Lattice of task intervals: A support for edge-finding in disjunctive scheduling

P. Torres[1], P. Lopez[1], P. Esquirol[1,2]

[1]LAAS-CNRS
7, avenue du colonel Roche
F-31077 Toulouse Cedex 4 – FRANCE

[2]INSA
Complexe Scientifique de Rangueil
F-31077 Toulouse Cedex 4 – FRANCE

e-mails:* {*ptorres, lopez, esquirol*}*@laas.fr

Abstract. In this paper, we present a structure to control efficiently constraint propagation rules in disjunctive scheduling. This structure is based on a lattice of task intervals. Task intervals are sets of tasks sharing for the same resource and involving an inclusion relation on their time windows. A lattice is proposed to organize the structuring of such task intervals. We discuss about the interest of this lattice for disjunctive scheduling and we show it is well-suited to support the control of consistency enforcing techniques such as edge-finding.

Key words: Scheduling, Constraint Propagation, Temporal Reasoning, Edge-Finding.

1 Introduction

In this paper, we are concerned with *disjunctive scheduling*. A set T of n tasks is to be performed on one disjunctive resource which can process only one task at a time. Tasks cannot be interrupted (non-preemption). 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 $\forall\, i = 1, \ldots, n$.

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 [7, 11, 13, 15]. The problem is seen as a Constraint Satisfaction Problem (CSP): the question is to find an assignment of the start time t_o of each task o within $[r_o, d_o - p_o]$, consistent with the resource constraint.

Here we more specifically focus on techniques for enforcing the problem consistency by constraint propagation [8, 9, 12]: an interval of possible start times is assigned to each task by computing extreme bounds for its processing, and propagation rules are applied to adjust these bounds so as to characterize as well as possible the feasible schedules. An important set of these rules refers to as *edge-finding*. It consists in determining whether a given task must or cannot be performed before or after a set of other competing tasks [4, 6, 10, 11].

To trigger efficiently this type of rules, one can take benefits from the notion of *task intervals* introduced in [5] by Caseau and Laburthe.

Task intervals are presented in the next section. Some elements are given concerning their interest in disjunctive scheduling but also their main shortcoming, *i.e.,* the costly interval maintenance. Thus we propose in Section 3 the *lattice* structure of task intervals as an alternative of this interval

maintenance. Section 4 discusses on the use of this structure with *edge-finding* techniques to support the resolution of scheduling problems. Finally, Section 5 presents the use of non-insertability conditions that can be drawn from the lattice structure to guide a Branch and Bound procedure.

2 Task intervals and edge-finding

2.1 Task intervals

To any pair of tasks $(o, o') \in T \times T$ such that: $r_o \leq r_{o'} \wedge d_o \leq d_{o'}$, one can associate a *task interval* denoted by $[o, o']$ and defined by the subset:

$$\{x \in T | r_o \leq r_x \wedge \ d_x \leq d_{o'}\} \tag{1}$$

This definition has been first introduced in [5]. Let us notice that the term "task interval" is a misnomer since a valid definition of an interval on a discrete set T is based on a unique binary ordering relation $\mathcal{R} : [o, o'] = \{x \in T | (o\mathcal{R}x) \wedge (x\mathcal{R}o')\}$. Definition (1) (related to the *set extension* of task interval $[o, o']$) refers to two time point relations ($\mathcal{R}_1 : r_o \leq r_x ; \mathcal{R}_2 : d_x \leq d_{o'}$). Actually, the interval that matters here is the time interval $[r_o, d_{o'}]$. However it is convenient to make this slight misuse of language and hence we keep the term task interval throughout the document.

A unique task interval is associated to any non empty subset $S \subseteq T$; it is sufficient to find one pair (o, o') such that: $r_o = \min_{s \in S} r_s = r_S$ and $d_{o'} = \max_{s \in S} d_s = d_S$. On the contrary, several terms of the form $[o, o']$ may denote the same task interval using tasks y such that $r_y = r_o$ and/or $d_y = d_{o'}$.

