Browsing news video using semantic threads

M.Sc. Thesis of

Ork de Rooij

23-12-2005

Abstract

This paper describes a novel approach for finding threads in video material using basic clustering techniques by combining knowledge from the content-based retrieval in video material domain and the topic detection and tracking domain. For this the notion of the semantic thread as an ordered list of video shots about the same semantic subject is proposed. A method for generating semantic threads from a large collection of video material is presented. Several standard algorithms for creating clusters are compared and a method for including both clusters and time to create threads is discussed. With these threads an interface for searching through a large dataset of video material is proposed and implemented. This interface is then evaluated with the TRECVID interactive retrieval task, where it ranked among the best interactive retrieval systems currently available. The interface proved to be very usefull for finding video material where the topic cannot be easily found by using traditional keyword search.

Contents

1 Introduction

Nowadays there is an ever increasing amount of video material available about various subjects. Due to advances in digital storage capacity most of this material is digitally stored. Thanks to this digital storing and the internet, video fragments can be quickly retrieved if you have selected the item that you want to view. But how do you find what you want to view?

Let's suppose we want to find all video fragments related to a topic. We would need to scan the total collection of video fragments, selecting those that match the topic. If we do this sequentially watching everything, this would take as long as the duration of all video fragments together. For instance: if we limit our search to recordings of all eight o clock news bulletins from one news channel for one year, and each news bulletin has an average duration of 30 minutes, we still have 182 hours of video to search through. Because we only watched the news bulletin of one specific channel, we have only seen information summarized based on their judgement of what we wanted to see. If we want to have more in-depth results for the topic we need to search though all recordings of that channel. This would yield 48 times as much recorded video: 8760 hours of recorded material. To search for something that happened last year you would need one man-year to search for it. That is still from only one news channel, for a reliable result more channels are needed.

The problem of finding a topic in a large set of data does not only exist in the video modality. The same holds for the audio and text modalities. In the text domain the problem of finding topics is generally easier to solve. This is because keywords can be derived from the text which have a clear meaning to a user. These keywords can then be used for searching through the dataset, and many techniques exist to enhance keyword based search. Instead of using the keywords only for searching they can also be used to cluster segments of text, thereby creating groups of same-topic text fragments. These groups depict a story about a certain topic in the text domain. This is known as topic detection [1]. A method which extracts and groups topic-related stories together in the video modality would be of great help for solving the problem of finding video about a topic. From having to look through everything we now only have to look at entire topic-related sequence and choose the sequence which represents the topic of what you want to find. And if each each sequence can be textually labelled with its contents then the correct sequence could be found by searching for labels. With all this combined it would be possible to quickly find whatever you are looking for. To do all this we need to extend the techniques of topic detection to the video modality to create the sequences, and develop a browsing interface to help the user to find the right sequence from within the entire news video collection.

The organization of this paper is as follows. A method for using topic detection together with video is given in chapter 2, followed by the primary research questions this paper tries to answer in chapter 3. Related work for these questions is given in chapter 4. An overview of possible techniques and methods is given in chapter 5 and an implementation is discussed in chapter 6. Evaluation and results of this implementation is discussed in chapter 7, followed by a conclusion in chapter 8.

2 Combining video analysis with topic detection

To be able to search through video we must first define what we mean with video, and how stories are told using video.

2.1 What is a video collection?

A video document isn't created by filming an entire event in one go and then releasing the entire film footage as the document. It is created by filming the event with possibly multiple cameras in multiple takes. The author of the video then combines, or directs, these different fragments of camera footage together to create the video. Because of this, the video can also be decomposed again into these fragments. If these fragments start and end at video cuts, the deliberate switching of one camera position to another, they are called shots.

How then is a story told to the viewer in the video modality? A story in video is conveyed by a sequence of shots. Each shot describes some series of semantic concepts to the viewer. The whole sequence of shots conveys the intended story. Each shot consists of a number of elements, or semantic concepts, that the director of the story wants conveyed to the viewer. If we somehow can capture what these semantic concepts are for each shot we can re-tell the story without showing the actual video. This is good, since this will allow us to search for a combination of semantic concepts to find a certain video instead of having to see all available video.

A video as a whole is however a difficult thing to handle. Even when video is split into shots, each shot is still one entity, with no other relation to other shots than the fact that it came from the originating video at a certain position. We need to have method of describing the contents of a shot. But what is a description of a shot? There is a lot of information contained in a shot, more than what we would consciously think about. For example: if we have a fragment of news video (figure 1) where the weatherman tells the audience about the weather of today, we might look at it just like that: a weather forecast. However for a computer it is not that simple. Let's look at all the possible things a computer might be able to extract. On screen there is a big high contrast graphical map visible, with lots of highly saturated colors. A computer might have a rule that says: if you see high contrast highly saturated colors with distinct regions, then this might be an artificially created image, such as a weather map. Also there is some text visible, which might be recognized by OCR software, which would lead to keyword searchable text. Also there is an auditory track accompanying the shot, which might be recognized by the computer and translated into, again, text. If we have good person recognition software it might just be possible that we can extract that the person is Glenn Schwartz, a weather forecaster for NBC news. Besides all this there is also meta information available which could also be of use to describe the shot, such as the channel, the language and the time and date on which this shot was aired. All these different extracted features of a shot together represent the description of a shot.

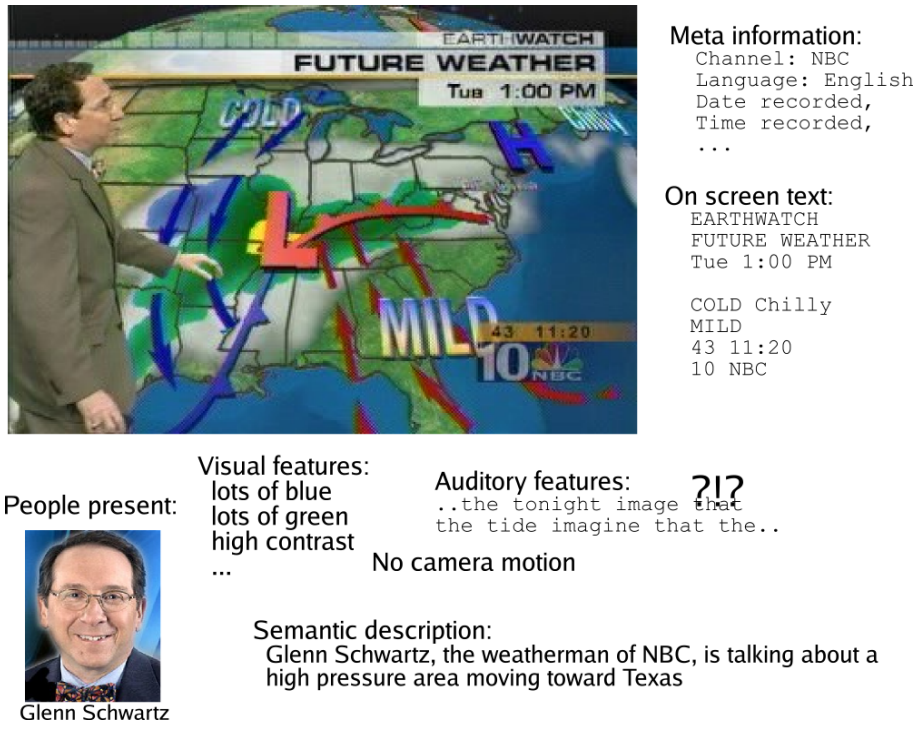


Figure 1: Example of how a shot can contain many features in different modalities

Now if we look at the description problem again we could say that the following categories of features can be distinguished:

- *visual features*: most common colors present, repeating textures, brightness, amount of change, graininess, ...
- *camera motion features*: panning, zooming, fading, ...
- *textual features*: text visible on screen, text in closed captioning
- *meta information features*: features extracted from the context of the shot, such as: air time, air date, channel, program, language, ...
- *auditory features*: loudness, amount of change in volume, voices present, music present, ...
- *semantic features*: who is present on the shot, what is happening, who is talking

The first five categories can, with varying degrees of success, currently be detected by a computer, but they only describe the appearance of the film. What a shot means to a viewer is not described by any one of these features, but by the semantic features. This problem of basic low-level features not describing the actual high-level abstraction of meaning is called the semantic gap [2].

