Resource constraints for preemptive job-shop scheduling

CLAUDE LE PAPE[1] AND PHILIPPE BAPTISTE[1,2] clp@challenger.bouygues.fr, baptiste@utc.fr

[1]*Bouygues, Direction Scientifique, 1, avenue Eugène Freyssinet, 78061 Saint-Quentin-en-Yvelines, France*

[2]*UMR CNRS 6599 HEUDIASYC, Université de Technologie de Compiègne, France*

Abstract. This paper presents an experimental study of constraint propagation algorithms for preemptive scheduling. We propose generalizations of non-preemptive constraint propagation techniques (based on timetables, on disjunctive constraints, and on edge-finding) to preemptive and "mixed" problems, i.e., problems in which some activities can be interrupted and some cannot. Another approach relies on incremental flow-based techniques. We theoretically compare these approaches and present an experimental comparison based on a branch and bound procedure for the preemptive variant of the job-shop scheduling problem. We show that both edge-finding and flow-based techniques allow the resolution of hard problem instances, including the preemptive variant of the famous FT10.

Keywords: Constraint propagation, preemptive scheduling, resource constraints, timetables, disjunctive constraints, edge-finding, network flows, job-shop scheduling

1. Preemptive disjunctive scheduling

Given a set of resources with given capacities, a set of activities with given durations and resource requirements, and a set of temporal constraints between activities, a "pure" scheduling problem consists of deciding when to execute each activity, so that both temporal constraints and resource constraints are satisfied. Most scheduling problems can easily be represented as instances of the constraint satisfaction problem (Kumar, 1992): given a set of variables, a set of possible values (domain) for each variable, and a set of constraints between the variables, assign a value to each variable, so that all the constraints are satisfied.

Several types of scheduling problems can be distinguished:

- In *disjunctive* scheduling, each resource can execute at most one activity at a time. In *cumulative* scheduling, a resource can run several activities in parallel, provided that the resource capacity is not exceeded.
- In *non-preemptive* scheduling, activities cannot be interrupted. Each activity A must execute without interruption from its start time to its end time. In *preemptive* scheduling, activities can be interrupted at any time, e.g., to let some other activities execute.

A non-preemptive scheduling problem can be encoded efficiently as a constraint satisfaction problem: two variables, $start(A)$ and $end(A)$, are associated with each activity A; they represent the start time and the end time of A. The smallest values in the

domains of *start*(*A*) and *end*(*A*) are called the earliest start time and the earliest end time of *A* (EST_A and EET_A). Similarly, the greatest values in the domains of *start*(*A*) and *end*(*A*) are called the latest start time and the latest end time of *A* (LST_A and LET_A). The duration of the activity is an additional variable, defined as the difference between the end time and the start time of the activity.

A preemptive scheduling problem is more difficult to represent: one can either associate a set variable (i.e., a variable the value of which will be a set) *set*(*A*) with each activity *A*, or define a 0-1 variable *W*(*A, t*) for each activity *A* and time *t*; *set*(*A*) represents the set of times at which *A* executes, while *W*(*A, t*) assumes value 1 if and only if *A* executes at time *t*. Ignoring implementation details, let us note that:

- the value of *W*(*A*, *t*) is 1 if and only if *t* belongs to *set*(*A*).
- assuming time is discretized, *start*(*A*) and *end*(*A*) can be defined, in both the preemptive and the non-preemptive case, by $start(A) = \min_{t \in set(A)}(t)$ and $end(A) = \max_{t \in set(A)}(t + 1)$; in the preemptive case, these variables are often needed to connect activities together by temporal constraints.
- in the non-preemptive case, *set*(*A*) = [*start*(*A*) *end*(*A*)), with the interval [*start*(*A*) *end*(*A*)) closed on the left and open on the right so that $|set(A)| = end(A) - start(A) = duration(A)$.

A number of researchers have recently designed, implemented, and evaluated constraint propagation techniques for non-preemptive disjunctive and cumulative scheduling. See, for example, (Smith, 1983), (Rit, 1986), (Le Pape & Smith, 1987), (Collinot & Le Pape, 1987), (Burke, 1989), (Prosser, 1990), (Burke & Prosser, 1991), (Erschler *et al*., 1991), (Le Pape, 1991), (Lopez, 1991), (Beck, 1992), (Smith, 1992), (Lopez *et al*., 1992), (Aggoun & Beldiceanu, 1993), (Smith & Cheng, 1993), (Varnier *et al*., 1993), (Caseau & Laburthe, 1994), (Cheng & Smith, 1994), (Laborie, 1994), (Le Pape, 1994), (Nuijten & Aarts, 1994), (Nuijten, 1994), (Baptiste & Le Pape, 1995a), (Baptiste & Le Pape, 1995b), (Caseau & Laburthe, 1995a), (Caseau & Laburthe, 1995b), (Cheng & Smith, 1995a), (Cheng & Smith, 1995b), (Caseau & Laburthe, 1996a), (Colombani, 1996), (Lévy, 1996), (Lock, 1996), (Nuijten & Aarts, 1996), (Baptiste & Le Pape, 1997). Coupled with branch and bound backtracking algorithms, these techniques proved to be successful on both academic and industrial problems. In comparison, preemptive scheduling problems (and "mixed" problems where some activities can be interrupted and some cannot) have received almost no attention from both the Operations Research and the Artificial Intelligence community. In this paper, we propose generalizations of non-preemptive constraint propagation techniques to preemptive and "mixed" disjunctive scheduling. For each technique (Sections 2 to 5), we first introduce the constraint propagation rules and algorithms developed by other researchers for the non-preemptive case; then, we present our generalization. Section 6 presents an experimental study based on the preemptive variant of the job-shop scheduling problem. Section 7 presents conclusions and future work.

2. Timetable constraints

2.1. The non-preemptive case

A common mechanism to propagate resource constraints in the non-preemptive case relies on an explicit data structure called "timetable" to maintain information about resource utilization and resource availability over time. Resource constraints are propagated in two directions: from resources to activities, to update activity time bounds (earliest and latest start and end times) according to the availability of resources; and from activities to resources, to update the timetables according to the time bounds of activities. Although several variants exist (Le Pape, 1988), (Fox, 1990), (Le Pape, 1994), (Smith, 1994), (Caseau & Laburthe, 1996a), (Lock, 1996), the propagation mainly consists of maintaining "arc-B-consistency" (Lhomme, 1993)[1] on the formula:

$$\sum_A [W(A, t) * capacity(A)] \leq capacity(t)$$

where $capacity(A)$ denotes the capacity required by activity A, $capacity(t)$ denotes the capacity available at time t, and $W(A, t)$ is an implicit 0-1 variable representing the Boolean value $[start(A) \leq t < end(A)]$.

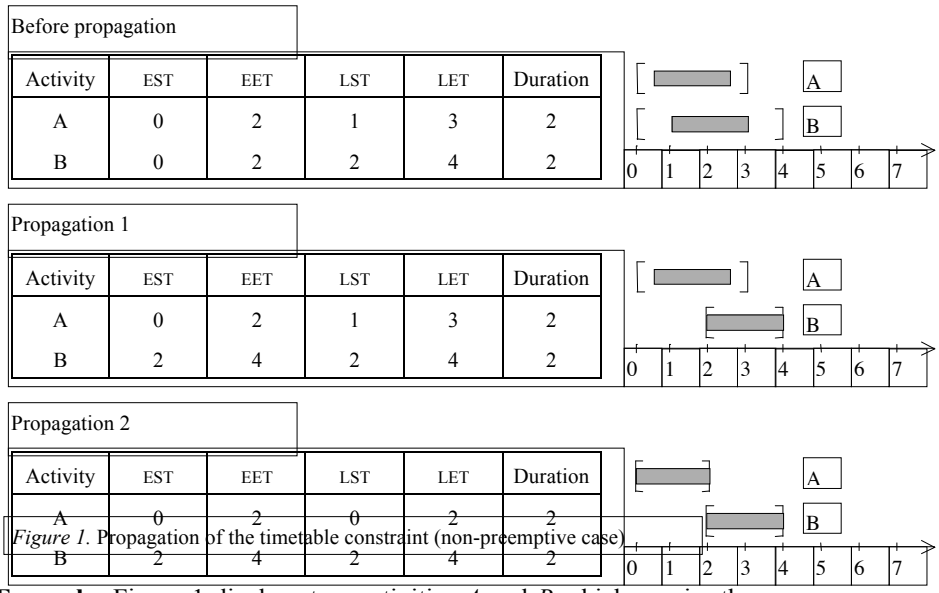


Figure 1. Propagation of the timetable constraint (non-preemptive case).

