A Language for Audiovisual Template Specification and Recognition

Jean Carrive[1], Pierre Roy[2], François Pachet[3], Rémi Ronfard[1]

[1] INA, 4 avenue de l'Europe
94 366 Bry-sur-Marne, France
{jcarrive, rronfard}@ina.fr

[2] INRIA, Domaine de Voluceau, Rocquencourt
78 153 Le Chesnay Cedex, France
Pierre.Roy@lip6.fr

[3] SONY CSL Paris, 6 rue Amyot
75 005 Paris, France
pachet@csl.sony.fr

Abstract. We address the issue of detecting automatically occurrences of high level patterns in audiovisual documents. These patterns correspond to recurring sequences of shots, which are considered as first class entities by documentalists, and used for annotation and retrieval. We introduce a language for specifying these patterns, based on an extension of Allen's algebra with the regular expression operator +, which denotes an iteration of arbitrary length. We propose a formulation of this pattern language using the constraint satisfaction framework, in which templates are represented as constraint problems. We propose an efficient representation of domains (all subsequences of a given graph) and filtering methods for the Allen constraints. We illustrate the resulting system on a corpus of real world news broadcast examples.

1 Introduction

Indexing and retrieving the contents of temporal media such as audio or video can be made more effective by associating manual annotations (metadata) at various temporal scales in the media. This process is known as *analytic indexing* and is notoriously time consuming, for two reasons. First, it is often difficult to state the rules to be followed in segmenting the media: how many segments ? how many levels of details ? Second, even when such rules can be set, the segmentation remains repetitive and time-consuming. In this paper, we present a framework designed to facilitate the two tasks of *defining* and *recognizing* temporal structures in video. An application of this framework is presented in the context of the DiVAN project, a prototype audiovisual digital library management system financed by ESPRIT.

In recent years, great effort has been made on automatically extracting low-level features from the video and audio streams [1]. Such tools allow for instance, for the visual part, to segment a video into *shots*, which are filmed in one single run of the camera, to detect gradual transitions between shots, as *dissolves* or *wipes,* to detect occurrences of a given logo or to identify regions of the screen corresponding to captions or faces. For the audio part, extraction typically consists in discriminating sound segments such as music or speech, or detecting occurrences of a given jingle in the audio track. Unfortunately, the resulting segments and features are not directly exploitable by documentalists, for two reasons. First, the segments extracted are of short duration. Even if a shot provides more synthetic information than a frame, it is still too "low-level" to be used as a reference for annotation. Second, extracted features usually bear too little semantic to be directly annotated. For instance, a face region detected in a shot makes no sense to the documentalist as such, and will be useful only if more contextual information is provided, to infer, e.g. that the face is indeed the reporter's face during a report, and that the text region on the bottom left of the screen is the reporter's name. The aim of *macro-segmentation* is precisely to extract higher level features and sequences from low level descriptors.

Macro-segmentation methods have been proposed for grouping together shots into longer and more meaningful segments, such as sequences or scenes. For example, [2] proposes an unsupervised algorithm which clusters shots into classes according to an image similarity measure; [3] present a rule-based approach to detect scene boundaries, founded for instance on empirically observed regularities concerning alternations of gradual transitions and cuts in traditional movies. At the opposite of these general methods, specific methods have been designed for specific types of documents, especially news broadcasts [4].

In this paper, we are interested in *collections* of documents which share common characteristics, such as anchor persons, sets, graphics, and which follow a common general scenario. Such collections can be for instance the 6:30 p.m. news broadcast on a given channel during 1998, or some specific Saturday night variety show. These documents generally present typical regularities which can be used for macro-segmentation. For instance, the temporal structure of short news broadcasts is often a succession of reports, a characteristic audio jingle being heard between reports which indicates the beginning of a new report, as illustrated by Fig.1. Reports can be detected provided that the document has been segmented into shots and that occurrences of the jingle have been detected.

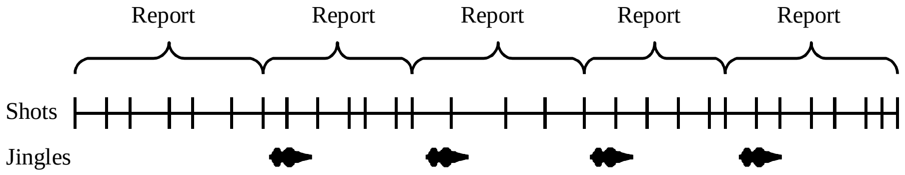

Fig. 1. Reports in a simple news broadcast

In this paper, we are interested in describing such sequences and in detecting occurrences of these sequences automatically in the video.

