HAL
open science

A Global Constraint for a Tractable Class of Temporal Optimization Problems

Alban Derrien, Jean-Guillaume Fages, Thierry Petit, Charles Prud'Homme

To cite this version:

Alban Derrien, Jean-Guillaume Fages, Thierry Petit, Charles Prud'Homme. A Global Constraint for a Tractable Class of Temporal Optimization Problems. Principles and Practice of Constraint Programming, CP 2015, Aug 2015, Cork, Ireland. 10.1007/978-3-319-23219-5_8 . hal-01241079

HAL Id: hal-01241079

https://hal.science/hal-01241079

Submitted on 9 Dec 2015

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

A Global Constraint for a Tractable Class of Temporal Optimization Problems

Alban Derrien[1], Jean-Guillaume Fages[2], Thierry Petit[1,3], and Charles Prud'homme[1]

[1] TASC (CNRS/INRIA), Mines Nantes, FR – 44307 Nantes, France
FirstName.LastName@mines-nantes.fr
[2] COSLING S.A.S., FR – 44307 Nantes, France
jg.fages@cosling.com
[3] Foisie School of Business, WPI, Worcester, MA 01609, USA
TPetit@wpi.edu

Abstract. This paper is originally motivated by an application where the objective is to generate a video summary, built using intervals extracted from a video source. In this application, the constraints used to select the relevant pieces of intervals are based on Allen's algebra. The best state-of-the-art results are obtained with a small set of ad hoc solution techniques, each specific to one combination of the 13 Allen's relations. Such techniques require some expertise in Constraint Programming. This is a critical issue for video specialists. In this paper, we design a generic constraint, dedicated to a class of temporal problems that covers this case study, among others. *ExistAllen* takes as arguments a vector of tasks, a set of disjoint intervals and any of the 2^{13} combinations of Allen's relations. *ExistAllen* holds if and only if the tasks are ordered according to their indexes and for any task at least one relation is satisfied, between the task and at least one interval. We design a propagator that achieves bound-consistency in $O(n+m)$, where n is the number of tasks and m the number of intervals. This propagator is suited to any combination of Allen's relations, without any specific tuning. Therefore, using our framework does not require a strong expertise in Constraint Programming. The experiments, performed on real data, confirm the relevance of our approach.

1 Introduction

The study of temporal relations between elements of a process is a very active topic of research, with a wide range of applications: biomedical informatics [5,20], law [17], media [2,4,8] etc. Temporal reasoning enables to analyze the content of a document in order to infer high-level information. In particular, a summary of a tennis match may be generated from the match recording with Artificial Intelligence techniques [2,4]. The summarization requires to extract some noticeable time intervals and annotate them with qualitative attributes using signal recognition techniques. Then, the summary generation may be formulated as a Constraint Satisfaction Problem (CSP) where variables are the video segments to

be displayed whereas constraints stem from different considerations: displaying relevant information, balancing the selected content, having nice transitions etc. Then, the CSP may be solved quite efficiently with a Constraint-Programming (CP) solver. At first sight, this seems an easy task.

Unfortunately, things get harder when it comes to practice. Designing a CP model requires some expert knowledge, in order to achieve good performances on hard problems. This is particularly true when one has to design a global constraint that would be missing in the solver. For instance, the summarization model of [2] relied on many disjunctions that are poorly propagated by constraint engines and thus lead to unsatisfiable performances. Therefore, the authors collaborated with CP experts to design ad hoc global constraints, which lead to significant speedups [4]. Alternatively, one may have used temporal logic models, to benefit from solution techniques and solvers dedicated the temporal CSP [18,7,9,6]. As the video summarization gets more sophisticated, all these approaches suffer from the need of specific and often intricate propagators/models. This is a critical issue for video specialists, who are rarely CP experts. Furthermore, one may need to include in her model other features available in state-of-the-art constraint solvers, such as standard global constraints and predefined search strategies. What is missing is a both expressive and efficient global constraint for modeling a relevant class of problems on time intervals.

We introduce the *ExistAllen* constraint, defined on a vector of tasks $\mathcal{T}$ and a set of disjoint intervals $\mathcal{I}$, respectively of size n and m. Given a subset $\mathcal{R}$ of Allen's relations [1], *ExistAllen* is satisfied if and only if the two following properties are satisfied:

1. For any task in $\mathcal{T}$ at least one relation in $\mathcal{R}$ is satisfied between this task and at least one interval in $\mathcal{I}$.
2. Tasks in $\mathcal{T}$ are ordered according to their indexes in the vector given as argument, *i.e.,* for any integer i, $1 \leq i < n$, the task T_i should end before or at the starting time of T_{i+1}.

In the context of video-summarization, tasks in $\mathcal{T}$ are the video segments that compose the summary. Fixed video sequences in $\mathcal{I}$ are extracted from the source according to some precise features. In this way, it is possible to constrain the content of the summary with qualitative information.

Considering the invariability of task processing times, we introduce a bound-consistency propagator for this constraint, suited to any of the 2^{13} subsets of Allen's relations. The time complexity of the most natural algorithm for this propagator is $O(n \times m)$. We propose an improved algorithm, running in $O(n+m)$ time. While *ExistAllen* may be used in different contexts, e.g., online scheduling, this paper is motivated by video-summarization. Our experiments on the Boukadida et al.'s application [4] demonstrate that using our generic constraint and its linear propagator is significantly better than the models built with standard constraints of the solver, and competitive with the ad hoc global constraint approach.

