

# On Not-First/Not-Last Conditions in Disjunctive Scheduling\*

Philippe Torres, Pierre Lopez

Laboratoire d'Analyse et d'Architecture des Systèmes du CNRS  
7 avenue du Colonel Roche, F-31077 Toulouse Cedex 4, France  
e-mails: {ptorres, lopez}@laas.fr

## Abstract

This paper is concerned with the development of constraint propagation techniques for the characterization of feasible solutions in disjunctive scheduling. In disjunctive scheduling, a set of uninterruptible tasks is to be performed on a set of resources. Each task has a release date, a deadline, and a fixed processing time; each resource can handle only one task at a time. Some of these propagation techniques are implemented by rules that deduce either mandatory or forbidden sequences between tasks or sets of tasks. For instance, certain rules indicate whether a given task must or cannot be performed before or after a set of other competing tasks. We focus our attention on the latter problem, known as the “Not-First/Not-Last” problem.

The genericity of propagation rules is a question of major importance. It induces that the result of the overall propagation must not depend on the order in which the inference rules are applied. Hence, one must search for completeness in the time-windows narrowing, in order to ensure the convergence of the propagation towards a unique fix-point.

An efficient algorithm is proposed. It guarantees the completeness of time-windows

---

\*Rapport LAAS 99304, European Journal of Operational Research, Vol.127, pp.332-343, 2000

narrowing due to not-first/not-last conditions. It has been integrated in a branch and bound procedure to solve job-shop instances. It has also been tested within several lower bounding procedures. Computational results are reported and the power and complementarity of not-first/not-last rules with other classical inference rules is discussed.

Keywords: Disjunctive scheduling; Constraint propagation, Edge-finding; Time-bound adjustments; Branch and bound; Job shop; Lower bounds

## 1 Introduction

In this paper, we are concerned with disjunctive scheduling. An instance of the disjunctive scheduling problem consists of a set  $T$  of  $n$  tasks to be performed on one disjunctive resource, which can process only one task at a time. Tasks cannot be interrupted (this corresponds to the non-preemptive case). A release time  $r_i$ , a deadline  $d_i$ , and a processing time  $p_i$  characterize each task  $i$  of  $T$ . The relation  $d_i - p_i \geq r_i$  holds  $\forall i = 1, \dots, n$ . Additional temporal constraints such as precedences due to production routings, time-lags, ... can also be considered.

The scheduling problem under study is seen as a Constraint Satisfaction Problem (CSP) [26]. The question is to find an assignment of the start time  $t_i$  of each task  $i$  within  $[r_i, d_i - p_i]$  that satisfies the set of constraints of the problem, particularly the resource constraint. In the recent past years, constraint-based approaches have been proven to be an efficient way to represent and solve  $\mathcal{NP}$ -hard scheduling problems such as the Job-Shop Scheduling Problem [13, 23, 25, 27]. These approaches use general CSP solving techniques, such as variable and value ordering, dead-end handling, and consistency enforcing.

In our work, we more specifically focus on techniques for enforcing the problem consistency by constraint propagation [14, 15, 24]. An interval of possible start times is assigned to each task by computing extreme bounds for its processing (so-called “heads” and “tails” in [9]). Propagation rules are applied to update (or adjust) these bounds. Thus these rules

deduce new constraints from the current search state so as to characterize as well as possible the feasible solutions, *i.e.*, the schedules that satisfy all the constraints.

Some of these deductions relate to rules that reveal whether (1) a given task must precede another task, or more generally whether (2) a task either (a) must or (b) cannot be performed before or after a set of other competing tasks [2, 10, 13, 20, 23]. These problems are respectively known as (1) immediate selections on a disjunction, (2a) immediate selections on a descendant or an ascendant set, (2b) Not-First/Not-Last problem [2, 10]. All these rules are also known under the general name “edge-finding”, as it consists in deducing ordering relations in the associated disjunctive graph [1].

Not-first/not-last conditions can be seen as part of immediate selections on a set. Indeed, the conjunctive precedences always induce non-conjunctive precedences (a given task must precede a set of other tasks implies that the whole set cannot be performed before this task), which lead to no additional information. Conversely, some sequential properties can be expressed only by non-conjunctive precedences. Domains of application of such rules concern an early detection of inconsistencies in time-resource constrained scheduling problems, and a support for the generation of solutions by pruning the search space without loss of solutions.

The rules involved in the not-first/not-last problem are well-known in the scheduling community [2, 7, 9, 10, 11, 20, 23]. They are often combined with non-insertability conditions to lead back to immediate selections. In this form, they are very effective to quickly bring useful information out and offer a reliable guide for problem solving.

However the detection of impossible sequences used alone gives weaker conditions than immediate selections do. Actually, many conditions found out by the rules are useless because they do not lead to time-window narrowing. Therefore concrete results on their efficiency and their coupling with other propagation rules have been seldom published. Often used in an incomplete way [11, 23], they seemed to prune too little regarding their cost.

In this paper, we accept the challenge to pay more attention to the study of not-first/not-last conditions and to show their interest. We propose a new algorithm to solve the problem, that is to derive all possible deductions, in terms of bound adjustments, induced by these

conditions. We will show how such conditions can improve the propagation process so as to solve scheduling instances in an amount of time significantly reduced from the time required without these conditions.