Therefore $\forall\, S \subseteq T$ with $p_S = \Sigma_{s \in S} p_s$, checking $d_S - r_S \geq p_S$ amounts to checking it only on task intervals. Given a set of n tasks competing for a disjunctive resource, one can generate the set of task intervals in $O(n^2)$ if a total ordering on the release dates and the deadlines is decided.

2.2 Rules

Edge-finding techniques integrate important rules that reveal whether a task must be performed before ($\ll$) or after ($\gg$) a set of other competing tasks. Given a task o and a set of competing tasks $S \subset T$ such that $o \notin S$, these rules can be written:

$$\left\{ \begin{array}{l} d_S - r_S < p_o + p_S \\ d_o - r_S < p_o + p_S \end{array} \right. \quad \Rightarrow \quad \left\{ \begin{array}{c} o \ll S \\ d_o \leftarrow \min(d_o, d_S - p_S) \\ \forall\, s \in S,\ r_s \leftarrow \max(r_s, r_o + p_o) \end{array} \right. \tag{2}$$

$$\left\{ \begin{array}{l} d_S - r_S < p_o + p_S \\ d_S - r_o < p_o + p_S \end{array} \right. \quad \Rightarrow \quad \left\{ \begin{array}{c} o \gg S \\ r_o \leftarrow \max(r_o, r_S + p_S) \\ \forall\, s \in S,\ d_s \leftarrow \min(d_s, d_o - p_o) \end{array} \right. \tag{3}$$

that means each rule triggers if two conditions hold:

- The first condition of rules (2) and (3) yields a *non-insertability* condition of o within S, denoted by $o \nmid S$.
- The second condition of rules (2) and (3) detects a *forbidden sequencing* between o and S; $o \not\ll S$ (resp. $o \not\gg S$) stands for o not before (resp. not after) S.

It has been shown in [5, 6] that task intervals are a nice and efficient support to apply edge-finding reasoning in disjunctive scheduling.

A task interval is a redundant structure; it aggregates symbolic and numerical information about the set of tasks it represents (its extension). On the one hand, a sharp insight on the tightness of time-windows and an easy application of inference rules are allowed but, on the other hand, each time-window adjustment deduced by edge-finding induces the modification or the removal of the current task intervals; the creation of new intervals can also occur. The set of tasks intervals must then be updated as soon as a time-bound is changed. Unfortunately, the algorithms in charge of the task interval maintenance are costly [5]. In the following, we propose the *lattice of task intervals* as an original alternative to the task interval maintenance. This ordered structure is proposed to offer a clear and efficient exploitation of the task intervals by edge-finding rules.

3 The lattice of task intervals

Formal definition of a lattice. Let $\mathcal{U}$ be a finite set of elements and $\mathcal{R}$ a binary relation on $\mathcal{U}$. If each pair of elements has both a lower bound and an upper bound, $(\mathcal{U}, \mathcal{R})$ is known to be a *lattice* [1, 2].

Lattice of task intervals. Let $\mathcal{U}$ be the set of task intervals. We define the *lattice of task intervals*, in short LTI, $(\mathcal{U}, \Subset)$ as the set of task intervals $\mathcal{U}$ ordered by the immediate inclusion relation $\Subset$. The immediate inclusion $\Subset$ can be defined as follows: Let I, J be two task intervals: if $I \Subset J$ then it does not exist a task interval K such that $I \Subset K \Subset J$. $(\mathcal{U}, \Subset)$ is a lattice because all pairs of task intervals admit a common lower bound $\emptyset$ and a common upper bound T.

Remark: Instead of introducing the immediate inclusion relation, one can consider the general inclusion relation (with transitivity property). Hence the LTI can be defined using the Hasse's diagram that removes all transitive relations.

Example. Figure 1(a) displays the time windows of 5 tasks A, B, C, D, E. Seven task intervals are found of which set extensions are: $\{A, B, C, D, E\}$, $\{B, C, D, E\}$, $\{A, B, C, D\}$, $\{B, C, D\}$, $\{D, E\}$, $\{B\}$, and $\{D\}$ (in the figure it is denoted ABCDE, ... for convenience). These task intervals are represented on the Hasse's diagram of Figure 1(b). Figure 1(c) is a computer-oriented representation of an LTI. Left and right branches are respectively ordered following the non-decreasing order of the release dates and the non-increasing order of the deadlines. An edge is labelled by the task involved in the updating of the current task interval when modifying the time-bounds.

