Algorithms for Propagating Resource Constraints in AI Planning and Scheduling: Existing Approaches and New Results

Philippe Laborie
ILOG S.A., 9, rue de Verdun, BP 85
F-94253 Gentilly Cedex, France
plaborie@ilog.fr

Abstract. This paper summarizes the main existing approaches to the propagation of resource constraints in Constraint-Based scheduling and identifies some of their limitations for using them in an integrated planning and scheduling framework. We then describe two new algorithms to propagate resource constraints on discrete resources and reservoirs. Unlike most of the classical work in scheduling, our algorithms focus on the precedence relations between activities rather than on their absolute position in time. They are efficient even when the set of activities is not completely defined and when the time window of activities is large. These features explain why they are particularly suited for integrated planning and scheduling approaches. All our algorithms are illustrated with examples. Some encouraging preliminary results are reported on pure scheduling problems.

1 Introduction

As underlined in [18], some tools are still missing to solve problems that lie between pure AI planning and pure scheduling. Until now, the scheduling community has focused on the optimization of big scheduling problems involving a well-defined set of activities and resource constraints. In contrast, AI planning research - due to the inherent complexity of plan synthesis - has focused on the selection of activities leaving aside the issues of optimization and the handling of time and complex resources. From the point of view of scheduling, mixed planning and scheduling problems have two original characteristics. First, as the set of activities is not completely known beforehand it's better to avoid taking strong scheduling commitments during the search (e.g. instantiating or strongly reducing the time window of an activity). Secondly, most of the partial plans handled by partial order planners (POP) or by hierarchical task network planners (HTN) make an extensive usage of precedence constraints between activities. And, surprisingly, until now the conjunction of precedence and resource constraints has not been deeply investigated, even in the scheduling field itself. Indeed, except for the special case of unary resources (for example in job-shop scheduling), disjunctive formulations of cumulative resource constraints are relatively new techniques and until now, they were mainly used for search control and heuristics [5, 14]. This paper proposes some new constraint propagation algorithms that strongly exploit the conjunction of precedence and resource constraints and allow a natural implementation of least-commitment planning and scheduling approaches. The first section of the paper describes our scheduling model. The second one summarizes the state-of-the-art scheduling propagation techniques and explains why most of them are not satisfactory for dealing with integrated planning and scheduling. In the next section, we describe the basic structure which the new algorithms we propose rely on: precedence graphs. Then, we present two original techniques for propagating resource constraints: the energy precedence algorithm and the balance algorithm. Finally, the last section of the paper describes how these propagation algorithms can be embedded in a least-commitment search procedure and gives some preliminary results on pure scheduling problems.

2 Model and Notations

Partial schedule. A **partial schedule** corresponds to the current scheduling information available at a given node in the search tree. In a mixed planning and scheduling problem, it represents all the temporal and resource information of a partial plan. A partial schedule is composed of activities, and temporal constraints and resource constraints. These concepts are detailed below.

Activities. An **activity** A corresponds to a time interval $[start(A), end(A))$ where $start(A)$ and $end(A)$ are decision variables denoting the start and end time of activity A. $start_{min}(A)$, $start_{max}(A)$, $end_{min}(A)$ and $end_{max}(A)$ will respectively denote the current earliest start time, latest start time, earliest end time and latest end time of activity A. The duration of activity A is a variable $dur(A) = end(A) - start(A)$. Depending on the problem, the duration may be known in advance or may be a decision variable. In a mixed planning and scheduling problem, a planning operator may be composed of one or several activities.

Temporal constraints. A **temporal constraint** is a constraint of the form: $min \leq t_i - t_j \leq max$ where t_i and t_j are either some variable representing the start or end time of an activity or a constant and min and max are two integer constants. Note that simple precedence between activities as well as release dates and due dates are special cases of temporal constraints.

Resources. The most general case of resources we consider in this paper is the reservoir resource. A **reservoir** resource is a multi-capacity resource that can be consumed, produced and/or just required over some time interval by the activities in the schedule. A reservoir has an integer maximal capacity and may have an initial level. As an example of a reservoir, you can think of a fuel tank. A **discrete resource** is a reservoir resource that cannot be produced. Discrete resources are also often called renewable or sharable resources in the scheduling literature. A discrete resource has a known maximal capacity that may change over time. A discrete resource allows for example to represent a pool of workers whose availability varies over time. A **unary resource** is a discrete resource with unit capacity. It imposes that all the activities requiring the same unary resource are totally ordered. This is typically the case of a machine that can only process one job at a time. Unary resources are the simplest and the most studied resources in scheduling as well as in AI planning.