1.1 Requirements and examples

In our context of automatic video indexing, a video sequence is made up of segments coming from automatic analysis tools. Those tools produce classified segments which are the primitive terms of our language. The primitive classes – or *analysis classes* – are for instance *Shot*, *Jingle*, *Text Region* or *Face Region*, for which there exist robust extraction algorithms [5, 6]. We further organize analysis classes hierarchically. For instance, the face region detection algorithm produces regions of screen containing a human face. Depending on the relative size of the region, shots can be classified into so-called *shot values*, which range from close-up (CU) where the face occupies approximately half of the screen, to the long shot (LS) where the human body is seen entirely. Intermediate shot values are medium close-up (MCU), medium shot (MS) and medium long-shot (MLS). In our case, primitive segments and their classes are represented in a Description Logic formalism, using the CLASSIC system [7, 8].

Classes of *sequences* can be defined by giving information on the temporal arrangement of primitive segments. Let us give three typical examples of such sequences.

Example 1: a simple one shot sequence. Fig.2 illustrates a simple sequence made up of only one medium close-up shot *during which* some text is displayed at the bottom left of the screen. In the “France 2” evening newscast from which the example is extracted, the text usually contains the name of the person on-screen, so this shot can be classified as *NamedPersonShot.*

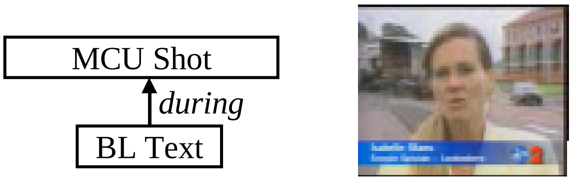

Fig. 2. *Named Person Shot* in a “France 2” evening newscast

Example 2: a sequence with a negative property. Some specific configurations can not be expressed by Allen’s temporal relation only. Consider the example of *reports*, illustrated by Fig.1. The boundaries of a report are roughly defined by two *successive* occurrences of a jingle. Two jingles j_1 and j_2 are successive if j_1 is *before*[1] j_2, and if there is no other jingle j_3 such that j_3 is *before* j_2 and j_1 is *before* j_3. To express this relationship, we need to introduce negation in our description language, with expressions such as: “there must be no instance of class C between some components n_1 and n_2 of the sequence”.

[1] *before* is one of the 13 Allen’s basic relations, which are presented below.

Example 3: *an iterated sequence.* The broadcast news illustrated by Fig.1 is made up of a set of contiguous reports (themselves made up of a set of contiguous shots bounded by two jingles). This example illustrates the need for specifying contiguous sequences of segments of arbitrary length. We define the notion of *iterated sequence* of class *C* as a sequence of contiguous segments of class *C*. Two segments are contiguous if they are in the *meets* Allen's basic relation. By definition, an iterated sequence contains at least one element, and its temporal extension is the temporal union of the temporal extensions of its elements.

1.2 State of the Art

Temporal information holding between the components of a sequence are given by Allen's temporal relations [9]. A temporal relation is defined from 13 basic relations which represent all the possible topological arrangements of two intervals placed on an oriented axis: *before* [b], *meets* [m], *overlaps* [o], *during* [d], *starts* [s], *finishes* [f] and their symmetric relations, *after* [a], *is-met* [mi], *is-overlapped* [oi], *is-during* [di], *is-started* [si], *is-finished* [fi], plus *equals* [e] which is its own symmetric relation. Common intuitions on relative temporal arrangement of two temporal objects can be expressed by temporal disjunctions of these basic relations. For instance, temporal inclusion may be expressed as {*starts* ∨ *during* ∨ *finishes* ∨ *equals*}. The complete Allen's interval algebra includes 2^{13} relations.

As we have already mentioned, primitive segments and classes are represented in a Description Logic (DL) formalism. In such a formalism, classes are described by *concepts* which are automatically organized into a taxonomy according to a generality – *subsumption*, or *is-a* – link, and segments are represented by *individuals* which are automatically classified by computing their most specific parents in the taxonomy. Concepts and individuals are defined by *descriptions* using syntactic operators. DL systems are usually unable to deal with temporal information, or more generally with *part-of* relations. Some propositions have been made to extend a DL language with temporal operators [10]. These works are mostly theoretical and no effective methods are designed for classifying temporal objects. In [11, 12], Weida proposes in the context of plan recognition a Constraint Satisfaction Problem (CSP) approach in which plans to be recognized are represented as *terminological constraint networks* which are constraint networks whose vertices are associated with concepts in a DL framework and whose edges are temporal relations. Weida, however, concentrates on the plan subsumption problem and the recognition is realized by a straightforward node-to node matching process. Moreover, Weida's language can accommodate only example 1, and does not address negation and iterative sequences.

The remaining of this paper is organized as follows. In the following of this section, we present our language for definition of *templates*. In section 2, we describe the template matching problem as a constraint satisfaction problem, we introduce a representation of domains which allows to represent implicitly arbitrary subsequences of the observation graph and we introduce filtering methods for temporal constraints and specific constraints. We describe these filtering methods as a set of demon procedures. In Section 3, we report on the use of our system on real world examples.

1.3 A Language for describing templates

Following the CSP approach pioneered by Weida we propose a language for describing classes of audio visual sequences, called *templates*. The goal of this language is to be flexible enough to accommodate all examples of typical video sequences such as the examples given above. This flexibility leads us to give up template subsumption, which is not necessary in our context.

