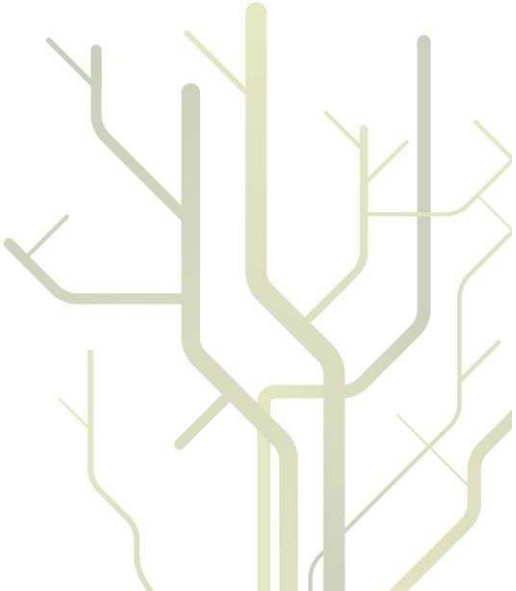
UNIVERSITY OF TROMSØ UIT

FACULTY OF SCIENCE AND TECHNOLOGY
DEPARTMENT OF COMPUTER SCIENCE

Time Constrained Video Playback

Webjørn Larsen

INF-3981
Master's Thesis in Computer Science
June, 2010



Abstract

The duration of video playlists, might be a problem when watching on mobile devices, while on the go. We are exploring the problem of how to reduce the size of a video playlist, while maintain the important parts of the videoclips in the playlist. We have designed algorithms that automatically can cut down a playlist by cutting the duration of clips or sometimes even removing clips. We have shown that our algorithm is better than a naive solution that just cut at the end. Furthermore we have looked at playlist cutting for Davvi. This means playlist cutting for the soccer domain. We have created a hint based cutting algorithm that uses domain knowledge to achieve good results compared to the other algorithms available.

Acknowledgements

First, I would like to thank Håvard D. Johansen, my advisor during this thesis. His guidance and feedback has been invaluable, while working on this thesis.

Thanks to Professor Dag Johansen, for his lecture on how to write a master's thesis, and for his inspiring lectures over the years.

I would also like to thanks the rest of the staff at the Computer Science Department, for all the support and help over the years, and Jan Fulgesteg in particular for all the help during my time as a student.

Special thanks to friends and family.

Contents

List of Figures

Chapter 1

Introduction

Watching video on the Internet is very popular. Many Internet sites now have video delivery as one of their main functions. This includes sites hosting user created content like YouTube[15] and Vimeo; sites hosting professionally generated contest like NRK Nett TV [3]; and sites hosting sports content like VG Live. YouTube is one of the most popular web sites for such activity. The videos available on YouTube are created and uploaded by the users themselves. This is one of the reasons why it is so popular. It is a way of sharing video with your friends and family. It is exciting to be able to see that several thousand of viewers, have watched the latest video that you uploaded.

Users are also increasingly accessing these video services on the go using portable devices like mobile phones and PDAs. A survey conducted by the Nielson Company shows that the rate of employment of mobile video access has increased with 52.2% from 2008 to 2009 [5]. The reason for this can not be explained by the new and exciting Internet services for video alone. It is also due to recent advances in technologies for battery, processor, memory, screen and others. This has made mobile devices quite capable video playback devices. We bring with us mobile phones almost everywhere. This means that they are the only video playback device available in many situations. Also data connectivity, including Internet access, is much better today on mobile phones, then just a few years ago. So streaming solutions like Smooth Streaming [7] becomes interesting in the context of mobile devices. This means that the new smart phones and Apple Iphone are used more as entertainment devices then merely for phone calls.

The common method for putting together music and videos is by creating playlists. These playlists can be shared through e-mail, instant messaging, or even published on the Web. Internet video services allows users to create and share playlists of the videoes, they have in their systems. The length of a video playlist might be a problem when watching on the go. It might either be that the user do not have time to watch the entire playlist, do not have time to download the entire playlist, or do not want to take the cost of downloading the entire playlist over the data plan in the subscription. For such reasons, having a

system that can take in a playlist and cut it down to a certain length and make the resulting video clips available for transfer to a mobile device would be very useful. It quickly becomes apparent that it will be very beneficial to be able to change cutting algorithm on the fly in the system, since it might be different cutting algorithms that works best for different video topics or domains.

Video summarization methods can be used, for example [9], to cut down the size of the video clip and we can be certain that we maintain the most important parts of the videoclip. The problem is that importance graphs or methods of detection how important certain part of a videoclip are in relation to say a search term is not easy to implement in large scale. So on the Internet today there are no video search provider that can point inside the video clip to the part of the video that the user was interested in. Typically you will be searching based on text description of the video, title, and user tags. So maybe the video clip you get back as a result from searching for Beckham and goal, gives you a video containing a goal scored by Beckham, but you will have to watch say 3 minutes of other goals before you arrive at the point in the video where Beckham curled his free kick into the top right corner. If there was an importance graph for the search term Beckham and the term goal, it would peak around the time in the video where Beckham indeed scored the goal. If we have such importance peaks it would only be a matter of taking along the parts of the clip with a importance rating over a certain point.

Video summarization techniques does not work because they were not created for the scenario with video playlists. Different methods have to be used for a video playlist, since it containing many short videoclips. The motivation for this thesis is that we need this reduction capability for video playlist, much the same as we need good search engines on the Internet today to find the relevant web-pages amongst billion of potential sources. The ability to cut down the size of videoclips, while maintain the interesting parts of the video would save the user time. Today a user would typically click around in the video, skipping ahead to see if there is something of interest or just watch the whole video. Skipping ahead at random points gives a high chance of missing the interesting part, and watching the whole video can be too time consuming.

1.1 Davvi

Davvi is a video archive retrieval system [14], that has been created at the University of Tromsø as a part of the iAD project. Davvi allows users to create on-the-fly personalized videos by combining video clips, so that they might be played as a continuous video. The system have two main components: a video search and composition component and a video delivery component.

The video search and composition component allows the users to find video clips by using search. From the search result the user can pick and choose which videoclips they want to see, by adding them to their video playlist. After having selected the videoclips, they can arrange the clips into the order they want to watch them, then the user just starts watching the playlist.

At this point the video delivery component of the system is used. The videoclips are cut into two second segments. A torrent-like HTTP-Streaming approach is used to deliver the video segments to the clients. This allows playback from any point in the video. Similar streaming solutions have been created by Move Network [12], Microsoft's Smooth Streaming [7], and Apple's HTTP Live streaming [11]. So based on the composition put together by the user, the user ends up only streaming down the parts of the video needed to watch the playlist composed by the user. This is efficient use of bandwidth.

The segments are encoded in such a way that they are small self contained videos. Davvi uses a closed group of pictures with H.264 video and MP3 audio. It also use an efficient custom made container [17] to avoid the large overhead of the MPEG2 streaming container, improve the startup and jump latency.

The segments are made available on the Internet, by using standard web-servers. A tracker holds information regarding where the different segments are stored and users can download them with HTTP GET requests. Davvi stores the video segments in different qualities on the web-servers, on the client side video segment quality is selected according to the available bandwidth, so a smooth video playback is possible.

Metadata for the Davvi system is extracted from web sources like VG Live[1] and BBC Sport[2]. Davvi uses extraction engines build to find the the textual description of the football matches and then match the time of the text event from the external sources, with the game time in the video. By combining these data sources, Davvi makes it possible to do Google like searches on events that happens in football matches. In the index in Davvi, events have fields like duration, type, date, hometeam, awayteam, game result, gametime of the event, etc.

At the moment, Davvi does not have support for reducing the size of the playlist, once it is created. This ability would be a key feature for watching on the go, it would be essential if we wanted to make automatic playlists generation based on search topics, because the user would only want to specify duration. And the system would then take the most interesting videoclips and generate a playlist, then reduce the duration based on user specification.

1.2 Problem Definition

In this thesis we investigate methods and algorithms for cutting down a video playlist so that it fits a user specified time constraint. There is a default naive solution, the user would start at the beginning and watch until he run out of time. The part of the playlist that is left unwatched is cut. We will call this method for end-cutting of playlist.

> *The goal of this thesis is to devise a video cutting algorithm for a Davvi like video-archive retrieval system that outperforms the default*

[1] `http://www.vglive.no`
[2] `http://www.bbc.co.uk/football`

end-cutting of playlist algorithm.

We will compare the methods we find and design against this end-cutting of playlist algorithm, since it is what you would have to do without any alternative method.

We will need to find or create other cutting algorithms that is capable of reducing the duration of playlists. And we must evaluate these cutting algorithms against each other to find out how they perform under different circumstances.

Given that the system has full understanding of which part of a playlist is important, then a solution would be to just remove unimportant video clips in the playlist until the the total time of the playlist is under the total time you have available for watching the video clips. The optimal solution is basically the Knapsack problem, which is known to be NP-complete [2]. Hence only an approximation would be possible when the number of items in the playlist goes beyond a certain point. However, existing video retrival systems do not provide a full understanding of videoclip importance. So we have to solve the problem under other conditions.

1.3 Interpretation, Scope and Limitations

In order to devise an playlist cutting algorithm for a Davvi like system, we will have to come up with the rating models for videoclips that is needed for the cutting algorithms to work. The cutting algorithms shall be designed in such a manner that they are possible to implemented in a Davvi based system. We will focus our efforts on implementing the parts that we need to be able to verify that the cutting algorithms work and that we can run the needed experiments and simulations to evaluate the behaviour of the cutting algorithms, and see how they perform against each other.

We assume that we have a rating for the video clips similar to what Davvi gives us. It can be ranking from the search result, or a rating based on user feedback or basically any other rating that is available in a Davvi based system.

We will need to come up with efficient algorithms for cutting video playlists and we need to be able to compare the different algorithms that we find and create with each other to see how they perform regarding maintaining the overall rating of the video playlist. Overall rating is the sum of the individual rating per clip in the playlist.

Another thing to remember is that a video playlist is an ordered collection of videos. The creator of the playlist put the videos together with a given order, it might have a meaning in itself. Sorting the playlist based on rank and taking along the highest ranking videoclips is not a viable solution.

We will also need to define how rating per clip is affected when we cut in a clip. What happens to rating of a clip that is cut down in length. Does it lose rating relative to the amount cut of the video clip, or does it maintain the rating? This is something that we will need to look into, because it will matter for our synthetic comparison between different cutting algorithms.

A good starting point would be with cutting algorithms that works well for any kind of videos. In other words we can not start with video cutting techniques that are domain specific. After covering general techniques we explore or discuss certain methods that we have discovered to be useful for certain domains of video clips. The general idea is that the more we know about the video clips the better we can design cutting algorithms that will hopefully cut away parts that is not critical for the video.

One of the goal of this thesis is to figure out how to use playlist cutting algorithm in existing systems, where it can reduce the playback time of a video playlist to a user specified duration and transfer only the required parts of video clips over to the client.

1.4 Method

A short definition of Computer Science from the article Computing as a Discipline by Denning et al. [6]

> "The discipline of computing is the systematic study of algorithmic processes that describe and transform information: their theory, analysis, design, efficiency, implementation, and application. The fundamental question underlying all of computing is, ' What can be (efficiently) automated?' "

In reason years newer definitions have been given. However the fundamental question. "What can be (efficiently) automated? " is just what we are looking at, we try to create algortihms to automate the process of cutting video playlists.

We will do this in the following way in this thesis, we will describe models of how videoclips can be rated and based on these models run simulations to discover the behavior of the cutting algorithms we have found and designed. This means that we will have to run simulations and create prototype implementation of the cutting algorithms. Our approach is probably not theoretical enough to be considered algorithmic, however we do not have a typical systems research either, it is a combination of the two.

1.5 Outline

In Chapter 2 we cover the background work regarding video encoding and cutting of video clips, video summarization and playlists. Chapter 3 presents a natural rating model for videoclips, and the Priority Curve Algorithm. Chapter 4 will contain a description of the naive video cutting algorithms we have created and simulations based on these algorithms for Davvi based systems. In Chapter 5 we present our hint based cutting algorithm for Davvi. Chapter 6 contains a case study of how to use hint based cutting in a SMIL based streaming system. Chapter 7 contains concluding remarks and future work. There is a summary in the end of each chapter that contains the key points from each chapter.

Chapter 2

Background

In this chapter we will cover the background material and describe the existing solutions that is relevant for this thesis. Since video summarization techniques are a way of reducing the length of videos we will look into this topic in particular. Also, because playlists are such an important concept of this thesis, we will give some background information regarding this topic as well.

2.1 Video Encoding and Cutting Video Clips

Video is encoded by key frames and the difference in the frames based on the key frames. A keyframe is a whole frame, with all the color information. Group Of Pictures (GOP) is an MPEG term, it is a collection of consecutive video frames. It specifies the structure in which intra-and inter-frames are arranged. A encoded video stream consists of successive GOPs. From the picture information in the GOPs visible frames are recreated.

Lets have a look at the MPEG2 format which is a commonly used format. MPEG2 frames can be either intra-coded frames (I-frame), predictive-coded frames (P-frames) or bidirectionally-predicted-coded frames (B-frame). An I-frame is based on itself, it does not require data from preceding or following frames. In other words it is a keyframe. P-frames uses the preceding frame to calculate the current visible frame. B-frame uses both the previous reference frame and the next reference frame to calculate the visible frame, because of this it provides more compression than P-frames. B-frames are never reference frames. In Figure 2.1 we can see the MPEG2 structure, a MPEG2 video consists of GoPs. A typical GOP for MPEG2 would look like this IBBPBBPBBPBBPBB(I). MPEG2 has I-frames every 15th frame, which gives a good cutablitiy. However MPEG2 format is not every efficient for transfer over the Internet because it is not compressed enough compared to other formats.

