# Reasoning with Conditional Time-intervals

**Philippe Laborie** and **Jérôme Rogerie**
ILOG
9 rue de Verdun
94253 Gentilly Cedex, France

**Abstract**

Reasoning with conditional time-intervals representing activities or tasks that may or may not be executed in the final schedule is crucial in many scheduling applications. In Constraint-Based Scheduling, those problems are usually handled by defining new global constraints over classical integer variables. The approach described in this paper takes a different perspective by introducing a new type of variable (namely a *time-interval variable*) that intrinsically embeds the notion of conditionality. This dual perspective facilitates the modelling process while ensuring a strong constraint propagation and an efficient search in the engine. The approach forms the foundations of the new generation of scheduling model and algorithms provided in ILOG CP Optimizer.

## Introduction

Many scheduling problems involve reasoning about activities or processes that may or may not be executed in the final schedule, the choice of executing or not a process being a decision variable of the problem. This is particularly true as scheduling is evolving in the direction of AI Planning whose core problem is precisely the selection of the set of activities to be executed. Indeed, most industrial scheduling applications present at least some of the following features:

- optional activities (operations, tasks) that can be left unperformed (with an impact on the cost) : typical examples are externalized, maintenance or control tasks,
- activities that can execute on a set of alternative resources (machines, manpower) with possibly different characteristics (speed, calendar) and compatibility constraints,
- set of operations that can be processed in different temporal modes (for instance in series or in parallel),
- alternative modes for executing a given activity, each mode specifying a particular combination of resources,
- alternative processes for executing a given production order, a process being specified as a sequence of operations requiring resources,
- hierarchical description of a project as a work-breakdown structure with tasks decomposed into sub-tasks, part of the project being optional (with an impact on the cost if unperformed), *etc*.



Modelling and solving these types of problems is an active topic in Constraint-Based Scheduling. Most of the current approaches are based on defining additional decision variables that represent the existence of an activity in the schedule or the index variable of the alternative resource/mode allocated to an activity and proposing new global constraints and associated propagation algorithms: *XorNode, PEX* in (Beck & Fox 1999), $DTP_{FD}$ in (Moffitt, Peintner, & Pollack 2005), *P/A Graphs* in (Barták & Čepek 2007), *alternative resource constraints* in (ILOG 1997) and (Nuijten *et al.* 2004).

In this paper, we introduce a different approach based on the idea that optional activities should be considered as first class citizen *variables* in the representation (we call them *time-interval variables*) and that the engine is extended to handle this new type of decision variable. Roughly speaking, it is the dual view compared with existing approaches: instead of defining new constraints over classical integer variables to handle optional activities, we introduce them as new variables in the engine. As we will see in the sequel of this paper, doing this offers several advantages:

- modelling is easy because the notion of optionality is intrinsic to the concept of *time-interval variable*: there is no need for additional variables and complex meta-constraints,
- the model is very expressive and separates the temporal aspects from the logical ones,
- constraint propagation is strong because the conditional domain maintained in *time-interval variables* naturally allows conjunctive reasoning between constraints,
- constraints in Constraint-Based Scheduling can be extended to efficiently propagate on *time-interval variables* without impacting their algorithmic complexity.

This paper focuses on the notion of *time-interval variable* and on the basic temporal and logical constraints between them. This framework forms the foundations of the new generation of scheduling model and algorithms embedded in CP Optimizer (ILOG 2008). Additional constraints are provided in CP Optimizer for modelling calendars and resources. Search methods, such as the approach presented

in (Laborie & Godard 2007) have been extended to handle *time-interval variables*. These extensions are out of the scope of the present paper.

## Conditional Time-interval Model

### Time-interval Variables

A **time-interval variable** $a$ is a variable whose domain $dom(a)$ is a subset of $\{\bot\} \cup \{[s,e) | s,e \in \mathbb{Z}, s \leq e\}$. A time-interval variable is said to be **fixed** if its domain is reduced to a singleton, i.e., if $\underline{a}$ denotes a fixed time-interval:

- time-interval is **non-executed**: $\underline{a} = \bot$; or
- time-interval is **executed**: $\underline{a} = [s,e)$

Non-executed time-interval variables have special meaning. Informally speaking, a non-executed time-interval variable is not considered by any constraint or expression on time-interval variables it is involved in. For example, if a non-executed time-interval variable $a$ is used in a precedence constraint between time-interval variables $a$ and $b$ this constraint does not impact time-interval variable $b$. Each constraint specifies how it handles non-executed time-interval variables.

