ECAI 2016
G.A. Kaminka et al. (Eds.)

doi:10.3233/978-1-61499-672-9-234

A Probabilistic Logic Programming Approach to Automatic Video Montage

Bram Aerts and **Toon Goedemé** and **Joost Vennekens** [1]

Abstract. Hiring a professional camera crew to cover an event such as a lecture, sports game or musical performance may be prohibitively expensive. The CAMETRON project aims at drastically reducing this cost by developing an (almost) fully automated system that can produce video recordings of such events with a quality similar to that of a professional crew. This system consists of different components, including intelligent Pan-Tilt-Zoom cameras and UAVs that act as "virtual camera men". To combine the footage of these different cameras into a single coherent and pleasant-to-watch video, a "virtual editor" is needed. This paper describes the development of such a component. We adopt a declarative approach, in which we build a model of the decision process that a human editor might follow to edit a video. To achieve a montage that obeys various cinematographic rules while at the same time retaining a natural, non-mechanical feel, we construct this model in a Probabilistic Logic Programming language. We demonstrate that the resulting system can be run in real-time and that it delivers montages that are almost indistinguishable from those made by a professional editor.

1 INTRODUCTION

Events such as lectures, sports games, musical performance, etc. are often recorded on video. When produced by a professional video crew, these recordings are typically of high quality, but also quite expensive. Because good-quality cameras are no longer overly expensive, organizers may be tempted to instead produce a "home-made" recording of their event, filmed by amateurs. However, while this option is much cheaper, the end results tend to be noticeably lower quality than those produced by a professional crew.

The research described in this paper fits within the CAMETRON project. The goal of this project is to fill the gap between videos produced by a professional crew and home-made productions, by creating a system to produce recordings whose quality approaches that of professionally produced video, but without the price tag. It will rely on an almost completely automatic system, thereby eliminating the expensive personnel cost. Its focus is on producing full-length reports of events, i.e., our goal is to produce videos that span the entire lengths of the event itself, rather that a summary or a set of highlights.

The CAMETRON system will consist of two separate components, mimicking the roles found in a typical film crew: virtual "camera men" (consisting of fixed cameras, pan-tilt-zoom cameras and cameras mounted on UAVs, together with the software to operate them) will capture the footage, while a virtual "editor" combines footage of the different cameras. The ongoing development of the first component has been described elsewhere [11, 5]. In this paper, we present the second component: a virtual editor system, that is able to produce a single coherent, qualitative video from a number of different feeds of raw footage from the same event.

Creating a video that is both interesting and easy-to-follow is not a straightforward task. Human editors typically follow a number of different—and sometimes contradictory—cinematographic "rules" to accomplish this task. To develop our virtual editor, we will follow a declarative approach, in which we explicitly represent these rules. This approach has the benefit that it offers a great deal of flexibility in deciding which rules should be taken into account and how they should take priority over each other. An additional advantage is that it also allows us to reuse the same knowledge to perform different tasks: we cannot only use the rules to generate a montage, but also to evaluate the quality of a given montage or to learn certain properties of good montages from given examples.

To represent the rules, we need a suitable knowledge representation language. A particular challenge in this application is that cinematographic rules are not strict: they are guidelines that are typically followed, but not always. Indeed, the rules may sometimes contradict each other, and even if they do not, a human editor may still choose to ignore a rule, simply because the result "feels" better. A virtual editor should therefore not rigidly follow the rules, but it should sometimes deviate from them in order to give the montage a more interesting and natural flavour, thereby mimicking the creativity of a human editor. For this reason, we have chosen to make use of a Probabilistic Logic Programming (PLP) language, which allows us to represent these rules in a non-deterministic way. This has the additional benefit that—just like a human editor—the system is able to produce different montages from the same input streams.

In this paper, we restrict attention to videos of relatively simple settings, like a lecture, a sports event or a concert. More challenging settings, such as fiction, are left for future work. Our work differs from existing approaches in various ways. First, unlike techniques that construct summaries of an event's highlights [9, 7], our method produces a video of the complete event. Second, unlike approaches that are developed for a specific setting [13, 18, 8], we do not assume a fixed number of cameras, a fixed camera setup or a particular subject matter. Third, our system assumes as input only video footage, in contrast to, e.g., the virtual editor systems that are present in game design and which use a 3D model of the scene [3, 10]. Fourth, our system is able to make the required editing decisions in real-time (for 25fps video), unlike, e.g., [4, 12]. Fifth, we incorporate variety of different cinematographic rules into our system and demonstrate that the resulting montages are almost indistinguishable from those produced by a professional editor. Existing work either does not perform such validation [12, 18, 8] or appears to have worse results than our method [4, 13]. Finally, systems such as [4, 12] are determinis-

[1] KU Leuven, Technology Campus De Nayer, Research Group EAVISE, Belgium, email: {b.aerts, toon.goedeme, joost.vennekens}@kuleuven.be

tic, in the sense that they always produce the same montage for the same video stream. The fact that our approach is more flexible in this respect may prove useful if the user for some reason does not like the montage initially proposed by the system.

The remainder of this paper is structured as follows. We describe Problog, the PLP system that we will use, in Section 2. In Section 3, we discuss a number of cinematographic rules. Section 4 describes our editing system in detail, in Section 5 we describe how this system can be used. Section 6 discusses the computational performance of this system, while Section 7 investigates the quality of its output. In Section 8, related work is discussed in more detail. We conclude in Section 9.