We therefore propose to extend Weida's formalism with specific constraints and with iterated sequences. For instance, a *NamedPersonShot* in Fig.2 is defined by the following template expression:

```
NamedPersonShot        constraints
   s1[BLText]              s1 d s2
   s2[MCUShot]
```

This example uses only an Allen relation and could therefore be expressed using Weida's formalism. The following expression is the template definition of a report in the broadcast news illustrated by Fig.1:

```
Report                 constraints
   s1[Shot]                s1 m s2
   s2[Shot+]               s2 m s3
   s3[Shot]                s1 {o s si di} s4
   s4[Jingle]              s3 {o s si di} s5
   s5[Jingle]              s1 s this
                           s2 f this
                           no Jingle between s4 s5
```

The '+' symbol in the definition of s_2 indicates an iterated sequence of shots; temporal relations between braces indicate a disjunction of Allen's basic relation; the "this" keyword is used to set constraints between the instances of the Report template and their components. Thus, the temporal extension of a report includes the first jingle, during which important information such as titles may be given, but not the last one, which is part of the next report. The last constraint in the definition indicates that no instance of the Jingle analysis class should appear between the jingles matched with the s_4 and s_5 vertices. The temporal relations between s_1 and s_4 (or between s_3 and s_5) state that the shot matched by s_1 must be the earliest shot having a non empty temporal intersection with the jingle matched by s_4.

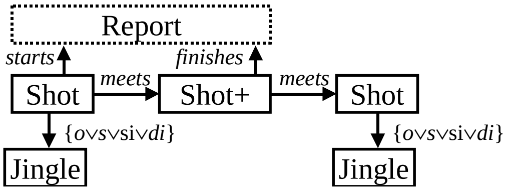

Fig. 3. Graphical representation of a "Report" template

Note that this templates doesn't fit for the first and the last report, which have to be dealt with specifically.

1.4 Definition and notation of templates

A template is a graph whose vertices are associated to classes or, recursively, other templates. Additionally, a vertex can represent an iterated sequence of instances of an analysis class or template. This vertices are called *iterated vertices*. Schematically, vertices of a template represent elements or sub-sets of the observation graph. The edges of a template represent the temporal relations to be satisfied between the observations matched by the vertices. Some additional constraints may be set between the vertices of a template, such as a constraint forbidding that an instance of some class *C* appear between the observations matched by two of the vertices (see example 2). Finally, the temporal extension of instances of a template may be defined by setting temporal constraints in the template definition between the instance itself and its components. We use the following notation:

- n : a vertex in a template definition
- n_C : a non iterated vertex associated with an analysis class C
- n_C^+: an iterated vertex associated with an analysis class C
- n_T: a non iterated vertex associated with a template T
- n_T^+ : an iterated vertex associated with a template T
- *OG* (*observation graph*) is the set of observations

Let t' be a set of observations, $t' \subset OG$. t' is an *instance* of template T', noted $t' \prec T'$, if and only if:

- every n_C of T' is matched with some observation o, $o \in$ t', $o \prec$ C
- every n_C^+ of T' is matched with some iterated sequence $o = \{o_1,\dots,o_m\}$, $o \subset t'$, $o_i \prec C$, $\forall\, i \in [1;m]$
- every n_T of T' is matched with some set of observations t, $t \subset$ t', $t \prec T$
- every n_T^+ of T' is matched with some iterated sequence $t = \{t_1,\dots, t_m\}$, $t \subset t'$, $t_i \prec T$, $\forall\, i \in [1;m]$
- temporal constraints defined by the edges of *T*, and specific constraints defined in *T*, are satisfied.

The goal of our study is therefore the following: given 1) an observation graph and 2) a template definition, find all the instances of the template in the observation graph.

1.5 Embedding templates

As we have seen, a template vertex may be associated with an analysis class or with another template. In this latter case, temporal constraints set on such a vertex have to be propagated on other vertices. The templates illustrated by Fig.4 define *SimpleReport* as two contiguous shots with a jingle being heard during the first shot, an *TwoReports* as two contiguous simple reports.

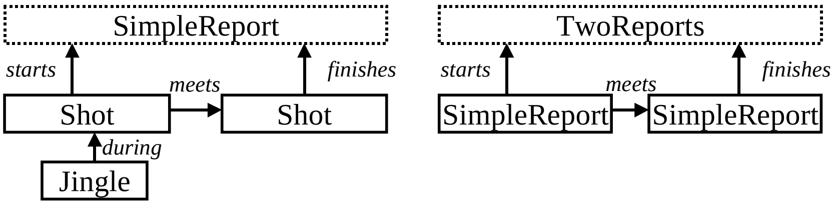

Fig. 4. Embedded templates in a template definition

The initial observation graph is only made of instances of analysis classes. In order to recognize embedded templates, one solution would consist in first recognizing instances of the *SimpleReport* template, then to add them into the observation graph, and finally to recognize instances of the *TwoReports* template. This supposes that the temporal extension of instance of *SimleReport* could be precisely computed from the temporal constraints set on "this" in the template definition, which is not necessarily the case. We thus choose to expand the components of the *SimpleReport* template into the *TwoReports* template. In that case, temporal constraints have to be propagated so that the definition of the whole constraint graph is complete. For reasons of simplicity, we choose to propagate all constraints using the potentially non complete 3-consistency algorithm proposed by Allen to minimize the temporal constraint network [9]. The resulting definition of the *TwoReports* template is illustrated by Fig.5.

