The priority curve algorithm for video summarization

M. Albanese\textsuperscript{a,}\textsuperscript{*}, M. Fayzullin\textsuperscript{b}, A. Picariello\textsuperscript{a}, V.S. Subrahmanian\textsuperscript{b}

\textsuperscript{a}Dipartimento di Informatica e Sistemistica, Università di Napoli, Napoli, Italy
\textsuperscript{b}Department of Computer Science, University of Maryland, College Park, MD, USA

Abstract

In this paper, we introduce the concept of a priority curve associated with a video. We then provide an algorithm that can use the priority curve to create a summary (of a desired length) of any video. The summary thus created exhibits nice continuity properties and also avoids repetition. We have implemented the priority curve algorithm (PriCA) and compared it with other summarization algorithms in the literature with respect to both performance and the output quality. The quality of summaries was evaluated by a group of 200 students in Naples, Italy, who watched soccer videos. We show that PriCA is faster than existing algorithms and also produces better quality summaries. We also briefly describe a soccer video summarization system we have built on using the PriCA architecture and various (classical) image processing algorithms.

Keywords: Video summarization; Video databases; Content based retrieval

1. Introduction

Despite the vast amount of work on video databases, and the existing work on summarizing video [1–6,29,30], there is no commonly accepted solution to the problem of automatically producing video summaries that take both content and user interest into account and scale to massive data applications. For example, if FIFA (the International Soccer Federation) wanted to sell videos of soccer games, there would be tens of thousands of such videos. Potential customers may wish to watch small clips of the video to decide which videos they wish to buy. FIFA may want video summaries to contain game highlights (such as goals, disputes, and so on). Though financial resources may be available to manually summarize each video, the ability to automatically summarize such videos is likely to be attractive. In a completely different setting, the archaeological site of Pompeii has vast video archives about different aspects of the site. In this application, the types of summaries would focus on the artistically or archaeologically important components.

In past work [7], we proposed a model for video summarization based on three important principles: the summary produced must be \textit{continuous}, must contain high \textit{priority} objects and actions/events occurring in the video, and must avoid \textit{repetition}. Unlike all prior work on video summarization that we are aware of, the CPR approach allows users or application developers to specify what their priorities are—the summary is then the best possible realization of their priority. As a consequence, our summaries are \textit{personalized} and not created as a one-size-fits-all solution.

This model was called the CPR model (continuity, priority, and non-repetition). We encoded the
relative importance of these parameters in terms of an objective function \( \text{eval} \) that evaluated the quality of a summary. The problem then was to find a summary \( S \) (set of video frames) from the video being summarized such that

1. the cardinality of the set of frames was less than or equal to a desired maximal summary length \( \ell \) and
2. there is no other summary \( S' \) satisfying the preceding condition such that \( \text{eval}(S) < \text{eval}(S') \).

We showed that the problem of finding an optimal summary (w.r.t. the objective function \( \text{eval} \)) was NP-complete—as a consequence, any exact algorithm to find such an optimal summary will be inefficient (even though we gave one) unless \( P = NP \). We therefore proposed three alternative heuristic algorithms called CPRdyn, CPRgen, and SEA to find summaries fast. All these methods attempted to use the objective function to find a good summary.

In this paper, we still retain the core idea from [7] that the CPR criteria are important. However, we use a completely different approach to finding good summaries fast. Our priority curve algorithm (PriCA for short) completely eliminates the objective function upon which the previous algorithms were based but captures the need for priority using priority curves. Instead, we leverage the following intuitions (shown pictorially in Fig. 1).

- **Block creation.** We first split the video into blocks—blocks could either be of equal sizes, or they could be obtained as a result of segmenting the video using any standard video segmentation algorithm [8–11]. The resulting segments are usually relatively small.
- **Priority assignment.** Each block is then assigned a priority based on the objects and events occurring in that block. Yet another alternative would use the audio stream and/or accompanying text associated with the video to identify the priority of each block. The priority assignment can be done automatically using object and event detection algorithms, or manually by employing a human annotator.\(^1\)
- **Peak detection.** We then plot a graph whose \( x \)-axis consists of block numbers and whose \( y \)-axis describes the block priorities. The peaks associated with this graph represent segments whose priorities are high. We identify the blocks associated with the peaks in this graph using a peak identification algorithm that we have developed.
- **Block merging.** Suppose now that several different blocks are identified as peaks. If two peaks are adjacent to one another, then it is likely that the two adjacent blocks in question jointly refer to a continuous event in the video. We therefore merge adjacent blocks on the intuition that the same or related events occur in these blocks even though the video segmentation algorithm has put them into different segments. This is because standard video segmentation algorithms use image information alone to segment and do not take semantics into account. But it may be possible to use audio and/or accompanying text streams to identify similar video blocks without relying on visual similarity alone.
- **Block elimination.** Next, we run a block elimination algorithm which eliminates certain unworthy blocks whose priority is too low for inclusion in the summary. This is done by analyzing the distribution of block priorities, as well as the relative sizes of the blocks involved, rather than by setting an artificial threshold.
- **Block resizing.** Finally, we run a block resizing algorithm that shrinks the remaining blocks so that the final summary consists of these resized blocks adjusted to fit the desired total length.

In this paper, we describe the PriCA algorithm and provide experimental results using a set of 50 soccer videos to show that the PriCA algorithm beats the best known previous algorithm both in terms of the computation time (to find a summary) and in terms of the resulting summary quality.

As most readers are certainly aware, the only really reasonable way to assess the quality of summaries is to have them evaluated by humans as any theoretical model may disagree with the

\(^1\)Our video summarization application uses both image processing algorithms and some manual annotations.
intuitions of humans (which are hard to express and capture computationally). We therefore used a group of 200 students at the University of Naples to test the summaries produced of the above 50 videos not only with PriCA, but also with other algorithms developed earlier.

2. The PriCA Algorithm

In this section, we describe how to implement each module of the PriCA algorithm. Some of these modules, such as block creation and priority assignment, can be implemented using classical image processing and recognition algorithms, while others require original algorithms.

2.1. Block creation module

The goal of the block creation module is to split the video into homogeneous blocks. As mentioned in the Introduction, each module in PriCA can be implemented in any number of ways. Here are some examples:

1. **Fixed blocks**: In the first way, the person interested in summarizing a collection of videos simply says that each block is a certain number of frames (e.g. she may say that a block is a collection of 1800 frames, representing 1 min of the video at a playback rate of 30 frames per second).

