Lagrangian Bounds for Just-In-Time Job-Shop Scheduling

Philippe Baptiste[1] Marta Flamini[2] Francis Sourd[3]

April 19, 2006

Abstract

We study the Job-Shop Scheduling problem with Earliness and Tardiness penalties. We describe two Lagrangian relaxations of the problem. The first one is based on the relaxation of precedence constraints while the second one is based on the relaxation of machine constraints. We introduce dedicated algorithms to solve the corresponding dual problems. The second one is solved by a simple dynamic programming algorithm while the first one requires the resolution of an NP-Hard problem by Branch and Bound. In both cases, the relaxations allow us to derive lower bounds as well as heuristic solutions. We finally introduce a simple local search algorithm to improve the best solution found. Computational results are reported.

[1]CNRS LIX laboratory, Ecole Polytechnique, Palaiseau, France. Philippe.Baptiste@polytechnique.fr
[2]Dipartimento di Informatica e Automazione, University Roma 3, Italy. flamini@dia.uniroma3.it
[3]CNRS LIP6 laboratory, Paris, France. Francis.Sourd@lip6.fr

1 Introduction

The importance of "Just-in-Time" inventory management in industry has motivated the study of theoretical scheduling models that capture the main features of this philosophy. Among these models, a lot of research effort has been devoted to earliness-tardiness problems — where both early completion (which results in the need for storage) and tardy completion are penalized. So far, due to the intractability of these problems, most of the effort has been dedicated to the one-machine problem. In this paper, we consider the earliness-tardiness problem in a complex job-shop environment.

We consider a job-shop scheduling problem that consists of a set of n jobs, $J = \{J_1, \ldots, J_n\}$, and a set of m machines $M = \{M_1, \ldots, M_m\}$. Each job J_i is a chain of n_i ordered operations, $O_i = \{o_i^1, \ldots, o_i^{n_i}\}$ where o_i^k is the k-th operation of job J_i. For each operation o_i^k we have a due date d_i^k and a processing time p_i^k. The operation o_i^k has to be processed by a specified machine $M(o_i^k) \in M$. Likewise, for each machine $M_u \in M$ we denote by $\mathcal{O}(M_u)$ the set of operations that have to be processed by machine M_u.

Our objective is to compute a feasible schedule, *i.e.*, completion times C_i^k for all operations o_i^k. In order to evaluate the cost of the schedule we define $E_i^k = \max(0, d_i^k - C_i^k)$ and $T_i^k = \max(0, C_i^k - d_i^k)$ as the earliness and the tardiness of operation o_i^k. An operation o_i^k is early (respectively tardy) if $E_i^k > 0$ (resp. $T_i^k > 0$).

For each operation o_i^k, we have two penalty coefficients α_i^k and β_i^k penalizing its early or tardy completion. Hence a cost function is defined considering that an early operation is penalized by the cost $\alpha_i^k E_i^k$ and a tardy operation is penalized by the cost $\beta_i^k T_i^k$. The objective to minimize is the sum of earliness and tardiness costs of all operations:

$$\min \sum_{i=1}^{n} \sum_{k=1}^{n_i} (\alpha_i^k E_i^k + \beta_i^k T_i^k) \tag{1}$$

Constraints of our problem are the following:

- The first operation starts after time 0, that is for any i in $\{1, \ldots, n\}$,

$$C_i^1 - p_i^1 \geq 0 \tag{2}$$

- Precedence constraints: for each pair of consecutive operations o_i^k and o_i^{k+1} of the same job, operation o_i^{k+1} cannot be processed before operation o_i^k is completed, that is, for $i = 1, \ldots, n$ and $k = 1, \ldots, n_i - 1$:

$$C_i^k \leq C_i^{k+1} - p_i^{k+1} \tag{3}$$

- Resource constraints (also known as disjunctive constraints): for each machine $M_u \in M$ two operations of two distinct jobs, o_i^k and o_j^h, in the set $\mathcal{O}(M_u)$, cannot be processed simultaneously, that is, for $i, j = 1, \ldots, n, i \neq j$, and for $k = 1, \ldots, n_i$ and $h = 1, \ldots, n_j$:

