Generalized disjunctive constraint propagation for solving the job shop problem with time lags

Christian Artigues, Marie-José Huguet, Pierre Lopez

Generalized disjunctive constraint propagation for solving the job shop problem with time lags

Christian Artigues[1,2] Marie-José Huguet[1,2] Pierre Lopez[1,2]

[1] CNRS ; LAAS ; 7 avenue du colonel Roche, F-31077 Toulouse, France
[2] Université de Toulouse ; UPS, INSA, INP, ISAE ; LAAS ; F-31077 Toulouse, France
e-mails: {artigues,huguet,lopez}@laas.fr

Abstract

In this paper we propose new insights based on an insertion heuristic and generalized resource constraint propagation for solving the job shop scheduling problem with minimum and maximum time-lags. To show the contribution of our propositions we propose a branch-and-bound algorithm and provide an experimental study. The results obtained conclude that our heuristic obtains feasible schedules with a better makespan than previous approaches, especially for instances with tightened time lags. The results also prove the interest of the constraint propagation generalization when time lags are considered.

Keywords: Job shop scheduling; time lags; constraint propagation; insertion heuristic.

1 Introduction

The job shop problem with minimum and maximum time-lags (JSPTL) is a generalization of the well-known job shop problem (JSP), in which there are time constraints restricting the minimum or the maximum distance between two successive job operations. The JSPTL involves a set of jobs that have to be processed on a set of machines. Each job i consists of a sequence of operations; (i, j) denotes the j^{st} operation of job i. Every operation must be assigned to a unique machine without interruption. The distance between the end of an operation (i, j) and the start of its successor $(i, j+1)$ is constrained to belong to interval $[TL^{\min}_{i,j,j+1}, TL^{\max}_{i,j,j+1}]$. Solving the JSPTL consists in sequencing all operations on the machines, such that successive operations of the same job satisfy time lag constraints and such that each machine processes at most one operation at a time. The objective is to find a schedule that minimizes the makespan.‰ The problem can be denoted by $Jm|TL_{i,j,j+1}|C_{\max}$.

The classical JSP is a well-addressed problem in the literature but only few articles are concerned with time-lag constraints. Wikum *et al.* [24] study single-machine problems with minimum and/or maximum distances between jobs and state that some particular single-machine problems with time-lags are polynomially solvable, even if the general case is NP-hard. Brucker *et al.* [3] show that many scheduling problems (such as multi-processor tasks or multi-purpose machines) can be modeled as single-machine problems with time-lags and propose a branch-and-bound method. A local search approach can be found in [13]. Caumond *et al.* [5] study the JSPTL considered in this paper. They propose an insertion heuristic and a memetic algorithm. Furthermore, since the JSPTL can be viewed as a special case of the resource-constrained project scheduling problem (RCPSP) with time lags, the relevant literature on this problem also apply to the JSPTL [15, 17]. However, in general, finding a feasible solution with time lags is an NP-complete problem for the RCPSP. This is not the case for the JSPTL, for which a trivial solution, called canonical schedule, can be obtained by a greedy algorithm [5]. The algorithm simply sorts jobs in an arbitrary order and, for each job taken in this order, schedules all its operations at the earliest possible time at the end of the partial schedule. For $|J|$ jobs, there are $|J|!$ so-obtained "canonical" schedules with the same makespan, which is the sum of the duration of all the operations. The interest of a specific study of the JSPTL comes from this property. However, for makespan minimization, the canonical schedule may have a very poor performance. In the presence of maximum time-lags, classical JSP heuristics cannot be easily extended and finding a non-trivial feasible schedule for the JSPTL is not simple due to maximum time-lags constraints.

In this paper, we propose a new heuristic for computing a feasible schedule for the JSPTL and a branch-and-bound method including new generalized resource constraint propagation techniques for solving the JSPTL. Constraint propagation is commonly used when solving decision problems from a constraint satisfaction perspective; planning and scheduling problems are not an exception to the rule (see for example [20]). The experiments show the interest of these generalized propagations compared to usual constraint propagation. Moreover, our method outperforms the best-known approaches in the literature for small instances.

This paper is organized as follows. In Section 2, we present the job shop problem with time-lags under study and a literature review. Section 3 is dedicated to new resource constraint propagation based on generalized time constraints. The proposed method for solving the JSPTL is detailed in Section 4. This method is based on a branch-and-bound procedure, on a new insertion heuristic for the JSPTL, and on the generalized resource constraint propagation. In Section 5, we provide an experimental study to evaluate the impact of the proposed insertion heuristic and to evaluate the interest of generalized resource constraint propagation for solving the JSPTL. Conclusions and future research directions are given in Section 6.

2 The job shop problem with time lags

2.1 Notations

The following notations are used throughout the paper:

- M: the set of machines;
- $J = \{J_i\}_{i=1..|J|}$: the set of jobs;
- T: the set of operations;
- T_μ: the set of operations which have to be processed on the same machine $\mu \in M$;
- (i, j): the j^{th} operation of job i;

- n_i: the last index of the operations of job i;
- $m_{i,j}$: the machine allocated to operation (i,j);
- $p_{i,j}$: the duration of operation (i,j);
- $TL^{\min}_{i,j,j+1}$ and $TL^{\max}_{i,j,j+1}$: the minimum and maximum time-lags between operations (i,j) and $(i,j+1)$, respectively;
- $st_{i,j}$ and $ft_{i,j}$: the start- and finish-times of operation (i,j), respectively (to be determined).

2.2 Modeling

The model of the JSPTL is based on a set of decision variables $X = \{st_{i,j}, ft_{i,j}\}_{(i,j)\in T}$, and on a set of constraints (time-lag and resource sharing constraints):

$$\begin{aligned}
&\min \quad C_{\max}, && && (1)\\
&\text{s.t. :} && && (2)\\
&C_{\max} \geq ft_{i,j}, && \forall (i,j) \in T, && (3)\\
&st_{i,j+1} \geq ft_{i,j} + TL^{\min}_{i,j,j+1}, && \forall i = 1,...,|J|,\ j = 1,...,n_i - 1, && (4)\\
&st_{i,j+1} \leq ft_{i,j} + TL^{\max}_{i,j,j+1}, && \forall i = 1,...,|J|,\ j = 1,...,n_i - 1, && (5)\\
&ft_{i,j} = st_{i,j} + p_{i,j}, && \forall (i,j) \in T, && (6)\\
&(st_{i,j} \geq ft_{k,l}) \vee (st_{k,l} \geq ft_{i,j}), && \forall (i,j),(k,l) \in T_\mu, \forall \mu \in M, && (7)\\
&st_{i,j} \geq 0, && \forall (i,j) \in T. && (8)
\end{aligned}$$

Constraints (3) state that the makespan is greater than or equal to the finish time of all operations of each job. Constraints (4) and (5) represent the minimum and maximum time lags between two consecutive operations of the same job i, respectively. Constraints (6) describe duration constraints. Constraints (7) state that operations competing for the same machine must be sequenced.

As an illustrative example, let consider the following JSPTL with three jobs composed of three operations described in Table 1. Each cell in this table gives the machine, the duration, and the maximum time-lag between this operation and its successor if it exists. In this instance, we consider that the minimum time-lags are all zero. In the following of this paper, operation durations are assumed to be fixed but this assumption does not impact our contribution. A canonical schedule corresponding to this problem is represented with a makespan equal to $\sum_{(i,j)\in T} p_{i,j} = 57$ in Fig. 1.

Table 1: Example of JSPTL

	Operation 1	Operation 2	Operation 3
J_1	$(m_3, 4, 2)$	$(m_1, 6, 2)$	$(m_2, 6, -)$
J_2	$(m_2, 6, 4)$	$(m_3, 11, 4)$	$(m_1, 8, -)$
J_3	$(m_1, 4, 2)$	$(m_2, 5, 2)$	$(m_3, 7, -)$

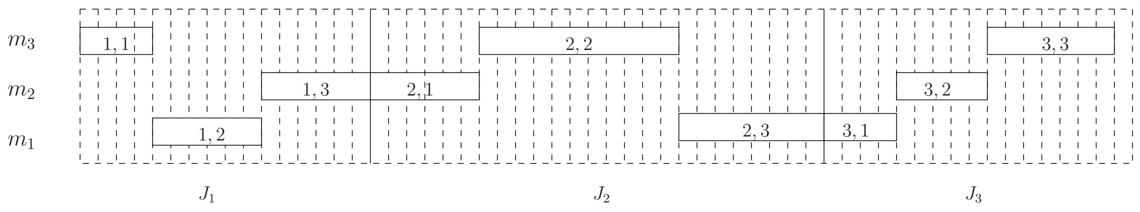

Figure 1: A canonical schedule for the example of Table 1

2.3 Graph representation

2.3.1 Disjunctive activity-on-node graph and operation time windows

The classical representation of the JSP is usually based on disjunctive activity-on-node (AON) graphs [9, 19]. This representation can be extended easily to the JSPTL [5]. The nodes are associated with operations and the arcs are partitioned into two sets: the set of conjunctive (oriented) arcs and the set of disjunctive (non oriented) edges. The conjunctive arcs represent the time lag constraints (4) and (5). For each job i and each operation $j < n_i$, there is an arc from (i, j) to $(i, j + 1)$ valuated by $p_{i,j} + TL^{\min}_{i,j,j+1}$ and an arc from $(i, j + 1)$ to (i, j) valuated by $-(p_{i,j} + TL^{\max}_{i,j,j+1})$. Dummy operations with zero duration, 0 and $*$, are introduced to represent the start and end of the schedule respectively. Operation 0 is a predecessor of the first operation of each job through a conjunctive arc valuated by 0. Operation $*$ is a successor of the last operation (i, n_i) of each job through a conjunctive arc valuated by p_{i,n_i}. Restricting to these conjunctive arcs, the longest path between node 0 and operation (i, j) gives the earliest start time of the operation $\underline{st}_{i,j}$. The earliest completion time is then $\underline{ft}_{i,j} = \underline{st}_{i,j} + p_{i,j}$. Given an upper bound UB on the makespan, the conjunctive graph can also be used to compute a latest start time $\overline{st}_{i,j}$ set to UB minus the longest path between (i, j) and $*$. The latest completion time is then $\overline{ft}_{i,j} = \overline{st}_{i,j} + p_{i,j}$. Hence for each operation, start time windows $[\underline{st}_{i,j}, \overline{st}_{i,j}]$ and end time windows $[\underline{ft}_{i,j}, \overline{ft}_{i,j}]$ are obtained.