In a previous work [18], an  $O(n \log n)$  algorithm dealing with not-first/not-last conditions has been reported. The main advantages of this algorithm are its low complexity, its simplicity, and the ability to derive only non-conjunctive precedences which allow a time-bound adjustment. Unfortunately, it does not meet the pursuit of an exhaustive search, which is of major importance in our work (see Section 2.3).

The paper is organized as follows. The problem is more precisely stated in Section 2. The question concerning the convergence of the propagation is particularly discussed. In Section 3, the algorithm and its specificity are presented; the algorithm is qualitatively compared with the procedure proposed in [3] by Baptiste and Le Pape which to our knowledge gave the first reported algorithm to perform all the corresponding deductions in quadratic time. Section 4 is dedicated to the algorithm validation on various job-shop instances.

## 2 The “Not-First/Not-Last Problem”

### 2.1 Immediate Selections

Among edge-finding techniques, immediate selections appear as important rules to decide whether a task is the input or the output of a clique of disjunctions [9, 10]. The immediate selection associated to an ascendant set is described by the rule below. Let  $S \subset T$ ,  $i \in T \setminus S$ :

$$\max_{s \in S} d_s - \min_{s \in S \cup \{i\}} r_i < \sum_{s \in S \cup \{i\}} p_s \implies \begin{cases} i \text{ must be processed after the tasks of } S \\ (\text{i.e., } i \text{ must be processed last}) \end{cases} \quad (1)$$

Once we find an ascendant set, we obtain the necessary adjustment:

$$r_i \leftarrow \max[r_i, \max_{S' \subseteq S} (\min_{s \in S'} r_s + \sum_{s \in S'} p_s)] \quad (2)$$

A symmetrical reasoning can be applied to detect descendant sets.

Obviously, rule (1) is the conjunction of a condition which implies that a task must be processed first or last among the clique and a condition which implies that this task cannot be processed first (Not-First condition). However few works have considered each of these conditions alone because of the absence of efficient algorithms to handle them. In the next section, we focus the presentation on the Not-First/Not-Last conditions.

## 2.2 Not-First/Not-Last Problem Statement

In the Not-First/Not-Last (in short *NF/NL*) problem, the forbidden positions of a single task relatively to a group of tasks are studied. More precisely, one can examine whether a task  $i$  cannot be processed before or after a given subset of tasks  $S$ . The rule in charge of detecting NF conditions is as follows ([7], page IV.6):

**Proposition 1.** *Let  $S \subset T$ ,  $i \in T \setminus S$ :*

$$\max_{s \in S} d_s - r_i < \sum_{s \in S \cup \{i\}} p_s \implies i \text{ NF in } S \cup \{i\} \quad (3)$$

Proof. Suppose  $\max_{s \in S} d_s - r_i < \sum_{s \in S \cup \{i\}} p_s$  and  $i$  precedes any of the tasks in  $S$ . Let  $s_l$  be the task to be scheduled last in  $S$ ; it yields  $t_{s_l} + p_{s_l} \geq r_i + \sum_{s \in S \cup \{i\}} p_s > \max_{s \in S} d_s$  which leads to a contradiction.  $\square$

The only-permitted partial permutations must place  $i$  after at least one task of  $S$ . It yields the following adjustment (Figure 1):

$$r_i \leftarrow \max[r_i, \min_{s \in S} (r_s + p_s)] \quad (4)$$

In the same way, we may search for revealing NL conditions and associated adjustments for deadlines.

**Remark** Immediate selections on disjunctions appear as particular cases of NF/NL when  $S$  is reduced to a singleton  $\{j\}$ . For  $(i, j) \in T^2$ ,  $i \neq j$ , one has:

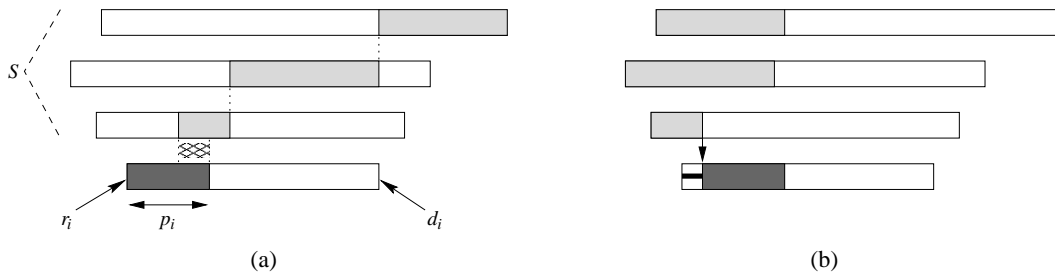


Figure 1: Not-first detection (a) and associated adjustment (b)

$$d_j - r_i < p_i + p_j \implies \begin{cases} i \text{ NF in } \{i, j\} \\ j \text{ NL in } \{i, j\} \end{cases} \implies \begin{cases} r_i \leftarrow r_j + p_j \\ d_j \leftarrow d_i - p_i \end{cases} \iff j \text{ precedes } i$$

### 2.3 Convergence of the Propagation: Fix-Point and Completeness

The genericity of propagation rules is a question of major importance. It induces that the result of the overall propagation must not depend either on the way an inference rule is applied or the order in which different rules are selected. Different behaviours can be expected from different orders of application of the rules on the same problem. This concerns the problem of the convergence of the propagation directly related to the *completeness* of the deductions. The completeness in the time-windows narrowing is pursued, in order to ensure the *convergence of the propagation towards a unique fix-point*.