$$C_i^k \geq C_j^h + p_i^k \quad \text{or } C_j^h \geq C_i^k + p_j^h \tag{4}$$

Beck and Refalo [1] investigate a special case of this problem in which all operations but the last ones of each job have null earliness and tardiness penalty costs. Their approach is based on a hybrid use of constraint programming and linear programming. The linear programming model is used to compute the best start times of the partially sequenced operations while constraint programming helps in fixing disjunctive constraints. Danna and Perron [5] and Danna et al. [6] propose another hybrid method using integer linear programming and local search which is proved to find better schedules. *However, one drawback of this model is that it leads to schedules in which the first $n_i - 1$ operations of each job J_i are processed as early as possible thus inducing waiting times (and storage costs) between the last two operations of the job, which contradicts the Just-in-Time philosophy.* In our model, this issue is solved by considering that $d_i = d_i^{n_i}$ is the due-date of J_i and the due date of any operation o_i^k is $d_i^k = d_i - \sum_{j>k} p_i^j$. Therefore, in the ideal situation where each operation completes at its due date, the job is on time and there is no waiting time between operations.

Several researchers have used Lagrangian relaxations for the job shop problem or for more complex problems such as the Resource Constrained Project Scheduling Problem [7]. In brief, two kinds of relaxations have been studied.

- Starting from a time indexed formulation, one can relax machine constraints. The subproblem to solve is then reduced to scheduling a operations subjected to precedence constraints with arbitrary cost functions. Depending on the initial scheduling constraints, authors use various techniques to solve this problem. Either dynamic programming when the structure of precedence constraints is a chain or a tree [2] or a cut computation in the general case [11] or a simplex-like algorithm [14].

- Chen and Luh [3] relax the precedence constraints as well as constraints linking completion times of operations to the total cost. Once all these constraints are relaxed, the subproblem to solve is decomposed into a set of polynomially solvable problems. A similar approach is used by Kaskavelis and Caramanis [10] to solve job-shop problems with storage costs and quadratic tardiness penalties.

In this paper, we evaluate the efficiency of the two relaxation techniques based respectively on the relaxation of precedence constraints and on the relaxation of resource constraints. The study of the former relaxation is motivated by the recent improvements in branch-and-bound algorithms for the one-machine earliness-tardiness scheduling problem [13]. To the best of our knowledge, this approach is new in the design of lower bounds for this type of problem. Another interest of this lower bound lies in the fact that it is stronger than the relaxation of Chen and Luh [3] because, in our approach, resource constraints are not relaxed at all, that is we have a pure relaxation of the precedence constraints.

This lower bound is introduced in Section 2 while the relaxation of the resource constraints, which follows the approach described in [2], is presented in Section 3. Section 4.1 studies the effectiveness of these lower bounds: first heuristics to compute upper bounds

are presented, then experimental results compare the two lower bounds to lower bounds provided by ILOG CPLEX and to the upper bounds.

2 Relaxing Precedence Constraints

In this section we rely on the basic formulation of the problem which is immediately derived from the definition equations (1-4) of the problem. To strengthen the up-coming relaxation, we define release dates $r_i^k = \sum_{1 \leq h \leq k-1} p_i^h$ for each operation o_i^k. We then have:

$$\min \sum_{i=1}^{n} \sum_{k=1}^{n_i} \alpha_i^k E_i^k + \beta_i^k T_i^k$$

$$u.c. \begin{cases} C_i^k - p_i^k \geq r_i^k & 1 \leq i \leq n, 1 \leq k \leq n_i \\ C_i^k \leq C_i^{k+1} - p_i^{k+1} & 1 \leq i \leq n, 1 \leq k \leq n_i - 1 \\ C_i^k \geq C_j^h + p_i^k \quad \textbf{or} \quad C_j^h \geq C_i^k + p_j^h & 1 \leq u \leq m, o_i^k, o_j^h \in \mathcal{O}(M_u) \\ E_i^k = \max(0, d_i^k - C_i^k) & 1 \leq i \leq n, 1 \leq k \leq n_i \\ T_i^k = \max(0, C_i^k - d_i^k) & 1 \leq i \leq n, 1 \leq k \leq n_i \end{cases}$$

We associate to the precedence constraint between operations o_i^k and o_i^{k+1} a Lagrangian multiplier, $\lambda_i^k \geq 0$. In the following C and λ denote respectively the vector of completion times and the vector of Lagrangian multipliers. The Lagrangian function $L(C, \lambda)$ can then be defined as follows.

$$L(C, \lambda) = \sum_{i=1}^{n} \sum_{k=1}^{n_i} \alpha_i^k E_i^k + \beta_i^k T_i^k + \sum_{i=1}^{n} \sum_{k=1}^{n_i - 1} \lambda_i^k \left(C_i^k - C_i^{k+1} + p_i^{k+1} \right) \tag{5}$$