Starting the playback of the video clips at an arbitrary places will not work out very well. If you cut at a non key frame the video playback will just be

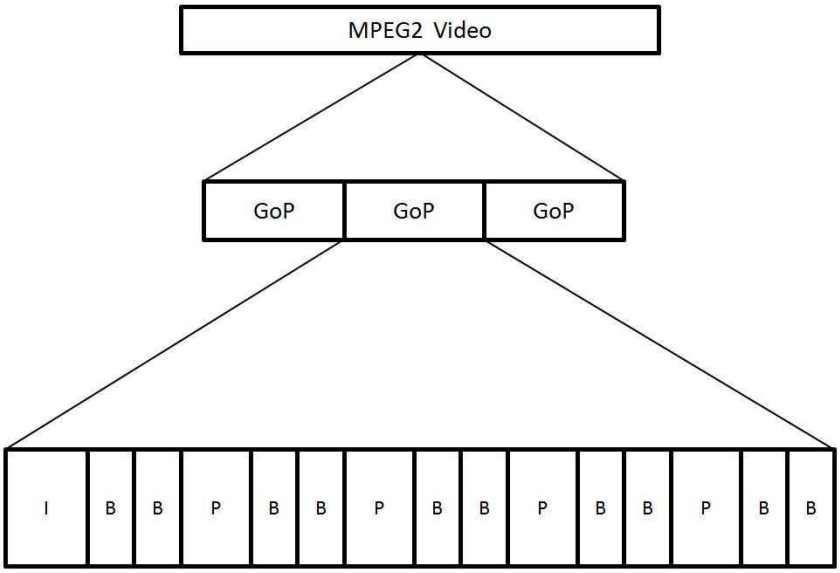


Figure 2.1: MPEG2 format Structure.

a jumble until the next keyframe, because of this it is not trivial to handle cutting in video. To make a smooth playback from an arbitrary point, the previous keyframe to that point must be transfered, so that we can compute a new start frame for the playback. The means extra data transfer and CPU usage, to compute a correct start frame, at the new starting point in the video.

Another option if we want to cut at a fine granularity would be to have the videos in question encoded in a format that have key frames at short intervals. It is also possible to re-encode the video clips so they have keyframes more often. Of course the more often there is a key frame in the video the larger the size of the video. This might lead to a conflict between the ability to cut at the intended places versus transfer cost. Making unnecessary large video files will remove the point of cutting away parts of the video clips, if the uncut version with a normal rate of keyframes is of equal size or smaller.

2.2 Video Analyse

By analyzing the visual content of a video, information about its content can be deducted. It is however alot harder for a computer to detect and find visual features in images or video then for a human. Our brain is very good at detecting

patterns and features in images and identify what they are. Computers typically work on a lower level and are not nearly as good as the human brain.

Color histogram is a common method of analyzing video. Techniques based on frame clustering, the video is cut into segments and clustered into groups based on color histograms. From each segment one or more key frame is extracted [10].

Techniques based on frame clustering by dimensionality reduction, groups video frames at fixed intervals based on a bottom up clustering of the video. High dimensional feature vectors are created for each frame and either reduced to a much lower dimensional space or by local approximations to high dimensional trajectories. The clustering of frames is performed in the lower dimensional space. Gong et al. [20] use singular value decomposition (SVD) to cluster frames evenly spaced in the video sequences.

Based on edge detection, it is possible to write detectors of specific object like a football or a basketball. Unfortunately such detectors are often error prone. For simple objects like a football it is not that hard to write a detector because a ball looks the same from every angle. To write a general car detector is a lot harder because you need to be able to recognize the car from every angle. A picture or a video frame of a object is a two dimensional representation of a three dimensional object. This makes it difficult to write a general car detector because there are alot of different sizes and forms of cars, and when you need to be able to detect them from any angle that they are shown on the picture or video frame it becomes difficult. This means that the detectors can often miss objects.

Perhaps the most successful field of object detection algorithms is facial recognizion. Facial recognizion uses either geometric approaches, that looks at distinguished features or photometric, which uses a statistical approach, images are distilled into values and comparing values with templates to eliminate variances. Facial recognizion software is often used by law-enforcement agencies. A new trend is 3-D face recognition. A three dimensional model of the face is created by sensors and more features are identified, such as the contour of nose and chin and eye sockets.

2.2.1 External metadata

Manual video annotation produces more accurate video meta data than existing video analytic tools. Although manual annotation is a labour intensive task, it is in some cases possible to obtain such data through external sources. For instance like described in Section 1.1, Davvi takes the text transcripts from football sites and match the time from the text events described on the websites, and match them to the time in the video and just based on that manage to find the important events in the football video.

Techniques using closed-caption or speech transcript are efficient in news programs, documentaries, or instructional videos. Most broadcast programs have closed captions, and for videos that does not have that it is possible to use speech recognition to create speech transcripts. Once the text and video

is matched, text summarization methods can be used to get a text summary. Based on the text summary video segments that corresponds to the text can be classified, this method can also be used for video summarization.

2.3 Video Summarization

Video summarization techniques can be divided into two different categories: reasoning based summarization and measure based summarization [9]. Reasoning based summarization techniques use logic and neural algorithms to detect certain combinations of events based on the information from different sources like video, natural language and audio [19]. Measure based summarization techniques use importance and similarity measures within the video to calculate the relevance value of video segments or frames. Possible criteria include time of segments, similarities between segments, combination of positional and temporal measures [20].

Most video summarization systems are based on key-frame extraction. Note that in the context of video summarization a key frame represents an interesting part of the video. This is different then in the context of video encoding, where keyframes are inserted at regular intervals to support skipping. An example is the Video Skimming System [19], which finds key-frames based on detecting important words in the audio part of the video.

Adami et al. [1] gives an overview of video shot clustering and summarization techniques for mobile applications. They describe several methods of video summarization, including static storyboards, dynamic video skims, and sports highlights. Static storyboards as end result is not interesting for us, because a picture containing a row of keyframes is not what we are after.

Taskiran et al. [4] gives a characterization of video summaries. They cover amongst other summary application domains, type of program content, whether it is event based or uniformly informative content and summary visualization methods. Video summaries can differ based on the application domain.

They give a number of different reasons why a summary might be created;

- Intrigue the viewer to watch the video, movie trailers are a good example.
- Give the viewer enough information to decide whether or not the viewer wishes to see the entire video.
- Decide whether or not he has seen the video before.
- Help find certain segments of interest in the video.
- Consider the relevance of the videoes returned by a search engine without the need to watch the entire video.
- Enable users of handheld devices to watch video easier without spending too much processing power.
- Give the viewer the most important information contained in the video.

For all the different reasons given above to create summaires of vidoes, it comes down to two main functions: indicative function and informative function. Indicative function is a summary that indicate which topics are in the original video. Informative function is a summary where the most important information in the video is covered as much as possible, often based on a user specified length to the summarization system. These two functions are not independent and most systems are a mixture of these two basic functions.

Event based content is video programs that is easily divided into events, good exmaples include talk shows, sports or news programs, all these programs have clearly defined boundaries, and it is possible to detect them. Uniformly informative content on the other hand does not have these clearly defined boundaries and are therefore harder to summarize, examples are sitcoms, presentation videos, soap operas and home movies.

Event based content is easier to find and handle since they are well-defined, it is possible to use domain knowledge to detect important events, and because of that achieve much better results then when using general summarization techniques. The problem with domain specific solutions is that you need to create new solutions for each domain and have people developing them that has the domain knowledge.

Speedup of playback is a technique that can get a reduction factor of 0.4-0.7 of the original length, user studies shows that speedup of 0.59 did not give a significant lose in comprehension [4]. The problem is that a summarization ratio of 0.59 might not be sufficient, often the goal is to reach 0.1-0.2

Techniques using domain knowledge, the majority of the work in this field is within sports programs. The main reason is that sports generally have clear rules and clearly defined events that are possible to detect, cheering crowds, stop in the game, replays of significant events, etc. The number of interesting events are also often quite small compared to the overall length of the sports program. Another important factor is that sports summaries are only relevant as news stories a short time after the sport event took place, so alot of work has been put into this field. The major television companies and news channels need to present the results in form of a short summary of the sport event in question to millions of views shortly after the event took place.

Soccer is a field that has received alot of attention. Replays can be used to detect the most important events. The producer will normally give replays of goals, red and yellow cards, large scoring chances, etc. Such replays can be used to generate a summary. A replay is often followed by a closeup of the key players or the crowd. There are techniques that detect whether or not the camera is zoomed in close to the field or not, and based on the dominant color classify frames as long, medium or close to the field. Crowd excitement is also used with great success, if the crowd explodes into a frenzy it is a very good indication that it might be event worth taking along in the summary.

Approaches using multiple information streams is a sensible solution since alot of the techniques when you combine them produces much better results then they do individually. Combining the information from image data, audio and closed-caption can increase the quality of the summary by making sure that

you pick out the most significant parts of the video.

Money et al. [16] gives an overview of video summarization. They go througt the field of video summarization classifying the work until 2008. This paper is a good starting point if you want to read more about video summarization.

2.4 Playlist

A playlist is basically a list of the video or music that you want to play. There are a number of different playlist formats including .m3u, .smil, .VLC, .asx, XSPF and .wpl. They are all typically either connected to a certain media player or created for a certain purpose like playlist sharing. While a playlist is basically a list of songs or video, playlists can contain a lot more information. All playlists have location of the media to play, and very often name, duration, artist, album, etc, in other words meta data regarding the songs or videos in the playlist.

Synchronized Multimedia Integration Language(SMIL) [8] is a W3C recommended XML markup language for creating multimedia presentations. Supporting media items like text, images, video and audio, links to other SMIL presentations and files on different web servers. SMIL markup is written in XML and has similarities to HTML. The current version is SMIL 3.0 and it became a W3C Recommendation in December 2008. SMIL is so much more then just a playlist format, however we are not going into the other features of the SMIL specifications. The main reason for our interest in SMIL is the support for start and end point specification for a media playback.

2.5 Summary

This chapter has covered background work. We have looked into video formats and encoding, video cutting, video summarization, playlist and playlist-formats.

Chapter 3

Natural Rating Model for Videoclips

We are going to cut away parts of videoclips to reduce playlist, and we aim to achieve better results than just randomly cut away at the videoclip and hope that we are lucky and managed to keep the important part of the video in the cutdown version. So to be able to do better then just random cutting, we need a way of rating the importance of the videoclip. Humans evaluate and rate video continuously as we watch it, and based on the topic we are looking for, we would naturally without even consider the fact, rate certain parts of the video more important that others. This is the process which we need to find a way of modeling down to a computable model.

3.1 Priority Curves

The finest level of granularity in a video clip is a frame. So if we were to attach a rating to a videoclip, the finest level would be to give rating per frame. This means that it is not possible to have continues graphs from a mathematical point of view. For human perception it is hard to understand what happens in a frame, so we would have to increase the duration of the part we would like to give a rating to, so maybe a GoP would be a suitable size, Davvi uses two second segments. The important fact is that we have rating at discrete unites of time, may that be 2, 5 or 10 seconds. Because of these rating values at certain point in the video, it is possible to make a importance graph over the videoclip. We will call these graphs for priority curves, because based on the priority curves we know which parts of the video clip we should prioritize, in keeping in the cut-down version of the video clip.

So we have established that there needs to be rating at discrete unites of time in the video clip. The ratings are set by video analysis techniques or manual annotation. So we have at computer level priority curves that shows how important the discrete part of the video is to the topic in question.

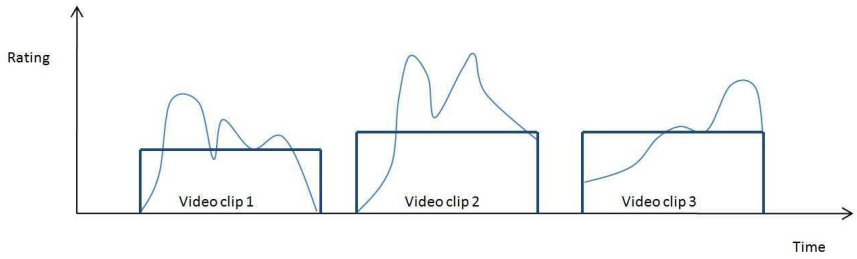


Figure 3.1: Natural rating model vs current state of art.

In Figure 3.1 we can see the current state of how rating is handled, each bar representing a fixed rating for the videoclip. The height is the rating and the lenght is the duration of the videoclip. The curve shows the real rating, this rating would be individual for each human being. The height of the bar is the rating a videoclip would get from a search engine. There is no information except for the the ranking from Davvi's search engine's text-based ranking algorithm. We can see that there is a huge different between the current model of rating for video clips and the model that best fits reality, priority curves. Video search today on the Internet would at best give a rating for the entire videclip, based on the search topic. So there is a huge gap between the fairly accurate model we have described and current state of the art, in the field of video search.

New priority curves would have to be made for every new topic, if the rating is topic based. It is possible to just have a general rating based on e.g user feedback, where we would aggregate the ratings from the users, then the video would just be rated as good or bad on a scale from e.g one to ten. In video search for entertainment purpose this might be good enough, but if the purpose is for an information retrieval system, we would need topic based ratings.

3.2 The Priority Curve Algorithm

The Priority Curve Algorithm (PCA) presented by Fayzullin et al. [9], is one method for generating summaries of continues video. For soccer matches, they detect important events like goals, and red and yellow cards by using video processing techniques and manual annotation. Then they rated each event with a number from one to ten where ten is most interesting. By doing this they created a importance graph for the video. Similar to the natural rating model we defined in Section 3.1. Then they run a peak detection algorithm and then based on how long the summary should be they pick the events from the video with the highest importance.

Here is a short description of the main phases in the PCA algorithm, as shown in Figure 3.2.

The first phase is the *block creation phase*. The continues video is cut into

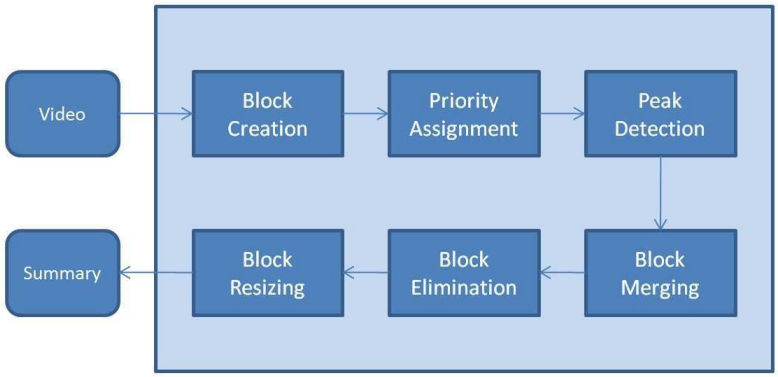


Figure 3.2: Priority Curve Algorithm Phase Overview.

a number of blocks. It might be fixed size for instance 1500 frames per block or video segmentation algorithms like for instance shot-boundary detection [21] can be used. Typically these segments are quite small in size. Block creation is decided by the application. However if the application support multiple ways of doing this, it is possible to allow the user to select which method should be used.