An algorithm which solves the NF/NL problem must then deduce *all* and not some of the temporal adjustments obtained by application of the rules. Completeness of the NF/NL algorithm is required if one wants to use it in a flexible (independent choice of the rules) and generic (result not dependent of the rules application order) rule-based scheduling tool. Moreover from an experimental view-point, completeness when applying the rules allows a sane behaviour of the propagation process, which is necessary for its evaluation.

A key point of the propagation algorithm concerns which subsets  $S \subset T$  must be considered to ensure both the completeness of the adjustments and a fast convergence towards the

fix-point (considering all the subsets of  $T$  would amount to a knapsack problem and would be too long [11]). These points are treated in the following sections where a temporal bound and a sufficient condition of non-existence of NF adjustments are presented (Section 3.1). They are included in the NF algorithm (Section 3.2).

### 3 An Algorithm for the Not-First/Not-Last Problem

#### 3.1 Formulation of NF/NL Rules

Considering the classical formulation of the Not-First rule, the usual bound used to define the latest start time ( $lst$ ) of a set of tasks  $S$  still unordered and competing for the same resource is computed as  $lst(S) = \max_{s \in S} d_s - \sum_{s \in S} p_s$ . We refine this upper bound on the starting time of  $S$  introducing  $lst_2(S)$ , computed by the following procedure (we suppose  $S$  sorted in non-increasing order of its deadlines; we define an array such that  $d_{S[k]} \geq d_{S[k+1]}$ ).

```

1: procedure latest_start_time
2:  $S =$  tasks sorted in non-increasing order of  $d_i$ 
3:  $lst_2(S) \leftarrow +\infty$ 
4: for  $k := 1$  to  $card(S)$  do
5:    $lst_2(S) \leftarrow \min[d_{S[k]}, lst_2(S)] - p_{S[k]}$ 
6: end for

```

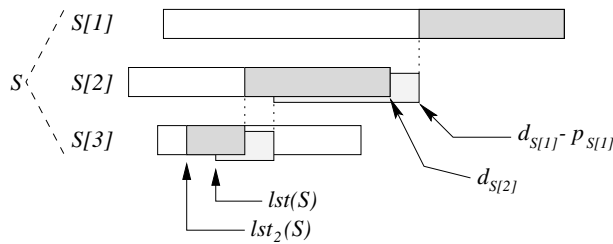


Figure 2: A tighter bound for the latest start time of a set

The bound  $lst_2(S)$  is tighter than  $lst(S)$  (Figure 2); in the worst case,  $lst_2(S) = lst(S)$  and if there exists an iteration  $k$  such that  $d_{S[k]}$  is lower than the current value of  $lst(S)$ , then  $lst_2(S) < lst(S)$ . Therefore, we can use this new bound in our rule and compute it in linear time if the competing tasks are already sorted. Stronger deductions are expected from the rules by using  $lst_2(S)$  instead of  $lst(S)$ . Furthermore, this bound gives a sufficient condition on the non-existence of subsets of  $S$  likely to trigger NF rule. With this bound, if a task  $i$  cannot be said NF in  $S \cup \{i\}$  then it is impossible to find a subset  $S'$  in  $S$  such as  $i$  is NF in  $S' \cup \{i\}$ . It yields the following proposition:

**Proposition 2.** *For all  $S \subset T$  and for all  $i \in T \setminus S$ :*

$$r_i + p_i \leq lst_2(S) \implies \nexists S' \subset S \text{ such that } r_i + p_i > lst_2(S') \quad (5)$$

Proof. Obvious since  $S' \subset S \implies lst_2(S) \leq lst_2(S')$ .  $\square$

**Particular case.** *For all  $i \in T$ , let  $T^i = T \setminus \{i\}$ :*

$$\left. \begin{aligned} r_i + p_i \leq lst_2(T^i) &\implies \nexists S' \subset T^i \text{ such that } r_i + p_i > lst_2(S') \\ &\implies \nexists S' \subset T^i \text{ such that } i \text{ NF in } S' \cup \{i\} \end{aligned} \right\} \quad (6)$$

Condition (6) allows us to avoid useless enumeration of sets of tasks and furnishes in  $O(n)$  time a stopping condition to the search for NF adjustments for a given task ( $O(n^2)$  for the whole set  $T$ ) if the tasks are already sorted.

Based on this property, an algorithm to solve the Not-First problem is proposed in the following paragraph. A symmetrical work on Not-Last conditions, omitted for the sake of conciseness, searches for refining the lower bound on the finishing time (*i.e.*, the earliest finish time or *eft*) of  $S$  from  $eft(S) = \min_{s \in S} r_s + \sum_{s \in S} p_s$  to  $eft_2(S)$  (the associated “NL algorithm” is similar to the NF algorithm given below).

