A Generic Algorithm for Solving the Not-First/Not-Last Problem

Abstract

This paper is concerned with the development of constraint propagation techniques for the characterization of feasible schedules. Some of such 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.

In the solution generation phase, the genericity of propagation rules is a question of major importance. It induces that the result of the overall propagation does not have to 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 narrowing due to not-first/not-last conditions. It has been integrated in a branch and bound procedure to solve job-shop instances. Computational results are reported and compared to other results obtained till now on the same field.

Keywords: Disjunctive Scheduling, Constraint Propagation, Branch and Bound, Temporal Reasoning.

1 Introduction

In this paper, we are concerned with disjunctive scheduling. A set T of n tasks is to be performed by a set of disjunctive resources, that is each resource processes only one task at a time. Tasks cannot be interrupted (this corresponds to the non-preemptive case). A release time r_i, a due date d_i, and a processing time p_i characterize each task i of T. The relation $d_i - p_i \geq r_i$ holds $\forall\, i = 1, \ldots, n$. 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) [20]. 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 explicitly defined or implicitly implied by the problem. 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 [9, 16, 19, 21]. 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 [10, 11, 17]. An interval of possible start times is assigned to each task by computing extreme bounds for its processing (so-called "heads" and "tails" in [5]). Propagation rules are applied to update (or adjust) these bounds, and then deduce new constraints from the current search state so as to characterize as well as possible the feasible schedules. Among different ways of characterization, we give importance to adjustments derived from deductions obtained in a necessary way.

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 [1, 6, 9, 14, 16]. These problems are respectively known as (1) immediate selections on a disjunction, (2a) immediate selections on an ascendant or a descendant set, (2b) Not-First/Not-Last problem [1, 6].

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 phase 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 [1, 5, 6, 7, 14, 16]. 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 [7, 16], 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. Note that these results can most probably strengthen one of the time-oriented branching schemes presented by Martin and Schmoys in [15]; the authors propose a branching-on-the-next-available-operation strategy: for the first available task they generate two branches, one corresponds to the task beginning immediately, the other means that the task is delayed (that is, the task is not-first in the set of unscheduled tasks since they restrict their search to active schedules).

In a previous work [13], 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 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 2.2).

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 [2] 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 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 S of n tasks. The rule in charge of detecting NF conditions is as follows:

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

$$\max_{s \in S} d_s - r_i < \sum_{s \in S \cup \{i\}} p_s \Longrightarrow i \text{ NF in } S \cup \{i\} \tag{1}$$

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

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

$$i \text{ NF in } S \cup \{i\} \Longrightarrow r_i \leftarrow \max[r_i, \min_{s \in S}(r_s + p_s)] \tag{2}$$

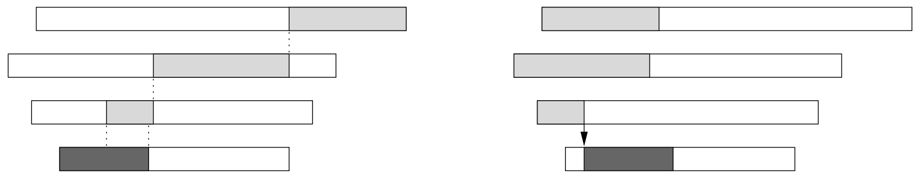

Figure 1: Not-first detection and associated adjustment

In the same way, we may search for revealing NL conditions and associated adjustments for due dates:

Proposition 2. *Let $S \subset T$, $j \in T \backslash S$:*

$$d_j - \min_{s \in S} r_s < \sum_{s \in S \cup \{i\}} p_s \Longrightarrow j \text{ NL in } S \cup \{j\} \Longrightarrow d_j \leftarrow \min[d_j, \max_{s \in S}(d_s - p_s)] \tag{3}$$

<u>Proof</u>. Similar to the proof of Proposition 2.

