Temporal Constraints for Video Interpretation

Van-Thinh VU, François BREMOND and Monique THONNAT

INRIA, research group ORION, 2004 route des Lucioles, Sophia Antipolis, 06902, France
{Thinh.Vu, Monique.Thonnat, Francois.Bremond}@sophia.inria.fr
http://www-sop.inria.fr/orion/orion-eng.html

Abstract. This paper presents an original approach for temporal scenario recognition for video interpretation. We propose a declarative model to represent scenarios and we use a logic-based approach to recognize pre-defined scenario models. To speed up the recognition process, we propose a new method to process the temporal operators of interval algebra and a method to extend the time interval of recognized scenarios. We have tested our representation formalism and the inference engine on two real video sequences.

1 Introduction

This paper presents recent works in scenario recognition for Automatic Video Interpretation (see Fig. 1). The goal is to recognize scenarios involved in a scene depicted by a video sequence. The recognition process takes as input (1) scenario models predefined by experts, (2) 3D geometric information of the observed environment and (3) a video stream of persons tracked by a vision module ([2], [18]). To represent scenarios, we first propose a language to describe scenario models and secondly a constraint verification approach to recognize in real time scenario occurrences.

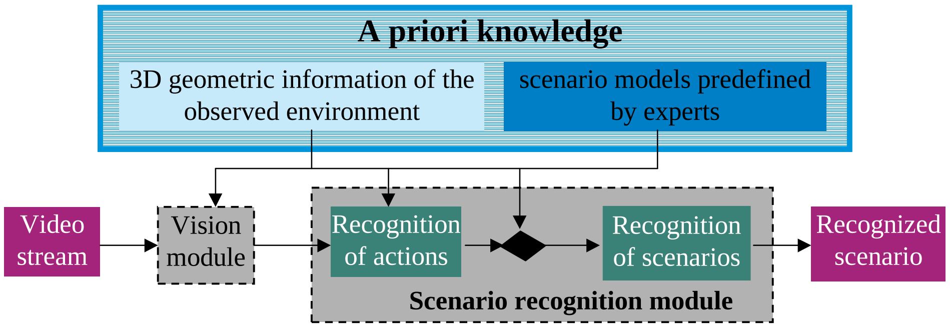
A priori knowledge
3D geometric information of the observed environment
scenario models predefined by experts
Video stream
Vision module
Recognition of actions
Recognition of scenarios
Recognized scenario
Scenario recognition module

Fig. 1. Overview of the Automatic Video Interpretation System.

For 20 years, the issues of temporal scenario representation and recognition have been studied. M. Ghallab [6] has represented a temporal scenario as a set of temporal constraints on time-stamped events. The recognition algorithm keeps and updates partial recognitions of scenarios using the propagation of temporal constraints based on RETE algorithm. C. Pinhanez and A. Bobick [12] have used Allen's interval algebra [1] and presented the specific algorithm reducing its complexity. S. Hongeng and R. Nevatia [7] have proposed a behaviour recognition method that uses the concurrence bayesian threads to estimate the likelihood of potential scenarios. N. Rota and M. Thonnat [14] have used a declarative representation of scenarios defined as a set of spatio-temporal and logic constraints. They have used a traditional constraint resolution technique to recognize scenarios. They have reduced the processing time for the recognition step by checking before the consistency of the constraint network using the AC4 algorithm [11].

The Temporal Constraint Satisfaction Problem (TCSP) has also been studied for a longtime. R. Dechter [3] has represented a scenario with a temporal constraint networks based on time intervals of delay between events. She has also used a path consistency algorithm to solve the network. L. Khatib, P. Morris, R. A. Morris and F. Rossi [10] have presented a method to solve the TCSP using preferences based on the order of the pair of events and the delay between them.

Most of these scenario representation are not easy to use and do not let the experts to describe their scenarios in a natural way. For example, they represent an event defined at one time point and do not manipulate events defined on intervals. All these techniques allow an efficient recognition of scenarios, but they are still some temporal constraints which can not be processed. For example, most of these approaches require that the scenarios are bounded in time.

2 Representation of Scenarios

Our goal is to explicit all the knowledge necessary for the system to be able to recognize scenarios. The description of this knowledge has to be declarative and in natural terms, so that the experts of the application domain can easily define and modify it. Thus, the recognition process uses only the knowledge represented by experts through scenario models.

We represent a scenario model with the list of the actors involved in the scenario and a set of constraints on these actors.