The disjunctive edges represent resource sharing constraints. Namely, there is a disjunctive edge between two distinct operations (i, j) and (k, l) as soon as $m_{i,j} = m_{k,l}$. A (complete) selection is an arbitrary (complete) orientation of the disjunctive edges such that each obtained arc is valuated by the duration of the origin operation. To represent a solution, the resulting graph must be free of positive length cycles. A selection satisfying these constraints is called a feasible selection. In a feasible selection, the length of the longest path from 0 to (i, j) represents a resource- and precedence-feasible earliest start time. The length of the longest path from 0 to $*$ gives the makespan of the represented solution. With these definitions, the JSPTL can be stated as the problem of finding a complete feasible selection of minimum makespan.

The disjunctive AON graph linked to Table 1 is depicted in Fig. 2. Dotted edges are the disjunctive edges while plain arcs are conjunctive arcs.

2.3.2 Disjunctive time-bound-on-node graph

To represent a broader variety of time constraints in scheduling problems, the disjunctive time-bound-on-node (TBON) graph has been introduced [11]. The disjunctive TBON representation is equivalent to the disjunctive AON graph, but we choose to use it in the

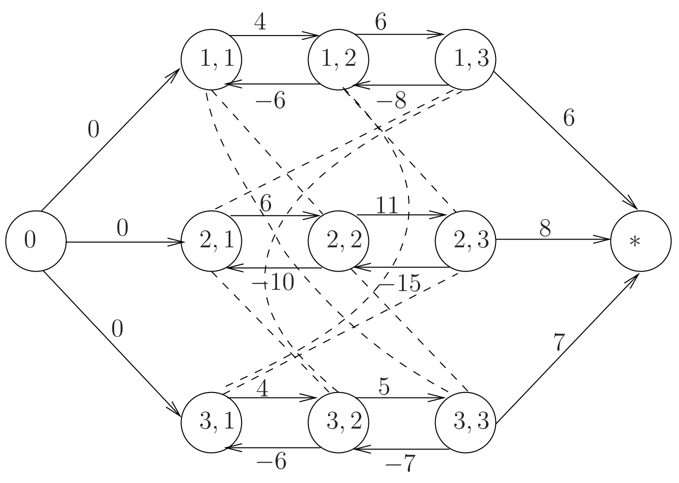

Figure 2: AON representation for the instance of Table 1

sequel since it allows the distinct visualization of the different components of the problem: duration time lags, start and finish times, although it yields a larger number of nodes.

In this representation, each operation is associated with two nodes representing its start- and finish- time. As for the disjunctive AON graph, there are conjunctive arcs and disjunctive edges. An additional node x_0 is introduced to represent the beginning of the scheduling time. As for the AON graph, a node $*$ can also be added to represent the end of the schedule.

A conjunctive arc linking two nodes x_u and x_v is labeled by the value δ_{x_u,x_v} linked to the minimum distance between the time bound variables: $x_u - x_v \geq \delta_{x_u,x_v}$. The disjunctive edges of the AON graph between (i,j) and (k,l) existing when $m_{i,j} = m_{k,l}$ are replaced for the TBON by a pair of exclusive conjunctive arcs: a conjunctive arc from $ft_{i,j}$ to $st_{k,l}$ and a conjunctive arc from $ft_{k,l}$ to $st_{i,j}$ both valuated by 0. For the JSPTL, the TBON graph conjunctive arcs can be built according to Table 2. A complete selection consists in choosing a single conjunctive arc for each exclusive pair. In a feasible selection, the longest path from each start node (respectively finish node) of each operation to node x_0 represents the latest start- (respectively finish-) time of operations. The makespan is the longest path from node x_0 to node $*$.

Table 2: Arc values for the JSPTL TBON graph

x_u	x_v	δ_{x_u,x_v}	condition
$st_{i,j}$	$ft_{i,j}$	$p_{i,j}$	−
$ft_{i,j}$	$st_{i,j}$	$-p_{i,j}$	−
$ft_{i,j}$	$st_{i,j+1}$	$TL^{\min}_{i,j,j+1}$	$j < n_i$
$st_{i,j+1}$	$ft_{i,j}$	$-TL^{\max}_{i,j,j+1}$	$j < n_i$
x_0	$st_{i,j}$	0	−
$ft_{i,j}$	$*$	0	−
$ft_{i,j}$	$st_{k,l}$	0	$m_{i,j} = m_{k,l}$

Fig. 3 represents the disjunctive TBON graph of the example presented in Table 1.

To simplify the representation, only the disjunctions on machine m_1, which correspond to three sets of disjunctive constraints, are represented in the figure (dotted lines).

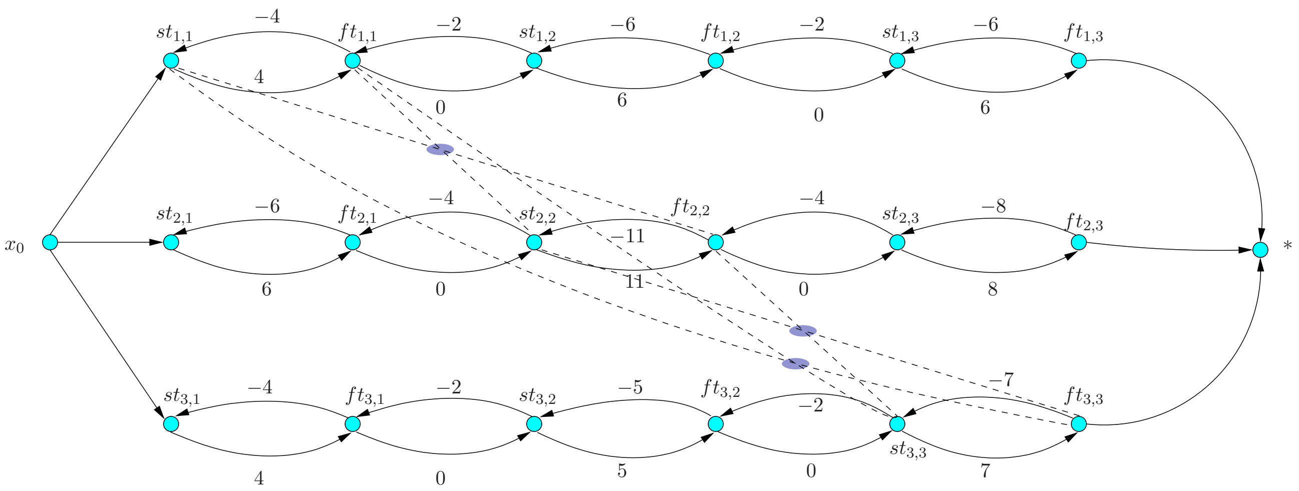

Figure 3: TBON representation for the instance of Table 1

3 Generalized constraint propagation

Constraint propagation amounts to reduce the decision variable domains and/or to explicitly generate induced constraints, depending on the constraint propagation level, by means of an active use of constraints. In this section, the objective function is not explicitly considered. Instead, a tentative upper bound of the makespan UB is se.t The purpose of the constraint propagation mechanism is to prove satisfiability of $C_{\max} \leq UB$. Since this is an NP-complete decision problem, this mechanism is not complete, *i.e.*, when an inconsistency arises during constraint propagation, the problem is unsatisfiable; otherwise, the problem is not proved to be satisfiable in the other case. We will present in Section 4 how we integrate this mechanism into a makespan minimization method for the JSPTL.

We distinguish between time constraint propagation, working on a set of *conjunctive temporal constraints* such as the conjunctive part of the TBON graph (see Section 2.3.2) or Simple Temporal Network in the Artificial Intelligence field [7], and resource constraint propagation tackling the disjunctive constraints or *disjunctive temporal problem* [21].

3.1 Time constraint propagation

To check the global consistency of a *conjunctive temporal problem* one might use filtering techniques such as arc-consistency (AC) or path-consistency (PC) that both run in polynomial time [6]. PC computes any binary constraint between each couple of time points by intersecting it with all paths going through a third time point. AC is a more restricted case of PC since it only updates the domain of each time point. For a conjunctive set of time constraints, the advantage of PC is that it computes the *complete minimal graph* of time constraints: for any two time points, PC provides the interval containing the values that are consistent with other constraints. An inconsistency arises when a positive length cycle is detected in the graph (either by AC or by PC). PC computes in the TBON graph the longest paths length between each operation time points x_u and x_v, denoted by a_{x_u,x_v} in $O(N^3)$, where N is the number of operations. For each conjunctive arc, a_{x_u,x_v} is initialized with δ_{x_u,x_v} and the longest paths are computed progressively by maintaining the relation $a_{x_u,x_w} \geq a_{x_u,x_v} + a_{x_v,x_w}$, for all u, v, w, such as in the Floyd-Warshall algorithm.

