A Theoretical and Experimental Comparison of Constraint Propagation Techniques for Disjunctive Scheduling

Philippe Baptiste
ILOG S.A.
2 Avenue Galliéni BP 85
94253 Gentilly Cedex FRANCE
baptiste@ilog.fr

Claude Le Pape
ILOG S.A.
2 Avenue Galliéni BP 85
94253 Gentilly Cedex FRANCE
lepape@ilog.fr

Abstract

Disjunctive constraints are widely used to ensure that the time intervals over which two activities require the same resource cannot overlap: if a resource is required by two activities A and B, the disjunctive constraint states that either A precedes B or B precedes A. The "propagation" of disjunctive constraints consists in determining cases where only one of the two orderings is feasible. It results in updating the time-bounds of the two activities. The standard algorithm for propagating disjunctive constraints achieves arc-B-consistency. Two types of methods that provide more precise time-bounds are studied and compared. The first type of method consists in determining whether an activity A must, can, or cannot be the first or the last to execute among a set of activities that require the same resource. The second consists in comparing the amount of "resource energy" required over a time interval $[t_1\ t_2)$ to the amount of energy that is available over the same interval. The main result of the study is an implementation of the first method in Ilog Schedule, a generic tool for constraint-based scheduling which exhibits performance in the same range of efficiency as specific operations research algorithms.

1 Introduction

Disjunctive constraints are widely used to ensure that the time intervals over which two activities require the same resource do not overlap in time [Erschler, 1976; Carlier, 1984; Le Pape, 1988; Smith and Cheng, 1993]. If a resource is required by two activities A and B, the disjunctive constraint states that either A precedes B (i.e., ends before B starts) or B precedes A. The "propagation" of disjunctive constraints consists in determining that only one of the two orderings is feasible. It results in updating the time-bounds (earliest start times and latest end times) of the two activities.

The most basic disjunctive constraint can be stated as follows:

$$[end(A) \leq start(B)]\ or\ [end(B) \leq start(A)]$$

In this formula, $start(A)$, $end(A)$, $start(B)$ and $end(B)$ denote **constrained variables**, which means that the values of $start(A)$, $end(A)$, $start(B)$ and $end(B)$ are initially unknown.

Constraint propagation consists in reducing the set of possible values for these variables: whenever the smallest possible value of $end(A)$ (earliest end time of A) exceeds the greatest possible value of $start(B)$ (latest start time of B), A cannot precede B; hence B must precede A; the time-bounds of A and B can consequently be updated with respect to the new temporal constraint $[end(B) \leq start(A)]$. Similarly, when the earliest possible end time of B exceeds the latest possible start time of A, B cannot precede A. When neither of the two activities can precede the other, a contradiction is detected. This way of propagating constraints enforces arc-B-consistency [Lhomme, 1993], i.e., arc-consistency restricted to updating the bounds of the variables representing the start times and the end times of activities. A constraint $c(v_1, ..., v_n)$ is said to be 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, ..., val_{i-1}, val_{i+1}, ..., val_n, in the domains of v_1, ..., v_{i-1}, v_{i+1}, ..., v_n, such that $c(val_1, ..., val_n)$ holds. Arc-B-consistency (where B stands for *bounds*) guarantees only that val_1, ..., val_{i-1}, val_{i+1}, ..., val_n exist for val_i equal to either the smallest or the greatest value in the domain of v_i. [Nuijten *et al.*, 1993a] provides a characterization of full arc-consistency when durations of activities are fixed.

In practice, two types of extensions of the basic disjunctive constraint are useful:

- Extensions allowing the representation of more complex constraints, including (1) activities that may or may not require the resource, (2) "state resources" to represent situations where an activity uses a resource only under specific conditions, and (3) setup times between activities that require the same resource. [Baptiste and Le Pape, 1995] describes such extensions.
- Extensions of the constraint propagation algorithm to deduce more precise time-bounds. When the amount of propagation is extended, the search space is more drastically pruned. Each decision requires more CPU time, but the number of problem-solving steps decreases. Results of experiments with flexi-

ble constraint propagation systems show that the amount of constraint propagation that enables a problem-solver to be the most efficient varies with the problem-solver, with the application, and with the problem-solving context [Van Hentenryck, 1989; Collinot and Le Pape, 1991]. The following sections present the results of an investigation of the techniques that can be used to extend the propagation of disjunctive constraints.