An actor can be a person detected as a *mobile object* by the recognition process or a *static object* of the observed environment. A person is represented by his/her characteristics: his/her position in the observed environment, width, velocity,…. A static object of the environment is defined as a priori knowledge (before processing) and can be either a zone of interest (a 2D polygonal as the entrance zone) or a piece of equipment (a 3D object as a desk). A zone is represented by its vertices. And a piece of equipment is represented by the vertices of its 3D bounding box. The zones and the equipment are represented in the scene context of the observed environment ([2], [18]). Static objects and mobile objects are called *scene-object*.

In our representation, person behaviors are also involved in states, events and scenarios. A state characterizes a person behavior constant during a given time interval.

An event defines a change of state at two successive instants ([2], [14], [18]). The notion of *scenario* is used as a generic name to combine these two notions of states/events. A scenario may be constituted by several sub-scenarios. A scenario involves at least one person, and is defined in a time interval. An interval is represented by its starting and ending time. Defining scenario on time interval is important for the experts to let them describe scenarios in the more natural way. Scenarios and scene-objects are called *entities* and defined by a generic class (see Fig. 2).

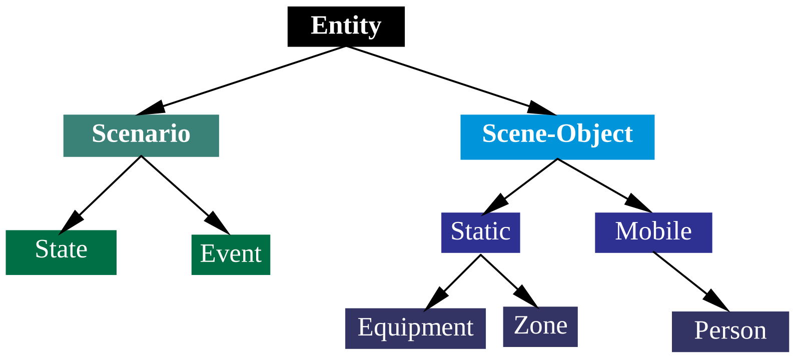

Fig. 2. Five types of entities are classified into "scenario" and "scene-object".

To describe a scenario model, we have used the following definitions.

Definitions:

O - a set of scene-objects,

V - a set of variables corresponding to scene-objects,

C = {c constraint | c : $V^k \rightarrow$ BOOL, k > 0},

$\mathcal{C} = \mathcal{P}(C)$ set of parts of C,

A = {attribute := value} set of affectations,

$\mathcal{A} = \mathcal{P}(A)$ set of parts of A

A n-actor scenario model is an element m = (*Actors*, *Constraints*, *Production*) of M = ($V^n \times \mathcal{C} \times \mathcal{A}$). *Actors* is a set of n variables expressing the actors involved in the scenario with their type and name. *Constraints* is a set of constraints defining the relationships between the actors relevant with the scenario. *Production* is a set of deduced characteristics describing the scenario once it has been recognized.

The n-uplet of scene-objects s = (o_1,…, o_n) is a solution at time t of the recognition process using the scenario model *m* if all the constraints of *Constraints* are satisfied when we assign the n variables of *Actors* with the n corresponding scene-objects of *s*. In this case the scenario model *m* is *satisfied* at the time of the recognition of *s*. A scenario *b* (called instance of *m*) will be produced using *s* with the characteristics given by *Production*. The scenario *b* is *recognized* and is added to the characteristics of the persons contained in *s*.

Fig. 3 shows a model of state "*close_to*": a person *p* is close to an equipment *eq*. It involves two actors a person *p* and an equipment *eq*. There is only one constraint which verifies that *p* is close to *eq*.

```
State(close_to,
   Actors((p : Person), (eq : Equipment) )
   Constraints((distance(p, eq) ≤ Close_Distance))
   Production((s : State)(Name = "close_to")) )
```

Fig. 3. A model of state "close_to": a person is close to an equipment.

We have defined a description language to represent *Constraints* and *Production* in a scenario model. These sets of constraints are expressed by logical predicates.

To express the logical predicates, we use the arithmetical operators (+, -, *, /), the comparison operators (<, ≤, =, != : difference, ≥, >) and the logical operators (! : not, & : and). To represent temporal relations between the scenarios, we use operators of the interval algebra (before, after, meets,…) [1].

We use the spatial operator "distance": to calculate the distance between two scene-objects. We use also the operator "exists" to verify whether given scenarios exist and satisfy several constraints.

A person *p* is considered to stay at an equipment *eq* if he/she is longtime close to the equipment. In Fig. 4, we define a model of this scenario.

```
Event(stays_at,
   Actors((p : Person), (eq : Equipment) )
   Constraints((exists ((state s: p close_to eq))
                    ((Duration of s ≥ 10))) )
   Production((st : Event)
             (Interval of st = Interval of s) ) )
```

Fig. 4. A model of scenario "stays_at": a person stays at an equipment.