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:

$$d_j - r_i < p_i + p_j \Longrightarrow \begin{cases} i \text{ NF in } \{i,j\} \\ j \text{ NL in } \{i,j\} \end{cases} \Longrightarrow \begin{cases} r_i \leftarrow r_j + p_j \\ d_j \leftarrow d_i - p_i \end{cases} \Longleftrightarrow j \prec i \ (j \text{ precedes } i)$$

2.2 Convergence of the propagation: Fix-Point and Completeness

In the solution generation phase, the genericity of propagation rules is a question of major importance. It induces that the result of the overall propagation does not have to 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*. The example presented below (taken from [1]) illustrates this notion of unique fix-point.

Example. The following table displays the characteristics of three tasks to schedule on a machine.

task	release date	due date	processing time
A	3	10	3
B	4	11	3
C	4	10	2

Let us try to update sequence C before $S = \{A, B\}$, that means: is $\max(d_A, d_B) - r_C \geq p_C + p_A + p_B$ satisfied ? As: $11 - 4 < 2 + 3 + 3$, it yields $C \not\prec S$. Then r_C is updated to $\min(r_A + p_A, r_B + p_B) = 6$.

Suppose now another task D has to be considered in S. This can be due to an incremental addition of a new activity, or to an updating process applied on D and which implies that this task will be necessarily scheduled in the interval $[r_A, d_B]$. Its characteristics are: $r_D = 4$; $d_D = 10$; $p_D = 1$. Of course, C cannot still be scheduled before $S = \{A, B, D\}$. However, the deduction on its release date updating is now $\min(r_A+p_A, r_B+p_B, r_D+p_D) = 5$, which is lower than 6 obtained with a less constrained problem.

The previous example shows that the way a rule is applied can have an influence on the final result. As mentioned previously, a similar drawback can occur considering different orders in the selection of various rules. This consequence is strongly unexpected from a flexible (independent choice of the rules) and generic (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.

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. In the following section, such an algorithm is proposed.

3 A generic 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 latest start time of a set of tasks S 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 due dates):

begin

  for $k := 1$ **to** $card(S)$ **do**

    $lst_2(S) \leftarrow \min[d_{S[k]}, lst_2(S)] - p_{S[k]}$ **od**

end

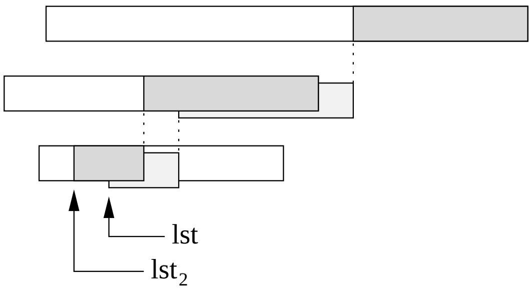

Figure 2: A tighter bound for the latest start time of a set

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 3. *For all $S \subset T$ and for all $i \in T \backslash S$:*

$$r_i + p_i \leq lst_2(S) \Longrightarrow \nexists S' \subset S \text{ such that } r_i + p_i > lst_2(S') \tag{4}$$

Proof. Obvious since $S' \subset S \Longrightarrow lst_2(S) \leq lst_2(S')$. □

Corollary. *For all $i \in T$, let $S = T \backslash \{i\}$:*

$$r_i + p_i \leq lst_2(S) \Longrightarrow \nexists S' \in S \text{ such that } r_i + p_i > lst_2(S')$$

This corollary allows us to avoid useless enumeration of sets of tasks and furnishes in $O(n \log n)$ time a condition to stop the search for NF adjustments.

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 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 date $r_{S[i]}$ (S sorted by non increasing order of due dates) on every conflicting task $S[i]$ as early as possible and returns TRUE if no adjustment is found (the fix-point of the NF subproblem is reached):

NF algorithm

<u>**begin**</u>

completeNF $\leftarrow$ TRUE;

<u>**for**</u> $i := 1$ <u>**to**</u> n <u>**do**</u>

$lst_2 \leftarrow +\infty$; $eftMin \leftarrow +\infty$;

<u>**for**</u> $j := 1$ <u>**to**</u> n <u>**do**</u>

<u>**if**</u> $(r_{S[j]} + p_{S[j]} > r_{S[i]}) \wedge (i \neq j)$

<u>**then**</u> $lst_2 \leftarrow \min(d_{S[j]}, lst_2) - p_{S[j]}$;