Resource constraints. A **resource constraint** defines how a given activity A will require and affect the availability of a given resource R. It consists of a tuple $< A, R, q, TE >$ where q is an integer decision variable describing the quantity of resource R consumed (if $q < 0$) or produced (if $q > 0$) by activity A and TE is a time extent that describes the time interval where the availability of resource R is affected by the execution of activity A. For example:

- $< A, R_1, -1, FromStartToEnd >$ is a resource constraint that states that activity A will require 1 unit of resource R_1 between its start time and its end time.
- $< A, R_2, q = [2,3], AfterEnd >$ is a resource constraint that states that activity A will produce 2 or 3 units of reservoir R_2 at its end time. This will increase the availability of R_2 after the end time of A.
- $< A, R_3, -4, AfterStart >$ is a resource constraint that states that activity A will consume 4 units of resource R_3 at its start time. This will decrease the availability of R_3 after the start time of A.

Of course, the same activity A may participate into several resource constraints. Note that the change of resource availability at the start or end time of an activity is considered to be instantaneous: we do not handle continuous changes.

Close Status of a Resource. At given node in the search, we say that a resource is **closed** if we know that no additional resource constraint on that resource will be added in the partial schedule when continuing in the search tree. In stratified planning and scheduling approaches where the planning phase is separated from the scheduling one, all the resources can be considered closed during scheduling as all the activities and resource constraints have been generated during the planning phase. Note also that in approaches that interleave planning and scheduling and implement a hierarchical search as in [11], it is also possible to identify as closed resources during the search the ones belonging to already processed abstraction levels.

3 Existing Approaches

From the point of view of Constraint Programming, a partial schedule is a set of decision variables (start, end, duration of activities, required quantities of resource) and a set of constraints between these variables (temporal and resource constraints). A solution schedule is an instantiation of all the decision variables so that all the constraints are satisfied. In Constraint Programming, the main technique used to prune the search space is **constraint propagation**. It consists in removing from the domain of possible values of a decision variable those values that we know for sure will violate some constraint. More generally, constraint propagation allows finding in the current problem some features shared by all the solutions reachable from the current search node; these features may be some domain restriction or some additional constraints that must be satisfied. Currently, in constraint-based scheduling there are two families of algorithms to propagate resource constraints: timetabling approaches and activity interaction techniques.

3.1 Timetabling

The first propagation technique, known as **timetabling**, relies on the computation for every date t of the minimal resource usage at this date by the current activities in the schedule [7]. This aggregated demand profile is maintained during the search and it allows restricting the domains of the start and end times of activities by removing those dates that would necessarily lead to an over-consumption of the resource. For simplicity reason, we describe this technique only on discrete resources and assuming all the time extents are $FromStartToEnd$. Suppose that an activity A requires $q(A) \in [q_{min}(A), q_{max}(A)]$ units of a given resource R and is such that $start_{max}(A) < end_{min}(A)$, then we know for sure that A will at least execute between

$start_{max}(A)$ and $end_{min}(A)$ and thus, it will require for sure $q_{min}(A)$ units of resource R on this time interval. For each resource R, a curve is maintained that aggregates all these demands that is:

$$C_R(t) = \sum_{\{<A,R,q,TE>/start_{max}(A) \leq t < end_{min}(A)\}} q_{min}(A)$$

It's clear that if there exists a date t such that $C_R(t)$ is strictly greater than the maximal capacity of the resource Q, the current schedule cannot lead to a solution and the search must backtrack. Furthermore, if there exists an activity B requiring $q(B)$ units of resource R and a date t_0 such that: $end_{min}(B) \leq t_0 < end_{max}(B)$ and $\forall t \in [t_0, end_{max}(B)), C_R(t) + q_{min}(B) > Q$ then, activity B cannot end after date t_0 as it would over-consume the resource. Indeed, you must remember that, as $end_{min}(B) \leq t_0$, B is never taken into account in the aggregation on the time interval $[t_0, end_{max}(B))$. Thus, t_0 is a new valid upper bound for $end(B)$. A similar reasoning can be applied to find new lower bounds on the start time of activities as well as new upper bounds on the quantity of resource required by activities. Moreover, this approach can easily be extended to all types of time extent and to reservoirs. The main advantage of this technique is its relative simplicity and its low algorithmic complexity. It is the main technique used so far for scheduling discrete resources and reservoirs. Unfortunately, these algorithms propagate nothing until the time windows of activities become so small that some dates t are necessarily covered by some activity. It means that unless some strong commitments are made early in the search on the time windows of activities, these approaches are not able to efficiently propagate. Furthermore, these approaches do not directly exploit the existence of precedence constraints between activities.