A model of event "*moves_close_to*" is shown in Fig. 5: a person *p* moves close to an equipment *eq*. An event of this model will be recognized if p is first far from eq and then close to eq. "(**state** s1: *p* **far_from** *eq*)" expresses that s1 is a state where the person p is far from the equipment eq. The production indicates how to compute the time duration of the recognized scenario.

```
Event(moves_close_to,
   Actors((p : Person), (eq : Equipment) )
   Constraints((exists ((state s1: p far_from eq)
                     (state s2: p close_to eq))
                    ((Duration of s2 ≤ 1)
                     (s1 before s2))) )
   Production((e : Event)
             (Interval of e = Interval of s2) ) )
```

Fig. 5. A model of event "moves_close_to": a person moves close to an equipment.

On Fig. 6 we show an example of a more complex scenario, "vandalism": a person p tries to "break up" an equipment eq. This scenario will be recognized if a sequence of five events described on Fig. 7 has been detected.

```
Scenario(vandlism,
   Actors((p : Person), (eq : Equipment))
   Constraints((exists((event e1: p moves_close_to eq)
                       (event e2: p stays_at eq)
                       (event e3: p moves_away_from eq)
                       (event e4: p moves_close_to eq)
                       (event e5: p stays_at eq) )
                      ((e1 before e2) (e2 meets e3)
                       (e3 before e4) (e4 before e5))))
   Production((s : Scenario)
              (Interval of s = Interval of e5) ) )
```

Fig. 6. A model of scenario "vandalism": a person tries to "break up" an equipment.

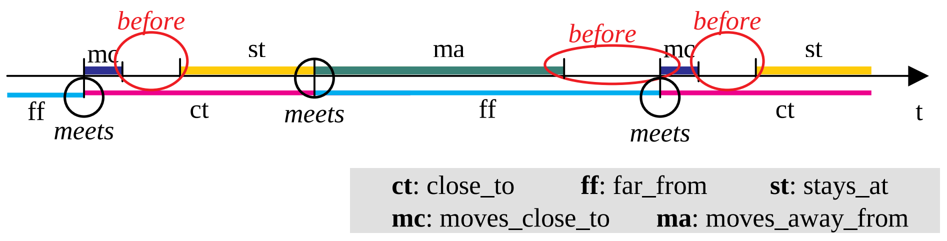

Fig. 7. Temporal constraints for the states and events constituting a scenario "vandalism".

We propose an algorithm (described in the next section) to recognize the scenarios pre-defined by experts using this formalism. The recognition process uses only the knowledge represented through scenario models.

3 Recognition of Scenarios

3.1 Overview of the Recognition Process

The scenario recognition process has to detect which scenario is happening from a stream of observed persons at each frame. The process takes as input (1) the scenario models pre-defined by experts, (2) the geometric information of the observed environment and (3) the persons tracked by a vision module. We suppose that the persons are correctly tracked: their characteristics (their position in the scene, their height,…) are well detected and at two successive frames, two persons having the same name correspond to the same real person.

To recognize the pre-defined scenario models at each frame, we first select a set of triggers that indicate which scenarios can be recognized and secondly we find solutions (set of actors at time ct) for each of these scenarios as described in Fig. 8.

```
Initiate list of triggers LT with
         the elementary models set
while LT ≠ ∅
   tr ← get first element of LT
   LT ← LT - {tr}
   if tr is of the third type of triggers
   then extend(LT, scenario contained in tr, ct)
   else
     m ← scenario model contained in tr
     Find_Solution(LT,list of actor variables of m,m,∅)
```

Fig. 8. The recognition process at each frame.

3.2 Selection of Models to be Recognized

The first step of the scenario recognition process is to compute which scenarios can be recognized at the current time. We call "trigger" such a scenario which can be recognized. There are three types of triggers: (1) the elementary scenarios, (2) more complex scenarios with selected actors and (3) more complex scenarios already recognized at the previous instant (see Fig. 8).

At each frame, we initiate the list LT of triggers with all elementary scenario models. An elementary scenario is a scenario that can be recognized at any time such as "close_to". An elementary scenario is always a state. The list LT is ordered by priority level defined by the experts.

Once we have recognized a scenario we have to add to the list of triggers LT all the more complex scenarios which are ended with the given recognized scenario. For that, before the processing, for each scenario model, we compute the set of *post-models* that correspond to the scenarios that can be recognized once the given scenario has been recognized. For example, once we have recognized the scenario "close_to", it is possible that the scenario "moves_close_to" would be recognized. So the list of post-models of the scenario "close_to" contains the scenario "moves_close_to".