### 3.2 NF Algorithm

The NF algorithm described below computes and stores new release dates  $r_{S[i]}$  ( $S$  sorted in non-increasing order of its deadlines) on every conflicting task  $S[i]$ .

```

1: procedure Not-First
2:  $S =$  tasks sorted in non-increasing order of  $d_i$ 
3: for  $i := 1$  to  $\text{card}(S)$  do
4:   Continue  $\leftarrow$  TRUE
5:    $\text{lst}_2 \leftarrow +\infty$ ,  $\text{eftMin} \leftarrow +\infty$ ,  $j \leftarrow 1$ 
6:   while ( $j \leq \text{card}(S)$  and Continue) do
7:     if ( $r_{S[j]} + p_{S[j]} > r_{S[i]}$  and  $i \neq j$ ) then
8:        $\text{lst}_2 \leftarrow \min(d_{S[j]}, \text{lst}_2) - p_{S[j]}$ 
9:        $\text{eftMin} \leftarrow \min(r_{S[j]} + p_{S[j]}, \text{eftMin})$ 
10:      if ( $\text{lst}_2 < r_{S[i]} + p_{S[i]}$ ) then
11:         $r_{S[i]} \leftarrow \text{eftMin}$ 
12:        Continue  $\leftarrow$  FALSE
13:      end if
14:    end if
15:     $j \leftarrow j + 1$ 
16:  end while
17: end for

```

Let us detail the major ideas of procedure Not-First.

- Line 2 initializes array  $S$ .
- The main loop (lines 3-17) consists in an iteration over all tasks for which we try to prove they are not first.
- The second loop (lines 6-16) considers the tasks that will contribute to define a set on which an NF condition could be detected on  $S[i]$ . This loop is iterated until all tasks are examined or the ‘Continue’ flag is false.
- The key-test of line 7 selects a task  $S[j]$  only if this task will be able to update the release date of  $S[i]$ . This means that a NF condition, if detected, necessarily induces a time-bound adjustment.
- Line 8 reproduces the computation presented in the procedure `latest_start_time` stated previously. This is used in line 10 to detect an NF condition on  $S[i]$ . In line 11 we update the value of the release date of  $S[i]$  to the value computed in line 9 and the ‘Continue’ flag is turned to FALSE (line 12); the procedure exits from  $j$ -loop, and next task  $S[i + 1]$  is considered.

For each task  $S[i]$ , this very simple  $O(n^2)$ -time and  $O(n)$ -space algorithm performs a linear search which selects candidate tasks for the updating of the release date of  $S[i]$ . It iteratively maintains the current bound  $lst_2$  given by the current set of candidate tasks, and stops as soon as an adjustment is possible for the current task  $S[i]$ .

This process must be run until no more deductions can be done. Hence, the NF algorithm and its dual NL have to be embedded in a loop repeated whilst an adjustment is processed. The global NF/NL procedure can be executed at most  $D \times n$  times where  $D = \max_{i=1, \dots, n} (d_i - r_i - p_i)$  denotes the maximum slack over all tasks  $i \in T$ . So the time complexity is in  $O(Dn^3)$  and the space usage remains linear.

### 3.3 Comparison with Related Work

To our knowledge, Carlier-Pinson were the first to use NF/NL conditions [8]. Embedded in their branching scheme, NF/NL conditions are used to detect the tasks that cannot be first or last among all the unscheduled tasks. Hence, a list of possible first tasks is characterized for branching. This is done very fast and leads to a drastic reduction of the search tree. However, the deductions are incomplete so the genericity is not guaranteed. Moreover, they are only useful in the context of a first/last branching scheme. Therefore, this consistency checking technique cannot be used independently of the branching scheme and in a modular way to tighten the time-windows of the tasks.

The algorithm presented in [3] by P. Baptiste and C. Le Pape is to our knowledge the unique alternative to guarantee the completeness of time-window narrowings due to NF/NL conditions. As in our algorithm, it consists of two dual parts: NF and NL. The NF (resp. NL) algorithm pre-computes, for each task  $i$ , temporal bounds  $\delta_{j,i}$  allowing the identification of task  $j$  which induces the greatest possible adjustment of  $r_i$  (resp.  $d_i$ ) in quadratic time. Nonetheless, several runs are needed to solve the NF (resp. NL) subproblem. Hence, this pre-computing effort can be put in question since a single adjustment performed by NF or NL does not longer ensure the completeness of the global process. Several (and not just two) local resolutions of the two NF and NL subproblems have to be realized until time-window stabilization, *i.e.*, until the fix-point is reached. The algorithm is proven to run in  $O(n^2)$  steps; therefore the solution of the global NF/NL problem is also obtained in  $O(Dn^3)$ -time. From the authors opinion, the implementation of their algorithm is encouraging in terms of pruning but quite disappointing in terms of time consuming.

Knowing that NF and NL subproblems will have to be solved more than once each, our procedure abandons the idea of local resolution at each step. For every task, the updating process finds out the *first* possible window adjustment, if any, achievable in a single pass, and performs it without extra computation. So, a simpler approximate processing is realized for each subproblem, in order to save unnecessary time consumptions. The sufficient stopping

condition enounced in (6) ensures that the global completeness is still fulfilled.

With a similar complexity, our algorithm has a clearer formulation. As we will see in the next section, it achieves time-windows narrowing faster and, used in conjunction with a “classical” edge-finding algorithm, offers a good trade-off in the search space reduction in regard to the CPU time consumed.

## 4 Computational Experience

### 4.1 Computation of Optimal Solutions

Consistency checking techniques are part of a more global framework aiming to solve scheduling problems stated as special instances of CSPs. Solving a scheduling CSP by a tree-search method implies three classical components: branching scheme, dead end handling and consistency enforcing.