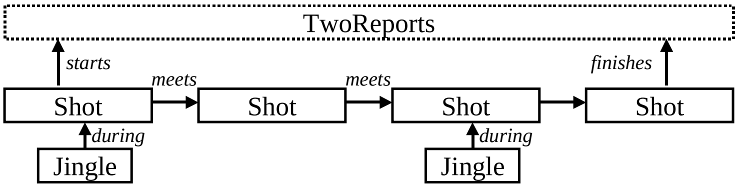

Fig. 5. Expansion of embedded templates

In the case of iterated vertices associated with templates, sub-templates can not be expanded in the same way. We adopt in this case the first solution mentioned for non iterated vertices associated with a template, which consist in first recognizing instances of embedded templates and to add them in the observation graph. This leads to limit the templates that can be iterated to so-called *bounded* templates, *i.e.*

templates for which the temporal extension of instances can always be computed. Bounded templates are roughly templates which are in one of the Allen's basic relation *equals*, *starts*, *meets*, *finishes* – or their symmetric relations – with some of their components.

2 Template matching as a Constraint Satisfaction Problem

We represent the template matching problem by a constraint satisfaction problem. In this section, we focus on the representation of domains, which is of key importance for the filtering procedures of constraints. For reasons of clarity, we will concentrate in this section on templates whose vertices are not associated with sub-templates but only with analysis classes.

2.1 Representation of domains

Each template is represented by a set of constrained variables. Variables representing non iterated vertices have a straightforward domain: all the nodes of the observation graph which are instances of the class associated with the vertex. For each non iterated vertex n_C of a template T, we create a variable v_C associated with n_C, whose domain is $dom(v_C) = \{x \in OG \mid \mathrm{x} \prec C\}$.

The problem is to represent the domains of iterated variables v_C^+ which are associated with each iterated vertices n_C^+ of the template. In principle, the domain of an iterated variable v_C^+ is the set of all possible iterated sequences made up of instances of C.

Of course, this set is very large, and grows exponentially with the number of observations. In the worst case – which is fortunately unlikely in our context – a observation set could happen to be made of N^2 nodes organized as in Fig.6. In this case, the number of sequences of length p is $(N-p+1)\times N^p$.

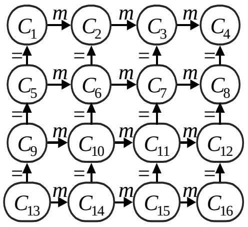

Fig. 6. Worst case with 16 observations

However, one can observe that some of these iterated sequences are subsequences of others. For a given iterated sequence made of N observations associated with class C, the number of possible iterated subsequences is: N subsequences of length 1, (N-1) subsequences of length 2, …, $N-p+1$ subsequences of length *p*, …, and 1 subsequence of length *N*. The total number of subsequences is therefore:

$$\sum_{i=1}^{N}(N-i+1)=\frac{N(N+1)}{2} \qquad (1)$$

This observation may be exploited to yield an efficient representation of the set of all possible iterated sequences. We call a *maximal sequence* an iterated sequence which is not a subsequence of another iterated sequence. For a given class *C*, we first compute the set of maximal sequences. We then represent implicitly each possible iterated sequence as a subsequence of one of the maximal sequences.

The computation of the complete set of maximal sequences is performed by a standard graph search algorithm: starting with the set of all edges between two contiguous instances of *C*, the algorithm walks through all possible paths from one edge until it finds the extremum vertices and records the corresponding maximal sequence. It then repeat the process until exhaustion of the set. The algorithm is of high theoretical complexity but is in practice (*i.e.* observation graphs are almost linear) very efficient. Furthermore, it is computed only once, for each class *C* appearing in an iterated variable.

In the next section we describe the indexing scheme for describing subsequences of maximal sequences.

2.2 An indexing scheme for subsequences of maximal sequences.

The representation of domains is critical for the efficiency of the resolution. In the CSP, each iterated vertex is represented by a constrained variable whose domain is the set of all iterated sequences, which is potentially huge. Handling and storing all the iterated sequences explicitly would be extremely costly, both in terms of space and time.

Fortunately, each subsequence T can be fairly represented by three integer numbers, namely 1) the index of a maximal sequence S of T, 2) the index of the first element of T in S and 3) the length of T (see Fig.7). This makes up an implicit and more compact characterization of iterated sequences.