The last category in the above list describes the other side of the semantic gap. It describes what happens in a shot in a high-level abstract way. Unfortunately this is also currently the most difficult to algorithmically determine. It is currently almost impossible for a computer to describe the semantic events of a shots just by looking at the shot itself.

However, what a computer can do is determine if a certain pre-described element exists in a shot, which could be done by writing specialized detector software which tries to find the specified element using only low level features from the video fragment. For example: to detect a tennis match in video material a detector can find certain low-level characteristics in video material which, combined, indicate that video fragments of a tennis match are being shown. For a tennis match these characteristics could be: a mostly greenish or brownish screen because that is the color of the tennis field, and mostly wide angle shots showing a lot of field. Numbers on the screen represent the current score. Names on screen indicate the players. A mostly silent audio track, with pops when the ball is being smashed, and cheering when a point has been made. A single loud voice of the judge and often a narrative style of voice from the commentator. All these characteristics when detected together in a sufficient amount increase the chance that a tennis match is visible.

A single detected concept however does not explain the contents of a shot. By combining all the outputs of a series of ”semantic concept detectors” we can create a concept vector, or feature vector, which describes the semantic content of a shot as a series of numbers. In this representation we have lost almost everything about style, visual layout and other basic features, but we should still have some form of the intended meaning, or story, of the shot. We have also reduced the dimensionality of one shot from several billion dimensions, ie. every pixel in every frame and every audio fragment of one shot, into just several hundreds. This makes comparing shots much easier, and any comparisons made will be based upon the story-content of a shot, not just on visual similarity. This is good.

Now that we can describe the shot as a semantic concept vector, we can ask ourselves how many concepts we need to accurately describe a shot in a story? This is actually more difficult. The first answer would be: as many as possible. But this poses a problem in terms of computational complexity: all available detectors will have to determine the presence of their semantic concept in every shot. However, once run, the comparisons needed to determine if shots are related can be done much faster because of the dimension reduction. This is an advantage especially when new video material is added later on to the system. The new video material only needs to be analyzed once and it can then be compared to the entire dataset of existing materal.

Another problem is that a large number of concepts will basically result in a lot of the concepts being zero; or not present in the shot. For example: in a shot of an important political figure walking though the park with his dog, there will be no sports, no wars, and no in-door meetings shown in the same scene. However: you do not know this beforehand, so the detectors are still necessary. The current state of art in detector technology available to us allows us to detect in the order of hundreds of semantic concepts in news video material.

Another important, until now unused, aspect of video is that it is constructed with a certain sequence in mind. Every shot has a certain place in the story of the video, and this place is a key element in telling the story. For instance: you cannot just swap the beginning of a movie with the end, because it would utterly destroy the story of that movie. The sequence of shots is intentional, and as such needs to be preserved. So besides the concept vector extracted from a shot the sequence in which it was aired along with other shots must also be preserved.

2.2 Searching through video

If we look at video as a sequence of fragments we can treat searching through video in the same way as we treat searching trough the textual domain: each fragment or shot is a paragraph in a 'document' and searching means retrieving the most relevant paragraphs of the entire set given some form of query. There are a number of ways to formulate such a query, each having some benefits and some problems. In the list below a couple of methods are described:

- **Query by keywords:** a traditional way of searching in the text domain is searching for keywords in the textual document database. If the words in a document match the list of supplied keywords the document is returned, otherwise it is not. More keywords matches in the document result in a higher ranking in the returned set of documents. In the video domain this is more difficult, since there isn't always an attached text document describing the contents of a video fragment. There are however sources in the video fragment which we can convert to the textual domain. Using speech recognition techniques it is possible to convert available speech in the video fragment to text, and when closed captioning is available for a fragment this can also be used. The resulting textual information can be treated in the same way as a textual document. However a problem arises when there is no closed captioning available, or nothing is said in the fragment. Also closed captions of live television events can contain a lot of typing errors, and speech recognition in uncontrolled environments or with cheap microphones is still very bad. This all severely limits the performance of keyword search.

- **Query by concepts:** this method resembles query by keywords, except that usually the number of searchable keywords is much smaller, and much better defined, than by query by keywords. Concept searching can be done by selecting a set of available concept detectors. Since a concept detector returns a possibility that a shot contains that concept, searching though a video database with this method can be done by returning a list of fragments that have high results for the selected concepts. The main difference between query by keywords and query by concepts is that the former is a data driven method. The information is already exactly there, it only needs to be found. Query by concepts however needs predefined concepts with trained concept detectors to be able to find the concept in new video. A major benefit of this method is that the concepts can also be learned from specific visual information, so this method can actually use the visual information. Another benefit of this method is that problems with bad typing or bad recognition in the text and audio modalities can be avoided. A drawback is that the amount of available concepts is limited which limits the expressiveness and precision of query by concepts.

- **Query by example:** with this form of searching we have an example fragment of video and we want to find video material which looks like this fragment. This can be done by determining the similarity between the concept vectors, and returning a list of nearest fragments. The benefit of searching by example is that all of the available concepts are

taken into account when searching, not just the ones that we think are most important. A problem with this method is that we need an example first. If we do not have an example to start with, one must first be found by another search method.

In the above examples we look at video fragments as document paragraphs, and try to find only these fragments. Video fragments however do not stand alone. A video fragment is related to the ones surrounding him, and a collection of fragments together, for instance a television program, form a story. This means that to find something in the video domain we must not only return a few fragments of each possible story, but also the sequence of fragments that contain the entire story: we must look at the timeline of the video fragments too. Another way to look at this is to compare an entire video with a complete textual document. We can search for these paragraphs but they only have meaning when we look at the entire document too.

3 Research questions

As previously mentionned in the introduction, there is a lot of similar content about the same topic in news video material. This happens when a channel broadcasts a rerun of a programme, or when the same interview is broadcasted multiple times throughout a day, to make sure that everyone has seen it. Content can also change a little but still remain mostly the same. This happens often with recurring items on a channel. For example the evening weather report on a channel is probably be going to look exactly the same the next day, or the day after that. The report itself changes, but it will still be recorded in the same studio, often with the same weatherman, and it will follow the same style. Furthermore similar content also happens across channels. For example: during big disaster somewhere in the world it often happens that multiple channels will transmit live imagery of that event. In this case the topic of the videos from those channels is the same, because it was aired at the same time. This also includes filming location and therefore visible scenery, and detectable semantic concepts, such as fires, car chases or explosions.

If duplicated content is linked together, then this would make things easier for search purposes: only one shot of a linked set would have to be found to find them all. For example: if we find one scene of a weather report, and weather reports are linked together, then all other weather reports are also found. Now a weather report isn't very interesting because weather reports from a specified channel would also have been found together based on visual similarity alone: all the weather reports share the same studio background, use the same colors, have the same textual content or introductory queues which can be detected and they all have approximately the same airtimes. It would be much more interesting if we can link topics together which have the same semantic content, even when the visual content alters a lot.