2 Bibliographical Study

In the literature, there are roughly two types of methods that provide more precise time-bounds. The first type of method, described in Section 2.1, consists in determining whether an activity A **must**, **can**, or **cannot** be the first or the last to execute among a set of activities S that require the same resource [Carlier and Pinson, 1990; Nuijten *et al.*, 1993b; Caseau and Laburthe, 1994]. The second type of method consists in comparing the amount of resource energy required over a time interval $[t_1\ t_2)$ to the amount of energy that is available over the same interval [Erschler *et al.*, 1991; Beck, 1992; Le Pape, 1994]. This form of propagation is discussed in Section 2.2.

2.1 Automatic Sequencing of Activities

Rather than just looking at pairs of activities $\{A\ B\}$, the first type of method compares the temporal characteristics of A to those of a set of activities Ω.

Let est_A denote the earliest possible start time of A, let_A denote the latest possible end time of A, and p_A denote the processing time of A (the smallest possible processing time of A if the processing time of A is not fixed). Let est_Ω denote the smallest of the earliest start times of the activities in Ω, let_Ω denote the greatest of the latest end times of the activities in Ω, and p_Ω denote the sum of the processing times of the activities in Ω. Let $A \prec B$ ($A \succ B$) mean that A is before (after) B and $A \prec \Omega$ ($A \succ \Omega$) mean that A is before (after) all the activities in Ω. The following rules apply:

$$\left[\begin{array}{l} let_\Omega - est_\Omega < p_A + p_\Omega \\ let_A - est_\Omega < p_A + p_\Omega \end{array}\right] \Rightarrow [A \prec \Omega]$$

$$\left[\begin{array}{l} let_\Omega - est_\Omega < p_A + p_\Omega \\ let_\Omega - est_A < p_A + p_\Omega \end{array}\right] \Rightarrow [A \succ \Omega]$$

New time-bounds can consequently be deduced. When A is before all activities in Ω, the end time of A is necessarily smaller than (or equal to) $let_\Omega - p_\Omega$. When A is after all activities in Ω, the start time of A is necessarily greater than (or equal to) $est_\Omega + p_\Omega$.

The technique which consists in applying these rules is known as **edge-finding** [Applegate and Cook, 1991]. (Actually, Applegate and Cook use the term "edge-finding" to denote both the above type of deduction, i.e., a "bounding" technique, and the non-deterministic choice between the possible orderings of activities, i.e., a "branching" technique. In the following, edge-finding is considered only as a deterministic deductive "bounding" technique.)

Notice that if n activities require the resource, there are potentially $O(n * 2^n)$ pairs $(A\ \Omega)$ to consider. An algorithm that performs all time-bound adjustments in $O(n^2)$ is presented in [Carlier and Pinson, 1990]:

- Compute "Jackson's preemptive schedule" for the resource under consideration. "Preemptive" means that an activity A can be interrupted to process another activity B. Jackson's preemptive schedule (JPS) is obtained by applying a very simple 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_i, compute the set Ψ of the activities which are not finished at $t = est_{A_i}$ on JPS. Let p_k^* be the residual duration on the preemptive schedule of the activity A_k at time $t = est_{A_i}$. Take the activities of Ψ in decreasing order of due dates and select the first activity A_j such that:

  $$est_{A_i} + p_{A_i} + \sum_{A_k \in \Psi / let_{A_k} \leq let_{A_j}} p_k^* > let_{A_j}$$

  If such an A_j exists, then post the new temporal constraints:

  $$\forall A_k \in \Psi\ let_{A_k} \leq let_{A_j} \Rightarrow A_k \prec A_i$$

  $$est_{A_i} \geq \max_{A_k \in \Psi / let_{A_k} \leq let_{A_j}} C_{A_k}^{JPS}$$

  where $C_{A_k}^{JPS}$ is the completion time of the activity A_k in JPS.

[Nuijten *et al.*, 1993b] presents a variant of this algorithm, which also runs in $O(n^2)$, but is simpler in the sense that it does not require the computation of Jackson's preemptive schedule.

[Caseau and Laburthe, 1994] presents a constraint-based technique based on the same principles. Two sets of rules based on the concept of "task intervals" are developed:

- **edge-finding within a task interval:** The edge-finding technique is very powerful but, as we have said, it cannot be naively applied to all the subsets Ω of activities requiring the same resource (since for n activities, this would lead to 2^n sets). To avoid this, Caseau and Laburthe associate to all couples of activities $(A\ B)$ scheduled on the same machine the set of activities $K_{(A\ B)}$ which will surely be scheduled between the earliest start time of A and the latest end time of B:

  $$K_{(A\ B)} = \{C / est_A \leq est_C \ and \ let_C \leq let_B\}$$

  $K_{(A\ B)}$ is called a task interval and is considered of interest only when A and B belong to $K_{(A\ B)}$. Deductive edge-finding propositions are applied to task intervals: for each activity C in $K_{(A\ B)}$, the rules determine whether C necessarily executes before (or after) all the other activities in $K_{(A\ B)}$.
- **edge-finding between a task interval and another task:** this second set of rules is used to deduce that a task C cannot be performed before (or

after) a task interval $K_{(A\ B)}$ to which it does not belong.