Under the assumption $\lambda_i^0 = \lambda_i^{n_i} = 0$ we can write (5) in a more compact form:

$$L(C, \lambda) = \sum_{i=1}^{n} \sum_{k=1}^{n_i} \alpha_i^k(\lambda) E_i^k + \beta_i^k(\lambda) T_i^k - K(\lambda) \tag{6}$$

where

$$\begin{cases} K(\lambda) &= \sum_{i=1}^{n} \sum_{k=1}^{n_i} \lambda_i^k (p_i^{k+1} + d_i^k - d_i^{k+1}) \\ \alpha_i^k(\lambda) &= \alpha_i^k - \lambda_i^k + \lambda_i^{k+1} \\ \beta_i^k(\lambda) &= \beta_i^k + \lambda_i^k - \lambda_i^{k+1} \end{cases}$$

For a given value of vector λ we search for the minimum $L(\lambda)$ of $C \rightarrow L(\lambda, C)$ under the remaining constraints.

$$L(\lambda) = \min \sum_{i=1}^{n} \sum_{k=1}^{n_i} \alpha_i^k(\lambda) E_i^k + \beta_i^k(\lambda) T_i^k - K(\lambda)$$

$$u.c. \begin{cases} C_i^k - p_i^k \geq r_i^k & 1 \leq i \leq n, 1 \leq k \leq n_i \\ C_i^k \geq C_j^h + p_i^k \quad \textbf{or} \quad C_j^h \geq C_i^k + p_j^h & 1 \leq u \leq m, o_i^k, o_j^h \in \mathcal{O}(M_u) \\ E_i^k = \max(0, d_i^k - C_i^k) & 1 \leq i \leq n, 1 \leq k \leq n_i \\ T_i^k = \max(0, C_i^k - d_i^k) & 1 \leq i \leq n, 1 \leq k \leq n_i \end{cases}$$

This problem can be decomposed into m independent single-machine scheduling subproblems with earliness and tardiness penalty costs, *i.e.*, $L(\lambda) = \sum_{u=1}^{m} L_u(\lambda) - K(\lambda)$, where $L_u(\lambda)$ is exactly

$$\min \sum_{o_i^k \in \mathcal{O}(M_u)} \alpha_i^k(\lambda) E_i^k + \beta_i^k(\lambda) T_i^k$$

$$u.c. \begin{cases} C_i^k - p_i^k \geq r_i^k & o_i^k \in \mathcal{O}(M_u) \\ C_i^k \geq C_j^h + p_i^k \quad \textbf{or} \quad C_j^h \geq C_i^k + p_j^h & o_i^k, o_j^h \in \mathcal{O}(M_u) \\ E_i^k = \max(0, d_i^k - C_i^k) & o_i^k \in \mathcal{O}(M_u) \\ T_i^k = \max(0, C_i^k - d_i^k) & o_i^k \in \mathcal{O}(M_u) \end{cases}$$

It is easy to see that $P_u(\lambda)$ is a single machine scheduling problem with Earliness/Tardiness penalties. This problem is NP-Hard in the strong sense but can be solved efficiently by an advanced Branch and Bound procedure of Sourd and Kedad-Sidhoum [13]. Then we look for the vector λ^* that maximizes the function $L(\lambda)$ by applying a standard sub-gradient procedure. At each iteration, we modify the vector of Lagrangian multipliers by adding $\rho\nabla$ where $\rho \in \mathbb{R}^+$ and where ∇ is a sub-gradient. At first, $\rho = 1$ and all multipliers equal 0 and then, ρ is multiplied by 0.7 when the value of the objective function is not improved for ten consecutive iterations. The algorithm is stopped when the parameter ρ is less than 10^{-3}.

As the Branch and Bound is very time-consuming, *we speed up the convergence* by the following heuristic. In the preliminary steps of the convergence, we use a basic heuristic

[9] to compute upper bounds of the single machine problems (therefore, we do not have at these steps valid lower bounds). This is used only, to converge towards a "good" λ. Once this preliminary convergence phase is achieved, we move to the Branch and Bound.

Instead of using a subgradient procedure, we have tried to use `Solvopt` implementation of Shor's algorithm [12]. However, due to the time required to solve the Lagrangian subproblems, we could not complete the convergenece process and the obtained solution was not better than the one obtained by the basic subgradient algorithm.