Let's call a a list of shots linked together based upon their semantic content a "semantic thread". In the paragraphs above it is stated that having threads would be beneficial to search. They also give extra meaning to the shots contained inside because all these shots have some semantic property in common. This brings us to the main research question this paper tries to answer:

> *If we have a large dataset of video material, is it possible to enrich this dataset by linking shots together as semantic threads, and can we then use these threads to enhance navigation through the dataset?*

Looking at this question and the paragraphs above we can derive a few properties we expect from semantic threads:

- a semantic thread gives extra meaning to the shots contained inside
- a semantic thread has a specified ordering of shots which gives extra meaning to the shots

- a semantic thread makes navigating in the dataset easier

To try to answer the question we first need to look at the term ”semantic thread” somewhat more closely. Let’s give a definition of a semantic thread based on the above requirements:

Definition 1 *A* **semantic thread** τ *is a linked sequence of camera shots in a specified order, based upon their semantic content.*

Shots in a semantic thread share some semantic meaning. In the example above: all shots of weather reports could be in the same thread. It is good if this is extended so that there are no shots outside the thread with the same semantic meaning. Continuing the example: there should be no shots of weather reports that are not in the *”weather reports”* thread. This enhances search even more since we not only have to find only one example of a weather report to find a lot of them, but we now only need one to find them all, and be sure that there are no more elsewere. Summarizing this we can define the contents of a semantic thread as follows: *a semantic thread contains shots that have a same meaning, and two different threads should not have the same meaning*.

This is rather vague so we need some sort of measurement methodology to be able to determine different meanings. If we look at all possible semantic meanings a shot can have as a field of possible meanings, then it must be possible to draw a line between two shots and measure its length to determine meaning-closeness. Let’s call such a line the semantic distance between two shots. As a definition:

Definition 2 Semantic distance S *is the distance between shots, in relation to their semantic meanings.*

This means that when a person looks at two shots with a low semantic distance between them, he would think the shots denote the same event. Using this definition we can give a more algorithmical definition for the contents of a semantic thread:

Property 1 *A* **semantic thread** *contains a low internal combined semantic distance, and a high semantic distance to other threads.*

According to this definition threads should not have a low semantic distance to each other, or simpler said the topics of both threads should not be the same. Shots themselves can however belong to multiple threads at once, since any shot does not necessarily have to contain only one topic. For example a shot could belong to the CNN thread, indicating the originating TV channel. The position in the CNN thread then indicates the time at which it was broadcast as well as what comes next and happened before the shot. This thread can be called a ”time thread”, since it signifies when and where this shot was broadcast. That same shot could also belong to one or more semantic threads describing different kinds of semantic content for that shot. For example: the same shot could belong to the *”presidential leader”* thread, the *”aircraft”* thread, and the *”people walking”* thread. These threads together in a shot could signify that a president is getting onboard an airplane, which seems to happen often on CNN. However the thread names dubbed above are just examples. Depending on the granularity of the threads there could very well have been a *”president of the united states gets onboard an airplane”* thread, in which case this would be the only extra thread the shot participates in.

The meaning of position in the case of semantic threads has not been defined yet. One of the properties stated above is that threads should make navigation easier. Since navigation usually means moving from something to something connected, we can define navigating a thread as moving from one shot to another shot next to or near the originating shot, in the same thread. If a semantic thread signifies some sort of similarity between shots in the thread (e.g. they all are shots of a president getting inside an airplane) then shots next to each other should give extra meaning to this. Would it be better to have shots next to each other because of some extra property of that shot? A possibility here is the property ”broadcast time”: shots next

to each other signify that the shots were aired after each other. Another possibility is using the internal semantic distance of a thread to determine positioning. For example: if the two shots with the lowest semantic distance between them are placed next to each other; this would ideally mean for the president example that shots of ”George W. Bush getting on the Airforce One” will be placed next to each other, while shots of ”Bill Clinton getting on some other airplane” will be grouped together somewhere else in the thread.

Since the main research question is not a simple one to answer, let’s try to separate the question above into a number of subquestions:

- *How can we find semantic threads in automatically annotated video material?*
  Using a dataset of existing video material: is it possible to cluster this material based upon its computationally generated semantic annotation? And for the created clusters: what is the optimal way to link the shots in each cluster to each other? And what is the best way to link clusters to each other?
- *How can the temporal sequence of video material be used in creating threads?*
  Obviously the temporal sequence is a thread in itself, but where does this thread stop? How many different threads can be made from the temporal information? How does the temporal information help in creating non-temporal threads? In what way should we be treating gaps in an otherwise continuous timeflow? Is a border between two programs a gap?
- *What is an effective way of presenting the thread structure to the user?*
  If we look at all the threads together the view would resemble a spiderweb of connected shots. To actually visualize all the threads like this would be confusing when the dataset is big. A method to show a subset of the entire collection of threads is required, and with that a method to intuitively navigate the shown subset across the rest of the available threads.

4 Related work

4.1 Topic detection and tracking

As mentioned in the introduction, the problem of finding blocks of related video material is not limited to the video modality alone. Many of the problems we face within the video domain also exist in the text domain. However story detection within the text domain is more mature, and because of this a short overview of topic detection is given here.

The topic detection and tracking (TDT) programme [1] consists of a number of tasks: story segmentation, topic detection, topic tracing, first story detection and link detection.

Story segmentation tries to create cohesive parts of story from a continuous stream of new data. This task especially is not limited to the text domain, but also applies to the spoken audio domain. *Topic detection* tries to cluster stories together from a set of data. *Topic tracking* tries to keep track of stories using an example set of stories. *First story detection* tries to determine if new data is the beginning of a new story, or a continuation of an earlier story, and *link detection* tries to establish wether two story clusters are related to eachother.

The topic detection task within the TDT programme tries to detect stories within a set of events by creating non-overlapping clusters of events. This can be done both globally, or incremental. With global clustering the entire database is clustered at once. With incremental clustering new events are clustered when they are added to the database. Incremental clustering is preferred for the TDT track since TDT mostly handles news stories and other information which is still 'happening'. In [3] methods and algorithms for TDT are discussed which focus on incremental clustering. Batches of data equal in length to an half hour news broadcast are added incrementally. This is based upon a real time approach: every new story has to be clustered somewhere now, mistakes in clustering cannot be corrected later on. The paper suggests user interaction as a partial solution for this. The basic principle of TDT clustering works as follows. Every term in a story is weighted, and the 1000 highest weighted terms form the story vector. This story vector is then used to cluster the story by determining its similarity to every other story in the collection using single link clustering: if there is a significantly similar cluster the story is added there, otherwise it forms a new cluster on its own. Variants on this method are discussed and evaluated, such as changing the method of selecting the story vector, or introducing a penalty when there is a large time difference between stories.

One of the problems stated in [3] with the topic detection task is that every event is added to exactly one cluster. This means that every event can only be part of one story. If an event discusses more than one story it cannot be part of the other story-clusters. Since 2004 hierarchical clustering methods have been proposed to allow for this overlap in clusters.

When new story items arise (in our case: when new video material has been broadcast) they will have to be incorporated into the existing framework of detected stories. For this the *topic tracking* task exists. The topic tracking task uses a selection of user-selected events to learn what the story is about, and then tries to find and return new events on the same story when they occur.

4.2 Semantic threads

In [4] the problem of finding structure in a video collection is analyzed by borrowing tools from the field of Natural Language Processing. They do this in a similar fashion as our method: shots with concept labels describe features in the shot. These concept labels are then used to cluster the shots together in groups. These groups are then merged using existing information about episode boundaries and story classifications to form storylines. Time-information is added to these storylines and the resulting dataset is then re-clustered to form clusters with high semantic similarity inside each cluster, and high semantic differences between clusters.

Differences between the methods described in [4] and our method are that we use generated concept labels for shots based upon automatic concept detection, whereas they use annotated concept labels. For small datasets this is feasible, but when datasets grow larger this is impossible to maintain. Using generated concept labels is a better solution. Also, an interesting side-effect of using annotated shots is that the generated storylines tend to represent annotator biases, due to the way the database was annotated.

Also the difference measure between shots in [4] is based upon a binary string comparison: a concept either exists in a shot or it doesn't. This limits the expressiveness of the concepts severely. For example: is a concept present in a video when the concept is only visible for a fraction of a second? The concept was clearly visible, but not as much as some other, more visible concepts. Should this concept then be included or not? When a concept visibility percentage is allowed however this problem dissapears. Also the concept detectors used in our approach only give an estimate of how much a concept could be in a shot.