4 Lattice of task intervals: a support for edge-finding rules

4.1 Construction

Let X be the list of tasks sorted in non-decreasing order of its release dates and Y be the list of tasks sorted in non-increasing order of its deadlines. We retain the lexicographic criterion to break the ties and obtain a total ordering on the release dates and the deadlines of the tasks (in the previous example, it yields $X = [A, B, C, E, D]$ and $Y = [E, A, C, D, B]$ and then, *e.g.,* $X[2] = Y[5] = B$).

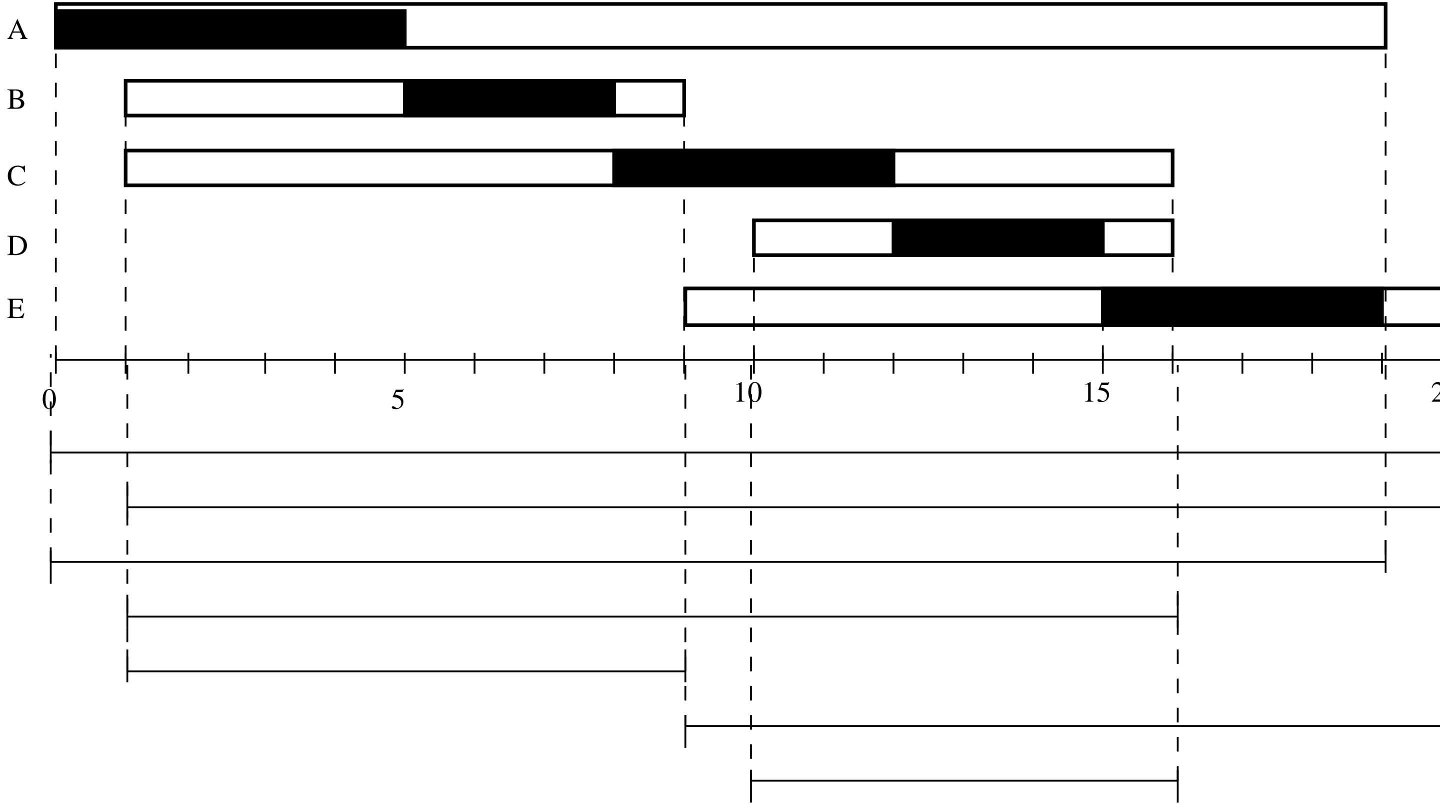

Figure 1: Example of a lattice of task intervals

Starting from the maximal task interval, *i.e.,* the task interval including all the tasks of T, two task intervals satisfying the $\Subset$ relation are generated by removing the beginning task on the one hand and the ending task on the other hand. Thus, to construct the LTI, each task interval $[X[i], Y[j]]$ is separated into a left son $[X[i+1], Y[j]]$ and a right son $[X[i], Y[j+1]]$; $\forall\, i, j \in [1, n-1]$:

$$[X[i+1], Y[j]] = [X[i], Y[j]] \setminus \{X[i]\} \tag{4}$$

$$[X[i], Y[j+1]] = [X[i], Y[j]] \setminus \{Y[j]\} \tag{5}$$

Characteristics:

- The LTI is exhaustive. The total ordering on the release dates and the deadlines gives access to all the possible $[r_{X[i]}, d_{Y[j]}]$ time intervals. Therefore, all the task intervals are represented in the LTI and can be cross-accessed using vectors X and Y.
- An inconsistency can be detected during the construction of the LTI. A sufficient condition of inconsistency for the task interval $[X[i], Y[j]]$ is $r_{X[i]} + p_{[X[i],Y[j]]} > d_{Y[j]}$.
- This structure is redundant, *i.e.,* two nodes with the same label represent two different time intervals (see Figure 1(c)). Elements $[X[i], Y[j]]$ that do not match the time interval $[r_{X[i]}, d_{Y[j]}]$ are not to be considered as task intervals.