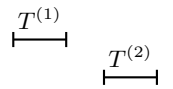
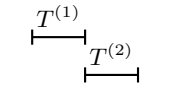
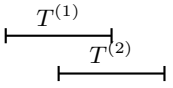
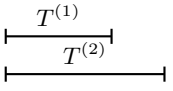
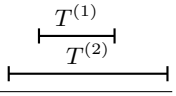
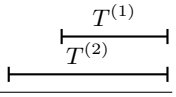
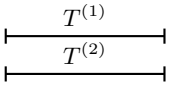
Symbol	Relation	
$T^{(1)}$ p $T^{(2)}$ ($T^{(2)}$ pi $T^{(1)}$)	$T^{(1)}$ *precedes* $T^{(2)}$	
$T^{(1)}$ m $T^{(2)}$ ($T^{(2)}$ mi $T^{(1)}$)	$T^{(1)}$ *meets* $T^{(2)}$	
$T^{(1)}$ o $T^{(2)}$ ($T^{(2)}$ oi $T^{(1)}$)	$T^{(1)}$ *overlaps* $T^{(2)}$	
$T^{(1)}$ s $T^{(2)}$ ($T^{(2)}$ si $T^{(1)}$)	$T^{(1)}$ *starts* $T^{(2)}$	
$T^{(1)}$ d $T^{(2)}$ ($T^{(2)}$ di $T^{(1)}$)	$T^{(1)}$ *during* $T^{(2)}$	
$T^{(1)}$ f $T^{(2)}$ ($T^{(2)}$ fi $T^{(1)}$)	$T^{(1)}$ *finishes* $T^{(2)}$	
$T^{(1)}$ eq $T^{(2)}$	$T^{(1)}$ *equal to* $T^{(2)}$	

Table 1. Allen's temporal algebra relations.

2 Background

In this section we give some background and fix the notations used in this paper.

2.1 Temporal Constraint Networks

Temporal reasoning has been an important research topic for the last thirty years. One may distinguish qualitative temporal reasoning, based on relations between intervals and/or time points, from quantitative reasoning where duration of a given event is represented in a numerical fashion. Allen's algebra [1] represents qualitative temporal knowledge by interval constraint networks. An interval constraint network is a directed graph where nodes represent intervals and edges are labelled with disjunctions of Allen's relations. Table 1 details those relations. Many state-of-the-art papers deal with generic solving techniques and tractability of temporal networks [6,7,9,18], including temporal problems with quantified formulas [10,12]. Most of these methods make no strong restriction on the constraint network to be solved.

A few techniques, more specialized, focus on optimization problems. The two most related to this paper are the following. Kumar et al. [11] consider temporal

problems with "taboo" regions, minimizing the number of intersections between tasks and a set of fixed intervals. These problems occur in scheduling applications, among others. In the context of video summarization, a recent paper [4] proposes the idea of using global constraints involving a set of ordered intervals. Each constraint is restricted to one specific relation in the 2^{13} combinations of Allen's relations. The propagators are not described.

2.2 Constraint Programming (CP)

CP is a problem solving framework where relations between variables are stated in the form of constraints, which together form a constraint network. Each variable x has a domain $D(x)$, whose minimum value is $\underline{x}$ and maximum value is $\overline{x}$. A *task* T in a set $\mathcal{T}$ is an object represented by three integer variables: s_T, its starting time, e_T, its ending time, and p_T, its processing time. The task should satisfy the constraint $s_T + p_T = e_T$. An interval I in a set $\mathcal{I}$ is a fixed task, defined by integer values instead of variables. A *propagator* is an algorithm associated with a constraint, stated on a set of variables. This propagator removes from domains values that cannot be part of a solution to that constraint. The notion of consistency characterizes propagator effectiveness. In this paper, we consider *bound*($\mathbb{Z}$)-*consistency* [3]. When domains are exclusively represented by their bounds (i.e., have no holes), *bound*($\mathbb{Z}$)-*consistency* ensures that for each variable x, $\underline{x}$ and $\overline{x}$ can be part of a solution of the constraint.

3 The *ExistAllen* constraint

This section introduces the *ExistAllen* constraint and its propagator. Let $\mathcal{T} = \{T_1, T_2, \ldots, T_n\}$ be a set of tasks, such that any task T_{i+1} must be scheduled at or after the end of task T_i. Similarly, we define a set of ordered Intervals $\mathcal{I} = \{I_1, I_2, \ldots, I_m\}$. From a subset $\mathcal{R}$ of Allen's relations, *ExistAllen* ensures that any task in $\mathcal{T}$ is related to at least one interval in $\mathcal{I}$.

Definition 1 (*ExistAllen*). *ExistAllen*$(\mathcal{T}, \mathcal{R}, \mathcal{I}) \Leftrightarrow$

$$\forall T \in \mathcal{T}, \quad \bigvee_{R \in \mathcal{R}} \bigvee_{I \in \mathcal{I}} T \; R \; I$$

$$\wedge \quad (\forall i, 1 \leq i < n, s_{T_{i+1}} \geq e_{T_i})$$

Example 1. When summarizing the video of a tennis match, it may be required that each segment (task) selected to be part the summary contains an applause. Applause intervals can be preprocessed [4]. Such requirement can then be formulated with a *ExistAllen* constraint, where $\mathcal{T}$ are the selected segments, $\mathcal{I}$ are the applause segments and $\mathcal{R}$ is set to $\{fi, di, eq, si\}$.

A symmetry can be used, where the problem is seen in a mirror: starting variables become ending variables and the lower bounds filtering of the mirror

relation is applied to the upper bounds (e.g. *starts* is propagated onto upper bounds using *finishes*, see Table 1). Therefore, we put the focus on the algorithms for the lower bounds of starting/ending task variables in $\mathcal{T}$. We consider here that processing times are exclusively updated by the constraints $s_T + p_T = e_T$.

3.1 Basic filtering: one relation, one task, one interval

The basic filtering rule specific to each Allen's relation can be derived from time-point logic relations [19], in order to state lower and upper bounds of starting and ending times of tasks. For instance, consider a task $T \in \mathcal{T}$, an interval $I \in \mathcal{I}$, and the relation *starts*. The relation is satisfied if and only if two conditions are met:

$$T\ s\ I \ \Leftrightarrow \ \ s_T = s_I \ \wedge \ e_T < e_I$$

The only filtering on the lower bounds of task T induced by the two conditions of relation *starts* is $\underline{s_T} \geq s_I$. On the same basis, we define in Table 2 the conditions and filtering rules for the 13 Allen's algebra relations between a task T and an interval I.