Therefore, when a scenario is recognized, we add to the list of triggers LT its post-models (more complex scenarios) with the actors of the recognized scenario. We can notice that all actor variables of the complex scenarios are not necessary instantiated at this stage. They will be looked for later in the process.

These more complex scenarios with selected actors correspond to the second type of trigger. To compute them, we have defined a trigger model which is composed of two scenario models (pre/post models) with the relationships between actors of the two models. For example, a trigger model corresponding to the scenario model "close_to" and its post-model "moves_close_to" contains two relations: one to tell that the variable person *p* of "moves_close_to" corresponds to the variable person *p* of "close_to", and another one for the variable equipment *eq*.

The third type of trigger corresponds to already recognized and complex scenarios. Once we have recognized a complex scenario at the previous instant and if the sub-scenario ending the complex one is extended, then it is possible that the complex scenario would still be recognized at the current time. Therefore we add to the list of triggers LT the complex scenario with its list of actors and its list of sub-scenarios. The extension process will be described in the next sections.

3.3 Finding Solutions of a Scenario Model

The process of finding all the solutions of a scenario model m, is realized thanks to the function "Find_Solution". This function selects an actor for each actor variable and check whether the selected actors satisfy the constraints defined within the scenario model m. This function is described briefly in Fig. 9. It takes as input the scenario model m, the list of actor variables lav to be instanciated, the list of actors A of the actor variables already instanciated and the list LT of triggers. At the beginning of the process, lav is initialized with all actor variables of m and A is the empty list of actors. When there are no more actor variables (lav is empty list), then A contains one solution (a selection of actors). The side effect of this function is to store the solutions once they are found.

```
Find_Solution(LT, lav, m, A)
   av ← first actor variable of lav
   while select_actor a for av in the domain of
              av[type] and verify_constraints(m, a, av)
      if av is not the last actor variable of lav
      then Find_Solution(LT, lav - {av}, m, A + {a})
      else
         create_instance b of m with A+{a}
         b′ ← search b in the recognized scenario set
         if success and if b is consecutive to b′
         then
            merge b and b′
            extend(LT, b′, ct)
         else
           store b in the recognized scenario set
           LT ← LT +
           {triggers of tye 2 created from m and A+{a}}
```

Fig. 9. Finding all solutions of a scenario model m.

The process "*select_actor*" chooses an actor for an actor variable av and to verify the constraints involving the chosen actor.

To speed up the recognition process we order the list of constraints of the scenario model m. For that, we define an order on the constraints:

$$\text{order}(c) = \max_{i} \{av_i \text{ appears in constraint } c\}$$

for example: order((speed of $av_2 < 3$)) = 2,
order((av_1 distance $av_2 < 100$)) = 2 and
order((av_5 distance $av_1 > 150$)) = 5.

When we choose an actor for the variable av_i, we only verify the constraints with the order i. The process "select_actor" terminates when an actor is selected for the variable av or when there is no more actor to be selected.

This method enables to test all the constraints c relative to a given variable av_i:

- if order(c) < i then c is not relative to av_i.
- if order(c) = i then we check if c is verified.
- if order(c) > i then c will be checked when all variables involved in c will be instanciated.

"*verify_constraints*" is a process to verify the constraints of the scenario model m involving a given actor variable av with the actor selected a. This process first verifies the atemporal constraints relative to av and then verifies the temporal constraints. To verify temporal constraints, we propose a new method described in section 3.4.

```
verify_constraints(m, a, av)
   while atc in atemporal constraints relative to av
      if atc is not satisfied then exit(false)
   while tc in temporal constraints relative to av
      verify_temporal_constraints(
             ltv = list of temporal variables of tc,
             ld = list of domains of ltv,
             lmk = list of markers of ltv, ∅)
      if not success then exit(false)
```

Fig. 10. Verification of constraints relative to an actor variable.

Each time a scenario b is recognized, we search if a previous scenario has been recognized with the same actors. If such a scenario b' is found and if b' is consecutive to b, we extend b' with b (stop time of b' becomes the current time). Moreover we try to extend all recognized scenarios that are ended with b' up to the current time. The extension process is discussed in the next sections.

The process "store" stores a scenario in a recognized scenario list. There is one list of recognized scenarios for each model of scenario and for each list of actors. The list of recognized scenarios is ordered in time with the more recent on the top of the list.

A second way to speed up the recognition process is to organize the variable domain with the variable type to reduce the search space. There are two types of variables: the actor variables (person, equipment) and the temporal variables ("moves_close_to", "vandalism"). The domain of a temporal variable corresponds to the list of recognized scenarios for the given scenario model and the given list of actors associated to the temporal variable. This point is explained in the next sections.