In a theoretical point of view, we note that $L(\lambda^*)$ strictly dominates the relaxation of Chen and Luh [3] as we only relax the precedence constraints while they also relax the constraints $E_i^k = \max(0, d_i^k - C_i^k)$ and $T_i^k = \max(0, C_i^k - d_i^k)$.

3 Relaxing Resource Constraints

In this section, we consider the Lagrangian relaxation of resource constraints. We rely on a time-indexed formulation of the problem [8]. We introduce a time horizon T that can be computed as follows:

$$T = \max_{i,k} d_i^k + \sum_{i,k} p_i^k$$

We also introduce the set of binary $\{0, 1\}$ variables, in which the generic variable x_{ik}^t refers operation o_i^k in the instant t. Each variable x_{ik}^t is defined as follows:

$$x_{ik}^t = \begin{cases} 1 & \text{if operation } o_i^k \text{ starts at time } t \\ 0 & \text{otherwise} \end{cases}$$

Considering this new set of variables, we can compute the earliness and tardiness costs as follows.

$$E_i^k = \sum_{t=0}^{T-1} x_{ik}^t \max(0, d_i^k - t - p_i^k) \qquad T_i^k = \sum_{t=0}^{T-1} x_{ik}^t \max(0, t + p_i^k - d_i^k)$$

Now consider the resource constraints. For each machine $M_u \in M$ and for each instant time $t \in [0, T]$, two operations in $\mathcal{O}(M_u)$, can not overlap, *i.e.*,

$$\sum_{i=1}^{n} \sum_{k:\begin{cases} 1 \leq k \leq n_i \\ M(o_i^k) = u \end{cases}} \sum_{\tau=t-p_i^k+1}^{t} x_{ik}^{\tau} \leq 1 \tag{7}$$

Hence, given a time instant $t \in [0, T]$ and a machine $u \in M$, the above constraint means that if an operation $o_i^k \in \mathcal{O}(M_u)$ starts before t and has not yet been completed in t, at least for $t - p_i^k + 1$ time instants before t, no other operations in $\mathcal{O}(M_u)$ can start. Altogether, the initial problem can be formulated as follows.

$$\min \sum_{i=1}^{n} \sum_{k=1}^{n_i} \alpha_i^k E_i^k + \beta_i^k T_i^k$$

$$u.c. \begin{cases} C_i^k = \sum_{t=0}^{T-1} t x_{ik}^t + p_i^k & 1 \leq i \leq n, 1 \leq k \leq n_i \\ C_i^k \leq C_i^{k+1} - p_i^{k+1} & 1 \leq i \leq n, 1 \leq k \leq n_i - 1 \\ E_i^k = \max(0, d_i^k - C_i^k) & 1 \leq i \leq n, 1 \leq k \leq n_i \\ T_i^k = \max(0, C_i^k - d_i^k) & 1 \leq i \leq n, 1 \leq k \leq n_i \\ \sum_{i=1}^{n} \sum_{k:\begin{cases} 1 \leq k \leq n_i \\ M(o_i^k) = u \end{cases}} \sum_{\tau=t-p_i^k+1}^{t} x_{ik}^{\tau} \leq 1 & 1 \leq u \leq m, 0 \leq t \leq T-1 \\ x_{ik}^t \in \{0, 1\} & 1 \leq i \leq n, 1 \leq k \leq n_i, 0 \leq t \leq T-1 \end{cases}$$

We associate a Lagrangian multiplier $\lambda_{ut} \geq 0$ to each resource constraint. As earliness and tardiness can be expressed as

$$E_i^k = \sum_{t=0}^{T-1} x_{ik}^t \max(0, d_i^k - t - p_i^k) \qquad T_i^k = \sum_{t=0}^{T-1} x_{ik}^t \max(0, t + p_i^k - d_i^k)$$

the Lagrangian function $L(\lambda)$ is exactly

$$\min \sum_{i=1}^{n} \sum_{k=1}^{n_i} \sum_{t=0}^{T-1} w_{ik}^t(\lambda) x_{ik}^t - K(\lambda)$$

$$u.c. \begin{cases} C_i^k = \sum_{t=0}^{T-1} t x_{ik}^t + p_i^k & 1 \leq i \leq n, 1 \leq k \leq n_i \\ C_i^k \leq C_i^{k+1} - p_i^{k+1} & 1 \leq i \leq n, 1 \leq k \leq n_i - 1 \\ x_{ik}^t \in \{0,1\} & 1 \leq i \leq n, 1 \leq k \leq n_i, 0 \leq t \leq T-1 \end{cases}$$