$$let_B - est_C - p_C - \sum_{D \in K_{(A\ B)}} p_D < 0 \Rightarrow C \not\prec K_{(A\ B)}$$

$$let_C - est_A - p_C - \sum_{D \in K_{(A\ B)}} p_D < 0 \Rightarrow C \not\succ K_{(A\ B)}$$

Time-bounds can be updated accordingly. For example, if C cannot be before $K_{(A\ B)}$, C is at least after one of the activities D of $K_{(A\ B)}$. Therefore, the earliest start time of C can be updated according to the following equation:

$$est_C = \max(est_C, \min_{D \in K_{(A\ B)}} (est_D + p_D))$$

A few remarks can be made about these rules. First, there are at most $O(n^2)$ task intervals to consider for a resource on which n activities are scheduled. Therefore, a computational time in $O(n^3)$ is, in the worst case, necessary to apply the rules to all the task intervals and all the activities of a given resource. In comparison, the edge-finding algorithm of Carlier and Pinson runs in $O(n^2)$. However, the implementation of task intervals is simpler (e.g., no need to compute Jackson's preemptive schedule) and can be applied in an incremental way [Caseau and Laburthe, 1994], so it is difficult to determine for which values of n Carlier and Pinson's algorithm will be the most efficient.

Another remark is that when the second set of rules is applied only to task intervals, the overall propagation system (which includes those rules as well as others) can lead to different results depending on the order in which constraints are propagated. In general, a set of propagation rules is said to have a "unique fixpoint" when the results of the overall propagation do not depend on the order in which the propagation steps are executed. Applying the second set of rules of Caseau and Laburthe to task intervals only "breaks" the "unique fixpoint semantics" of constraint propagation. This means that, if other propagation rules apply, the "fixpoint" that it is reached may depend on the propagation order. This fact is illustrated below on a simple example.

Let A, B, C, D be four activities to schedule on a unary resource with respect to the following release and due dates:

Activity	Duration	Release date	Due date
A	3	6	14
B	3	7	15
C	1	0	20
D	2	8	20

The following figure displays the $[est_X\ let_X)$ intervals of the four activities.

```
- - - - - - * * * * * * * * - - - - - - - -   A (3)
- - - - - - - * * * * * * * * - - - - - - -   B (3)
* * * * * * * * * * * * * * * * * * * * - -   C (1)
- - - - - - - - * * * * * * * * * * * * - -   D (2)
```

Let us try now to update the time-bounds according to the previous rules.

- $K_{(A\ A)} = \{A\}$. No deduction.
- $K_{(A\ B)} = \{A, B\}$. Let us try to deduce something with D:

  $$[15 - 8 - 2 - (3 + 3) = -1 < 0] \Rightarrow D \not\prec K_{(A\ B)}$$

  $$[20 - 6 - 2 - (3 + 3) = 6 \geq 0] \Rightarrow \textit{no deduction}$$

  D cannot be before $K_{(A\ B)} = \{A, B\}$. So est_D is updated to $\min(est_A + p_A, est_B + p_B) = 9$.
- $K_{(A\ C)} = \{A, B, D\}$. No deduction.
- ...

Let us restart the example with $est_C = 7$ and $let_C = 14$, i.e., assuming other propagation rules, related to other constraints, have allowed the deductions that C cannot start before 7 and cannot end after 14. These new values reduce the domain of the possible dates of execution of C. Therefore, the propagation should deduce at least as much new information as before. Otherwise, we would get different fixpoints by applying the different propagation rules in different orders.

```
- - - - - - * * * * * * * * - - - - - - - -   A (3)
- - - - - - - * * * * * * * * - - - - - - -   B (3)
- - - - - - - * * * * * * * - - - - - - - -   C (1)
- - - - - - - - * * * * * * * * * * * * - -   D (2)
```

- $K_{(A\ A)} = \{A\}$. No deduction.
- $K_{(A\ B)} = \{A, B, C\}$. Let us try to deduce something with D:

  $$[15 - 8 - 2 - (3 + 3 + 1) = -2 < 0] \Rightarrow D \not\prec K_{(A\ B)}$$

  $$[20 - 6 - 2 - (3 + 3 + 1) = 5 \geq 0] \Rightarrow \textit{no deduction}$$

  D cannot be before $K_{(A\ B)} = \{A, B, C\}$. So est_D is updated to $\min(est_A + p_A, est_B + p_B, est_C + p_C) = 8$. The deduction $est_D = 9$ is not done!
- $K_{(A\ C)} = \{A, C\}$. No deduction.
- ...