**Branching scheme** In our work, the branching scheme chosen is inspired from [2]. The heuristic for resource selection is based on the minimum global slack. At each node of the tree, the current solution is extended by choosing a possible-first operation to sequence before all the other unscheduled operations on the current resource. An earliest-start time rule gives the priority in the operation selection. A latest start-time rule is used as a tie-breaker.

**Dead end handling** Chronological backtracking is performed to escape from dead ends.

**Consistency enforcing** The minimal consistency enforcing applied at each node consists in applying the calcLB1-calcUB1 algorithm of Nuijten [23]. This “edge-finder” deduces necessary sequencings between sets of conflicting operations and performs the adjustments of “heads” and “tails” of Carlier-Pinson ([10]) in  $O(n^2)$ .

Are the NF/NL rules worth their applying? To answer this question, our algorithm has been implemented in C++ language and tests have been run on a Sun/Sparc Ultra-5 Workstation

64 MB. The goal of these experiments is to evaluate the power of the NF/NL rules used in addition to Nuijten’s classical edge-finder.

For our experiments we consider the job-shop scheduling problem (JSSP). A finite set of jobs must be processed on a finite set of machines that are disjunctive resources. Each job consists of a sequence of operations of fixed durations; it must be processed on every machine and it has its own predetermined machine sequence (routing). The problem is to determine the job sequences on the machines in order to minimize the makespan, *i.e.*, the duration in which all operations for all jobs are completed. The JSSP is well known to be an  $\mathcal{NP}$ -hard problem in the strong sense. See [17] for a recent overview of the techniques used and the important results obtained.

Starting with an obvious lower bound (*e.g.*, the maximal length of the jobs to be processed), a binary search is performed on the makespan. For each value of the makespan to consider, the branch and bound searches for a solution. We compare the results obtained by applying at each node Nuijten’s edge-finder (column ‘EF alone’) with the same procedure reinforced by the NF/NL algorithms of Baptiste and Le Pape (column ‘EF + NF/NL BLP’) on the one hand, and by the approach proposed in this paper on the other hand (column ‘EF + NF/NL proposed’). The branching scheme and chronological backtracking are used throughout the three search procedures.

Table 1 reports the number of backtracks (BT) and CPU time in seconds needed to obtain an optimal solution on some well-known  $10 \times 10$  instances of JSSPs [1]. The amount of CPU time and the number of backtracks needed for the proof of optimality are reported between brackets. Two major conclusions can be drawn from these results:

- The raw results remain rather far in CPU time and number of backtracks from the most performing constraint-based procedures (see [5, 11]). Different reasons can be found to explain the relatively high number of backtracks. First, the dichotomic approach can lead us to apply constraint propagation rather far from the optimum. There, constraint propagation is no more efficient and a *phase transition* phenomenon can appear [16].

Instances		Consistency enforcing algorithms applied				
		EF alone		EF + NF/NL		
Name	Opt	BT	CPU	BT	CPU (BLP)	CPU (proposed)
ABZ5	1234	146464 (35234)	1017 (249)	46515 (11788)	706 (178)	532 (134)
ABZ6	943	19093 (5558)	117 (35)	7596 (2146)	104 (31)	76 (23)
FT10	930	60884 (22542)	735 (274)	34833 (11389)	994 (341)	757 (259)
LA19	842	127962 (33155)	1392 (353)	37427 (8986)	1158 (283)	863 (211)
LA20	902	35420 (7939)	312 (74)	15898 (3777)	326 (77)	248 (59)
ORB1	1059	14360 (5095)	170 (62)	6697 (2408)	202 (74)	153 (57)
ORB2	888	55060 (18309)	530 (169)	19325 (5977)	481 (147)	362 (111)
ORB3	1005	516848 (61821)	4756 (585)	197398 (24953)	4626 (619)	3553 (475)
ORB4	1005	56832 (12622)	1202 (296)	35588 (7894)	1505 (356)	1144 (271)
ORB5	887	124772 (4736)	913 (53)	75622 (2425)	1165 (63)	903 (47)
total		1157695 (206741)	11144 (2150)	476899 (81743)	11267 (2169)	8591 (1647)

Table 1: Results on ten  $10 \times 10$  JSSP instances

For example, on ORB5, 62369 backtracks are spent before finding a solution at a distance of more than 10% from the optimum, which represent more than 82% of the total number of backtracks needed to find the optimal solution (this value must also be compared with the 2425 backtracks needed to prove the optimality). Moreover, the branching scheme is in fact very simple and do not include any “entropic” guidance [11]. The systematic search performed by the chronological backtracking often leads to long times to recover from a dead end. It must be said that our interest in developing such a procedure was not optimization-oriented but rather towards the evaluation of the efficiency of the NF/NL rules in pruning the search space versus the CPU time needed to apply them in an exhaustive way. Hence, it is more instructive to read the results in a comparative way between the three procedures.

- It appears that applying the NF/NL rules in addition to a classical edge-finder leads to dramatic search space reductions. Actually, it yields a 58% drop in the total number of

backtracks needed to solve the problems tested in Table 1 (drop of 60% for the proofs of optimality). Concerning the CPU time, since BLP obtains results similar to EF alone our approach is comforted in the sense where the mean CPU time is 23% better.