The semantics of constraints defined over time-interval variables is described by the properties that fixed time-intervals must have in order the constraint to be true. If a fixed time-interval $\underline{a}$ is executed and such that $\underline{a} = [s,e)$, we will denote $s(\underline{a}) = s$ its integer start date, $e(\underline{a}) = e$ its integer end date and $d(\underline{a}) = d$ its non-negative integer duration. The execution status $x(\underline{a})$ will be equal to 1. For a fixed time-interval that is non-executed, $x(\underline{a}) = 0$ and the start, end and duration are meaningless.

### Logical Constraints

Execution status of time-interval variables can be further restricted by logical constraints. The **execution constraint** $\mathrm{exec}(a)$ states that a given time-interval variable must be executed. The semantics of the execution constraint on a fixed time-interval $\underline{a}$ is:

$$\mathrm{exec}(\underline{a}) \overset{d}{\leftrightarrow} \big(x(\underline{a}) = 1\big)$$

In the basic model described in this paper, we only consider unary and binary logical constraints between execution statuses, that is, constraints of the form of 2-SAT clauses over execution statuses: $[\neg]\,\mathrm{exec}(a)$ or $[\neg]\,\mathrm{exec}(a) \vee [\neg]\,\mathrm{exec}(b)$. For example if $a$ and $b$ are two conditional time-intervals such that when time-interval $a$ is executed then $b$ must be executed too, it can be modelled by the constraint $\neg\,\mathrm{exec}(a) \vee \mathrm{exec}(b)$.

### Temporal Constraints

The temporal constraint network consists of a Simple Temporal Network (STN) extended to the execution statuses. For instance, a temporal relation $\mathrm{endBeforeStart}(a,b)$ states that *if both time-intervals $a$ and $b$ are executed* then the end of $a$ must occur before the start of $b$.

The semantics of the relation $\mathrm{TC}(\underline{a},\underline{b},z)$ on a pair of fixed time-intervals $\underline{a}$, $\underline{b}$ and for a delay value $z$ depending on the temporal relation type TC is given on Table 1.

| Relation | Semantics |
|---|---|
| startBeforeStart | $x(\underline{a}) \wedge x(\underline{b}) \Rightarrow s(\underline{a}) + z \leq s(\underline{b})$ |
| startBeforeEnd | $x(\underline{a}) \wedge x(\underline{b}) \Rightarrow s(\underline{a}) + z \leq e(\underline{b})$ |
| endBeforeStart | $x(\underline{a}) \wedge x(\underline{b}) \Rightarrow e(\underline{a}) + z \leq s(\underline{b})$ |
| endBeforeEnd | $x(\underline{a}) \wedge x(\underline{b}) \Rightarrow e(\underline{a}) + z \leq e(\underline{b})$ |
| startAtStart | $x(\underline{a}) \wedge x(\underline{b}) \Rightarrow s(\underline{a}) + z = s(\underline{b})$ |
| startAtEnd | $x(\underline{a}) \wedge x(\underline{b}) \Rightarrow s(\underline{a}) + z = e(\underline{b})$ |
| endAtStart | $x(\underline{a}) \wedge x(\underline{b}) \Rightarrow e(\underline{a}) + z = s(\underline{b})$ |
| endAtEnd | $x(\underline{a}) \wedge x(\underline{b}) \Rightarrow e(\underline{a}) + z = e(\underline{b})$ |

Table 1: Temporal constraints semantics

Note that in general, the delay $z$ specified in a temporal constraint can be a variable of the problem rather than a fixed value. For simplicity, we assume in this paper that it is always a fixed value.