where w_{ik}^t and K are simple functions of λ

$$w_{ik}^t(\lambda) = \alpha_i^k \max(0, d_i^k - t - p_i^k) + \beta_i^k \max(0, t + p_i^k - d_i^k) + \sum_{u=1}^{m} \sum_{s=t}^{t+p_i^k-1} \lambda_{us}$$

and

$$K(\lambda) = \sum_{u=1}^{m} \sum_{t=0}^{T-1} \lambda_{ut}$$

It is easy to see that $L(\lambda)$ can be decomposed as $\sum_{i=1}^{n} L_i(\lambda) - K(\lambda)$ where $L_i(\lambda)$ equals

$$\min \sum_{k=1}^{n_i} \sum_{t=0}^{T-1} w_{ik}^t(\lambda) x_{ik}^t$$

$$u.c. \begin{cases} C_i^k = \sum_{t=0}^{T-1} t x_{ik}^t + p_i^k & 1 \leq k \leq n_i \\ C_i^k \leq C_i^{k+1} - p_i^{k+1} & 1 \leq k \leq n_i - 1 \\ x_{ik}^t \in \{0,1\} & 1 \leq k \leq n_i, 0 \leq t \leq T-1 \end{cases}$$

Solving a single subproblem consists in scheduling the operations $o_i^1 \ldots o_i^{n_i}$ of job J_i, considering only the precedence constrains among them, in order to minimize an arbitrary objective function. As noticed by several authors [2, 11], each problem can be solved by dynamic programing in $O(n_i T)$.

We search for the optimal value of Lagrangian multiplier vector, λ^*, that maximizes the function $L(X^*, \lambda)$ iterating the above computation by applying the sub-gradient method described in Section 2.

Note that we could use the r_i^k values as defined in Section 2 to tighten the formulation of the problem : $\forall t < r_i, x_{ik}^t = 0$. However, this does not help as in the solution of the Lagrangian problem we always have $\forall t < r_i, x_{ik}^t = 0$.

Like in the first relaxation, we have tried to use `Solvopt` [12]. Once again, this was not competitive with the subgradient procedure. This is likely due to the large number of multipliers and to the non-negativity constraints.

4 Experimental Results

4.1 Upper Bounds

In order to evaluate the quality of the lower bounds, we have implemented simple heuristics to derive upper bounds. First, we have used the Lagrangian relaxations to build feasible (suboptimal) schedules. Such techniques, known as primalization procedures, often provide reasonably good upper bounds. At the end of both sub-gradient procedures, we have a relaxed schedule, *i.e.*, a schedule that does not meet all the constraints (in the first case, some precedence constraints might be violated, while in the second one, operation requiring the same machine might overlap in time).

- We first convert the relaxed schedule into a feasible schedule. Our basic idea is to rely on the relaxed schedule to iteratively build a left shifted job-shop schedule that is feasible. This is done as follows: At each iteration, select an unscheduled operation, whose job predecessor is scheduled, and whose completion time (in the relaxed schedule) is minimum. Schedule this operation as soon as possible after the completion time of its predecessor and after the first time point at which the machine it requires is available.

- This left shifted schedule is then improved as follows: We build a precedence graph of all operations in which we add an edge between consecutive operations of the

same job and consecutive operations of the same machine. We then look for an optimal schedule of operations meeting the precedence graph. This problem reduces to a project scheduling problem with Early/Tardy cost functions (and no resource constraints). This can be solved in polynomial time by [4] or by linear programming.

Unfortunately, our tests have shown that these feasible solutions are not near-optimal. To improve the upper bound, we apply a simple local search method starting from the best known solution. A solution is defined as a the sequence of operations on all machines. As mentioned earlier, given a arbitrary set of sequences, we can compute in polynomial time the optimal schedule meeting this sequence. Of course, we can also detect that there is cycle in the sequence.

Two moves are considered in our local search. Given a sequence, we either swap two randomly chosen operations or we insert an operation at some other place in its sequence. We change the current sequence as soon as the cost of the resulting sequence is improved (provided of course that no cycle is induced by the move).