Several variants of PC algorithms exist. In this paper, we consider an incremental version of path-consistency called IFPC (Incremental Full Path Consistency) which efficiency was shown on the JSP [18]. Basically, since our solving method (see Section 4) adds progressively precedence constraints to build a complete selection, the IFPC algorithm reaches an $O(N^2)$ complexity for each added precedence constraint.

3.2 Generalized disjunctive constraint propagation

Standard disjunctive constraint propagation rules (see *e.g.*, [1]), are based on operation time windows. We propose in this section a generalization of these rules to incorporate the all-pairs longest path lengths a_{x_u,x_v}.

Propagation based on disjunctive pairs

Based on IFPC algorithm for time constraints, a generalization of the simplest disjunctive constraint propagation rule, called Forbidden Precedence (FP), has previously been proposed in [2, 8, 12]. We recall here this generalization. FP propagation considers the disjunctive constraint $[(i,j) \prec (k,l) \vee (i,j) \succ (k,l)]$. The principle of FP consists in adding to the set of conjunctive constraints, one of the precedence constraints involved in the disjunction; if it leads to an inconsistency then this precedence is forbidden (therefore the reverse is mandatory). It yields:

$$\forall \mu \in M, \forall (i,j),(k,l) \in T_\mu, \text{ if } \overline{ft}_{k,l} - \underline{st}_{i,j} < p_{i,j} + p_{k,l} \text{ then } (i,j) \not\prec (k,l). \tag{9}$$

The generalization of FP, denoted by GFP, is based on the value of the binary constraints deduced by IFPC between the two variables $st_{k,l}$ and $ft_{i,j}$:

$$\forall (i,j),\ (k,l) \in T_\mu,\ \text{ if } a_{st_{k,l},ft_{i,j}} > 0 \text{ then } (i,j) \not\prec (k,l) \tag{10}$$

where $a_{st_{k,l},ft_{i,j}}$ is the length of the longest path from $st_{k,l}$ to $ft_{i,j}$.

Since by definition, $a_{st_{k,l},ft_{i,j}}$ verifies $a_{st_{k,l},ft_{i,j}} \geq p_{k,l} - \overline{ft}_{k,l} + \underline{st}_{i,j} + p_{i,j}$, it can be proved (see Fig. 4) that the generalization dominates the classical formulation with the same complexity (it explores the same set of disjunctions).

Propagation based on disjunctive sets of operations

We generalize two other disjunctive constraint propagation rules, named Latest Starting time Last (LSL) and Earliest Finishing time First (EFF) by [4]. Rule LSL aims at sequencing an operation i after a set of operations competing for the same resource. Symmetrically, EFF sequences an operation (i,j) before a set of operations competing for the same resource. The classical formulation of LSL is:

$$\forall \mu \in M, \forall S \subset T_\mu, \forall (i,j) \in T_\mu \backslash S, \text{ if } \max_{(k,l) \in S' \subseteq S} (\overline{ft}_{k,l} - p_{k,l}) < \underline{st}_{i,j} + p_{i,j} \text{ then } S' \prec (i,j). \tag{11}$$

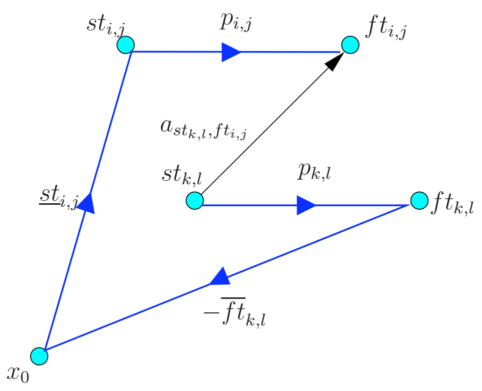

Figure 4: Minimal TBON graph for GFP

LSL can be generalized, using the same principle as FP and GFP: the generalization is based on the length of the longest path from $st_{k,l}$ to $ft_{i,j}$, $\forall (k,l),(i,j) \in T_\mu, \forall \mu \in M$. In the minimal complete graph obtained by IFPC algorithm it corresponds to $a_{st_{k,l},ft_{i,j}}$. The new formulation is then:

$$\forall \mu \in M, \forall S \subset T_\mu, \forall (i,j) \in T_\mu \backslash S, \text{ if } \min_{(k,l)\in S' \subseteq S} a_{st_{k,l},ft_{i,j}} > 0 \text{ then } S' \prec (i,j). \tag{12}$$

This generalization of LSL is denoted by GLSL. Symmetrically, the generalization GEFF can be obtained for EFF. The algorithm for applying GLSL has the same complexity as those for LSL (it explores the same set of operations).

The demonstration of this result follows the same principle as for the generalization of FP rule: $a_{st_{k,l},ft_{i,j}}$ deduced by IFPC is the length of the longest path from $st_{k,l}$ to $ft_{i,j}$, $\forall (k,l) \in S'$, *i.e.*, $a_{st_{k,l},ft_{i,j}} \geq p_{k,l} - \overline{ft}_{k,l} + \underline{st}_{i,j} + p_{i,j} \forall (k,l) \in S'$ (see Fig. 5).

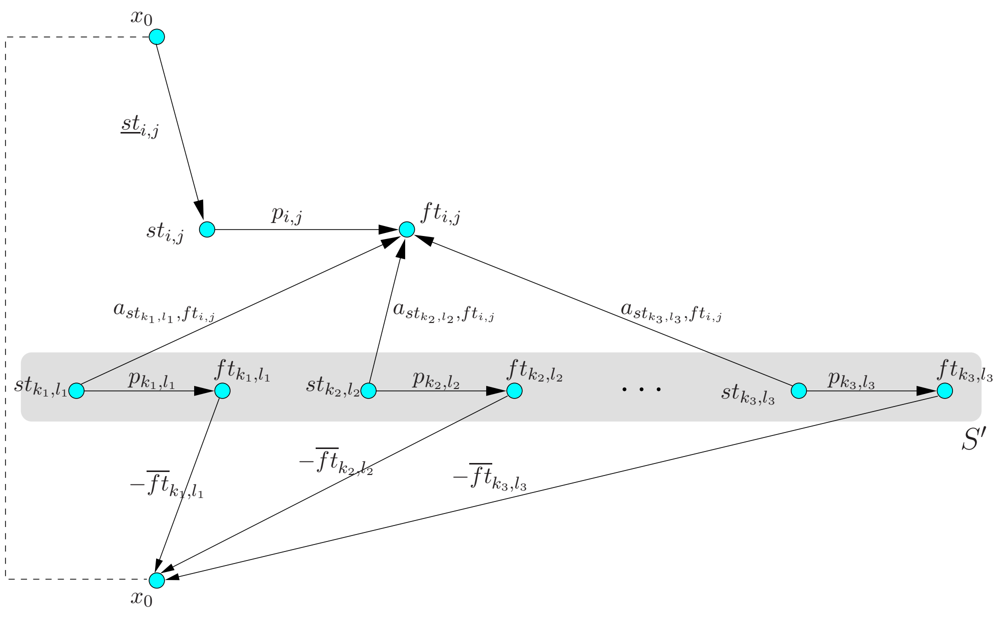

Figure 5: Minimal TBON graph for GLSL

Propagation based on energetic reasoning

We propose a new generalization of the forbidden precedence with energetic reasoning (FPE). FPE also considers a set of operations competing for the same resource, but it also involves the minimum consumption of operations over given intervals $[t_1, t_2]$. For a given machine and a given operation (k, l) over an interval $[t_1, t_2]$, the classical formulation of the minimal consumption is [10, 16]:

$$\underline{w}((k,l), t_1, t_2) = \max[0, \min(p_{k,l}, t_2 - t_1, \underline{ft}_{k,l} - t_1, t_2 - \overline{st}_{k,l})].$$

Rule FPE [16] considers the disjunctive constraint between two operations (i, j) and (m, n) and computes the minimal consumption of all other operations (k, l) using the same resource over the interval bounded by earliest start time of (i, j) and latest finish time of (m, n):

$\forall \mu \in M, \forall (i,j), (m,n) \in T_\mu, T^*_\mu = T_\mu \setminus \{(i,j), (m,n)\},$

$$\text{if } \overline{ft}_{m,n} - \underline{st}_{i,j} < p_{i,j} + p_{m,n} + \sum_{(k,l) \in T^*_\mu} \underline{w}((k,l), \underline{st}_{i,j}, \overline{ft}_{m,n}) \text{ then } (i,j) \not\prec (m,n). \quad (13)$$

The generalized formulation of the minimal consumption of operation (k, l) can be obtained by using the longest paths computed by IFPC (see Fig. 6):

$$\underline{w}^{\text{ext}}((k,l), st_{i,j}, ft_{m,n}) = \max[0, \min(p_{k,l}, a_{st_{i,j},ft_{m,n}}, a_{st_{i,j},ft_{k,l}}, a_{st_{k,l},ft_{m,n}})]. \quad (14)$$

The two first terms $p_{k,l}$ and $a_{st_{i,j},ft_{m,n}}$ correspond to the trivial cases, when operation (k, l) if fully processed inside the interval $[st_{i,j}, ft_{m,n}]$, and when it covers the interval, respectively. Indeed $a_{st_{i,j},ft_{m,n}}$ represents a lower bound of the length of interval $[st_{i,j}, ft_{m,n}]$. The third term considers the case where (k, l) is left-shifted for which (k, l) overlaps $[st_{i,j}, ft_{m,n}]$ by $a_{st_{i,j},ft_{k,l}}$. Similar reasoning leads to the expression of the last term, when the operation is right-shifted.