Relation	Conditions	Filtering
$T\ p\ I$	$e_T < s_I$	
$T\ pi\ I$	$e_I < s_T$	$\underline{s_T} > e_I$
$T\ m\ I$	$e_T = s_I$	$\underline{e_T} \geq s_I$
$T\ mi\ I$	$s_T = e_I$	$\underline{s_T} \geq e_I$
$T\ o\ I$	$s_T < s_I$ $e_T < e_I$ $e_T > s_I$	 $\underline{e_T} > s_I$
$T\ oi\ I$	$s_T < e_I$ $s_T > s_I$ $e_T > e_I$	 $\underline{s_T} > s_I$ $\underline{e_T} > e_I$
$T\ s\ I$	$s_T = s_I$ $e_T < e_I$	$\underline{s_T} \geq s_I$

Relation	Conditions	Filtering
$T\ si\ I$	$s_T = s_I$ $e_T > e_I$	$\underline{s_T} \geq s_I$ $\overline{e_T} > e_I$
$T\ d\ I$	$s_T > s_I$ $e_T < e_I$	$\underline{s_T} > s_I$
$T\ di\ I$	$s_T < s_I$ $e_T > e_I$	 $\underline{e_T} > e_I$
$T\ f\ I$	$s_T > s_I$ $e_T = e_I$	$\underline{s_T} > s_I$ $\underline{e_T} \geq e_I$
$T\ fi\ I$	$s_T < s_I$ $e_T = e_I$	 $\underline{e_T} \geq e_I$
$T\ eq\ I$	$s_T = s_I$ $e_T = e_I$	$\underline{s_T} \geq s_I$ $\underline{e_T} \geq e_I$

Table 2. Allen's algebra lower bound filtering on each variable of a task T.

3.2 A first propagator

We propose a propagator based on two procedures. Again, we only present the algorithms adjusting lower bounds of starting times and ending times.

```
Require: Global Variable : s*, e*
 1: procedure ExistAllenQuadratic(𝒯, ℛ, ℐ)
 2:     for i = 1 to n do                                   ▷ Loop over tasks
 3:         s* ← s̄_{T_i} + 1                ▷ Intialize to a value out of the domain
 4:         e* ← ē_{T_i} + 1
 5:         for j = 1 to m do                               ▷ Loop over Intervals
 6:             AllenAllRelation(T_i, ℛ, I_j)
 7:         end for
 8:         s_{T_i} ← s*
 9:         e_{T_i} ← e*
10:         if i < n − 1 then propagate(e_{T_i}, ≤, s_{T_{i+1}}) end if
11:     end for
12: end procedure
```

Algorithm 1: Main procedure.

The main procedure, ExistAllenQuadratic (Algorithm 1), takes as arguments the sets $\mathcal{T}$ and $\mathcal{I}$ and a subset $\mathcal{R}$ of Allen's relations. Algorithm 1 considers all tasks in $\mathcal{T}$ and checks the relations according to intervals in $\mathcal{I}$. At the end of the procedure, the bounds of variables in $\mathcal{T}$ are updated according to the earliest support. If no support has been found, a domain is emptied and the solver raises a failure exception. The order between the tasks is maintained by the procedure propagate at line 10, whose propagation is obvious.

Algorithm 1 calls the procedure AllenAllRelation(Algorithm 2, line 6), which performs the check for one task and one interval, with all the relations in $\mathcal{R}$. In order to define a procedure instead of a function returning a pair of bounds, we use two global variables s^* and e^*, storing permanently the two lowest adjustments, respectively for the lower bounds of the starting and ending time of the current task. An adjustment of the bound of one such variable is made in Algorithm 2 if and only if the current relation gives a support which is less than the lowest previously computed support.

```
Require: Global Variable : s*, e*
 1: procedure AllenAllRelation(T, R, I)
 2:     for all r ∈ R do
 3:         if checkCondition(T, r, I) then
 4:             s* ← min(s*, seekSupportStart(T, r, I))
 5:             e* ← min(e*, seekSupportEnd(T, r, I))
 6:         end if
 7:     end for
 8: end procedure
```

Algorithm 2: Update of s^* and e^*.

The function CHECKCONDITION(T, r, I) (line 3) returns true if and only if a support can be found. Consider again the example of relation *starts*. From the condition induced by Table 2, col. 2, we have:

$$\text{CHECKCONDITION}(T,\ s,\ I) \Leftrightarrow \underline{s_T} \leq s_I \wedge \overline{s_T} \geq s_I \wedge \underline{e_T} < e_I.$$

If the conditions are met then a support exists. SEEKSUPPORTSTART(T, s, I) returns the lowest support for the starting time variable, that is, s_I. As no filtering is directly induced for the ending time variable, the minimal support returned, for the relation s by SEEKSUPPORTEND(T, s, I) is $\max(\underline{e_T},\ s_I + \underline{p_T})$. For each Allen's algebra relation, the three functions are similarly derived from Table 2.

Lemma 1. *The time complexity of Algorithm 1 is in $O(n \times m)$.*

Proof. As the number of Allen's relations is constant (at most 13), and the call of the three functions (lines 3, 4 and 5 in Algorithm 2) are in $O(1)$, the whole filtering of lower bounds is performed in $O(n \times m)$ time complexity.

Theorem 1. *The propagator based on Algorithm 1 and its symmetric calls for upper bounds of starting/ending variables in $\mathcal{T}$, ensure* bounds*($\mathbb{Z}$)*-consistency *if processing times are fixed integers.*

Proof. After the execution of Algorithm 1, the optimal update has been done for each lower bound, according to the current upper bounds. The filtering of upper bounds is symmetric. Therefore, providing that durations of tasks are fixed, the propagator ensures *bound*($\mathbb{Z}$)*-consistency* when a fixpoint is reached.