4.2 Instances

For each $(n, m) \in (\{10, 15, 20\} \times \{2, 5, 10\})$, we have generated 8 instances. Each instance is named according to the following pattern `I-n-m-DD-W-ID`. The meaning of `DD`, `W` and `ID` is explained below. Jobs are processed exactly once on each machine and the machine $M(o_i^k)$ in which operation o_i^k has to be processed is chosen randomly. Processing times are chosen in $[10, 30]$ and due dates are chosen in such a way that

- Either the distance between due dates of consecutive operations is exactly equal to the processing time of the last operation (`DD = tight`)
- Or, the distance between due dates of consecutive operations is exactly equal to the processing time of the last operation plus a random number taken in [0, 10] (`DD = loose`).

The earliness and tardiness costs (α, β) are either both chosen randomly in $[0.1, 1]$ (`W = equal`) or α is chosen randomly in $[0.1, 0.3]$ while β is chosen randomly in $[0.1, 1]$ (`W = tard`). The latter generation scheme corresponds to the situation where tardiness costs dominates earliness costs, which is usually the case in real-case problems.

Two instances (`ID = 1`) or (`ID = 2`) have been generated for each combination of parameters. Hence leading to $3 \times 3 \times 2 \times 2 \times 2 = 72$ instances of the problem. All of them are available online[1].

4.3 Results

Our results are reported in Tables 1, 2 and 3. In these tables, the columns "LBR" and "LBP" respectively reports the lower bounds given by the Lagrangian resource constraint relaxation and the Lagrangian precedence relaxation. Respective computation times are indicated in the next columns ("CPU"). The column "CPLEX" reports the lower bound computed by ILOG CPLEX 9.1 after 600s using the mixed integer program defined at the beginning of Section 2. The column "UB" shows the upper bound, which corresponds to the best solution found either by our heuristics or by ILOG CPLEX. Finally, the column "GAP" indicates the gap between the best lower bound and the upper bound. It is equal to $\frac{\text{UB}-\text{LB}}{\text{UB}}$.

For the computation of LBP for the instances with 20 jobs, we have had to limit the number of steps of the sub-gradient procedure in order to keep reasonable computation times. The number of steps has been limited to 20 (after the preliminary step which calls the heuristic), which means that at most 400 branch-and-bound have to be solved. Moreover we also stopped the procedure when a branch-and-bound procedure reaches the time limit of 1000s. Therefore, some lower bounds are not available: it means that the time limit has been reached at the first iteration of the sub-gradient procedure.

[1] http://www-poleia.lip6.fr/∼sourd/project/etjssp/

Instance	LBR	CPU	LBP	CPU	CPLEX	UB	GAP(%)
I-10-2-tight-equal-1	**434**	13	433	8.4	289	453	4.2
I-10-2-tight-equal-2	357	7	**418**	3.9	332	458	8.7
I-10-5-tight-equal-1	**660**	32	536	61	389	826	20.1
I-10-5-tight-equal-2	592	23	**612**	29	304	848	27.8
I-10-10-tight-equal-1	**1126**	121	812	240	668	1439	21.7
I-10-10-tight-equal-2	**1535**	363	819	560	985	2006	23.5
I-10-2-loose-equal-1	218	6	**219**	4.1	161	225	2.6
I-10-2-loose-equal-2	**313**	10	298	6.6	245	324	3.4
I-10-5-loose-equal-1	**1263**	114	1205	61	759	1905	33.7
I-10-5-loose-equal-2	**878**	126	780	104	558	1010	13.1
I-10-10-loose-equal-1	**331**	117	294	222	247	376	11.9
I-10-10-loose-equal-2	**246**	62	211	170	162	260	5.3
I-10-2-tight-tard-1	168	7	**174**	12	89	195	10.7
I-10-2-tight-tard-2	**143**	7	138	35	107	147	2.7
I-10-5-tight-tard-1	**361**	41	322	230	189	405	10.8
I-10-5-tight-tard-2	420	26	**461**	81	280	708	34.9
I-10-10-tight-tard-1	**574**	158	408	551	310	855	32.9
I-10-10-tight-tard-2	**666**	207	469	588	465	800	16.7
I-10-2-loose-tard-1	413	9	408	12	**416**	416	0.0
I-10-2-loose-tard-2	135	9	**137**	8.7	131	138	0.7
I-10-5-loose-tard-1	**168**	42	159	79	97	188	10.6
I-10-5-loose-tard-2	**355**	17	313	40	112	572	37.9
I-10-10-loose-tard-1	**356**	228	314	345	281	409	12.9
I-10-10-loose-tard-2	**138**	60	119	152	106	152	9.2