3.2 Activity Interactions

The second family of algorithms is based on an analysis of **activity interactions**. Instead of considering what happens at a date t, it considers some subsets Ω of activities competing for the same resource and performs some propagation based on the position of activities in Ω. Some classical activity interaction approaches are summarized below.

Disjunctive Constraint. The simplest example of such an algorithm is the disjunctive constraint on unary resources [8]. This algorithm analyzes each pair of activities (A, B) requiring the same unary resource and, whenever the current time bounds of activities are so that $start_{max}(A) < end_{min}(B)$, it deduces that as activity A necessarily starts before the end of activity B is must be completely executed before B and thus, $end(A) \leq start_{max}(B)$ and $start(B) \geq end_{min}(A)$. Actually, the classical disjunctive constraint can be generalized as follows: whenever the temporal constraints are so that the constraint $start(A) < end(B)$ must hold, it adds the additional constraint that $end(A) \leq start(B)$. Note that this algorithm is the exact counterpart in scheduling of the disjunctive constraint to handle unsafe causal links in POCL planners proposed in [13]. Unfortunately, such a simple constraint only works in the restricted case of unary resources.

Edge-Finding. Edge-finding techniques [3, 16] are available for both unary and discrete resources. On a unary resource, edge-finding techniques detect situations where a given activity A cannot be executed after any activity in a set Ω because there would not be enough time to execute all the activities in $\Omega \cup A$ between the earliest start time of activities in $\Omega \cup A$ and the latest end time of activities in $\Omega \cup A$. When such a situation is detected, it means that A must be executed before all the activities in Ω and it allows to compute a new valid upper bound for the end time of A. More formally, let Ω be a subset of activities on a unary resource, and $A \notin \Omega$ another activity on the same unary resource. Most of the edge-finding technique can be captured by the rule (1) $\Rightarrow$ (2) where:

$$\begin{array}{l}(1)\ end_{max}(\Omega \cup A) - start_{min}(\Omega) < dur(\Omega \cup A) \\ (2)\ \ end(A) \leq \min_{\Omega' \subset \Omega} (end_{max}(\Omega') - dur(\Omega'))\end{array}$$

Similar rules allow to detect and propagate the fact that a given activity must end after all activities in Ω (*Last*), cannot start before all activities in Ω (*Not First*) or cannot end after all activities in Ω (*Not Last*). Furthermore, edge-finding techniques can be adapted to discrete resources by reasoning on the resource energy required by the activities that is, the product *duration* $\times$ *required quantity*. Most of the edge-finding algorithms can be implemented to propagate on all the activities A and all the subsets Ω with a total complexity in $O(n^2)$.

Energetic Reasoning. As for the edge-finding techniques, energetic reasoning [9] analyzes the current time-bounds of activities in order to adjust them by removing some invalid values. A typical example of energetic reasoning consists in finding pairs of activities A, B on a unary resource such that ordering activity A before B would lead to a dead-end because the unary resource would not provide enough "energy" between the earliest start time of A and the latest end time of B to execute A, B and all the other activities that necessarily needs to execute on this time window. More formally, if C is an activity and $[t_1, t_2)$ a time window, the energy necessarily required by C on the time window $[t_1, t_2)$ is:

$$W_C^{[t_1,t_2)} = min(end_{min}(C) - t_1, t_2 - start_{max}(C), dur(C), t_2 - t_1)$$

Thus, as soon as the condition below holds, it means that A cannot be ordered before B and thus, must be ordered after. It allows to update the earliest start time of A and the latest end time of B.

$$end_{max}(B) - start_{min}(A) < dur(A) + dur(B) + \sum_{C \notin \{A,B\}} W_C^{[start_{min}(A), end_{max}(B))}$$

Other adjustments of time bounds using energetic reasoning exist that allow, for example to deduce that an activity cannot start at its earliest start time or cannot end at its latest end time. Furthermore, energetic reasoning can easily be extended to discrete resources.