The counterexample shows that the second set of rules, breaks "unique fixpoint semantics" when applied only to task intervals. Having different possible fixpoints is not a real problem when developing a specific algorithm, but may be annoying in the context of a generic constraint programming tool. Indeed, it means that a user of the tool can face different execution behaviors depending on the order in which constraints are posted. Such variability makes it less easy to use the tool for the development of complex scheduling applications.

2.2 Energy-Based Reasoning

The second type of method consists in comparing the amount of resource energy required over a time interval $[t_1\ t_2)$ to the amount of energy that is available over the same interval.

In [Erschler *et al.*, 1991], the authors analyze the effect of time and resource constraints on the admissibility of schedules. They study how activity characteristics and resource constraints can induce new constraints which allow to restart the propagation. Although Erschler, Lopez and Thuriot have worked on discrete resources (i.e., resources the capacity of which can be greater than one), from now on, we focus on the case of unary resources (i.e., resources of capacity 1).

Let us define the concept of **required energy consumption** $W_A^{[t_1\ t_2)}$ of an activity A over an interval $[t_1\ t_2)$. The required energy consumption is the smallest amount of time during which A will be executed on the interval:

$$W_A^{[t_1\ t_2)} = \min(p_A, t_2 - t_1, est_A + p_A - t_1, t_2 - let_A + p_A)$$

Of course, $W_A^{[t_1\ t_2)}$ is null if the previous quantity is negative. Two deduction rules apply:

- The first deduction rule is based upon the idea of picking two activities A and B and trying to find a contradiction when sequencing A before B. To achieve this, the required consumption of all the activities over the interval $[est_A\ let_B)$ is computed and compared to the provided amount of resource during the same interval.

$$let_B - est_A < \sum_{C \notin \{A\ B\}} W_C^{[est_A\ let_B)} + p_A + p_B \Rightarrow B \prec A$$

- The second deduction rule directly updates time-bounds. Let us choose an activity A and an integer x in the interval $[est_A\ let_A)$. We are going to check on the interval $[est_A\ x)$ whether A can or cannot start at its earliest start time. If it cannot, then est_A will be increased. Let us first define the quantity $\Sigma_W = \sum_{C \neq A} W_C^{[est_A\ x)} + \min(p_A, x - est_A)$. The deduction rule is:

$$x - est_A < \Sigma_W \Rightarrow est_A = est_A + \sum_{C \neq A} W_C^{[est_A\ x)}$$

  In fact, Erschler, Lopez and Thuriot use a weaker bound for the earliest end time of the activity: $x + \Sigma_W - (x - est_A)$. For a non-interruptible activity with fixed duration, this leads to the new value $est_A + \sum_{C \neq A} W_C^{[est_A\ x)} + \min(p_A, x - est_A) - p_A$ for est_A. The rationale for this bound is that a minimal energy of $\Sigma_W - (x - est_A)$ must be flushed out of the $[est_A\ x)$ interval. However, when $x - est_A$ is smaller than p_A, then $p_A - (x - est_A)$ energy is already consumed out of the $[est_A\ x)$ interval. Hence, it is correct to replace $x + \Sigma_W - (x - est_A)$ by $x + \sum_{C \neq A} W_C^{[est_A\ x)} + p_A - (x - est_A)$, which leads to the bound given above.

  Obviously, there are many values of x for which this rule could be used. Erschler, Lopez and Thuriot suggest that the appropriate values of x are the earliest and latest start and end times of activities (which seems reasonable but is yet to be proven).

These rules are quite efficient. In a simple experiment, we combined the first of these rules with the edge-finding technique of Carlier and Pinson restricted to the overall set of yet unordered operations. With the resulting algorithm, a *very hard* job-shop scheduling problem known as MT10 was solved to optimality (by branch-and-bound with constraint propagation performed at each node of the search tree) in about eight hours of CPU time on a HP715/50 workstation. Eight hours is significantly more than the times obtained in [Carlier and Pinson, 1990] and [Caseau and Laburthe, 1994] for the same problem; yet the result struck us given that our algorithm was extremely simple.