Example: Figure 1 displays two activities A and B which require the same resource of capacity 1. The latest start time LST_A of A is smaller than its earliest end time EET_A. Hence, it is guaranteed that A will execute between $LST_A = 1$ and $EET_A = 2$. Over this period, $W(A, t)$ is set to 1 and the corresponding resource amount is no longer available for B. Since B cannot be interrupted and cannot be finished before 1, the earliest start time of B is updated to 2 (propagation 1). Then, $W(B, t)$ is set to 1 over the interval [2, 4), which

results in a new propagation step, where the latest end time of A is set to 2 (propagation 2).

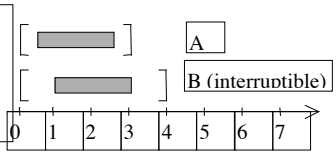
2.2. The preemptive case

At the first glance, it seems that the main principle of the timetable mechanism directly applies to both the preemptive and the mixed case. However, an important difference appears in the relation between the five variables $W(A, t)$, $set(A)$, $start(A)$, $end(A)$, and $duration(A)$. The earliest start time EST_A can easily be set to "the first time t at which $W(A, t)$ can be 1." Similarly, the latest end time LET_A can easily be set to 1 + "the last time t at which $W(A, t)$ can be 1." However, the earliest end time EET_A must be computed so that there possibly exist $duration(A)$ time points in $set(A) \cap [EST_A\ EET_A)$, and the latest start time LST_A must be computed so that there possibly exist $duration(A)$ time points in $set(A) \cap [LST_A\ LET_A)$. These additional propagation steps make the overall propagation process far more complex.

In the reverse direction, it is important to notice that $W(A, t)$ cannot be set to 1 as soon as $LST_A \leq t < EET_A$. The only situation in which $W(A, t)$ can be deduced to be 1 is when no more than $duration(A)$ time points can possibly belong to $set(A)$. This is unlikely to occur before decisions (choices in a search tree) are made to instantiate $set(A)$. Therefore, constraint propagation cannot prune much.

Before propagation

Activity	EST	EET	LST	LET	Duration
A	0	2	1	3	2
B	0	2	2	4	2



Propagation 1

Activity	EST	EET	LST	LET	Duration
A	0	2	1	3	2
B	0	3	2	4	2

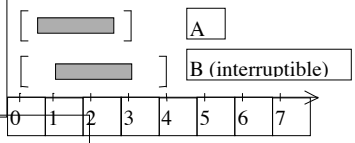


Figure 2. Propagation of the timetable constraint (mixed case)

Example: Given the data of Figure 1, the timetable mechanism cannot deduce anything if both activities can be interrupted. Figure 2 shows what happens when only B can be interrupted. As in Figure 1, it is guaranteed that A will execute between $LST_A = 1$ and $EET_A = 2$. Over this period, the corresponding resource amount is no longer available for B. The earliest end time of B is then set to 3. Then the propagation process stops since there is no time point at which B is guaranteed to execute.

Both costly and in most cases ineffective, the timetable mechanism appears far from satisfactorily applicable to preemptive problems. Several researchers incorporated *some*

possibilities of preemption in their constraint-based algorithms or applications (e.g., interrupt a machining operation in favor of a planned machine maintenance but not in favor of another machining operation, interrupt an activity at most once or twice (Zweben *et al*., 1993), (Smith, 1994), (Le Pape, 1996), (Pegman *et al*., 1997)). Few did attack the general problem of preemptive and mixed scheduling. Demeulemeester (1992) developed a branch and bound algorithm for a particular preemptive cumulative scheduling problem: the preemptive resource-constrained project scheduling problem. This algorithm relies on memorizing states rather than on constraint propagation to prune the search space. Baptiste (1994) reports on a tentative implementation, in ILOG SOLVER (Puget, 1994), (Puget & Leconte, 1995), of preemptive timetable constraints. The reported results confirm that even on simple problems the propagation process is rather slow. Let us note, however, that the timetable mechanism can easily be generalized to other types of resources, such as resources of capacity $m > 1$, or resources which must be in a specific state for an activity to execute (Le Pape, 1994). Such is not the case for the techniques described in the following sections.

3. Disjunctive constraints

3.1. The non-preemptive case

In non-preemptive disjunctive scheduling, two activities A and B which require a common resource R cannot overlap in time: either A precedes B or B precedes A. If n activities $A_1 \ldots A_n$ require R, the resource constraint can be implemented as $n*(n-1)/2$ (explicit or implicit) disjunctive constraints. As for timetable constraints, variants exist in the literature (Erschler, 1976), (Carlier, 1984), (Esquirol, 1987), (Le Pape, 1988), (Smith & Cheng, 1993), (Varnier *et al*., 1993), (Baptiste & Le Pape, 1995a), but in most cases the propagation consists of maintaining arc-B-consistency on the formula:

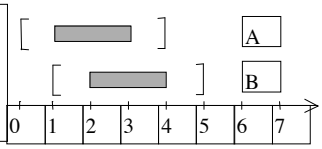
$$[end(A) \leq start(B)] \text{ or } [end(B) \leq start(A)].$$

Disjunctive constraints provide more precise time bounds than the corresponding timetable constraints. Indeed, if an activity B is known to execute at some time t between the earliest start time EST_A and the earliest end time EET_A of A, $LST_B \leq t < EET_A$. Then, B must precede A and the propagation of the disjunctive constraint implies $start(A) \geq EET_B > t$.

The following example shows that, in some cases, disjunctive constraints propagate more than timetable constraints.

Before propagation

Activity	EST	EET	LST	LET	Duration
A	0	2	2	4	2
B	1	3	3	5	2



Propagation 1

Activity	EST	EET	LST	LET	Duration
A	0	2	1	3	2
B	2	4	3	5	2

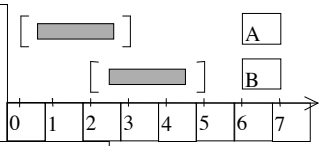


Figure 3. Propagation of the disjunctive constraint (non-preemptive case)

Example: Figure 3 displays two activities A and B which require the same resource of capacity 1. The earliest end time of each activity does not exceed its latest start time, so the timetable constraint cannot deduce anything. On the contrary, the propagation of the disjunctive constraint imposes $end(A) \leq start(B)$ which, in turn, results in updating both EST_B and LET_A.

3.2. The preemptive case

In the preemptive disjunctive case, the fact that activities A and B cannot overlap is most naturally represented by two alternative formulas:

$$set(A) \cap set(B) = \varnothing$$

or

$$\forall t, [W(A, t) = 0] \text{ or } [W(B, t) = 0].$$

When one of these formulas is adopted, the preemptive disjunctive constraints and the corresponding preemptive timetable constraints deduce the same time bounds. However, a simple rewriting of the non-preemptive disjunctive constraint

$$\begin{aligned} & [start(A) + duration(A) \leq end(B) - duration(B)] \\ \text{or} \quad & [start(B) + duration(B) \leq end(A) - duration(A)] \end{aligned}$$

suggests an additional preemptive disjunctive constraint:

$$\begin{aligned} & [start(A) + duration(A) + duration(B) \leq end(A)] \\ \text{or} \quad & [start(A) + duration(A) + duration(B) \leq end(B)] \\ \text{or} \quad & [start(B) + duration(A) + duration(B) \leq end(A)] \\ \text{or} \quad & [start(B) + duration(A) + duration(B) \leq end(B)] \end{aligned}$$

which can serve as a complement to $set(A) \cap set(B) = \varnothing$. Note that in the mixed case, the first (fourth) disjunct can be removed from the disjunction if A (respectively, B) cannot be interrupted.

Example: In the example of Figure 4, the propagation of the redundant constraint provides $start(A) \leq 1$ and $end(A) \geq 5$.