A fixpoint is not necessarily reached after running Algorithm 1 twice, once to filter lower bounds and once to filter upper bounds. Indeed, the pass on upper bounds can filter values which were previously supports for lower bounds. Let's consider ExistAllen($\mathcal{T}, \mathcal{R}, \mathcal{I}$) depicted in Figure 1 wherein: $\mathcal{T} = \{T_1 = \langle s_{T_1} = [0,2], p_{T_1} = [2,4], e_{T_1} = [4,6] \rangle\ ,\ T_2 = \langle s_{T_2} = [5,6], p_{T_2} = [4,5], e_{T_2} = [10,10] \rangle\}$, $\mathcal{R} = \{d, di\}$ and $\mathcal{I} = \{I_1 = [1,5], I_2 = [6,8]\}$.

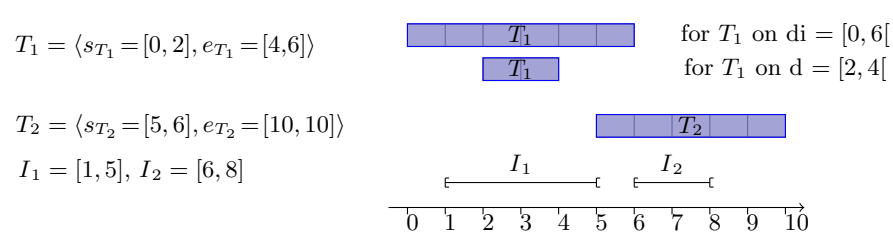


Fig. 1. Several phases may be required to get a fixpoint when the processing times of tasks are not fixed.

We now simulate the calls of Algorithm 1 required to reach a fixpoint. No values are filtered during the run of Algorithm 1 on lower bounds, since di provides the minimal support for $\underline{s_{T_1}}$, d provides the minimal support for $\underline{e_{T_1}}$ and di provides the minimal supports for T_2. On the run of Algorithm 1 on upper bounds, since no relation provides support for 6 from s_{T_2}, the value is removed. The ordering constraint (Algorithm 1, line 10) removes 6 from e_{T_1}. Thus, di is no longer valid to provide a support for T_1. Consequently, a second call to Algorithm 1 on lower bounds has to be done and the minimal value for $\underline{s_{T_1}}$ will then be 2, given by relation d.

However, when task processing times are constants, the minimum support of the ending date of an activity is the minimum support of its starting date plus the constant. Therefore, the issue mentioned in the previous example cannot occur. The fixpoint can be reached in two passes. One may note that, in this case, we could improve the algorithm by ordering Allen's relations. As our target application involves variable processing times, we do not detail this simplification.

3.3 A linear propagator

This section introduces an improved propagator, running in $O(n + m)$ time complexity.

First, one may observe that the satisfaction of the relation *precedes* can be done in constant time. Indeed, if a task T can precede the last interval I_m, the lower bounds are a support. And if not, task T cannot precede any interval. The same way relation *precedes inverse* can be symmetrically checked with the first intervals. Therefore, to simplify the presentation and without loss of generality, we now consider that relations p and pi can be isolated and treated separately. For each task, they can be checked in constant time. In this section we exclude them from $\mathcal{R}$.

Second, as the sets $\mathcal{T}$ and $\mathcal{I}$ are chronologically ordered, we can exploit dominance properties that lead to a linear propagator. We now provide those properties and their proof, as well as the ExistAllenLinear procedure. As in Section 3.2, we focus on lower bounds of starting/ending variables of tasks in $\mathcal{T}$.

Property 1. An interval I which starts after a support y for the ending time of a task T cannot provide a support for e_T lower than y.

Proof. Consider a task T, $\mathcal{R} \subseteq \{m, mi, o, oi, s, si, d, di, f, fi, eq\}$, y a support for e_T and I an interval, such that $y < s_I$. From filtering rules in Table 2, col. 3, no relation in $\mathcal{R}$ provides a support lower than y for e_T. □

Property 2. For any of the relations in $\{mi, oi, s, si, d, f, eq\}$, an interval I which starts after a support y for the ending time of a task T cannot provide a support for s_T lower than y.

Proof. Similar to proof of Property 1. □

Property 3. For any of the relations in $\{m, o, di, fi\}$, an interval I which starts after a support for the ending time of a task T can provide a support for s_T. The lowest support for T is then given by the interval I in $\mathcal{I}$ having the lowest s_I.

Proof. Let T be a task and r be a relation, $r \in R = \{m, o, di, fi\}$, x be a support for s_T, y be a support for e_T and I be the interval in $\mathcal{I}$ with the lowest s_I, such that $s_I > y$. For each $r \in R$, we distinguish two cases.

Case 1 (T is related to I with relation r, $T\ r\ I$). As the support for e_T is at least s_I and no rule on starting time is explicitly defined by r (Table 2, col. 3), then the support for s_T is at least $s_I - \overline{p_T}$. Given that all intervals from $\mathcal{I}$ are chronologically ordered, no interval greater than I can provide a lower x.

Case 2 (T is not related to I with relation r: $\neg(T\ r\ I)$). Consider the m relation. Task T can meet interval I only if $s_I \in [\underline{e_T}, \overline{e_T}]$ (Table 2 col. 2). As it exists a support y with a value $e^* < s_I$ and the relation m is not satisfied, we have $\overline{e_T} < s_I$. Given that all intervals are chronologically ordered, no interval with a greater index can meet the task T. A similar reasoning can be applied to the other relations in R. □

Thanks to Properties 1, 2 and 3, we can improve the Algorithm 1 by stopping the iteration over intervals in $\mathcal{I}$ for a given task T (Line 6) if $e^* < s_I$. We now provide two properties from which, for a given task T, the iteration over intervals in $\mathcal{I}$ does not have to always start at the first interval of $\mathcal{I}$.