### Hybrid N-ary Constraints

This section describes two constraints over a group of time-intervals. Both constraints are hybrid in the sense that they combine logical and temporal aspects. They allow a hierarchical definition of the model by encapsulating a group of time-intervals into one high-level time-interval. Here is an informal definition of these constraints:

- **Span constraint**. The constraint $\mathrm{span}(a_0, \{a_1, .., a_n\})$ states that, if time-interval $a_0$ is executed, it spans over all executed time-intervals from the set $\{a_1, .., a_n\}$. That is, time-interval $a_0$ starts together with the first executed time-interval from $\{a_1, .., a_n\}$ and ends together with the last one. Time-interval $a_0$ is not executed if and only if none of time-intervals $\{a_1, .., a_n\}$ is executed.
- **Alternative constraint**. The constraint $\mathrm{alternative}(a_0, \{a_1, .., a_n\})$ models an exclusive alternative between $\{a_1, .., a_n\}$. If time-interval $a_0$ is executed then exactly one of time-intervals $\{a_1, .., a_n\}$ is executed and $a_0$ starts and ends together with this chosen one. Time-interval $a_0$ is not executed if and only if none of time-intervals $\{a_1, .., a_n\}$ is executed.

More formally, let $\underline{a_0}, \underline{a_1}, ..., \underline{a_n}$ be a set of fixed time-interval variables.

The **span constraint** $\mathrm{span}(\underline{a_0}, \{\underline{a_1}, ..., \underline{a_n}\})$ holds if and only if:

$$\begin{aligned} \neg x(\underline{a_0}) \quad &\Leftrightarrow \quad \forall i \in [1,n], \neg x(\underline{a_i}) \\ x(\underline{a_0}) \quad &\Leftrightarrow \quad \begin{cases} \exists i \in [1,n], x(\underline{a_i}) \\ s(\underline{a_0}) = \min_{i \in [1,n], x(\underline{a_i})} s(\underline{a_i}) \\ e(\underline{a_0}) = \max_{i \in [1,n], x(\underline{a_i})} e(\underline{a_i}) \end{cases} \end{aligned}$$

The **alternative time-intervals constraint** $\mathrm{alternative}(\underline{a_0}, \{\underline{a_1}, ..., \underline{a_n}\})$ holds if and only if:

$$\begin{aligned} \neg x(\underline{a_0}) \quad &\Leftrightarrow \quad \forall i \in [1,n], \neg x(\underline{a_i}) \\ x(\underline{a_0}) \quad &\Leftrightarrow \quad \exists k \in [1,n] \begin{cases} x(\underline{a_k}) \\ s(\underline{a_0}) = s(\underline{a_k}) \\ e(\underline{a_0}) = e(\underline{a_k}) \\ \forall j \neq k, \neg x(\underline{a_j}) \end{cases} \end{aligned}$$

## Complexity

Although both the logical constraint network (2-SAT) and the temporal constraint network (STN) are polynomially solvable frameworks, finding a solution to the basic model described above that combines the two frameworks is NP-Complete (even without alternative and span constraints). The proof is a direct consequence of the fact the model allows the expression of the temporal disjunction between two time-intervals $a$ and $b$. Indeed, such a temporal disjunction can be modelled using 4 additional conditional time-intervals $a_1$, $a_2$, $b_1$ and $b_2$ with the following constraint set:

$$\begin{aligned}
&\mathrm{exec}(a); && \mathrm{exec}(b);\\
&\mathrm{startAtStart}(a, a_1); && \mathrm{startAtStart}(a, a_2);\\
&\mathrm{startAtStart}(b, b_1); && \mathrm{startAtStart}(b, b_2);\\
&\mathrm{endAtEnd}(a, a_1); && \mathrm{endAtEnd}(a, a_2);\\
&\mathrm{endAtEnd}(b, b_1); && \mathrm{endAtEnd}(b, b_2);\\
&\mathrm{endBeforeStart}(a_1, b_1); && \mathrm{endBeforeStart}(b_2, a_2);\\
&\neg\mathrm{exec}(a_1) \vee \neg\mathrm{exec}(a_2); && \mathrm{exec}(a_1) \vee \mathrm{exec}(a_2);\\
&\neg\mathrm{exec}(b_1) \vee \neg\mathrm{exec}(b_2); && \mathrm{exec}(b_1) \vee \mathrm{exec}(b_2);\\
&\neg\mathrm{exec}(a_1) \vee \mathrm{exec}(b_1); && \mathrm{exec}(a_1) \vee \neg\mathrm{exec}(b_1);
\end{aligned}$$

## Graphical Conventions and Examples

The following sections illustrate some examples of models. We are using the following graphical conventions:

- Time-interval variables are represented by a box. When necessary, the time-interval duration is specified inside the box. A dotted box represents an optional time-interval variable.
- Temporal constraints are represented by plain arrows. Depending on the type of temporal constraint, the end-points of the arrow are connected with the appropriate time-points of the time-interval variables.
- Logical constraint are represented by dotted arrows and denote an implication relation (for instance $\mathrm{exec}(a) \Rightarrow \mathrm{exec}(b)$). In case the arrow starts or ends at a cross, the corresponding end-point of the implication is the negation of the execution status.
- Span constraints are represented by a box (the spanning time-interval variable) containing the set of spanned time-interval variables.
- Alternative constraints are represented by a multi-edge labelled by XOR.

### Alternative Modes with Compatibility Constraints

Suppose an activity $a$ can be executed in $n$ possible modes $\{a_i\}_{i\in[1,n]}$ with duration $da_i$ for mode $a_i$ and an activity $b$ can be executed in $m$ possible modes $\{b_j\}_{j\in[1,m]}$ with duration $db_j$ for mode $b_j$. Furthermore, there are some mode incompatibility constraints $(i, j)$ specifying that mode $a_i$ for $a$ is incompatible with mode $b_j$ for $b$. This model is represented on Figure 1, incompatibilities $(i, j)$ are modelled by implications $\mathrm{exec}(a_i) \Rightarrow \neg\mathrm{exec}(b_j)$ (that is: $\neg\mathrm{exec}(a_i) \vee \neg\mathrm{exec}(b_j)$). Of course, in practical applications, time-interval variables $a_i$ and $b_j$ will require some conjunction of resources but, as mentioned in the introduction, this is out of the scope of the present paper. If binary logical constraints are insufficient to model the compatibility rules, the execution status of time-intervals $\mathrm{exec}(a_i)$ can also be used in standard constraint programming constructs such as n-ary logical or arithmetic expressions or table constraints (Bessière & Régin 1997).

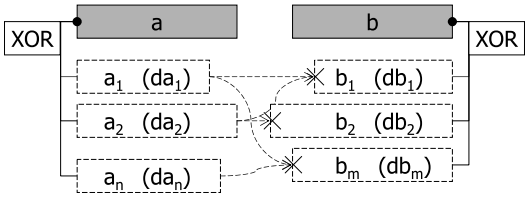


Figure 1: A model for alternative modes

### Series/Parallel Alternative

Figure 2 depicts a model where a job is composed of two operations $a$ and $b$ that can be executed either in series or in parallel (in this case, both operations are constrained to start at the same date). This is modelled by two alternatives $\mathrm{alternative}(a, \{a_1, a_2\})$ and $\mathrm{alternative}(b, \{b_1, b_2\})$ with $(a_1, b_1)$ describing the serial and $(a_2, b_2)$ describing the parallel execution. Logical constraints $\mathrm{exec}(a_i) \Leftrightarrow \mathrm{exec}(b_i)$ are added to ensure a consistent selection of the alternative.

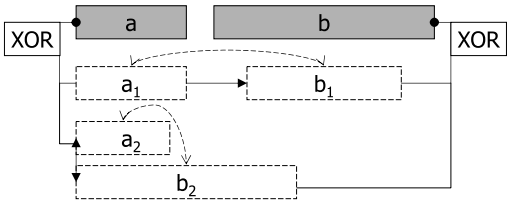


Figure 2: A model for a series/parallel alternative

A similar pattern can be used for any disjunction of a combination of temporal constraints on a pair of time interval variables.

### Alternative Recipes

Figure 3 describes a set of 3 alternative recipes. The global process $a$ is modelled as an alternative of the 3 recipes $r_1$, $r_2$ and $r_3$. Each recipe is a time-interval variable that spans the internal operations of the recipe. Implication constraints between a recipe and some of its internal operations (for instance $\mathrm{exec}(r_3) \Rightarrow \mathrm{exec}(o_{31})$) mean that operation is not optional in the recipe. Note that the opposite implications (for instance $\mathrm{exec}(o_{31}) \Rightarrow \mathrm{exec}(r_3)$) are part of the span constraint. This pattern can be extended to a hierarchy of spanning tasks for modelling complex work-breakdown structures in project scheduling.