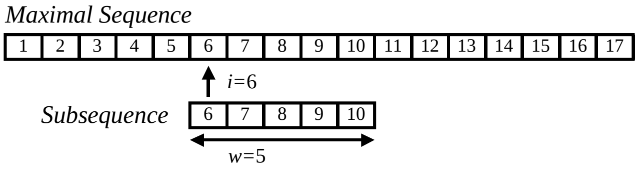

Fig. 7. Subsequences of maximal sequences

Let us present the internal representation of domains in the BackJava system. Integer domains are stored as a list of integer ranges rather than as mere list of integer numbers. For instance, the set {1, 2, 3, 4, 5, 6, 8, 10, 11, 12, 13, 14, 15} is represented by the list {[1,6]; [8]; [10,15]}.

This representation is obviously much more compact than the explicit representation, except for very sparse domains. This is also extremely efficient: primitive operations on domains, such as changing the bounds and removing an element, are much faster than the same operations performed on lists. Note that this representation is particularly efficient for dense integer domains.

An arbitrary domain, that is a domain containing arbitrary objects as opposed to integers, is represented in BackJava by an integer domain and a one-to-one mapping that allows to compute an element from its index in the integer domain, and the way around. For instance, domain {A, B, C, D} will be represented by integer domain {[1,4]} and mapping: A→1; B→2; C→3; D→4.

This representation is very efficient, even though at first sight it may seem more cumbersome and less efficient than the list of elements itself. In effect, domains are stored at each step of the search, and this representation allows us to save only the integer domain because the list of elements and the mapping remain the same throughout the search. This saves a substantial amount of space. The primitive operations on arbitrary domains are the same than operations on integer domains except composed with the mapping. For instance, in our example, to remove element A from, the system simply removes the image of a by the mapping from the integer domain. The overhead represented by the mapping is therefore critical for efficiency.

To efficiently represent iterated domains, we have to design a one-to-one mapping of the three-dimensional implicit representation of subsequences onto integers. We want this mapping, whose description follows, to be fast to compute, and to create as dense an integer domain as possible.

For a given maximal sequence *S* of length *N*, the idea is to sort subsequences of *S* by their length. We then number the sequences starting from 0 as follows:

- from 0 to *N*-1 : subsequences of length 1 ;
- from *N* to 2*N*-2 : subsequences of length 2 ;
- etc.
- $\frac{N(N+1)}{2}-1$: subsequence of length *N*.

Let *F* be the indexing function of subsequences. $F(w,i,N)$ is the index of the subsequence of length w, whose first element is at position *i* in a maximal sequence of length *N*. The index of the first subsequence of length *w*, with *w* ≥ 2, is the number of subsequences of length less than *w*:

$$F(w,0,N)=\sum_{j=1}^{w-1}(N-j+1)=N(w-1)-\frac{(w-1)(w-2)}{2} \qquad (2)$$

Finally, the indexes of subsequences are therefore obtained by:

$$F(w,i,N)=F(w,0,N)+i \qquad (3)$$

In the case of several maximal sequences, the maximal sequences are themselves ordered in a table *Shift*. The indexes of subsequences are systematically shifted by the total number of subsequences of preceding maximal sequences.

Finally, a procedure is applied to remove possible – though unlikely – duplications of subsequences. Indeed, the subsequence representation scheme is redundant, since a given sequence may appear in several maximal sequences. We solve this problem by computing all sequences having multiple definitions, and removing the redundant ones.

This mapping *F* is at the same time easy to compute and very economic (*i.e.* it maps sequences onto a quasi-minimal range). When needed, an explicit representation of sequence *s* is easily computed from its index *k*. The maximal sequence *S* of length *N* to which *s* belongs is determined by scanning the *Shift* table. The length *w* of *s* is given by equation (4) which gives the integer part of the first root of function *F*.

$$w = E\left(\frac{2N-3-\sqrt{(2N-3)^2-8(N+k+1)}}{2}\right) \qquad \textbf{(4)}$$

The index *i* of the first element of *s* in *S* is given by: $i = k - F(w, 0, N)$.

2.3 Filtering procedures for temporal constraints

We solve the CSP using a complete enumeration procedure, interlaced with a domain reduction phase, which allows to speed up the resolution by pruning branches of the search-tree. The domain reduction phase is critical for the overall efficiency of the resolution, especially for large problems such as template matching on large-scale corpuses of documents.

The most widely-used approach to domain reduction is to enforce arc consistency [13] for every constraint at each step of the search. This is referred to as the real-full look-ahead method [14]. Computing arc consistency in general is prohibitive. The alternative is to use a weaker domain reduction scheme: *constraint filtering* [15].

Intuitively, the idea behind constraint filtering is to find the best trade-off between the efficiency (i.e. number of values removed) and the cost of the domain reduction phase. Technically, there are three essential aspects of constraint filtering. First, the property of arc consistency is an ”upper limit” for domain reduction. We can save time by performing weaker domain reduction. Second, since we consider constraint individually, we can adapt the domain reduction procedure according to the nature of the constraint. Third, for a given constraint, we can trigger different domain reduction procedures depending on the states of its variables.