Table 1: Instances with 10 Jobs

Instance	LBR	CPU	LBP	CPU	CPLEX	UB	GAP(%)
I-15-2-tight-equal-1	2902	136	**3316**	143	627	3559	6.8
I-15-2-tight-equal-2	1253	17	**1449**	76	377	1579	8.2
I-15-5-tight-equal-1	964	26	**1052**	260	251	1663	36.7
I-15-5-tight-equal-2	1630	79	**1992**	355	433	2989	33.4
I-15-10-tight-equal-1	**4389**	555	3662	1138	1173	8381	47.6
I-15-10-tight-equal-2	**3539**	825	2564	2454	912	7039	49.7
I-15-2-loose-equal-1	1014	13	**1032**	33	290	1142	9.6
I-15-2-loose-equal-2	**490**	10	472	56	133	520	5.0
I-15-5-loose-equal-1	2449	218	**2763**	1569	569	4408	37.3
I-15-5-loose-equal-2	**2818**	323	2773	676	630	4023	29.9
I-15-10-loose-equal-1	**758**	267	628	1836	223	1109	31.6
I-15-10-loose-equal-2	**1242**	395	979	2880	382	2256	44.9
I-15-2-tight-tard-1	720	16	**786**	741	103	913	13.9
I-15-2-tight-tard-2	843	11	**886**	72	162	956	7.3
I-15-5-tight-tard-1	1008	39	**1014**	556	140	1538	34.1
I-15-5-tight-tard-2	**626**	34	547	10280	129	843	25.7
I-15-10-tight-tard-1	**649**	77	467	141283	118	972	33.2
I-15-10-tight-tard-2	**955**	268	761	81347	244	1656	42.3
I-15-2-loose-tard-1	616	13	**650**	62	95	730	10.9
I-15-2-loose-tard-2	**278**	13	277	148	84	310	10.3
I-15-5-loose-tard-1	**1098**	110	1005	3538	225	1723	36.2
I-15-5-loose-tard-2	**314**	29	313	3384	62	374	16
I-15-10-loose-tard-1	**258**	106	233	1295	48	312	17.3
I-15-10-loose-tard-2	**476**	243	454	3586	122	855	44.3

Table 2: Instances with 15 Jobs

Instance	LBR	CPU	LBP	CPU	CPLEX	UB	GAP(%)
I-20-2-tight-equal-1	1747	16	**1901**	1730	243	2008	5.3
I-20-2-tight-equal-2	858	14	**912**	37	94	1014	10.1
I-20-5-tight-equal-1	**2506**	152	2244	418	304	3090	18.8
I-20-5-tight-equal-2	4923	329	**5817**	7585	830	7537	22.8
I-20-10-tight-equal-1	6656	1226	**6708**	1970	842	12951	48.2
I-20-10-tight-equal-2	**5705**	1190	-	-	928	9435	39.5
I-20-2-loose-equal-1	2388	25	**2546**	1250	264	2708	6.0
I-20-2-loose-equal-2	2970	63	**3013**	351	340	3318	9.2
I-20-5-loose-equal-1	5571	639	**6697**	1042	632	9697	30.9
I-20-5-loose-equal-2	5496	518	**6017**	839	879	8152	26.2
I-20-10-loose-equal-1	**3538**	1243	3099	5083	372	6732	47.4
I-20-10-loose-equal-2	**1344**	279	1150	2268	94	2516	46.6
I-20-2-tight-tard-1	**1515**	26	-	-	114	1913	20.8
I-20-2-tight-tard-2	**1375**	31	1327	805	156	1594	13.7
I-20-5-tight-tard-1	2507	142	**3244**	735	212	4147	21.8
I-20-5-tight-tard-2	**1633**	154	-	-	197	1916	14.7
I-20-10-tight-tard-1	**3003**	623	2764	9704	272	5968	49.6
I-20-10-tight-tard-2	**2740**	865	-	-	358	3788	27.6
I-20-2-loose-tard-1	**1194**	22	1189	447	129	1271	6.0
I-20-2-loose-tard-2	734	13	**735**	73	63	857	14.2
I-20-5-loose-tard-1	2177	249	**2524**	2707	218	3377	25.3
I-20-5-loose-tard-2	2643	147	**3060**	892	134	5014	39.0
I-20-10-loose-tard-1	**2462**	678	2436	22196	151	6237	60.5
I-20-10-loose-tard-2	**1226**	392	-	-	129	1830	33

Table 3: Instances with 20 Jobs

When comparing the two Lagrangian lower bounds, we observe that LBR is better than LBP for nearly all instances with 10 machines. Conversely, for instances with only 2 machines, LBP often outperforms LBR. Unsurprisingly, the relaxation of precedence constraints requires more computation time since we have to solve m instances of an NP-hard problem at each step of the sub-gradient algorithm. Unfortunately, these instances are sometimes very difficult since the optimum cannot be proved within 1000s.