A good starting point to learn more about edge-finding and energetic reasoning on unary resources is [1] where the authors describe and compare several variants of these techniques. Although these tools (edge-finding, energetic reasoning) are very efficient in pure scheduling problems, they suffer from the same limitations as timetabling techniques. Because they consider the absolute position of activities in time rather than their relative position, they will not propagate until the time windows of activities have become small enough and the propagation may be very limited in case the current schedule contains many precedence constraints. Furthermore, these tools are available for unary and discrete resources only and are difficult to generalize to reservoirs.

The following sections of this paper describes two new techniques to propagate discrete and reservoir resources based on analyzing the relative position of activities rather than their absolute position. These algorithms fully exploit the precedence constraints between activities and propagate even when the time windows of activities are still very large which is typically the case in least-commitment planners and schedulers. Of course these new propagation algorithms can be used in cooperation with the existing techniques we just described above. Both of our algorithms are based on the precedence graph structure described in the section below.

4 Precedence Graph

4.1 Definitions

A **resource event** x on a given resource R is a time-point variable at which the availability of the resource changes because of an activity. A resource event always corresponds to the start or end point of an activity. Let:

- $t(x)$ denote the time-point variable of event x. $t_{min}(x)$ and $t_{max}(x)$ will respectively denote the current minimal and maximal value in the domain of $t(x)$.
- $q(x)$ denote the relative change of resource availability due to event x with the convention that $q > 0$ denotes a resource production and $q < 0$ a resource consumption. $q_{min}(x)$ and $q_{max}(x)$ will respectively denote the current minimal and maximal value in the domain of $q(x)$.

There is of course an evident mapping between the resource constraints on a resource and the resource events. Note that all time extents are associated a unique resource event except for $FromStartToEnd$ that is associated two.
A **precedence graph** on a resource R is a directed graph $G_R = (V, E_{\leq}, E_{<})$ where $E_{<} \subset E_{\leq}$ and:

- V is the set of resource events on R
- $E_{\leq} = (x, y)$ is the set of precedence relations between events of the form $t(x) \leq t(y)$.
- $E_{<} = (x, y)$ is the set of precedence relations between events of the form $t(x) < t(y)$.

The precedence graph on a resource aims at collecting all the precedence information between events on the resource. These precedence information may come from: (1) temporal constraints in the initial statement of the problem, (2) temporal constraints between activities in the same planning operator, (3) search decisions (e.g. causal link, promotion, demotion, ordering decisions on resources) or (4) may have been discovered by propagation algorithms (e.g. unsafe causal links handling, disjunctive constraint, edge-finding, etc.) or simply because $t_{max}(x) \leq t_{min}(y)$. When new events or new precedence relations are inserted, the precedence graph incrementally maintains its transitive closure. This leads to a worst-case complexity of $O(n^2)$ to maintain the precedence graph. The precedence relations in the precedence graph as well as the initial temporal constraints are propagated by an arc-consistency algorithm. Given an event x in a precedence graph and assuming the transitive closure has been computed, we define the following subsets of events:

- $S(x)$ is the set of events simultaneous with x that is the events y such that $(x, y) \in E_{\leq}$ and $(y, x) \in E_{\leq}$
- $B(x)$ is the set of events before x that is the events y such that $(y, x) \in E_{<}$

- $BS(x)$ is the set of events before or simultaneous with x that is the events y such that $(y, x) \in E_{\leq}$, $(y, x) \notin E_{<}$ and $(x, y) \notin E_{\leq}$
- $A(x)$ is the set of events after x that is the events y such that $(x, y) \in E_{<}$
- $AS(x)$ is the set of events after or simultaneous with x that is the events y such that that $(x, y) \in E_{\leq}$, $(x, y) \notin E_{<}$ and $(y, x) \notin E_{\leq}$
- $U(x)$ is the set of events unranked with respect to x that is the events y such that $(y, x) \notin E_{\leq}$ and $(x, y) \notin E_{\leq}$

Note that $(S(x), B(x), BS(x), A(x), AS(x), U(x))$ is a partition of V. An example of precedence graph with an illustration of these subsets is given on Figure 1 and corresponds to a schedule with the 6 resource constraints: $< A_1, R, -2, FromStartToEnd >$, $< A_2, R, [-10, -5], AfterStart >$, $< A_3, R, -1, AfterStart >$, $< A_4, R, 2, AfterEnd >$, $< A_5, R, 2, AfterEnd >$, $< A_6, R, 2, AfterEnd >$ and some precedence relations. The subsets are relative to the event x corresponding to the start of activity A_1.