### Temporal Disjunction

The graphical representation of the model for temporal disjunction which shows that the basic framework is NP-Complete is depicted on Figure 4.

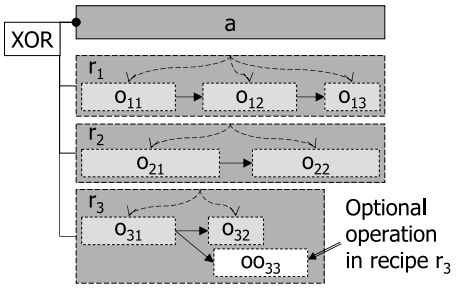


Figure 3: A model for alternative recipes

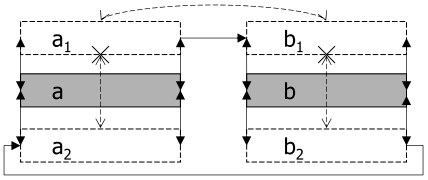


Figure 4: A model for temporal disjunction

## Constraint Propagation

### Time-interval Variables

The domain of a time-interval variable $a$ is represented by a tuple of ranges ($[x_{min}, x_{max}]$, $[s_{min}, s_{max}]$, $[e_{min}, e_{max}]$, $[d_{min}, d_{max}]$). $[x_{min}, x_{max}] \subseteq [0, 1]$ represents the domain of the execution status of $a$. $x_{min} = x_{max} = 1$ means that $a$ will be executed whereas $x_{min} = x_{max} = 0$ means that it will not be. $[s_{min}, s_{max}] \subseteq \mathbb{Z}$ represents the conditional domain of the start time of $a$, that is, the minimal and maximal start time *would $a$ be executed*. Similarly, $[d_{min}, d_{max}] \subseteq \mathbb{Z}$ and $[e_{min}, e_{max}] \subseteq \mathbb{Z}$ respectively denote the conditional domain of the duration and end time of $a$.

The time-interval variable maintains the internal consistency between the temporal bounds $[s_{min}, s_{max}]$, $[e_{min}, e_{max}]$ and $[d_{min}, d_{max}]$ that are due to the relation $d = e - s$. When the temporal bounds become inconsistent (for instance because $s_{min} > s_{max}$ or because $e_{min} - s_{max} > d_{max}$), the time-interval execution status is automatically set to false ($x_{max} = 0$). Of course, if execution status of the time-interval is already true ($x_{min} = 1$), this will trigger a failure. Just like other classical variables in CSPs:

- The domain of time-interval variables can be accessed thanks to accessors: $is[True|False]$, $get[Start|End|Duration][Min|Max]$
- It can be modified thanks to *modifiers*: $set[True|False]$, $set[Start|End|Duration][Min|Max]$ ,
- Events can be attached to the change of the domain so as to trigger constraint propagation. In this context, accessors are also available to access the previous value of the domain which is useful for implementing efficient incremental constraints: $getOld[Start|End|Duration][Min|Max]$.

The following sections describe how logical, temporal and hybrid n-ary constraints are handled by the engine.

### Logical Network

All 2-SAT logical constraints between time-interval execution statuses of the form $[\neg]\,\mathrm{exec}(a) \vee [\neg]\,\mathrm{exec}(b)$ are aggregated in a *logical network* similar to the implication graph described in (Brafman 2001). The objectives of the logical network are:

- The detection of inconsistencies in logical constraints.
- An $O(1)$ access to the logical relation that can be inferred between any two time-intervals $(a, b)$.
- A traversal of the set of time-intervals whose execution is implied by (resp. implies) the execution of a given time-interval variable $a$.
- The triggering of some events as soon as a new implication relation is inferred between two time-intervals $(a, b)$ in order to wake up constraint propagation.

The above services are incrementally ensured when new logical constraints are added to the network or when the execution status of a time-interval is fixed. They are used by the propagation, as for instance in the temporal network presented in next section, and by the search algorithms.