During the LTI construction, we assume the eventuality of redundancies and inadequate sets of tasks (in Figure 1(c), $\{C, D\}$ is redundant with $\{B, C, D\}$ and will have to be removed). The

following section will present two sets of rules allowing us to suppress redundant or inadequate elements of the LTI.

4.2 Cuts in the lattice

4.2.1 Redundancies due to identical time-bounds.

A unique time-window bounds a given set of tasks but the opposite is wrong. In Figure 1(a), consider the subset $\{B, C, D\}$. It is bounded by the sole time interval $[1, 16]$. Conversely, given the time interval $[1, 16]$, it is possible to consider three task intervals matching this time interval: $\{B, C, D\}$, $\{B, D\}$ and $\{C\}$. The task interval of greatest extension, in this case $\{B, C, D\}$, is called the maximal task interval.

Proposition 1 *Let $[X[i], Y[j]]$ be a task interval, if $\exists\, k > i$ such that $r_{X[k]} = r_{X[i]}$ then $[X[k], Y[j]]$ is a non-maximal task interval.*

Proof: $r_{X[i]} = r_{X[k]} \Rightarrow$ time intervals $[r_{X[i]}, d_{Y[j]}]$ and $[r_{X[k]}, d_{Y[j]}]$ are equal. Since $k > i$, $[X[k], Y[j]] \subset [X[i], Y[j]]$ (due to the $\Subset$ relation).

Corollary It is useless to apply rules (2) and (3) on non-maximal task intervals.

Proof: Since $[X[k], Y[j]] \subset [X[i], Y[j]]$ and $r_{X[k]} = r_{X[i]}$, $d_{Y[j]} - r_{X[k]} - p_{[X[k],Y[j]]} > d_{Y[j]} - r_{X[i]} - p_{[X[i],Y[j]]}$. Let $o \in T$ such that $o \notin [X[i], Y[j]]$ (*i.e.,* $o \notin [X[k], Y[j]]$). If $d_{Y[j]} - r_{X[i]} - p_{[X[i],Y[j]]} - p_o \geq 0$ then necessarily $d_{Y[j]} - r_{X[k]} - p_{[X[k],Y[j]]} - p_o > 0$ and no deduction can be found by rules (2) and (3) between o and $[X[k], Y[j]]$. Conversely, if a sequencing is detected and triggers rule (2) or (3), it would be worth considering the maximal task intervals, *i.e.,* $[X[i], Y[j]]$, to derive the greatest time-bound adjustments.