The second phase is the *priority assignment phase*. Based on video analyzing techniques important events are found in the video. Segments are prioritized based on user preferences. Segments containing goals for instance will get the priority ten, while red cards segments will get seven and corners maybe three. These priorities are of course just examples, but they make sense. Goals are the most important event type in a soccer match, and if there is a red card you would want to see it. So a high priority on important segments makes sense.

Once the blocks have a priority, the *peak detection phase* starts. A peak detection algorithm is executed to find the parts of the video with high priority. This is the core process of the PCA approach, it finds consecutive blocks that have a high priority. The peak detection algorithm uses two important parameters r and s: r is the numbers of blocks in a peak and s is the relative height of a peak based on the surrounding blocks. The peak detection algorithm checks whether the sum of the r blocks divided by the sum of the 2r blocks (see Figure 3.3) is larger than the value s. If it is then it has detected a peak. For example, consider the blocks 3 to 6 in Figure 3.3. If we apply the PCA algorithm on those blocks using $r = 4$ and $s = 0.65$ then the sum of r (blocks 3-6) is 36, and the sum of the 2r(block 1-8) is 48. $36/48 = 0.75$, in other words larger then the parameter s. So r is a peak. As we can see the Priority Curve Algorithm checks whether r number of consecutive blocks are a peak in their surrounding. It is a relative value. This means that it is possible to detect low peaks. In the examples in Figure 3.3 there is a high peak, however it is possible to detect peaks

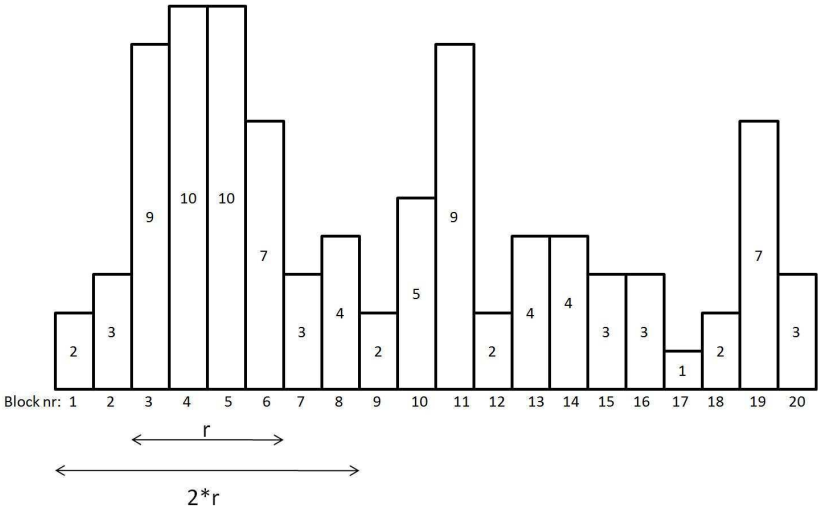


Figure 3.3: Peak detection example 1.

with lower top values, but to be able to say that they are peaks the surrounding must be even lower.

In Figure 3.4 we see that the peak detection algorithm has moved further along the video clip. And based on the figure it seems like we have arrived at another peak. However given $r = 4$ and $s = 0.65$, the sum of r(blocks 10-13) is 20 and 2*r(blocks 8-15) is 33 , $20/33 = 0.6060$, which is below s, so it is not a peak. As we can see the values of r and s are determining which blocks are detected as peaks. Setting correct r and s is up to the application programmer according to the PCA paper.

After having selected only the peak blocks from the video, the *block merging phase* merges adjacent blocks. This is done to preserve continuity in the video summary. At this time there is also executed an algorithm to eliminate any blocks that are identical. Typically goals will be replayed in a video of a soccer match, however in a summary you do not want to waste time showing the same goal twice.

After the block merging phase we have a smaller number of blocks, however the summary might still be too large. So the *block elimination phase* is executed. It removes blocks with low relative priority compared to the rest of the blocks in the summary. It basically calculate a threshold and removes any block that has a rating below that threshold. The threshold might for instance be computed like this, we take an average of all the rating of the blocks, then we remove the blocks that have a rating lower then a quarter of the standard deviations. This might remove low peaks that was taken along based on the peak detection

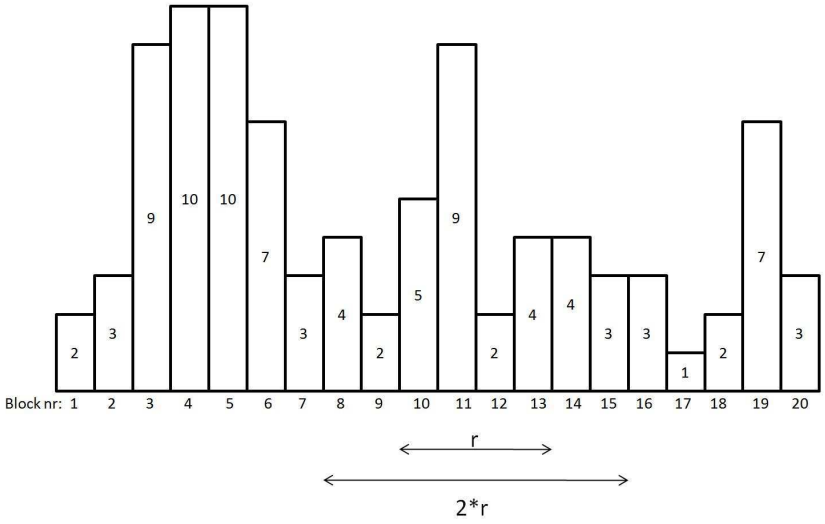


Figure 3.4: Peak detection example 2.

algorithm.

In the end the summary might still be too large and then the remaining blocks are resized in the *block resizing phase*. This is done by removing some frames from each block based on how much each block is contributing to the overall rating of the summary. So an important block is only cut a little bit, while a less important block will be cut more.

3.3 PCA's Applicability for Davvi

The Priority Curve Algorithm looked so promising on paper regarding how it managed to take along only the highly rated parts of the video, that we had to run some simulations based on the algorithm to see how it really worked. We also needed to check how it worked in our setting with playlists and multiple videoclips. In this section we will present the simulation results from our PCA simulations.

This is how the PCA simulations were implemented in Python. Playlist objects contains videoclip objects. Videoclip objects contain block objects. The blocks were given a random rating. When we cut in a videoclip we remove one or more blocks from the list of blocks that belong to that videoclip. This means that rating is lost every time a block is removed. We do not have time as a duration factor for videoclips. It is number of blocks that decides how long a videoclip is in these simulations. In reality the blocks would have a fixed length

for instance 2 seconds like Davvi, but for our simulations it does not matter. We also keep all the original generated rating data, so that we can check the cutting simulations against the original data to see if it works correctly.

We will use the same length on the videoclips, since these simulations are more about finding the peaks in the priority curves, then to see how it performs on videoclips of different length. We can still set how many blocks there are in the videoclip. It is just that all the videoclips in the playlist will have the same number of blocks.

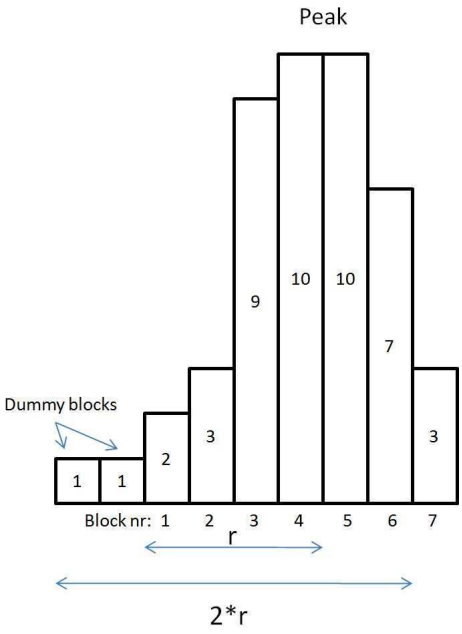


Figure 3.5: Shows how dummy blocks have be added at the start of the videoclip.

The beginning and the end of each videoclip have to be handled as a special case, because if we are to start detecting peaks from the first block in the videoclip, then the 2r blocks would start at a non-existing block. To work around this we use dummy blocks at the start and the end of the videoclip. These dummy blocks will have rating set to one. In Figure 3.5 we see an example, which uses $r = 4$ this means that we have to add two dummy blocks in the start and the same will have to be done at the end of the videoclip to support peak detection from the first to the last block in the videoclip. This might be a potential error source, because it will probably increased the chance of getting peaks in the start and the end of the videoclips. However, we do not see any other way of handling this.

Our PCA simulation will be running parameters $s = 0.60$ and $r = 4$. We have configured the simulation in the following fashion. 100 playlists each con-

taining 20 videoclip. Each of the videoclips contains 20 blocks. The rating per block is randomly set to a number between one and ten.

This gives us the following results for ideal random data[1]. The average rating per block in PCA summaries of the playlists are 7.29. The average number of blocks taken along per playlist is 93 out of 400. The average total rank per playlist after it was cut down by the PCA is 677.89, while the average total uncut rank per playlist is 1889.78. This gives a average relative total rank of 0.358. The average summary size is 23.2% of the original size.

We can see from these results that the PCA manages to reduce the playlists to around 23.2% on average. This is expected because $r = 4$ means that we check for peaks that are 4 blocks long, and with 1 peak per videoclip. and the peak has a length of 4 out of 20 blocks per videoclip equal 20%. Also the ideal random data typically have peaks that are 4-6 blocks long based on how the random numbers are generated. With one peak per videoclip getting an averge size of 23.2% was as expected.

Setting a good r value is important for the PCA, if r is too low or too high it will have a significant impact on how many peaks are detected. When we were running our simulations. We come to realize that a video clip that had the same rating on all or the majority of the blocks, even if it was high rating, would not get a hit on the Priority Curve Algorithm. This is because the relative difference between the r blocks in the possible peak you are checking against the 2r blocks is not large enough.

We also ran into another problem when we were generating random data for our PCA simulation. When the rating per block was a random number between one and ten, it often did not created detectable peaks in the ratings. So once again the peak detection algorithm showed a weakness. Still after thinking about it, we can not attack the algorithm itself, because it was designed for long continues video with certain events that would create a detectable peak in the rating, like a goal or a yellow card, but it is a weakness for our purpose.

So to make simulation data that was random, but still managed to have peaks that was detectable, we had to take more care when generating the video clip ratings. We created something we will call ideal random data. We created 3 data sets, one front important, one middle important and one end important. Underneath we have listed the datasets from our Python code.

```
DataSet0=[8,8,7,6,3,3,3,3,3,3]
DataSet1=[3,3,3,7,8,8,7,3,3,3]
DataSet2=[3,3,3,3,3,3,6,7,8,8]
```

So whenever we created rank to a block in an event, we first on videoclip level randomly picked one of the datasets. So lets say we picked DataSet0, the first block would then get a rank of 8 +/- 2, a value ranging from 6-10. So while the numbers would be random, proper peaks would be created most of the time. It is of course possible with a rank list like [6,6,5,5,5,5,5,5,5,5] , but

[1] See futher down in this section for the description of ideal random data.

it is not very likely. Most of the time we will get a peak within the data, when we used this method.

For events with more blocks then ten we map the block down to one of the ten values in the dataset. E.g with 20 blocks, the two first blocks uses the first entry in the dataset, the next two blocks uses the second entry and so on. Using this method we managed to create somewhat random data which contained peaks that the PCA algorithm should be able to detect.

We have also run the PCA algorithm on totally random data, where each rank was set by picking a random number between one and ten. We have also implemented a exponential smoothing algorithm to generate smoother random data, but we think neither random or exponential smoothing of random numbers gave good rating of the video clips.

We do not think totally random is a good way to generate realistic data for our simulations. The reason is that the video clips would not have a rating that is all over the place, high one block, then low, then medium, etc. It would follow certain patterns like front, middle or end important. At the very least it would increase or decrease gradually, which totally random rank data does not do, it jumps around randomly.

3.4 Discussion of PCA

The PCA algorithm is better then any other we have found regarding how to choose the most important parts of the video. Of course one problem still remains, and it is how to find and rate the important parts of the video. Regarding how this is done the article does not really give much new insight. They only give reference to video processing techniques and they also state that they use manual annotation. While this is possible in a small scale, manual annotation is something that we need to move away from, since it is simply not feasible to manually annotate the amount of video that is available on the Internet. In small scale systems or certain domains, it might be possible, for instance a sports reporter or journalist that is already writing about or reporting on a soccer match, might highlight the important parts of the video with corresponding tags, and based on this it would be possible to do importance searches in the video from that soccer match. But only the most popular games get that kind of attention, so unless it is possible to automatically detect and tag parts of the video with either key-words or a rating, it is not possible in a large scale to create summarizations this way because the importance graph would be missing, for the majority of the videos. Still in the soccer video domain it is working quite well and we will move forward with a system based for the soccer domain, much like they did with the Priority Curve Algorithm.

The big difference between what we want to do and what they did in the Priority Curve Algorithm is that they were creating a summary from a long continues video, we however are starting out with many short videoclips that may or may not have anything to do with each other. So it is not given that the entire algorithm will work on video clips that are as small as the ones we

have. In many ways only the top of the priority curve was taken along when the videoclip was created, so the entire video clip is high priority from the beginning. However due to time constrains we are forced to still cut away more from the playlist. So in a sense we have to assume we are removing interesting content, however we have to raise the threshold for a block to be taken along in the new cut of the videoclip.