Nodes $\{l_i\}_{i\in[1,n]}$ in the graph correspond to execution statuses $\mathrm{exec}(a)$ or their negation $\neg\,\mathrm{exec}(a)$ and an arc $l_i \rightarrow l_j$ corresponds to an implication relation between the corresponding boolean statuses. For instance a constraint $\mathrm{exec}(a) \vee \mathrm{exec}(b)$ would be associated with an arc $\neg\,\mathrm{exec}(a) \rightarrow \mathrm{exec}(b)$. As links $l_i \rightarrow l_j$ and $\neg l_j \rightarrow \neg l_i$ are equivalent, for each time-interval variable $a$, only one node has to be considered in the network, either the one corresponding to $\mathrm{exec}(a)$ or the one corresponding to $\neg\,\mathrm{exec}(a)$. Fixed execution statuses are skipped from the network as in this case binary constraints are reduced to unary constraints. Strongly connected components of the implication graph are collapsed into a single node representing the logical equivalence class and the transitive closure of the resulting directed acyclic graph is maintained as new arcs are added. The logical network becomes inconsistent when it allows to infer both relations $l_i \rightarrow \neg l_i$ and $\neg l_i \rightarrow l_i$.

The time and memory complexity of the logical network for performing the transitive closure is quadratic with the length of the implication graph. In usual scheduling problems, this length tends to be small compared with the number of time-interval variables. Typically, this length is related with the depth of the work-breakdown structure.

### Temporal Network

Temporal constraints [start|end][Before|At][Start|End] are aggregated in a *temporal network* whose nodes $\{p_i\}_{i\in[1,n]}$ represent the set of time-interval start and end points. If $p_i$ is a time-point in the network, we denote $x(p_i)$ the (variable) boolean execution status of the time-interval variable of $p_i$ and $t(p_i)$ the (variable) date of the time-point. An arc $(p_i, p_j, z_{ij})$ in the network denotes a minimal delay $z_{ij}$ between the two time-points $p_i$ and $p_j$ *would both time-points be executed*, that is: $x(p_i) \wedge x(p_j) \Rightarrow (t(p_i) + z_{ij} \leq t(p_j))$.

It is easy to see that all temporal constraints can be represented by one or two arcs in the temporal network. Furthermore, the duration of the time-interval is also represented by two arcs, one between the start and the end time-point labelled with the minimal duration and the other between the end and the start time-point labelled by the opposite of the maximal duration. Let $t_{min}(p_i)$ and $t_{max}(p_i)$ denote the current conditional bounds on the date of time-point $p_i$. Depending on whether $p_i$ denotes the start or the end of a time-interval variable, these bounds are the $s_{min}$, $s_{max}$ or the $e_{min}$, $e_{max}$ values stored in the current domain of the time-interval variable of $p_i$.

The main idea of the propagation of the temporal network is that for a given arc $(p_i, p_j, z_{ij})$, whenever the logical network can infer the relation $x(p_i) \Rightarrow x(p_j)$ the propagation on the conditional bounds of $p_i$ (time-bounds *would* $p_i$ *be executed*) can assume that $p_j$ will also be executed and thus the arc can propagate the conditional bounds from time-point $p_j$ on $p_i$: $t_{max}(p_i) \leftarrow \min(t_{max}(p_i), t_{max}(p_j) - z_{ij})$. Similarly, if the relation $x(p_j) \Rightarrow x(p_i)$ can be inferred by the logical network then the other half of the propagation that propagates on time-point $p_j$ can be performed: $t_{min}(p_j) \leftarrow \max(t_{min}(p_j), t_{min}(p_i) + z_{ij})$. This observation is crucial: it allows to propagate on the conditional bounds of time-points even when their execution status is not fixed. Of course, when the execution status of a time-point $p_i$ is fixed to 1, all other time-points $p_j$ are such that $x(p_j) \Rightarrow x(p_i)$ and thus, the bounds of $p_i$ can be propagated on all the other time-points. When all time-points are surely executed, this propagation boils down to the classical bound-consistency on STNs. When the two time-points of an arc have equivalent execution status, the arc can be propagated in both directions, this is in particular the case for arcs corresponding to time-interval durations. When the time-bounds of the extremities of an arc $(p_i, p_j, z_{ij})$ become inconsistent, the logical constraint $x(p_i) \Rightarrow \neg x(p_j)$ can be added to the logical network.