Before propagation

Activity	EST	EET	LST	LET	Duration
A	0	4	2	6	4
B	2	3	3	4	1

Propagation 1

Activity	EST	EET	LST	LET	Duration
A	0	5	1	6	4
B	2	3	3	4	1

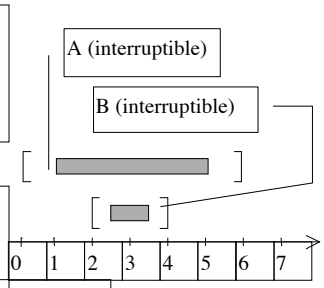


Figure 4. Propagation of the disjunctive constraint (preemptive case).

4. Edge-finding

4.1. The non-preemptive case

The term "edge-finding" is often used in non-preemptive disjunctive scheduling (Applegate & Cook, 1991), (Baptiste & Le Pape, 1995b), (Caseau & Laburthe, 1995b). It denotes both a "branching" and a "bounding" technique. The branching technique consists of ordering activities that require the same resource. At each node, a set of activities Ω is selected and, for each activity A in Ω, a new branch is created where A is constrained to execute first (or last) among the activities in Ω. The bounding technique consists of deducing that some activities from a given set Ω must, can, or cannot, execute first (or last) in Ω.

In the following, p_A denotes the minimal duration of A, EST_Ω the smallest of the earliest start times of the activities in Ω, LET_Ω the greatest of the latest end times of the activities in Ω, and p_Ω the sum of the minimal durations of the activities in Ω. Let $A \ll B$ ($A \gg B$) mean that A executes before (after) B and $A \ll \Omega$ ($A \gg \Omega$) mean that A executes before (after) all the activities in Ω. Once again, variants exist (Pinson, 1988), (Carlier & Pinson, 1990), (Carlier & Pinson, 1994), (Caseau & Laburthe, 1994), (Nuijten, 1994), (Brucker & Thiele, 1996), (Lévy, 1996), (Martin & Shmoys, 1996), but the following rules capture most of the edge-finding bounding technique:

$$\forall \Omega, \forall A \notin \Omega, [LET_{\Omega \cup \{A\}} - EST_\Omega < p_\Omega + p_A] \Rightarrow A \ll \Omega$$
$$\forall \Omega, \forall A \notin \Omega, [LET_\Omega - EST_{\Omega \cup \{A\}} < p_\Omega + p_A] \Rightarrow A \gg \Omega$$
$$A \ll \Omega \Rightarrow [end(A) \leq \min_{\Omega' \subseteq \Omega}(LET_{\Omega'} - p_{\Omega'})]$$
$$A \gg \Omega \Rightarrow [start(A) \geq \max_{\Omega' \subseteq \Omega}(EST_{\Omega'} + p_{\Omega'})]$$

If n activities require the resource, there are a priori $O(n * 2^n)$ pairs (A, Ω) to consider. An algorithm that performs all the time-bound adjustments in $O(n^2)$ is presented in (Carlier & Pinson, 1990). It consists of a "primal" algorithm to update earliest start times and a "dual" algorithm to update latest end times. The primal algorithm runs as follows:

- Compute "Jackson's preemptive schedule" (JPS) for the resource under consideration. JPS is the preemptive schedule obtained by applying the following priority rule: whenever the resource is free and one activity is available, schedule the activity A for which LET_A is the smallest. If an activity B becomes available while A is in process, stop A and start B if LET_B is strictly smaller than LET_A; otherwise continue A.
- For each activity A, compute the set Ψ of the activities which are not finished at $t = EST_A$ on JPS. Let p_B* be the residual duration on the JPS of the activity B at time t. Take the activities of Ψ in decreasing order of latest end times and select the first activity C such that:

$$EST_A + p_A + \sum_{B \in \Psi - \{A\} \mid LET_B \leq LET_C} (p_B^*) > LET_C$$

If such an activity C exists, then post the following constraints:

$$A \gg \{B \in \Psi - \{A\} \mid LET_B \leq LET_C\}$$
$$start(A) \geq \max_{B \in \Psi - \{A\} \mid LET_B \leq LET_C} (C_B^{JPS})$$

where C_B^{JPS} is the completion time of activity B in JPS.

Example: Figure 5 presents the JPS of a resource of capacity 1 required by 3 activities. On this example, the edge-finding propagation algorithm deduces $start(A) \geq 8$, when the timetable and the disjunctive constraint propagation algorithms deduce nothing.

	EST	LET	Duration
A	0	17	6
B	1	11	4
C	1	11	3

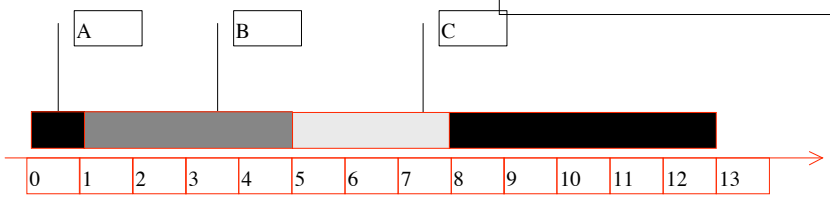


Figure 5. The Jackson's preemptive schedule of 3 activities A, B, C

Nuijten (1994) and Martin and Shmoys (1996) present variants of this algorithm, which also run in $O(n^2)$, but do not require the computation of Jackson's preemptive schedule. Carlier and Pinson (1994) present another variant, which runs in $O(n*log(n))$ but requires much more complex data structures. Baptiste (1995) and Martin and Shmoys (1996) establish an interesting property of the edge-finding technique: considering only the resource constraint and the current time bounds of activities, the algorithm computes the earliest start time at which each activity A could start if all the other activities were interruptible. This suggests a logical extension of the technique to preemptive and mixed cases: for each activity A requiring the resource, if A is not interruptible, the non-preemptive edge-finding bound applies; if A is interruptible then, considering only the resource constraint and the current time bounds, it would be nice to determine the earliest start and end times between which A could execute if all the activities were interruptible.

4.2. The preemptive case

Let us define $\rangle\rangle$ so that $A \rangle\rangle \Omega$ means "A ends after all activities in Ω" and substitute $\rangle\rangle$ for » in the rules of the primal algorithm.

$$\forall \Omega, \forall A \notin \Omega, [LET_\Omega - EST_{\Omega \cup \{A\}} < p_\Omega + p_A] \Rightarrow A \rangle\rangle \Omega$$
$$A \rangle\rangle \Omega \Rightarrow [start(A) \geq \max_{\Omega' \subseteq \Omega}(EST_{\Omega'} + p_{\Omega'})]$$

When A cannot be interrupted, these two rules remain valid (even if other activities can be interrupted) and the adjustment of EST_A is the same than in the non-preemptive case. When A can be interrupted, the first rule is still valid but the second is not. However, the second rule can be replaced by a weaker one:

$$A \rangle\rangle \Omega \Rightarrow [end(A) \geq \max_{\Omega' \subseteq \Omega}(EST_{\Omega' \cup \{A\}} + p_{\Omega' \cup \{A\}})]$$

This leads to a more general primal edge-finding algorithm:

- Compute Jackson's preemptive schedule JPS.
- For each activity A, compute the set Ψ of the activities which are not finished at $t = EST_A$ on JPS. Let p_B* be the residual duration on the JPS of the activity B at time t. Take the activities of Ψ in decreasing order of latest end times and select the first activity C such that:

$$EST_A + p_A + \sum_{B \in \Psi - \{A\} \mid LET_B \leq LET_C} (p_B^*) > LET_C$$

If such an activity C exists, then post the following constraints:

$A \rangle\rangle \{B \in \Psi - \{A\} \mid LET_B \leq LET_C\}$

$start(A) \geq \max_{B \in \Psi - \{A\} \mid LET_B \leq LET_C} (C_B^{JPS})$ if A cannot be interrupted

$end(A) \geq EST_A + p_A + \sum_{B \in \Psi - \{A\} \mid LET_B \leq LET_C} (p_B^*)$ if A can be interrupted

Example: In the example of Figure 5, the algorithm above deduces $start(A) \geq 8$ if A cannot be interrupted. It deduces $end(A) \geq 13$ if A can be interrupted.

It is proven in (Baptiste, 1995) that considering only the resource constraint and the current time bounds of activities, this algorithm computes:

- when A is not interruptible: the earliest time at which A could start if all the other activities were interruptible.
- when A is interruptible: the earliest time at which A could end if all the other activities were interruptible.

Nuijten's edge-finding algorithm can be modified in a similar fashion. The following algorithm is equivalent to the algorithm sketched above, assuming $ACTS$ is a list of the activities that require the resource, in increasing order of earliest start times, and $CONCLUDE(A, t_1, t_2)$ memorizes that A cannot end before t_1 if A is interruptible and that A cannot start before t_2 otherwise.