While using this algorithm, we noticed that the number of backtracks required to prove the optimality of the solution varied with the order in which constraints were posted. This observation suggested that our implementation did not have a unique fixpoint. Yet we have been unable to put together a simple example with two different fixpoints.

Energetic reasoning is also used in the Tosca system of Beck [Beck, 1992] and in the energetic resources of Ilog Schedule [Le Pape, 1994]. Beck uses a data structure called "habograph" to compute the required energy consumption and the maximal energy available over each time interval $[t_1\ t_2)$ up to a given discretization of time. When the required energy consumption exceeds the maximal energy available, a failure occurs. When the required energy consumption is close to the maximal energy available, a constraint threat is indicated. The habograph can also be used to determine that an activity cannot fully execute during a given time interval $[t_1\ t_2)$, in a way similar to applying the rules above, but with predetermined time intervals rather than intervals depending on the time-bounds of activities.

In Ilog Schedule, an energetic resource represents the fact that the amount of energy that can be used over given intervals of time is limited. Roughly, an energetic resource does not represent a resource per se, but the amount of work performed by the resource over regular intervals of time. The capacity of an energetic resource is consequently not measured as a number of machines or as a number of men and women, but as a number of machining hours, or as a number of human-days, human-weeks or human-months, spent for the performance of the activities. The following code, for example, creates an energetic resource to represent the fact that at most 50 human-days of work can be spent each week. The time unit is supposed to be the day and date 0 corresponds to the beginning of the first week.

```
CtDiscreteEnergy* W = new CtDiscreteEnergy(...);
W->makeTimeTable(0, 7 * numberOfWeeks, 7, 50);
```

Energetic reasoning is thus performed for each time interval $[t_1\ t_2)$ of the form $[7n\ 7(n+1))$. In fact, an energetic resource of Ilog Schedule corresponds to a "piece of habograph" used to update the time-bounds of activities. Energetic resources are occasionally used in Ilog Schedule applications to implement redundant constraints (that result in more constraint propagation being performed).

2.3 Theoretical Comparison

The previously described mechanisms have been implemented in several systems and shown to give interesting results as far as computational time is concerned. One issue however to be dealt with is the management of the additional data structures used to implement these techniques. Indeed, these data structures may be more or less costly to maintain both in terms of CPU time and in terms of memory consumption.

The task intervals of [Caseau and Laburthe, 1994] require the maintenance of a data structure in $O(n^2)$ where n is the number of activities requiring a given resource.

Note that, in principle, this memory space consumption could be avoided, but at some expense in terms of CPU time. Similarly, the habograph of [Beck, 1992] uses a memory space in $O((h/g)^2)$ where h is the scheduling horizon and g the grain of the habograph. An energetic resource in ILOG SCHEDULE uses a memory space in $O(h/g)$ only since the energetic analysis is restricted to time intervals of size g. Using implementation cues similar to those used in the sequential time-tables of ILOG SCHEDULE [Le Pape, 1994], the habograph could be reduced to time intervals $[est_A\ let_B)$ such that est_A is the earliest start time of an activity A and let_B the latest end time of another (or the same) activity B. The space complexity would then be $O(n^2)$.

Erschler, Lopez and Thuriot do not say much about their implementation choices in this respect. However, the use of the energetic rules for all pairs $[est_A\ let_B)$ obviously requires either $O(n^2)$ in space, or $O(n^3)$ in CPU time. The less consuming techniques are those of [Carlier and Pinson, 1990] and [Nuijten *et al.*, 1993b].

The following table summarizes the complexity analysis of the various propagation techniques we have discussed. The time complexity that is given corresponds to one iteration (over the n activities of the same resource) of the constraint propagation process. A question mark is used when we are not sure about the actual complexity of the technique. Two things that are difficult to measure a priori do not appear in the table: (1) the typical number of iterations necessary to reach the fixpoint; (2) the "pruning power" of the propagation that is being performed. In the next section, we present experiments performed to assess the effectiveness of some of these propagation techniques: arc-B-consistency, edge-finding, and energetic reasoning.

The table also includes a new algorithm developed by Carlier and Pinson, which runs in $O(n * log(n))$ time [Carlier and Pinson, 1994]. We have not yet found the time to study this algorithm in detail. The results of the next section are based on the $O(n^2)$ algorithms of [Carlier and Pinson, 1990] and [Nuijten *et al.*, 1993b].