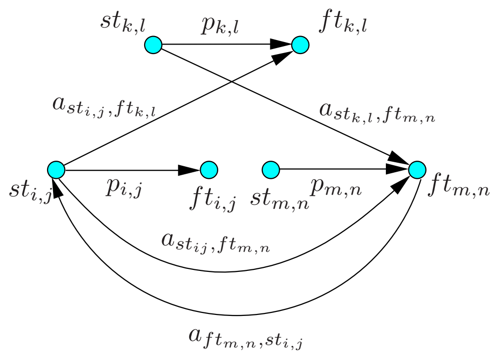

Figure 6: Minimal graph for minimal consumption

The generalized forbidden precedence with energetic reasoning (GFPE) is then based on the following extension of the minimal consumption:

$\forall \mu \in M, \forall (i,j), (m,n) \in T_\mu, T^*_\mu = T_\mu \setminus \{(i,j), (m,n)\},$

$$\text{if } -a_{ft_{m,n},st_{i,j}} < p_{i,j} + p_{m,n} + \sum_{(k,l) \in T^*_\mu} \underline{w}^{\text{ext}}((k,l), st_{i,j}, ft_{m,n}) \text{ then } (i,j) \not\prec (m,n), \quad (15)$$

where $-a_{ft_{m,n},st_{i,j}}$ is the maximal length of interval $[st_{i,j}, ft_{m,n}]$.

Unfortunately, there is no dominance between FPE and GFPE as it can be illustrated in the following example.

Let us consider four operations (i, j), (m, n), (k, l), and (u, v) competing for the same machine. The TBON graph of Fig. 7 synthesizes the useful data.

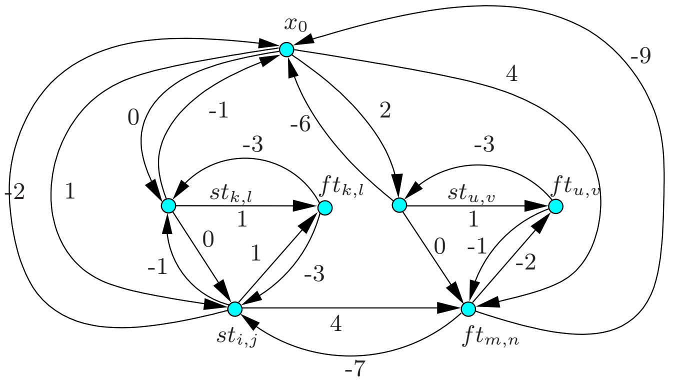

Figure 7: Example for energetic-based rules

Let us consider the disjunctive constraint between (i, j) and (m, n). The minimal consumptions of operations (k, l) and (u, v) over the interval $[\underline{st}_{i,j}, \overline{ft}_{m,n}]$ are then:

- $\underline{w}((k, l), \underline{st}_{i,j}, \overline{ft}_{m,n}) = \max[0, \min(1, 9-1, 1-1, 9-1)] = 0$,
- $\underline{w}((u, v), \underline{st}_{i,j}, \overline{ft}_{m,n}) = \max[0, \min(1, 9-1, 3-1, 9-6)] = 1$.

For the same operations, the generalized minimal consumptions are:

- $\underline{w}^{\text{ext}}((k, l), st_{i,j}, ft_{m,n}) = \max[0, \min(1, 4, 1, 4)] = 1$,
- $\underline{w}^{\text{ext}}(u, v), st_{i,j}, ft_{m,n}) = \max[0, \min(1, 4, 2, 0)] = 0$.

Suppose that minimal durations of operations (i, j) and (m, n) are $p_{i,j} = 4$ and $p_{m,n} = 3$, rule FPE (13) tests the following condition: $9 - 1 < 4 + 3 + 0 + 1$ which is not verified then it allows no deduction. However, rule GFPE (15) provides us with the test $7 < 4 + 3 + 1 + 0$ which holds and implies $(i, j) \not\prec (m, n)$. In this example, rule GFPE dominates rule FPE. However, if on the same example, we change the value of the arc from x_0 to $st_{k,l}$ by interval $[1, 2]$ and the value of the arc $st_{i,j}$ from $ft_{k,l}$ by interval $[0, 3]$, the classical minimal consumptions are then:

- $\underline{w}((k, l), \underline{st}_{i,j}, \overline{ft}_{m,n}) = \max[0, \min(1, 9-1, 2-1, 9-2)] = 1$,
- $\underline{w}(u, v), \underline{st}_{i,j}, \overline{ft}_{m,n}) = \max[0, \min(1, 9-1, 3-1, 9-6)] = 1$,

and the value of generalized minimal consumptions are:

- $\underline{w}^{\text{ext}}((k, l), st_{i,j}, ft_{m,n}) = \max[0, \min(1, 4, 0, 4)] = 0$,
- $\underline{w}^{\text{ext}}(u, v), st_{i,j}, ft_{m,n}) = \max[0, \min(1, 4, 2, 0)] = 0$.

The application of rule FPE produces the test $9 - 1 < 4 + 3 + 1 + 1$ which holds and then it is deduced that $(i, j) \not\prec (m, n)$. Conversely, with rule GFPE, the test $7 < 4 + 3 + 0 + 0$ does not permit any deduction. On this second example, rule FPE subsumes rule GFPE.

In this section, we showed the theoretical contribution of the generalized propagation rules GFP, GEFF, and GLSL. Nevertheless, an experimental validation is needed to prove their practical contribution. Indeed, the generalization is based on an IFPC algorithm which is more time-consuming than an AC-like algorithm (for instance a Bellman-Ford algorithm). Moreover, we showed in the previous example that there is no dominance between rules FPE and GFPE.

We must also recall that, for solving disjunctive scheduling problems, Edge Finding (denoted here by EdFi) [1] is generally considered as an efficient propagation rule. Note that EdFi produces the same conclusions as LSL or EFF. However, at present time, EdFi does not seem to be generalizable, as EFF and LSL do. Our, objective is then to evaluate experimentally to what extend the generalized rules can also be efficiently triggered.

4 Proposed solving method

In order to evaluate the contribution of generalized resource constraint propagation, the proposed formulations are included in a basic branch-and bound procedure presented in this section. We also propose a job insertion heuristic to obtain an initial upper bound.

4.1 A job insertion heuristic

4.1.1 Issues for greedy heuristics in the presence of maximum time lags

For the JSP, priority-rule based greedy heuristics (list algorithms) obtain very quickly an approximate solution. These heuristics consider a set of candidate operations to be scheduled (*i.e.*, the operations are ready and the machine on which they are to be performed is free). The operations are ordered considering a (static or dynamic) priority rule and scheduled so as to build a partial feasible schedule. More precisely, each operation selected with the priority rule is inserted in the partial sequence on the machine it requires without changing this partial sequence.

To solve the JSPTL, the implementation of greedy heuristics has to cope with another difficulty. Indeed, proceeding as described above may lead to an unsatisfiable solution because we can easily have cases where all insertion positions for the selected operation violate the maximum time lag constraints.

4.1.2 Principle of the proposed heuristic

Actually, for general time-lag problems (like RCPSPs with time lags), answering the question whether there exists a feasible solution is itself an NP-complete problem. As already mentioned, however, there are trivial "canonical" solutions for the JSPTL, in which jobs are sequenced consecutively.

Aware of this property, Caumond *et al.* [5] have designed a list heuristic based associated with a method for repairing partial solutions already built. The list heuristic works by selecting with a priority rule at each step an operation among the set of candidate operations (whose predecessor has already been scheduled) and the operation is appended at the end of the partial schedule. The repairing mechanism takes place when no candidate operation can be appended without violating a time lag constraint. In the worst case, this heuristic generates the canonical schedule.

We propose here a new heuristic for the JSPTL. Our idea is to take advantage of the fact that the time lag constraints occur only inside the job. Our heuristic builds a list of jobs and takes each job consecutively according to this list. At each step, the operations of the current job are all inserted in the partial schedule according to the increasing operation number.

For instance, considering again the example of Table 1, if job J_1 is selected, the heuristic schedules the operations $(1,1)$ then $(1,2)$ and then $(1,3)$ on their respective machines while checking time-lag constraints. Once a job has been scheduled, it is considered as fixed and the start times of its operations cannot be changed.

The complete schedule of the set of operations of a job leads to idle-time intervals on the machine. During the schedule of operations of the next job, our heuristic tries to schedule these operations in the idle-time intervals (insertion positions) previously created. Hence we define our heuristic as a job insertion procedure. Such heuristics were previously proposed for job-shop problems without maximum time lags [14, 23]. For the JSPTL, the situation where no insertion position exists for the current operation also occurs for the job insertion heuristic and it can be necessary to make several attempts to schedule the operations of the current job, as shown by the following example. Let us consider that our heuristic has already scheduled jobs J_1 and J_2 (see Fig. 8), it has now to insert job J_3. The idle time interval on m_1 allows scheduling operation $(3,1)$ at its earliest date on m_1.