For each latest end time *emax* of an activity in *ACTS*
  $P \leftarrow 0$, $g \leftarrow -\infty$, $H \leftarrow -\infty$
  For A in *reverse*(*ACTS*)
    If $LET_A \leq emax$
    Then $P \leftarrow P + p_A$
      $g \leftarrow max(g, EST_A + P)$
      If ($emax < g$) raise a contradiction and exit
    $G_A \leftarrow g$
  For A in *ACTS*
    If $LET_A \leq emax$
    Then $H \leftarrow max(H, EST_A + P)$
      $P \leftarrow P - p_A$
    Else If ($EST_A + P + p_A > emax$)
      Then *CONCLUDE*(A, $EST_A + P + p_A$, G_A)
      If ($H + p_A > emax$)
      Then *CONCLUDE*(A, $H + p_A$, g)

The proof that this algorithm is equivalent to the JPS-based algorithm follows the proof of Nuijten's algorithm in (Nuijten, 1994). First, if A cannot be interrupted, the new algorithm makes the same conclusions than Nuijten's algorithm, so the proof in (Nuijten, 1994) applies to the new algorithm. Let us now assume that A can be interrupted. It is proven in (Baptiste, 1995) that the earliest time at which A could end if all the other activities could be interrupted is equal to the maximal value of $EST_{\Omega\cup\{A\}} + p_{\Omega\cup\{A\}}$ for Ω triggering the edge-finding rules. The earliest end times computed by the new algorithm are, when they are used, equal to $EST_{\Omega\cup\{A\}} + p_{\Omega\cup\{A\}}$ for such Ω. To prove that the best possible bound is reached, consider the two cases distinguished in (Nuijten, 1994): if A precedes all the activities of Ω in *ACTS*, either Ω or a superset of Ω is detected by the first test ($EST_A + P + p_A > emax$); if some activity of Ω precedes A in ACTS, either Ω or a superset of Ω is detected by the second test ($H + p_A > emax$). In both cases, a bound greater than or equal to $EST_{\Omega\cup\{A\}} + p_{\Omega\cup\{A\}}$ is found.

This algorithm can be further improved:

- When A can be interrupted and *set*(A) is known to contain a series of time intervals $I_1 \ldots I_n$, A can be replaced by (n + 1) activities $A_1 \ldots A_n$ A', with each A_i forced to execute over I_i and A' with the same earliest start time and latest end time than A and a duration equal to ($p_A - \Sigma_{1 \leq i \leq n}\ duration(I_i)$).
- When A can be interrupted and either ($EST_A + P = emax$) or ($H = emax$) in the course of the algorithm, it is certain that A cannot start before *emax*. Hence, the algorithm can also be used to update the earliest start times of interruptible activities.

Remark: When activities have fixed durations, the computation and the use of G_A and g to compute $\max_{\Omega'\subseteq\Omega}(EST_{\Omega'} + p_{\Omega'})$ serves only to avoid repeated iterations of the algorithm. Indeed, suppose a purely preemptive edge-finding algorithm is used and suppose A is not interruptible. The purely preemptive edge-finding algorithm uses the following rules:

$$\forall \Omega, \forall A \notin \Omega, [LET_\Omega - EST_{\Omega \cup \{A\}} < p_\Omega + p_A] \Rightarrow A \gg \Omega$$
$$A \gg \Omega \Rightarrow [end(A) \geq \max_{\Omega' \subseteq \Omega} (EST_{\Omega' \cup \{A\}} + p_{\Omega' \cup \{A\}})]$$

When constraint propagation stops, the earliest end time of A is set to a value EET_A such that if all activities were interruptible, there would be a schedule S of the resource such that (1) A does not start before EST_A and (2) A ends at EET_A. If the duration of A is fixed, the propagation of the duration constraint $start(A) + duration(A) = end(A)$ guarantees that when constraint propagation stops $EST_A + p_A = EET_A$. Consequently, A is not interrupted in S, which implies that the non-preemptive edge-finding algorithm cannot find a better bound for EST_A.

5. Flow constraints

Régin (1994) describes an algorithm, based on matching theory, to achieve the global consistency of the "all-different" constraint. This constraint is defined on a set of variables and constrains these variables to assume pairwise distinct values. Régin's algorithm maintains arc-consistency on the n-ary "all-different" constraint, which is shown to be more powerful than achieving arc-consistency for the $n*(n-1)/2$ corresponding binary "different" constraints.

Basically, Régin's algorithm consists of building a bipartite graph $G(X, Y, E)$ where X is a set of vertices corresponding to the variables of the "all-different" constraint, Y is a set of vertices corresponding to the possible values of these variables, and E is a set of edges (x, y), $x \in X$, $y \in Y$, such that $(x, y) \in E$ if and only if y is a possible value for x. As a result, the "all-different" constraint is satisfiable if and only if there exists a 0-1 function f on E such that:

$$\forall x \in X, \Sigma_{(x,y)\in E} f(x, y) = 1$$
$$\forall y \in Y, \Sigma_{(x,y)\in E} f(x, y) \leq 1$$

In addition, a given value y_j is a possible value for a given variable x_i if and only there exists a 0-1 function f_{ij} such that:

$$\forall x \in X, \Sigma_{(x,y)\in E} f_{ij}(x, y) = 1$$
$$\forall y \in Y, \Sigma_{(x,y)\in E} f_{ij}(x, y) \leq 1$$
$$f_{ij}(x_i, y_j) = 1$$

The problem of finding such a function (flow) f or f_{ij} can be solved in polynomial time. In addition, the current value of f can be used to generate f_{ij} at low cost, and to compute the new value of f when the domain of a variable changes. See Régin (1994, 1995, 1996) for details and extensions.

Notice that when all activities have duration 1, Régin's algorithm can be directly applied. In the preemptive case, this can be generalized to activities of arbitrary durations by seeing each activity A as $duration(A)$ sub-activities of duration 1. Then, each sub-activity has to pick a value (the time at which the sub-activity executes) and the values of the sub-activities that require a given resource have to be pairwise distinct. However, under this naive formulation, both the number of variables and the number of

values would be too high (dependent on the sum of the durations of the activities) for practical use. This led us to another formulation where the nodes x in X correspond to activities, and the nodes y in Y correspond to a partition of the time horizon in n disjoint intervals $I_1 = [s_1\ e_1)$ ... $I_n = [s_n\ e_n)$ such that $[s_1\ e_n)$ represents the complete time horizon, $e_i = s_{i+1}$ $(1 \leq i < n)$, and $\{s_1 \ldots s_n\ e_n\}$ includes all the time points at which the information available about $W(A, t)$ changes (Figure 6 illustrates this formulation on a small example). In particular, $\{s_1 \ldots s_n\ e_n\}$ includes all the earliest start times and latest end times of activities, but it can also include bounds of intervals over which $W(A, t)$ is constrained to be true or false (in this sense, the flow model is more general than preemptive edge-finding, but it does not generalize to the mixed case). E is defined as the set of pairs (x, y) such that activity x can execute during interval y. The maximal capacity $c_{max}(x, y)$ of edge (x, y) is set to $|y|$, and the minimal capacity $c_{min}(x, y)$ of edge (x, y) is set to $|y|$ if x is constrained to execute over y and to 0 otherwise. As a result, the preemptive resource constraint is satisfiable if and only if there exists a function f on E such that:

$$\forall x \in X, \Sigma_{(x,y)\in E} f(x, y) = duration(x)$$
$$\forall y \in Y, \Sigma_{(x,y)\in E} f(x, y) \leq |y|$$
$$\forall e \in E, c_{min}(e) \leq f(e) \leq c_{max}(e)$$

	EST	LET	Duration
A	0	10	5
B	2	4	1
C	4	6	1
D	6	8	1

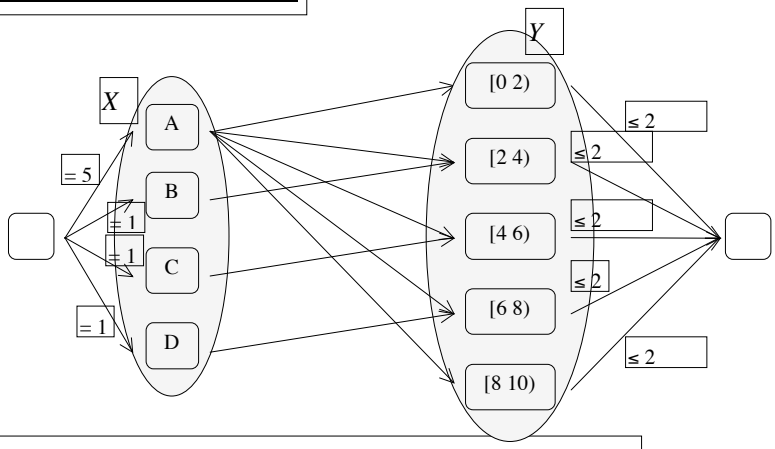


Figure 6. The bipartite graph corresponding to activites *A*, *B*, *C*, *D*

Similar models are commonly used in Operations Research. For example, Federgruen & Groenevelt (1986) use a more general model to solve particular polynomial scheduling problems with multiple parallel resources operating at different speeds. Following Régin (1994), what we propose below is to use network flow techniques, not only to find solutions to polynomial sub-problems, but also to update the domains of the variables.