We need to make one more remark before we present our filtering scheme and an example of filtering procedure. When the domains of the variables of a temporal constraint have not been much reduced, the constraint is usually arc consistent. In this situation, it is useless to spend time looking for inconsistent values to remove, whereas when one of the domain has become very small, there are inconsistent values in the other domain.

This remark advocates the use of a filtering scheme in which a constraint is filtered only when one of the domains of its variables have been dramatically reduced.

We chose to trigger filtering methods only when *one* of the variables is instantiated. This is arbitrary, but achieves in practice an efficient compromise. Besides, it allows very simple and fast filtering methods to be designed (as shown by the filtering method for *meets*).

The implementation of our filtering scheme is based on specific objects, called *demons* [16]. A demon is a link between a constraint and a variable, which react to modifications of the state of the variable, such as instantiation, and in turn triggers the corresponding filtering procedure for the associated constraint.

Temporal constraints AllenCt_R (v_1, v_2) are all binary, and parameterized by an arbitrary Allen relation *R*. We distinguish between two cases, depending on the nature of the variable which is *not* instantiated. For the sake of conciseness, we assume that v_1 is the variable that is instantiated – so v_2 is *not* instantiated.

V_2 is non iterated. Disjunctions of Allen relations are represented by a bit array of length 13, a value of 1 meaning that the corresponding Allen relation holds, using an arbitrary ordering of Allen relations. The satisfaction of the constraint is therefore achieved through a logical AND test between the observed temporal relation and the constraint. We will therefore remove from *dom*(v_2) all values which do not satisfy relation *R* with the temporal extension of the value of v_1. Note that we don't have to consider here whether v_1 is iterated or not, for only the temporal extension of the value of v_1 is taken into account.

V_2 is iterated. The problem here is to remove from *dom*(v_2) all the values which do not satisfy relation *R* with (the value of) v_1, without enumerating *dom*(v_2). This method is designed as follows.

Firstly, two main basic access protocols are defined to access the domain of an iterated variable:

- remove(*D*, *s*, *w*, *i*): removes from domain *D* the subsequence of the s^{th} maximal sequence, of length *w*, whose first element is at index *i* in the maximal sequence. This methods removes from *D* the integer: $F(w, i, N_s) + Shift[s]$, with *N* being the length of the s^{th} maximal sequence.
- remove(*D*, *s*, *w*, *b*, *e*): remove from domain *D* all subsequences of the s^{th} maximal sequence, of length *w*, whose first element is between *b* and *e* in the maximal sequences. This methods removes from *D* the interval $[f_1 ; f_2]$, with $f_1 = F(w, b, N_s) + Shift[s]$ and $f_2 = F(w, e, N_s) + Shift[s]$.

Second, we define 13 domain reduction methods, for each basic Allen relation r_i. These methods remove from the domain of v_2 all values which do satisfy r_i with the value of v_1. For instance, the method for the *after* relation is:

```
basicRemove_after(v_1, v_2)
   for s=0 to number of maximal sequences - 1
```

```
    $M_s$ = s^th maximal sequence
    i = index of 1^st element in $M_s$ just *after* val($v_1$)
    for w=1 to length($M_s$) – i
      remove(dom($v_2$), w, i, length($M_S$)-w))
```

For each r_i which is *not* element of *R*, execute basicRemove$_{ri}$ (v_1, v_2), *i.e.* the method which removes from *dom*(v_2) all the values which are in the r_i relation with the value of v_1.

For specific cases, however, we can define more efficient methods, which avoid executing up to 12 of the basic domain reduction methods. For instance, in Fig.8, v_1 is associated with the *Jingle* vertex of the template, and v_2 is associated with *Shot+*. if *v1* is instantiated with *jingle*$_1$, it is possible to remove from *dom*(v_2) all the iterated sequences which do *not* start with S_6.

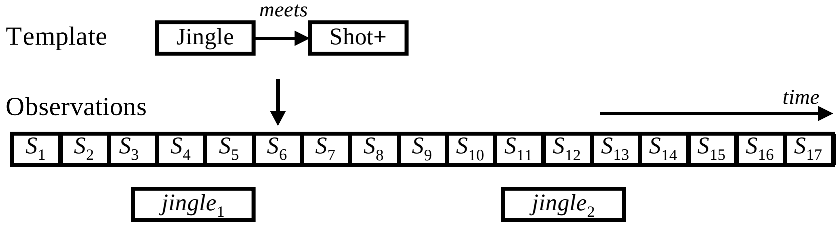

Fig. 8. Domain reduction method for the *meets* relation

Specific methods have been implemented for some Allen's basic relations, as *equals*, *before*, *after*, *meets* or *is-met*. The method for the *meets* relation is given below.

```
keep_meets($v_1$, $v_2$)
  for s=0 to number of maximal sequences - 1
    $M_s$ = s^th maximal sequence
    i = index of 1^st element in $M_s$ which *meets* val($v_1$)
    $N_s$ = length($M_s$)
    for w=1 to $N_s$
      $ind_1$ = 0
      $ind_2$ = min(i-1, $N_s$–w)
      $ind_3$ = i+1
      $ind_4$ = $N_s$-w
      if $ind_2 \geq ind_1$
        remove(D, s, w, $ind_1$, $ind_2$)
      if $ind_4 \geq ind_3$
        remove(D, s, w, $ind_3$, $ind_4$)
```

This particular filtering method actually achieves arc consistency for the constraint when one of the variables is instantiated. The complexity of the general method to achieve arc consistency, as presented in [13] for instance, is linear in the size of the domain of the non-instantiated variable. More precisely, for each maximal subsequence of length *N*, it is linear in the number of subsequences, namely

$\frac{N(N+1)}{2}$.