2. **Segmentation-based block creation**: Alternatively, we may use any classical video segmentation algorithm \([8–11]\) to split the video into a set of blocks. Each block is a segment returned by the segmentation algorithm. The video is thus represented as a sequence of blocks, possibly of varying sizes.

3. **Audio-based block creation**: A third method is to use audio streams associated with video in order to perform the desired segmentation. For example, every time we detect a new speaker, we may create a new segment. Thus for example, if person \(p_1\) speaks during the first 100 frames of the video followed by person \(p_2\) for another 40 frames followed by person \(p_3\) for 200 frames, we would have three segments of video segmented by audio. There are numerous speaker recognition algorithms \([12]\) in the literature. An alternative way in which to use the audio is to associate a vector with each audio-slice (which is a fixed duration of audio—usually a very small duration). This vector captures important audio properties such as pitch, intensity, loudness, etc. When two consecutive audio-slices have vectors whose Euclidean distance is below a threshold, we merge the slices together and find the vector of the merged slice. We then check whether the merged slice can be merged with the next audio slice and so on. When this is not possible, we have a completed segment and the new slice forms the initial part of a new segment.

In the applications of PriCA that we have been working on to date (using soccer videos), we chose to segment them by shot boundaries. In spite of a long history of research in this field, the problem of the shot boundary detection has not been completely solved yet. Sports video is arguably one of the most challenging domains for robust shot boundary detection due to

1. multidimensional aspects such as strong color or texture correlation between shots, due to a single dominant background color (soccer field, etc.),
2. large camera and object motions, and
3. cuts and gradual transitions (such as fades and dissolves) often present in sports video clips.

To detect shots in soccer videos, we have adapted an algorithm from \([13]\), based on the observation that frames belonging to the same shot are more similar than frames from different shots. The heart of the algorithm is based on the biological mechanisms of visual attention. The term “attention” captures the cognitive functions that are responsible for filtering out unwanted information and bringing to consciousness what is relevant for the observer \([14]\). They propose a novel similarity function based on a combination of color, texture, and shape features and the scanpath (which describes how the eye focuses on different parts of an image) of each single frame.

We set up the problem of detecting a shot change, given the change of consistency, as a Bayesian inference of the observer from his own visual behavior.

The proposed scheme allows the detection of both cuts and dissolves between shots using a single technique, rather than a set of dedicated methods. Also, it is well grounded in visual perception theories and allows us to overcome usual shortcomings of many other techniques proposed so far. Further, the proposed focus of attention (FOA)
representation is robust with respect to smooth view changes.

2.2. Priority assignment module

The segmented video produced by the Block Creation Module is fed into a priority assignment module which examines each block and assigns a priority to it.

In general, priorities can be assigned in one of two ways:

1. Qualitative prioritization. In this approach, a partially ordered set of priorities is initially determined by the summarization application developer. Each block is assigned a qualitative priority. When the priority lattice is small, the user might be able to state priorities of blocks very easily. However, there are obvious philosophical issues to be considered—suppose we have a simple lattice consisting of the priority values “low, medium, ok, high” with low $\leq$ medium, low $\leq$ ok, ok $\leq$ high, medium $\leq$ high. In this case, what does it mean for a user to say that the priority of block $b_1$ is ok but the priority of block $b_2$ is medium? These two priority values are incomparable.

2. Quantitative prioritization. Alternatively, the person summarizing a soccer video may specify priorities as follows (goal: 10, red card: 7, yellow card: 2, corner kick: 3.6, fight: 10, and so on). In this case, the user is using a quantitative scale. The advantage of the quantitative scale is that it allows a continuum of values and provides a quantified difference between the priorities of two blocks. For example, the above qualitative assignments say that a block with a yellow card is one fifth as important as one with a goal. This cannot be easily realized in the qualitative approach. It is important to note that in our framework, it is the relative ratio of the priorities that matters, not the priorities themselves. For example, if we replace all the priorities given above by multiplying by 10 (i.e. goal: 100, red card: 70, yellow card: 20, corner kick: 36, fight: 100, and so on), we would get exactly the same results.

It is important to note that the summary can and should change if the priorities are changed—even small changes can cause the resulting summary to change. This is not surprising and is true of any knapsack-style problem [15]—moreover, [7] shows that the knapsack style problem is reducible to the summarization problem.

Thus, though the qualitative prioritization method is an interesting idea, we chose to go with the more numeric quantitative prioritization method.

The priority assignment component can also be implemented in many ways, e.g. by using classical object/event recognition algorithms [16–18] which have been extensively studied in the literature. In our soccer video summarization application, for example, the priority assignment is done by using image processing algorithms for events such as goal shot detection or red card detection. Alternatively, in a military surveillance application that wants to assign high priority to gunshot detection in video, an image analysis program that identifies gunshots or explosions may be used to assign high priorities to such events. In a civilian surveillance application, high priority might indicate events denoting the entry or exit of a person into or from a monitored site. As a last resource, a video can also be annotated by a human being [31,32].

Our implementation was applied to the problem of automatic event detection in soccer videos [33]. We were interested in detection of such events as “goal”, “celebration”, “yellow card”, and “red card”. To detect these events, we have used the strategy proposed in [19]. These algorithms aggregate the shots extracted in the preliminary shot detection stage in order to form scenes characterized by the presence of “goal” events and match highlights. Each frame of the extracted shot is analyzed using a feed-forward neural network with a back-propagation algorithm. The network has been trained considering: geometric features; colors and texture; the presence of players (foreground, portrait or whole figure); detection of the ball, of the field and of the goal-mouth; detection of red yellow cards. Particular attention is given to the audio analysis in order to improve the detection or to detect relevant events such as goals and celebration: in fact, it is not difficult to believe that in a scene containing a “celebration” following a goal, the audio signal is inevitably higher than in normal actions of the game. In particular we have adopted a simple but efficient RMS calculus. Eventually, the goal event has been characterized through simple reasoning about the conjunctive simultaneous presence of several detected events, such as the presence of the player (foreground, portrait or whole figure), of the ball, of the field and of the goal area and of the celebration.
2.3. Peak identification module

Once the priorities have been assigned to each block, block IDs (increasing with time), together with their priorities, are shipped to the peak detection module. This module creates a graph whose $x$ axis consists of block IDs, and whose $y$ axis shows the priority of each block. The Peaks() algorithm we have developed can automatically find peaks in this priority curve. Fig. 2 shows an example graph and the peaks involved. Intuitively, a peak consists of a sequence of blocks containing high priority events.