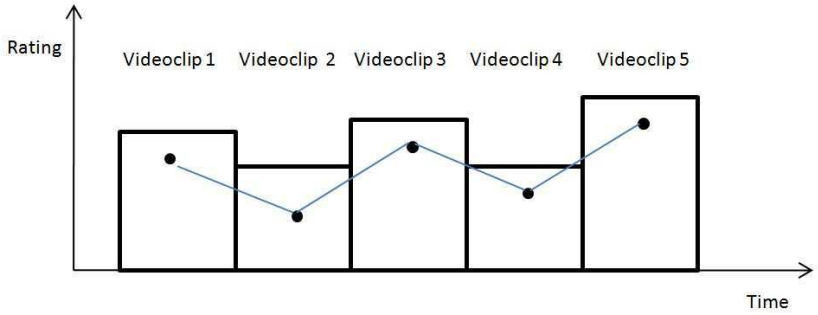
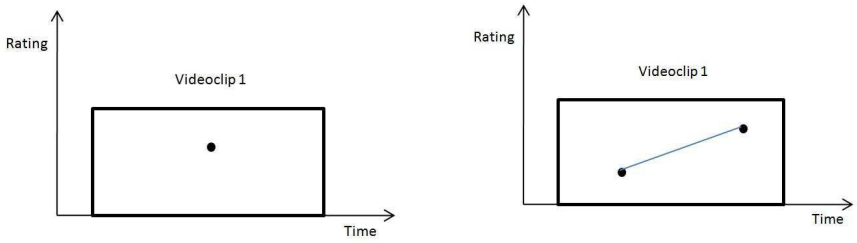


Figure 3.6: Priority curve based on one data point per videoclip

The main issue is that Davvi only have one rank per videoclip for the search term in question. This means that if we were to use the Priority Curve Algorithm, we would have only one data point per videoclip in the priority curve, see Figure 3.6. The PCA algorithm clearly needs more data points to work as intended. To make matters worse, there might not be any direct connection between the videoclips, except that the creator decided to put them after each other, so it could be very wrong to run the PCA based on such a setting.

If there is just one data point (see Figure 3.7a) it is not possible to decide which part of the videoclip that is the most interesting. There must at least be two rating points in the video clip, for any priority based cutting to be possible. We can see in Figure 3.7b that once there are at least two points it is possible to draw a line between the two points and based on this line decide how the cutting should be done for the videoclip.

The key to using the PCA efficiently is to set good r and s values. This is not very easy. Still it seems like it is better to set slightly too large r, then to set it too small. It also seems to be hard to set r unless you have some knowledge about how long typically peaks in the data should be. This also means that if some peaks are narrow, and some peaks are wide, it might not be possible to detect both types of peaks, since the r value will only fit one of them. We experienced this problem ourself, when we were testing the simulation code. The simualation reported back that the peak detection algortihm was missing alot of peaks in the playlist. It turned out that the r value was set too low, which meant that many of the wider peaks were not detected. This shows one of the problems of the algorithm.



(a) One data point (b) Two data points

Figure 3.7: Videoclips with just one and two data points.

Yet another the problem with PCA was that it was very hard to have control on how long the summary would become, because the r and s values are application depended and if there are alot of peaks in a video clip the PCA will take along more of the video clip. For example a match with many goals will give a summary that is longer then a match with fewer goals, since PCA will find more peaks in the match with many goals then in the other match. This shows that PCA is not suited to cut the playlist to a certain percent size, e.g 60%. When we realized this we understood that we could not use the PCA algorithm for the cutting we needed to do.

Also, we can not use the priority curve model, because Davvi does not support the rating needed to correctly use the model, which again means that we can not use the Priority Curve Algorithm.

3.5 Summary

In this chapter we have defined the priority curve model for rating of videoclips. We have found and presented the Priority Curve Algorithm. We ran simulations to see the behavior of the PCA algorithm. Based on this we have come to the conclusion that priority curves and the PCA does not suite our needs for fine grained cutting that is needed in Davvi.

Chapter 4

Cutting Algorithms for Davvi based System

In the previous chapter we presented the priority curve model for internal rating in videoclips. We also presented the Priority Curve Algorithm. Based on simulations and our findings on how the PCA worked we came to the conclusion that priority curves and the PCA does not suite our needs for fine grained cutting. Davvi only have a single rank per videoclip for the search term in question, so in this chapter we are looking at cutting algorithms that work on single ranked videoclips.

We need models that tells us how to perform and evaluate these cutting algorithms, and so we will first presenting the models we going to use in the simulations we present later. We will present models for videoclip length distribution and models for how the ranking of a videoclip is influenced by cutting in the videoclip. We will also present simulation results and show how the different algorithms compare to each others under the different models.

4.1 Playlist, Videoclip and Rank Model

A playlist is a collection of one or more videoclips, that is put together either by a person or by a video search engine. We are working under the assumption that the playlist is a result from a video search engine like Davvi. The Davvi paper however is missing clearly defined models for playlist, videoclip and rank. Also cutting of playlists have until now not been supported so the definitions of how this should be modeled and executed has currently not been described. We are now going to define how these models looks in a Davvi based system.

4.1.1 Playlist model

A playlist contains of one or more videoclips, each videoclip has a rank. A videoclip is specified by a start and end point. When we cut a video clip

we change either start or end-point or both of them. When we talk about a video playlist it is normally a list with a given order. There might be relations between the videoclips. For instance, the videoclips might be from the same soccer match, and they should then be organized in such a way that goals are shown in the same order as they are scored.

Another assumption that we make is that our search results are narrow and precise. It is not likely that anyone would want to watch the playlist of a video search result if the result contains 1000 000 search results, and total playtime is lets say 15 hours and 40 minutes. While the problem of reducing the amount of videoclips to a watchable size is an interesting problem, it is not something we will look into. We are looking at more specified video searches.

There has been work done that shows that users only look at the top 20 search results regarding search on web pages, of course search on web pages and video search is a bit different but still the lesson learned is that unless the document or item is in the top 20 results, users are not going to look for the information they are after deeper in the result set. And since it is much more time consuming to watch video clips compared to scaning through web page search results, working under the assumption that users will typically just use the top 20 results in video search is not that far fetched. Typically 10-40 videoclips per playlist is a realistic size to assume.

We have until now said that the playlist is the result of a search result in a video search engine, but that might not be the case. It could also be generated by a recommendation system based on a user profile, which contains say favorite player, favorite team, rivaling teams, etc and gives ranked video clips to the user in playlists for consumption after each new soccer match has been played. These video clips would not be ordered on rank but on time causality. This fits with our assumption that it is not given that playlists are sorted by rank.

4.1.2 Videoclip model

In vidoe search systems like Davvi, each videoclip depicts a single event. In the soccer domain this could for instance be a goal, a read card being issued, or a scoring opportunity. A video search result will assign each mathcing videoclip a single rank value.

Rank model

Each videoclip has a single rank in the simulations we are now going to run. However we also need a rank model, which tells us how the rank of the videoclip is affected by cutting in the videoclip. Here we have three models, fixed rank, where the rank is not changed by cutting in the clip, increased rank where the rank increases when we cut in the clip, because the rest of the clip becomes more important and the last model is decreasing rank where the rank decreases as we cut away part of the clip. Reason being that by removing frames we lose information, which again means that we lose some of the rating.

An example of a videoclip that only becomes more and more important when cut down in size might be a goal. The rating of the clip gets better as we cut away the beginning of the clip because it only contains the foreplay to the goal, so we improve the event rating of the goal by only showing the goal.

The idea behind fixed rank is that we do not change the rating when cutting in the videoclip, since we can not be sure whether we increase or decrease the value to the end user. We trust the rating from the system and assumes it will hold even if we cut in the videoclip. This model clearly works best for cutting algorithms that are removing whole clips, since any algorithm that does not remove whole clips will get the same total rank of the playlist until whole videoclips have been cut away.

Distribution models

We do not have realistic length distribution data, so we will have to explore syntetic distributions. In our simulations we will use two different distributions for the length of the videoclips: normal distribution and expo-variance distribution. We are covering both of these cases since we are not certain what the typical distribution would be like. It is probably domain specific, and as such both distribution models are interesting.

In the current implementation of Davvi, the length of each event is set to 30 seconds. An event is found by matching text description with a time-stamp to the video, and then they assume that by taking along 15 seconds before the time-stamp and 15 seconds after the time-stamp should be enough to make sure the entire event is inside the videoclip. This means that the normal distribution model is the model that comes closes to Davvi's fixed length distribution. Based on this we could say that the events are 30 seconds and just go with that, but we think that the current implementation is simplifying it too far. Assuming that all events are 30 seconds is most likely not correct. The reason why this method was used was probably because it was easy to implement.

A way of fixing this issue would be to allow user feedback when they watch the events to give new boundaries for the events. By aggregating enough user data, it should be possible to set new and more appropriate start and end points for the events.

4.2 Naive Cutting Algorithms for Davvi

In this section we will describe three different naive cutting algorithms that is possible under Davvi's current single ranked videoclip model. We will use pseudo code and plain text to describe them.

4.2.1 End cutting of Playlist

The first algorithm is end cutting of playlist. We briefly described it in the introduction and named it the default solution. In this section we will explaining

how end cutting of playlist algorithm works. Every videoclip has a time variable, that represents the duration of the videoclip. So when we need to cut the duration of the playlist, we start from the front of the playlist and checks for each videoclip whether the time of the videoclip plus the total time of the videoclips already taken along is lower the the new time constraint for the playlist. If it is lower we will add the videoclip to the list of videoclips that is part of the new version of the playlist. If it is higher we have come to the point in the playlist where we need to cut the videoclip and only take along a part of the videoclip. The rest of the videoclips after that videoclip in the playlist will be cut away.

```
list=[]
newtime=t
totaltime=0
foreach videoclip in Playlist:
   if newtime > totaltime + videoclip.time
      totaltime+=videoclip.time
      list.append(videoclip)
   else
      partial=cut(videocclip)
      list.append(partial)
      break

return list
```

4.2.2 Uniform Start & End Cut

The second algorithm we will cover is uniform start & end cut. This is how the uniform start & end cut algorithm works. Based on total lenght of the original playlist and the new time constraint, it is possible to calculate a total cut amount. The algorithm will divide the total cut amount on the number of videoclips in the playlist to calculate how much that needs to cut away from each videoclip. Once this is done, it is only a matter of iterating through the playlist and for each videoclip set a new start and end point based on the amount that needs to be cut away. This means that the amount that needs to be cut per videoclip is divided by two, since we plan on cutting both at the start and at the end. Half of the cut amount per videoclip is added to the start point, and the same amount is substracted from the end point. In the end we have to check whether or not the duration of the videoclip is still above zero, if it is larger then zero we add it to the list of videoclips still in the playlist. The algorithm described here is simplified because in our simulations we have to handle the situation when the amount to cut per videoclip is larger then the duration of the videoclip.

```
list=[]
totalCutAmount=a
```

```
perVideoClip=totalCutAmount / len(Playlist)

foreach videoclip in Playlist:
     videoclip.start+(perVideoClip/2)
     videoclip.end-(perVideoClip/2)

     if (videoclip.end - videoclip.start) > 0:
           list.append(videoclip)

return list
```

4.2.3 Lowest Rank First(LRF)

The third and final naive cutting algorithm we will look at is lowest rank first. This is how the lowest rank first algorithm works. For the purpose of cutting, we sort the playlist based on rank and removes the lowest ranking videoclips first. The lowest ranking videoclip we take along might be cut so it fits inside the new time contraint. So when we have sorted the playlist on rank we can just start to play from the top of the playlist until you run out of time. This means that we can use the same algorithm as end cutting of playlist. Except that we need to restore the original order amongst the videoclips still in the playlist after we are done cutting.

```
list=[]
newtime=t
time=0
Playlist=sortOnRank(Playlist)
foreach videoclip in Playlist:
   if newtime > time + videoclip.time
      time+=videoclip.time
      list.append(videoclip)
   else
      partial=cut(videocclip)
      list.append(partial)
      break
list=restoreNormalOrder(Playlist,list)
return list
```

4.3 Behavior study of Naive Cutting Algorithms

In this section we will present and discuss our simulations of the three different naive cutting algorithms for Davvi. First we are going to describe how we

structured and executed the simulations of the cutting algorithms. Then we will present graphs showing the behaviour of the algorithms under the different rank models and length distribution models.

A playlist is a list of videoclip objects. The videoclip objects have a rank, and start and end points that can be modified. We randomly generate a rank, and duration by setting start and end points, per videoclip object in the playlist. Then based on cutting algortihm we reduce the size of the videoclip by modifiying the start and end points. If the amount that needs to be cut is larger then the size of the videoclip, the whole videoclip is removed from the playlist, and at this point, the total rating of the playlist would decrease. Based on rank model that is used, rating might change even if whole videoclips are not removed. See increased and decreased rank model. We used 100 playlists each containing 10 videoclips in the simulations. We decreased the size of the playlist with one percent per cut, this means that there are 100 data points per graph, since we cut from 100% to 1% of playlist size.

4.3.1 Normal Distribution of Videoclip Length

Fixed Rank

Figure 4.1 shows simulation results where the rank of the videoclips does not change as we cut in the videoclips. In all the figures we have on the y-axis a relative value. The relative value is total rank in playlist divided by uncut total rank in the playlist. This means that higher is better. On the x-axis we have the size of the playlist in percent. We can see here that the 3 different cutting algorithms clearly behave differently. Uniform start & end cut is performing much better then the two other cutting algorithms. This is not unexpected, since cutting uniformly from each of the videoclips will maintain the total rank in the playlist. The only way the total rank in the playlist can become smaller is if a videoclip is cut away totally. Due to normal distribution of videoclip length we can see that it is first around 30% of total size that uniform start & end cut starts to lose total rank. That is because at that point whole videoclips starts to be cut away. It starts to drop significantly around 20%. As expected end cutting of playlist performs the worst, and lowest rank first is between the other two in maintaining total rank.

Increasing Rank

Figure 4.2 shows simulation results where the rank of the videoclip increases as the videoclip is getting smaller. The more we cut in the videoclip the more important the rest of the videoclip is for the playlist. The result that really stands out here is the fact that uniform start & end cut goes above 1 in relative value. The relative value is total rank in playlist divided by uncut total rank in playlist. Since the rank of a videoclip is increased when we cut the videoclip the total rank of a cut playlist can be larger then the uncut total rank of that playlist.