The complexity of the method we present is, for each maximal sequence, in the length of the sequence, as opposed to $\frac{N(N+1)}{2}$.

These filtering methods as well as the general constraint satisfaction algorithm were implemented in the BackJava system [16].

2.4 Filtering specific constraints

The specific constraints described above as “no instance of class *C* between vertices n_1 and n_2” are filtered in a straightforward way. When one of the variable associated with n_1 or n_2 is instantiated – say v_1 –, we remove from *dom*(v_2) all values *v* such that there exists an instance of *C* *after* the value of v_1 and *before* *v*.

3 Experimentations

In this section, we present two sets of experiments. The first one was made on a corpus of six different evening news broadcasts recorded in 1996, which have been entirely segmented and labeled by hand as a reference [17]. In this first experiment, we had only one exemplar per collection, and the task was to design templates suitable for all collections. The other experiment was made in the context of the DiVAN project, and used five different exemplars of the same evening news collection (Soir3). In this second experiment, the task was to verify that results obtained on one exemplar could be applied to the whole collection.

3.1 Reports from the “M6” broadcast news

The first experiment concerns the recognition of the temporal structure of the evening broadcast news of the “M6” French channel which is illustrated by Fig.1. The template of a report has been described above. As mentioned, specific templates must be defined for the first and the last reports. Provided that the first shot has been classified as *FirstShot*, the template of the first report is given by the following template:

```
FirstReport           constraints
   s1[FirstShot]         s1 m s2
   s2[Shot+]             s2 m s3
   s3[Shot]              s3 {o s si di} s4
   s4[Jingle]            s1 s this
                         s2 f this
                         no Jingle between s1 s3
```

The *LastReport* template is defined in a similar manner. The temporal structure of the whole newscast is simply a succession of a *FirstReport*, an iterated sequence of *Reports* and an *LastReport*:

```
NewsCast                 constraints
   s1[FirstReport]          s1 m s2
   s2[Report+]              s2 m s3
   s3[LastReport]           s1 s this
                            s3 f this
```

We applied the newscast template to a “M6” newscast containing 174 shots and 10 jingles. The recognition process gives exactly 1 matching for the newscast, which is made of 1 first report, a sequence of 9 reports, and 1 last report, which means that all reports have been recognized and that each report was recognized only once. Note that simpler definitions of reports which could seem more intuitive may lead to recognize a great number of reports. Thus, the following template defines a report as a sequence of shots “between” two successive jingles. When applied to the same newscast, this template gives 485 reports.

```
NaiveReport              constraints
   s1[Shot+]                s1 {si oi} s2
   s2[Jingle]               s1 {m o} s3
   s3[Jingle]               s1 e this
                            no Jingle between s2 s3
```

3.2 Reports from the “France 3” broadcast news

In this experiment, we are interested in comparing two ways of recognizing reports from the evening broadcast news of the “France 3” French channel. The first method which serves as a reference, defines a report as the sequence of shots appearing between two successive shots of the anchor person. The second method is used when shots of the anchor person cannot be detected and benefits from the observation that shots of the anchor person generally start or finish with a gradual transition as a dissolve. A report is then defined as the sequence of shots appearing between two successive gradual transitions. A typical edition of this newscast is illustrated by Fig.9.

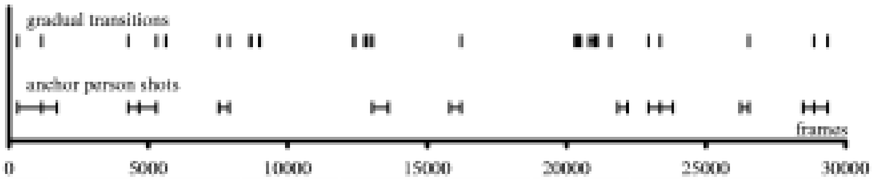

Fig. 9. Shots of the anchor person and gradual transitions in a typical edition of a “France 3” broadcast news

Gradual transitions which appear in the course of a report lead to an over-segmentation of reports; shots or sequences of shots of the anchor person neither started nor finished by a gradual transition lead to unrecognized reports. Shots or sequences of shots of the anchor person both started and finished by gradual transitions lead to misclassified reports. Results obtained in five different editions of the “France 3” broadcast news are summarized in Table 1.