Property 4. An interval I which ends before the ending time of a task T_i cannot provide a support for the next task T_{i+1}.

Proof. Let T_i be a task and I an interval such that $e_I < \underline{e_{T_i}}$. Then $e_I < \underline{s_{T_{i+1}}}$. T_{i+1} cannot be in relation with I. □

Property 5. Given a task T, there exists at most one interval between the interval I_i with the highest ending time such that $e_{I_i} < \underline{e_T}$ and the interval I_j with the lowest starting time such that $s_{I_j} > \underline{e_T}$.

Proof. Let T be a task, let I_i be the interval with the highest e_I such that $e_I < \underline{e_T}$, we have then $e_{I_{i+1}} > \underline{e_T}$, and let I_j be the interval with the lowest s_{I_j} such that $s_{I_j} > \underline{e_T}$, we have then $s_{I_{j-1}} < \underline{e_T}$. Given that all intervals are chronologically ordered, and that $s_{I_{j-1}} < e_{I_{i+1}}$, we have that $j - 1 \leq i + 1$, that is $j - i \leq 2$. As the difference between indices i and j is at most 2, there is at most one interval between i and j. □

Thanks to Properties 4 and 5, we know that the next task cannot be in relation with any interval whose index is lower than or equal to $j - 2$. We can improve the Algorithm 1 by starting the iteration over intervals for a given task at $j - 1$.

Require: Global Variable : s^*, e^*
1: **procedure** ExistAllenLinear($\mathcal{T}$, R, $\mathcal{I}$)
2: $j \leftarrow 0$
3: **for** i = 1 to n **do** ▷ Loop over Tasks
4: $s^* \leftarrow \overline{s_{T_i}} + 1$ ▷ Intialize to a value out of the domain
5: $e^* \leftarrow \overline{e_{T_i}} + 1$
6: **repeat** ▷ Loop over Intervals
7: $j \leftarrow j + 1$
8: AllenAllRelation(T_i, $\mathcal{R}$, I_j)
9: **until** $j < m$ and $e^* < s_{I_j}$ ▷ Stop iteration, see Properties 1, 2 and 3
10: $\underline{s_{T_i}} \leftarrow s^*$
11: $\overline{e_{T_i}} \leftarrow e^*$
12: **if** $i < n - 1$ **then** propagate(e_{T_i}, $\leq$, $s_{T_{i+1}}$) **end if**
13: $j \leftarrow max(0, j - 2)$ ▷ Set next interval index, see Properties 4 and 5
14: **end for**
15: **end procedure**

Algorithm 3: Linear Algorithm for Main Procedure.

By construction, Algorithms 3 and 1 do exactly the same filtering.

Theorem 2. *The time complexity of Algorithm 3 is in $O(n + m)$.*

Proof. The number of evaluated intervals is equal to $m+2\times(n-1)$: every time a new task is evaluated, the algorithm goes two intervals back. The new algorithm is then in $O(n + m)$. □

3.4 Improvements

In this section, we describe three improvements brought to the *ExistAllen* constraint in order to improve its efficiency in practice.

First, the Algorithm 2 can be adapted to store, for a given task, the index of the first interval which satisfied the conditions of a relation. Indeed, intervals located before that interval do not provide a support for the task (and they will never do in the current search sub-tree). By doing so, useless calls to check-Condition can be avoided since they will always return false. In practice, an operation is added after the line 7 in Algorithm 3 to put in j the maximum between j and the first satisfying interval for the task evaluated. These indices are automatically updated upon backtrack.

Similarly, the tasks whose variables have been modified since the last call of the procedure have to be memorized. Thus, the for-loop (line 3-14 in Algorithm 3) can start from the index of the first modified task. Moreover, following tasks that have not been modified can be skipped safely.

Finally, our generic framework enables to replace some remarkable combinations of the Allen's algebra relations with *meta-relations*. By doing so, even if the complexity of the Algorithm 3 remains the same, the number of operations made to check conditions and seek supports for the combined relations may be

reduced. For instance, a "*contains*" meta-relation, as described in Example 1, which expresses $\{fi, di, eq, si\}$, can save up to three calls of the methods in the for-loop in Algorithm 2, lines 2-7. Note that since $\{p, pi\}$ are handled in a particular way by the constraint, it is even more efficient to limit the combinations to relations in $\{m, mi, o, oi, s, si, d, di, f, fi, eq\}$. Adding meta-relations is easy in our implementation since we use a facade design pattern to define the 13 relations. Some meta-relations may require to define their inverse, in order to filter on upper bounds. This is not the case for "*contains*", though. Indeed, the mirror relation of fi is si, the mirror relation of si is fi, while the mirror relation of di is di and the mirror relation of eq is eq.

4 Evaluation

The main contribution of this work is an "expressivity gain", which leads to reducing the investment necessary to build and maintain a model. Nevertheless, it is important to check if this gain does not come at the price of efficiency. In this section, we empirically evaluate the impact of the proposed filtering algorithm. First, we recall the video summarization problem. Second, we show that the expressive *ExistAllen* constraint we introduced is very competitive with the state-of-the-art dedicated approach.

4.1 Problem description

The video summarization problem of [4] consists in extracting audio-visual features and computing segments from an input video. The goal is to provide a summary of the video. More precisely, they consider tennis match records.

Several features (games, applause, speech, dominant color, etc.) are extracted as a preprocessing step, in order to compute time intervals that describe the video. Then, the problem is to compute segments of the video that will constitute the summary. The number of segments to compute and their minimal and maximal duration are given as parameter, as well as the summary duration. In this case-study, the purpose is to build a tennis match summary with a duration between four and five minutes, composed of ten segments, whose duration varies between 10 and 120 seconds.