Figure 5 depicts the problem described in (Barták & Čepek 2007) modelled in our framework. If the deadline for finishing the schedule is 70, the propagation will infer that the alternative $BuyTube$ cannot be executed as there is not enough space between the minimal start time of $GetTube$ (1) and its maximal end time (50) to accommodate its duration of 50. Note that if the duration of operation $BuyTube$ was lower than 49 but if the sum of the durations of operations $SawTube$ and $ClearTube$ was greater than 49, then the propagation would infer that the alternative $SawTube \rightarrow ClearTube$ is impossible because these two operations have equivalent execution status and thus, the precedence arc between them can propagate in both directions.

Figure 6 illustrates another example of propagation. The model consists of a chain of $n$ identical optional operations $\{o_i\}_{i\in[1,n]}$ of duration 10 that, if executed, need to be executed before date 25 and are such that $\text{exec}(o_{i+1}) \Rightarrow \text{exec}(o_i)$. Although initially, all operations are optional, the propagation will infer that only the first two operations can be executed and will compute the exact conditional minimal start and end times for the two possibly executed operations.

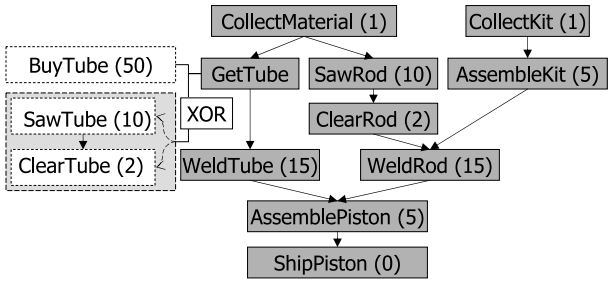


Figure 5: Example of propagation

We are not aware of any other framework that is capable of inferring, by constraint propagation only, such type of information on purely optional activities.

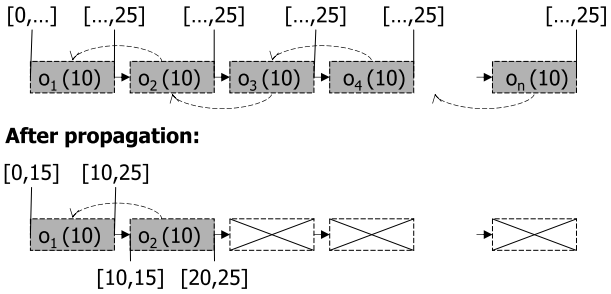


Figure 6: Example of propagation

Most of the classical algorithms for propagating on STNs can be extended to handle conditional time-points. In CP Optimizer, the initial propagation of the temporal network is performed by an improved version of the Bellman-Ford algorithm presented in (Cherkassky, Goldberg, & Radzic 1996) and the incremental propagation when a time-bound has changed or when a new arc or a new implication relation is detected is performed by an extension of the algorithm for positive cycles detection proposed in (Cesta & Oddi 1996). The main difference with the original algorithms is that propagation of the temporal bounds is performed only following those arcs that are allowed to propagate on their target given the implication relations. This propagation allows for instance to infer that a set of optional time-intervals with equivalent status forming a positive cycle in the temporal network cannot be executed.

## Hybrid N-ary Constraints

The propagation of both the *span* and *alternative* constraints follow the same pattern. First, the part of the propagation that can be delegated to the logical and temporal networks is transferred to them:

- In the case of a span constraint $\text{span}(a_0, \{a_1, ..., a_n\})$, the set of implications $\text{exec}(a_i) \Rightarrow \text{exec}(a_0)$ are treated by the logical network and the set of arcs $\text{startBeforeStart}(a_0, a_i)$ and $\text{endBeforeEnd}(a_i, a_0)$ by the temporal network.

- In the case of an alternative constraint $\text{alternative}(a_0, \{a_1, ..., a_n\})$, the set of implications $\text{exec}(a_i) \Rightarrow \text{exec}(a_0)$ are treated by the logical network and the set of arcs $\text{startAtStart}(a_0, a_i)$ and $\text{endAtEnd}(a_i, a_0)$ by the temporal network.

The rest of the propagation is performed by the specific constraints themselves:

- The span constraint propagates the fact that when $a_0$ is executed, there must exist at least one executed time-interval variable $a_i$ that starts at the same date as the start of $a_0$ (and the symmetrical relation for the end).
- The alternative constraint propagates the fact that no two time-interval variables $a_i$ and $a_j$ can simultaneously execute and maintains the conditional temporal bounds of time-interval variable $a_0$ as the constructive disjunction of the conditional temporal bounds of possibly executed time-interval variables $a_i$. Propagation events on time-interval variables allow writing efficient incremental propagation for the alternative constraint.