Another approach uses semantic pathfinder networks[5] to aid in visual navigation. Using Generalized Similarity Analysis they extract structural patterns from a dataset consisting of a collection of conference papers. The spatial layout is transformed from a similarity matrix to a 3D representation. The pathfinder methods removes redundant information from the similarity matrix by validating all links with the triangular inequality condition. Links between impossible configurations of objects are removed and the resulting graph of papers is then visualized using force directed graph drawing algorithms. The resulting visualization is then viewable using a VRML enabled web browser. The dataset used in this paper is based on a collection of 169 conference papers, and the relations between them, but the method could be adapted for use in the video domain. This would provide an interesting method of creating a linked visualization of shots. However the visualization (example in figure 2) cannot be easily extended to accommodate larger datasets without cluttering the display and making the system unusable. And although a video dataset could be used to generate the 3D graph, the visualization itself is only capable of displaying textual information. If keyframes were to be displayed the visualization would become even more cluttered.

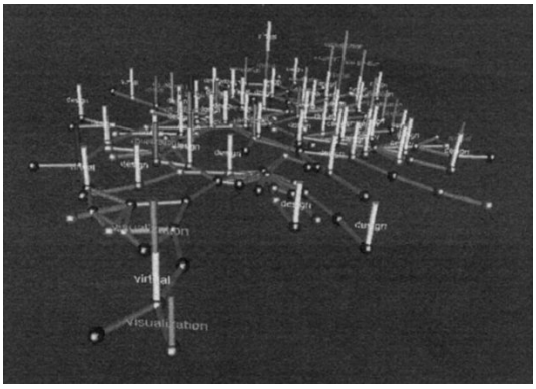


Figure 2: Example shot from [5], this system shows a three dimensional representation of a collection of conference papers.

Another method for visualizing large video collections is proposed in [6]. This method maps the high dimensional image feature-space to a two dimensional space while keeping the structure in feature-space between the images intact. The resulting visualization gives a two dimensional representation where visual similar images are close together. This system is combined with a GUI system which features interactive learning to augment the resultset[7]. See figure 3.

This system works very well for visual similar imagery but this poses a problem when something needs to be found which is not visual similar. Because the system shows a lot of small keyframes at once, navigating though the visualization when there are no visual similar images can be hard.

4.3 Adding temporal information

Combining content based similarity browsing with the temporal domain is not new. In [8] a method is proposed for cluster-temporal browsing, combining both the timeline of videos and the content-based similarity in one view. In [9] this is extended to large news video databases. The resulting interface gives the user a grid with temporal ordering on the horizontal axis, and shot similarity on the vertical axis. See figure 4 for an example. The temporal adjacency is only used for enriching the resulting browser application, not during the clustering process itself.

In [10] another video retrieval system is outlined. Amongst other things the system includes a temporal view, in which the next and previous shots for the current shot are shown on a fisheye view timescale. The system also implements a graph network browsing method using so called NN^k networks, which visualizes a network of related shots, see figure 5. The NN^k network and the timescale are however not combined into one view.

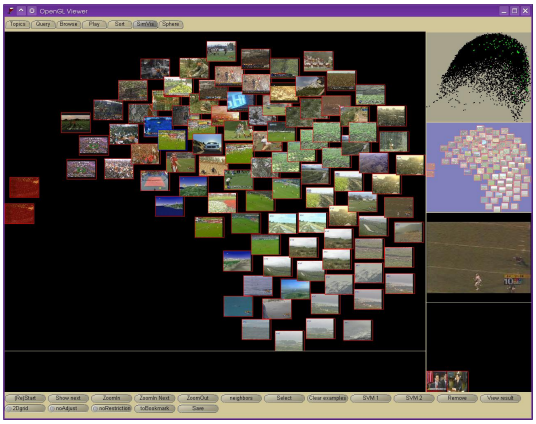


Figure 3: The GalaxyBrowser system from [6] and [7], this visualization maps a high dimensional feature-space to a two dimensional space while keeping the structure in feature-space between the images intact.

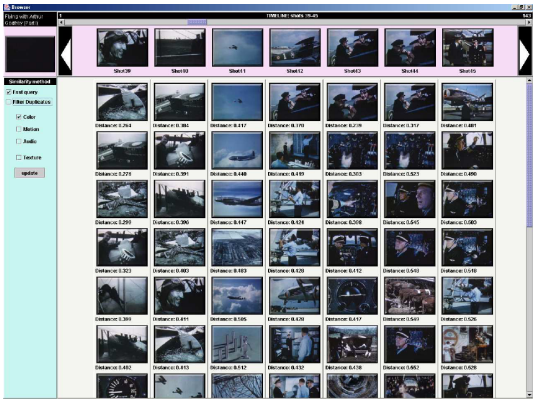


Figure 4: System from [8], ©Mediateam. This system shows the time on the horizontal, and the cluster based content related shots on the vertical axis

5 Determining threads in semantically annotated video

In this chapter a method for determining threads in semantically annotated video is proposed. The task of creating threads is broken up in a number of smaller tasks. See figure 6 for an overview.

In summary, the proposed method works as follows: first a video feed is segmented into shots. These shots are then run through a series of concept detectors. Each concept detector returns a probability value of that concept being in the shot, and these probabilities together form the concept vector for a shot. With the concept vectors threads can then be created by first clustering the shots based on similarity between shots and then adding a navigation order. This is explained in more detail in the next sections.

5.1 Finding threads

To create a thread τ we need a group of related shots $s_0, ..., s_n$. These groups can be created by creating clusters within the entire collection of shots in the dataset. For this we need a dataset with learned semantic concepts. Our prototype uses the TRECVID 2004 and 2005 [11] benchmark datasets which contain a selection of news broadcasts, combined with an automated semantic annotation. For the TRECVID dataset we use the semantic concept detector framework as described in [12] and [13]. Each concept detector tries to detect the presence of a certain concept in a shot. The quality of the detectors vary, and some detectors are very unreliable.

Since the domain of the TRECVID dataset is quite broad we cannot estimate in advance how many groups of related shots there will be. An important question therefore is: how many threads, and thereby clusters, do we need, and how does the number of available semantic detectors or the total number of shots in the database influence the amount of threads?

To answer this question it is better to ask ”What should be inside a cluster?”. To answer this we can look back to definition 1 on page 9. Since we are trying to create a thread from a cluster, the same rules apply to clusters as to threads: shots with a low semantic distance to each other must be clustered, and all shots which have a high semantic distance from the members of this cluster should not be inside that cluster.

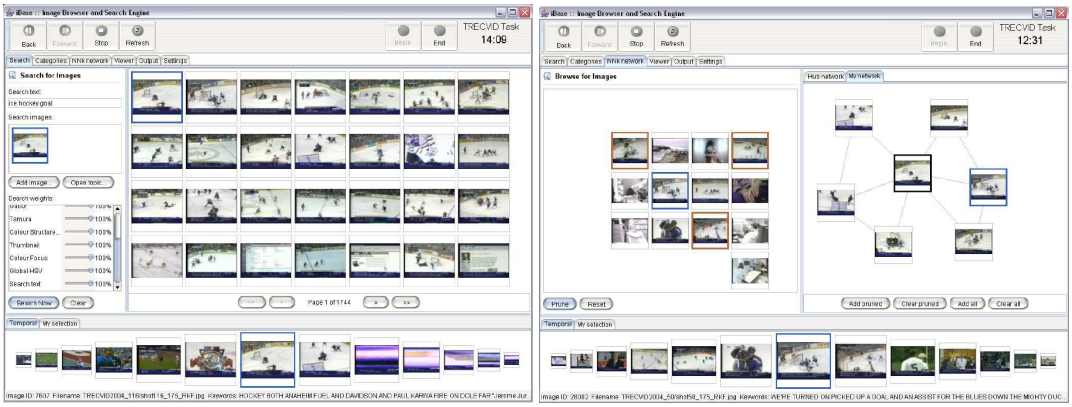


Figure 5: System from [10], ©Imperial College London. The left figure shows a temporal browsing pane with the timeline visible with a fisheye effect. The right figure shows the graph network browsing pane.