$eftMin \leftarrow \min(r_{S[j]} + p_{S[j]}, eftMin)$;

<u>**if**</u> $(lst2 < r_{S[i]} + p_{S[i]})$

<u>**then**</u> $r_{S[i]} \leftarrow \max(r_{S[i]}, eftMin)$;

completeNF $\leftarrow$ FALSE;

<u>**exit**</u> <u>**fi**</u> <u>**fi**</u> <u>**od**</u> <u>**od**</u>

<u>**end**</u>

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 $S[i]$. If for all i, no adjustment is detected for $S[i]$, the 'completeNF' flag is set and the NF problem is solved. Otherwise, the flag is turned to FALSE, the procedure exits from j-loop, and next task $S[i+1]$ is considered. This must be processed until no more deductions can be done. Hence, the NF algorithm and its dual NL have to be embedded in a loop checking their completeness flags before exiting:

NF/NL Algorithm
<u>**begin**</u>
  <u>**while**</u> $\neg$(completeNF) $\vee$ $\neg$(completeNL) <u>**do**</u>
    NF Algorithm;
    NL Algorithm <u>**od**</u>
<u>**end**</u>

The global NF/NL algorithm can be executed at most $D \times n$ times where $D = \max_{i=1,\ldots,n}(d_i - r_i - p_i)$ denotes the maximum over all tasks $i \in T$ of the slack of a task. So the time complexity is in $O(Dn^3)$ and the space usage remains linear in the size of the input data.

3.3 Comparison with related work

The algorithm presented in [2] is to our knowledge the unique alternative to guarantee the completeness of the time-windows narrowing due to NF/NL conditions. As in our algorithm, it consists of two dual parts, NF and NL, but the main philosophy relies here on locally solving each subproblem at each step. Actually, Baptiste and Le Pape solves the NF in a complete way in a single run by pre-computing, for each task i, temporal bounds Δ_{ij} allowing the identification of task j which induces the maximal adjustment of r_i. This computing effort, done to carry out the maximal NF adjustment for i, can be put in question by the close links between NF and NL. Indeed, a single adjustment performed by NF does not longer ensure the completeness of NL. Several (and not just two) complete resolutions of the two NF and NL subproblems have to be realized until time-window stabilization, *viz.* until the fix-point is reached. Hence even if NF and NL are separately and sequentially solved in quadratic time, the time complexity of the global NF/NL solution is also obtained in $O(Dn^3)$.

From the authors opinion, the implementation of their algorithms is encouraging in terms of pruning but quite disappointing in terms of time consuming.

Our approach is similar in the sense that it also has two symmetric algorithms to solve the NF and the NL separately. The main difference is that, knowing that NF and NL will have to be solved more than once each, we relax the constraint of completeness to the global NF/NL and abandon the idea of solving the NF subproblem in a single run. For every task, the adjusting process finds out the first possible window updating, if any, achievable in a single pass, and performs it without extra computation. An approximate processing is then realized for each subproblem, in order to save unnecessary time consumptions; but the global completeness is still fulfilled (figure 3).

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, offer a good trade-off in the search space reduction in regard to the CPU time consumed.