## 4.2 Computation of Lower Bounds

The previous results are encouraging but recent works [12, 17] report that hybrid methods appear today as the most efficient way to compute optimal solutions. Their main strength is to combine different techniques to obtain the best of them. For example, local search methods are very efficient to find good upper bounds [22] and their use in conjunction with our approach would surely lower significantly the number of backtracks spent by our optimal procedure in phase transition areas.

Conversely, constraint propagation techniques are especially designed to help proving optimality and find good lower bounds. Computing lower bounds is a good criterion to evaluate the power and complementarity of different constraint propagation rules. So, we tested different lower bounding techniques (LB1, LB2 and LB3) using different amounts of constraint propagation (R1, R2 and R3 are the sets of rules employed for the reduction of the time-windows).

### Lower bounding techniques

LB1 is obtained by simple propagation (it corresponds to the first value of the makespan for which no infeasibility is derived).

LB2 is also obtained by simple propagation but a technique called “global operations” introduced in [10] and further developed under the name of “shaving” by [21] is used on top of the propagation. Two shaving techniques are employed. For each unscheduled pair of tasks  $(i, j)$ , the first technique sequences  $i$  before  $j$  (resp.  $i$  after  $j$ ) then applies constraint propagation rules with the hope to raise an infeasibility and thus proving the necessary sequencing of  $i$  after  $j$  (resp.  $i$  before  $j$ ). The second shaving technique tries to reduce

the time-windows of the tasks. In this purpose, a task is constrained to start within a reduced part of its time-window and constraint propagation is performed. If an infeasibility arises, the task cannot be processed during this reduced time-window and an adjustment of its original time-window can be performed. A bisection search on the time-window is performed to ensure the greatest possible reduction. To limit the computational time, each shaving technique is only applied once per analysed makespan.

LB3 is the lower bound obtained by sequencing only the tightest machine but the constraint propagation is performed on all the machines during the search. The shaving techniques described above are applied before the start of the search. We used the branching scheme described earlier and arbitrarily limited to 15000 the number of backtracks allowed to a given makespan in order to keep reasonable the computational overhead.

### Constraint propagation rules

We computed LB1, LB2 and LB3 with different amounts of constraint propagation. Willing to evaluate the power and efficiency of Not-First/Not-Last rules versus other propagation rules, we defined three different bases of rules called R1, R2 and R3:

- R1 corresponds to the constraint propagation performed by the Nuijten’s edge-finder.
- R2 adds the proposed NF/NL algorithm to R1.
- R3 adds two other complete constraint propagation algorithms to R2 inspired by rules defined in [11] (exclusion (“case is-after” and its dual)) and [19] (energetic rule “ $\mathfrak{R}3$ ”).

The benchmarks are 33 job-shops from the literature (references can be found in [17]) and are accepted as “hard” problems. The results are reported in Table 2. To make easier the reading of Table 2, we only reported at each column the improved bounds; a hyphen means that the lower bound is equal to the previous value reported on the same line for the problem considered (lower bounds in bold are equal to the optimum). LB/UB are the best known lower and upper bounds (reported from the survey of Jain and Meeran [17]). The “CPU

time span” lines report the minimal and maximal running times observed over all the  $10\times 10$  and  $20\times 20$  tested instances.

LB1-R2 improves the lower bounds found by LB1-R1 for 14 problems out of 33. LB2-R2 improves the lower bounds found by LB2-R1 for 22 problems out of 31 (two lower bounds were optimal after LB2-R1). Finally, LB3-R2 improves 25 out of the 30 non optimal lower bounds computed by LB3-R1. Hence, it seems that the complementarity between the deductions made by Nuijten’s edge-finder and the NF/NL rules increases with the amount of propagation performed: by simple propagation (LB1), the addition of a NF/NL constraint propagation algorithm only slightly improves the bound found by Nuijten’s edge-finder while the shaving mechanism (used in LB2 and LB3) is much more efficient by adding NF/NL adjustments.

We now consider the aggregated results of LB2 and LB3 reported in Table 3. Line  $\Delta$  stands for the relative distance from the best known lower bounds; an index of 100 is given for the average distance found by LB2 and LB3 when the Nuijten’s edge-finder is used alone. Performing an extra amount of consistency checking using R3 gives a slightly better bound than R2, while it needs much more computation time. It seems that R2 covers most of the deductions made by R3 and hence offers a good trade-off between the quality of the bound obtained and the CPU time needed to find it.

Nonetheless, some notorious problems like LA21 or LA29 remain indifferent to the technique used or the amount of propagation performed, and the initial lower bound far from the optimum is not improved. It seems that bound-consistency alone (consistency checking based on the tightening of the time-windows) is not powerful enough yet to tackle these problems in a reasonable time.