Technically, let $b_1,...,b_n$ be the blocks in the video (e.g. after the segmentation process). Let $p_i$ denote the priority of block $b_i$ as determined by the priority assignment module.

**Definition 2.1.** ($(r,s)$-peak). Suppose $b_1,...,b_n$ is a video, $r \in (0,n/2]$ is an integer and $s \in [0,1]$ is a real number. Blocks $b_j,...,b_{j+r}$ are said to be an $(r,s)$-peak iff

$$\frac{\sum_{j-i<j+2r} p_i}{\sum_{j-i<j+2r} i} \geq s.$$

Suppose we wish to check if a sequence of $r$ blocks $S_1 = b_j,...,b_{j+r}$, constitutes a peak. The above definition looks at $r/2$ blocks before the sequence as well as $r/2$ blocks after the sequence, i.e. the sequence $S_2 = b_{j-r/2},...,b_j,...,b_{j+r},...,b_{j+3r/2}$ is considered. This latter sequence $S_2$ is of width $2r$.

We sum up the priorities of all blocks in $S_2$—let us call this sum $s_2$. Likewise, we sum up the priorities of all blocks in $S_1$ and call this priority $s_1$. Clearly, $s_1 \leq s_2$. If $s_1/s_2$ exceeds or equals $s$, then we decide that the contribution of the priorities of the peaks in $S_1$ is much larger than that in $S_2/S_1$ and so $S_1$ constitutes a peak.

It is important to note that $r$ and $s$ must be chosen by the application developer. However, our system contains default values for $r,s$ that the developer can change should he so desire. We discuss four cases below:

- **$r,s$ both large**: When $r$ and $s$ are both large, we find segments that are big (i.e. consist of a large number of blocks) where the priority is very high throughout this large segment. The disadvantage of having a large $r$ is that “local” peaks in a sequence of $r$ blocks may get missed.

- **$r$ large, $s$ small**: This option is not a very good one. If one chooses $s$ to be small, the number of sequences of blocks that form $r,s$ peaks will be very large.

- **$r$ small, $s$ large**: In contrast to the first case above, peaks in this case will consist of a relatively small number of blocks, but the peaks are unlikely to contain further subpeaks within them (which can happen in the first case above). When $r$ is small, one may be tempted to infer that lots of $(r,s)$-peaks will be found—however, this really depends on $s$. When $s$ is large, it seems unlikely that lots of $(r,s)$-peaks will be found unless the priority of blocks is more or less even throughout the video.

- **$r,s$ both small**: We do not recommend this option—a small $s$ is likely to produce a vast number of $(r,s)$-peaks.

Our recommendation is to pick $r$ small (but not too small) and $s$ large.

Fig. 2 shows two examples of peaks corresponding to $r,s$ values of $(6,0.65)$ and $(4,0.6)$, respectively. Dotted rectangles signify peaks, with $s$-values shown for the most significant peaks. As seen from the figure, peaks often occur in clusters. While the upper graph corresponds to wide ($r=6$) peaks, parameters in the lower graph allow for narrower ($r=4$) and slightly lower ($s=0.6$ as opposed to $s=0.65$) peaks. This is consistent with our discussion above that when $s$ drops in value, the number of peaks goes up. As result, the lower graph
contains more peaks and smaller clusters. In our implementation, the default values of \( r, s \) are \( r = 4, s = 0.6 \).

The Peaks algorithm below takes a sequence of video blocks and \( r, s \) values as input, and returns all blocks that belong to \( (r, s) \)-peaks:

**Algorithm Peaks**

\( v \) is a sequence of \( \text{card}(v) \) block-priority pairs

\( r \) is the peak width

\( s \) is the peak height

begin

\( \text{Res} := \emptyset \)

for each \( j \in [r, \text{card}(v) - r] \) do

\( \text{center} := 0 \)

\( \text{total} := 0 \)

for each \( \langle b_i, p_i \rangle \in v \) such that \( i \in (j - r, j + r] \) do

\( \text{total} := \text{total} + p_i \)

end for

for each \( \langle b_i, p_i \rangle \in v \) such that \( i \in (j - \frac{r}{2}, j + \frac{r}{2}] \) do

\( \text{center} := \text{center} + p_i \)

end for

if \( \text{center} \geq s \) then

\( \text{Res} := \text{Res} \cup \{ \langle b_i, p_i \rangle \in v | i \in (j - \frac{r}{2}, j + \frac{r}{2}] \} \)

end if

end for

return \( \text{Res} \)

end

The Peaks() algorithm slides a \( 2r \)-wide window along a sequence of blocks, computing the total sum of block priorities in that window (\( \text{total} \)). It then computes the sum of block priorities in a narrower \( r \)-wide window in the middle of the \( 2r \)-wide window (\( \text{center} \)). When the ratio of these two sums \( \frac{\text{center}}{\text{total}} \) exceeds the threshold \( s \), all blocks in the \( r \)-wide window are picked as a peak.

**Example 2.1.** Consider the very small fragment shown in Fig. 3. At some time, the Peaks() algorithm will focus its window of length \( 2r \) on the segment from \( j - (r/2) \) to \( j + (3r/2) \) shown in the figure. It will compute the sum of the priorities of the blocks in the entire window of length \( 2r \) (which is \( 5 + 41 + 8 = 54 \)) as well as the sum of the priorities of the window of length \( r \) in the center of the window of length \( 2r \)—the priority there is 41. As a consequence, the ratio of these is \( \frac{41}{54} = 0.76 \). If 0.76 exceeds the \( s \) that the user has picked, then the sequence of blocks from \( j \) to \( j + r \) is considered a peak.

**Example 2.2.** Consider the 35 block sequence shown in Fig. 4. We now describe how the Peaks() algorithm finds the peaks in this figure. Suppose \( r = 6 \) and \( s = 0.8 \).