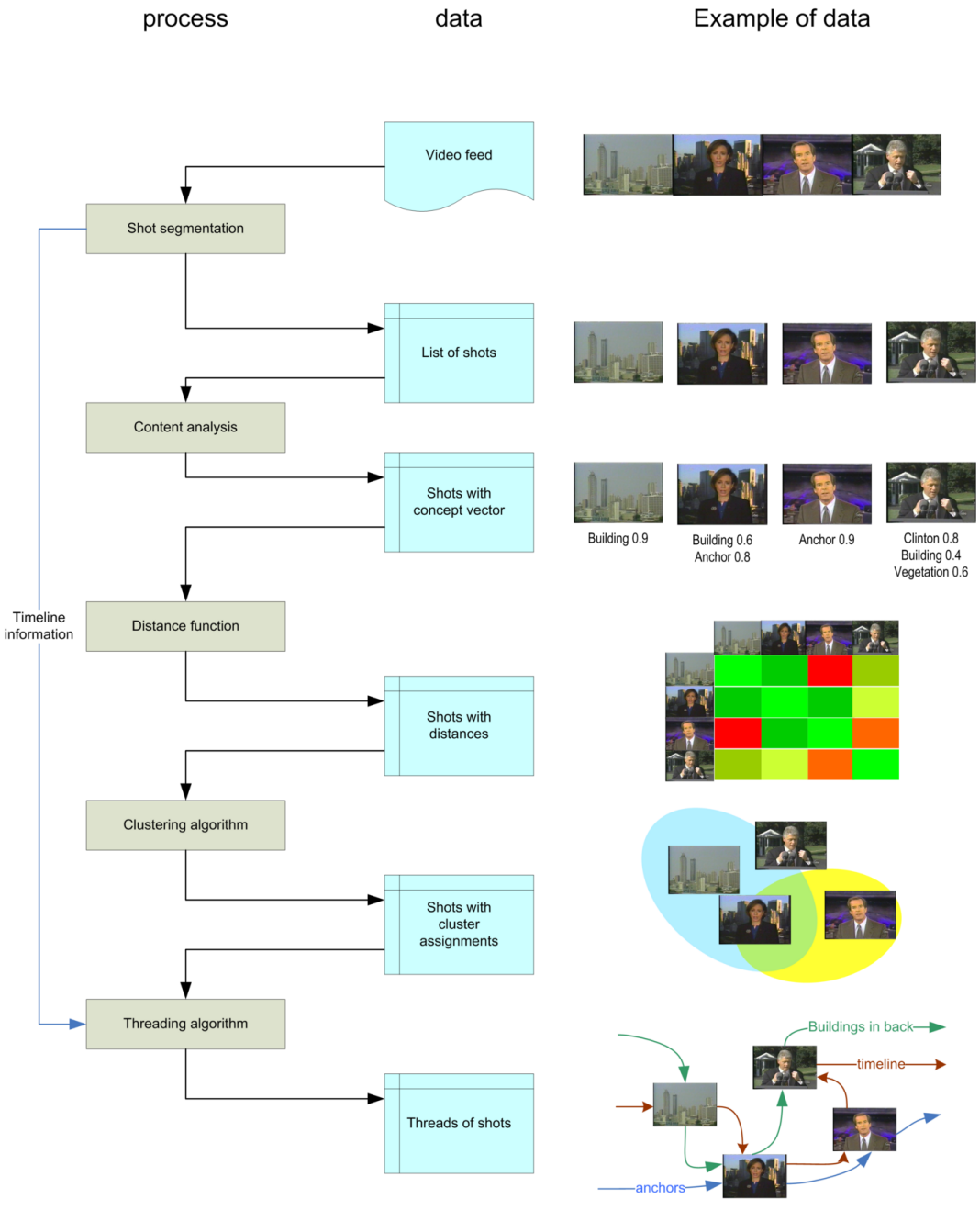


Figure 6: Process overview

5.2 Removing unwanted shots

Shots in a thread have a semantic relationship to each other, in practice this means that every shot should contain at least one detected concept. Given a limited number of concept detectors it is however not probable that every shot will have at least one detected concept. If these shots are then allowed to be clustered together with the rest of the shots they could end up in threads as garbage. Since the amount of zero concept shots depends on the selected dataset and the available concept detectors it is not known beforehand how many shots will fall into this category. Therefore it is necessary to define what to do with these shots if they occur.

A solution could be to prune them before any clustering happens. In this case pruning shots is the action of removing shots from further processing at an early stage so that they are not used for calculations and will not show up in the resulting threads. In summary there are two reasons why pruning might be necessary:

- **a conceptual reason:** generated concept vectors for shots more or less describe the content of each shot, but since there are only a limited number of categories for detection, a problem arises when a shot doesn't fit into any category, i.e. all its concept features are near-zero. The next steps will however place the shot in some thread, and the GUI will show that thread to the user who then has no idea that the shot wasn't useful at all.
- **a computational reason:** When the amount of video material keeps increasing the number of shots increases as well. Since the amount of needed computations scale exponentially, creating threads would soon become computationally infeasible. By limiting the number of shots in the next steps to only the top-N 'perfect shots' from the dataset we can still cope with large datasets.

Conceptual pruning, or pruning because of the conceptual reason, is simple: when the highest value in the concept vector for a shot v_s falls below a specified threshold Γ the shot must be pruned:

Algorithm 1 Conceptual pruning

```
S = list of all shots
A = [ ]
for all s in S:
        if max_{v_s} < Γ
                prune s
        else
                add s to A
return s
```

When computational pruning is required however a problem arises as to which shots must then be thrown away. All shots already have some semantic meaning so it is best not to throw them away at all. The resulting subset should at least have the same characteristics and the same semantic concept distribution as the larger dataset, or otherwise pruning leads to a conceptually biased subset, which would result in biased threads. The following algorithm was developed to do this:

In the resulting selection each concept has a roughly equal amount of shots and the selection does not contain zero vectors. The algorithm also ensures that at least a top-N shots from each concept detector is included, even when that detector has very low values compared to other detectors. Also a maximum number of shots may be specified for computational reasons without this leading to a non-homogenized subset of the shots.

Algorithm 2 Computational pruning

1. For each concept: order all shots in a ranked list with the shot with the highest value for that concept first.
2. Round-robin select from each list the first shot that isn't previously selected.
3. The first top-N selected items from each ranked list are accepted automatically. The remaining (bottom end) of the list must also be pruned using the conceptual pruning threshold.
4. As soon as a predetermined maximum number of shots is reached the algorithm stops.

5.3 Prerequisites of threads

After pruning, each remaining shot now contains at least one detected concept. With this information we can create a distance measurement between shots. But how do we measure distance between concept vectors? This depends on two things: wether the concepts in relation to each other have equal importance, and the distance metric used to determine shot dissimilarity S_{pq}.

Determining importance of concepts can be done in multiple ways. It is possible to look at the entire dataset and see how much each concept is represented in the dataset, and use this to create an inverted weighting so that each concept is represented equally. It is also possible to measure the quality of each concept detector by evaluating the concept detector against a ground truth of a testing dataset, and use this quality index as a weighting factor so that better concepts are more important in determining distance. These weightings are both made on a technical level.

On a conceptual level there is also a distinction to be made: the available concepts aren't homogeneously distributed across all possible subjects in real life, or in this case: the news video domain. This is more evident when we look at a small number of concept detectors. For instance, is the distance between *golf* and *football* smaller than the distance between *golf* and *airplane*? It would seem so, since golf and football are both a kind of sport.