Technique	Time	Space
arc-B-consistency	$O(n^2)$	$O(n)$
edge-finding	$O(n^2)$	$O(n)$
	$O(n * log(n))$	$O(n)$
task intervals	$O(n^3)$	$O(n^2)$
energetic reasoning	$O(n^3)$	$O(n)$
habograph	(?)	$O((h/g)^2)$
energetic resources	$O(n * h/g)$	$O(h/g)$

3 Experimental Study

3.1 Implemented Algorithms

In addition to arc-B-consistency, three "extended" constraint propagation algorithms have been implemented on top of version 1.0 of ILOG SCHEDULE.

- The ERA algorithm is based on energetic reasoning.
- The EFJ algorithm performs edge-finding based on Jackson's preemptive schedule.
- The EFN algorithm performs edge-finding based on the method described in [Nuijten *et al.*, 1993b].

ERA

In a preliminary experiment, we combined the first rule of Erschler, Lopez and Thuriot (cf. Section 2.2) with the edge-finding technique of Carlier and Pinson restricted to the overall set of yet unordered operations. Using the resulting algorithm, we noticed that the number of backtracks required to prove the optimality of the solution varied with the order in which constraints were posted. We consequently changed the rule as follows: rather than computing the energy required over the time interval $[est_A\ let_B)$, we compute the energy W that would be required by other activities between the latest start time of A and the earliest end time of B if A were scheduled before B. We deduce that if A is before B, then at least W units of time must elapse between the end of A and the beginning of B. We can then use W to update, still under the hypothesis that A is before B, the latest start time of A and the earliest end time of B. As long as W increases and is not large enough to prove that A cannot be before B, we iterate.

Our implementation of energetic reasoning uses data structures in $O(n)$ (rather than $O(n^2)$). Consequently, the energetic reasoning algorithm runs in $O(n^3)$, even without iterating the energy calculation. Within the same $O(n^3)$ loop, the algorithm directly applies the edge-finding technique of Carlier and Pinson to all the 3-tuples and $(n-1)$-tuples of activities (where n denotes the number of unscheduled activities requiring a given resource), without computing Jackson's preemptive schedule.

EFJ

The second constraint propagation algorithm is a direct implementation in ILOG SCHEDULE of the technique described in [Carlier and Pinson, 1990]. However, our implementation computes Jackson's preemptive schedule in $O(n^2)$ rather than $O(n * log(n))$. This allows us to reuse data structures predefined in ILOG SCHEDULE rather than implementing new ones: it would be interesting to determine the number of activities n at which the price of additional data structures would be balanced by the resulting CPU time savings.

The algorithm also applies the rules of Caseau and Laburthe to a unique task interval for each resource, i.e., the task interval consisting of all unscheduled activities. It runs in $O(n^2)$ and uses data structures in $O(n)$. It is applied iteratively as long as it results in updates of earliest and latest start and end times.

EFN

The third constraint propagation algorithm is similar but uses the algorithm of [Nuijten *et al.*, 1993b] rather than computing Jackson's preemptive schedule. It runs in $O(n^2)$ and uses data structures in $O(n)$.

Normally, the two edge-finding algorithms result in the same deductions. However, in the case of the algorithm of [Nuijten *et al.*, 1993b], we were incidentally able to deduce a bit more than the pure edge-finding technique. Indeed, each time the earliest start time of an activity is updated, we can note that the activity cannot be the first to execute among the yet unscheduled activities of the considered resource. This information can then be used to prune the search tree.

Such deductions are also possible in the case of the algorithm of Carlier and Pinson, but they are less numerous. As we shall see in the next section, the two algorithms provide slightly different results in terms of CPU time and number of backtracks.

3.2 Results

The following table compares the three constraint propagation methods on four problems taken from the constraint programming and operations research literature. (Results on more than eighty problems are available in [Baptiste, 1994].) The first problem is a bridge construction scheduling problem that is often used as a benchmark problem within the constraint programming community [Van Hentenryck, 1989]. The other problems are job-shop scheduling problems with 10 jobs and 10 machines (hence with 100 activities) considered to be particularly hard to solve to optimality. The optimization algorithm consists of a branch-and-bound backtracking search, with constraint propagation being performed at each node of the search tree. Details about the problems and the optimization algorithms are available in [Baptiste, 1994].

In the table, PROB denotes the problem instance and ALGO the constraint propagation algorithm used to solve it. BT and CPU denote the total number of backtracks and CPU time needed to find an optimal solution and prove its optimality. BT-PR and CPU-PR denote the number of backtracks and CPU time needed for the proof of optimality. CPU times are expressed in seconds on a HP715/50 workstation, rounded either to the closest second or to the closest tenth of a second.