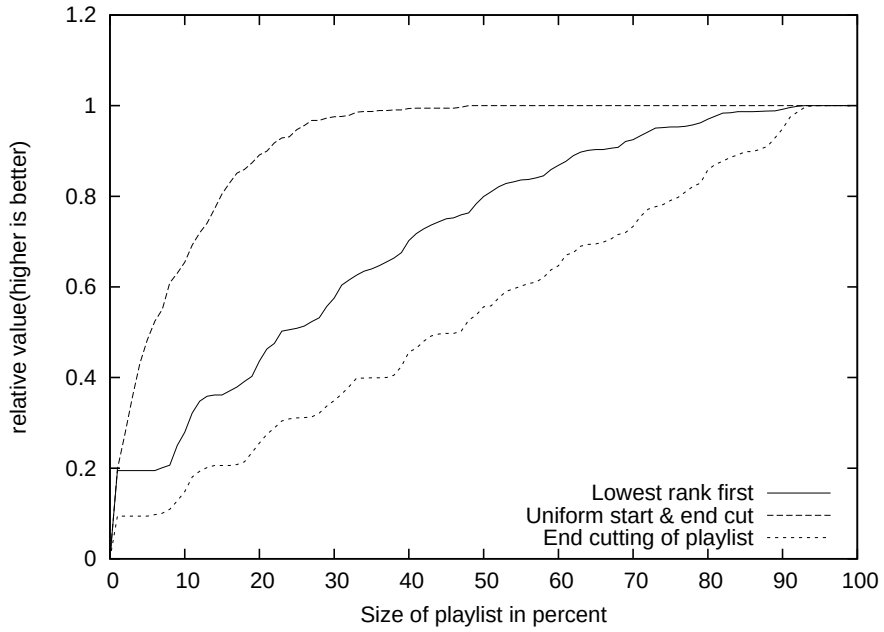


Figure 4.1: This figure contains the three simulation graphs with fixed rank as the videoclips are cut down in size.

Around 20% of total playlist size is the best achieved result for uniform start & end cut. That is the point where most of the videoclips are still in the playlist and they have received increase in their individual ranking because they have been cut down in size. When the playlist is cut further down in size whole videoclips are starting to be removed from the playlist and as such the total ranking starts to decrease fast.

Decreasing Rank

Figure 4.3 shows us the results where the rank of a videoclip is reduced as the we cut away parts of it. This figure does not really show anything new in itself. However we can see that uniform start & end cut is closer to the other two algorithms that in the previous figures, but it still performs much better at maintaining total rank in the playlist.

4.3.2 Exponential Distribution of Videoclip Length

Fixed Rank

Figure 4.4 shows the 3 different cutting algorithms again, but this time there is an expo-variance distribution on length of the videoclips. The most significant change compared to normal distribution of length is that uniform start & end

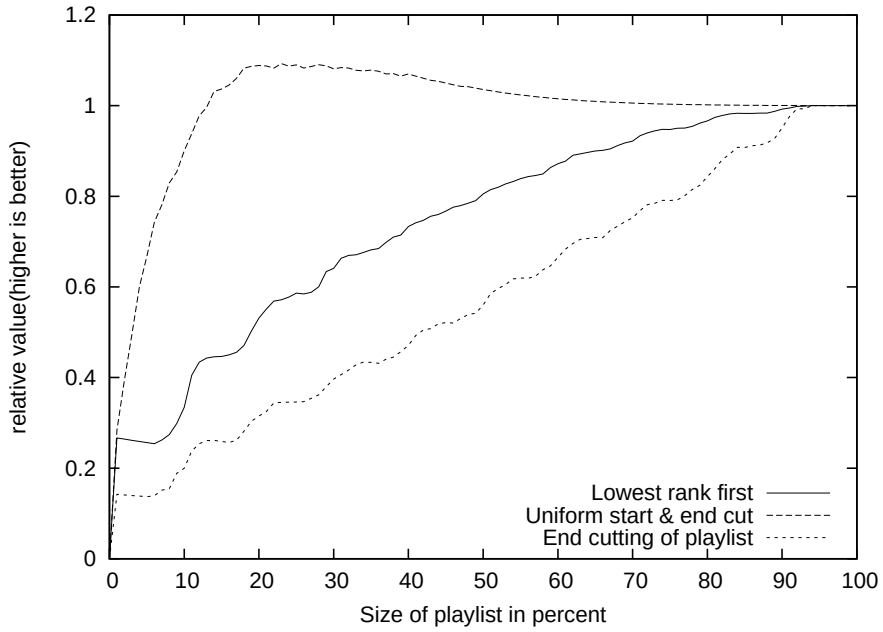


Figure 4.2: This figure contains the three simulation graphs with increasing rank as the videoclips are cut down in size.

cut is no longer the best algorithm, lowest rank first is performing much better. The reason is most likely that the removable of a long low ranking videoclip will happen much more often with exponential distribution of videoclip length and as such lowest rank first works so much better.

Increasing Rank

Figure 4.5 is pretty similar to Figure 4.4, however the graphs in Figure 4.5 manages to maintain slightly higher relative value for smaller percent size. This is of course as expected, since the ranking is increased as videoclips are cut.

Decreasing Rank

Figure 4.6 is also as expected, the relative value for rank is dropping slightly faster, since cutting away at videoclips will decrease the ranking of the videoclip. There isnothing else that really stands out from this simulation result.

4.4 Evaluation of Results from Simulations

Based on the results from the simulations we can say that for playlists with normal distributed videoclip length, uniform start & end cut is the best algorithm.

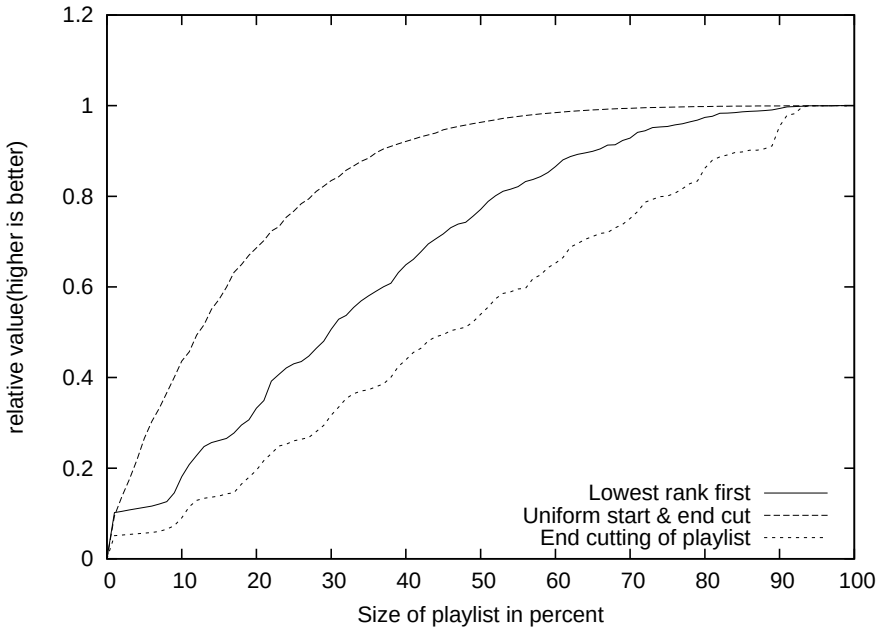


Figure 4.3: This figure contains the three simulation graphs with decreasing rank as the videoclips are cut down in size.

It clearly maintained better total rank in the playlist for all three of the ranking models we used.

For exponential distribution of videoclip length somewhat surprising lowest rank first algorithm was the best. However after careful consideration of how the algorithm worked it turned out to be a fair result. There is a high chance of getting a long low ranking videoclip in the playlist, and this fact will of course make lowest rank first algorithm work much better compared to the other two algorithms. Because it will always start by cutting away the lowest ranking videoclips. Another interesting result for exponential distribution of videoclip length is that ranking model seemed to have very little overall impact on the result graphs.

The simulations we have ran here supports our initial thoughts that different algorithms would perform well under different circumstances. End cutting of playlist, the default solution to the cutting problem was overall the worst algorithm regardless of ranking model and videoclip length distribution. This shows that the problem needs to be looked at carefully and that the default solution does not provide good enough results.

So what we can learn from these simulations. First of all that uniform start & end cut seems to be the way to go for cutting of playlists with normal distributed videoclip lengths. This also means that it fits very well for a Davvi based system.

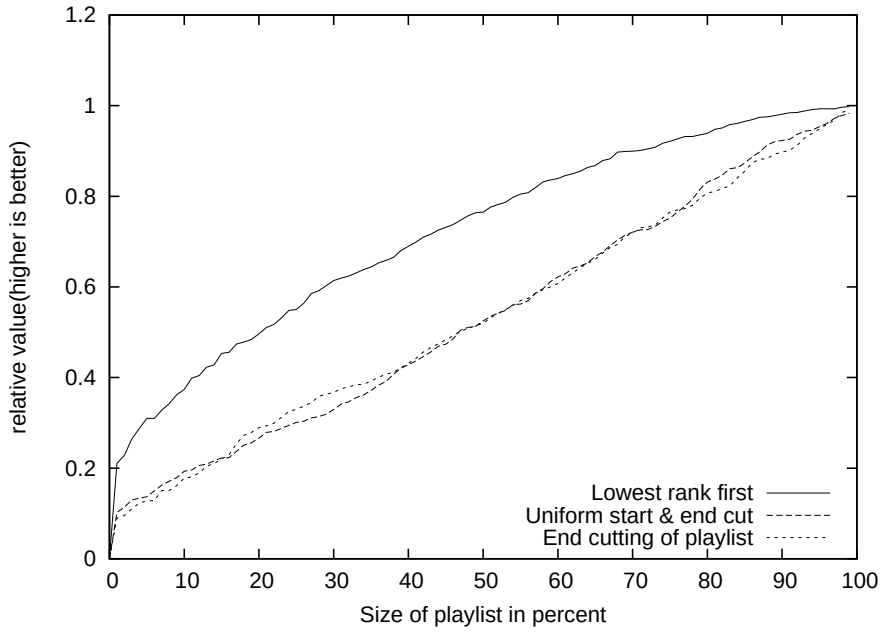


Figure 4.4: This figure contains the three simulation graphs with fixed rank as the videoclips are cut down in size, and exponential distribution on length.

Another important thing to consider is that based on these simulations that say that uniform start & end cut is best, it also means that uniform start cut and uniform end cut will perform as well in the simulations. In reality of course they will perform with difference based on where in the videoclip the important part of the video is, however for the simulations that we have run, all of them are considered equivalent. There is just one rank and how we end up cutting at the start, end or both at the start and the end does not matter for the simulations.

When this is implemented all three uniform cutting models should be available, and hopefully based on either user feedback or domain specific knowledge it is possible to give a hint whether the most important part is in the start, the middle, or the end of the videoclip, and based on the hint use the corresponding cutting algorithm. Of course to be able to do this we would need support from the underlying index. Not only will the video search need to return a playlist with rating but also give a hint per videoclip in the playlist whether the videoclip is front, middle or end important. Ideally we would want a importance graph so that we could do something similar to the Priority Curve Algorithm, but just getting the hint regarding where the important part of the video is, would help alot. Because the current videoclip model in Davvi, that only have a single rank per videoclip, is a bit too simple to be able to perform reasoning based cutting.

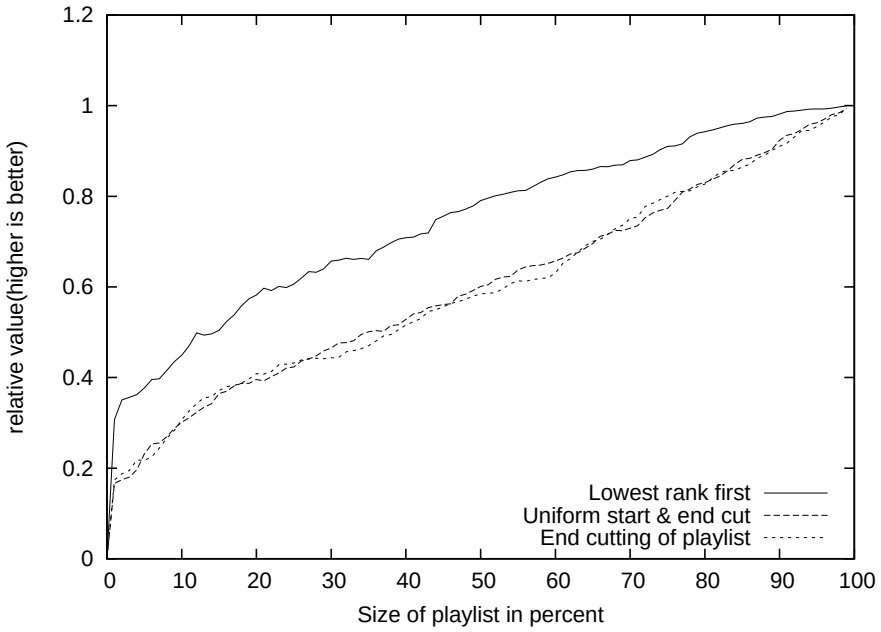


Figure 4.5: This figure contains the three simulation graphs with increasing rank as the videoclips are cut down in size, and exponential distribution on length.

4.5 Summary

In this chapter we have presented three cutting algorithms: end cutting of playlist, uniform start & end cut and lowest rank first. We evaluate the performance of these algorithms under three different ranking models; fixed rank, increasing rank and decreasing rank. We also covered two videoclip length distributions; normal distribution and exponential distribution. Based on this we have ran simulations and presented the results. We learned that uniformed start & end cut was best algorithm for normal distributed videoclip length and that lowest rank first was best algorithm for exponential distributed videoclip length. The ranking models for the videoclips did not influence the results regarding which algorithm was the best. We also saw that the default solution end cutting of playlist was not working well under either of the distribution models and regardless of ranking model it overall came out last among the three algorithms.