The fact that concepts can influence each other inside the concept vector this does not influence the values in the concept vector. But it makes a difference when we look at the distance between two concept vectors, or two shots. This leads to the same question: should a shot with only the *golf* element in its concept vector have a lower semantic distance to a shot with *football* or to a shot with *airplane*? If the answer is yes here then this means that the distance between shots with related concepts should be lower than those of shots with unrelated concepts. This also means that shots with related concepts will be placed in the same thread faster. Threads with multiple concepts would be biased to have concepts together which mean something to each other. For example threads with different kinds of sport could be created, or threads with different kinds of vehicles.

To implement this semantic concept closeness in the semantic distance calculation we need to convert semantic nearness to a weighting scheme which can then be incorporated into a distance function. A possible solution for this would be to add a matrix which defines the semantic distance from one concept to each other concept, and then use this matrix for the distances. To construct such a matrix an independent metric for semantic distance between everyday concepts is necessary. An ontology of all possible subjects with the concepts linked in could provide this metric. For instance, if WordNet[14] were to be used to link all concepts together then the distance between concepts could be measured as the number of steps necessary to get from one concept to another concept using, for example hyponym and hypernym relationships. If we now measure all concepts to each other we can construct a number-of-steps matrix for all concepts. This matrix can then be inverted and normalized to create a matrix which determines the semantic relatedness of the concepts.

For simplicity we use a conceptual equal distance between concepts, meaning that *golf*, *football* and *airplane* have all equal importance. Then we can construct a distance matrix made up from the similarity S_{pq} between shots p and q using well-known distance metrics. This paper compares three methods: manhattan distance, Euclidean distance and histogram intersection.

Euclidean distance

The Euclidean distance is one of the most common methods. As stated in section 5.3 it is possible that the detectors used to create the concepts vectors can have different accuracies. This is the case both the 2004 and the 2005 dataset. Testing the concept detector performance against a test set would provide a accuracy score for each detector. This can then be used as a weightmap in the distance function to lower the effect of bad detectors. For the 2004 FST dataset these weights are known and described in [12]. See table 1 for the weights. A higher value means higher accuracy as measured during testing. Because the detectors have got different accuracies the weighted version of the Euclidean distance algorithm has been used. (eq. 1) Less accurate detectors are given lower weights.

$$S_{pq}^{w} = \sqrt{\sum_{i=1}^{N}(p_i - q_i)^2 \dot{w}_i} \qquad \begin{array}{l} p \text{ and } q\text{: the feature vectors of two compared shots.} \\ w\text{: weight vector} \end{array} \tag{1}$$

Feature	Weight	Feature	Weight
aircraft	0.1	albright	0.12
anchor	0.99	animal	0.37
baseball	0.54	basketball	0.30
beach	0.13	bicycle	0.09
boat	0.42	building	0.53
car	0.75	cartoon	0.75
clinton	0.37	dowjones	0.89
financial	0.7	football	0.46
golf	0.24	graphics	0.92
icehockey	0.71	monologue	1.00
outdoor	0.90	overlayed_text	0.99
people	0.91	people_walking	0.83
road	0.51	soccer	0.01
sport	0.98	studio	0.98
train	0.07	vegetation	0.72
violence	0.31	weather	1.00

Table 1: Weights for TREC 2004 feature vectors according to [12]

Manhattan distance

The manhattan distance algorithm or absolute value distance (see eq. 2) is a distance metric which measures the distance between two vectors as a summation of the distances between each pair of elements in both vectors. For concept vectors this means that the manhattan distance algorithm will do the following:

- A concept that is not in both shots will not have influence in the similarity $|low - low| = low$
- A concept that is in one shot, but not in the other, will have influence. $|high - low| = high$
- A concept that is in both shots will not have influence. $|high - high| = low$

This means that only shots with concepts which are available in one shot and not in the other get placed further apart.

$$S_{pq} = \sum_{i=1}^{N} |p_i - q_i| \qquad p \text{ and } q\text{: the feature vectors of two compared shots.} \tag{2}$$

Histogram Intersection

Another method, histogram intersection ([15], eq. 3), originaly comes from the field of computer vision. It is usually used to find known objects in other images. It does this by only taking into account colors that are in both images simultaneously. Pixels in one image, but not in the other do not reduce the similarity. Pixels that exist in both images (a match) increase the similarity. Because of this the histogram intersection match value is viewpoint, occlusion and resolution independent and the background of an object has no effect in it.

Those aspects are also wanted if we look at the concept vector as an histogram. This can best be explained by looking at the effects of using histogram intersection on concept vectors:

- A concept that isn't in both shots will result in $min(p, q) = 0$, and not have influence. This seems acceptable.
- A concept that exists in one shot but not in the other should not have influence in the similarity, because $min(p, q) = low$.
- However if there is a concept that scores high in both shots then it will have influence, because $min(p, q) = high$.

This means that vectors with one or more concepts in common will be very close to each other. The difference with manhattan distance here is that concepts with low values in the two compared shots do not increase the similarity. This is good since for every shot a large part of the concepts will have zero values.

Histogram intersection originally compares a target image with a query image, this would yield an asymmetric similarity matrix. Our system uses an adapted version of histogram intersection which provides symmetric similarity matrices. See equation 4.

$$S_{tq} = \frac{\sum_i min(t_i, q_i)}{\sum_i q_i} \qquad t\text{: target histogram, } q\text{: query histogram.} \tag{3}$$

$$S_{pq} = \frac{\sum_i min(p_i, q_i)}{\sum_i max(p_i, q_i)} \qquad p \text{ and } q\text{: the feature vectors of two compared shots.} \tag{4}$$

Clustering

Now that we have looked at different types of similarity, we can look at standard clustering algorithms to see how they can create usable clusters for the next step.

K-means clustering

K-means clustering is initialized by randomly selecting a number of shots. The K-means clustering algorithm iterates over two steps:

1. assign all shots to the nearest cluster-center
2. determine the new cluster centers for each cluster

After every pass the new sum of squares (δ) is calculated using 5. As soon as the sum of squares does not significantly drop anymore the clusters should be optimal.

$$\delta = \sum_{c \in C} \sum_{i \in c} S_{d_i \sigma_c} \qquad \begin{array}{l} C\text{: clusters} \\ i\text{: elements in cluster } c \\ \sigma_c\text{: the center of cluster } c \end{array} \tag{5}$$

A disadvantage of K-means clustering is that each cluster roughly has the same circular shape in feature space, and that outliers in the dataset can have too much influence. However, since the concept vector dataset is bounded between 0 and 1 this is not a big issue. Another problem with K-means clustering is that it creates spherical formed clusters in the concept vector space.

Single-link and complete-link clustering

As an alternative to K-means clustering an hierarchical clustering method was also investigated. The generic algorithm for hierarchical clustering works as follows:

Algorithm 3 Hierarchical clustering

for all x **in** shots **do**
    $cluster_x = shot_x$
$C_{pq} = S_{pq} =$ similarity matrix for all shots p, q

find lowest distance in C:
do:
        $cluster_x \leftarrow merge(cluster_x, cluster_y)$
        $C_{pq} \leftarrow mergesimilaritymatrix(cluster_x, cluster_y)$ using merge function
until threshold

In comparison to K-means the single link and complete link clustering algorithms return clusters with non-uniform sizes. It is possible to have clusters of size 1. The difference between single and complete link is that with single link the distance matrix is recalculated using the minimal distances between the two merging clusters, and with complete link the maximal distances. Another criteria could also be specified for ending the cluster-generation process. For instance: not stopping at a specific number, but when internal distances are greater than x. This is a solution that needs examining, since the number of clusters is not known beforehand.

This chapter describes how hierarchical clustering and K-means clustering can be used for creating clusters in the news video dataset. We can now look at how to create threads from these clusters. For this we first need to look at how we can use the original broadcast time information in the threads.