## Conclusions

Throughout this paper we have considered specific conditions for sequencing in disjunctive scheduling. More precisely, we study the not-first/not-last deductions which reveal that a task cannot be performed before/after a set of other tasks. While this kind of deduction

Instances			Lower bounding technique applied								
			LB1			LB2			LB3		
Name	Size	LB/UB	R1	R2	R3	R1	R2	R3	R1	R2	R3
FT10	10×10	930	855	858	868	902	–	–	907	912	914
ABZ5	10×10	1234	1126	–	1127	1191	1199	1201	1200	1207	–
ABZ6	10×10	943	889	–	890	934	941	–	934	941	–
ABZ7	20×15	656	651	–	–	–	–	–	–	652	–
ABZ8	20×15	645/665	608	–	–	620	–	–	–	–	621
ABZ9	20×15	661/679	630	–	–	643	–	–	646	647	648
LA16	10×10	945	901	909	–	943	<b>945</b>	–	944	<b>945</b>	–
LA18	10×10	848	803	–	809	<b>848</b>	–	–	–	–	–
LA19	10×10	842	755	756	763	814	821	822	816	822	–
LA20	10×10	902	836	851	–	890	897	898	893	897	898
LA21	15×10	1046	1033	–	–	–	–	–	–	–	–
LA22	15×10	927	913	–	–	924	–	–	<b>927</b>	–	–
LA24	15×10	935	889	892	–	909	912	–	915	920	–
LA25	15×10	977	919	–	–	950	955	956	952	955	957
LA29	20×10	1152	1119	–	–	–	–	–	1120	–	–
LA36	15×15	1268	1233	–	–	1258	1262	–	1267	–	–
LA38	15×15	1196	1106	–	–	1143	1145	–	1158	1162	1171
LA39	15×15	1233	1221	–	–	1230	1231	–	1232	–	–
LA40	15×15	1222	1190	1192	–	1206	1207	1208	–	1209	1210
ORB1	10×10	1059	975	–	–	1006	1013	–	1036	1045	1048
ORB2	10×10	888	812	–	815	860	864	865	863	867	869
ORB3	10×10	1005	906	907	–	961	965	–	975	978	981
ORB4	10×10	1005	898	–	–	968	972	973	997	<b>1005</b>	–
ORB5	10×10	887	810	822	–	856	858	860	868	872	873
ORB6	10×10	1010	946	947	–	984	989	–	991	999	–
ORB7	10×10	397	358	365	–	387	389	–	388	390	391
ORB8	10×10	899	894	–	–	<b>899</b>	–	–	–	–	–
ORB9	10×10	934	901	909	–	929	–	–	–	932	–
ORB10	10×10	944	923	–	–	941	943	<b>944</b>	941	943	<b>944</b>
YN1	20×20	826/888	782	784	–	809	811	812	811	812	813
YN2	20×20	861/909	818	819	825	840	841	–	849	852	853
YN3	20×20	827/893	799	–	–	815	816	817	815	817	–
YN4	20×20	918/968	881	884	885	895	–	–	904	905	906
10 × 10 CPU time span (s)			0.1–1	0.1–1	0.1–1	20–160	35–210	60–350	30–1700	40–3100	80–5500
20 × 20 CPU time span			1–2 s	1–2 s	1–6 s	0.8–2.6 ks	0.9–4.1 ks	1.6–7.5 ks	5–14 ks	4–29 ks	11–56 ks
Mean dev. from best LB (%)			5.38	5.13	5.01	2.14	1.88	1.84	1.59	1.29	1.19

Table 2: Lower bounds on hard instances of JSSPs

Set of rules	R1	R2	R3
$\Delta$	100	84.5	80.4
Approximate time factor	1	1.8	3.8

Table 3: Aggregated results from LB2 and LB3

already exists in consistency enforcing of scheduling, it has been rather neglected in the sense of no particular attention has been paid till now. The paper is concerned with the resolution of the Not-First/Not-Last problem, that means we strive towards the detection of all the not-first/not-last conditions and the associated adjustments, so as to offer a generic reasoning. Hence we propose a simple, polynomial, efficient, but also complete algorithm to solve the Not-First/Not-Last problem.

We have then evaluated the usefulness of considering not-first/not-last conditions for scheduling problems. First, a branch and bound procedure has been devised to solve job-shop instances; it serves to highlight the benefits obtained with not-first/not-last reasoning in terms of reduction of the number of backtracks to reach an optimal solution. It also shows for the first time that considering explicitly not-first/not-last conditions are encouraging in terms of CPU time savings. Next, some experiments have been managed to compute lower bounds on hard instances in order to evaluate the power and complementarity of different constraint propagation rules. The main result is that the stronger the propagation techniques are, the better the complementarity between not-first/not-last reasoning and classical edge-finding. This result could deserve to be delved, in particular as an eventual support for local search to obtain good upper bounds; recent works using constraint propagation and especially the shaving technique to prune large parts of neighborhood report excellent results [12]. The use of constraint propagation in local search methods and particularly the shaving techniques employed are promising fields of research.

Finally, we claim that the results presented in this paper are not restricted to disjunctive scheduling problems. Indeed they can contribute to derive strong deductions also for

cumulative instances where disjunctive parts can be exhibited (see [4]). The NF/NL deductions could for example be incorporated in global procedures developed for the Resource-Constrained Project Scheduling Problem ([6] for example).

## Acknowledgements

The authors want to thank Patrick Esquirol for many discussions on the topic and Philippe Baptiste for sharing his experience and advices with us.

## References

- [1] D. Applegate and W. Cook. A computational study of the job-shop scheduling problem. *ORSA Journal on Computing*, 3(2):149–156, 1991.
- [2] P. Baptiste and C. Le Pape. A theoretical and experimental comparison of constraint propagation techniques for disjunctive scheduling. In *Proc. of the 14<sup>th</sup> IJCAI*, Montréal, Canada, 1995.
- [3] P. Baptiste and C. Le Pape. Edge-finding constraint propagation algorithms for disjunctive and cumulative scheduling. In *Proc. of U.K. planning and scheduling SIG meeting*, Liverpool, November 1996.
- [4] P. Baptiste and C. Le Pape. Constraint propagation and decomposition techniques for highly disjunctive and highly cumulative project scheduling problems. In *Proc. of the 3<sup>rd</sup> International Conference on Principles and Practice of Constraint Programming*, Schloss Hagenberg, Austria, 1997.
- [5] P. Baptiste, C. Le Pape, and W.P.M. Nuijten. Constraint-based optimisation and approximation for job-shop scheduling. In *Proc. of the 14<sup>th</sup> IJCAI*, Montréal, Canada, 1995.