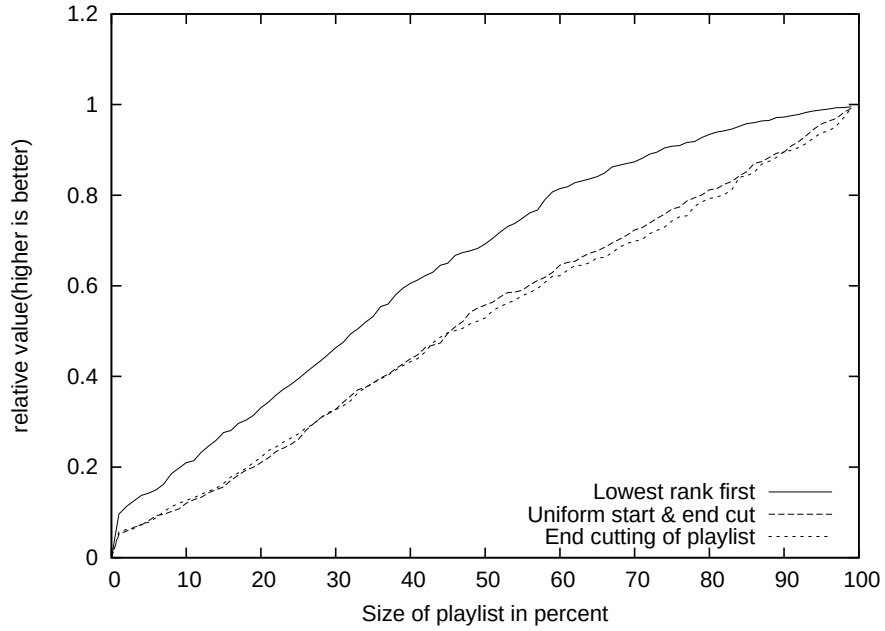


Figure 4.6: This figure contains the 3 simulation graphs with decreasing rank as the videoclips are cut down in size, and exponential distribution on length.

Chapter 5

Hint-Based Cutting

In Chapter 3 we defined and established the optimal videoclip rating model and in Chapter 4 we defined the current videoclip rating model for Davvi based videosearch. Our observation so far is that neither of the models will work well for the playlist reduction that we want to achieve.

Based on the issues found with the PCA and the non existing ratings for topics within videoclips, we realize that we have to go for another rating model. While hopefully the topic rating within videoclips will be available in the future, based on what is available today the PCA algorithm does not really work under current conditions. And the current single ranked videoclip model in Davvi does not support the reasoning based cutting that we want to achieve. Reasoning based cutting is cutting that is done based on a logical reason instead of just cutting away a part the videoclip and hope for the best. The reason might be that we know that the end is more important to the viewers then the start of the videoclip. And because we know this we can make a reason based decision on how we should cut the videoclip.

So we have to come up with a model that is more sofisticated then just a single rank for the entire videoclip, and yet does not need the extensive rating data that is needed for Priority Curves. In this chapter we will present a new videoclip rating model and a cutting algorithm which is based on this new model.

5.1 The Hint-Based Rating Model

The new model is called the hint model. It assumes that all the natural ratings can be classifed as either start, middle or end important, based on where the most significant peak is in the videoclip. In Figure 5.1 we can see examples of three priority curves being mapped over to our standardized model, as either front, middle or end important. So we go from an infinite amount of possible curves to just three curves. This means that we only need to be able to perform three different types of cuts:

- Start from the front of the videoclip and cut towards the end.
- Start cutting from the end toward the start.
- Cut from both the start and the end.

It also means that we will not be able to do an optimal cut all the time, since as we can see in the example in Figure 5.1 the upper middle curve does not really match well with the middle important curve it is mapped to, because the rating drops in the middle. It still matches better then with the other two curves, but is far from ideal.

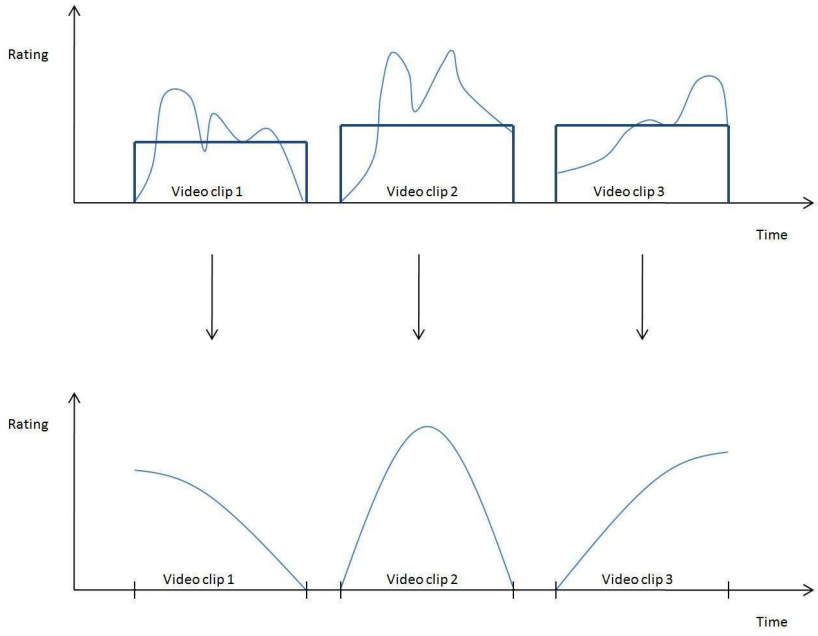


Figure 5.1: Going from a priority curves model to a hint model.

Another reason why we end up going for this model is that the PCA running on priority curves does not give us enough control on the size of the cutdown videoclips. If there are many peaks, little of the videoclip will be removed, while if there is only a few peaks, most of the video clip will be cut. As argued in Section 3.4, our particular solution must be able to cut down a playlist to a more specific size.

In the hint based model it is easier to cut away a certain percentage of the videoclip, and be fairly certain that the most important part is still taken along in the cutdown version. It might not always be the case that the most important

part is taken along, but we are looking for models that will work well most of the time. Since the model that work well all the time, priority curves, did not give us the nessacery size control when cutting with PCA.

When we also consider the amount of work associated with creating priority curves, a simpler model becomes more attractive. Priority curves need rating data that most likley requires human interaction to create in the first place. Davvi gathers human written descriptions of the events in football matches from for instance VG Live. The creation of the original event descriptions are a high cost operation, and a slow process, it happens in real time as the football match is played. Another issue is that priority curves requires all the rating data stored along with the videoclips. For every search term in the videoclip's text description, a priority curve would have to be created for the videoclip. This means that there will be alot of rating data that needs to created, maintained, and processed when a videoclip is cut down in size.

What if there is an interesting part in the start and the end, well then the videoclip should probably be split up into two difference video clips. We will assume that the event detection algorithm has managed to segment the video into videoclips with only one event per video clip.

5.2 Hint-Based Cutting in Davvi

Davvi only have one rank per videoclip, however it also have a type field in the index, and based on this type field it is often possible to decide, by using domain knowledge, whehter or not the videoclip in question is start, middle or end important.

Our cutting algorithm is based on the knowledge and understand we gained from running the naive cutting algorithm simulations and the PCA simulations. It will use the type field in Davvi as a hint on how to perform the cutting of the videoclip.

Our cutting algorithm works like this. Based on domain knowledge we assume it is possible to give a hint to the cutting algorithm, whether the important part is in the start, the middle or the end of the video clip. Based on this knowledge we can use uniform cut (start, end or start & end) to reduce the size of the video clip.

We will see how well our algorithm will manage to maintain the overall rating in the video clips and playlists by comparing it against uniform start& end cut and end cutting of playlist algorithms. These simulations are executed in the same manner as the PCA simulations in Chapter 3.

However since it is not possible to use domain knowledge or predict any hints for random data. We use the peak detection part of the PCA to set the hint for our hint based cutting algorithm in the simulations.

Figure 5.2 shows us that even with totally random data the hint-based cutting algorithm that we have created outperforms uniform start & end cut and end cutting of playlist algorithms. We can see that it is not by a large margin, but it is still a definite result. The reason why the results are so close to each

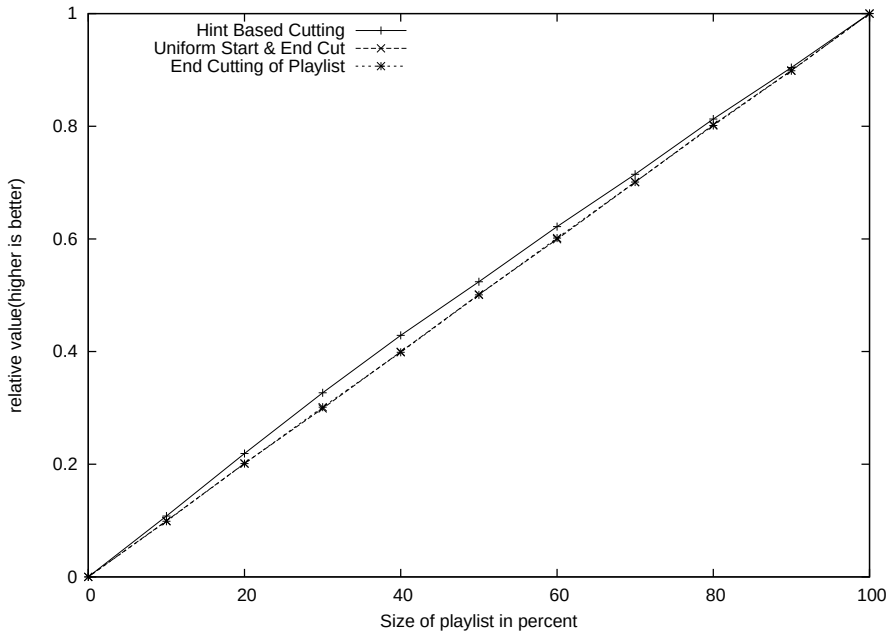


Figure 5.2: Hint based Cutting, UniformStart&End, and End Cutting with Random data.

other for random data is that there are few peaks in the random data and even if there is detected a peak in the random data, there might be another peak or random ratings that is together as large as the peak that was detected by the PCA. The rating is evenly spread across the videclip. So the consequence of not finding the peak is not that big. If the rating data is like this our solution is better, but only by a small margin. However reasoning based cutting in totally random data is a contradiction in itself, because there is no reason or clear patterns in random data.

Figure 5.3 shows us the results for the hint-based cutting algorithm with ideal random data (see Section 3.3 for description of ideal random data). We can see that it outperforms the other two with a large margin. This is of course as expected, since the PCA algorithm manage to find the peak in more or less every video clip, and we use the proper uniform cutting algorithm accordingly to the peak information. This way we maintain the peak in the video clip as long as possible.

A problem with domain knowledge is that we can not expect to be able to correctly set the hint all the time. There might be videoclips that is not following the domain norm. If our solution needs the hint to be correct all the time, then it has a significant weakness. Figure 5.4 shows that even if the hint for the hint-based cutting algorithm is correct only 50% of the time, it still outperforms the other cutting algorithms. Ideal random data was used in this

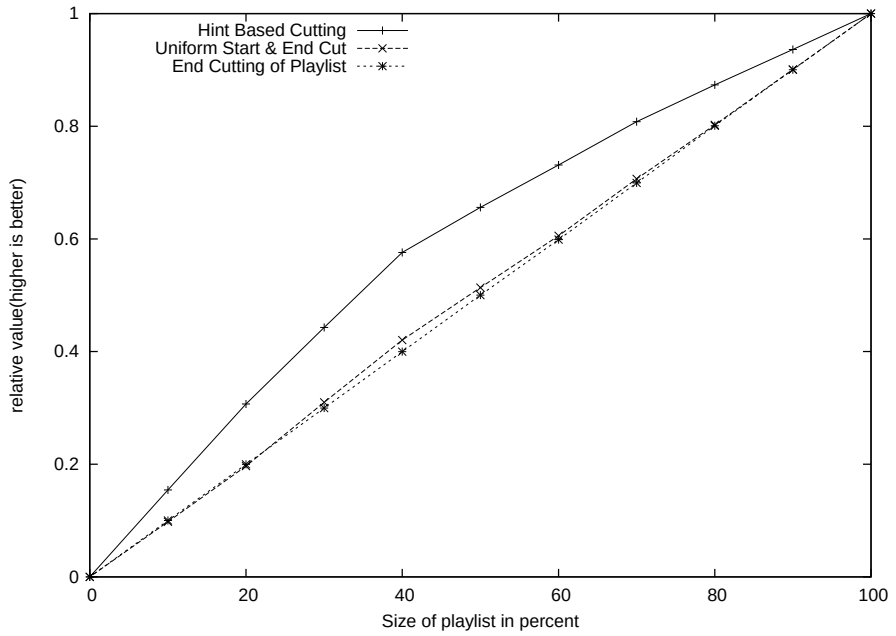


Figure 5.3: Hint based Cutting, UniformStart&End, and End Cutting with ideal random data.

simulation. This gives a clear indication that our solution will work quite well based on hints set by domain knowledge.

One major weakness in PCA is the ability to cut to a user specified size. Lets say we want to remove just 10% from a video clip, it is hard to set r and s parameters of the PCA algorithm so that it will give us the requested size of the video clip. If we do not know how many interesting parts (peaks) there is in the video is it more or less impossible to cut 10% with the PCA. Because of this fact we decided to only use the PCA algorithm to support us in the choice of how you should do a uniform cut. We run the PCA to decide where the most significant peak in the video clip lays, either the start, the middle or the end. Then we choose which cutting algorithm based on this information. UniformCutStart, UniformCutEnd or UniformCutStart&End. While our hint-based cutting algorithm simulations uses the peak detection part of the PCA algorithm, we do not really need this feature, the algorithm does not care how the hints are set, as long as there is a way of setting the hints.

One of the problem with PCA was that it was very hard to have control on how long the summary would become. However we have created a cutting algorithm that can cut a playlist from 100 percent to one percent with a fine granularity, based on block size. So our cutting algorithm would not have this problem even if we have priority curves and uses PCA peak detection to set the hint.

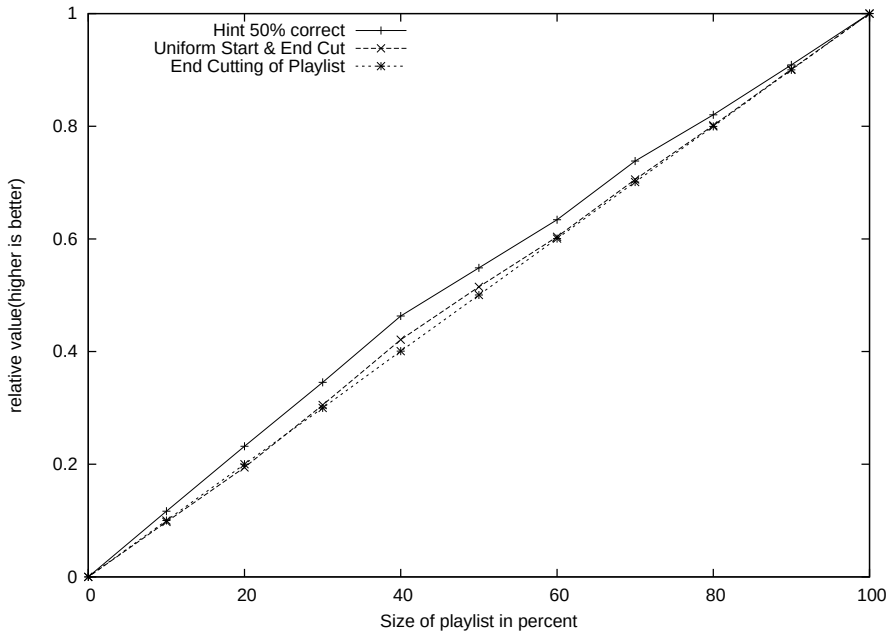


Figure 5.4: Hint based Cutting 50% correct, UniformStart&End, and End Cutting with ideal random data.