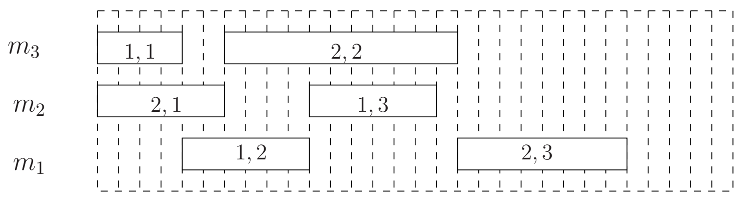

Figure 8: Insertion of Jobs 1 and 2

However, the schedule of $(3,2)$ on m_2 cannot be done, considering simultaneously its processing time of 5, the idle times when m_2 is free, and satisfying the maximum time-lag constraint of 2 between $(3,1)$ and $(3,2)$ (situation displayed in Fig. 9(a) with the representation in black of two occurrences of $(3,2)$, before and after operation $(1,3)$). Job 3 insertion can be canceled and the next insertion position for $(3,1)$ is tested, and the process is carried out until the job can be inserted (see Fig. 9(b)).

As in the method given in [5], in the worst case, our heuristic schedules the jobs according to the canonical schedule.

4.1.3 Algorithm

Our heuristic is based on the insertion of each job one by one in the schedule. The following static orders of jobs considered for building input jobs lists are:

- lexicographical and anti-lexicographical orders;

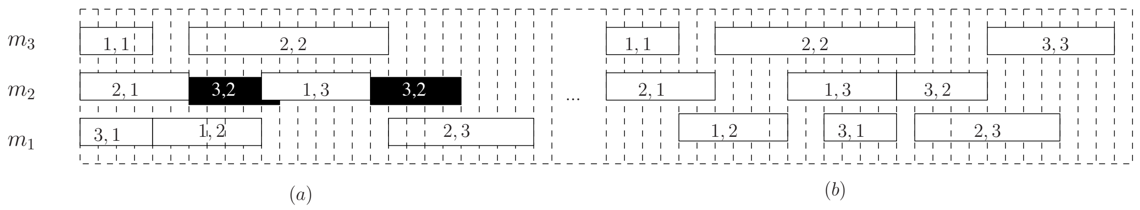

Figure 9: Insertion of Job 3

- ascending and descending orders of $TL^{\max}_{i,j,j+1}, \forall J_i \in J$;
- ascending and descending orders of $\sum_{j=1}^{n_i} p_{i,j}, \forall J_i \in J$;
- ascending and descending orders of $\sum_{j=1}^{n_i} TL^{\max}_{i,j,j+1} + \sum_{j=1}^{n_i} p_{i,j}, \forall J_i \in J$.

Once entirely scheduled, a job u is fixed, meaning that each of its operations (u, v) have a fixed start and finish times $ST_{u,v}$ and $FT_{u,v}$. We let however a full degree of freedom to the operations of the job being currently inserted.

For such a job i, suppose operations $(i, 1), \ldots, (i, j-1)$ have been inserted in intervals $[\underline{I}_1, \overline{I}_1], \ldots, [\underline{I}_{j-1}, \overline{I}_{j-1}]$ and that operation (i, j) is tentatively inserted in some interval $[\underline{I}_{j-1}, \overline{I}_{j-1}]$. Each $\underline{I}_q$ is fixed and equal to 0 or to some $FT_{u,v}$ and each $\underline{I}_q$ is also fixed and equal to ∞ or to some $ST_{u,v}$. Hence the constraints for (i, j) insertion can be represented by the insertion TBON graph displayed in Figure 10.

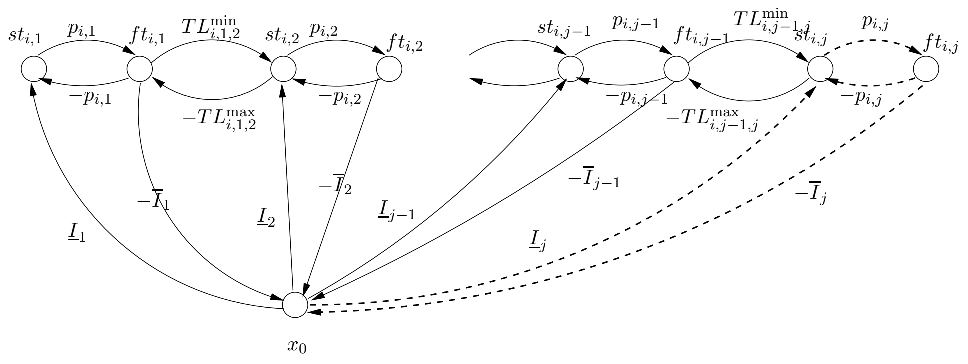

Figure 10: Insertion graph

Proposition 1. *Insertion of (i, j) in $[\underline{I}_j, \overline{I}_j]$ is feasible, if and only if*

$$\begin{aligned} \overline{I}_j - \underline{I}_j &\geq p_{i,j}, & (16)\\ \overline{I}_j - \underline{ft}^{j-1}_{i,j-1} &\geq TL^{\min}_{i,j-1,j} + p_{i,j}, & (17)\\ \underline{I}_j - \overline{ft}^{j-1}_{i,j-1} &\leq TL^{\max}_{i,j-1,j}, & (18) \end{aligned}$$

where $\underline{ft}^{j-1}_{i,j-1}$ is the length of the longest path from x_0 to $ft_{i,j-1}$ and $-\overline{ft}^{j-1}_{i,j-1}$ is the length of the longest path from $ft_{i,j-1}$ to x_0 in the insertion graph resulting from $(i, j-1)$ insertion.

Proof. The insertion of (i, j) in $[\underline{I}_j, \overline{I}_j]$ is feasible if and only if it does not generate any positive length elementary cycle in the insertion graph. Since the new nodes $st_{i,j}$ and

$ft_{i,j}$ are connected to the insertion graph only via nodes x_0 and $ft_{i,j-1}$, any created cycle of positive length traverses either x_0 or $ft_{i,j-1}$, or both. There is only one elementary cycle traversing only x_0: $(x_0, st_{i,j}, ft_{i,j}, x_0)$ of length $\underline{I}_j + p_{i,j} - \overline{I}_j$ which yields condition (16). There is one elementary cycle traversing only $ft_{i,j-1}$: $(ft_{i,j-1}, st_{i,j}, ft_{i,j-1})$ of length $TL^{\min}_{i,j-1,j} - TL^{\max}_{i,j-1,j} \leq 0$ unless the problem is trivially unfeasible. The remaining cycles traverse both nodes x_0 and $ft_{i,j-1}$ and the largest one takes either the longest path from x_0 to $ft_{i,j-1}$ of length $\underline{ft}^{j-1}_{i,j-1}$, which yields condition (17) or the longest path from $ft_{i,j-1}$ to x_0, of length $-\overline{ft}^{j-1}_{i,j-1}$, with yields condition (18). $\square$

Once (i,j) is inserted in $[\underline{I}_j, \overline{I}_j]$, we need to compute values $\underline{ft}^{j}_{i,j}$ and $\overline{ft}^{j}_{i,j-1}$ for insertion of $(i, j+1)$. The insertion graph displayed in Figure 10 shows that the longest path from x_0 to $ft_{i,j}$ is either path $(x_0,\ st_{i,j},\ ft_{i,j})$ or a path issued from $ft_{i,j-1}$ and traversing only arcs from the previous insertion graph between x_0 and $ft_{i,j-1}$. Symmetrically, the longest path from $ft_{i,j}$ to 0 is either reduced to arc $(ft_{i,j}, x_0)$ or going to $ft_{i,j-1}$ and traversing only arcs from the previous insertion graph between $ft_{i,j-1}$ and x_0. This yields the following $O(1)$ update of the (i,j) time window:

$$\begin{aligned} \underline{ft}^{j}_{i,j} &= \max(\underline{I}_j + p_{i,j}, \underline{ft}^{j-1}_{i,j-1} + TL^{\min}_{i,j-1,j} + p_{i,j}), && (19)\\ \overline{ft}^{j}_{i,j} &= \min(\overline{I}_j, \overline{ft}^{j-1}_{i,j} + TL^{\max}_{i,j-1,j} + p_{i,j}). && (20) \end{aligned}$$

The job insertion algorithm is displayed in Algorithm 1. For each job according to list $\mathcal{L}$, the heuristic scans the operations. Each time a feasible interval (indexed by q_j) is encountered on $m_{i,j}$ for (i,j), the algorithm tries to insert $(i, j+1)$ on $m_{i,j+1}$. If no feasible interval is found for (i,j) then the algorithm scans again the possible intervals for $(i, j-1)$ starting with the interval indexed by $(q_{j-1}+1)$, and so on.

Note the worst-case time complexity of the procedure for inserting a single job is $O(|J|^m)$, which is exponential in the number of machines. However, for some instances the number of machines may be small. To reduce the CPU time, the number of tested insertion positions may be restricted arbitrarily before reaching the canonical position for each job. However, in our experiments it has never been necessary.

4.2 Branch-and-Bound procedure

4.2.1 Binary search

The proposed Branch and Bound (B&B) procedure has been designed to check whether a solution having a makespan lower than or equal to a given value exists or not. This B&B is integrated into a binary search procedure which iterates on makespan values. The principle is as follows:

1. Compute an initial lower bound LB (by applying the selected constraint propagation rules) and an initial upper bound UB (with the proposed job insertion heuristic).

