[28] J.M. Van den Akker, H. Hoogeveen and S. Van de Velde, "Parallel machine scheduling by column generation", *Operations Research*, vol. **47**, pp. 862–872, 1997.

[29] J.M. Van den Akker, C.A.J. Huskens and M.W.P. Savelsbergh, "Time-Indexed formulations for machine scheduling problems: column generation", *INFORMS Journal on Computing*, vol. **12**, pp. 111–124, 2000.

[30] J.P. Watson, L. Barbulescu, A.E. Howe and L.D. Whitley, "Algorithm Performance and Problem Structure for Flow-shop Scheduling", *Proceedings of the Sixteenth National Conference on Artificial Intelligence*, pp. 688–695, 1999.

[31] C.Y Zhang, P.G. Li, Y.Q. Rao and Z.L. Guan, "A very fast TS/SA algorithm for the job shop scheduling problem", *Computers & Operations Research*, **35**, pp. 282–294, 2008.

Instance	LB1	LB2	UB	GAP	B&B	CPU
1	143.16	144.12	145*	1.27	1	5
2	144.00	–	144*	0.00	1	1
3	132.12	–	133*	0.66	1	4
4	139.50	–	140*	0.38	1	1
5	140.00	–	140*	0.00	1	1
6	152.81	153.65	157*	2.67	9	16
7	139.66	–	140*	0.24	1	1
8	171.44	172.10	174*	1.47	5	15
9	130.79	131.18	132*	0.92	1	3
10	149.00	–	149*	0.00	1	1

Table 1: Flow-Shop: 5 jobs, 3 machines

Instance	LB1	LB2	UB	GAP	B&B	CPU
1	202.99	203.87	207*	1.94	13	85
2	227.00	–	227*	0.00	1	2
3	192.58	194.70	196*	1.74	5	23
4	195.07	195.60	197*	0.98	3	17
5	203.66	–	204*	0.17	1	2
6	210.66	212.10	214*	1.57	3	19
7	215.70	217.00	218*	1.05	1	12
8	192.39	194	194*	0.83	1	10
9	260.66	261.74	264*	1.26	5	29
10	199.59	200.64	204*	2.16	3	25

Table 2: Flow-Shop: 5 jobs, 5 machines

Instance	LB1	LB2	UB_0	UB_1	UB_2	GAP	B&B	CPU
1	474.99	476.04	486	486	486	2,26	102	900
2	483.01	483.49	486	486*	486*	0.61	23	221
3	394.59	395.21	404	403	403	1.99	119	900
4	388.38	389.50	395	393*	393*	1.18	45	287
5	437.54	439.09	448	447	447	1.12	111	900
6	438.80	439.52	447	447	447	1.83	84	900
7	377.99	378.62	386	386	386	2.07	124	900
8	497.87	499.76	504	504*	504*	1.22	13	170
9	386.35	387.11	396	394	394	1.94	122	900
10	420.85	421.55	428	426*	428	1.21	88	900

Table 3: Flow-Shop: 10 jobs, 3 machines

Instance	LB1	LB2	UB_0	UB_1	UB_2	GAP	B&B	CPU
1	586.62	591.26	615	607	603	2.72	27	900
2	630.16	631.87	656	656	658	3.50	21	900
3	548.22	549.97	578	571	575	3.99	25	900
4	567.96	568.74	600	600	591	3.89	25	900
5	569.39	570.75	597	597	596	3.98	35	900
6	581.79	584.93	607	604	604	3.36	33	900
7	575.49	576.32	599	599	586	1.79	18	900
8	622.39	624.81	640	635	635	1.98	30	900
9	512.18	513.69	541	538	535	4.26	31	900
10	659.31	661.04	673	673	673	1.60	17	900

Table 4: Flow-Shop: 10 jobs, 5 machines

Instance	LB	UB	CPU	LB [3]	UB [3]	v^*	GAP
I-10-2-tight-equal-1	460	462*	8	434	(453)	462	0.43
I-10-2-tight-equal-2	437	449*	18	418	458	449	2.67
I-10-2-loose-equal-1	218	225*	2	219	225	225	3.11
I-10-2-loose-equal-2	313	320*	13	313	324	320	2.19
I-10-2-tight-tard-1	178	180*	3	174	195	180	1.11
I-10-2-tight-tard-2	144	146*	2	143	147	146	1.37
I-10-2-loose-tard-1	413	416*	8	413	416	416	0.72
I-10-2-loose-tard-2	136	138*	3	137	138	138	1.45
I-15-2-tight-equal-1	3325	3363	1800	3316	3559	3345	0.60
I-15-2-tight-equal-2	1470	1480*	1069	1449	1579	1480	0.67
I-15-2-loose-equal-1	1028	1042*	138	1032	1142	1042	1.34
I-15-2-loose-equal-2	491	498*	32	490	520	498	1.40
I-15-2-tight-tard-1	783	791*	725	786	913	791	1.01
I-15-2-tight-tard-2	901	906*	45	886	906	906	0.55
I-15-2-loose-tard-1	646	655*	289	650	730	655	1.37
I-15-2-loose-tard-2	279	280*	8	278	310	280	0.35
I-20-2-tight-equal-1	1922	1946	1800	1901	2008	1934	0.62
I-20-2-tight-equal-2	933	942*	590	912	1014	942	0.96
I-20-2-loose-equal-1	2535	2548	1800	2546	2708	2548	0.51
I-20-2-loose-equal-2	3057	3070	1800	3013	3318	3070	0.42
I-20-2-tight-tard-1	1657	1672	1800	1515	1913	1672	0.90
I-20-2-tight-tard-2	1438	1474	1800	1375	1594	1449	0.76
I-20-2-loose-tard-1	1195	1204*	459	1194	1271	1204	0.75
I-20-2-loose-tard-2	760	769*	887	735	857	769	1.17
I-10-5-tight-equal-1	662	690	1800	660	826	690	4.05
I-10-5-tight-equal-2	749	764*	1374	612	848	764	1.96
I-10-5-loose-equal-1	1627	-	1800	1263	1905	-	14.06
I-10-5-loose-equal-2	882	-	1800	878	1010	984	10.37
I-10-5-tight-tard-1	362	380	1800	361	405	380	4.74
I-10-5-tight-tard-2	568	1724	1800	461	708	-	19.77
I-10-5-loose-tard-1	168	176*	361	168	188	176	4.54
I-10-5-loose-tard-2	452	528	1800	355	572	486	7.00

Table 5: Job-shop: earliness/tardiness costs, 10-15-20 jobs, 2 machines; 10 jobs, 5 machines

Instance	LB root	v^*	CPU	GAP
1	79.07	80	1	1.16
2	83.00	83	1	0.00
3	79.46	82	3	3.10
4	77.83	79	2	1.48
5	80.24	82	4	2.15
6	78.00	80	2	2.50
7	80.00	81	1	1.23
8	85.00	85	1	0.00
9	83.00	83	1	0.00
10	82.00	82	1	0.00

Table 6: Job-shop: 10 jobs, 5 machines, unit processing times

Instance	LB root	v^*	CPU	GAP
1	123.07	131	52	6.05
2	119.34	129	129	7.50
3	117.82	122	5	3.43
4	119.00	126	32	5.56
5	120.45	126	11	4.40
6	124.63	130	61	4.13
7	136.00	136	2	0.00
8	125.63	129	12	2.61
9	125.00	131	77	4.58
10	121.25	128	29	5.27

Table 7: Job-shop: 10 jobs, 10 machines, unit processing times