2 PRELIMINARIES: CP-LOGIC AND THE PROBLOG SYSTEM

CP-logic [20] is an expressive PLP language, based on Sato's distribution semantics [19]. A theory in CP-logic consists of a set of rules of the following form:

$$\alpha_0 :: A_0 \quad ; \quad \cdots \quad ; \quad \alpha_n :: A_n \quad :- \quad \phi. \tag{1}$$

Here, ϕ is a formula (typically, a conjunction of literals), the A_i are atoms and the α_i are probabilities with $\sum_i \alpha_i \leq 1$. Each such rule represents a non-deterministic causal mechanism: the body ϕ triggers a non-deterministic event which causes at most one of the A_i; for each i, α_i is the probability that A_i is the outcome of this event.

We first consider only the ground case, in which no variables are allowed. In this case, the formal semantics of CP-logic can be characterized in terms of the well-founded semantics for normal logic programs as follows. Suppose that, for a rule of form (1), we probabilistically either replace this rule by one of the rules $A_i :- \phi$, each with probability α_i, or we remove this rule altogether with probability $1 - \sum_i \alpha_i$. If we do this independently for each of the rules r in a CP-logic theory T, then we probabilistically reduce T to a normal logic program P. By π_T, we denote the resulting probability distribution over these normal logic programs P. The formal semantics of the CP-logic theory is then the probability distribution P_T over possible worlds (i.e., Herbrand interpretations) that is defined by $\pi^*_T(I) = \sum_{P \models I} \pi_T(P)$; here, the sum is taken over all normal logic programs P that have the interpretation I as their well-founded model. These semantics are only well-defined for theories T that have the property that all logic programs P that can be produced from them (with non-zero probability) have a two-valued well-founded model (which is then also the unique stable model of P). CP-logic disallows theories that do not have this property.

A small example is the following CP-logic theory T_{SB}. It describes two children, Billy and Suzy, who each may decide to throw a rock at a bottle. Both children throw with 75% accuracy.

```
0.5::throws(billy).
0.6::throws(suzy).
0.75::breaks :- throws(billy).
0.75::breaks :- throws(suzy).
```

According to this theory, for instance $\pi^*_{T_{SB}}(breaks) = 0.5 * 0.6 * 0.75 + 0.5 * 0.4 * 0.75 + 0.5 * 0.6 * (1 - (1 - 0.75)^2) = 0.65625$.

This semantics is extended to the non-ground case by viewing a non-ground rule as a template for the set of all of its ground instantiations. This approach is identical to how variables are treated in Answer Set Programming [14, 15]. It requires all rules to be finitely groundable. This can be ensured by disallowing function symbols and requiring that all variables must appear in a positive body atom.

The above example is therefore equivalent to:

```
0.5::throws(billy).
0.6::throws(suzy).
0.75::breaks :- throws(X).
```

The Problog system [16] is a state-of-the-art inference system for CP-logic. It supports such inference tasks as querying the probability $\pi^*_T(A)$ of an atom A and sampling a particular interpretation I according to this distribution π^*_T. Its input language extends CP-logic with a number of features that make it easier to write Prolog-style programs. In particular, it allows lists, the use of predicates such as `findall` and the use of variables as probability labels in the head. In the remainder of this paper, we will use this input language.

The small example given above can also be tried out in the online Problog inference engine: `http://tinyurl.com/jqdtrlm`

3 CINEMATOGRAPHIC MODEL

Whether they are movies, documentaries, lecture recordings, or any other form of edited video, all compositions tend to follow some generally accepted cinematographic rules. These rules exist to avoid confusing the viewer. While more experimental movies may deliberately violate these rules to shock or confuse the viewer, they are typically obeyed in the kind of objective event reports that form the focus of this paper. Indeed, following these rules leads to informative, pleasant-to-watch reports, that do not confuse or distract the viewer.

Cinematographic rules can be divided in two main groups. The first group are rules that need to be taken into account while filming. These rules concern concepts such as depth of field, rule of thirds, headroom and the 180 degrees rule as described in e.g. [11]. These rules are outside the scope of this paper, because they need to be taken into account by the virtual "camera men", not the virtual editor. We therefore assume that the raw video feeds satisfy these rules.

The second group of rules concern the ways in which multiple video streams should be put together. These are the focus of our editing system. Before discussing these rules, we define some terminology. A *shot* is a sequence of successive frames from the same camera. A *cut* is a transition from one shot to the next. A *montage* is a sequence of shots. Figure 1 shows a montage composed of shots taken from different video streams. Note also that, at any particular point in time, some of the cameras may not be producing footage.

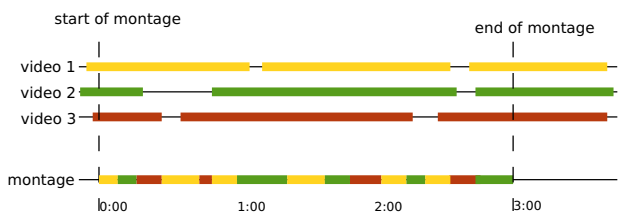

Figure 1. Making a montage out of video streams.

Length of shots. A shot should neither be too long nor too short. If it is too long, the viewer will get bored and his attention will drift. On the other hand, when shots are too short, the video becomes too jumpy and the viewer is presented with too much information in too little time. In general, shot lengths between 7 and 90 seconds are considered acceptable. However, the length of a shot also depends

on the amount of action in the scene: in energetic scenes, shorter shots can accentuate the ongoing action, while longer shots are more suitable for static scenes. For instance, a lecture recording will tend to have longer shots than a recording of a basketball game.

In order to prevent the montage from appearing too mechanical or predictable, there should also be some variation in the shot length. Instead of striving towards some "ideal" shot length, we should therefore make use of different shot lengths, within the bounds of what is neither too long nor too short.