2. Repeat:

   (a) Set $H \leftarrow (LB + UB)/2$;

   (b) Determine a solution of the JSPTL with makespan lower than or equal to H or prove that there is no solution;

Algorithm 1: Job insertion heuristic

```
for each job $i \in \mathcal{L}$ do
    $j \leftarrow 1$; inserted $\leftarrow$ false;
    $q \leftarrow 1$ (first interval index on $m_{i,j}$);
    while not inserted do
        for each possible interval on $m_{i,j}$ starting from $q$ do
            if Conditions (16-18) are verified then
                store in $q_j$ the interval index on $m_{i,j}$;
                compute $\underline{ft}^j_{i,j}$ and $\overline{ft}^j_{i,j}$ according to expressions (19) and (20);
                inserted $\leftarrow$ true; break;
            end
        end
        if inserted=false then
            set $j \leftarrow j-1$; $q \leftarrow q_j + 1$;
        else
            set $j \leftarrow j+1$; $q \leftarrow 1$;
            inserted $\leftarrow$ false;
        end
    end
    Fix start times for job $i$ and update intervals for job $i+1$;
end
```

(c) If a solution is obtained, then set UB to the obtained makespan, else set $LB \leftarrow H + 1$;

3. Until $LB = UB$.

For Step 2(b), a basic B&B method is used whose goal is to compare the pruning power of the various studied constraint propagation rules under a common tree search scheme. This B&B is limited by a maximum CPU time. Its main components are described in the following sections.

4.2.2 Branching scheme

The adopted branching scheme has been previously proposed in [22]. At each node, we consider the machine with the maximal load and, for this machine, we consider the list of the conflicting operations not yet scheduled. This branching scheme produces one node for each partial schedule of one operation before all the others. For instance, at a given node, let consider $L = \{a, b, c, d\}$ a list of conflicting operations for the most loaded machine, the branching scheme produces four nodes corresponding to constraints $a \prec \{b, c, d\}$, $b \prec \{a, c, d\}$, $c \prec \{a, b, d\}$ and $d \prec \{a, b, c\}$.

4.2.3 Constraint propagation

At each node of the B&B, constraint propagation rules are applied to check the consistency of the subproblem. If an inconsistency arises, the node cannot lead to a solution and it is then fathomed. At each node, the constraint propagation algorithm (see Algorithm

2) starts with IFPC algorithm and follows with resource constraint propagation, until a fixed point is reached or as soon as an inconsistency is detected. In this algorithm, X is the set of decision variables and C is the set of conjunctive time constraints.

Algorithm 2: Propagation algorithm

```
$C'$: the new constraints at the considered node;
IFPC($X$, $C$, $C'$);
flag ← true; **while** *flag* **do**
    flag ← false; **for** $m \in M$ **do**
        Resource Constraint Propagation(*Rules, X, C, New_ C, flag*);
        /* where Rules is a set of constraint propagation rules and
        New_C are new time constraints produced by the propagation
        process */
    **end**
    **if** *flag* **then**
        IFPC($X$, $C$, *New_ C*);
    **end**
**end**
```

For each node, a new set of constraints, named C', has to be taken into account. The first step of the propagation algorithm initializes IFPC algorithm with the precedence constraints C' associated to the node. IFPC algorithm propagates constraints in C' to the rest of the problem (X, C). For each resource, the procedure *Resource_ Constraint_ Propagation(Rules,X,C,New_ C,flag)* propagates the set of propagation rules R into the problem (X, C). These propagations lead to new constraints New_C which will be taken into account by IFPC algorithm at the next iteration of Algorithm 2. This propagation algorithm stops when no more deduction can be done (flag = true). It is designed such that several combinations of rules which propagate resource constraints can be chosen based on those defined in Section 5.1.

5 Computational results

5.1 Experimental setting

All the algorithms were coded in Ada 95 using GNAT 4.4.1 compiler and the tests were performed on a 2.33 GHz computer with 4 GB of RAM running under Linux Red Hat 4.4.1-2.

Instances

Our experiments were conducted on instances presented in [5]. These instances have been obtained from classical JSP benchmarks where time lags have been added. These instances are identified by their name and a minimum (α) and a maximum (β) time-lag coefficients: Name_α_β. Minimum and maximum time-lags are calculated for each job i with these coefficients, the duration of the job, and the number of operations of this job n_i by $TL^{\min}_{i,j,j+1} \leftarrow \alpha \times (\sum_{j=1}^{n_i} p_{i,j})/n_i$ and $TL^{\max}_{i,j,j+1} \leftarrow \beta \times (\sum_{j=1}^{n_i} p_{i,j})/n_i$. In the proposed instances, minimum time lags are null ($\alpha = 0$) and note that there is an identical maximum time lag for each operation in the same job. There are two set of instances:

- Set 1: instance ft06, and la01 to la05, with $\beta \in \{0, 0.25, 0.5, 1, 2, 3, 5, 10\}$,

- Set 2: instance la06 to la40, with $\beta \in \{0, 0.5, 1, 2 \text{ or } 3, 10\}$.

In [5], only a subset of these instances were tested:

- Subset 1a: instance ft06, and la01 to la05, with $\beta \in \{0, 0.5, 1, 2\}$,

- Subset 2a: instance la06 to la08, with $\beta \in \{0, 0.5, 1, 2, 10\}$.

Combinations of resource constraint propagation rules

One of the aim of these experiments is to experimentally evaluate the efficiency of the generalized resource constraint propagation rules. Thus, we choose to test the two following combinations to compare the results with or without the generalization for resource constraint propagation:
{FP,EFF,LSL} (1) and
{GFP,GEFF,GLSL} (2).

We test separately the generalization of the energetic rule FPE, as no theoretical dominance could be exhibited. Therefore we compare the three following combinations:
{FP,FPE,EFF,LSL} (5),
{GFP,FPE,GEFF,GLSL} (6), and
{GFP,GFPE,GEFF,GLSL} (7).

We also evaluate the impact of the Edge Finding (EdFi) rule, for which no generalization could be provided, for each of the above combinations:
{FP,EFF,LSL,EdFi} (3),
{GFP,GEFF,GLSL,EdFi} (4),
{FP,FPE,EFF,LSL,EdFi} (8),
{GFP,FPE,GEFF,GLSL,EdFi} (9), and
{GFP,GFPE,GEFF,GLSL,EdFi} (10).

For the experiments, we consider the above-defined 10 combinations of resource constraint propagation rules.

5.2 Global results

Tables from 3 to 6 present the results we obtain on the two sets of instances presented in Section 5.1. We do not present our results for instances from la26 to la40 since our B&B cannot improve the initial bounds on these instances. In these tables, the first column corresponds to instance name (I) and the first row to maximum time-lag coefficients (β). For each instance, we give the bounds obtained for the makespan ($[LB, UB]$), the CPU time associated, and the combination of rules which obtained this result. If $UB = LB$ we only give the optimum, otherwise our algorithm was stopped after a time-out (TO) fixed to 10 minutes (for a B&B iteration). The bounds noted with an asterisk mean that they improved the initial ones.

Table 3 shows that for instance set 1, 42 instances out of 48 are solved with our method. The initial bounds for the makespan are improved on 5 out of the 6 remaining instances.

The table shows that the constraint propagation rule combination (9) is generally the only one to reach and prove optimality. This combination includes all the generalized rules except the energetic reasoning rule, combining with Edge Finding. The table also shows that no-wait instances with a maximum time lag coefficient equal to 0 are much harder to solve with our method.

For instance set 2, the B&B method is only able to solve 6 instances to optimality out of 100 with rather large maximum time lags. For 5 of these instances, combination rule (9) is still the only one to obtain this result. For 26 of the 94 instances not solved to optimality, the method is able to improve the initial bounds. This occurs more often for large maximum time lags than for small maximum time lags.

Table 3: Global results for Set1

I \ β	0	0.25	0.5	1	2	3	5	10
ft06	73* 11 *6-9*	64* 7 *9*	63* 8 *4-6-9-10*	58* 9 *4-6-7-9-10*	55* 5 *1-3-4-5-6-8-9-10*	55* 3 *5-6-8-9*	55* 3 *1-5-6-8-9*	55* 3 all
la01	971* 1877 *9*	819* 682 *9*	758* 436 *9*	683* 89 *9*	666* 80 *8-9*	666* 57 *4-8-10*	666* 55 *8-9*	666* 53 *4-8-9-10*
la02	[859*;1082] 765 TO	[782*;830*] 1004 TO	742* 661 *9*	686* 390 *9*	673* 788 *9*	660* 261 *9*	655* 76 *9*	655* 75 *9*
la03	[805*;834*] 1441 TO	721* 968 *9*	679* 930 *9*	640* 282 *9*	630* 223 *9*	617* 122 *9*	617* 166 *9*	598* 104 *9*
la04	[537;1027] 507 TO	760* 1464 *9*	703* 724 *9*	646* 373 *9*	619* 61 *9*	605* 57 *9*	596* 336 *9*	590* 82 *9*
la05	[715*;836*] 746 TO	[694*;744*] 1185 TO	622* 417 *9*	593* 366 *9*	593* 96 *9*	593* 50 *9*	593* 55 *9*	593* 50 *9*

5.3 Contribution of generalized resource constraint propagation

To confirm the interest of the generalization rules, we scored the different combinations of the two sets of instances as follows: if none of the combinations finds the optimal solution, they obtain the score of 0. When some or all combinations find the optimal solution they are ordered by ascending CPU time. The first combination is given the score of 10, the second is given the score of 9, etc.