5.3 Evaluation

We have with simulations shown that our cutting algorithm based on hint set by the PCA peak detection is better then just doing a uniform cut. Further more, even if we do not have the importance rating for the video clips, we might have domain knowledge that makes it possible to set the hints, that we used the PCA to set in the simulations. An example, a goal in a soccer match would typically be end important, it is nice to see the foreplay to the goal, but the most important part of the goal is the few seconds where the football crosses the goal line. So when the time constraint gets tighter, more and more of the foreplay to the goal will be removed.

We have checked for two sets of rating data, ideal data random, and total random data. It is more likely that real rating data would place itself somewhere between the two. However we are fairly certain that rating data would be closer to the ideal data that to totally random data. The reason is that people would typically consider a certain point in the video clip as the most important, etc a goal, and the foreplay leading to the goal as almost as important as the goal. This example shows that a totally random rank data model does not fit reality very well. Just consider a situation where the rating jumps up and down for each block, this does not make sense. The rating might be high some block, then low some blocks, etc, but not jumping around randomly.

Still with random data, our algorithm is better then the other two cutting algorithms. We think it is very likely that real rating data would land closer to our ideal data, then totally random data.

Another important aspect of the hint model and our hint based cutting algorithm is the fact that the hint which is set based on domain knowledge does not need to be correct all the time. The higher correctness percentage on the hints the better our algorithm will perform in maintaining the rating of the videoclips. Still we have shown in Figure 5.4 that we perform better then the other cutting algorithms even if we only have the hints correct 50% of the time.

Our hint model uses 3 different importance curves. However, it is not a problem extending this model to for instance 5 curves. See Figure 5.5 for an example of a possible extended hint model. An extended hint model is a model that has more curves then the three we defined. This would be a natural adaption of the model, if there is lets say 5 different typical importance curves for the videoclips. However, if we extend it, we must be able to set the hints accordingly based on domain knowledge. If we can manage to set the hints correcly the extended hint model will of course beat our current model that only uses 3 different curves. The reason why we did not go for this extended model is that we think for a Davvi based system in the soccer domain, vidoeclips fits the front, middle or end important curves very well.

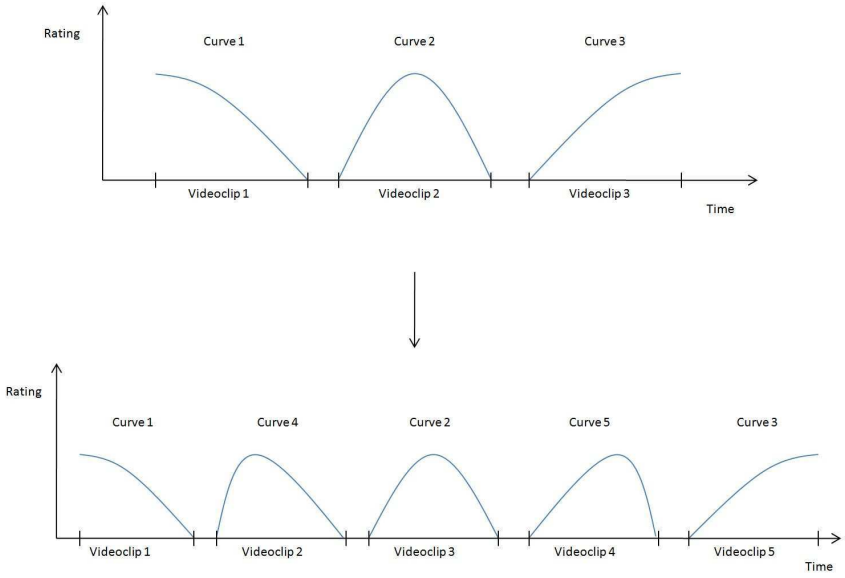


Figure 5.5: This figure shows an extended hint model, that goes from 3 to 5 types of importance curves.

The only thing we are not certain about is if we should use the curves 1, 2, and 3 or if the curves 4, 2 and 5 (see Figure 5.5) would give a better view of how the importance rating should be in a 3-curve hint model. The question is whether the most important part of the videoclip is entirely at the start or slighly after the start of the videoclip, the choice is between curve 1 or curve 4. And the same issue occurs at the end. If the most important part is entirely at end or is it slightly before the end, the choice is between curve 3 and 5.

As far as implementation goes, it is slightly more complex to support curves 4 and 5 compared to curves 1 and 3. With curve 1 and 3 we just cut by removing blocks from the side that had a low rating. If we were to implement cutting based on curve 4 and 5, it would be possible to cut like this, for each time we cut, we remove one block at the start and two or three blocks at the end for curve 4, and do the opposite for curve 5.

With simulations there is always the problem that we write the code that generate the data which we presents. So we need to be objective in how to write the code, so we do not write simulation code that slightly favors what we want to achieve. A part of the reason for the different models in Chapter 4, were to see if the different cutting algorithms performed differently under other settings. We acknowledge that there are probably more models or situations that should have been considered for completeness. However, we had to limit the simulations some what. We also acknowledge that we did not implement the entire PCA algorithm, we focused on the peak detection and the block elimination parts of the algorithm. However, based on how we wrote the simulations, the other parts of the PCA would not have affect the result, the way we see it.

A large potential source of error is the fact that we had to generate the rating data for the peak detection algorithm in particular. Still the simulations shows that given a certain type of data, that we believe is likely, our cutting algorithm works better then the others.

5.4 Summary

In this chapter we have presented the hint model and the hint-based cutting algorithm. We have shown that our hint-based cutting algorithm outperforms uniform start & end cut, and end cutting of playlist. We have also shown that our hint based solution does not need to have the hint set correct all the time to outperform the other cutting algorithms.

Chapter 6

Case Study

The hint based cutting algorithm, which we presented in Chapter 6 was designed for a Davvi based system. Davvi has a video storage solution where videoclips are divided into two second segments. Due to that design choice, it will be easy to fit our cutting algorithm into that system. To get a new cut of the videoclip we only have to modify the list of the segments that belongs to the videoclip. However, now we are going to look at current streaming based systems, to see how our cutting algorithm easily can be adopted on top of current streaming solutions. An assumption in this case study is that the existing streaming system have a type field similar to Davvi, so it is possible to set the hint based on the type field.

6.1 System Overview

We will look at a SMIL based streaming system. In Figure 6.1 we can see the system architecture, that we will use in this case study. We can see how the client-server communication is in a simple streaming system. The client requests a metadata file from the server. (1) The client receives the metadata file from server. (2) Based on the metadata file the client requests the streaming to start (3). The client receive streaming data from the video storage on the server. (4)

6.1.1 Client Side

What we need to add is the ability to reduce the playlist on the client side. The clients graphical user interface (GUI) will at the top have a text box for input of search terms and an OK button that will send the search term(s) to the server. The server will return a playlist in the form of a SMIL metadata file. The GUI will make the time information available for the client (see Figure 6.2). Underneath the input text box there will be a slider that the client can move left and right to decide how long the cut down version of the playlist shall

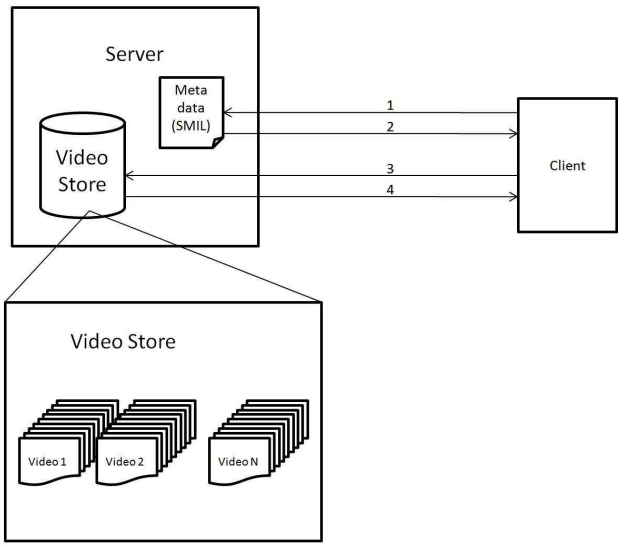


Figure 6.1: System architecture.

be. Below the silder the original playlist returned from the server will be shown. The total length of the playlist will be shown in the duration field to the right of the slider, and the slider will be positioned to the right side, since it will not be possible to ask for a longer version of the playlist then the sum of the duration of the video clips in the playlist.

6.1.2 Server Side

In SMIL based streaming systems we would on the client side send the amount that needs to be cut to the server side. On the server side we would run code similar to this:

```
def hintBasedCutting(playlist,amount):
    elements=len(playlist)
    perVideo=float(amount/float(elements))
    res=[]
    for p in playlist:
        if p[1]==0:
            res.append(cutStart(p,perVideo))
        elif p[1]==1:
            res.append(cutEven(p,perVideo))
        else:
            res.append(cutEnd(p,perVideo))
```

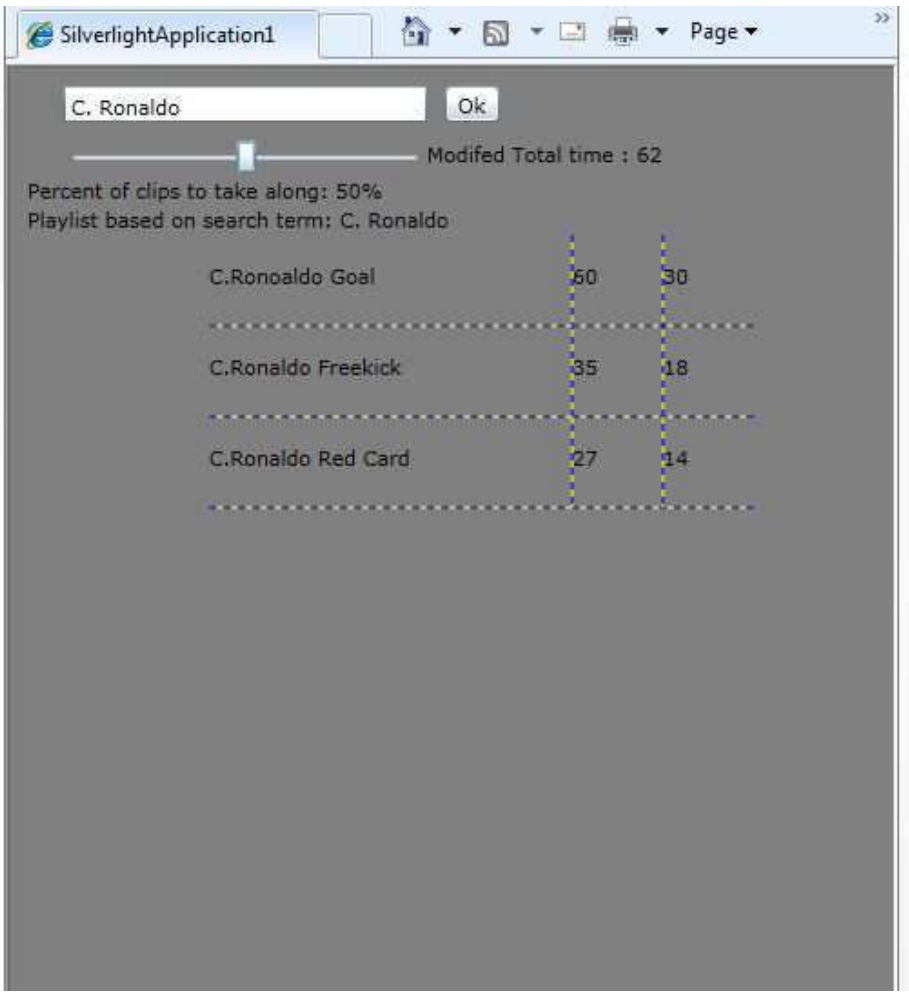


Figure 6.2: Early example of the GUI.

```
    res.append("\t\t</seq>\n")
    return res


def cutStart(data,perVideo):
    src=data[2]
    tmp="\t\t\t<"+src[0:len(src)-1]
    tmp+=" clipBegin=\""+str(perVideo)
    tmp+="s\" clipEnd=\""+str(data[0])+"s\" />\n"
```

```
    return tmp

def cutEven(data,perVideo):
    start=perVideo/2.0
    end=perVideo/2.0
    src=data[2]
    tmp="\t\t\t<"+src[0:len(src)-1]
    tmp+=" clipBegin=\""+str(start)
    tmp+="s\" clipEnd=\""+str(data[0]-end)+"s\" />\n"
    return tmp

def cutEnd(data,perVideo):
    src=data[2]
    tmp="\t\t\t<"+src[0:len(src)-1]
    tmp+=" clipBegin=\""+str(0)
    tmp+="s\" clipEnd=\""+str(data[0]-perVideo)+"s\" />\n"
    return tmp
```

The code modifies the server side metadata files. After the server side script has been run, it is possible to use the default communication pattern between client and server as described above. The new SMIL playlist would be used instead of the original to enable streaming of the reduced playlist. Full source code is on the CD-Rom. However the rest of the code handles reading of files and XML parsing so it is not that interesting.

We have implemented a python script that can reduce the length of a SMIL playlist. The python script takes as input a SMIL playlist containing resource links to Real Video (RV) formatted video clips and a cut duration. The python script then generates a new SMIL playlist file where we have divided the amount we needed to cut to each video. This solution uses our hint-based cutting algorithm. There are 3 different cut modes: cutStart, cutEnd and CutEven. They cut at the start, at the end and cutEven cuts both at the start and the end. Cutting in this setting means adding clipbegin and clipend and calculating new start and end times for the videoclips.