- **Window from 1 to 12:** We initially start by looking at the first 12 blocks. The sum of the priorities of these 12 blocks is 59. If we look at the window of size 6 centered at the middle of the first 12 blocks (these are the blocks 4–9), the sum of the priorities is 36. The ratio, \( \frac{36}{59} \), is below \( s = 0.8 \).
- **Window from 2 to 13:** We now slide the window of length \( 2r \) one place to the right. At this time, the sum of the 12 block window is 60 and the sum of the 6 center blocks (blocks 5–10) is 44. The ratio is therefore \( \frac{44}{60} \) which is below \( s = 0.8 \).
- **Window from 3 to 14:** We now slide the window of length \( 2r \) one place to the right. At this time, the sum of the 12 block window is 58 and the sum of the 6 center blocks (blocks 6–11) is 48. The ratio is therefore \( \frac{48}{58} \) which is greater than \( s \). Therefore, blocks 6–11 are returned as a peak.

The algorithm continues in a similar fashion, finding peaks in blocks 18–23 \( (r = \frac{24}{29}) \) and 28–33 \( (r = \frac{39}{48}) \).
Complexity of Peaks() algorithm: The Peaks() algorithm has complexity of $O(r \cdot \text{card}(v))$—hence it is linear with respect to the number of input blocks.

Note that the performance of the Peaks() algorithm can be improved by avoiding computation of center and total iteratively in each iteration of the outer loop. After the first iteration of the outer loop, these values can be updated in constant time. Though including these optimizations complicates the algorithm somewhat, it is well worth doing—as shown in the algorithm OptPeaks() below.

Algorithm OptPeaks($v, r, s$)

$v$ is a sequence of block-priority pairs

$r$ is the peak width

$s$ is the peak height

begin

RES$:=\emptyset$

center$:=0$

total$:=0$

for each $(b_i, p_i) \in v$ such that $i \in [1, 2 \cdot r]$ do

total$:=\text{total} + p_i$

end for

for each $(b_i, p_i) \in v$ such that $i \in (\frac{r}{2}, \frac{3r}{2}]$ do

center$:=\text{center} + p_i$

end for

if center$\geq s$ then

RES$:=\text{RES} \cup \{(b_i, p_i) \in v | i \in (\frac{r}{2}, \frac{3r}{2}]\}$

end if

for each $j \in [2 \cdot r + 1, \text{card}(v)]$ do

total$:=\text{total} + p_j - p_{j-2}$

center$:=\text{center} + p_{j-\frac{r}{2}} - p_{j-\frac{3r}{2}}$

if center$\geq s$ then

RES$:=\text{RES} \cup \{(b_i, p_i) \in v | i \in (j-\frac{3r}{2}, j-\frac{r}{2}]\}$

end if

end for

return RES

end

Complexity of the OptPeaks() algorithm: The OptPeaks() algorithm has complexity of $O(\text{card}(v))$—hence while its complexity is linear with respect to the number of input blocks, it does not depend on the window size $r$ as in the case of Peaks()'s complexity.

2.4. Block merging module

The set of blocks thus identified in each peak is then shipped to the block merging module that examines these blocks and tries to determine if any of them can be merged. For example, it may turn out that there may be three blocks—the first containing the play just before a goal, the second containing the goal itself, while the third shows the post goal celebration. Either a standard clustering algorithm can be used to cluster blocks that should be merged together or a set of rules can be used to determine conditions under which multiple contiguous blocks can be merged together into a new block (whose priority equals the sum of the priorities of the blocks being merged).

Technically speaking, suppose $r, s$ are fixed either to the default values in our system or after the application developer changes the values to ones he likes. The peak identification algorithm eliminates all blocks that are not $(r, s)$-peaks for the $r, s$ values determined in this way. Let Peaks($v, r, s$) be the set of all blocks from the original video that contain peaks. Consider the set $\{(b_i, b_{i+1}) | b_i, b_{i+1} \in \text{Peaks}(v, r, s)\}$ of all pairs of blocks that are adjacent to each other. In general when adjacent blocks are peaks, there is some possibility (though it is not 100% certain) that they may describe the same event. For example, in our soccer video summarization application, we may have one block (peak block) that describes a goal event. The camera may have been switched to another block that also describes the same goal—however, the segmentation algorithm creating the blocks may treat these events as different events when in fact they describe the same event. This may be due to the fact that the segmentation algorithm is based on shot detection algorithms or other similar algorithms that solely use visual features to detect segments. The main goal of the block merging module is to merge adjacent blocks that may be very similar, so that repeating blocks can be treated as a single block in the later processing steps (such as resizing).

A block similarity function is a function $\text{sim}$ that takes two blocks as input and returns a non-negative real number as output. The bigger the number returned, the more similar the blocks are considered to be. There are many ways in which we could implement block similarity functions—for example, any multidimensional image analysis algorithm for matching images based on similarity can be used here [20]. Here are a few examples:

1. $\text{sim}_{\text{diff}}$: One possibility is that we could use any classical image differencing algorithm $\text{idiff}$ [21] to
return the similarity between two frames and we could set the similarity between the two blocks to be the similarity between the two most similar frames, drawn from each block. A multidimensional analysis technique [20] may be used for this.

2. \( \text{sim}_{\text{random_idiff}} \): An alternative is that the similarity algorithm randomly selects \( b \) frames from each block (where \( b \) is set to some small number) and then finds two frames, one from each block, that are maximally similar. This is a variant of the above algorithm and can also benefit from multidimensional comparison methods.

3. \( \text{simaudio} \): We could use an audio detection algorithm that extracts the audio associated with each frame and then uses audio features such as pitch, loudness, etc. to associate a vector with the entire block. The similarity between the two blocks is some function that is inversely proportional to the Euclidean (or other) distance measure between the two blocks. This is yet another example of a multidimensional matching method.

4. \( \text{simtext} \): In the event that the videos in question have an accompanying text transcript, we could identify the text blurb associated with each of the two blocks and set the similarity of the two blocks to be equal to the similarity between the two text transcripts using any classical method to evaluate similarities between text documents.

5. \( \text{simkeywords} \): Suppose we have a given set \( \mathcal{X} \) of keywords of interest. For each block \( b \), we associate a vector \( \tilde{b} \) of length \( |\mathcal{X}| \)—the \( i \)th entry in the vector denotes the frequency of occurrences of the \( i \)th keyword (or its synonyms). We then merge two adjacent blocks \( b_1, b_2 \) iff the Euclidean distance between their associated vectors is below a given threshold. Note that the keyword vector can be replaced by other vectors traditionally used in information retrieval [22].