For the bridge construction (BDG), the table also provides results obtained using arc-B-consistency (ARC) as a constraint propagation procedure. Such results are not provided for the job-shop problems because 10x10 job-shop problems were generally not solved with ARC after three hours of computational time. ERA results for ABZ5 are not available for the same reason. The table shows that the number of backtracks significantly decreases when more propagation is being performed. On the *simple* bridge problem, the algorithm based on arc-B-consistency performs better than the others in terms of CPU time: the cost of additional constraint propagation is not balanced by the reduction of the search effort. On the *more complex* job-shop scheduling problems, the cost of additional constraint propagation is more than balanced by the reduction of the search effort.

PROB	ALGO	BT	CPU	BT-PR	CPU-PR
BDG	ARC	176	.3	149	.1
	ERA	15	.6	15	.2
	EFJ	14	.6	12	.1
	EFN	14	.5	12	.1
MT10	ERA	99189	9651	19243	1854
	EFJ	52174	2052	8148	333
	EFN	51980	2194	8137	355
ABZ5	EFJ	45187	1369	19574	590
	EFN	44552	1491	19246	649
ABZ6	ERA	1792	155	439	40
	EFJ	1076	49	305	13
	EFN	1068	52	302	14

The three algorithms were also tested on an industrial project scheduling problem submitted by a customer of ILOG, and found to be extremely difficult to solve. The problem consists of scheduling two projects that require common resources. There are forty-five activities and five resources to consider. Each activity requires up to four resources. Within each project, activities are subjected to precedence constraints. Three of the resources are unary resources. The number of activities that require each unary resource is close to thirty. The two other resources are discrete resources with capacity greater than one.

There are two optimization criteria: the goal is to minimize the end times of two activities, one for each project. As the projects rely on common resources, these two optimization criteria are conflicting. As a result, the final user wants to impose upper bounds on the two criteria, and wants the system to tell whether there exists a solution satisfying these upper bounds. The following table provides the CPU times obtained for different values of the upper bounds of the two criteria. If an algorithm is incapable of solving an instance within one hour of CPU time, that is reported by "–". The results show that the algorithms which use edge-finding strongly outperform the other algorithms.

	ALGO	120	125	130	135	140
120	ARC	6	39	131	1188	1
	ERA	4	4	4	39	12
	EFJ	0	0	0	30	1
	EFN	1	1	1	31	1
125	ARC	40	387	1771	–	1
	ERA	4	4	4	118	12
	EFJ	0	0	0	50	1
	EFN	1	1	1	50	2
130	ARC	100	1172	–	–	1
	ERA	4	4	4	938	12
	EFJ	0	0	0	109	1
	EFN	1	1	1	117	2
135	ARC	356	–	–	–	1
	ERA	26	143	934	2718	12
	EFJ	10	62	157	271	1
	EFN	10	58	171	299	2
140	ARC	1	1	1	1	1
	ERA	12	12	12	12	12
	EFJ	1	1	1	1	1
	EFN	1	1	1	2	2

4 Conclusion

Following these experiments, an industrial version of the edge-finding algorithm of [Nuijten *et al.*, 1993b] was implemented in a new version (1.1) of ILOG SCHEDULE. We also improved our search strategy for solving job-shop scheduling problems. The new strategy consists of:

1. generating a first solution (very quickly),
2. iteratively improving the solution by keeping part of the best solution (i.e., keeping some ordering decisions, selected randomly with a probability p that decreases over time) and searching with a limit on the number of backtracks per iteration,
3. systematically exploring the search space when p falls below a given threshold.

The following table provides the average results obtained over three runs, on the ten 10x10 job-shop scheduling problem instances used by Applegate and Cook in their computational study of the job-shop scheduling problem [Applegate and Cook, 1991]. CPU times are expressed in seconds on an IBM RS6000 workstation. Over the ten instances, the total number of backtracks for ILOG SCHEDULE is 215256, while the total number of nodes explored by Applegate and Cook's algorithm is 674128. The use of edge-finding techniques allows the user of ILOG SCHEDULE to enjoy the flexibility inherent to constraint programming, with performance in the same range of efficiency as specific operations research algorithms, such as the one reported in [Applegate and Cook, 1991].

PROB	BT	CPU	BT-PR	CPU-PR
MT10	13684	235.8	4735	67.3
ABZ5	19303	282.1	4519	61.3
ABZ6	6227	100.6	312	4.7
LA19	18102	269.5	6561	91.0
LA20	40597	496.7	20626	227.2
ORB1	22725	407.3	6261	108.0
ORB2	31490	507.1	14123	228.7
ORB3	36729	606.1	22138	342.6
ORB4	13751	213.7	1916	23.7
ORB5	12648	210.9	2658	36.5

Acknowledgments

The authors would like to thank Jean-François Puget, Younès Alaoui, Michel Leconte, Wim Nuijten, and all the members of the ILOG SOLVER development and consulting team for their contributions to the design and continual improvement of ILOG SCHEDULE. Thanks also to the referees for their helpful comments.