Baptiste (1995) provides two algorithms for the search of a compatible flow f (SCF). The first algorithm uses Herz's algorithm, as described in (Gondran & Minoux, 1995), to construct the compatible flow, starting from $f(x, y) = 0$ for all x and all y. It runs in $O(|X| * |Y| * \Sigma_{x \in X} duration(x))$. The second algorithm builds a variant of Jackson's preemptive schedule which respects the intervals during which activities are required to execute. This can be done in $O(|Y| * log(|Y|))$. This schedule is then used as an initial (possibly incompatible) flow, repaired by Herz's algorithm in $O(|X| * |Y| * F)$, where F denotes the sum, over the activities, of the sizes of the intervals included in $[EST_A\ LET_A]$ during which the activity A is not allowed to execute (for reasons that are not directly related to the use of the resource by other activities).

To reduce variable domains, the most natural generalization of Régin's algorithm consists of varying $c_{min}(e)$ and $c_{max}(e)$ for each edge e in turn. The following algorithm updates the minimal flow $c_{min}(x, y)$ that can pass through an edge (x, y). The maximal flow $c_{max}(x, y)$ is obtained in a similar fashion.

- Set $u = c_{min}(x, y)$ and $v = c_{max}(x, y)$.
- While $(u \neq v)$
    - Set $w = \lfloor (u + v) / 2 \rfloor$
    - Search for a compatible flow f with $f(x, y) \leq w$.
    - If such a flow f exists, set $v = w$, otherwise set $u = w + 1$.
- Set $c_{min}(x, y) = u$.

It is proven in (Baptiste, 1995) that this adjustment of edge capacities (AEC) can be done for all edges (x, y) in $O(|X|^2 * |Y| * H)$, where H denotes the overall time horizon $e_n - s_1$. This complexity is reached by systematically reusing the previous flow as a start point when computing the flow f with the new constraint $f(x, y) \leq w$.

Then the following rules can be applied:

$$c_{max}(x, y) = 0 \Rightarrow \forall t \in y, W(x, t) = 0$$
$$c_{min}(x, y) = |y| \Rightarrow \forall t \in y, W(x, t) = 1$$
$$c_{min}(x, [s_i\ e_i)) \neq 0 \Rightarrow [start(x) \leq e_i - c_{min}(x, [s_i\ e_i))]$$
$$c_{min}(x, [s_i\ e_i)) \neq 0 \Rightarrow [end(x) \geq s_i + c_{min}(x, [s_i\ e_i))]$$

However, SCF and AEC are not sufficient to determine the best possible time bounds for activities. Let us consider, for example, the four activities A, B, C, D defined on Figure 6. In this case, $c_{min}(A, I)$ remains equal to 0 for all I; yet A cannot start after 3 and cannot end before 7. However, the flow model can be used to compute the best possible earliest end times. First, given x and the intervals $y_1 \ldots y_n$ (sorted in reverse chronological order) to which x is connected, one can find the maximal integer k such that there exists a compatible flow f with $f(x, y_i) = 0$ for $1 \leq i < k$. Then, one can compute the minimal flow $f_{min}(x, y_k)$ through (x, y_k), under the constraints $f(x, y_i) = 0$ for $1 \leq i < k$. Under these conditions, $end(x) \geq s_k + f_{min}(x, [s_k\ e_k))$ provides the best possible earliest end time for x. It is shown in (Baptiste, 1995) that this global update of time bounds (GUTB) can be done for all activities x in $O(|X|^2 * |Y| * H)$. As for AEC, this complexity is reached by systematically reusing the previous flow as a start point for computing the new flow when an additional capacity constraint is added.

Let us remark that the incrementality of Herz's algorithm is a key factor for both the worst-case and the practical complexity of SCF, AEC and GUTB. Strongly polynomial algorithms (with complexity independent of the schedule duration) could be used for the search of a compatible flow (Gondran & Minoux, 1995), but in practice the use of such non-incremental algorithms would probably make the SCF, AEC and GUTB algorithms less efficient.

6. Experimental study

6.1. The preemptive job-shop scheduling problem

To evaluate the constraint propagation algorithms presented in the preceding sections, we developed a branch and bound procedure for the preemptive job-shop scheduling problem (PJSSP), the variant of the job-shop scheduling problem (JSSP) in which all activities are interruptible. More precisely, one is given a set of jobs and a set of machines. Each job consists of a set of activities to be processed in a given order. Each activity is given an integer processing time and a machine on which it has to be processed. A machine can process at most one activity at a time. Activities may be interrupted at any time, an unlimited number of times. The problem is to find a schedule, i.e., a set of execution times for each activity, that minimizes the makespan, i.e., the time at which all activities are finished. The decision variant of the PJSSP is NP-complete in the strong sense (Garey & Johnson, 1979).

6.2. A dominance criterion

The most successful exact (branch and bound) approaches for the non-preemptive JSSP consist of ordering the set of activities $ACTS(M)$ which require the same machine M. At each node, a machine M and a set $\Omega \subseteq ACTS(M)$ are selected. For each activity A in Ω, a new branch is created where A is constrained to execute first (or last) among the activities in Ω. This decision is then propagated, through some variant of the edge-finding bounding technique (Carlier & Pinson, 1990), (Applegate & Cook, 1991), (Carlier & Pinson, 1994), (Baptiste & Le Pape, 1995b), (Caseau & Laburthe, 1995b).

For the PJSSP, this branching scheme is not valid since activities are interruptible, and thus cannot just be ordered. However, the dominance criterion introduced below allows the design of branching schemes which in a sense "order" the activities that require the same machine.

DEFINITION 1 *For any schedule S and any activity A, we define the "due date of A in S"* $d_S(A)$ *as:*

- *the makespan of S if A is the last activity of its job;*
- *the start time of the successor of A otherwise.*

DEFINITION 2 *For any schedule S, an activity A_k has priority over an activity A_l in S ($A_k <_S A_l$) if and only if either $d_S(A_k) < d_S(A_l)$ or $d_S(A_k) = d_S(A_l)$ and $k \leq l$. Note that $<_S$ is a total order.*

THEOREM 1 *For any schedule S, there exists a schedule J(S) such that:*

1. ***J(S) meets the due dates**: $\forall A$, the end time of A in J(S) is at most $d_S(A)$.*
2. ***J(S) is "active"**: $\forall M$, $\forall t$, if some activity $A \in ACTS(M)$ is available at time t, M is not idle at time t (where "available" means that the predecessor of A is finished and A is not finished).*
3. ***J(S) follows the $<_S$ priority order**: $\forall M$, $\forall t$, $\forall A_k \in ACTS(M)$, $\forall A_l \in ACTS(M)$, $A_l \neq A_k$, if A_k executes at time t, either A_l is not available at time t or $A_k <_S A_l$.*

Proof: We construct $J(S)$ chronologically. At any time t and on any machine M, the available activity that is the smallest (according to the $<_S$ order) is scheduled. $J(S)$ satisfies properties 2 and 3 by construction. Let us suppose $J(S)$ does not satisfy property 1. Let A denote the smallest activity (according to $<_S$) such that the end time of A in $J(S)$ exceeds $d_S(A)$. We claim that:

- the schedule of A is not influenced by the activities A_k with $A <_S A_k$ (by construction);
- for every activity $A_k <_S A$, the time at which A_k becomes available in $J(S)$ does not exceed the time at which A_k starts in S (because the predecessor of A_k is smaller than A).