6. \( \text{simvec} \): As is often common in image processing, we could associate a color and/or texture histogram with each block and return the similarities between the histograms using a multidimensional distance metric such as root mean squared distance or the \( L_1 \) metric [23].

As mentioned earlier, the \( \text{Peaks}(v, r, s) \) takes the same video, together with the \( r, s \) values used to identify peaks earlier. To simplify the block merging process, let us assume that \( \text{Peaks}(v, r, s) \) returns a set of block-priority pairs of the form \( (b_i, p_i) \), as opposed to a set of blocks, and adjacent blocks can be concatenated with the \( \oplus \) operator. \( \oplus \) can be implemented in any number of ways. Some easy ways to merge blocks is to simply concatenate the blocks together or to chromatically compose them together using operations such as fades and dissolves. The block merging algorithm then takes as input, any block similarity function between blocks (those listed above are just a few examples—many others are also possible), together with a set of block-priority pairs, and returns a new set of merged blocks-priority pairs, as follows.

Algorithm Merge(\( v, \text{sim}() \), \( d \))

\( v \) is a sequence of block-priority pairs

\( \text{sim}() \) is a similarity function on blocks

\( d \) is the merging threshold

begin

\( \text{Res} := \emptyset \)

\( B := \text{first block-priority pair } (b_1, p_1) \in v \)

for each \( (b_j, p_j), (b_{j+1}, p_{j+1}) \in v \) do

if \( \text{sim}(b_j, b_{j+1}) \geq d \) then

\( B := (B, b \oplus b_{j+1}, B.p + p_{j+1}) \)

else

add \( B \) to the tail of \( \text{Res} \)

\( B := (b_{j+1}, p_{j+1}) \)

end

end for

add \( B \) to the tail of \( \text{Res} \)

return \( \text{Res} \)

end

The Merge() algorithm considers all pairs of blocks \( b_j, b_{j+1} \), concatenating them together into a bigger block \( B.b \) (short for \( B.block \)), as long as \( \text{sim}(b_j, b_{j+1}) \) value stays above the threshold \( d \). The priority \( B.p \) of the newly merged block is computed as the sum of individual priorities of its parts.

Example 2.3. Let us continue with Example 2.2. The peaks identified are 6–11, 18–23, and 28–33. The Merge() algorithm merges these blocks as follows. For the sake of this example, let us define \( \text{sim}(b_1, b_2) = 1 - |p_1 - p_2|/(p_1 + p_2) \), where \( p_1 \) and \( p_2 \) are priorities of blocks \( b_1 \) and \( b_2 \), respectively, and set the threshold \( d = 0.9 \). Given these parameters, blocks 8–10 will be merged into a single new block with \( p = 28 \) and so will blocks 19–21 (\( p = 16 \)), 28–29 (\( p = 11 \)), and 30–32 (\( p = 33 \)). Thus, the total number of blocks decreases from 18 to 11 after merging.
Complexity of Merge() algorithm: The Merge() algorithm has linear complexity with respect to the number of blocks in its input.

2.5. Block elimination module

The set of blocks produced after merging is then shipped to a block elimination module. This module eliminates blocks whose priority is too low. For example, it may turn out that 10 merged blocks are returned after the block merging algorithm and these 10 blocks have a total of 5000 frames. If we want a summary consisting of just 3600 frames, we may have to realize that 10 blocks have a total of 5000 frames. If we define the concept of a cluster w.r.t. a threshold. In addition, we would like to consider eliminating blocks that are repetitive. For example, if we compute the average priority of the 10 blocks above to be 25 and the standard deviation to be 3, then we may want to eliminate all blocks with a priority under 16 (this is the classical statistical model which says that for a normal distribution, most objects in the distribution must occur within 3 standard deviations of the mean). Other statistical rules can also be used here.

Suppose $S$ is the set of blocks from the original video after the block merging step has been applied to the set of blocks in Peaks$(v, r, s)$. In the block elimination module, we would like to remove from this set, all blocks whose priorities are less than a certain threshold. In addition, we would like to consider eliminating blocks that are repetitive. For example, in our soccer application, we may have replays of a goal long after the goal was scored. Both the original goal and the later replay may have high priorities, but our summary should probably not include both of them.

The block elimination module may use a similarity function similar to those used in the block merging module to first identify similar blocks. Any similarity function similar to those used in the block merging module may be used here in the following manner.

We say that blocks $b_1, b_2$ are equivalent, denoted $b_1 \sim b_2$, w.r.t. $\text{sim}$ iff $\text{sim}(b_1, b_2) \geq t$ for some threshold $t$. In other words, the blocks are considered similar if the similarity function assigns them a similarity score that exceeds a given threshold. Note that any number of methods for multidimensional matching may be used here to determine if two blocks are similar. For example, we may have two different histograms (color, texture) associated with the blocks, and we may decide to use methods to compare color and/or texture histograms to determine if they are equivalent (which would mean that their similarity exceeds some threshold). Any such methods can be incorporated into our system as a possible implementation of $\sim$.

One may think that the $\sim$ is an equivalence relation, but in general it may not be so. The reason for this is that the fact that the similarity between $b_1$ and $b_2$ exceeds threshold $t$ and the fact that the similarity between $b_2$ and $b_3$ exceeds threshold $t$ does not imply that the similarity between $b_1$ and $b_3$ exceeds threshold $t$. As a consequence, we need to define the concept of a cluster w.r.t. a threshold $t$.

**Definition 2.2** ($t$-cluster). Suppose $\mathcal{B}$ is a set of blocks and $\text{sim}$ is a similarity function. A $t$-cluster of $\mathcal{B}$ is any set $B \subseteq \mathcal{B}$ such that for all $b_1, b_2 \in B, \text{sim}(b_1, b_2) \geq t$.

In other words, a $t$-cluster consists of blocks that are highly similar to each other (i.e. have similarity level $t$ or more according to the selected similarity function).

Given a set of blocks $\mathcal{B}$ returned by the block merging module, our goal is to split $\mathcal{B}$ into $t$-clusters. The key idea is that when a cluster has lots of blocks, it may be possible to just retain one of those blocks rather than keeping all of them as these blocks are all deemed to be similar. This introduces the concept of a $t$-partition given below.