3.4 Resolution of Temporal Constraints

In the previous section, we have described the algorithm to find the solutions of a scenario model. Every non-elementary scenario model contains at least one constraint "exists" to express that the scenario is constituted with sub-scenarios. The relations between sub-scenarios are mainly temporal. These temporal relations are represented by interval algebra operators [1]. We can use the algorithm described in the previous section (atemporal constraints) for the verification of constraint "exists" (temporal constraints). In this case, the procedure "Find_Solution" computes all solutions by testing all combinations of sub-scenarios. This is not interesting because (1) we are just interested in knowing whether one solution exists and (2) we compute combinations of sub-scenarios which can not be a solution (inefficient processing). More exactly, the algorithm for atemporal constraints processes every constraints in the same way, so the temporal operators are used as numerical operators. In this section we describe our algorithm to process temporal constraints taking advantage of time intervals.

In section 2, we use a constraint "exists" to indicate that a scenario is recognized depending of the existence of the list of sub-scenarios. A constraint "exists" is composed of two parts, the first one defines the m temporal variables with their domain corresponding to the m sub-scenarios. These variables are ordered in time: the oldest is the last variable and the latest is the first variable. The second part is a set of temporal constraints expressing temporal relations between the sub-scenarios. A solution of a constraint "exists" is a list of previously recognized sub-scenarios verifying the corresponding temporal constraints.

This algorithm is written briefly in Fig. 11 as a procedure *verify_temporal_constraints*. It takes as input (1) a list ltv of temporal variables (corresponding to sub-scenarios recognized on a time interval), (2) a list ld of domains of these variables, (3) a list lmk of markers on these domains and (4) the partial solution S of the constraint "exists". At the beginning of the process, ltv is initiated with the list of temporal variables of the constraint "exists" and the partial solution S is initiated to ∅. At the end of the process, ltv is the empty list of temporal variables and S contains a complete solution (a list of sub-scenarios) if a solution was found.

```
verify_temporal_constraints(ltv, ld, lmk, S)
   tv ← first element of ltv
   d  ← first element of ld
   mk ← first element of lmk
   if tv is the oldest variable of ltv
   then find_sub_scenario(tv, d, mk, oldest, α)
        if success
        then S ← S + {α}
             change makers lmk with the sub-scenarios
                                          constituting S
             exit(true)
        else exit(false)
   else find_sub_scenario(tv, d, mk, order of tv, α)
        if success
        then verify_temporal_constraints(ltv - {tv},
                       ld - {d}, lmk - {mk}, S + {α})
        else exit(false)
```

Fig. 11. Algorithm for the verification of temporal constraints defined in a constraint "exists".

The procedure "find_sub_scenario" finds a recognized scenario α corresponding to a temopral variable tv defined in domain d of sub-scenarios. A marker mk is used to limite the search domain. is_last_variable indicates if we start the search by looking at the more recent/older sub-scenario. α is a solution if it satisfies all constraints relative to the temporal variable tv. Because the variables are ordered in time, only the constraints which order (as previously defined) corresponds to the variable tv, need to be verified. This procedure is written briefly in Fig. 12.

```
find_sub_scenario(tv, d, mk, variable_order, α)
   if variable_order is the oldest
   then
      while tv in [mk, first element of d]
         verify all constraints relative to tv
         if success
         then
            α ← tv
            exit(true)
   else if variable_order is the latest
   then
      verify all constraints relative to tv with the
                                       first element of d
      if success
      then
            α ← tv
            exit(true)
   else
      while tv in [first element of d, mk]
         verify all constraints relative to tv
         if success
         then
            α ← tv
            exit(true)
   exit(false)
```

Fig. 12. Find a sub-scenario for a temporal variable of a constraint "exists".

The domain d is the list of already recognized sub-scenarios of the same type (the same scenario model and the same list of actors). Each sub-scenario is defined on a time interval and the list is temporally ordered. This order is strict in the sens that there are no overlapping sub-scenarios. The marker indicates where the previous "find_sub_scenario" process ended in the domain of sub-scenarios (i.e. which was the last sub-scenario used to recognize the given scenario model m). There is one marker for each temporal variable corresponding to one domain of sub-scenarios. This marker enables us to limit the search domain. At the end of the process "*verify_temporal_constraints*", if a solution is found, the markers of lmk are changed to the sub-scenarios used to recognize the current solution of scenario model m.