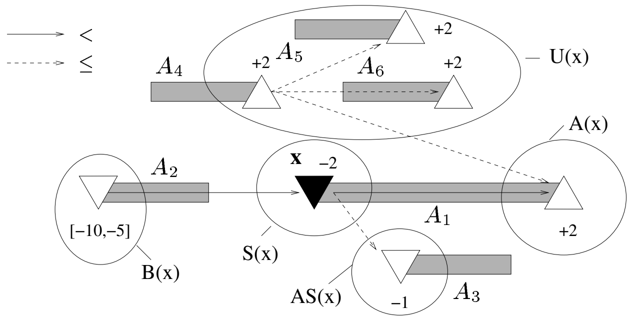

Fig. 1. An Example of Precedence Graph

4.2 Implementation and Complexity

As we will see in next section, our propagation algorithms often need to query the precedence graph about the relative position of two events on a resource so this information needs to be accessible in $O(1)$ on our structure. It explains why we chose to implement the precedence graph as a matrix that stores the relative position of every pair of events. Furthermore, on our structure, the complexity of traversing any subset of events (e.g. $B(x)$ or $U(x)$) is equal to the size of this subset. Note that the precedence graph structure is extensively used in ILOG Scheduler and is not only useful for the algorithms described in this paper. In particular, the precedence graph implementation in ILOG Scheduler allows the user to write his own complex constraints that rely on this graph as for example the one involving alternative resources and transition times described in [10].

5 New Propagation Algorithms

5.1 Energy Precedence Constraint

The **energy precedence constraint** is defined on discrete resources only. As it does not require that the resource be closed, it can be used at any time during the search. The idea is as follows (for simplicity, we assume that all the resource constraints have a time extent $FromStartToEnd$). Suppose that Q denotes the maximal capacity of the discrete resource over time. If x is a resource event and Ω is a subset of resource constraints that are constrained to execute before x, then the resource must provide enough energy to execute all resource constraints in Ω between the earliest start times of activities of Ω and $t(x)$. More formally:

$$t_{min}(x) \geq \min_{<A,R,q,TE> \in \Omega}(start_{min}(A)) + \sum_{<A,R,q,TE> \in \Omega}(q_{min}(A) * dur_{min}(A))/Q$$

A very simple example of the propagation performed by this constraint is given in Figure 2. If we suppose that the maximal capacity of the discrete resource is 4 and all activities must start after time 0, then by considering $\Omega = \{A_1, A_2, A_3, A_4\}$, we see that event x cannot be executed before time $[0]+[(2*10)+(2*8)+(2*8)+(2*2)]/4 = 14$. Of course, a symmetrical rule can be used to find an upper bound on $t(x)$ by considering the subsets Ω of resource constraints that must execute after x. The same idea as the energy precedence constraint is used in [19] to adjust the time-bounds of activities on different unary resources.

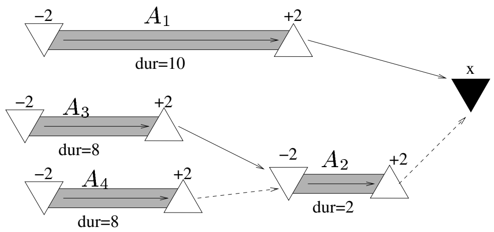

Fig. 2. Example of Energy Precedence Propagation

It's important to note that the energy precedence algorithm propagates even when the time window of activities is very loose (in the example of Figure 2, the latest end times of activities may be very large). This is an important difference with respect to classical energetic and edge-finding techniques that would propagate nothing in this case. The propagation of the energy precedence constraint can be performed for all the events x on a resource and for all the subsets Ω with a total worst-case time complexity of $O(n * (p + log(n))$ where n is the number of the events on the resource and p the maximal number of predecessors of a given event in the graph ($p \leq n$). Note that when the discrete resource has a maximal capacity profile that varies over time, the algorithm can take into account some fake resource constraints with instantiated start and end times to accommodate the maximal capacity profile.