- [6] P. Brucker, S. Knust, A. Schoo, and O. Thiele. A branch and bound algorithm for the resource-constrained project scheduling problem. *European Journal of Operational Research*, 107:272–288, 1998.
- [7] J. Carlier. *Ordonnancement à contraintes disjonctives*. Thèse de troisième cycle, Université Paris VI, 1975.
- [8] J. Carlier and E. Pinson. An algorithm for solving the job-shop problem. *Management Science*, 35(2):164–176, 1989.
- [9] J. Carlier and E. Pinson. A practical use of Jackson’s preemptive schedule for solving the job-shop problem. *Annals of Operations Research*, 26:269–287, 1990.
- [10] J. Carlier and E. Pinson. Adjustment of heads and tails for the job-shop problem. *European Journal of Operational Research*, 78:146–161, 1994.
- [11] Y. Caseau and F. Laburthe. Improving branch and bound for jobshop scheduling with constraint propagation. In M. Deza, R. Euler, and Y. Manoussakis, editors, *Proc. of the 8<sup>th</sup> Franco-Japanese–4<sup>th</sup> Franco-Chinese Conference on Combinatorics and Computer Science*, Brest, France, July 1995.
- [12] Y. Caseau and F. Laburthe. Effective forget-and-extend heuristics for scheduling problems. In *Proc. of the First International Workshop on Integration of AI and OR techniques in Constraint Programming for Combinatorial Optimization Problems (CP-AI-OR’99)*, Ferrara, Italy, 1999.
- [13] J. Erschler and P. Esquirol. Decision-aid in job shop scheduling: A knowledge based approach. In *Proc. of IEEE International Conference on Robotics and Automation*, pages 1651–1656, San Francisco, CA, 1986.
- [14] J. Erschler, F. Roubellat, and J.-P. Vernhes. Finding some essential characteristics of the feasible solutions for a scheduling problem. *Operations Research*, 24(4):774–783, 1976.

- [15] J. Erschler, F. Roubellat, and J.-P. Vernhes. Characterizing the set of feasible sequences for  $n$  jobs to be carried out on a single machine. *European Journal of Operational Research*, 4(3):189–194, 1980.
- [16] S.A. Grant and B.M. Smith. The phase transition behaviour of maintaining arc-consistency. Technical Report 95.25, Division of Artificial Intelligence, University of Leeds, United Kingdom, August 1995.
- [17] A.S. Jain and S. Meeran. Deterministic job-shop scheduling: Past, present and future. *European Journal of Operational Research*, 113:390–434, 1999.
- [18] M.-L. Levy, P. Lopez, and B. Pradin. Décomposition temporelle et caractérisation de solutions admissibles pour le problème d’ordonnancement à une machine. *RAIRO-Recherche Opérationnelle/Operations Research*, 33(2):185–208, 1999.
- [19] P. Lopez. *Approche énergétique pour l’ordonnancement de tâches sous contraintes de temps et de ressources*. Thèse de Doctorat, Université Paul Sabatier, Toulouse, 1991.
- [20] P. Lopez and P. Esquirol. Consistency enforcing in scheduling: A general formulation based on energetic reasoning. In *Proc. of the 5<sup>th</sup> International Workshop on Project Management and Scheduling*, pages 155–158, Poznan, Poland, 1996.
- [21] P. Martin and D.B. Shmoys. A new approach to computing optimal schedules for the job-shop scheduling problem. In *Proc. of the 5<sup>th</sup> International Conference on Integer Programming and Combinatorial Optimization, IPCO’96*, pages 389–403, Vancouver, British Columbia, June 1996. LNCS, volume 1084, Springer Verlag. W.H. Cunningham, S.T. McCormick, and M. Queyranne (editors).
- [22] E. Nowicki and C. Smutnicki. A fast taboo search algorithm for the job-shop problem. *Management Science*, 42(6):797–813, 1996.
- [23] W.P.M. Nuijten. *Time and resource constrained scheduling – A constraint satisfaction approach*. PhD thesis, Eindhoven University of Technology, 1994.

- [24] W.P.M. Nuijten and E.H.L. Aarts. A computational study of constraint satisfaction for multiple capacitated job-shop scheduling. In *Proc. of the 4<sup>th</sup> International Workshop on Project Management and Scheduling*, pages 166–173, Leuven, Belgium, 1994.
- [25] S. Smith and C. Cheng. Slack-based heuristics for constraint satisfaction scheduling. In *Proc. of AAAI-93*, pages 139–144, Washington, DC, 1993.
- [26] E.P.K. Tsang. *Foundations of Constraint Satisfaction*. Academic Press Ltd., London, 1993.
- [27] P. van Hentenryck. *Constraint Satisfaction in Logic Programming*. Logic Programming Series. MIT Press, Cambridge, MA, 1989.