5.4 Adding temporal information

Time in the news video domain could be described as follows: for each day there can be number of channels with recorded material. Each recording can contain a number of television programs, and each program consists of a series of consecutive shots. These shots within one series are time-related to each other and are gapless. Each program in a day is time related to each other, but there could be gaps in between, with programs that are not recorded. Each day in the entire domain has a time-relation to the other days in the domain. The complete timeflow for a channel can therefore be described as a sequence of days each containing a sequence of programs each containing a sequence of shots. See figure 7 for a graphical representation.

Furthermore there can be more than one channel broadcasting on the same day. This means that several programs can coexist at the same time. In the Topic Detection & Tracking domain this is denoted as having multiple sources. Each channel produces its own timeline.

A problem here is the effect of previously pruned shots. When the middle shot of three consecutive shots is pruned, can the outer two shots then still be considered gaplessly linked? The answer is no, but a solution could be to guess information about the pruned shot from the nearby shots and re-insert the shot in the thread. This was not investigated further in our system.

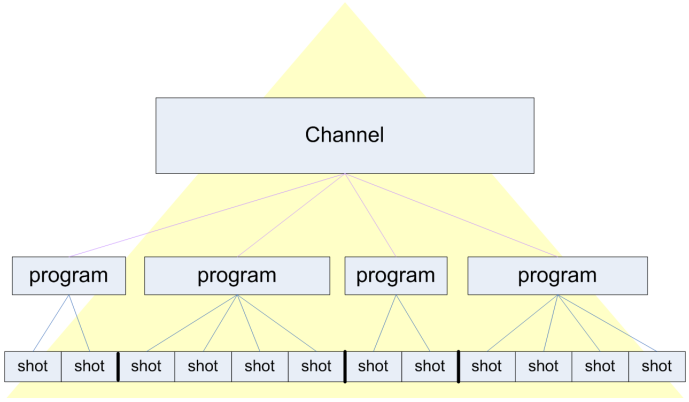


Figure 7: Time is a hierarchy

5.5 Linking the shots

Now that there are clusters of related shots, and multiple resolutions of time, we can link the shots together as threads. This can be done in various ways: we can link the shots together based on their broadcast date and time, we can link the shots together based upon the clusters they belong to, each cluster creating a cluster-thread, and thereby creating a thread with related shots, and we can link clusters together.

- **time threads** τ_c^{time}: each channel c represents a thread beginning with the very first shot recorded for that channel and ending with the very last shot recorded from that channel.

- **cluster threads** $\tau^{cluster}$: from each cluster a thread of semantically related shots can be created by adding a navigation order. The navigation ordering of shots within a thread can be done either according to time, or according to some other criteria.

5.5.1 Time threads

Each channel represents a time thread τ_{time} beginning with the very first shot recorded for that channel and ending with the very last shot recorded from that channel. But this would also mean that all these extra threads need to be integrated in the resulting GUI somehow. In this section the possibility of merging these threads together to form a single time thread is discussed.

To generate a single time thread from these channel time threads is however not trivial. The resulting channel time threads could then be joined together to form a single pseudo time thread spanning across the entire video collection if we allow events happening on the same time, ie two different programs airing on the same time on different channels, being placed after each other, but how do we do that? Let's first list all the possible methods for combining the time threads together. See figure 8 for a graphical representation.

Figure 8: Different methods for constructing the time threads

The following methods for combining the time threads together can be described. See figure 8 for a graphical representation.

A **place the channels after each other**
- \+ not too much gaps in the time thread, and related shots within a single channel stay near each other.
- \- shots about the same subject aired on multiple channels at the same time (for example: a live car chase, shown on multiple channels) will be very far apart.

B^1 **place the programs after each other**
- \+ programs which air on or about the same time (for example: the eight o clock news broadcast) will be placed next to each other, making it easier to see related subjects.
- \- channels with only one program or a few very lengthy programs will interfere with this method, essentially reducing it to method A.

B^2 **place preset blocks of time after each other** For example all 11:00-12:00 blocks, then all 12:00-13:00 blocks, or all shots from a certain day
- \+ better than A, and without the problem with long programs.
- \- it could mean gaps in programs if we choose a small interval. But a lengthy interval makes finding related shots in programs on other channels less efficient.

B^3 **interweave at every commercial break**
- \+ every Westernized television channel nowadays almost always has commercial breaks. They are already introduced gaps in the programming, so interweaving programs from commercial break to commercial break doesn't seem unnatural.
- \- not every channel has commercial breaks.

C **interweave the shots** This generates a 'real' time thread in which every next shot was aired after the previous shot.
- \+ this generates a real timeline
- \- the shots next to each other that have no relation to each other at all. This problem increases enormously as the number of channels increases.

D **keep separate time threads** This also generates 'real' time threads, but this also complicates the GUI, which will then have the problem of how to display the different time threads
- \+ the problem doesn't need to be solved, leaving the task to the user to determine which time thread or ordering of threads is best.
- \- increases the burden on the user, which could make the system less efficient

Interweaving of shots from different channels to ensure that the timeline keeps being a perfect timeline (method C) would therefore not be recommended: the shots next to each other would have no relation to each other at all. Method A concatenates the different channels, and thereby removes all possibilities of having related live events near each other, with no obvious extra benefits. In this case it would be better to use method D, which leaves the different channels completely unconnected. What remains is the question of if we want to interweave the shots together (method B) or if we just want to have multiple time threads (method D).

If we look at the different interweaving options, then B^2 is the simplest to implement, since it doesn't require information from the channel, such as when the next program starts or when a commercial break takes place. However it shares, to a lesser extent, the same disadvantages as method A and C: it introduces gaps where there shouldn't be and live events still aren't right next to each other.

Methods B^1 and B^3 both create threads without gaps in the story and, especially with method B^3, live events stay near each other. However, during very profound live events, such as an airplane crash, it happens that some channels drop their commercials in favor for giving more information about the event. This would break up this method. The same holds for method B^1.

As a result of all this it seems best not to combine time threads from different channels, but leave them apart (method D). In this case the GUI, and therefore the user, gets control over when to see different channels next to each other, and when not to.

5.5.2 Cluster threads

From each cluster a thread of semantically related shots $\tau_{cluster}$ can be created by adding a navigation N: $\tau_{cluster} = N(s_{c1} \rightarrow s_{c2} \rightarrow ... \rightarrow s_{cn})$. The navigation ordering within each cluster thread can be done in different ways:

- **time-based** Order the shots in a cluster based upon their airing time.

  This will often result in seeing the same shot multiple times in a GUI, since typically a time thread is also visible.

- **closest neighbour** Order the shots in a cluster so that each shots gets linked to its closest unconnected neighbour:

  This ensures that shots that are next to each other usually have a very low distance between each other, which means that shots with similar semantic content are near each other. This method differs from the traveling salesman problem in that it is not important that the resulting path is the shortest path possible.

The closest neighbour algorithm used in this approach works as follows:

Algorithm 4 Closest neighbour

S_{pq} : distance table for all pairs of shots p, q in the cluster
$connections_x$: number of connections for shot x
$\tau^{cluster}$: resulting thread

for all x **in** shots:
    $connections_x = 0$

$S_{sorted} \leftarrow$ **sort** S_{pq} from lowest distance to highest

for each pair of shots p,q in S_{sorted} :
    if $connections_p \leq 1$ **and** $connections_q \leq 1$
        increment $connections_p$ **and** $connections_q$ **by** 1
        append $(p \rightarrow q)$ **to** $\tau^{cluster}$
    else skip distance
until all shots are connected

6 Basic experiments

6.1 Clustering and threading engine

In this chapter the proposed methods and algorithms for each step from chapter 5 are evaluated, and a working prototype is build. The algorithms chosen in this section are used in the complete system evaluation in chapter 7.