5.2 Balance Constraint

The **balance constraint** is defined on a reservoir resource. When applied to a reservoir, the basic version of this algorithm requires the reservoir to be closed. When applied to a discrete resource, the resource may still be open. The basic idea of the balance constraint is to compute, for each event x in the precedence graph, a lower and an upper bound on the reservoir level just before and just after x. The reader will certainly find some similarities between this constraint and the Modal Truth Criterion on planning predicates first introduced in [6]. Actually this is not surprising as the balance constraint can be considered as a kind of MTC on reservoirs that only detects some necessary conditions[1]. Given an event x, using the graph we can compute an upper bound on the reservoir level at date $t(x) - \epsilon$ just before x assuming (1) All the production events y that **may** be executed strictly before x are executed strictly before x and produce as much as possible that is $q_{max}(y)$; (2) All the consumption events y that **need** to be executed strictly before x are executed strictly before x and consume as little as possible that is $q_{max}(y)$; and (3) All the consumption events that **may** execute simultaneously or after x are executed simultaneously or after x. More formally, if L_{init} is the initial level of the reservoir, P the set of production events and C the set of consumption events, this upper bound can be computed as follows:

$$L^{<}_{max}(x) = L_{init} + \sum_{y \in P \cap (B(x) \cup BS(x) \cup U(x))} q_{max}(y) + \sum_{y \in C \cap B(x)} q_{max}(y) \qquad (1)$$

Applying this formula to event x on Figure 1 if with suppose $L_{init} = 2$ leads to $L^{<}_{max}(x) = 2 + [2 + 2 + 2] + [-5] = 3$. In a very similar way, it is possible to compute $L^{<}_{min}(x)$, a lower bound of the level just before x; $L^{>}_{max}(x)$, an upper bound of the level just after x and $L^{>}_{min}(x)$, a lower bound of the level just after x. For each of these bounds, the balance constraint is able to discover four types of information: **dead ends**, new bounds for **resource usage variables** and **time variables** and new **precedence relations**. For symmetry reasons we only describe the propagation based on $L^{<}_{max}(x)$.

Discovering dead ends. Whenever $L^{<}_{max}(x) < 0$, we know for sure that the level of the reservoir will be negative just before event x so the search has reached a dead end.

Discovering new bounds on resource usage variables. Suppose there exists a consumption event $y \in B(x)$ such that $q_{max}(y) - q_{min}(y) > L^{<}_{max}(x)$. If y would consume a quantity q such that $q_{max}(y) - q > L^{<}_{max}(x)$ then, simply by replacing $q_{max}(y)$ by $q(y)$ in formula (1), we see that the level of the reservoir would be negative just before x. Thus, we can find a better lower bound on $q(y)$ equal to $q_{max}(y) - L^{<}_{max}(x)$. On the example of Figure 1, this propagation would restrict the consumed quantity at the beginning of activity A_2 to $[-8, -5]$ as any value lower than -8 would lead to a dead end.

Discovering new bounds on time variables. Formula (1) can be rewritten as follows:

$$L^{<}_{max}(x) = (L_{init} + \sum_{y \in B(x)} q_{max}(y)) + (\sum_{y \in P \cap (BS(x) \cup U(x))} q_{max}(y))$$

If the first term of this equation is negative, it means that some production events in $BS(x) \cup U(x)$ will have to be executed strictly before x in order to produce at least:

$$\Pi^{<}_{min}(x) = -L_{init} - \sum_{y \in B(x)} q_{max}(y)$$

[1] When the reservoir is not closed, one can imagine extending our propagation algorithm into a real truth criterion on reservoirs that would allow justifying the insertion of new reservoir producers or consumers into the current schedule. This interesting extension clearly worth to study but is out of the scope of this paper.

Let $P(x)$ denote the set production events in $BS(x) \cup U(x)$. We suppose the events $(y_1, \cdots, y_i, \cdots, y_p)$ in $P(x)$ are ordered by increasing minimal time $t_{min}(y)$. Let k be the index in $[1, p]$ such that:

$$\sum_{i=1}^{k-1} q_{max}(y_i) < \Pi^{<}_{min}(x) \leq \sum_{i=1}^{k} q_{max}(y_i)$$

If event x is executed at a date $t(x) \leq t_{min}(y_k)$, not enough producers will be able to execute strictly before x in order to ensure a positive level just before x. Thus, $t_{min}(y_k) + 1$ is a valid lower bound of $t(x)$. On Figure 1 if $L_{init} = 2$, $\Pi^{<}_{min}(x) = 3$, and this propagation will deduce that $t(x)$ must be strictly greater than the minimal between the earliest end time of A_5 and the earliest end time of A_6.