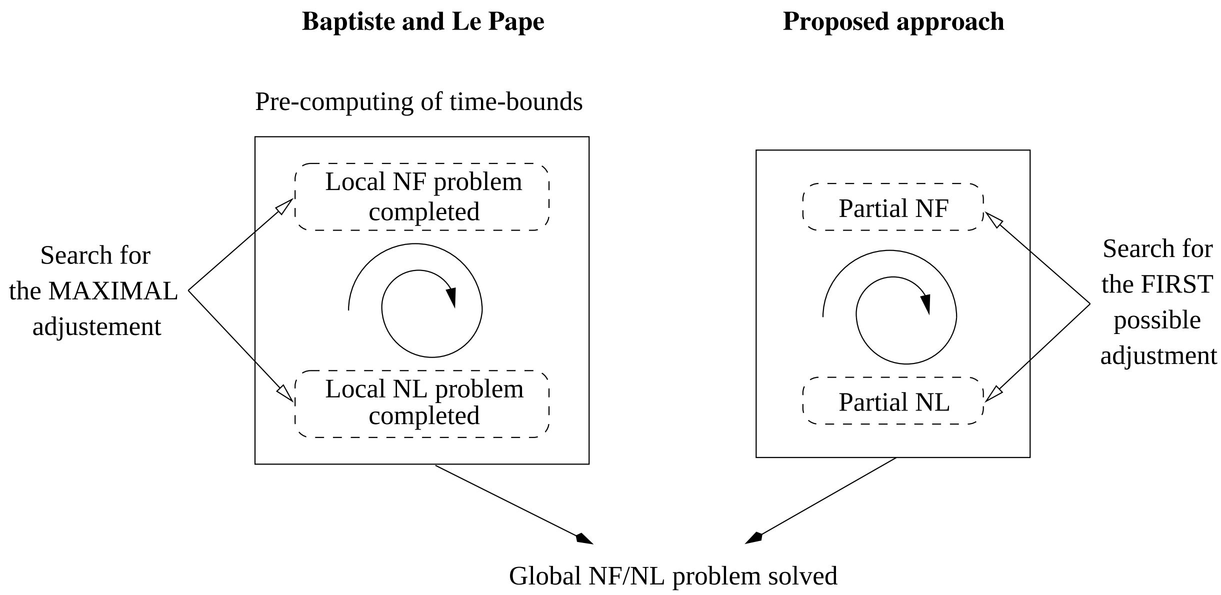

Figure 3: Two possible approaches to solve the NF/NL problem

4 Computational Experience

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

- The branching scheme chosen is a very simple and classical one. At each node of the tree, the current solution is extended by choosing a possible first operation to sequence on the current resource. A earliest start time-latest start time rule gives the priority in the operation selection.
- Chronological backtracking is performed to escape from dead ends.
- The consistency enforcing applied at each node consists in applying the calcLB1-calcUB1 algorithm of Nuijten [16]. this "edge-finder" deduces necessary sequencings between sets of conflicting operations. We compare the result of this consistency enforcing with the same reinforced by the NF/NL algorithms of Baptiste and Le Pape and the one proposed.

Starting with an obvious lower bound (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.

The goal of these experiments is to evaluate the power of the NF/NL rules used in addition to Nuijten's classical edge-finder. Are the NF/NL rules worth their applying? Table 1 reports the number of backtracks and CPU time needed to obtain an optimal solution on some well known hard instances of job-shop problems. A comparison is made between the three different consistency enforcing patterns applied. The branching scheme and chronological backtracking are used throughout the three search procedures.

Several conclusions can be drawn from these results :

- The raw results remain far in CPU time and number of backtracks from the most performing constraint-based procedures (see [18, 7]). 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. The branching scheme is in fact very simple and do not include any entropic guidance [7]. The systematic search performed by the chronological backtracking often leads to long times to recover from a dead end. So, 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. The total number of backtracks needed to solve the problems tested in table 1 lowers from 4,296,800 to 1,294,068. The total CPU time for all the problems drops from 96,500 s to 52,315 s. Two exceptions have to be quoted. On the ft20 instance from [12], the extra-consistency enforging performed by the not-first/not-last algorithm contribute in misleading the branching scheme. Solving the orb04 instance, the number of backtracks removed by the NF/NL algorithm is not sufficient to balance the computing effort needed. For all the other problems, the CPU time needed to obtain a solution is always lowered. Our approach is comforted in the sense where the mean CPU time is 21

- As shown in [2], the NF/NL rules, applied in a complete fashion, subsume the immediate selections between pairs of tasks. So, no disjunctive pair algorithm is needed to obtain the associated deductions.

- A phase transition phenomenon appears on some instance. For example, on the orb04 instance, it is clear that proving the optimality of the problem is less difficult than finding a solution at some distance of the optimum. This is obvious on figure 4. For a given makespan, the time reported (in seconds) is the CPU time needed to prove infeasibility or to find a solution. An inbetween constrained problem reduce the efficiency of the consistency enforcing applied although that finding a solution not that far from the optimum remains still hard without some reliable guidance. "Is more consistency enforcing in a phase transition worth its applying?" remains an open question.