In order to obtain a good summary (from the end-user point of view), this process is subject to constraints, such as:

- (1a) a segment should not cut a speech interval,
- (1b) a segment should not cut a game interval,
- (2) each selected segment must contain an applause interval,
- (3) the cardinality of the intersection between the segments and the dominant color intervals must be at least one third of the summary,

On the assumption that an applause indicates an interesting action, the presence of applause in the summary must be maximized, *i.e.,* the cardinality of the intersection between the computed segments and the applause time intervals must be as large as possible.

Table 3. Match features.

Name	Total duration	# Speech	# Applause	# Dominant color	# Games
M_1	2h08	571	271	1323	156
M_2	1h22	332	116	101	66
M_3	3h03	726	383	223	194

4.2 Benchmark

We consider the model implementation as well as a 3-instance dataset (see Table 3) kindly provided by Boukadida et. al. [4].

The model is encoded using integer variables. A segment is represented by three variables to indicate its start, duration and end. If constraints (1a) and (1b) are easily ensured by forbidding values for the segment start and end variables, most constraints have been encoded using ad hoc propagators. This holds on constraint (2), whereas it could be handled with a single *ExistAllen* constraint wherein $\mathcal{T}$ is the set of selected segments, $\mathcal{R}$ is equal to $\{fi, di, eq, si\}$ and $\mathcal{I}$ is the set of applause time intervals. Therefore, to evaluate the practical impact of the linear-time *ExistAllen* propagator, four models are considered.

1. `decomp`: constraint (2) is explicitly represented by the disjunction depicted in Section 3, Definition 1, using primitive constraints of the solver.
2. `allen(n.m)`: constraint (2) is represented by an *ExistAllen* constraint, using the quadratic propagator presented in section 3.2,
3. `allen(n+m)`: constraint (2) is represented by an *ExistAllen* constraint, using the linear propagator described in sections 3.3 and 3.4.
4. `ad hoc`: constraint (2) is represented with the dedicated constraints of [4]. Such model is given for reference only as neither its complexity nor its consistency level are known.

Each of the four models has been implemented in Choco-3.3.0 [16].[4] Each of the instances was executed with a 15 minutes timeout, on a Macbook Pro with 8-core Intel Xeon E5 at 3Ghz running a MacOS 10.10, and Java 1.8.0_25. Each instance was run on its own core, each with up to 4096MB of memory.

In order to compare the efficiency on the four models, we first consider a static search heuristic: the variables representing the segment bounds are selected in a lexicographic order and assigned to their lower bounds. In this way, we ensure that the same search tree is explored, finding the same solutions in the same order, and that only the running time is evaluated. The comparative evaluations of the four models are in reported in Figure 2. Each plot is associated with an instance and indicates the improvement of the objective function over time. Recall that the x-axis are in logscale. The three plots are similar.

First of all, a strict reformulation (`decomp`) is clearly not competitive with the models with specific constraints: `decomp` is always the slowest. This has to

[4] `http://www.choco-solver.org`

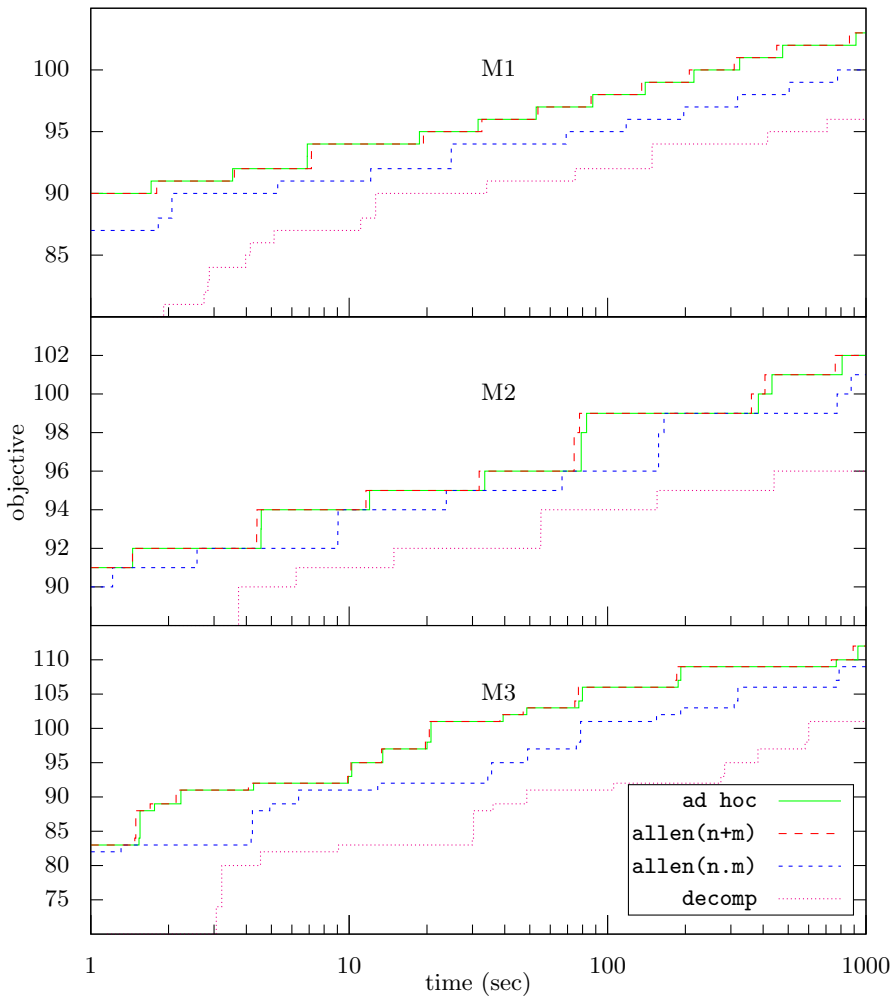


Fig. 2. Comparative evaluation of `decomp`, `ad hoc`, `allen(n.m)` and `allen(n+m)` on the three matches $M1$, $M2$ and $M3$ with a static search heuristic. The plots report the evolution of the objective value to be maximized with respect to the cpu time in seconds. The x-axis are in logscale.