A symmetrical proposition and its associated corollary can be derived in the same way from redundancies of deadlines. Considering the LTI from the example of Figure 1, the task intervals dashed in Figure 2 are the consequence of the redundancies in the release dates of the tasks B and C (redundant in regard to their right ascendant), and the deadlines of C and D (redundant in regard to their left ascendant).

Figure 2: Cuts based on identical time-bounds

4.2.2 Inadequate sets of tasks.

According to definition (1), $\forall\, [X[i], Y[j]] \in (\mathcal{U}, \Subset)$, inequalities $r_{X[i]} \leq r_{Y[j]}$ (a) and $d_{X[i]} \leq d_{Y[j]}$ (b) must hold to ensure $[X[i], Y[j]]$ is a task interval.

Proposition 2 *$\forall\, i, j$ such that $X[i] = Y[j]$, any interval $[X[k], Y[j]]$ (resp. $[X[i], Y[l]]$), with $k > i$ (resp. $l > j$) is either not a task interval or a non-maximal task interval.*

Proof: $k > i \Rightarrow r_{X[k]} \geq r_{X[i]}$. If $r_{X[k]} > r_{X[i]}$ then $[X[k], Y[j]]$ cannot be a task interval because $Y[j] = X[i]$ and then (a) is not satisfied. If $r_{X[k]} = r_{X[i]}$ then, if $[X[k], Y[j]]$ is a task interval, it is non-maximal (see Proposition 1). The proof is similar for $[X[i], Y[l]]$, $l > j$ (condition (b) not satisfied or $d_{Y[l]} = d_{Y[j]}$ and $[X[i], Y[l]]$ is a non-maximal task interval).

In Figure 3 are dashed the task intervals not valid due to Proposition 2 with $X[i] = Y[j] = A$. The left branch of dashed task intervals represent task intervals finishing with A but no longer including this task, hence these sets of tasks do not match with their related time interval in the LTI. The same reasoning stands for sets of tasks starting with, but not including A.

Figure 3: Inadequate sets due to $X[i] = Y[j] = A$

The two cuts are applied exhaustively in the example of Figure 1 and the resulting lattice is reported in Figure 4. From a theoretical maximum number of 25 task intervals, the real number of relevant task intervals to examine by edge-finding is reduced to 7.

Figure 4: The LTI after the cuts

4.2.3 Particular cases of LTI.

Depending on the temporal structure of the problem, the rate of redundant or not valid task intervals is highly variable. The two extreme cases are reported in Figure 5. Given n competing tasks, the number of task intervals to consider varies from n for a fully pyramidal problem to $\frac{n(n+1)}{2}$ for a strictly terraced problem. In a practical way, the number remains far from the theoretical n^2 possible task intervals.

4.3 Applying edge-finding on the lattice

Edge-finding rules (2) and (3) aim to discover mandatory ordering between a task o and a subset of tasks S such that $o \notin S$. The condition $o \notin S$ implies a consequence on the set of task intervals to consider on the lattice. In fact, $o \notin S \Rightarrow r_o < r_S$ (a) or $d_o > d_S$ (b). These conditions divide the LTI into four distinctive areas with common possible deductions:

- The first area groups together all the task intervals including o; hence this area is irrelevant to trigger the rules.