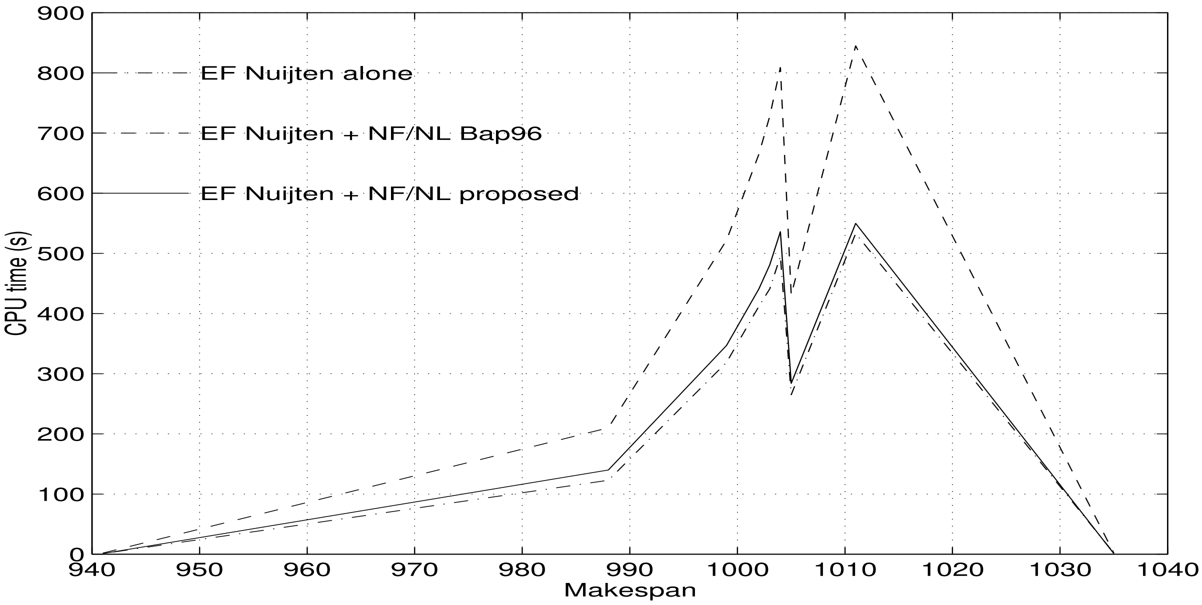

Figure 4: Phase transition phenomenon - instance orb04 (optimum 1005)

Conclusions

In this paper, we have presented a simple, polynomial, and efficient algorithm to solve the Not-First/Not-Last Problem. The main idea is to detect all the not-first/not-last conditions for sequencing and evaluate their usefulness for solving scheduling problems with disjunctive constraints. A branch and bound procedure has been devised to solve job-shop instances and experimental results have been carried out. They show that our algorithm contibutes to improve the results obtained by other authors in the same topic. Though, this work is still in progress and the current results are far from the best branch and bound procedures.

It could be interesting to include some "intelligent" backtracking in our search or perform a restart and search along a different path if the procedure remains "stuck" for too long in a sub-tree in order to obtain more "competitive" results.

As stated before, the major goal of this paper was to focus on some "forgotten" consistency enforcing rules. Applied in a new and complete way, the Not-first/Not-last rules have proven their usefulness in guiding a search tree procedure and shown a real complementarity with a classical edge-finder. Furthermore, 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 [3]). Our algorithm could for example be incorporated in global procedures developed for the Resource-Constrained Project Scheduling Problem (RCPSP) ([4] or [8] for example).

Acknowledgements

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

References

[1] P. Baptiste and C. Le Pape. A theoretical and experimental comparison of constraint propagation techniques for disjunctive scheduling. In *Proc. of the 14^{th} IJCAI*, Montreal, Canada, 1995.

[2] 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.