Order of shots. Certain transitions between shots work well, while others should be avoided. When two consecutive shots mismatch, this is referred to as a *jump cut*. Jump cuts can occur when there is either too much or too little difference between the shots. When the footage of two different cameras is almost, but not completely the same, a "jumpy" effect occurs when switching from one to the other. As a rule, there should be a minimum angle of 30 degrees between the view points of two different cameras in order to avoid these kinds of jump cuts.

On the other hand, when the footage of two different cameras is completely different, the viewer will feel lost when switching from one to the other, because he has no clue about the context of the new subject. A quite common order of shots is therefore to start with a *long shot*, that establishes an overview of the scene. The next shot is then typically a *medium shot*, that shows a closer picture of, e.g., a particular person in the scene. Then, a *close-up shot* of the person's face could follow. The close-up can then be followed up with another overview or medium shot.

Although this order of shots is often pleasing, it is not necessarily always followed. Indeed, as with the shot length, there should also be some variation in the sequence of different kinds of shots in order to avoid montages that appear too mechanical.

Continuity. To ensure coherence between shots, continuity should be respected. Cutting between disparate locations should be avoided, because it violates spatial continuity.When multiple cameras are filming one scene, all actions taking place in this scene should appear fluid and continuous. In other words, when switching between cameras, footage before and after the switch should be contiguous. This principle is called temporal continuity.

In this paper, our goal is to record footage of an event that takes place in a single location. We can therefore assume that spatial continuity is satisfied by our camera setup. In addition, our goal is also to cover the full length of the event (as opposed to summarizing the highlights). Therefore, temporal continuity will be satisfied because subsequent shots in the video will correspond to subsequent events in real life.

Action and reaction. Video coverage of an event should capture all the relevant action. When filming an action, three separate phases are important: premeditation, the action itself and the reaction. First, whenever an action is about to take place, there is a brief moment in advance when the person is thinking about undertaking this action. To prepare the viewer for the action that is about to take place, showing this premeditation is important. Second, showing the action itself is of course the most important thing. Third, when the action is complete, the viewer expects to see the reaction of a person to it.

4 OVERVIEW OF THE EDITING SYSTEM

An overview of our editing system is shown in Figure 2. It takes as input a number of different video streams, together with an analysis of each of these streams. This analysis is performed by computer vision algorithms, which have been described elsewhere [11, 1, 2, 6]. For each frame in each stream, we expect these algorithms to provide the following information:

- Whether people are present in the shot;
- Which kind of shot (long shot, medium shot, ...) it is;
- Which person is the most prominent subject of the shot;
- Which action (walking, talking, ...) the main subject performs.

The goal of our editing system is to decide for each point in time which of the available camera feeds will be used. The output of the system is the single video stream that is thus constructed.

The Editing system we propose consists of 3 components. The first component is the preprocessor, which comes in after the computer vision. This is a program written in python, which synchronizes camera footage, cleans up noisy detection data and parses this data to a format readable by problog. The problog core works out a montage, taking into account the cinematographic rules described in 3. After that, another python program edits the original video streams into one final montage, in the order the problog core calculated.

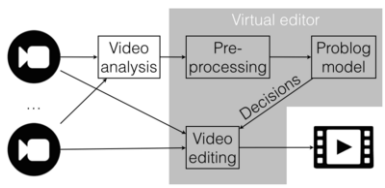

Figure 2. Overview of the editing system.

4.1 Data Representation

We assume that all cameras are synchronized with respect to a global time line. This time line is divided into discrete frames. For each frame, we need to represent which kind of footage each of the available cameras is providing for that particular frame.

We represent camera sources as $camera(ID)$. When a new camera connects, or any active camera disconnects, it is possible to add or remove the corresponding ID. A connected camera does not necessarily produce useful footage at each point in time, e.g., a UAV does not produce footage while at its loading station. Valid video footage is represented as $frame(Frame, CameraID)$. Even when a camera is providing footage, this may not be usable if the camera is being adjusted. We denote this as $adjust(Frame, CameraID, Type)$ where $Type$ is either $zoom$ or $focus$.

Valid video footage may contain people or not. The video analysis system assigns each person—or rather, group of pixels that might be a person—a detection score that indicates how likely this is to be an actual person. We assume that the detection with the highest score corresponds to the most prominent person in the frame. We identify this person by means of a $person(Frame, CameraID, PersonID)$ fact. When a person is detected, we make a distinction between the different shot types: long shot, medium shot or close-up. We represent this as $shot\_type(Frame, CameraID, Type)$. In addition, the person may perform a specific action: stand still, walk, point, or talk. We

represent this as $action(Frame, CameraID, Action)$. For images in which no person is detected, we make a distinction between $overview$ shots and $noperson$ shots. An $overview$ shot is one in which people are too small to be detected by the video analysis, whereas a $noperson$ shot is one that actually does not contain any people. The video analysis may use knowledge of the position of the different cameras in the scene to try to distinguish between the two.

4.2 Cinematographic model

The editor system is based on a Problog model of the decision process that a human editor might follow to produce a live montage. At each point in time (i.e., for each frame), the editor needs to decide which camera to use, based on the footage that he has previously used and the footage is currently available from the different cameras (and possibly also a small lookahead at future footage to avoid, e.g., switching to a camera that is about to stop producing footage).

For every time point T, the program decides which camera C to use at that time. This decision is recorded in a predicate $use\_camera(T, C, Torig)$. The third argument records the starting time point $Torig$ of the shot that is ongoing at time T. This is merely for efficiency reasons, since we could also recompute $Torig$ by looking at the preceding $use\_camera(T', C, \_)$ atoms with $T' < T$.