Let M be the machine on which A executes. In $J(S)$, the activities $A_k \in ACTS(M)$ such that $A_k <_S A$ are scheduled in accordance with Jackson's rule, applied to the due dates $d_S(A_k)$. Since $d_S(A)$ is not met, and since Jackson's rule is guaranteed to meet due dates whenever it is possible to do so (cf. (Carlier & Pinson, 1990)), we deduce that it is **impossible** to schedule the activities $A_k \in ACTS(M)$ such that $A_k <_S A$ between their start times in S and their due dates in S. This is absurd since in S these activities **are** scheduled between their start times and their due dates. So, the hypothesis that $J(S)$ violates property 1 is contradicted.

Example: Figure 7 displays a schedule S and its "Jackson derivation" $J(S)$.

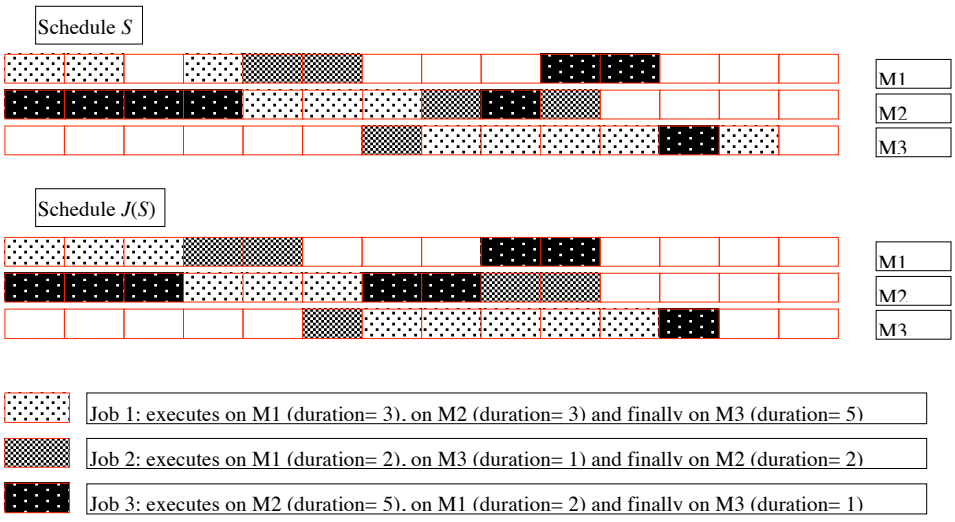


Figure 7. A preemptive schedule and its Jackson derivation

6.3. Branching scheme

We call $J(S)$ the "Jackson derivation" of S. Since the makespan of $J(S)$ does not exceed the makespan of S, at least one optimal schedule is the Jackson derivation of another schedule. Thus, in the search for an optimal schedule, we can impose the characteristics of a Jackson derivation to the schedule under construction. In this section, we present two branching procedures in which this result is used to solve the PJSSP.

Each of them is integrated in the following makespan minimization algorithm:

1. Compute an obvious upper bound UB of the makespan and an initial lower bound LB.
2. Select a value V in $[LB, UB)$.
3. Constrain the makespan to be lower than or equal to V and run the branching procedure. If a solution is found, set UB to the makespan of the solution; otherwise, i.e., if the search procedure fails, set LB to $V + 1$.
4. Iterate steps 2 and 3 until $UB = LB$.

The first branching scheme consists of ordering the activities according to an hypothetical $<_S$ order. For each machine M, an ordered list L_M of activities, initially empty, is developed as follows:

1. Select a machine M such that the set $K_M = ACTS(M) - L_M$ is not empty.
2. Select an activity A_k in K_M (e.g., the one with the smallest latest end time). Add A_k to the end of the list L_M. Use Jackson's rule to schedule the activities of L_M according to the L_M priority order and impose the resulting earliest end times. Keep the other activities of K_M as alternatives to be tried upon backtracking.
3. Iterate until all the activities are ordered or until all alternatives have been tried.

This branching scheme is attractive since it mimics the edge-finding branching technique that is often used in non-preemptive disjunctive scheduling. Yet, our first experiments have been disappointing. This led us to develop another branching scheme which more heavily exploits the dominance criterion.

1. Let t be the earliest date such that there is an activity A available (and not scheduled yet!) at t.
2. Compute K, the set of activities available at t on the same machine than A.
3. Compute NDK, the set of activities which are not "dominated" in K (as explained below).
4. Select an activity A_k in NDK (e.g., the one with the smallest latest end time). Schedule A_k to execute at t. Propagate the decision and its consequences according to the dominance criterion. Keep the other activities of NDK as alternatives to be tried upon backtracking.
5. Iterate until all the activities are scheduled or until all alternatives have been tried.

Needless to say, the power of this branching scheme highly depends on the rules that are used to (a) eliminate "dominated" activities in step 3 and (b) propagate "consequences" of the choice of A_k in step 4. The dominance criterion is exploited as follows:

- Whenever $A_k \in \mathit{ACTS}(M)$ is chosen to execute at time t, it is set to execute either up to its earliest end time or up to the earliest start time of another activity $A_l \in \mathit{ACTS}(M)$ which is not available at time t.
- Whenever $A_k \in K$ is chosen to execute at time t, any other activity $A_l \in K$ can be constrained not to execute between t and the end of A_k. At times $t' > t$, this reduces the set of candidates for execution: A_l is dominated by A_k, hence not included in NDK. In step 4, redundant constraints can also be added: $\mathit{end}(A_k) + \mathit{rp}_t(A_l) \leq \mathit{end}(A_l)$, where $\mathit{rp}_t(A_l)$ is the remaining processing time of A_l at time t; $\mathit{end}(A_k) \leq \mathit{start}(A_l)$ if A_l is not started at time t.
- If $A_k \in \mathit{ACTS}(M)$ is the last activity of its job, A_k is not candidate for execution at time t if another activity $A_l \in \mathit{ACTS}(M)$, which is not the last activity of its job, or such that $l < k$, is available at time t (A_k is dominated by A_l).

The proof that these reductions of the search space do not eliminate all optimal schedules follows from the fact that $J(S)$ schedules are dominant. Indeed, in a $J(S)$ schedule, (1) an activity cannot be interrupted unless a new activity becomes available on the same resource, (2) an activity A_k cannot execute when another activity A_l is available, unless $A_k <_S A_l$, and (3) we cannot have $A_k <_S A_l$ if A_k is the last activity of its job and either A_l is not the last activity of its job or $l < k$.

An open question at this point is whether there exists an optimal solution S such that $J(S) = S$. This would allow us to constrain the search even more. For example, as soon as an activity A_k would be given priority over an activity A_l, we could constrain the successor of A_l not to start before the successor of A_k. This could have a dramatic impact on the search space.

6.4. Experimental results

The second branching scheme was used to evaluate the various constraint propagation techniques developed in this paper. The disjunctive constraint $\mathit{set}(A) \cap \mathit{set}(B) = \emptyset$ and the flow-based algorithms, SCF, AEC, and GUTB, were implemented in ILOG SOLVER (Puget, 1994) on a RS6000 workstation. The mixed edge-finder was implemented in CLAIRE (Caseau & Laburthe, 1996b) on a PC Dell 200MHz running Windows NT.

Table 1 summarizes the results on 20 well-known instances of the job-shop scheduling problem. The first two columns indicate the version of the resource constraint that was used and the problem instance(s) under consideration. This can be a unique instance like "FT06" or, for "easy" instances, a series of instances similar in size and toughness, like "LA01 to LA10." In the latter case, the table provides average results over the whole series. All the instances we use are available from the job-shop directory in the OR benchmark library (http://www.ms.ic.ac.uk/info.html), except the CAR instances which can be found in the flow-shop directory.

Column “BT” provides the total number of backtracks needed to solve the problem. Column “CPU” provides the total CPU time in seconds, on a PC for the mixed edge-finder, and on an RS6000 for the other algorithms. Columns “BT(pr)” and “CPU(pr)” provide the number of backtracks and CPU time spent in proving that the optimal solution is, indeed, optimal. Results appear only when the considered version of the resource constraint enabled the branch and bound algorithm to solve the considered instance(s) in a reasonable amount of time. (For the smallest problems (FT06 to CAR4), at most 5000 backtracks were allowed for each iteration of the makespan minimization procedure.)