[3] 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^{rd} *International Conference on Principles and Practice of Constraint Programming*, Schloss Hagenberg, Austria, 1997.

[4] 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.

[5] 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.

[6] J. Carlier and E. Pinson. Adjustment of heads and tails for the job-shop problem. *European Journal of Operational Research*, 78:146–161, 1994.

[7] 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^{th} *Franco-Japanese–*4^{th} *Franco-Chinese Conference on Combinatorics and Computer Science*, Brest, France, July 1995.

[8] E. Demeulemeester and W. Herroelen. A branch and bound procedure for the multiple resource-constrained project scheduling problem. *Management Science*, 38:1803–1818, 1992.

[9] 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.

[10] 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.

[11] 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.

[12] H. Fisher and G.L. Thompson. Probabilistic learning combinations of local job-shop scheduling rules. In J.F. Muth and G.L. Thompson, editors, *Industrial Scheduling*, pages 225–251. Prentice Hall, Englewood Cliffs, NJ, 1963.

[13] 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*, 1998. To appear.

[14] P. Lopez and P. Esquirol. Consistency enforcing in scheduling: A general formulation based on energetic reasoning. In *Proc. of the 5^{th} International Workshop on Project Management and Scheduling*, pages 155–158, Poznan, Poland, 1996.

[15] P. Martin and D.B. Schmoys. A new approach to computing optimal schedules for the job-shop scheduling problem. In *Proc. of the 5^{th} Conference on Integer Programming and Combinatorial Optimization*, 1996.

[16] W.P.M. Nuijten. *Time and resource constrained scheduling - A constraint satisfaction approach.* PhD thesis, Eindhoven University of Technology, 1994.

[17] 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^{th} International Workshop on Project Management and Scheduling*, pages 166–173, Leuven, Belgium, 1994.

[18] C. Le Pape P. Baptiste and W.P.M. Nuijten. Constraint-based optimisation and approximation for job-shop scheduling. In *Proc. of the 14^{th} IJCAI*, Montreal, Canada, 1995.

[19] S. Smith and C. Cheng. Slack-based heuristics for constraint satisafaction scheduling. In *Proc. of AAAI-93*, pages 139–144, Washington, DC, 1993.

[20] E.P.K. Tsang. *Foundations of Constraint Satisfaction.* Academic Press Ltd., London, 1993.

[21] P. van Hentenryck. *Constraint Satisfaction in Logic Programming.* Logic Programming Series. MIT Press, Cambridge, MA, 1989.

Problems		Consistency enforcing algorithms applied					
		EF alone		EF + NF/NL Bap96		EF + NF/NL proposed	
Name	Size	Backtracks	CPU time	Backtracks	CPU time	Backtracks	CPU time
ABZ5	10×10	203,870	3,564	54,432	2,236	54,432	1,760
ABZ6	10×10	34,756	462	11,914	379	11,914	333
ESQ1	10×10	2,735	38	1,084	34	1,084	33
ESQ2	10×10	629	7	239	6	239	5
ESQ3	10×10	876	12	417	16	417	12
ESQ4	10×10	374	16	253	18	253	12
ESQ5	10×10	315	1	19	0	19	0
ESQ6	10×10	1,298	24	448	21	448	16
FT06	6×6	4	0	4	0	4	0
FT10	10×10	168,042	4,498	74,403	4,453	74,403	3,746
FT20	20×5	5	0	112,970	1,031	112,970	912
LA02	10×5	52	0	18	0	18	0
LA19	10×10	1,192,514	19,926	213,765	10,080	213,765	8,658
LA20	10×10	71,830	1,131	17,787	873	17,787	625
ORB01	10×10	16,128	415	7,529	505	7,529	363
ORB02	10×10	121,410	2,152	31,505	1,440	31,505	1,127
ORB03	10×10	2,020,052	55,611	577,942	35,792	577,942	27,882
ORB04	10×10	76,756	2,591	44,704	4,214	44,704	2,781
ORB05	10×10	385,154	6,052	144,635	5,117	144,635	4,050

Table 1: Results on instances of JSSP