We define $use\_camera(T, C, Torig)$ by means of the following two rules. The value of this predicate depends on two decisions that the editor must make: first, whether to cut away from the ongoing shot and, if so, which other camera to switch to. The first decision is recorded by the predicate $change(T)$ and we will discuss below how it is made. The effect of not changing is of course simply that the ongoing shot continues.

```
use_camera(T,C,Torig):-
    previous_camera(T,C,Torig),
    not(change(T)).
previous_camera(T,C,Torig):-
    time(T),Tp is T-1,use_camera(Tp,C,Torig).
```

If the editor does decide to change at time T, it will pick at random one of the other cameras C that are good candidates to switch to at that particular time (as given by $change\_candidate(T, C)$).

```
use_camera(T, C, T):-
    time(T),change(T),
    findall(Cam,change_candidate(T,Cam),Cams),
    uniform(Cams,C,T).
```

The predicate $uniform(Cams, C, T)$ expresses that C is randomly selected, according to a uniform distribution, from the list $Cams$. The third argument T is present to ensure that, at different time points, a different camera may be selected from the same list. The definition of $uniform$ is as follows. In order to make a uniform selection from a list $[H \mid T]$ of length N, this predicate either selects the head H with probability $\frac{1}{N}$, or it performs a uniform selection from the list T of length $N-1$ with the remaining probability $1-\frac{1}{N}$.

```
uniform(List,El,ID):-
    length(List,L),uniform(List,L,El,ID).
uniform([H|T],N,H,ID):-
    P is 1/N,s(P,[H|T],ID).
uniform([H|T],N,E,ID):-
    P is 1/N,not(s(P,[H|T],ID)),
    NN is N-1,uniform(T,NN,E,ID).
P::s(P,_,_).
```

The predicate $change(T)$ contains the essence of our cinematographic model. In general, there are two reasons to change: we can either do so because we want to, or because we have to. The latter case occurs when the current camera no longer provides usable footage, either because it no longer produces any footage at all, or because it starts to zoom or refocus.

```
change(T):-not(can_stay(T)).
can_stay(T):-
    previous_camera(T,C,_),
    frame_ok(T,C).
frame_ok(T,C):-frame(T,C),not(adjust(T,C,_)).
```

In addition to changing because we have to, we might also switch cameras because we want to. In general, this will occur if there is at least one camera whose footage we prefer over that of the current shot and if we have already met the minimal shot length requirement.

```
change(T):-
    change_candidate(T,_),not(too_short(T)).
```

In order to decide whether the current shot is long enough, we of course need to know the current shot length. This is easily computed by means of the third argument of the use_cam predicate:

```
shot_length_until(T, Len):-
    previous_camera(T,C,Torig),Len is T-Torig.
```

The minimal shot length is not a fixed number. Some shots are definitely long enough and some shots are definitely too short, but there is also a gray area, in which an editor might make a judgment call to cut away slightly sooner than he would like in order to, e.g., better capture the beginning of an action. For this reason, we define too_short in a probabilistic way, as shown in Figure 3.

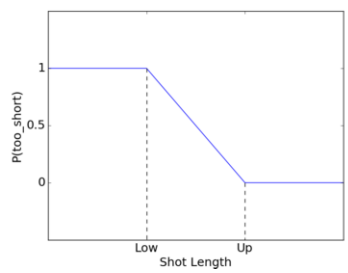

Figure 3. The probability of a shot being considered too short.

Here, Up is an upper bound on the minimal shot length (i.e., longer than Up is never too short), while Low is a lower bound (i.e., shorter than Low is always too short).

```
P::too_short(T):-
    shot_length_until(T,Len),
    min_length_ub(Up),min_length_lb(Low),
    Len=<Up,P is min(1,(Up-Len)/(Up-Low)).
```

We similarly define a predicate too_long in terms of max_length_lb and max_length_ub. The effect of the current shot becoming too long is that this gives a reason to switch to a different camera:

```
P::too_long(T):-
    shot_length_until(T,Len),
    max_length_ub(Up),max_length_lb(Low),
    Len>=Low,P is min(1,(Len-Low)/(Up-Low)).
should_switch(T):-too_long(T).
```

We can now determine whether there exist cameras that it would be appropriate to switch to. Recall that if we find at least one such $change\_candidate$, we will terminate the current shot unless it is still too_short. To make this decision, we divide the other cameras into three different categories: poor, fair and good change candidates. A good candidate is one that we always want to switch to, i.e., the existence of such a camera is in itself a reason to change. Poor and fair candidates are those that we might switch to if the current shot is getting too long and there is no better alternative available, i.e., we only consider switching to a fair candidate if there are no good candidates and we only consider switching to a poor candidate if there are neither fair nor good candidates.

```
change_candidate(T,C):-
    change_candidate(T,C,good).
change_candidate(T, C):-
    change_candidate(T,C,fair),
    should_switch(T),
    not(change_candidate(T,_,good)).
change_candidate(T,C):-
    change_candidate(T,C,poor),
    should_switch(T),
    not(change_candidate(T,_,fair)),
    not(change_candidate(T,_,good)).
```

A poor candidate is any camera to which it is possible to change to. If, in addition, the candidate camera provides a good transition from the camera that is currently in use, it is considered a fair candidate. Finally, a good candidate is a fair candidate that also has more interesting footage than the current camera.

```
change_candidate(T,C,good):-
    change_candidate(T,C,fair),
    better_frame(T,C).
change_candidate(T,C,fair):-
    change_candidate(T,C,poor),
    good_transition(T,C).
change_candidate(T,C,poor):-can_change(T,C).
```

The predicate can_change is similar to the predicate can_stay that was defined above, but it is more stringent. In addition to demanding that the camera is currently providing a usable frame ($frame\_ok(T, C)$), can_change also demands that the camera will keep on providing usable frames in the near future. This is to prevent us from falling below the minimal shot length by cutting to a camera that is about to stop filming.

```
can_change(T,C1):-
    previous_camera(T,C,_),camera(C1),C\=C1,
    future_ok(T,C1).
future_ok(T,C):-
    camera(C),min_length_lb(M),T2 is T+M,
    not(between(T,T2,T1),not(frame_ok(T1,C))).
```