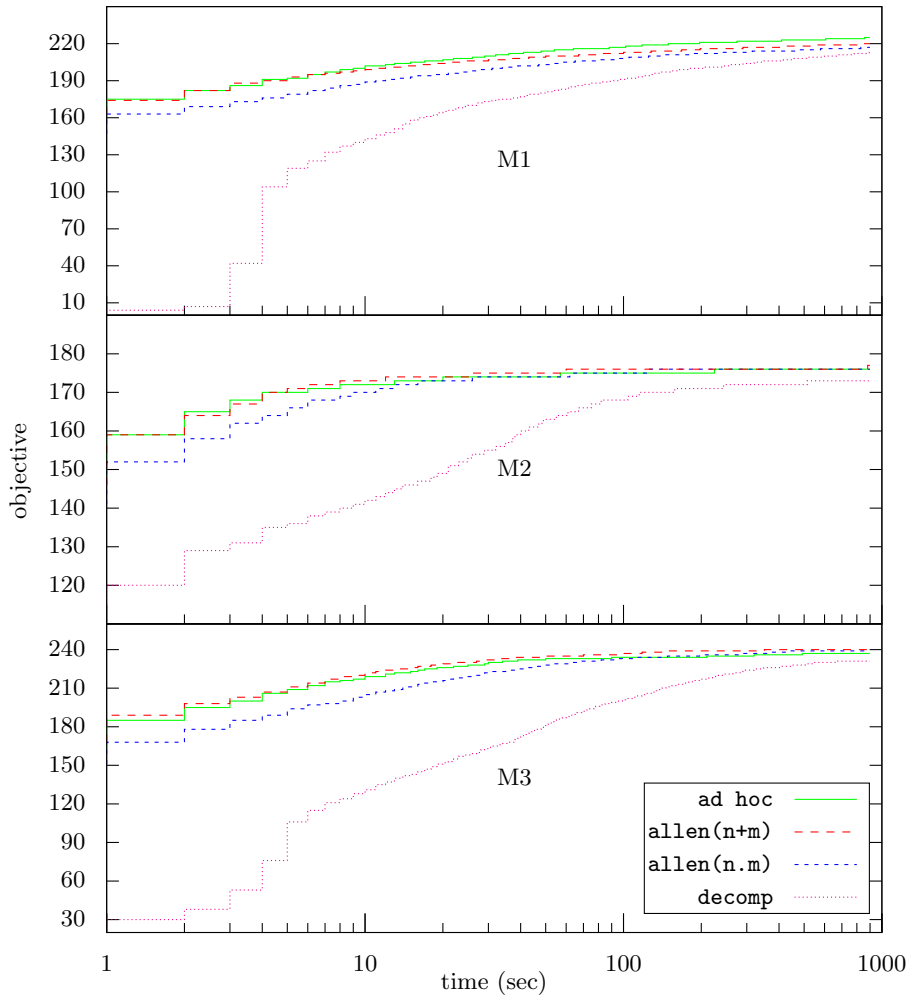


Fig. 3. Comparative evaluation of `decomp`, `ad hoc`,`allen(n.m)` and `allen(n+m)` on the three matches $M1$, $M2$ and $M3$ with ABS and PGLNS. The plots report the evolution of the objective value to be maximized with respect to the cpu time in seconds. The x-axis are in logscale.

do with the huge number of constraints and additional variables it requires to express the *ExistAllen* constraint. As an example in the match $M1$ where there are 10 tasks, 271 intervals and 4 relations, each triplet $\langle T, R, I \rangle$ is expressed by three new binary constraints, each of them reified by a new boolean variable. Second, the quadratic propagator improves the filtering of a strict reformulation. Third, as expected, the performances are even better with the linear propagator. For information purpose, the results of the `ad hoc` model (as used in [4]) are reported. `allen(n+m)`, our generic constraint is shown to be the most effective model, mildly faster than `ad hoc`. This confirms that our generic linear propagator, besides being expressive and flexible and offering guarantees on its complexity and consistency level, is very efficient in practice.

The four models have also been evaluated with the configuration described in [4], that is, using Activity-based search [13] combined with Propagation-Guided Large Neighborhood Search [14].[5] Due to the intrinsic randomness of ABS and PGLNS, each resolution was run 30 times. Thus, the average objective values are reported in Figure 3. Recall that the x-axis are in logscale. The dynamic strategy offered by the combination of ABS and PGLNS enables to reduce the differences between the various models. Although to a slightly lesser extent, the order between efficiency of the four models is preserved when applying a more aggressive search strategy heuristic.

5 Conclusion

We introduced *ExistAllen*, a generic constraint defined on a vector of tasks and a set of disjoint intervals, which applies on any of the 2^{13} combinations of Allen's algebra relations. This constraint is useful to tackle many problems related to time intervals, such as the video-summarization problem [4], used as a case study. From a technical viewpoint, we proposed a generic propagator that achieves bound-consistency in $O(n + m)$ worst-case time, where n is the number of tasks and m the number of intervals, whereas the most natural implementation requires $O(n \times m)$ worst-case time. Our experiments demonstrate that using our technique is very competitive with the best ad hoc approach, specific to one particular combination of relations, while being much more expressive.

Future work includes the extension of this approach to several task sets, in order to tackle problems beyond the context of video-summarization. In the Satellite Data Download Management Problem [15], Earth observation satellites acquire data that need to be downloaded to the ground. The download of data is subject to temporal constraints, such as fitting in visibility windows. Using *ExistAllen* constraint, the visibility windows are the intervals, the amount of acquired data fixes the duration of the download tasks, and the relation required is *during*. The tasks can be ordered with respect to the type of acquired data and their need to be scheduled.