We also observe that both LBR and LBP are better than the lower bound returned by ILOG CPLEX. For all our instances, ILOG CPLEX finds the best lower bound only once (in fact it is the only instance, namely `I-10-2-loose-tard-1`, that can be solved within 600s). We also observe that the quality of the lower bound significantly deteriorates when the number of jobs increases.

While the gap is moderate for instances with 2 machines, it is rather large for several 10 machine instances. These results show that these problems are greatly intractable.

5 Conclusion

Two Lagrangian relaxations for the job-shop have been compared: the relaxation of the resource constraints and the relaxation of the precedence constraints. When the number of machines is large enough, the relaxation of the resource constraints most often leads to better lower bounds. Still, there is a large gap for most of the studied instances and we believe that a lot of work remains to be carried on Just-In-Time Job-Shop Scheduling.

Acknowledgments

The authors are grateful to Filippo Focacci, Claude Le Pape, and Dario Pacciarelli for enlightening discussions on Just-In-Time scheduling.

References

[1] Beck, J.C. and Refalo, P. (2003). A Hybrid Approach to Scheduling with Earliness and Tardiness Costs. *Annals of Operations Research* **118**, 49–71.

[2] Chen, H., Chu, C. and Proth, J.M. (1998). An Improvement of the Lagrangean Relaxation Approach for Job Shop Scheduling: A Dynamic Programming Method. *IEEE Transactions on Robotics and Automation* **14**, 786–795.

[3] Chen, A. and Luh, P. (2003). An alternaltive framework to Lagrangian relaxation approach for job shop scheduling. *European Journal of Operational Research* **149**, 499–512.

[4] Chrétienne, Ph. and Sourd F. (2003). PERT Scheduling with Convex Cost Functions. *Theoretical Computer Science* **292**, 145–164.

[5] Danna, E. and Perron, L. (2003). Structured vs. Unstructured Large Neighborhood Search: A Case Study on Job-Shop Scheduling Problems with Earliness and Tardiness Costs. In: Principles and Practice of Constraint Programming CP 2003. *Lecture Notes in Computer Science* **2833**, 817–821.

[6] Danna, E., Rothberg, E. and Le Pape, C. (2003). Integrating Mixed Integer Programming and Local Search : A Case Study on Job-Shop Scheduling Problems, Proceedings of CPAIOR'03. http://www.crt.umontreal.ca/cpaior/

[7] Demeulemeester, E.L. and Herroelen, W.S. (2002). *Project Scheduling: A Research Handbook*. Kluwer Academic Publishers.

[8] Dyer, M.E. and Wolsey, L.A. (1990). Formulating the single machine sequencing problem with release dates as a mixed integer problem. *Discrete Applied Mathematics* **26**, 255-270.

[9] Hendel, Y. and Sourd F. (2006). Efficient neighborhood search for the one-machine earliness-tardiness scheduling problem. To appear in *European Journal of Operational Research*.

[10] Kaskavelis, C.A. and Caramanis M.C. (1998). Efficient Lagrangian relaxation algorithms for industry size job-shop scheduling problems. *IIE Transactions* **30**, 1085–1097.

[11] Möhring, R., Schulz, A., Stork, F., and Uetz, M (2003). Solving Project Scheduling Problems by Minimum Cut Computations. *Management Science* **49**, 330–350.

[12] Kappel, F. and Kuntsevich, A.V. (2000). An implementation of Schor's $r-$algorithm. *Computational Optimization and Applications* **15**, 193–205.

[13] Sourd, F. and Kedad-Sidhoum, S. (2003). The one machine problem with earliness and tardiness penalties. *Journal of Scheduling* **6**, 533–549.

[14] Vanhoucke, M., Demeulemeester, E. and Herroelen, W. (2001): An Exact Procedure for the Ressource-Constrained Weighted Earliness-Tardiness Project Scheduling Problem. *Annals of Operations Research* **102**, 179–196.