There are three ways to process a temporal variable tv depending on the order of tv in the scenario model m (corresponding to the constraint "exists"). (1) If tv is the last variable, then we select the sub-scenario recognized at the current time. If this scenario exists and verifies the temporal constraints, it can be part of the solution S. If not there is no solution for S because scenario happening before the current time have already been checked. (2) If tv is the oldest variable, then we look for the oldest solution from the maker (corresponding to the last used sub-scenario to recognize the scenario model m) up to the latest sub-scenario. So we look for the oldest solution to get the recognized scenario with the longest duration. (3) In other cases (neither the oldest nor the latest variable), we look for the latest solution from the latest sub-scenario up to the marker. We choose this order (give priority to the latest solution) to process these temporal variables, because this is statistically the most efficient order based on experiments done with various scenario models. However, there are still some models (worst cases) that require numerous combinations of sub-scenarios to find a solution.

3.5 Extension of the Recognized Scenarios

A way to speed up the recognition process is to reuse what the recognition process did in the previous times. For that, we suggest to extend a scenario recognized at the previous time to the current time when its last sub-scenario continues to be recognized at the current time.
In our representation, the duration of a scenario is a time interval. In fact, a scenario model can be recognized several times. Its number of occurrences is the duration of the recognized scenario. If a scenario s has been recognized at frame t_1 and ended by a sub-scenario s_e, and s_e is also extended at frame t_2 then we have two cases to process s at frame t_2: (1) s is automatically extended to t_2 ("stays_at", "vandalism") and (2) there are some constraints of s are not satisfied with the new state of the observed environment at frame t_2 ("moves_close_to", "moves_away_from"). The scenarios model of group (2) may contain constraints on the duration of s_e or on the existence/non-existence of several other recognized scenarios. Fig. 13 shows an observed sequence (of states) and the recognized scenarios constituting a scenario "vandalism".

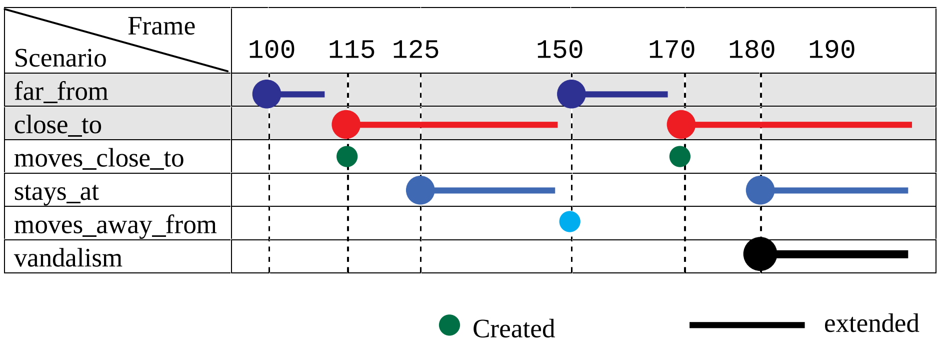

Fig. 13. An observed sequence (of states) and the recognized scenarios constituting a scenario "vandalism".

When a recognized scenario is extended, it makes an extension sequence to the recognized scenarios ended by it. The process to extend a recognized scenario to the current instant ct is shown briefly in Fig. 14.

```
extend(LT, b, ct)
   if stop time of b < ct
      if b has simple temporal constraints
      then b is automatically extended to ct
      else LT ← LT + {trigger created from b,
                    actors of b and sub-scenarios of b}
   for every s in list of scenarios ended by b
      extend(LT, s, ct)
```

Fig. 14. Extension of a recognized scenario.

For example, at the frame 190 of the sequence shown in Fig. 13, when the scenario "close_to" is extended to be ended at 190, it makes an extension sequence shown in Fig. 15. The scenario "moves_close_to" is not extended because its constraint on the duration of its sub-scenario "close_to" is not satisfied (duration of s2 <= 1).

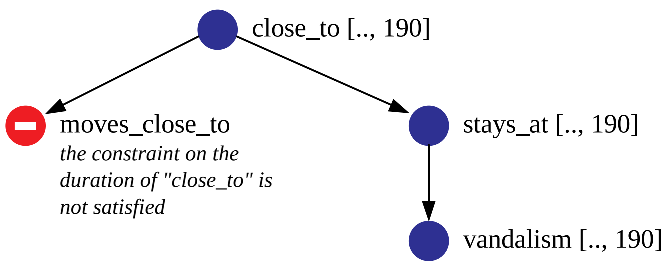

Fig. 15. Extension sequence at frame 190 of the observed sequence shown in Fig. 13.

To organize the recognized scenarios, we use an oriented graph to express the relations between the entities in the system. We will discus more detailed about this graph in the next section.