**Definition 2.3** ($t$-partition). Suppose $\mathcal{B}$ is a set of blocks. A $t$-partition of $\mathcal{B}$ is a set $\mathcal{B}_1, \ldots, \mathcal{B}_r$ where $\mathcal{B}_1 \cup \ldots \cup \mathcal{B}_r = \mathcal{B}$ and $\mathcal{B}_i \cap \mathcal{B}_j = \emptyset$ and for all $1 \leq i < j \leq r$ it is a $t$-cluster.

Intuitively, a $t$-partition splits a set of blocks into clusters $\mathcal{B}_i$ such that each cluster is a $t$-cluster.

It is easy to see that a valid $t$-partition of $\mathcal{B}$ simply consists of the set $\{\{b\}|b \in \mathcal{B}\}$. In other words, if we simply split $\mathcal{B}$ by taking each element of $\mathcal{B}$ and making it into a singleton set cluster, we would have a valid $t$-partition. Clearly, this defeats our intent to group multiple blocks together. To ensure this, we need to define the concept of a maximal $t$-partition.

**Definition 2.4** (maximal $t$-partition). Suppose $\mathcal{B}_1, \ldots, \mathcal{B}_r$ is a $t$-partition of $\mathcal{B}$. We say that $\mathcal{B}_1, \ldots, \mathcal{B}_r$ is a maximal $t$-partition iff there are no $1 \leq i < j \leq r$ such that $\mathcal{B}_i \cup \mathcal{B}_j$ is also a $t$-cluster.

A maximal $t$-partition forces clusters that can possibly be merged to in fact be merged.

Our goal now is to first find a maximal $t$-cluster of $\mathcal{B}$ where $\mathcal{B}$ is the set of blocks returned by the block merging step. This is relatively easy: suppose...
\( \mathcal{B} = \{b_1, \ldots, b_m\} \). We first put \( b_1 \) into a cluster by itself and then we check if \( b_2 \) has similarity level \( t \) or more with this cluster. If so, we add \( b_2 \) into the cluster—if not, we go on to \( b_3 \). This process is repeated till we add as many blocks to the cluster associated with \( b_1 \) as possible while ensuring that this cluster is a \( t \)-cluster. We then continue to repeat this process to create cluster with the blocks not in this initial cluster. Finally, the process is completed by selecting one block from each cluster and eliminating all other blocks, as shown in the following algorithm.

Algorithm Cluster\((v, t)\)
\[
\begin{align*}
v & \text{ is a sequence of block-priority pairs} \\
t & \text{ is the clustering threshold} \\
\text{begin} & \text{ Res} := \emptyset \\
\text{while } v \neq \emptyset & \text{ do} \\
& \text{ B} := \emptyset \\
& \text{ for each } b \in v \text{ do} \\
& \quad \text{ if } \min_{b' \in B} \text{sim}(b, b') \geq t \text{ then} \\
& \quad \quad \text{ B} := \text{B} \cup \{b\} \\
& \quad \text{ v} := \text{v}\setminus\{b\} \\
& \text{ end if} \\
& \text{ end for} \\
& \text{ Res} := \text{Res} \cup \{\text{highest priority block from B}\} \\
\text{end while} & \text{ return Res} \\
\text{end}
\]

It is easy to see that the Cluster algorithm above is proportional to the number of blocks in the input to the algorithm—as such Cluster runs very fast indeed. Moreover, our Cluster algorithm allows us to use any notion of block similarity whatsoever when creating a summary.

Of course, our architecture allows other clustering methods [24] to be used in place of ours to identify blocks as being “equivalent” and clustered together.

Once the clusters are identified, our block elimination module picks one element from each cluster. In a sense, this one element is a representative of that cluster. Selecting an element from a given cluster can be done in many possible ways:

1. **Random selection.** One option is to randomly choose any member of the cluster.
2. **Prioritized selection.** Another option is to select a member of the cluster that has maximal priority. This strategy has the advantage that even though all blocks in a cluster are similar, one may have slightly higher priority than another and we might as well choose it.

3. **Prioritized ratio selection.** Another option is to select a member of a cluster that has the maximal priority vs. size ratio. As blocks can have varying sizes, it may turn out that the block with the largest priority is also pretty large—in this case, it may be better to choose a smaller block with fairly high priority as this smaller block contributes less frames towards the overall summary, thus allowing blocks from other clusters to be utilized.

4. **Size-oriented selection.** Another option is to merely be parsimonious and say that the block with the smallest size in each cluster will be selected.

Other strategies are also possible: rather than select one block from each cluster, we may be able to select multiple blocks. It could well be the case that a given \( \mathcal{B} \) has only five clusters. In this case, any of the mechanisms to select a single block from each cluster yields a total of five blocks and it is conceivable that these five blocks do not jointly account for the total summary length. Suppose \( b_{SS} \) is any strategy to select blocks from a cluster, and suppose we have split \( \mathcal{B} \) into clusters \( \mathcal{B}_1, \ldots, \mathcal{B}_r \).

In this case, we could iteratively make one pass through the clusters \( \mathcal{B}_1, \ldots, \mathcal{B}_r \), select a block from each cluster. If at the end of this, the total size of the selected blocks is below \( k \times sf \) where \( sf \geq 1 \) is some scaling factor, then we continue to select blocks from the clusters \( \mathcal{B}_1, \ldots, \mathcal{B}_r \) till this condition is violated. At this point, we stop and return all the selected blocks to the block resizing module. It is okay that the scaling factor be greater than or equal to 1 because the last component of our architecture may resize blocks if needed.