Fig. 11 represents the score (Y axis) obtained by the different combinations (X axis) on Sets 1 and 2. It is confirmed in this figure that the best combination of rule is (9), *i.e.*, {GFP,FPE,GEFF,GLSL,EdFi} ahead of the combination (8) *i.e.*, {FP,FPE,EFF,LSL,EdFi} not far the combination (6) *i.e.*, {GFP,FPE,GEFF,GLSL}. We can also note that if we

Table 4: Global results for Set 2 (1/3)

β I	0	0.5	1	2	10
la06	[926;1770] 25 TO	[926;1471] 526 TO	[926;1391] 524 TO	[926;927*] 960 TO	[926;927*] 707 TO
la07	[869;1536] 530 TO	[869;1430] 529 TO	[869;1065*] 754 TO	[869;1205] 521 TO	[869;1123] 518 TO
la08	[863;1640] 528 TO	[863;1454] 529 TO	[863;1052*] 587 TO	**863*** 558 *9*	**863*** 260 *9*

Table 5: Global results for Set 2 (2/3)

β I	0	0.5	1	3	10
la09	[951;1859] 530 TO	[951;1706] 529 TO	[951,1215*] 594 TO	[951,956*] 1764 TO	[951-955*] 752 TO
la10	[958-1666] 526 TO	[958;1530] 524 TO	[958;1147*] 562 TO	**958*** 312 *8*	[958;1151] 517 TO
la11	[1222;2323] 572 TO	[1222;1873] 575 TO	[1222;1796] 565 TO	[1222;1489] 552 TO	[1222;1304*] 721 TO
la12	[1039;2011] 577 TO	[1039;1777] 580 TO	[1039;1543] 570 TO	[1039;1391] 559 TO	[1039;1153] 544 TO
la13	[1150;2219] 583 TO	[1150;1888] 574 TO	[1150;1814] 568 TO	[1150;1240*] 754 TO	[1150;1306*] 939 TO
la14	[1292;2145] 585 TO	[1292;1954] 581 TO	[1292;1598*] 894 TO	**1292*** 1327 *4*	[1292,1301*] 1105 TO
la15	[1207;2276] 573 TO	[1207;1997] 577 TO	[1207;1693] 566 TO	[1207;1268*] 1077 TO	[1207;1499] 543 TO

Table 6: Global results for Set 2 (3/3)

β I	0	0.5	1	3	10
la16	[1285*;1908] 1030 TO	[660;1577] 583 TO	[660;1377] 572 TO	[660;1183] 564 TO	[660;1183] 556 TO
la17	[683;1776] 583 TO	[683;1397] 587 TO	[683;1218] 581 TO	**785*** 907 *9*	**784*** 1131 *9*
la18	[623;2005] 578 TO	[623;1533] 582 TO	[623;1370] 570 TO	[840*;870*] 788 TO	[840*;870*] 799 TO
la19	[685;2066] 579 TO	[685;1551] 579 TO	[685;1030*] 693 TO	[808*;930*] 873 TO	[801*;915*] 779 TO
la20	[744;2300] 582 TO	[744;1627] 584 TO	[744;1146*] 761 TO	[866*;987*] 904 TO	[866*;987*] 706 TO
la21	[935;2748] 907 TO	[935;2213] 905 TO	[935;1870] 901 TO	[935,1555] 780 TO	[935;1452] 717 TO
la22	[830;2740] 934 TO	[830;1677] 883 TO	[830;1581] 816 TO	[830;1540] 771 TO	[830;1312] 714 TO
la23	[1032;2972] 917 TO	[1032;2354] 918 TO	[1032;1894] 842 TO	[1032;1441] 759 TO	[1032;1441] 712 TO
la24	[857;2905] 904 TO	[857;2217] 854 TO	[857;1784] 832 TO	[857;1400] 735 TO	[857;1171*] 1152 TO
la25	[864;2706] 878 TO	[864;1966] 876 TO	[864;1791] 827 TO	[864;1199*] 1164 TO	[864;1142*] 1116 TO

compare the combinations with and without the generalization of rules FP, EFF and LSL, the combination with these generalizations obtain better scores than without.

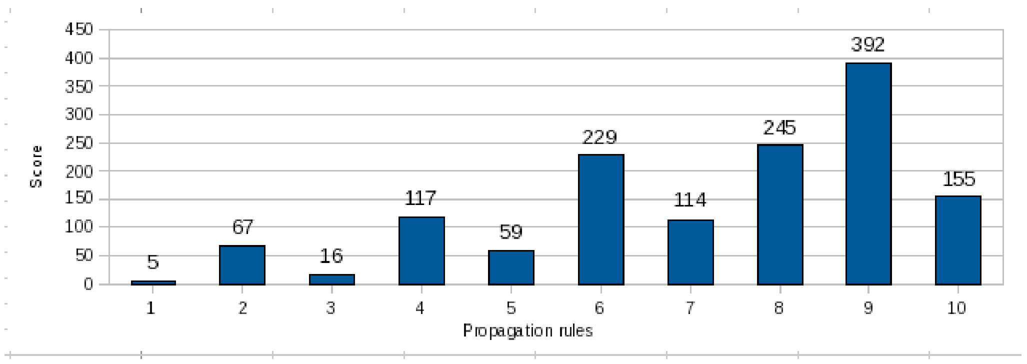

Figure 11: Efficiency of the CP rule combinations on Sets 1 and 2

Fig. 11 also shows that the use of EdFi rule does not cover all the propagations obtained with the generalization of rules FP, EFF, and LSL. Unfortunately, we can note that rule GFPE is not efficient enough as it increases the CPU time.

5.4 Comparison with other methods

In this section, we compare our approach with the memetic algorithm proposed in [5]. The comparisons are carried out only on the instances used in [5] (see Section 5.1): Subset 1a (Table 7) and Subset 2a (Table 8). In each table, the first column gives the instance name and the second one indicates either the optimum value in Table 7 (*Optimum*) or the optimum of the corresponding classical job shop (*i.e.*, without time lags) in Table 8 (*JS-Opt*). This latest optimum of the job shop problem without time lags is obviously a trivial lower bound of the job shop problem with time lags. The three next columns present the lower bound (LB) and the upper bound (UB) obtained with our method, and the corresponding CPU time in seconds. For the memetic algorithm, the results correspond to four runs per instance [5]. The remaining columns present the best results obtained with this algorithm, the average time to obtain the best improvement (Tm) over these runs and the global average time (TTm) of these runs. The results written in bold correspond to the optimum results obtained.

These tables show that, on Subset 1a, we obtain better results than [5], except on the no-wait instances ($\beta = 0$). This is not the case with Subset 2a as shown in Table 8. We only prove the optimum value for the less constrained instance la08_0_10 while Caumond *et al.* prove it for two other instances with a coefficient β equal to 10. Moreover, their upper bounds are better than those found with our method.

5.5 Efficiency of the job heuristic

The proposed heuristic is based on a given order for jobs to be inserted. Several orders (see Section 4.1.3) were compared on the set of instances and the most efficient is the descending order of job durations. It reaches the best makespan value for more than 33% over all the instances (Sets 1 & 2). For the instances under study, since time lags are strongly correlated to job durations, the orders based on time lags are not more efficient

Table 7: Comparison on Subset 1a

Instance	Optimum	LB	UB	CPU	Memetic algorithm	Tm/TTm
ft06_0_0	73	**73**	**73**	11	77	35.5/157.25
ft06_0_0.5	63	**63**	**63**	8	**63**	72/149.75
ft06_0_1	58	**58**	**58**	9	**58**	1/144.25
ft06_0_2	55	**55**	**55**	5	**55**	2.25/133.5
la01_0_0	971	**971**	**971**	1877	**971**	149/280
la01_0_0.5	758	**758**	**758**	436	867	149/278
la01_0_1	683	**683**	**683**	89	723	164/265
la01_0_2	666	**666**	**666**	80	**666**	86/230
la02_0_0	937	859	1082	TO	**937**	149/278
la02_0_0.5	742	**742**	**742**	661	872	80/285
la02_0_1	686	**686**	**686**	390	723	150/267
la02_0_2	673	**673**	**673**	788	683	167/238
la03_0_0	820	805	834	TO	**820**	234/1028
la03_0_0.5	679	**679**	**679**	930	685	211/293
la03_0_1	640	**640**	**640**	282	641	206/278
la03_0_2	630	**630**	**630**	223	648	150/253
la04_0_0	887	537	1207	TO	923	196/285
la04_0_0.5	703	**703**	**703**	724	769	104/273
la04_0_1	646	**646**	**646**	373	662	151/273
la04_0_2	619	**619**	**619**	61	631	83/245
la05_0_0	777	715	836	TO	797	144/256
la05_0_0.5	622	**622**	**622**	417	678	135/260
la05_0_1	593	**593**	**593**	366	615	92/244
la05_0_2	593	**593**	**593**	96	**593**	36/207

Table 8: Comparison on Subset 2a

Instance	JS-Opt	LB	UB	CPU	Memetic algorithm	Tm/TTm
la06_0_0	926	926	1770	525	1392	341/779
la06_0_0.5	926	926	1471	526	1153	545/773
la06_0_1	926	926	1391	524	1101	403/735
la06_0_3	926	926	1391	524	1101	405/739
la06_0_10	926	926	927	707	**926**	14/14
la07_0_0	890	869	1536	530	1329	599/907
la07_0_0.5	890	869	1430	529	1132	573/784
la07_0_1	890	869	1065	754	1009	532/736
la07_0_3	890	869	1079	659	975	477/710
la07_0_10	890	869	1123	518	**890**	39/39
la08_0_0	863	863	1640	528	1385	327/851
la08_0_0.5	863	863	1454	529	1164	470/772
la08_0_1	863	863	1052	587	1013	541/742
la08_0_3	863	863	1052	587	1013	544/746
la08_0_10	863	**863**	**863**	260	**863**	17/17

than orders based on job durations, and the combination of both job durations and time lags does not improve the results. Globally, we obtain about 25% of best solutions for the descending (ascending) order of time lags, from 12% to 14% for lexicographical orders, and between 5% and 7% for ascending orders of time lags and durations.