## Conclusion and Future Work

This paper presents a framework for reasoning about conditional time-intervals that serves as the foundation of the new generation of scheduling model and algorithms in CP Optimizer. It introduces the notion of conditional *time-interval variables* as first-class citizen decision variables in the constraint programming language and engine. This new type of variable allows easily writing expressive models for scheduling applications involving conditional activities while ensuring a strong constraint propagation and an efficient search in the engine. This framework can also be used in AI temporal planning to strengthen the propagation in approaches such as the one proposed in (Vidal & Geffner 2006).

The basic model introduced in this paper has been complemented in CP Optimizer with a set of useful global constraints for scheduling problems including time-interval sequencing (unary resources), cumulative (discrete resources, reservoirs) and state reasoning (state resources) as well as with numerical expressions to use time-interval bounds (start, end, duration, execution status) in classical CSP constraints. The classical global constraint propagation algorithms for scheduling (timetabling, disjunctive constraint, edge-finding, *etc.*) have been adapted to handle the execution status of time-interval variables. See for instance (Vilím, Barták, & Čepek 2005) for edge-finding.

As illustrated on Figure 7, coupling global constraints with logical and temporal network information allows performing stronger conjunctive deductions on the bounds of time-interval variables and inferring new logical and temporal relations.

The search method described in (Laborie & Godard 2007) has been extended to handle time-interval variables. It consists of a tree search for producing an initial solution followed by a self-adapting iterative improvement method. Beside constraint propagation, the logical and temporal networks described in this paper, which are linear structures that convey important information about the structure of the problem, are also used to compute relaxations to guide tree search and to define the moves of the iterative method. Future work will consist in enhancing constraint propagation and search following this general pattern.

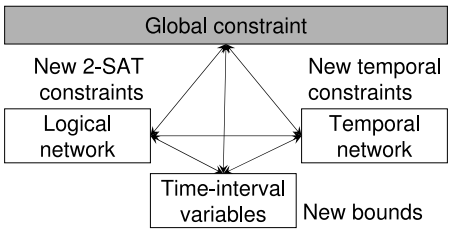


Figure 7: General approach for propagation

## References


Barták, R., and Čepek, O. 2007. Temporal networks with alternatives: Complexity and model. In *Proc. FLAIRS-2007*.

Beck, J. C., and Fox, M. S. 1999. Scheduling alternative activities. In *Proc. AAAI-99*.

Bessière, C., and Régin, J.-C. 1997. Arc consistency for general constraint networks: preliminary results. In *Proc. IJCAI'97*, 398–404.

Brafman, R. I. 2001. A simplifier for propositional formulas with many binary clauses. In *Proc. IJCAI-01*, 515–522.

Cesta, A., and Oddi, A. 1996. Gaining efficiency and flexibility in the simple temporal problem. In *Proc. TIME-96*.

Cherkassky, B.; Goldberg, A.; and Radzic, T. 1996. Shortest paths algorithms: Theory and experimental evaluation. *Mathematical Programming* 73:129–174.

ILOG. 1997. *ILOG Scheduler 4.0 Reference Manual*.

ILOG. 2008. ILOG CP Optimizer. Webpage: http://www.ilog.com/products/cpoptimizer/.

Laborie, P., and Godard, D. 2007. Self-adapting large neighborhood search: Application to single-mode scheduling problems. In *Proc. MISTA-07*.

Moffitt, M. D.; Peintner, B.; and Pollack, M. E. 2005. Augmenting disjunctive temporal problems with finite-domain constraints. In *Proc. AAAI-2005*.

Nuijten, W.; Bousonville, T.; Focacci, F.; Godard, D.; and Le Pape, C. 2004. Towards an industrial manufacturing scheduling problem and test bed. In *Proc. PMS-2004*, 162–165.

Vidal, V., and Geffner, H. 2006. Branching and Pruning: An Optimal Temporal POCL Planner based on Constraint Programming. *Artificial Intelligence* 170:298–335.

Vilím, P.; Barták, R.; and Čepek, O. 2005. Extension of o(n log n) filtering algorithms for the unary resource constraint to optional activities. *Constraints* 10:403–425.