Table 1 shows that both the mixed edge-finder and the GUTB algorithm allow the resolution of “tough” problems like CAR5 (with optimal makespan 7667) and FT10 (900). Part of the differences between the edge-finder and the GUTB algorithm are due to differences in implementation, e.g., different computers and different sorting functions, so further comparison is not possible. Interestingly enough, the instances that appear the most difficult in the non-preemptive case, CAR5 and FT10 (Baptiste, 1994), are also the most difficult in the preemptive case.

Table 2 shows the results obtained by GUTB on the ten 10∗10 (i.e., 10 machines ∗ 10 jobs = 100 activities) instances used by Applegate and Cook (1991) in their computational study of the (non-preemptive) job-shop scheduling problem. Five of these instances (ABZ6, LA19, LA20, ORB2, and ORB5) were solved to optimality in a few hours of CPU time, one (FT10) was allowed more time to terminate, and four (ABZ5, ORB1, ORB3, and ORB4) remained open. For these instances, column “OPT” provides the best lower and upper bound that have been achieved. Otherwise, column “OPT” provides the value of the optimal makespan.

Table 1. Results obtained on 20 instances of the preemptive job-shop scheduling problem.

Constraint	Instances	BT	CPU	BT(pr)	CPU(pr)
Disjunctive	FT06	6353	3.5	4775	2.6
Edge-finder	FT06	3	0.1	2	0.0
	LA01-10	1	0.2	1	0.0
	CAR1-4	9	0.2	1	0.0
	CAR5	97927	582.6	26034	155.3
	CAR6-8	2870	23.4	937	7.5
	FT10	140903	2105.6	41255	624.0
SCF	FT06	24	0.3	21	0.1
	LA01-10	1196	9.2	1	0.0
AEC	FT06	5	0.5	2	0.1
	LA01-10	112	25.9	1	0.1
	CAR1-4	461	61.5	11	2.0
	CAR6-8	6947	1644.9	1403	351.3
GUTB	FT06	6	0.4	2	0.0
	LA01-10	9	11.0	1	0.0
	CAR1-4	27	13.0	1	0.1
	CAR5	73135	10295.8	19265	2673.7
	CAR6-8	3593	663.6	819	146.5
	FT10	254801	97585.7	49817	19626.6

Table 2. GUTB results on ten 10∗10 instances of the preemptive job-shop scheduling problem.

	OPT	BT	CPU	BT(pr)	CPU(pr)
FT10	900	254801	97585.7	49817	19626.6
ABZ5	1159 / 1219				
ABZ6	924	17578	3955.5	10879	2268.3
LA19	812	39286	7150.1	14184	2482.4
LA20	871	5494	1483.6	1627	463.8
ORB1	991 / 1054				
ORB2	864	56863	11199.2	20203	3835.3
ORB3	951 / 1254				
ORB4	977 / 980				
ORB5	849	16457	4721.3	4496	1296.6

Table 3. Edge-finding results on ten 10∗10 instances of the preemptive job-shop scheduling problem.

	OPT	BT	CPU	BT(pr)	CPU(pr)
FT10	900	140903	2105.6	41255	624.0
ABZ5	1203	1192553	15628.0	338597	4430.9
ABZ6	924	17699	307.8	8157	134.3
LA19	812	34637	564.3	10928	176.4
LA20	871	2779	59.4	998	22.7
ORB1	1035	347647	5182.4	85085	1278.3
ORB2	864	53127	709.4	16189	220.9
ORB3	973	6804127	96917.7	1947325	27884.2
ORB4	980	97654	1201.8	37122	461.3
ORB5	849	10380	158.6	4151	61.6

Table 3 provides the results obtained with the edge-finding algorithm on the same ten instances. All of these instances have been solved to optimality. Other instances we have solved include FT20 (in 0.4 second), LA11 to LA15 (0.4 second), LA16 (145 minutes), LA17 (1 second), LA18 (4 minutes), LA21 (65 hours), LA22 (4 seconds), LA23 (1 second), LA24 (44 hours), LA26 (1 second), LA28 (1 second), LA30 (1 second), LA31 to LA35 (4 seconds), LA37 (110 minutes), ORB6 (39 minutes), ORB7 (10 minutes), ORB8 (1 second), ORB9 (3 minutes), and ORB10 (1 minute). Let us note that, in the non-preemptive case, ORB3 also appears to be one of the most difficult

10∗10 instances (Applegate & Cook, 1991), (Baptiste & Le Pape, 1995b), (Caseau & Laburthe, 1995b), (Colombani, 1996). Such is not the case for LA16 (also a 10∗10) which is considered "easy" in the non-preemptive case.

7. Conclusion and perspectives

In this paper, we have presented a variety of constraint propagation techniques for preemptive disjunctive scheduling, some of which generalize nicely to the "mixed" case in which some activities can be interrupted and some cannot. Experimental results have shown that two of these techniques, (1) edge-finding and (2) global update of time bounds (GUTB), allow the resolution of hard instances such as the preemptive variant of the famous FT10. Let us remark that a combination of the two techniques is not likely to be useful when all the activities are interruptible and only time-bound constraints are imposed. Indeed, the characterization of the preemptive edge-finding algorithm proves that the best possible bounds are obtained. A combination might however be useful in more complex situations: on the one hand, the mixed edge-finding algorithm explicitly deals with non-interruptible activities, and thus can be more efficiently applied to the mixed case; on the other hand, if an interruptible activity cannot execute during some time intervals, the GUTB algorithm can take these intervals into account.

These results encourage us (and hopefully will encourage other researchers) to pursue work in the application of constraint programming to preemptive and mixed scheduling problems:

- Until now, most of our efforts have been focused on constraint propagation. More work is needed to evaluate the interest of different heuristics and branching strategies. Based on our results, the PJSSP currently appears to be much harder than the non-preemptive JSSP. An important reason for this is that we have not been able to reuse the concept of "bottleneck resource" in an efficient way. An open question is how the "bottleneck" concept can be used, without throwing away the dominance criterion which appears crucial in reducing the size of the search tree.
- Most of the results presented in this paper concern resources of capacity 1. More work is needed to generalize these techniques to resources of arbitrary capacity.
- Other constraint propagation techniques, such as energetic reasoning (Lopez, 1991) or shaving (Carlier & Pinson, 1994), (Martin & Shmoys, 1996), can be worth investigating.

Acknowledgments

Part of the work presented in this paper was done while the second author was finishing a Master's thesis at ILOG S.A. The authors want to thank Jean-François Puget, Michel Leconte, Wim Nuijten, Jean-Charles Régin, Henri Béringer, Jacques Carlier, Yves Caseau, François Laburthe, Bruno de Backer, and Ulrich Junker, for many enlightening discussions on constraints, flows, and edge-finding. We also thank the referees for their numerous comments, which hopefully led to significant improvements of this paper.

Note

1. Given a constraint c over n variables $v_1 \ldots v_n$ and a domain D_i for each variable v_i, c is "arc-consistent" if and only if for any variable v_i and any value val_i in the domain of v_i, there exist values $val_1 \ldots val_{i-1}\ val_{i+1} \ldots val_n$ in $D_1 \ldots D_{i-1}\ D_{i+1} \ldots D_n$ such that $c(val_1 \ldots val_n)$ holds. Arc-B-consistency, where B stands for bounds, guarantees only that $val_1 \ldots val_{i-1}\ val_{i+1} \ldots val_n$ exist for val_i equal to either the smallest or the greatest value in D_i.

References

Aggoun, A. & Beldiceanu, N. (1993). "Extending CHIP in Order to Solve Complex Scheduling and Placement Problems," Mathematical and Computer Modelling 17, 57-73.

Applegate, D. & Cook, W. (1991). "A Computational Study of the Job-Shop Scheduling Problem," ORSA Journal on Computing 3, 149-156.

Baptiste, Ph. (1994). "Constraint-Based Scheduling: Two Extensions," MSc Thesis, University of Strat(P) -6 cl.7599c24 (t) -22 (") -5 (C) -7 (o

Caseau, Y. & Laburthe, F. (1996b). "CLAIRE: A Parametric Tool to Generate C++ Code for Problem Solving," Working Paper, Bouygues, Direction Scientifique, Saint-Quentin-en-Yvelines, France.