Discovering new precedence relations. There are cases where we can perform an even stronger propagation. Suppose there exists a production event y in $P(x)$ such that:

$$\sum_{z \in P(x) \cap (B(y) \cup BS(y) \cup U(y))} q_{max}(z) < \Pi^{<}_{min}(x)$$

Then, if we had $t(x) \leq t(y)$, we would see that again there is no way to produce $\Pi^{<}_{min}(x)$ before event x as the only events that could eventually produce strictly before event x are the ones in $P(x) \cap (B(y) \cup BS(y) \cup U(y))$. Thus, we can deduce the necessary precedence relation: $t(y) < t(x)$. For example on Figure 1, the balance algorithm would discover that x needs to be executed strictly after the end of A_4. Note that a weaker version of this propagation has been proposed in [4] that runs in $O(n^2)$ and does not analyze the precedence relations between the events of $P(x)$.
Like for timetabling approaches, one can show that the balance algorithm is **sound**, that is, it will detect a dead end on any fully instantiated schedule that violates the reservoir resource constraint. In fact, the balance algorithm does not even need the schedule to be fully instantiated: for example, it will detect a dead end on any non-solution schedule as soon as all the production events are ordered relatively to all the consumption events on a resource. Furthermore, when all events x on a reservoir of capacity Q are so that $L^{<}_{max}(x) \leq Q$, $L^{>}_{max}(x) \leq Q$, $L^{<}_{min}(x) \geq 0$, and $L^{>}_{min}(x) \geq 0$ - in that case, we say that event x is **safe** - then, any order consistent with the current precedence graph satisfies the reservoir constraint. In other words, the reservoir is solved. This very important property allows stopping the search on a reservoir when all the events are safe and even if they are not completely ordered. Note also that, according to the concepts introduced in [14], the balance constraint can be seen as an algorithm that implicitly detects and solves some deterministic MCSs on the reservoir while avoiding the combinatorial explosion of enumerating these MCSs. The balance algorithm can be executed for all the events x with a worst-case complexity in $O(n^2)$ if the propagation that discovers new precedence relations is not turned on, in $O(n^3)$ for a full propagation. In practice, there are many ways to shortcut this worst case and in particular, we noticed that the algorithmic cost of the extra-propagation that discovers new precedence relations was negligible. In our implementation, at each node of the search, the full balance constraint is executed until a fix point is reached.

6 First Results

We implemented a complete and relatively simple search procedure on reservoirs that selects pairs of unsafe events (x, y) and creates a choice point by adding either the relation $t(x) < t(y)$ or $t(y) \geq t(x)$. The heuristics for selecting which pair of events to order relies on the bounds on reservoir levels $L^{<}_{max}(x)$, $L^{>}_{max}(x)$, $L^{<}_{min}(x)$, and $L^{>}_{min}(x)$ computed by the balance constraint. These levels can indeed be considered as some texture measurements [2] projected on the schedule events. Until now, few benchmarks are available on problems involving temporal constraints and reservoirs. The only one we are aware of is [15] where the authors generate 300 project scheduling problems involving 5 reservoirs, min/max delays between activities and minimization of makespan. From these 300 problems, 12 hard instances could not be solved to optimality by their approach. We tested our algorithms on these 12 open problems[2]. The results are summarized on the table below. The size of the problem is the number of activities. The bounds are the best lower and upper bounds of [15]. The times in the table were measured on a HP-UX 9000/785 workstation. We can see that all of the 12 open problems have been closed in less than 1 minute CPU time. Furthermore, our approach produces highly parallel schedules as the balance constraint implements some sufficient conditions for a partial order between events to be a solution. In the optimal solutions there may be up to 10 activities possibly executing in parallel in the partial order.

Problem	**Size**	*Lower bound*	*Upper bound*	**Optimal**	**CPU Time (s)**
#10	50	*92*	*93*	92	0.21
#27	50	*85*	$+\infty$	96	22.18
#82	50	*148*	$+\infty$	no solution	0.15
#6	100	*203*	*223*	211	1.81
#12	100	*192*	*197*	197	1.61
#20	100	*199*	*217*	199	1.54
#30	100	*196*	*218*	204	56.64
#41	100	*330*	*364*	337	1.70
#43	100	*283*	$+\infty$	no solution	53.90
#54	100	*344*	*360*	344	1.26
#58	100	*317*	*326*	317	1.13
#69	100	*335*	$+\infty$	no solution	7.36

We also tested the energy precedence constraint on unary resources. For this purpose, we wrote a very simple least-commitment search procedure[3] based on the precedence graph that orders pairs of activities on a unary resource and aims at finding very good first solutions. We benched this search procedure on 44 famous job-shop problems (namely: *abz5-9*, *ft6*, *ft10*, *orb1-10*, *la1-30*) with the energy precedence constraint as well as the disjunctive and the edge-finder constraint. In average, the makespan of the first solution (without using any restart or randomization) produced by our approach is only 7.35% greater than the optimal makespan whereas the average distance to optimal of the best greedy algorithms so far [17] is 9.33% on the same problems.