[5] The configuration for ABS and PGLNS are the default one described in [13] and [14].

Acknowledgements The authors thank Haykel Boukadida, Sid-Ahmed Berrani and Patrick Gros for helping us modeling the tennis match summarization problem and for having providing us their dataset.

References

1. James F. Allen. Maintaining knowledge about temporal intervals. *Commun. ACM*, 26(11):832–843, 1983.
2. Sid-Ahmed Berrani, Haykel Boukadida, and Patrick Gros. Constraint satisfaction programming for video summarization. In *Proceedings of the 2013 IEEE International Symposium on Multimedia*, ISM '13, pages 195–202, Washington, DC, USA, 2013. IEEE Computer Society.
3. C. Bessière. Constraint propagation. Research report 06020 (Chapter 3 of the Handbook of Constraint Programming, F. Rossi, P. van Beek and T. Walsh eds. Elsevier 2006.), LIRMM, 2006.
4. Haykel Boukadida, Sid-Ahmed Berrani, and Patrick Gros. A novel modeling for video summarization using constraint satisfaction programming. In George Bebis, Richard Boyle, Bahram Parvin, Darko Koracin, Ryan McMahan, Jason Jerald, Hui Zhang, Steven M. Drucker, Chandra Kambhamettu, Maha El Choubassi, Zhigang Deng, and Mark Carlson, editors, *Advances in Visual Computing - 10th International Symposium, ISVC 2014, Las Vegas, NV, USA, December 8-10, 2014, Proceedings, Part II*, volume 8888 of *Lecture Notes in Computer Science*, pages 208–219. Springer, 2014.
5. Philip Bramsen, Pawan Deshp, Yoong Keok Lee, and Regina Barzilay. Finding temporal order in discharge summaries. pages 81–85, 2006.
6. Berthe Y. Choueiry and Lin Xu. An efficient consistency algorithm for the temporal constraint satisfaction problem. *AI Commun.*, 17(4):213–221, 2004.
7. Rina Dechter, Itay Meiri, and Judea Pearl. Temporal constraint networks. *Artif. Intell.*, 49(1-3):61–95, 1991.
8. ZeinAlAbidin Ibrahim, Isabelle Ferrane, and Philippe Joly. Temporal relation analysis in audiovisual documents for complementary descriptive information. In Marcin Detyniecki, JoemonM. Jose, Andreas Nrnberger, and C.J. van Rijsbergen, editors, *Adaptive Multimedia Retrieval: User, Context, and Feedback*, volume 3877 of *Lecture Notes in Computer Science*, pages 141–154. Springer Berlin Heidelberg, 2006.
9. Manolis Koubarakis. From local to global consistency in temporal constraint networks. *Theor. Comput. Sci.*, 173(1):89–112, 1997.
10. Manolis Koubarakis and Spiros Skiadopoulos. Querying temporal and spatial constraint networks in PTIME. *Artif. Intell.*, 123(1-2):223–263, 2000.
11. T. K. Satish Kumar, Marcello Cirillo, and Sven Koenig. Simple temporal problems with taboo regions. In Marie desJardins and Michael L. Littman, editors, *Proceedings of the Twenty-Seventh AAAI Conference on Artificial Intelligence, July 14-18, 2013, Bellevue, Washington, USA*. AAAI Press, 2013.
12. Peter B. Ladkin. Satisfying first-order constraints about time intervals. In Howard E. Shrobe, Tom M. Mitchell, and Reid G. Smith, editors, *Proceedings of the 7th National Conference on Artificial Intelligence. St. Paul, MN, August 21-26, 1988.*, pages 512–517. AAAI Press / The MIT Press, 1988.
13. Laurent Michel and Pascal Van Hentenryck. Activity-based search for black-box constraint programming solvers. In *Integration of AI and OR Techniques in Contraint Programming for Combinatorial Optimzation Problems - 9th International*

Conference, CPAIOR 2012, Nantes, France, May 28 - June1, 2012. Proceedings, pages 228–243, 2012.

14. Laurent Perron, Paul Shaw, and Vincent Furnon. Propagation guided large neighborhood search. In *Principles and Practice of Constraint Programming - CP 2004, 10th International Conference, CP 2004, Toronto, Canada, September 27 - October 1, 2004, Proceedings*, pages 468–481, 2004.
15. Cédric Pralet, Gérard Verfaillie, Adrien Maillard, Emmanuel Hebrard, Nicolas Jozefowiez, Marie-José Huguet, Thierry Desmousceaux, Pierre Blanc-Paques, and Jean Jaubert. Satellite data download management with uncertainty about the generated volumes. In *Proceedings of the Twenty-Fourth International Conference on Automated Planning and Scheduling, ICAPS 2014, Portsmouth, New Hampshire, USA, June 21-26, 2014*, 2014.
16. Charles Prud'homme, Jean-Guillaume Fages, and Xavier Lorca. *Choco3 Documentation*. TASC, INRIA Rennes, LINA CNRS UMR 6241, COSLING S.A.S., 2014.
17. Frank Schilder. Event extraction and temporal reasoning in legal documents. In Frank Schilder, Graham Katz, and James Pustejovsky, editors, *Annotating, Extracting and Reasoning about Time and Events*, volume 4795 of *Lecture Notes in Computer Science*, pages 59–71. Springer Berlin Heidelberg, 2007.
18. Peter van Beek and Robin Cohen. Exact and approximate reasoning about temporal relations. *Computational Intelligence*, 6:132–144, 1990.
19. Marc Vilain, Henry Kautz, and Peter van Beek. Readings in qualitative reasoning about physical systems. chapter "Constraint Propagation Algorithms for Temporal Reasoning: A Revised Report", pages 373–381. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1990.
20. Li Zhou and George Hripcsak. Methodological review: Temporal reasoning with medical data-a review with emphasis on medical natural language processing. *J. of Biomedical Informatics*, 40(2):183–202, April 2007.