3.6 Management of the Recognized Scenarios

We organize the recognized scenarios in a graph (called graph of solutions) by its type ("close_to", "far_from",…) and by its actors (person, equipment,…). The first actor of a scenario is always a person (see 2). We organize this graph in such way that the list of actors of a recognized scenario shows the path to access to the scenario.

The graph of solutions is an oriented graph. The number of entrance nodes of this graph is the number of persons a priori in the system. There are three types of nodes in this graph, the first one is (1) the actors of recognized scenarios, the second one (2) is the scenario models predefined by expert and the last one (3) is the time interval of the recognized scenarios. The nodes successive to the persons are of type (2). The terminating nodes are of type (3). Its arcs show the path to access to recognized scenarios by the ordered list of actors.

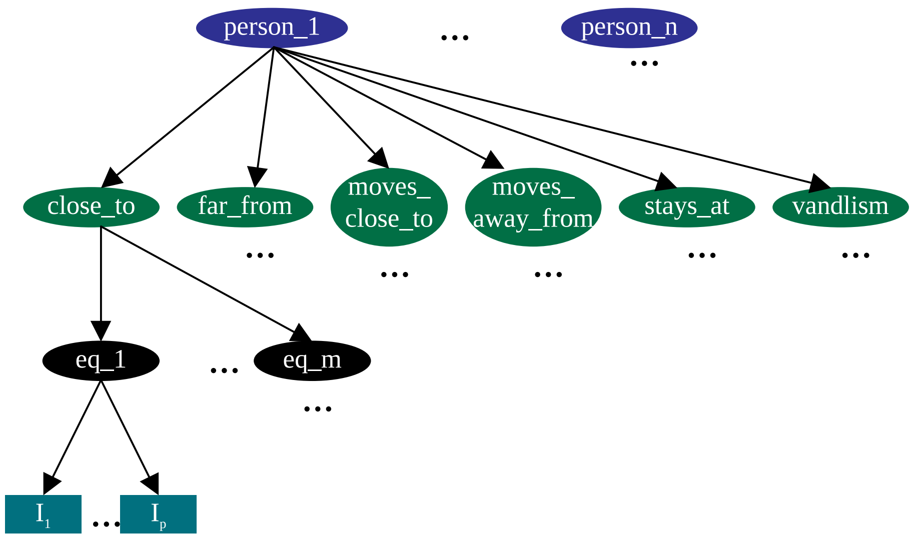

Fig. 16. Graph of solutions.

For example: the path to access to the time interval of the first instance of state "close_to" of the person_1 relative to the equipment eq_1 is person_1 $\rightarrow$ close_to $\rightarrow$ eq_1 $\rightarrow$ I_1.

4 Results

We have done two types of experiments: the first experiment consists in testing our recognition algorithm on real scenarios for a metro monitoring application. For that we have defined a description language to build seventeen models of states, events and scenarios. We have defined the model of states of the relative distance of a person to a piece of equipment (ticket vending machine), or the distance between two persons. We have defined the models of events "moves close to", "enters a zone". We have defined the model of scenarios that usually occur in metro scenes, such as "vandalism". We have correctly recognized these models on a video of 340 frames (10 frames/second) of a Nuremberg metro station. This video contains few persons and 2 ticket vending machines. The average processing time per frame is 0.1ms (millisecond) and the maximal processing time per frame is 0.2ms.

The second experiment consists in evaluating the processing time of the recognition algorithm. For that, we have defined three complex scenario models with 5, 6 and 9 actor variables in a bank branch monitoring application. We have tested these models on a video sequence of 500 frames (10 frames/second) with about 5 persons, 20 pieces of equipment (e.g. counter, chair) and 6 interesting zones (e.g. entrance to the safe). In the case of 14 previous scenario models, the average processing time per frame is

10ms and the maximal processing time per frame is 16ms. In the case of 16 complex scenario models, the average processing time per frame is 40ms and the maximal processing time per frame is 70ms. In the case of the complex scenario model with 9 actor variables, the processing time is above 1 minute and can not be used for real time application. These tests can be found in our demo page: http://www-sop.inria.fr/orion/personnel/Thinh.Vu/demos/Scenario_Recognition/index.html.

These experiments show that the proposed recognition algorithm is efficient and can be used in real world applications to recognize temporal scenarios. However, there are still some limitations when the scenario models are too complex.

5 Conclusion

In the paper, we present a scenario recognition algorithm for video interpretation which is able to process temporal constraints in an efficient way.
To represent temporal scenarios, we defined a generic model that can represent states, events and scenarios defined on time intervals. This generic model can combine various types of constraints (logic, spatial and temporal). We have also defined a description language to help experts of the application domain to describe scenario models in a declarative way.