Figure 6.3 shows us the original SMIL metadata file, we had to slightly modifiy it, we have added two lines of comments per resouce link, it is length and hint. The python script will read the comments and based on the length and hint information run the hint based cutting algorithm. With hint set to zero, it means that the clip should be cut in the beginning. Hint set to one, means that the clip should be cut from both sides, and finally hint set to two means that the clip should be cut from the end.

To convert the original SMIL metadata files to the the format that our hint-based cutting algorithm expects, we would have to run a script that adds the hint and length information to the already existing SMIL metadata files on the servers. The length information is not a problem, it should be easy enought to extract from the video files or it might already be in the metadata files. The

```
<smil xmlns="http://www.w3.org/2001/SMIL20/Language"
xmlns:rn="http://features.real.com/2001/SMIL20/Extensions">
        <head>
                <meta name="title" content="Video Playlist"/>
                <meta name="author" content="Webjorn Larsen"/>
                <layout>
                        <region id="video_region"/>
                </layout>
        </head>
        <body>
                <seq>
                        <!-- length:63s-->
                        <!-- hint:0-->
                        <video src="File:///C:/Users/webjorn/Desktop/CoD.rv" region=
                        <!-- length:63s-->
                        <!-- hint:1-->
                        <video src="File:///C:/Users/webjorn/Desktop/IronMan.rv" re
                        <!-- length:130s-->
                        <!-- hint:2-->
                        <video src="File:///C:/Users/webjorn/Desktop/Hulk.rv" regio
                </seq>
        </body>
</smil>
```

Figure 6.3: SMIL playlist before cutting algorithm has been run.

hint information can be set based on the type field. However, lets look at other possible ways, it is possible to set the hint either with manual annotation in smaller systems. Another solution would be to allow the users to give feedback on the videoclips, whether they are front, middle, or end important.

```
top/CoD.rv" region="video_region" regPoint="center" regAlign="center" clipBegin="40.0s" clipEnd="63s" />
top/IronMan.rv" region="video_region" regPoint="center" regAlign="center" clipBegin="20.0s" clipEnd="43.0s"
top/Hulk.rv" region="video_region" regPoint="center" regAlign="center" clipBegin="0s" clipEnd="90.0s" />
```

Figure 6.4: Clipbegin and Clipend added in SMIL playlist after cutting algorithm has been run.

Figure 6.4 shows us the a part of the SMIL metadata after the cutting algorithm has been ran. We can see in the marked area that clipbegin and clipend have been added. Based on this information the SMIL compatible systems like Real Player [13] knows that it should start to play from the clipbegin and stop at clipend. If this SMIL playlist was on a server with the videoclips setup for streaming, for instance over the Real Time Streaming Protocol (RTSP) [18]. We would stream down the new cut of the playlist, that was created by our python script. In the streaming setting the client would only receive the data for the parts of the videoclips that was specified in the SMIL metadata file on the server.

6.2 Findings of the Case Study

This case study shows how easy our hint-based cutting algorithm can be put to use also in other settings then Davvi based systems. We managed to adapt it to a SMIL based streaming system by just writing around 100 lines of code. This shows that our solution has a general applicability.

To test and verify the Python script we played the SMIL playlists in Real Player. The output playlist from our Python script would play a reduced version of the original playlist. This shows an easy way to automatically cut video playlists. This made visual tests of the cutting algorithm possible.

There needs to be a minimum size on the videoclips in the system. We came to this conclusion when we were testing our cutting algorithm in the SMIL-based prototype cutting for RealPlayer. We have not found a single limit, because it might be individual for each person and it might also be different between domains. At a certain point a videoclip will be cut down to a duration that is so small that it will no longer give the intended meaning to the viewer. When this happens the rating for the videoclip will be zero. This means that there is a size threshold, and when the uniform cutting across the videoclips have cut a videoclip down to the threshold, the videoclip needs to be excluded from further cutting. In the end all the videoclips in the playlist will be cut down to the size threshold and we must instead start to remove whole videoclips, if we have to further reduce the duration of the playlist.

6.3 Summary

In this chapter we looked at how we can use our hint based cutting algorithm on current streaming systems. We decided to look at the much used SMIL format. It is a common metadata format for streaming. We have shown that our algorithm does not need a huge rewrite of current systems to be used. It only requires a simple client side GUI and a few lines of code on the server side to rewrite the metadata file according to the new time constraint. Then the normal client-server communication pattern can be used on the new SMIL metadata file instead.

Chapter 7

Concluding Remarks

In this thesis we have adressed the problem of watching vidoes when the user has only a limited time available. The goal of this thesis is to come up with a cutting algorithm that is better then the default algorithm end cutting of playlist. The motivation for this was that we saw the need for the ability to reduce the size of video playlists to fit within a user defined time constraint and reduce the size of the videoclips for transfer to mobile devices. The cutting algorithm should be possible to implement in a Davvi based system, see Section 1.1 for a description of the Davvi system.

We started out by defining the natural rating model for videoclips in Section 3.1. We called it the priority curve model. Having found the Priority Curve Algorithm when looking at video summarization in Chapter 2. This algorithm was a natural candidate to use the Priority Curve model, being designed for a similar kind of model. We ran a series of simulations to see the behavior of this algorithm and concluded that it was not suited to cut videoclips to a specific size in a Davvi system. In particular, Davvi does not have the needed rating data for the videoclips to support the creation of priority curves, which the PCA algorithm needs.

In Chapter 4 we therefore describe and define how cutting can be done in Davvi's single rank per videoclip model. In Section 4.1 we define the models that were missing in the Davvi paper to be able to run simulations to check the behavior of three naive cutting algorithms: end cutting of playlist, uniform start & end cut , and lowest rank first. In Section 4.2 we present the simulation results by showing graphs of how the total rating in the playlists develops as the playlists are cut in size. We showed that, uniform start & end cut is best among the three cutting algorithms for playlists with normal distributed videoclip length. For playlists with expo-variance videoclip length distribution it was lowest rank first that was the best of the three algorithms.

The hint model and the hint based cutting algorithm is presented in Chapter 5. The reason for coming up with this new model was that the priority curves was not possible to support in a Davvi based system. The three naive cutting algorithm that was possible under Davvi's single ranked model was too simple.

It was not possible to do reasoning based cutting decisions. So in Section 5.1 we define the hint model. In the hint model, every priority curve can be mapped over to one out of three possible curves: front, middle or end important. The mapping is based on where in the priority curve the highest peak is found. This means that we have reduced the number of possible rating curves from infinite to three curves. The reason we call it the hint model is that in Davvi we are going to set the hint of how to perform the cutting based on domain knowledge. This can be done because the Davvi search index have a type field.

In Section 5.2 we present simulation results that shows that our hint based cutting algorithm is better then uniform start & end cut and end cutting of playlist algorithm. We also show that the hint does not need to be correct all the time for the hint based model to be better then the naive cutting algorithms. If the hint is correct 50% of the time, it is enough to outperform the naive cutting algorithms.

In Chapter 6 we have a case study of how we can use the hint based cutting algorithm in current streaming systems. This shows that our algorithm is not only possible to implement in a Davvi based system, but also have applicability in a more general setting.

7.1 Conclusion

In this thesis we show that it is possible to cut down the length of a playlist automatically and in a better manner then the default solution end cutting of playlist. We have further defined and described how cutting of playlists can be done in a Davvi based system. By covering three naive cutting algorithms for Davvi under different ranking models and distributions of videoclip length. We have also shown that the ideal model, priority curves at this time is not a good solution, because we do not have the rating data available to make Priority Curves for every videoclip. We also show that PCA cannot in general cut a playlist reliably down to an arbitrarily size.

We have based on the research done in this thesis designed a new cutting algorithm call hint-based cutting, for a Davvi based system that is better at maintaining total rating in the videoclip then any of the naive cutting algorithms.

If techniques to automate the creation of priority curves becomes available in the future our hint based cutting algorithm can use this rating data to better calculate whether the videoclip is front, middle, or end important. So it is capable of not just handling todays situation but can also put to use better rating models that might be realistic in the future. However, at this point in time, we at best have domain knowledge that might give us a hint as to whether the videoclip is front, middle or end important.

Our cutting algorithm hint based cutting does not care how the hints are set, we used PCA's peak detection in our simulations, but if it is set based on domain knowledge, we can still run the uniform cutting algorithm based on this hint information.

This means that our solution covers all the scenarios. If we have priority curves available, we can use peak detection to find out if the most significant part of the video is at the front, middle or end of the videoclip. If we have domain knowledge, we can still set the hints based on that, and even if we do not have anything except one rating per videoclip, our simulations for uniform start & end cutting have shown that it is the best choice amongst the naive cutting algorithms we have considered for normal distributed videoclip length. This means that our solution is performing very well in the majority of the scenarios we have considered.

7.2 Future Work

To gain even better understanding of how our hint-based cutting algorithm would behave in a real-life deployment scenario, the next step would be to conduct a user survey. Such an user survey should let the users decide whether they liked best the reduced playlists generated by the hint based cuttting algorithm or end cutting of playlist algorithm.

Looking into how accurate we can set hints based on domain knowledge for a Davvi based system would also be interesting. Since the hint based cutting algorithm is somewhat dependent on the ability to set correct hints.

One of the problem when streaming video to mobile phones is the data transfer cost. The data-plans offered by a typical mobile subscription are not very good at the moment. It will either cost a lot of money or the speed will be decreased because a download limit has been reached. One of the reasons for these subscriptions are the capacity in the mobile networks. The bandwidth in an area is shared between all mobile users connected to the same cell tower. So a pricing scheme that makes people chose to only use their data-plan when they really need it is beneficial to maintain capacity in the mobile network. This means that we do not want to download or stream video data we are not going to watch. Based on our work it is possible to implement systems that will more efficently use the bandwidth and download limit in the subscriptions, because it is possible to cut the video prior to starting the download or the streaming. It would be interesting to explore how our approach can be used in this context.

Bibliography

[1] Nicola Adami, Sergio Benini, and Riccardo Leonardi. An overview of video shot clustering and summarization techniques for mobile applications. In *MobiMedia '06: Proceedings of the 2nd international conference on Mobile multimedia communications*, pages 1–6, New York, NY, USA, 2006. ACM.

[2] Rene Beier and Berthold Vöcking. Random knapsack in expected polynomial time. In *STOC '03: Proceedings of the thirty-fifth annual ACM symposium on Theory of computing*, pages 232–241, New York, NY, USA, 2003. ACM.

[3] Hans-Tore Bjerkaas. Nrk nett tv. Website. `http://www1.nrk.no/nett-tv`.

[4] A. Amir D. Ponceleon C. M. Taskiran, Z. Pizlo and E. J. Delp. Automated video program summarization using speech transcripts. *IEEE Trans. Multimedia*, 8(4):775–791, 2006.

[5] David L. Calhoun and Susan D. Whiting. Television, internet and mobile usage in the U.S. A2/M2 Three Screen Report 1Q 2009, The Nielsen Company, Online Publication at `http://blog.nielsen.com/nielsenwire/wp-content/uploads/2009/05/nielsen_threescreenreport_q109.pdf`, May 2009.

[6] D. E. Comer, David Gries, Michael C. Mulder, Allen Tucker, A. Joe Turner, and Paul R. Young. Computing as a discipline. *Commun. ACM*, 32(1):9–23, 1989.

[7] Microsoft Corporation. Smooth streaming: The official microsoft iss site. Software. `http://www.iis.net/download/smoothstreaming`.

[8] Thierry Michel et al. Synchronized multimedia integration language. Website. `http://www.w3.org/TR/SMIL3/`.

[9] M. Fayzullin, V. S. Subrahmanian, M. Albanese, and A. Picariello. The priority curve algorithm for video summarization. In *MMDB '04: Proceedings of the 2nd ACM international workshop on Multimedia databases*, pages 28–35, New York, NY, USA, 2004. ACM.

[10] A. Hanjalic and H. Zhang. An integrated scheme for automated video abstraction based on unsuperivsed cluster-validity analysis. *IEEE Trans. Circuits Syst. Video Technol*, 9(8):1280–1289, 1999.

[11] Apple Inc. Http live streaming overview. Website. `http://developer.apple.com/iphone/library/documentation/NetworkingInternet/Conceptual/StreamingMediaGuide/Introduction/Introduction.html`,.

[12] Move Networks Inc. Move networks. Website. `http://www.movenetworks.com/`.

[13] RealNetworks Inc. Real player version 1.1. Software. `http://eu.real.com/realplayer`.

[14] Dag Johansen, Håvard Johansen, Tjalve Aarflot, Joseph Hurley, Åge Kvalnes, Cathal Gurrin, Sorin Zav, Bjørn Olstad, Erik Aaberg, Tore Endestad, Haakon Riiser, Carsten Griwidz, and Pål Halvorsen. Davvi: a prototype for the next generation multimedia entertainment platform. In *MM '09: Proceedings of the seventeen ACM international conference on Multimedia*, pages 989–990, New York, NY, USA, 2009. ACM.

[15] YouTube LLC. Youtube - broadcast yourself. Website. `http://www.youtube.com`.

[16] Arthur G. Money and Harry Agius. Video summarisation: A conceptual framework and survey of the state of the art. *J. Vis. Comun. Image Represent.*, 19(2):121–143, 2008.

[17] Haakon Riiser, Pål Halvorsen, Carsten Griwodz, and Dag Johansen. Low overhead container format for adaptive streaming. In *MMSys '10: Proceedings of the first annual ACM SIGMM conference on Multimedia systems*, pages 193–198, New York, NY, USA, 2010. ACM.

[18] H. Schulzrinne, A. Rao, and R. Lanphier. Real time streaming protocol (rtsp), 1998.

[19] Howard D. Wactlar, Takeo Kanade, Michael A. Smith, and Scott M. Stevens. Intelligent access to digital video: Informedia project. *Computer*, 29(5):46–52, 1996.

[20] Xin Liu Yihong Gong. Video summarization using singular value decomposition. In *Computer Vision and Pattern Recongnition, 2000 Proceedings. IEEE Conference on*, volume 2, pages 174–180. IEEE, 2000.

[21] Jian Zhou and Xiao-Ping Zhang. A web-enabled video indexing system. In *MIR '04: Proceedings of the 6th ACM SIGMM international workshop on Multimedia information retrieval*, pages 307–314, New York, NY, USA, 2004. ACM.