Reports from anchor person	Reports from gradual transitions	Misclassified reports	Missed reports
12	15	3	1
11	24	11	0
10	17	6	1
13	26	4	1
10	31	6	0

Table 1. Recognition of reports using shots of the anchor person, and using gradual transitions

The computation time for all experiments is always under 5 seconds on a PC (266 Mhz), which is largely enough in our context.

Templates for the Soir3 and other collection are currently being investigated in the prototype DiVAN system, with real implementations of all segmentation and classification tools. In this context, the analysis classes and templates associated with a collection are stored in the metadata database, along with their definitions. They are then used to parameterize the automatic macrosegmentation of all programs in the collection. Macrosegments corresponding to a recognized template are presented to the documentalists as candidate database entries, for confirmation and manual annotation.

4 Conclusion

We have described a CSP formulation of a template matching problem in the context of audio visual document indexing. The formulation yields a language for expressing templates which is flexible enough to accommodate for most of the regularities occurring in high level structures. The implementation of the language using the CSP formalism yields an efficient and sound solving procedure. The resulting system improves on existing approach in macro-segmentation, by providing an efficient yet general framework to the issue.

Various aspects of the system may benefit from improvements. For instance, the template completion could be avoided and a graph analysis could allow to propagate only necessary temporal constraints, thereby limiting the number of redundant constraints.

Finally, the experiments showed that templates specified using simple primitives produce macrosegmentations which are useful for documentalists.

Current work focus on the definition of templates for other classes of audiovisual documents, to validate the approach on a large scale.

5 References

1. Brunelli, R., O. Mich, and C.M. Modena, *A Survey on the Automatic Indexing of Video Data*. Journal of Visual and Image Representation, 1999. **10**: p. 78-112.
2. Yeung, M. and B. Liu. *Efficient matching and clustering of video shots*. in *IEEE International Conference on Image Processing*. 1995.
3. Aigrain, P., Joly, P., Longueville, V., *Medium Knowledge-Based Macro-Segmentation of Video into Sequences*, in *Intelligent Multimedia Information Retrieval*, A.P.M. Press, Editor. 1997.
4. Watclar, H.D., *et al.*, *Intelligent Access to Digital Videos: Informedia Project.* IEEE Computer, 1996. **29**(5): p. 46-52.
5. Garcia, C. and G. Tziritas, *Face Detection Using Quantized Skin Color Regions Merging and Wavelet Packet Analysis.* IEEE Transactions on Multimedia, 1999. **1**(3): p. 264--277.
6. Garcia, C. and X. Apostolidis. *Text Detection and Segmentation in Complex Color Images.* in *IEEE International Conference on Acoustics, Speech, and Signal Processing*. 2000. Istambul, Turkey.
7. Carrive, J., F. Pachet, and R. Ronfard. *Using Description Logics for Indexing Audiovisual Documents*. in *Proceedings of the International Workshop on Description Logics*. 1998. Trento, Italy.
8. Borgida, A., Brachman, R.J., McGuiness, D.L., Resnick, L.A. *CLASSIC: A Structural Data Model for Objects*. in *ACM SIGMOD Int. Conf. on Management of Data*. 1989.
9. Allen, J.F., *Maintaining knowledge about temporal intervals.* Communications of the ACM, 1983. **26**: p. 832-843.
10. Artale, A. and E. Franconi, *Temporal Description Logics*, in *Handbook of Time and Temporal Reasoning in Artificial Intelligence (forthcoming)*, v.B. Vila, Boddy, Fisher, Gabbay, Galton and Morris, Editor. 1999, MIT Press.
11. Weida, R., Litman, D. *Terminological Reasoning with Constraint Networks and an Application to Plan Recognition*. in *Proceedings of the Third International Conference on Principles of Knowledge Representation and Reasoning (KR'92)*. 1992. Cambridge, Massachussetts.
12. Weida, R. *Knowledge Representation for Plan Recognition*. in *IJCAI 95 Workshop on the Next Generation of Plan Recognition Systems*. 1995. Montréal, Québec, Canada.
13. Mackworth, A., *Consistency in Networks or relations.* Artificial Intelligence, 1977. **8**(1): p. 99-118.
14. Nadel, B., *Constraint Satisfaction Algorithms.* Computational Intelligence, 1989. **5**: p. 188-224.
15. Roy, P., A. Liret, and F. Pachet. *The Framework Approach for Constraint Satisfaction*. in *ACM Computing Survey Symposium on "Object Oriented Application Frameworks", to be published in March 2000*. 1999.
16. Roy, P., A. Liret, and F. Pachet, *A Framework for Objected-Oriented Constraint Satisfaction Problem*, in *Implementing Application Frameworks: Object-Oriented Frameworks at work*, M. Fayad, D. Schmidt, and R. Johnson, Editors. 1999, Wiley & Sons.
17. Ruiloba, R., *et al. Towards a Standard Protocol for the Evaluation of Video-to-Shots Segmentation Algorithms*. in *First European Workshop on Content-Based Multimedia Indexing (CBMI'99)*. 1999. Toulouse, France.