We present a recognition algorithm which stores the scenarios recognized at previous steps in lists ordered in time and organized based on the types of scenario models and on the actors involved in the scenarios. Thanks to these lists of recognized scenarios, we obtain an efficient recognition algorithm able to process in real time (10 frames/second) complex scenarios involving up to seven actors. We have accelerated the constraint resolution algorithm by structuring the search space of the constraints using markers in the temporal variable domains. We present also a method to extend the time interval of recognized scenarios to avoid to redo the recognition process done in the past.

We have validated the scenario models and the scenario recognition algorithm on a metro and a bank application. For real world scenario models, the scenario recognition algorithm was efficient and able to recognize these models in real time. However, there are still some complex scenarios (worst cases) where the scenario recognition algorithm gets into combinatory explosion. Our future works consist in analyzing these worst cases and enhance the processing of temporal constraints in these cases.

References

[1] James F. Allen: *Towards a general theory of action and time*. Artificial Intelligence, 23:123-154, 1984.

[2] François Bremond: *Environnement de résolution de problèmes pour l'interprétation de séquences d'images*. Thèse, INRIA-Université de Nice Sophia Antipolis, 10/1997.

[3] Rina Dechter. *Temporal constraint networks*. Artificial Intelligence, 49 (1991), pp61-95, Elsevier Science Publishers B.V.

[4] Christophe Dousson: *Suivi d'évolutions et reconnaissance de chroniques*. Thèse, Université Paul Sabatier de Toulouse, 09/1994.

[5] Christophe Dousson and Malik Ghallab: *Suivi et reconnaissance de chroniques*. Revue d'intelligence artificielle, Vol.8, N°1, pp.29-61, 1994.

[6] Malik Ghallab: On Chronicles: *Representation, On-line Recognition and Learning*. 5th International Conference on Principles of Knowledge Representation and Reasoning (KR'96), Cambridge (USA), 5-8 November 1996, pp.597-606.

[7] Somboon Hongeng and Ramakant Nevatia: *Multi-Agent Event Recognition*. International Conference on Computer Vision (ICCV2001), Vancouver, B.C., Canada, 9-12/07/2001.

[8] Yuri Ivanov, Chris Stauffer and Aaron Bobick: *Video Surveillance of Interactions*. In 2nd International Workshop on Visual Surveillance, pp82-89, Fort Collins, Colorado, 06/1999.

[9] Tony Jebara and Alex Pentland: *On Reversing Jensen's Inequality*. In Neural Information Processing Systems 13, NIPS 13, 12/2000.

[10] Lina Khatib, Paul Morris, Robert A. Morris and Francesca Rossi: *Temporal Constraint Reasoning With Preferences*. IJCAI 2001, pp322-327.

[11] Roger Mohr and Thomas C. Henderson: *Arc and Path Consistency Revisited*. Research Note, Artificial Intelligence, pp225-233, vol28, 1986.

[12] Claudio Pinhanez and Aaron Bobick: *Human Action Detection Using PNF Propagation of Temporal Constraints*. M.T.T Media Laboratory Perceptual Section Technical Report No. 423, 04/1997.

[13] Nathanael Rota: *Contribution à la reconnaissance de comportements humains à partir de séquences vidéos*. Thèse, INRIA-Université de Nice Sophia Antipolis, 10/2001.

[14] Nathanael Rota and Monique Thonnat: *Activity Recognition from Video Sequences using Declarative Models*. 14th European Conference on Artificial Intelligence (ECAI 2000), Berlin, Proceeding ECAI'00 – W. Horn (ed.) IOS Press, Amsterdam, 20-25/08/2000.

[15] Nathanael Rota and Monique Thonnat: *Video Sequence Interpretation for Visual Surveillance*. 3rd IEEE International Workshop on Visual Surveillance, VS'00, pp 59-67, Dublin, Ireland Proceeding IEEE, 07/2000.

[16] Eugenia Ternovskaia: *Automata Theory for Reasoning about Actions*. The 16th International Joint Conference on Artificial Intelligence (IJCAI-99), Stockholm, Sweden, 31/07-06/08/1999.

[17] Catherine Tessier: *Reconnaissance de scènes dynamiques à partir de données issues de capteurs: le projet PERCEPTION*. Rapport technique, Onera-Cert, 2 avenue Edouard-Belin, BP4025 31055 Toulouse Cedex France, 08/1997.

[18] Monique Thonnat and Nathanael Rota: *Image understanding for visual surveillance application*. Third international workshop on cooperative distributed vision CDV-WS'99, pp51-82, Kyoto, Japan, 19-20/11/1999