Cheng, C.-C. & Smith, S.F. (1994). "Generating Feasible Schedules under Complex Metric Constraints," Proc. 12th National Conference on Artificial Intelligence, 1086-1091, MIT Press.

Cheng, C.-C. & Smith, S.F. (1995a). "Applying Constraint Satisfaction Techniques to Job-Shop Scheduling," Technical Report, Carnegie Mellon University, Pittsburgh, Pennsylvania.

Cheng, C.-C. & Smith, S.F. (1995b). "A Constraint-Posting Framework for Scheduling under Complex Constraints," Proc. AAAI-SIGMAN Workshop on Intelligent Manufacturing Systems, 64-75.

Collinot, A. & Le Pape, C. (1987). "Controlling Constraint Propagation," Proc. 10th International Joint Conference on Artificial Intelligence, 1032-1034, Morgan Kaufmann.

Colombani, Y. (1996). "Constraint Programming: An Efficient and Practical Approach to Solving the Job-Shop Problem," Proc. 2nd International Conference on Principles and Practice of Constraint Programming, 149-163, Springer-Verlag.

Demeulemeester, E. (1992). "Optimal Algorithms for Various Classes of Multiple Resource-Constrained Project Scheduling Problems," PhD Thesis, Katholieke Universiteit Leuven, Leuven, Belgium.

Erschler, J. (1976). "Analyse sous contraintes et aide à la décision pour certains problèmes d'ordonnancement," Thèse de Doctorat d'Etat, Université Paul Sabatier, Toulouse, France (in French).

Erschler, J., Lopez, P., & Thuriot, C. (1991). "Raisonnement temporel sous contraintes de ressource et problèmes d'ordonnancement," Revue d'Intelligence Artificielle 5, 7-32 (in French).

Esquirol, P. (1987). "Règles et processus d'inférence pour l'aide à l'ordonnancement de tâches en présence de contraintes," PhD Thesis, Université Paul Sabatier, Toulouse, France (in French).

Federgruen, A. & Groenevelt, H. (1986). "Preemptive Scheduling of Uniform Machines by Ordinary Network Flow Techniques," Management Science 32, 341-349.

Fox, B.R. (1990). "Chronological and Non-Chronological Scheduling," Proc. 1st IEEE Annual Conference on Artificial Intelligence, Simulation and Planning in High Autonomy Systems.

Garey, M.R. & Johnson, D.S. (1979). "Computers and Intractability. A Guide to the Theory of NP-Completeness," W. H. Freeman and Company.

Gondran, M. & Minoux, M. (1995). "Graphes et Algorithmes," Eyrolles (in French).

Kumar, V. (1992). "Algorithms for Constraint Satisfaction Problems: A Survey," AI Magazine 13, 32-44.

Laborie, P. (1994). "Planifier avec des contraintes de ressources," Proc. 2èmes rencontres des jeunes chercheurs en intelligence artificielle (in French).

Le Pape, C. & Smith, S.F. (1987). "Management of Temporal Constraints for Factory Scheduling," Proc. IFIP TC 8/WG 8.1 Working Conference on Temporal Aspects in Information Systems, 159-170, North-Holland.

Le Pape, C. (1988). "Des systèmes d'ordonnancement flexibles et opportunistes," PhD Thesis, University Paris XI, Orsay, France (in French).

Le Pape, C. (1991). "Constraint Propagation in Planning and Scheduling," Technical Report, Stanford University, Palo Alto, California.

Le Pape, C. (1994). "Implementation of Resource Constraints in ILOG SCHEDULE: A Library for the Development of Constraint-Based Scheduling Systems," Intelligent Systems Engineering 3, 55-66.

Le Pape, C. (1996). "An Application of Constraint Programming to a Specific Production Scheduling Problem," Belgian Journal of Operations Research, Statistics and Computer Science (to appear).

Lévy, M.-L. (1996). "Méthodes par décomposition temporelle et problèmes d'ordonnancement," PhD Thesis, Institut National Polytechnique de Toulouse, Toulouse, France (in French).

Lhomme, O. (1993). "Consistency Techniques for Numeric CSPs," Proc. 13th International Joint Conference on Artificial Intelligence, 232-238, Morgan Kaufmann.

Lock, H.C.R. (1996). "An Implementation of the Cumulative Constraint," Working Paper, University of Karlsruhe, Karlsruhe, Germany.

Lopez, P. (1991). "Approche énergétique pour l'ordonnancement de tâches sous contraintes de temps et de ressources," PhD Thesis, Université Paul Sabatier, Toulouse, France (in French).

Lopez, P., Erschler, J., & Esquirol, P. (1992). "Ordonnancement de tâches sous contraintes : une approche énergétique," RAIRO APII 26, 453-481 (in French).

Martin, P. & Shmoys, D.B. (1996). "A New Approach to Computing Optimal Schedules for the Job-Shop Scheduling Problem," Proc. 5th International Conference on Integer Programming and Combinatorial Optimization.

Nuijten, W.P.M. & Aarts, E.H.L. (1994). "Constraint Satisfaction for Multiple Capacitated Job-Shop Scheduling," Proc. 11th European Conference on Artificial Intelligence, 635-639, John Wiley and Sons.

Nuijten, W.P.M. (1994). "Time and Resource Constrained Scheduling: A Constraint Satisfaction Approach," PhD Thesis, Eindhoven University of Technology, Eindhoven, The Netherlands.

Nuijten, W.P.M. & Aarts, E.H.L. (1996). "A Computational Study of Constraint Satisfaction for Multiple Capacitated Job-Shop Scheduling," European Journal of Operational Research 90, 269-284.

Pegman, M., Forward, N., King, B., & Teal, D. (1997). "Mine Planning and Scheduling at RTZ Technical Services," Proc. 3rd International Conference and Exhibition on the Practical Application of Constraint Technology, 273-285, The Practical Application Company.

Pinson, E. (1988). "Le problème de job-shop," PhD Thesis, University Paris VI, Paris, France (in French).

Prosser, P. (1990). "Distributed Asynchronous Scheduling," PhD Thesis, University of Strathclyde, Glasgow, United Kingdom.

Puget, J.-F. (1994). "A C++ Implementation of CLP," Technical Report, ILOG S.A., Gentilly, France.

Puget, J.-F. & Leconte, M. (1995), "Beyond the Glass Box: Constraints as Objects," Proc. International Symposium on Logic Programming, MIT Press.

Régin, J.-C. (1994). "A Filtering Algorithm for Constraints of Difference in CSPs," Proc. 12th National Conference on Artificial Intelligence, 362-367, MIT Press.

Régin, J.-C. (1995). "Développement d'outils algorithmiques pour l'Intelligence Artificielle. Application à la chimie organique," PhD Thesis, University Montpellier II, Montpellier, France (in French).

Régin, J.-C. (1996). "Generalized Arc-Consistency for Global Cardinality Constraint" Proc. 13th National Conference on Artificial Intelligence, 209-215, MIT Press.

Rit, J.-F. (1986). "Propagating Temporal Constraints for Scheduling," Proc. 5th National Conference on Artificial Intelligence, 383-388, MIT Press.

Smith, S.F. (1983). "Exploiting Temporal Knowledge to Organize Constraints," Technical Report, Carnegie Mellon University, Pittsburgh, Pennsylvania.

Smith, S.F. (1992). "Knowledge-Based Production Management: Approaches, Results and Prospects," Production Planning and Control 3, 350-380.

Smith, S.F. & Cheng, C.-C. (1993). "Slack-Based Heuristics for Constraint Satisfaction Scheduling," Proc. 11th National Conference on Artificial Intelligence, 139-144, MIT Press.

Smith, S.F. (1994). "OPIS: A Methodology and Architecture for Reactive Scheduling," In: Zweben, M. & Fox, M. (editors), "Intelligent Scheduling," Morgan Kaufmann.

Varnier, C., Baptiste, P., & Legeard, B. (1993). "Le traitement des contraintes disjonctives dans un problème d'ordonnancement : exemple du Hoist Scheduling Problem," Proc. 2èmes journées francophones de programmation logique, 343-363 (in French).

Zweben, M., Davis, E., Daun, B., & Deale, M.J. (1993). "Scheduling and Rescheduling with Iterative Repair," IEEE Transactions on Systems, Man, and Cybernetics 23, 1588-1596.