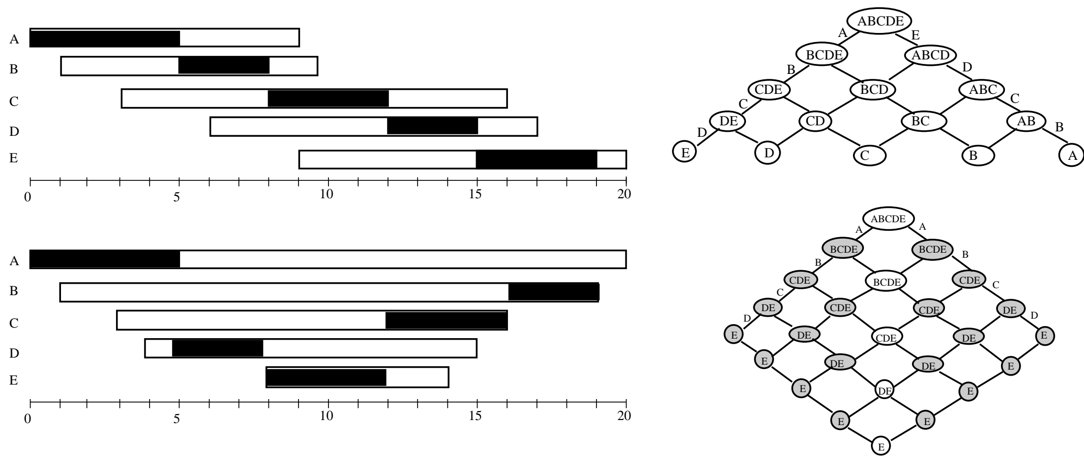

Figure 5: Terraced and pyramidal structure with their LTI

- Condition (a) holds in the second area. If $o \nmid S$ then $o \ll S$.
- Condition (b) holds in the third area. If $o \nmid S$ then $o \gg S$.
- Both conditions (a) and (b) are verified in area 4. If $o \nmid S$ then $o \ll S$ or $o \gg S$.

Remark: As said previously, the consistency of the task intervals of the LTI is checked during the construction phase. So, we assume every task interval to be consistent during the propagation of edge-finding rules.

Figure 6 illustrates the partition of the LTI of Figure 4 for task C. Searching for deductions on C leads us to only consider task intervals located in the areas 2, 3 and 4.

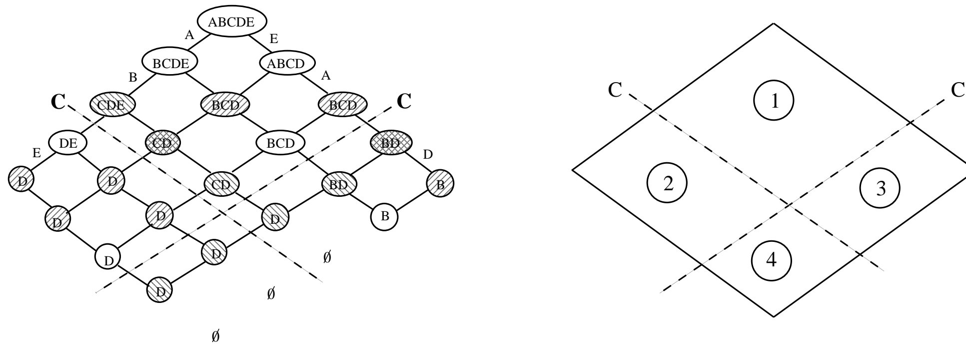

Figure 6: Partition of the LTI when reasoning on task C

Sub-lattices of candidate task intervals are now defined for each task of the problem. The strategy chosen is a breadth-first search in the sub-lattices. Indeed, due to the inclusion relation between the elements in the LTI, the task intervals with the largest set extension are tried first. The strongest sequencing deductions found out by edge-finding are expected to be those implying the largest sets of tasks, and thus solving the largest number of disjunctions.

Some explanations on the completeness of constraint propagation are necessary to understand the use of the LTI for edge-finding. An algorithm in charge of the propagation of a rule has to be complete to ensure the genericity property [14]. The completeness in the application of a rule is reached when no more deductions can be made on the current state of the problem; hence it is

necessary to process more than once the propagation algorithms (each adjustment of release dates or deadlines can trigger stronger deductions than those already found). Assuming this fact, the propagation algorithm in charge of analyzing the LTI does not search for the maximal time-window tightening on one run but rather searches for the first possible adjustment for a task, stores it and then proceeds with the next task. When finished, the adjustments are performed and propagated to the other constraints of the problem (such as precedences). A LTI is generated from the new state of the problem and the former LTI is dropped. This offers an alternative to the costly maintenance of task intervals.