As discussed above, the difference between a poor candidate and a fair one lies in the quality of the transition that can be achieved by switching from the current camera to the candidate.

```
good_transition(T,C):-
    previous_camera(T,Cp,_),Tp is T-1,
    shot_type(Tp,Cp,STp),shot_type(T,C,STcur),
    shot_transition(T,STp,STcur).
```

Here, $shot\_transition(T, STp, STcur)$ represents the fact that, at time T, it is a good idea to switch from shot type STp to shot type $STcur$. As mentioned before, we do not want to impose a strict order in which the different shot types are used. For this reason, $shot\_transition$ will be a probabilistic predicate. It will allow all possible transitions in principle, but assign a higher probability to those transitions that are generally considered better.

```
P::shot_transition(T,From,To):-
    time(T),qualtity(From,To,P).
quality(ls,ls,0.8).
quality(ls,ms,1).
...
```

The first argument of the $shot\_transition$ predicate is needed to allow different choices to be made at different time points, i.e., it should be possible that a transition from ls to ms is considered a good idea at time point t but not at $t' \neq t$. The $quality$ predicate defines the probability that each kind of transition is considered a good idea, as shown in Table 1.

Table 1. The probabilities of different shot transitions.

	To				
From	ls	ms	cu	os	np
long **s**hot	0.8	1	0.2	0.8	0.1
medium **s**hot	1	1	0.6	0.8	0.1
close **u**p	0.8	0.7	0.2	0.8	0.1
overview **s**hot	0.8	0.7	0.1	0.8	0.1
no **p**erson	1	1	0.2	1	0.1

The difference between a fair candidate and a good candidate is that the latter not only offers a good transition but also provides more interesting footage. This will mainly be determined by the actions that are taking place in the footage. Again, in order to avoid giving the montage a mechanical feel, we do not place a strict priority on the different kinds of actions that might occur. Instead, we assign the different kinds of actions a probability that they will be selected in the video. Selecting an action can only happen at the time point when it starts. The video analysis module detects the actions of talking, walking and pointing. We also introduce a special action, called *emerging*, that we assign to a person who newly appears in the footage.

```
0.55::select_action(Taction,C,emerge):-
    start_person(Taction,C).
0.80::select_action(Taction,C,talk):-
     start_action(Taction,C,talk).
0.55::select_action(Taction,C,walk):-
    start_action(Taction,C,walk).
0.65::select_action(Taction,C,point):-
    start_action(Taction,C,point).
start_action(T,C,AT):-
    action(T,C,AT),Tprev is T-1,
    not(action(Tprev,C,AT)).
start_person(T,C):-
    person(T,C,ID),Tprev is T-1,
    not(person(Tprev,C,ID)).
```

As explained before, it is important to not just show the action itself, but also the "premeditation" leading up to it. The length of this lead-in may depend on the action in question. Moreover, as with shot lengths, there is a window of acceptable options, rather than a single fixed value. We define these windows for the different actions as follows:

```
pre_action_window(emerge,0, 1).
pre_action_window(talk,  2, 10).
pre_action_window(walk,  0, 2).
pre_action_window(point, 0, 5).
```

If an action AT starts at time T_{action} in the footage of camera C, then we can cut to camera C at any time point in the interval $[T_{action} - Max, T_{action} - Min]$, where $[Min, Max]$ is the pre-action window of action AT, as defined by the above predicate $pre\_action\_window(AT, Min, Max)$. For instance, if a $talk$ action is initiated at time point 20, we can cut at any time point in the interval [10,18]. However, in doing so, we should take care that whenever we decide to switch at time T_{switch}, the camera actually produces a stream of valid frames between T_{switch} and T_{action}. For instance, if the camera produces no frames between time point 14 and 16, we should not yet switch to it at time point 12.

Starting from time point T, the following predicate gathers into a list all the time points $T' \leq T$ such that camera C has produced only valid frames between T' and T. We do not go back arbitrarily far in the history, but only look Len frames back.

```
valid_since(T,C,Len,[]):-not(frame_ok(T,C)).
valid_since(T,C,-1, []).
valid_since(T,C,Len,[T|Tail]):-
    Len>=0,frame_ok(T,C),
    Tp is T-1,Newlen is Len-1,
    valid_since(Tp,C,Newlen,Tail).
```

We can now use this predicate to gather into a $List$ all time points at which we could possibly switch to a camera C that shows action AT starting at T_{action}.

```
valid_action_window(Taction,C,List,AT):-
    pre_action_window(AT,Min,Max),
    T is Taction-Min,Delta is Max-Min,
    valid_since(T,C,Delta,List).
```

If we now switch to camera C at any time point $T_{switch} \in List$, we will have good footage from T_{switch} up to the time point T_{action} when action AT starts. In addition to this, we also want to avoid showing an action that does not last long enough for the viewer to make sense of it. We therefore also require that the person doing the action remains visible for a minimal amount of time after the start of the action.

```
action_visible(Taction,C,AT):-
    minimal_length(M),Tmin is Taction+M-1,
    not(between(Taction,Tmin,T),
        not(frame_ok(T,C),person(T,C,PID))).
```

If the starting action that we have selected (with $select\_action$) satisfies both of these conditions (i.e., before and after the starting time of the action there is a sufficient amount of valid footage), then we can select one of its possible switching times (as given by the $valid\_action\_window$ predicate) as a time to cut to the camera in question.

```
better_frame(T,C):-
    select_action(TAct,C,Act),
    action_visible(TAct,C,Act),
    valid_action_window(TAct,C,List,Act),
    uniform(List,T,TAct).
```