After removing repetitions, our block elimination module computes the mean \( \mu \) and standard deviation \( \sigma \) for the priorities of blocks in \( S \). Given a real number \( m \geq 0 \), let us define a function \( \text{Drop}(S, m) \) that drops from \( S \) all blocks whose priorities are less than \( \mu - m\sigma \). \text{Drop()} can be easily implemented by iterating over all blocks returned by the \text{Merge()} algorithm. Thus, the result of

\[
\text{Drop(Cluster(Merge(Peaks(v, r, s), d), t), m)}
\]

will be a set of all non-repeating high-priority merged peaks taken from \( v \), with respect to the \( r, s, d, m \) parameters.
Algorithm Resize $(v,k)$

$v$ is a sequence of block-priority pairs
$k$ is the desired summary length

begin
  $Res := \emptyset$
  $p_{\text{total}} = \sum_{(b,p) \in v} p$
  $p' := 0$

  for each $(b,p) \in v$
    if $\text{len}(b) \leq \frac{p}{p_{\text{total}}} \cdot k$
      $Res := Res \cup \{(b,p)\}$
      $v := v \setminus (b,p)$
      $p' := p' + p$
      $k' := k' + \text{len}(b)$
    end if
  end for

  $k := k - k'$

  for each $(b,p) \in v$
    $\text{alloc} := \text{round}(\frac{p}{p_{\text{total}}} \cdot k)$
    $b' := b$ truncated to $\text{alloc}$ frames
    $Res := Res \cup \{(b',p)\}$
    $p' := p' - p$
    $k := k - \text{alloc}$
  end for

  return $Res$
end

The $\text{Resize}()$ algorithm collects output blocks in $Res$ and starts by copying all blocks whose length $\text{len}(b)$ is smaller than the number of frames they would be allocated in the summary $(p \cdot k / p_{\text{total}})$. It then computes the remaining number of unallocated frames in $k$. All remaining blocks are truncated proportionally to their priorities to fit into remaining $k$ frames.

Example 2.5. Let us continue with Example 2.2. There are four blocks that have survived merging and elimination:

- blocks 8–10 with priority $p = 28$,
- blocks 19–21 with priority $p = 16$,
- blocks 28–29 with priority $p = 11$, and
- blocks 30–32 with priority $p = 33$.

Notice that all four are merged blocks, hence there are ranges instead of single numbers. Assuming that each “original” block corresponds to a single frame and the user requested a summary of 5 frames, let us see what the $\text{Resize}()$ algorithm does to our summary. First of all, given the total priority $p_{\text{total}} = 88$, all blocks will have to be resized. Block 8–10 has an allocation of $\frac{5 \cdot 28}{88} = 1.59$ frames. As we cannot split frames, this block has to be truncated to two frames 8–9. Block 19–21 has an allocation of $\frac{3 \cdot 33}{60} = 0.8$ and therefore gets truncated to a single
frame 20. By repeating this process, we also obtain frames 28 and 31. Thus, the final summary is made of frames 8, 9, 20, 28, 31.

**Complexity of Resize() algorithm:** The Resize() algorithm has complexity of $O(\text{card}(v))$—hence it is linear with respect to the number of input blocks.

3. Implementation

To evaluate the efficiency and effectiveness of PriCA, we have implemented a prototype system in JAVA on top of Oracle 8i and MS Access backends on a Windows 2000 platform. The system consists of approximately 2500 lines of code.

The prototype allows to both index and summarize videos, and consists of the PriCA algorithm implementation, as well as CPRgen, CPRdyn, and SEA algorithms [7], and a user interface for specifying the desired summary content. In addition, the system is capable of automatically segmenting video into shots and detecting soccer-related events, for annotation purposes.

In a typical example of system’s usage, the user selects a video she would like to process and the system checks whether this video is already indexed or not. In the latter case, the system will offer to index the video. Once the video is indexed, the user can modify the indexing, query the database to find specific blocks, or summarize the video.

Fig. 5 shows the indexing/querying interface to the previously indexed video. In this example, the user asks the system to retrieve all the blocks in which the action goal occurs. The resulting summary is listed at the left side of the interface and can be viewed through the video player in the top left corner of the interface.

Fig. 6 shows the summarization interface. Using controls in this interface, the user specifies the desired summary features. The system allows the user to select any of the four algorithms, and interface controls change according to the selected algorithm.

Fig. 6 shows an example of the summary specification. The blocks in the summary created by PriCA are listed on the left while Fig. 7 shows a representative frame for each of five resulting blocks.
Note that in order to summarize a video, we can use data structures such as those in AVIS [25] to determine what activities and objects occur in frames. This will clearly speed up the algorithms within the PriCA model.

4. Experiments

In this section, we describe a set of experiments conducted to evaluate the performance of the PriCA system.

A key issue in automated summary construction is the evaluation of the quality of the summary with respect to the original data. There seems to be general consensus on the non-existence of some universally accepted solution—this is mainly due to the many different approaches to the video summarization problem.

A number of alternative approaches are thus available. Considering user based evaluation methods, a group of users is asked to provide an evaluation of the summaries or to accomplish certain tasks (i.e. answering questions) with or without the knowledge of the summary, thus measuring the effect of the summary on their performance. Alternatively, for summaries created using a mathematical criterion, the corresponding value can be used directly as a measure of quality. However, all these evaluation techniques present several drawbacks; user-based ones are difficult and expensive to set up and their bias is non-trivial to control, whereas mathematically based ones are difficult to interpret and compare to human judgement.

We first compare the PriCA algorithm to the CPRgen, CPRdyn, and SEA algorithms proposed in [7], both in terms of time spent to compute a summary and quality of resulting summaries, then try to compare our results to the results of other authors. As the only way of evaluating quality of summaries is via human subjects, we engaged a group of 200 students from the University of Naples.

Our data set consisted of about 50 soccer videos, totaling about 80 h. The videos were segmented into blocks and annotated, as described in Section 3. The resulting blocks had an average length of about 10 s, with a relatively low variance.

**Performance:** To assess performance, we fixed the desired length of the summary to 60 s. We then varied the number of candidate blocks in the 4–75 range, by choosing an increasing number of events and subjects of interest.

The processing times were computed for each algorithm by averaging the results of 10 executions for each video. Fig. 8 shows times taken by different algorithms. From this figure, we can conclude that the PriCA algorithm outperforms the other three algorithms. This is true even without using the optimization for Peaks() mentioned earlier.

**Quality:** To assess the quality of results produced by the four algorithms being compared, we asked a group of approximately 200 students at the University of Naples to rate the resulting summaries on a 1–5 scale. The experiment was repeated three times, with desired summary lengths of 2, 4, and 6 min, for all videos. The results, shown in Fig. 9, indicate that summaries produced by the PriCA algorithm have been rated best in 48%, 46%, and 45% of all cases, respectively. These percentages are significantly better than those for the other three algorithms. Table 1 reports the standard deviation of user judgements for each algorithm, evaluated for both each selected summary length and independently of the length. The table clearly shows that the PriCA algorithm performs better than the others in terms of distribution of human judgements. Note that the standard deviation is approximately 10% of the total score range 0..5 for PriCA. This shows statistically that the algorithm is more robust than the other algorithms. Moreover, as the standard deviation is only about 10% this means that the system is quite robust.