7 Conclusion and Future Work

This paper describes two new algorithms for propagating resource constraints on discrete resources and reservoirs. These algorithms strongly exploit the temporal relations in the partial schedule and are able to propagate even if the time windows of activities

[2] All the other problems were easily solved using our approach.

[3] The C++ code of this search procedure is available in the distribution of ILOG Scheduler.

are still very large. Furthermore, on discrete resource, they do not require the resource to be closed. These features explain why they particularly suit integrated approaches to planning and scheduling. From the standpoint of pure scheduling, these algorithms are powerful tools to implement **complete** and **efficient** search procedures based on the relative position of activities. An additional advantage of this approach is that it produces **partially ordered solutions** instead of fully instantiated ones. These solutions are more robust. All the algorithms described in this paper have been implemented and are available in the current version of ILOG Scheduler [12]. As far as AI Planning is concerned, future work will mainly consist in studying the integration of our scheduling framework into a HTN or a POP Planner as well as improving our search procedures.

References

[1] P. Baptiste and C. Lepape. A theoretical and experimental comparison of constraint propagation techniques for disjunctive scheduling. In *Proceedings IJCAI-95*, 1995.

[2] C. Beck, A. Davenport, E. Sitarski, and M. Fox. Texture-based heuristics for scheduling revisited. In *Proceedings AAAI-97*, 1997.

[3] J. Carlier and E. Pinson. A practical use of Jackson's preemptive schedule for solving the job-shop problem. *Annal of Operation Research*, 26:269–287, 1990.

[4] A. Cesta and C. Stella. A time and resource problem for planning architectures. In *Proceedings ECP-97*, 1997.

[5] A. Cesta, A. Oddi, and S. Smith. A constraint-based method for project scheduling with time windows. Technical Report, CMU RI Technical Report, 2000.

[6] D. Chapman. Planning for conjunctive goals. *Artificial Intelligence*, 32:333–377, 1987.

[7] B. Drabble and A. Tate. The use of optimistic and pessimistic resource profiles to inform search in an activity based planner. In *Proceedings AIPS-94*, pages 243–248, 1994.

[8] J. Erschler. *Analyse sous contraintes et aide à la décision pour certains problèmes d'ordonnancement*. PhD thesis, Université Paul Sabatier, 1976.

[9] J. Erschler, P. Lopez, and C. Thuriot. Raisonnement temporel sous contraintes de ressources et problèmes d'ordonnancement. *Revue d'IA*, 5(3):7–32, 1991.

[10] F. Focacci, P. Laborie, and W. Nuijten. Solving scheduling problems with setup times and alternative resources. In *Proceedings AIPS-00*, pages 92–101, 2000.

[11] F. Garcia and P. Laborie. *New Directions in AI Planning*, chapter Hierarchisation of the Search Space in Temporal Planning, pages 217–232. IOS Press, Amsterdam, 1996.

[12] ILOG. ILOG Scheduler 5.1 Reference Manual, 2001. http://www.ilog.com/.

[13] S. Khambhampati and X. Yang. On the role of disjunctive representations and constraint propagation in refinement planning. In *Proceedings KR-96*, 1996.

[14] P. Laborie and M. Ghallab. Planning with sharable resource constraints. In *Proceedings IJCAI-95*, pages 1643–1649, 1995.

[15] K. Neumann and C. Schwindt. Project scheduling with inventory constraints. Technical Report WIOR-572, Universität Karlsruhe, 1999.

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

[17] D. Pacciarelli and A. Mascis. Job-shop scheduling of perishable items. In *Proceedings INFORMS-99*, 1999.

[18] D.E. Smith, J. Frank, and A.K. Jonsson. Bridging the gap between planning and scheduling. *Knowledge Engineering Review*, 15(1), 2000.

[19] F. Sourd and W. Nuijten. Multiple-machine lower bounds for shop scheduling problems. *INFORMS Journal of Computing*, 4(12):341–352, 2000.