In addition to switching to a different camera because it is starting an interesting new action, we might also switch because the footage of the current camera has become less interesting than the footage of another. We consider footage showing a person to be typically more interesting than footage not showing a person, and footage in which someone is talking more interesting than other footage.

```
0.55::better_frame(T,C):-
    previous_camera(T,Cp,_),
    not(any_person(T,Cp)),any_person(T,C,_).
any_person(T,C):- person(T,C,_).
0.60::better_frame(T,C):-
    previous_camera(T,Cp,_),
    not(action(T,Cp,talk)),action(T,C,talk).
```

This concludes the usual decision process of the editor. A special case that we have not yet discussed is the very first time point of the montage. Here, we prefer an overview shot. Failing that, we might choose a long shot, medium shot, close-up, or, as a final resort, a no-person shot. We record this preference in a list and gather the list of all possible candidates, taking these preferences into account. Then we select randomly one of the candidates.

```
initial_priority([os, ls, ms, cu, np]).
initial_candidate(T,C):-
    initial_priority(List),T2 is T+1,
    initial_candidate(T2, C, List).
initial_candidate(T, C, [First|Tail]):-
    shot_type(T,C,First),future_ok(T,C).
initial_candidate(T,C,[First,Second|Tail]):-
    not(shot_type(T,C,First),future_ok(T, C)),
    initial_candidate(T,C,[Second|Tail]).
use_camera(T,C,T):-initial_time_point(T),
    findall(Cam,initial_candidate(T,Cam),Cams),
    uniform(Cams,C,T).
```

5 USING THE CINEMATOGRAPHIC MODEL

The Problog model of the previous section describes the decision process that an editor might follow in order to produce a montage in accordance with cinematographic rules. This model defines a probability distribution over all possible ways in which a number of given input streams can be edited into a single video. It can be used to perform different tasks.

By *querying* the probability of a given montage according to this distribution, we can obtain an estimate of the quality of this montage: highly probable montages satisfy many of the rules, whereas unlikely montages contain many "violations". *Sampling* from this distribution will produce a single montage. This is of course the task that we focus on in this paper. Because the probability distribution defined by the Problog program assigns a higher probability to montages that respect more of the cinematographic rules, sampling is more likely to produce a good montage than a poor one. If we compute different samples for the same input, we will obtain a different montage each time. We view this as a desirable property, because it corresponds to how human editors perform their task: it is unlikely that a human editor would produce precisely the same output each time, if he were asked to edit together the same input streams multiple times. Capturing this variance in our virtual editor helps to ensure that our montages have a natural feel.

An alternative to sampling is to compute the *most likely* montage, given a set of input streams. However, such an approach would have significant disadvantages when compared to the sampling approach:

- The task of computing the most likely outcome is computationally significantly harder than that of sampling from a distribution.
- To compute the most likely montage, we need to consider the entire time line at once. Such an approach would therefore only be usable in an offline editing system. By contrast, as we will discuss in Section 6, sampling can be used to implement a system that edits input streams in (slightly delayed) real-time.
- Under the same circumstances, the most likely montage will always make the same choices (namely, it will choose the most likely alternative). By contrast, sampling will occasionally make a less likely choice, thereby producing a less mechanical and more interesting montage.

However, a disadvantage of the greater variation allowed by the sampling approach is that it may occasionally produce a poor montage. To compensate for this, our editing system will construct a set of samples. From this set, the most likely sample will be returned as the output of our system. By making this set larger, we increase the computational cost of our method, but reduce the risk of returning a poor montage. Our current implementation uses 10 samples.

In addition to *querying* and *sampling*, the Problog system also contains algorithms that perform the task of *parameter learning*. Currently, the probabilistic parameters of our Problog model have been manually filled in, based on our understanding of cinematographic rules. An alternative approach would be to derive these parameters automatically from a number of examples videos. In this case, we might also be able to train our model to emulate particular editing styles. We will investigate this topic in future work.

6 ACHIEVING REAL-TIME PERFORMANCE

In order to create a montage, our editing system calls on Problog to compute a number of different samples according to the probability distribution defined by the cinematographic model. There are two main parameters that affect the computational performance of the sampling algorithm: the number of time points and the number of cameras. Indeed, for each time point T and camera C, the sampling procedure must decide whether to switch to C at time T. We expect that the number of cameras will not vary greatly and will be dictated to a large extent by the setting in which the video must be produced. By contrast, the number of time points is a more flexible parameter. Indeed, the granularity of the time line used by our Problog model need not coincide with that of the actual input video. In other words, it is not necessary that each frame of the incoming video streams corresponds to an individual time point in the Problog model.

We will group together a number of actual frames into a sequence that we call a *pframe*. These pframes will act as time points for the Problog model. The number of real frames that go into one pframe is therefore a key parameter of our system. On the one hand, this number affects the performance of the sampling algorithm: the higher it is, the faster execution will be. On the other hand, this number also affects the quality of the montage that is produced, because our system will only be able to switch cameras at the start of a new pframe. Therefore, the smaller the pframes are, the more fine-grained this decision process will be and the higher the quality of the output.

Figure 4 shows the execution time needed to compute 10 samples in function of the length of a single pframe. The y-axis expresses the execution time as a fraction of the length of the video stream that needs to be edited. If this value is ≤ 1, we have a system that is able to make the required editing decisions in real-time. The figure shows a graph for both a 3-camera and a 6-camera setup. In both cases, taking a pframe to be equal to a single real frame ($= 0.04s$, since the video was shot at $25fps$) produces a runtime that is much too slow. However, as the length of a pframe increases, the runtime drastically decreases. For the 3-camera setup, we reach real-time performance as soon as at least 6 frames are combined in a single pframe. With 6 cameras, we need 10 frames per pframe. Because it is unlikely that a viewer would be able to tell the difference between cutting $0.4s$ earlier or later in a montage, we consider this an acceptable way of reaching real-time performance.