5 Use of non-insertability conditions in a branching scheme

Edge-finding rules (2) and (3) trigger if two conditions are verified between a task o and a set of tasks S ($o \notin S$): a non-insertability condition alone ($o \nmid S$) and a forbidden sequencing condition (either $o \not\ll S$ or $o \not\gg S$). It is then deduced that o must be sequenced before or after the whole set S. From a time-window tightening point of view, this deduction is often very interesting and since [3], many researchers have been interested in edge-finding techniques. Less powerful, forbidden sequencings alone lead also to more constrained time-windows (see [14] for a survey on the problem and an efficient way to recover the related deductions). Here, we are interested in the non-insertability conditions we can obtain from a LTI.

If $o \nmid S$ then necessarily, either $o \ll S$ or $o \gg S$. This condition is equivalent to a new disjunction between o and **all** the tasks of S. A forbidden sequencing between o and S must be proven to solve the disjunction. Without this condition, the sequencing disjunction between o and S remains. To our knowledge, these conditions alone are ignored by the current edge-finding algorithms because no time-windows tightening can be obtained directly, and non-insertability conditions have never been used in a branching scheme to guide a search tree procedure. Nevertheless, $o \nmid S$ subsumes all the disjunctions concerning the pairs (o, s), $\forall\, s \in S$. A single disjunction replaces Card(S) (the cardinal of S) disjunctions which induces a less combinatorial problem.

Most search tree procedures have branching schemes taking a sequencing decision on a pair of tasks in disjunction at each node of the tree. The branching could be modified to post in priority the disjunctions $o \nmid S$ found during the application of the rules on the current node. In theory, the sets S with maximal cardinality are expected to provide the strongest reduction in terms of number of backtracks and should be posted first.

A condition $o \nmid S$ can be obtained alone (without proving $o \not\ll S$ or $o \not\gg S$) in area 4 of the LTI ($r_o < r_S$ and $d_o > d_S$). $o \nmid S$ alone implies to eject a certain part of the processing of o outside $[r_S, d_S]$ but with an indecision about the sequencing of o and S. The reduction in the number of disjunctions is proportional to Card(S) and therefore area 4 should be explored in a breadth-first manner to obtain the condition of non-insertion implying the sets of maximal cardinality.

This is just an overview on the interest of non-insertability conditions between a task and a set of tasks and their use in a branching scheme has still to be implemented and validated.

6 Conclusion

In this paper, an original structure tailored to perform edge-finding rules on a set of competing tasks has been proposed. Sets of tasks viewed as task intervals are organized in a lattice. The constraint propagation can take advantage of properties such as cuts or precise location to perform

the edge-finding rules on a minimal set of task intervals. First experiments on Job Shop instances give competitive results in terms of CPU time.

Furthermore, the interest of non-insertability conditions between a task and a set of other tasks has been discussed. The lattice offers a good support to detect these conditions. They can be stored to give a theoretical guidance to a search tree procedure based on the selection of disjunctive pairs. Our work on these conditions is still in progress and has to be developed to prove the efficiency (or not) of these conditions in a branching scheme.

References

[1] F.D. Anger and R.V. Rodríguez. Lattice structure of temporal interval relations. *Applied intelligence*, 6:29–38, 1996.

[2] H. Bestougeff and G. Ligozat. *Outils logiques pour le traitement du temps*. Etudes et recherches en informatique. Masson, Paris, 1989.

[3] J. Carlier and E. Pinson. An algorithm for solving the job-shop problem. *Management Science*, 35(2):164–176, 1989.

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

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

[6] 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 8th Franco-Japanese–4th Franco-Chinese Conference on Combinatorics and Computer Science*, Brest, France, July 1995.

[7] 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.

[8] 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.

[9] 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.

[10] C. Le Pape and P. Baptiste. Constraint-based scheduling: A theoretical comparison of resource constraint propagation rules. In *Proc. of ECAI 98 Workshop on Non Binary Constraints*, 1998.

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

[12] 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 4th International Workshop on Project Management and Scheduling*, pages 166–173, Leuven, Belgium, 1994.

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

[14] P. Torres and P. Lopez. A generic algorithm for solving the not-first/not-last problem in disjunctive scheduling. In *Proc. of the 6th International Workshop on Project Management and Scheduling*, pages 305–308, Istanbul, Turkey, 1998.

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