With our method, the CPU time for solving a given instance of Set 1 is 1.02 s in average. For Set 2 (limited to la06–la25), the time is of 25.17 s. If we refine these results, we obtain that the method needs 1.14 s for Subset 1a and 5.13 s for Subset 2a.

Table 9 presents a comparison between the best variant of our job insertion heuristic (JI) and the operation insertion heuristic (OI) presented by Caumond *et al.* in [5]. This evaluation was limited to Subset 1a because the results obtained by the OI heuristic of [5] are only available for these instances.

For the JI heuristic, the average deviation above optimum over all instances is 33.15% whereas the average deviation of the OI heuristic is 47.63%, which represents a significant gain.

Table 9: Heuristic comparison on Subset 1a

	Optimum	Initial UB	JI heuristic	JI dev	OI heuristic	OI dev
ft06_0_0	73	197	96	31.51	83	13.70
ft06_0_0.5	63	197	72	14.29	109	73.02
ft06_0_1	58	197	72	24.14	58	0.0
ft06_0_2	55	197	70	27.27	55	0.0
la01_0_0	971	2849	1258	29.56	1504	54.89
la01_0_0.5	758	2849	1063	40.24	1474	94.46
la01_0_1	683	2849	928	35.87	1114	63.10
la01_0_2	666	2849	967	45.20	948	42.34
la02_0_0	937	2643	1082	15.47	1416	51.12
la02_0_0.5	742	2643	1011	36.25	1207	62.67
la02_0_1	686	2643	935	36.30	1136	65.60
la02_0_2	673	2643	928	37.89	895	32.99
la03_0_0	820	2383	1081	31.83	1192	45.37
la03_0_0.5	679	2383	930	36.97	1085	59.79
la03_0_1	640	2383	886	38.44	931	47.47
la03_0_2	630	2383	808	28.25	787	24.92
la04_0_0	887	2507	1207	26.61	1346	51.75
la04_0_0.5	703	2507	870	23.76	1156	64.44
la04_0_1	646	2507	1010	43.03	857	32.66
la04_0_2	619	2507	892	44.10	838	35.38
la05_0_0	777	2283	1080	34.62	1224	57.53
la05_0_0.5	622	2283	935	50.32	1208	94.21
la05_0_1	593	2283	814	37.27	964	62.56
la05_0_2	593	2283	749	26.31	683	15.18

In Table 10, we present the average gap above the optimum for both heuristics on subcategories of instances from Subset 1a. Considering the different time lags, we can notice that, for $\beta \in \{0, 0.5, 1\}$, which correspond to the most constrained time lags, we obtain an average gap of at most 35.84% while the OI heuristic obtains an average gap of at least 44.90%. On the contrary, for $\beta = 2$, the OI average gap is only of 25.13% while it reaches of 34.84% for our JI heuristic. This result suggests that our heuristic is better suited for tight time lags than the Caumond *et al.*'s one.

Table 10: Average gap for the two heuristics

Time-lags	JI heuristic	OI heuristic
*_0_0	**28.27** %	45.73 %
*_0_0.5	**33.64** %	74.76 %
*_0_1	**35.84** %	44.90 %
*_0_2	34.84 %	**25.13**%

6 Conclusions and future work

This paper addresses the job shop scheduling problem with minimum and maximum time-lags (JSPTL). We presented generalizations of several standard disjunctive constraint propagation rules based on all-pair longest paths in a conjunctive Time-Bound-On-Node graph and their application for the job shop problem with maximum time lags. To exhibit the contribution of the generalized rules for this problem we proposed a Branch-and-Bound algorithm embedded in a binary search to solve the JSPTL.

The obtained results show that the branch and bound incorporating generalized rules associated with edge-finding solves more instances in a predetermined amount of time than the standard rules, except for the energetic reasoning rule. Moreover, for small instances, our branch-and-bound procedure outperforms a genetic algorithm from the literature.

We also presented a simple heuristic based on job insertion. This heuristic was used to improve the results of our binary search, and compared with a heuristic presented by other authors. We conclude that our heuristic always obtains feasible schedules with a better makespan for tight maximum time lags. Despite a high worst-case complexity, the heuristic performs well in practice with small CPU time requirements.

There are several further research directions. First, it could be helpful to succeed generalizing the Edge-Finding rule. Second, we tested a restricted number of combinations and more experiments could be done to find a more powerful association of propagation rules. Last, the proposed branch-and-bound was designed to validate the new constraint propagation rules, but not to obtain benchmark results on the JSPTL instances. Designing more powerful exact method for the JSPTL is a fruitful research direction. Especially, more efforts have to be carried out to improve the search scheme.

7 Acknowledgment

The authors would like to thank Marion Dugas (MSc INSA Toulouse) for her help on this work.

References

[1] P. Baptiste, C. Le Pape, and W. Nuijten. *Constraint-based scheduling: Applying constraint programming to scheduling problems*. Kluwer Academic Publishers, 2001.

[2] P. Brucker. Scheduling and constraint propagation. *Discrete Applied Mathematics*, 123:227–256, 2002.

[3] P. Brucker, T. Hilbig, and J. Hurink. A branch and bound algorithm for a single machine scheduling with positive and negative time-lags. *Discrete Applied Mathematics*, 94:77–99, 1999.

[4] Y. Caseau and F. Laburthe. Improved CLP scheduling with task intervals. In P. van Hentenryck, editor, *Proceedings of the 11th International Conference on Logic Programming*, pages 369–383, Santa Margherita, Ligure, Italy, 1994. MIT Press.

[5] A. Caumond, P. Lacomme, and N. Tchernev. A memetic algorithm for the job-shop with time-lags. *Computers and Operations Research*, 35(7):2331–2356, 2008.

[6] R. Dechter. *Constraint Processing*. Morgan Kaufmann, San Francisco, May 2003.

[7] R. Dechter, I. Meiri, and J. Pearl. Temporal constraint networks. *Artificial Intelligence*, 49:61–95, 1991.

[8] S. Demassey, C. Artigues, and P. Michelon P. Constraint propagation based cutting planes: an application to the resource-constrained project scheduling problem. *INFORMS Journal on Computing*, 17(1):52–65, 2005.

[9] J. Erschler, G. Fontan, and F. Roubellat. Potentiels sur un graphe non conjonctif et analyse d'un problème d'ordonnancement à moyens limités. *RAIRO-Operations Research*, 13(4):363–378, 1979. [In French].

[10] J. Erschler and P. Lopez. Energy-based approach for task scheduling under time and resources constraints. In *Proceedings of the 2nd International Workshop on Project Management and Scheduling (PMS'90)*, pages 115–121, Compiègne, France, 1990.

[11] P. Esquirol, M.-J. Huguet, and P. Lopez. Modeling and managing disjunctions in scheduling problems. *Journal of Intelligent Manufacturing*, 6:133–144, 1995.

[12] M.-J. Huguet, P. Lopez, and T. Vidal. Dynamic task sequencing in temporal problems with uncertainty. In *AIPS 2002 Workshop on "On-line Planning & Scheduling"*, Toulouse, France, April 2002.

[13] J. Hurink and J. Keuchel. Local search algorithms for a single-machine scheduling problem with positive and negative time-lags. *Discrete Applied Mathematics*, 112(1-3):179–197, 2001.

[14] T. Kis and A. Hertz. A lower bound for the job insertion problem. *Discrete Applied Mathematics*, 128(2-3):395–419, 2003.

[15] R. Kolisch and R. Padman. An integrated survey of project scheduling. *OMEGA International Journal of Management Science*, 29:2001, 1997.

[16] P. Lopez and P. Esquirol. Consistency enforcing in scheduling: A general formulation based on energetic reasoning. In *Proceedings of the 5th International Workshop on Project Management and Scheduling (PMS'96)*, pages 11–13, Poznań, France, 1996.

[17] K. Neumann, C. Schwindt, and J. Zimmermann. *Project scheduling with time windows and scarce resources*. Springer, 2002.

[18] L. R. Planken. Incrementally solving the STP by enforcing partial path consistency. In Ruth Aylett and Ivan Petillot, editors, *Proceedings of the 27th Workshop of the UK Planning and Scheduling Special Interest Group (PlanSIG 2008)*, pages 87–94, December 2008.

[19] B. Roy and B. Sussmann. Les problèmes d'ordonnancement avec contraintes disjonctives. Technical Report D.S. No. 9 bis, SEMA, Paris, 1964. [In French].

[20] M. Salidoa, A. Garridoa, and R. Barták (Eds). Special issue on "Constraint satisfaction techniques for planning and scheduling problems". *Engineering Applications of Artificial Intelligence*, 21(5), 2008.

[21] K. Stergiou and M. Koubarakis. Backtracking algorithms for disjunctions of temporal constraints. *Artificial Intelligence*, 120:81–117, 2000.

[22] P. Torres and P. Lopez. On not-first/not-last conditions in disjunctive scheduling. *European Journal of Operational Research*, 127(2):332–343, 2000.

[23] F. Werner and A. Winkler. Insertion techniques for the heuristic solution of the job shop problem. *Discrete applied mathematics*, 58(2):191–211, 1995.

[24] E.D. Wikum, D.C. Llewellynn, and G.L. Nemhauser. One-machine generalized precedence constrained scheduling problem. *Operations Research Letters*, 16:87–99, 1994.