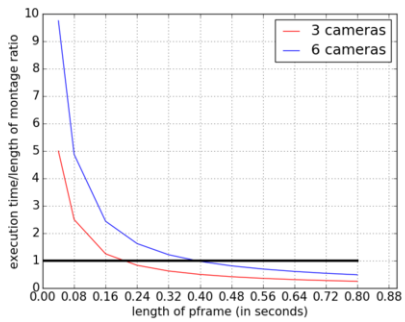

Figure 4. Execution time vs. length of pframes (in seconds)

7 EXPERIMENTAL RESULTS

Our system is able to produce real-time edits of different video streams. The only question remaining is whether the resulting montages are of a quality similar to those produced by professional editors. Since there is no single right way to edit video, we have no "ground truth" to compare the output of our system to. Instead, we have subjected the virtual editor to a "Turing test": we have asked a number of test subjects to distinguish between the output of our system and a professionally made montage of the same video streams.

The test case used in this experiment is a lecture recording made by three cameras. This footage was edited by a professional editor, who was present during the recording. From the entire video stream, we selected a fragment of three minutes in which there was a lot of "action", namely one speaker introducing another speaker, who then came to the stage. We presented 58 students with two video clips: this particular fragment as edited by the professional and the same fragment edited from the same input streams by our system. The student were then asked to identify the clip produced by the professional. As an incentive, a small prize (worth around 75€) was given to a random student among those who guessed correctly.

The two video clips can be viewed online:

1. https://www.youtube.com/watch?v=7vrfhzD4G0c
2. https://www.youtube.com/watch?v=Bzl10YKGeI4

In case the reader would like to perform this experiment himself: the professionally edited video is Montage i, where i is the 22^{nd} digit in the decimal expansion of π.

Of the 58 students, 31 ($= 53\%$) correctly identified the professionally edited clip. This difference between this outcome and one that could be produced by random guessing is not statistically significant ($T[57] = 0.5; p = 0.6$), i.e., the data does not allow to reject the hypothesis that the subjects were unable to tell the difference between the professional editor and our system. In addition, the subjects were also asked to indicate (on a scale of 1 to 3) how confident they were in

their choice. The results are shown in Figure 5. Those students who guessed *in*correctly were on average actually slightly *more* confident of their choice than those who guessed correctly, although again the difference is not statistically significant (a $\chi^2[2] = 1.69$ test provides a p-value of 43%).

We conclude that our editing system indeed provides a good approximation of the quality delivered by a professional editor for this particular case study of lecture recording.

Figure 5. Confidence of subjects in their choice.

8 RELATED WORK

There exist several systems that perform automatic video montage for the purpose of creating a summary of some event(s). Examples in the scientific literature are [7, 9]. Also commercially available software offers this functionality, e.g., Muvee[2], Aescripts[3], Magisto[4] and Google Photos[5]. While related, this is essentially a different problem from the one that we have considered in this paper, where we want to produce full-length coverage of an event.

This problem of producing full-length coverage is also studied by [12]. Their approach also takes into account cinematographic rules regarding shot transitions and view selection. However, it requires about one second of computation time per frame and does not include an experimental validation of the quality of the produced video. In [17], a general scheduling algorithm is proposed to achieve optimal observability using multiple adjustable sensors. This algorithm is applied to the video editing problem in [4]. It requires cameras with (partially) overlapping field of views, and constructs a 3D-map with areas of higher interest. Quality of view is determined by amount of action, number of objects, visible events and a combined object score. The video montage is then made by maximizing this quality, while simultaneously minimizing inter-camera switching. This is a significantly different approach from ours, in which the cinematographic rules are not as explicitly present. The quality of videos created by this system was compared to that of professionally edited videos in an experiment similar to ours: 83% of their test subjects labeled the computer generated montage as professionally edited, while 93% of the test subjects labeled the professionally edited video as such. While this is not precisely the same experiment as we performed, this 10% difference strikes us as more significant than the 3% deviation from a 50-50 split that we observed. In addition, this method also requires $0.16s$ of decision making time per frame, which does not suffice to reach real-time performance at 25fps.

Both of these methods reduce the video editing problem to an optimisation problem. On the one hand, this explains their greater computational complexity, while, on the other hand, it also means that these methods can only edit each particular set of input streams in one particular way. By contrast, our probabilistic sampling method is more flexible and may offer the user a number of different alternative montages to choose from.

In [13], an automatic video editor is developed specifically for lecture recording. Three cameras are used, each with a specific purpose: an overview camera, an audience facing camera and a lecturer tracker. This work includes cinematographic rules that are specific to this setting (e.g., if a person in the audience asks a question, show that person in the montage). The quality of the produced montages was tested by an experiment in which subjects were asked to assign a score of 1 to 5 to both an automatically produced and professionally made montage. The automatically generated video scored an average of 2.8, while man-made video had an average score of 3.7. Again, this difference of 22.5% is more significant that the differences we observed in our experiments. This system was later extended to allow a greater variety of settings [18]. This work also added some additional rules to the system, but their impact on the quality of the montages was not experimentally verified.

Another setting-specific system is that of [8], which performs offline automatic video editing of a theater play. This system uses a setup with one static camera, from which a range of sub-views are cut to create multiple shots. The quality of the montages produced by this system was not experimentally verified.