We also performed a statistical analysis of the results in order to check for statistical significance. We first randomly picked two subsets of the
reviewers enrolled in the experiments and compared their judgements using the well-known statistical t-test. We obtained t-values smaller than 0.8. This indicates that our results are statistically valid. In particular we obtained even smaller t-values for larger values of the desired summary length, thus indicating a stronger agreement among reviewers about the quality of the summary when evaluating a longer summary.

It is easy to compare the four algorithms shown above because they share the same underlying assumptions: segmentation into shots, summarization based on high-level descriptions, summaries as sequences of shots. However, things quickly become complex when we try to compare the results of totally different algorithms on different data sets in different domains. In the next section, we discuss several evaluation results obtained by other researchers and compare them to our result, but this comparison has to be taken with a bit of salt for above reasons.

5. Related work

Our work falls into the domain of personalized video summaries in which a developer building summaries of a video collection can use our PriCA algorithm to ensure that summaries produced reflect what he thinks are important for the summaries to contain. We believe that personalized video summaries should have the following key features:

1. Personalizable. The summarization method must easily support personalization to a given application.
2. Seamless improvement. The summarization method must be sufficiently generic so that application specific image processing algorithms can be neatly plugged in and upgraded when a better method is found.
3. User length specification. The user should be able to set an a priori bound on the length of the summary, rather than have the system decide what the summary length should be.
4. Multifaceted. The system should be able to leverage both audio and video techniques in summarizing a video, not just rely entirely on one facet of a video (i.e. the video stream).
5. Quality of summary. The quality of the summary produced should reflect what the user thinks is important for his needs.
6. Computation speed. The summary should be computable at high speed.

We briefly review contributions in related work below and then conclude this section with a table (Table 2) that summarizes the pros and cons of each of the leading approaches.

Most current approaches to video summarization fall in two broad categories:

- Reasoning-based summarization: Reasoning-based approaches use logic or neural algorithms to detect certain combinations of events based on the information from different sources (audio, video, natural language). Examples of such approaches are video skims from the Informedia Project by Smith and Kanade [6] and movie trailers from the MoCA project by Lienhart et al. [5]. Sometimes multiple characteristics of a video stream are employed simultaneously: the video analysis is combined with the audio analysis (speech, music, noise, etc.) and even with the
textual information contained in closed captions. The heuristics, used to identify video segments of interest with respect to all these characteristics, are encoded with logical rules or neural networks.

- **Measure-based summarization**: Measure based-approaches use various importance and similarity measures within the video to compute the relevance value of video segments or frames. Possible criteria include duration of segments, inter-segment similarities, and combination of temporal and positional measures. These approaches can be exemplified by the use of singular value decomposition (SVD) by Gong and Liu [3], or the shot-importance measure by Uchihashi and Foote [2].

It is worth noting that most systems summarize video by key-frame extraction. For example, the Video Skimming System [6] finds key frames in documentaries and news-bulletins by detecting important words in the accompanying audio. Systems like MoCA [5] compose film previews by picking special events, such as zooming of actors, explosions, shots, etc. Finally, Yahiaoui et al. [27] propose an automatic video summarization method in which they define and identify what is the most important content in a video by means of similarities and differences among videos. They also suggest a new criterion to evaluate the quality of the summaries that have been created, through the maximization of an objective function. In contrast to the previous work discussed above, our paper introduces a more general framework which takes into account user content preferences (via block priorities) and produces continuous summaries while avoiding repetition. In addition, our method is not limited to a certain type of videos, but general enough to address many different classes of videos.

In [26], Shao et al. propose an approach to automatically summarize musical videos, based on an analysis of both video and audio tracks. They evaluated the quality of the summaries through a subjective user study and compared the results with those obtained by analyzing either audio track only and video track only. The subject enrolled in the experiments rated **conciseness** and **coherence** of the summaries on a 1–5 scale. The **conciseness** parameter does not have a corresponding one in our framework, since users explicitly specify the desired length of the summaries in our framework. The coherence parameter is similar, though not equivalent, to our quality parameter. It is important to note that this work only focuses on music videos, whereas our work is applicable to a virtually any video.

Just to consider another example, [28] reports an average value of 93.7—on a 1–100—scale for a **meaningfulness** parameter, that is evaluated through a user study too.

Table 2 summarizes the strengths and weaknesses of various video summarization methods based on the six parameters mentioned at the beginning of this section.

### 6. Conclusions

There is growing interest in summarizing video. Commercial enterprises with large video banks such as the US NBA or NCAA sports organizations, as well as military organizations deploying Predator and other video, have huge amounts of interest in identifying and summarizing video. For example, the NBA would like to summarize basketball games showing key events in the game so that sports fans purchase the videos based on watching key events in the summaries. Other sports organizations have similar interests. Military organizations have great interest in being able to summarize surveillance
videos so that key events of interest can be highlighted in the summaries.

In this paper, we have proposed a new architecture for creating video summaries. We have introduced the novel concept of a block priority curve and shown how the peaks in this curve can be used to create video summaries. Unlike prior work in video summarization that we are aware of, our approach is not limited to selecting key frames, but attempts to consider the fact that high priority events must be included in the summary and that it is important for the summary to be reasonably continuous and to avoid repetition. We have conducted detailed experiments which clearly show that the proposed PriCA algorithm is faster and produces much better summaries than our previous algorithms described in [7].

There is a ton of work that remains to be done in the future. A key area that we have mentioned in this paper, but not described in detail, relates to the problem of actually summarizing video using a mix of audio, text and raw video streams associated with a video file. For example, news reports are often accompanied by text streams as well as audio streams and these streams can be invaluable in eliciting content.

Another major research topic relates to the problem of summarizing video based on context. In this paper, we can summarize video based on priorities—however, a number of methods can be used to express those priorities and often to automatically learn those priorities so that the summaries shown to different users are different.

Acknowledgements

We thank the anonymous reviewers for many excellent comments that have significantly improved the quality of the paper. This work was supported in part by the Army Research Lab under contracts DAAL0197K0135 and DAAD190320026, the CTA on Advanced Decision Architectures, by ARO contracts DAAD190010484 and DAAD190310202, by DARPA/RL contract number F306029910552, and by NSF grants 0205489 and IIS0329851.

References