References

[Applegate and Cook, 1991] David Applegate and William Cook. A Computational Study of the Job-Shop Scheduling Problem. *ORSA Journal on Computing,* 3(2):149-156, 1991.

[Baptiste, 1994] Philippe Baptiste. Constraint-Based Scheduling: Two Extensions. MSc Thesis, University of Strathclyde, 1994.

[Baptiste and Le Pape, 1995] Philippe Baptiste and Claude Le Pape. Disjunctive Constraints for Manufacturing Scheduling: Principles and Extensions. In *Proceedings of the Third International Conference on Computer Integrated Manufacturing,* Singapore, 1995.

[Beck, 1992] Howard Beck. Constraint Monitoring in TOSCA. In *Working Papers of the AAAI Spring Symposium on Practical Approaches to Planning and Scheduling,* Stanford, California, 1992. American Association for Artificial Intelligence.

[Carlier, 1984] Jacques Carlier. Problèmes d'ordonnancement à contraintes de ressources : algorithmes et complexité. Thèse de Doctorat d'Etat, Université Paris VI, 1984 (in French).

[Carlier and Pinson, 1990] Jacques Carlier and Eric Pinson. A Practical Use of Jackson's Preemptive Schedule for Solving the Job-Shop Problem. *Annals of Operations Research,* 26:269-287, 1990.

[Carlier and Pinson, 1994] Jacques Carlier and Eric Pinson. Adjustment of Heads and Tails for the Job-Shop Problem. *European Journal of Operational Research,* 78:146-161, 1994.

[Caseau and Laburthe, 1994] Yves Caseau and François Laburthe. Improved CLP Scheduling with Task Intervals. In *Proceedings of the Eleventh International Conference on Logic Programming,* Santa Margherita Ligure, Italy, 1994.

[Collinot and Le Pape, 1991] Anne Collinot and Claude Le Pape. Adapting the Behavior of a Job-Shop Scheduling System. *International Journal for Decision Support Systems,* 7(3):341-353, 1991.

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

[Erschler *et al.*, 1991] Jacques Erschler, Pierre Lopez et Catherine Thuriot. Raisonnement temporel sous contraintes de ressource et problèmes d'ordonnancement. *Revue d'Intelligence Artificielle,* 5(3):7-32, 1991 (in French).

[Le Pape, 1988] Claude Le Pape. Des systèmes d'ordonnancement flexibles et opportunistes. PhD Thesis, University Paris XI, 1988 (in French).

[Le Pape, 1994] Claude Le Pape. Implementation of Resource Constraints in ILOG SCHEDULE: A Library for the Development of Constraint-Based Scheduling Systems. *Intelligent Systems Engineering,* 3(2):55-66, 1994.

[Lhomme, 1993] Olivier Lhomme. Consistency Techniques for Numeric CSPs. In *Proceedings of the Thirteenth International Joint Conference on Artificial Intelligence,* pages 232–238, Chambéry, France, 1993. International Joint Committee on Artificial Intelligence.

[Nuijten *et al.*, 1993a] W. P. M. Nuijten, E. H. L. Aarts, D. A. A. van Erp Taalman Kip and K. M. van Hee. Randomized Constraint Satisfaction for Job-Shop Scheduling. In *Proceedings of the AAAI-SIGMAN Workshop on Knowledge-Based Production Planning, Scheduling and Control,* pages 251–262, Chambéry, France, 1993.

[Nuijten *et al.*, 1993b] W. P. M. Nuijten, E. H. L. Aarts, D. A. A. van Erp Taalman Kip and K. M. van Hee. Job-Shop Scheduling by Constraint Satisfaction. Computing Science Note 93/39, Eindhoven University of Technology, 1993.

[Smith and Cheng, 1993] Stephen F. Smith and Cheng-Chung Cheng. Slack-Based Heuristics for Constraint Satisfaction Scheduling. In *Proceedings of the Eleventh National Conference on Artificial Intelligence,* pages 139–144, Washington, District of Columbia, 1993. American Association for Artificial Intelligence.

[Van Hentenryck, 1989] Pascal Van Hentenryck. *Constraint Satisfaction in Logic Programming.* MIT Press, Cambridge, Massachusetts, 1989.