9 CONCLUSIONS AND FUTURE WORK

This paper has presented an automated video editing system, which may play a role in reducing the cost of producing a video report of an event such as a lecture, sports game or musical performance. We have developed this system in a declarative way, by building a model of the non-deterministic decision process that a human editor could follow in order to produce a montage. By using the state-of-the-art Probabilistic Logic Programming system Problog, we are to sample from this distribution and thereby produce a montage. We have demonstrated that the computations needed to make all the required decisions can be performed in real-time and that—at least for the particular case study of lecture recording—the quality of the produced montage is almost indistinguishable from that produced by a professional editor.

In future work, we will examine the performance of this system in more complex settings than lecture recording and investigate how the same declarative model may be used for machine learning of the probabilistic parameters.

ACKNOWLEDGEMENTS

This work was funded by the KU Leuven Research Fund as part of the GOA project “CAMETRON”. The authors thank Dries Hulens for providing the video analysis module used in this work.

[2] http://www.muvee.com/home
[3] http://aescripts.com/automated-video-editing
[4] https://www.magisto.com/how-it-works
[5] https://photos.google.com

REFERENCES

[1] Punarjay Chakravarty, Sayeh Mirzaei, Tinne Tuytelaars, et al., 'Who's speaking?: Audio-supervised classification of active speakers in video', in *Proceedings of the 2015 ACM on International Conference on Multimodal Interaction*, pp. 87–90. ACM, (2015).

[2] Punarjay Chakravarty and Tinne Tuytelaars, 'Cross-modal supervision for learning active speaker detection in video', *arXiv preprint arXiv:1603.08907*, (2016).

[3] David B Christianson, Sean E Anderson, Li-wei He, David H Salesin, Daniel S Weld, and Michael F Cohen, 'Declarative camera control for automatic cinematography', in *AAAI/IAAI, Vol. 1*, pp. 148–155, (1996).

[4] Fahad Daniyal and Andrea Cavallaro, 'Multi-camera scheduling for video production', in *Visual Media Production (CVMP), 2011 Conference for*, pp. 11–20. IEEE, (2011).

[5] F. De Smedt, D. Hulens, and T. Goedem, 'On-board real-time tracking of pedestrians on a UAV', in *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR) Workshops. Embedded Vision Workshop*, (2015).

[6] Ali Diba, Ali Mohammad Pazandeh, Hamed Pirsiavash, and Luc Van Gool, 'Deepcamp: Deep convolutional action & attribute mid-level patterns', in *CVPR 2016, International Conference on Computer Vision and Pattern Recognition.*, (2016).

[7] Yanwei Fu, Yanwen Guo, Yanshu Zhu, Feng Liu, Chuanming Song, and Zhi-Hua Zhou, 'Multi-view video summarization', *Multimedia, IEEE Transactions on*, **12**(7), 717–729, (2010).

[8] Vineet Gandhi, Remi Ronfard, and Michael Gleicher, 'Multi-clip video editing from a single viewpoint', in *Proceedings of the 11th European Conference on Visual Media Production*, p. 9. ACM, (2014).

[9] Andreas Girgensohn, John Boreczky, Patrick Chiu, John Doherty, Jonathan Foote, Gene Golovchinsky, Shingo Uchihashi, and Lynn Wilcox, 'A semi-automatic approach to home video editing', in *Proceedings of the 13th annual ACM symposium on User interface software and technology*, pp. 81–89. ACM, (2000).

[10] Li-wei He, Michael F Cohen, and David H Salesin, 'The virtual cinematographer: a paradigm for automatic real-time camera control and directing', in *Proceedings of the 23rd annual conference on Computer graphics and interactive techniques*, pp. 217–224. ACM, (1996).

[11] Dries Hulens, Toon Goedemé, and Tom Rumes, 'Autonomous lecture recording with a ptz camera while complying with cinematographic rules', in *Computer and Robot Vision (CRV), 2014 Canadian Conference on*, pp. 371–377. IEEE, (2014).

[12] Hao Jiang, Sidney Fels, and James J Little, 'Optimizing multiple object tracking and best view video synthesis', *Multimedia, IEEE Transactions on*, **10**(6), 997–1012, (2008).

[13] Qiong Liu, Yong Rui, Anoop Gupta, and Jonathan J Cadiz, 'Automating camera management for lecture room environments', in *Proceedings of the SIGCHI conference on Human factors in computing systems*, pp. 442–449. ACM, (2001).

[14] V.W. Marek and M. Truszczynski, 'Stable models and an alternative logic programming paradigm', in *The Logic Programming Paradigm: a 25-Year Perspective*, eds., K.R. Apt, V.W. Marek, M. Truszczynski, and D.S. Warren, 375–398, Springer, Berlin, (1999).

[15] I. Niemelä, 'Logic programs with stable model semantics as a constraint programming paradigm', *Annals of Mathematics and Artificial Intelligence*, **25**, 241–273, (1999).

[16] L. De Raedt, A. Kimmig, and H. Toivonen, 'ProbLog: A probabilistic Prolog and its application in link discovery', in *Proceedings of the 20th International Joint Conference on Artificial Intelligence (IJCAI)*, pp. 2462–2467, (2007).

[17] Mohammad Rezaeian, 'Estimation entropy and optimal observability', in *PerCom/PerSeNS conference*, (2006).

[18] Yong Rui, Anoop Gupta, Jonathan Grudin, and Liwei He, 'Automating lecture capture and broadcast: technology and videography', *Multimedia Systems*, **10**(1), 3–15, (2004).

[19] T. Sato and Y. Kameya, 'PRISM: A language for symbolic-statistical modeling', in *Proceedings of IJCAI*, (1997).

[20] J. Vennekens, M. Denecker, and M. Bruynooghe, 'CP-logic: A language of causal probabilistic events and its relation to logic programming', *Theory and Practice of Logic Programming*, **9**(3), 245–308, (2009).