6.2 The datasets

The final prototype uses the TRECVID ([16]) benchmark datasets from 2004 and 2005. The TRECVID 2004 benchmark dataset contains 64 hours of ABC World News Tonight and CNN Headline News video material, all recorded from television in the timespan October 1998—December 1998. The 2005 dataset consists of video material from 7 different channels in three different languages: English, Chinese and Arabic. For each year TRECVID provides the datasets split up in two parts: the feature search development set (FSD) which can be used for development and training purposes, and the feature search test set (FST) which must be used during the actual benchmark.

The TRECVID 2004 FST dataset consists of 33367 individual shots. For this dataset 32 concept detectors have been created. These are listed in figure 20.

The TRECVID 2005 FSD dataset consists of 43654 individual shots from recorded material in a 17 day timespan. For each shot 81 calculated concepts are available. The TRECVID 2005 FST dataset consists of 45765 individual shots. For each shot 101 concept detectors are available, these are listed in figure 21.

Pruning the 2004 dataset

For the 2004 dataset limiting the amount of used shots for further processing is fairly simple. There are only 32 concepts available, and the chance that all values for a shot are below preset minimum is fairly high. See figure 9 for more details. For example: if we accept only shots with $\Gamma = 0.4$, then a large quantity of shots actually is not accepted at all. 27998 shots of the 33367 (84%) would be pruned, and only 5369 (16%) shots would remain. This is because a large portion (70%) of the dataset has a maximum detector value of 0.1.

If we look at the threads this amount of pruning results in complications. Either there will be many gaps in the resulting time thread if we decide to remove the pruned shots also from the time thread, or very many shots in the time thread will not have a related cluster thread since those shots were pruned. If we were to actually prune this much it would mean that only 16% of the data would be used for searching and browsing, and only results from these 16% could be selected. If we assume for a moment that all the shots for a topic are homogeneously distributed across the 2004 dataset then only a maximum recall of 0.16 is possible. That is not good. It would be better to drastically lower Γ so that at least 90% of the data is included, even when it contains garbage. For the 2004 FST dataset this would mean that Γ must be set to 0.06.

Pruning the 2005 dataset

The amount of concept detectors of both 2005 datasets and the fact that some detectors produce very high values (as can be seen in figure 9), pruning the dataset in the same manner as the 2004 dataset would not mean any significant pruning. For example, if we try to prune using the same method as with the 2004 dataset: pruning the FSD dataset at $\Gamma = 0.4$ would remove 109 shots from the 43654 available shots. (0.25%). From the FST dataset only 129 of 45765 shots would be pruned (0.28%).

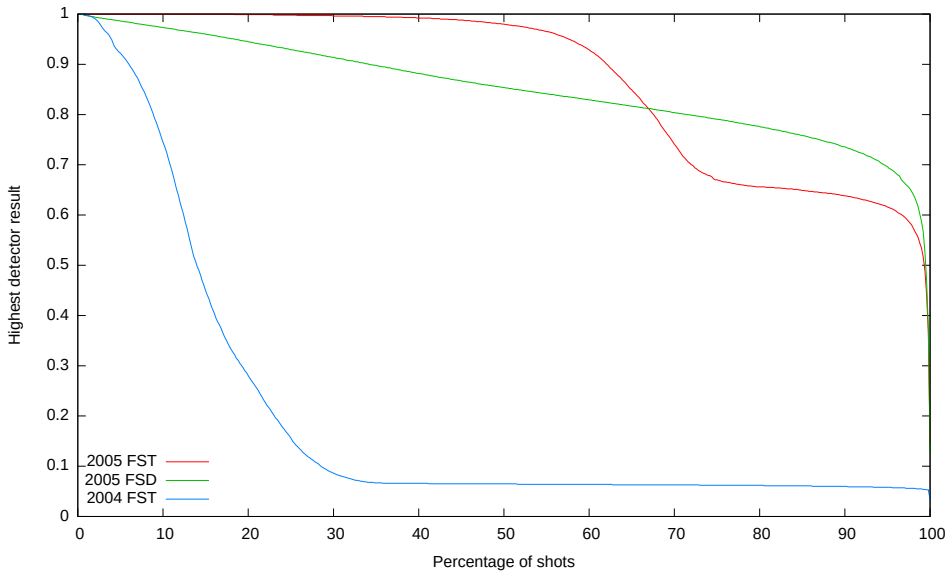


Figure 9: Maximum value from concept detectors for different datasets. The x axis shows the shots sorted by highest detected concept value, the y axis displays this value. From this graph can be read how many percent of the shots contain at least one detected concept. As we can see only 20% of the shots from the 2004 FST dataset have at least one detected concept, while the 2005 dataset has at least one detected concept for every almost every shot.

As stated before in section 5.2 there can be two reasons behind pruning. Conceptual pruning is needed when there are shots without semantic annotation. Computational pruning is required when the dataset is too large. Since the values of the detectors are so high pruning isn't required for the 2005 datasets. During the early research stages of the threading engine many technical difficulties led to believe that it would be practically impossible to create threads using the entire 2005 dataset. To cope with this problem the computational reason behind pruning was devised. However at a later stage the initial computational problems encountered with the 2005 datasets were solved programmatically, and as a result no pruning was needed at all.

6.3 Determining similarity

Each remaining shot in the TRECVID dataset now contains at least one detected concept. With this information we can create a semantic distance function between shots which adheres to definition 2 on page 2. As discussed in section 5.3 we assume equal distances between concepts. We can now construct a distance matrix made up from the similarity S_{pq} between shots p and q using well-known distance metrics. Before we do that however it is good to have an intuitive feeling for the dataset first.

An analysis was done to verify that using self-similarity on the entire news video domain is a valid approach. To this end a similarity-space browser was developed to analyse the resulting similarity matrices from the different metrics have enough granularity for creating semantic threads. In figure 10 a part of the similarity matrix for the 2004 FST TRECVID dataset is shown. Each line represents a shot in the dataset, and the shots are grouped together by the highest concept value for each shot. This results in groups of shots each with the same concept in common. The groups are then sorted by their highest concept according to the alphabetical list of concepts in figure 20. The large dark squares on the diagonal line denote the similarity for shots with the same highest concepts, the darker areas not on the diagonal line represent similarities between shots with a different highest concept. The coloration shows the cluster in which each shot would end up. For this example the histogram intersection metric is used in combination with K means clustering for the coloration.

This picture tells us the following things. First the number of shots for each highest concept varies a lot. Some concepts have a lot of shots, and some nearly none. This doesn't mean that those concepts aren't in the dataset, only that another concept had slightly higher concept values for those shots. Secondly, the internal similarity for a block of shots with the same highest concept are often reasonably low. This can be seen from the dark squares on the diagonal. This is both due to the use of the histogram intersection metric and the fact that for many shots only one concept ranks high.

Now that the dataset itself is somewhat more visible the metrics themselves van be evaluated. This paper compared manhattan distance, Euclidean distance and histogram intersection. A description for each metric was given in chapter 5.3. During early testing it was determined that using a weighted version of Euclidean distance is not better than using histogram intersection as a metric. Also concept detector accuracy weights were not available at the time for the 2005 FST dataset, which led to the use of unweighted histogram intersection in favor of weighted Euclidean distance. Also manhattan distance is influenced too much by the common low detector values in compared shots, histogram intersection provides a higher contrast between related and unrelated shots. The final system uses histogram intersection as the main distance metric.

If we compare K-means clustering with the hierarchical clustering methods then the hierarchical methods seem to be better suited to the task. One mayor drawback of hierarchical clustering as described above is that it is neither time nor space efficient if we scale to a large dataset. Complete link efficiency is $O(n^2 log(n))$ (with n = shots) in time, and $O(n^2)$ in space. While it is possible to create threads for the 2004 dataset using hierarchical clustering, it was unfortunately practically impossible to extend this to the 2005 dataset.

If we look at the space constraint only this results in a problem for the 2005 dataset: a similarity matrix of 45765^2 items has to be created. That equals 2094 milion individual distances in memory. Since the algorithm requires a lookup on this entire

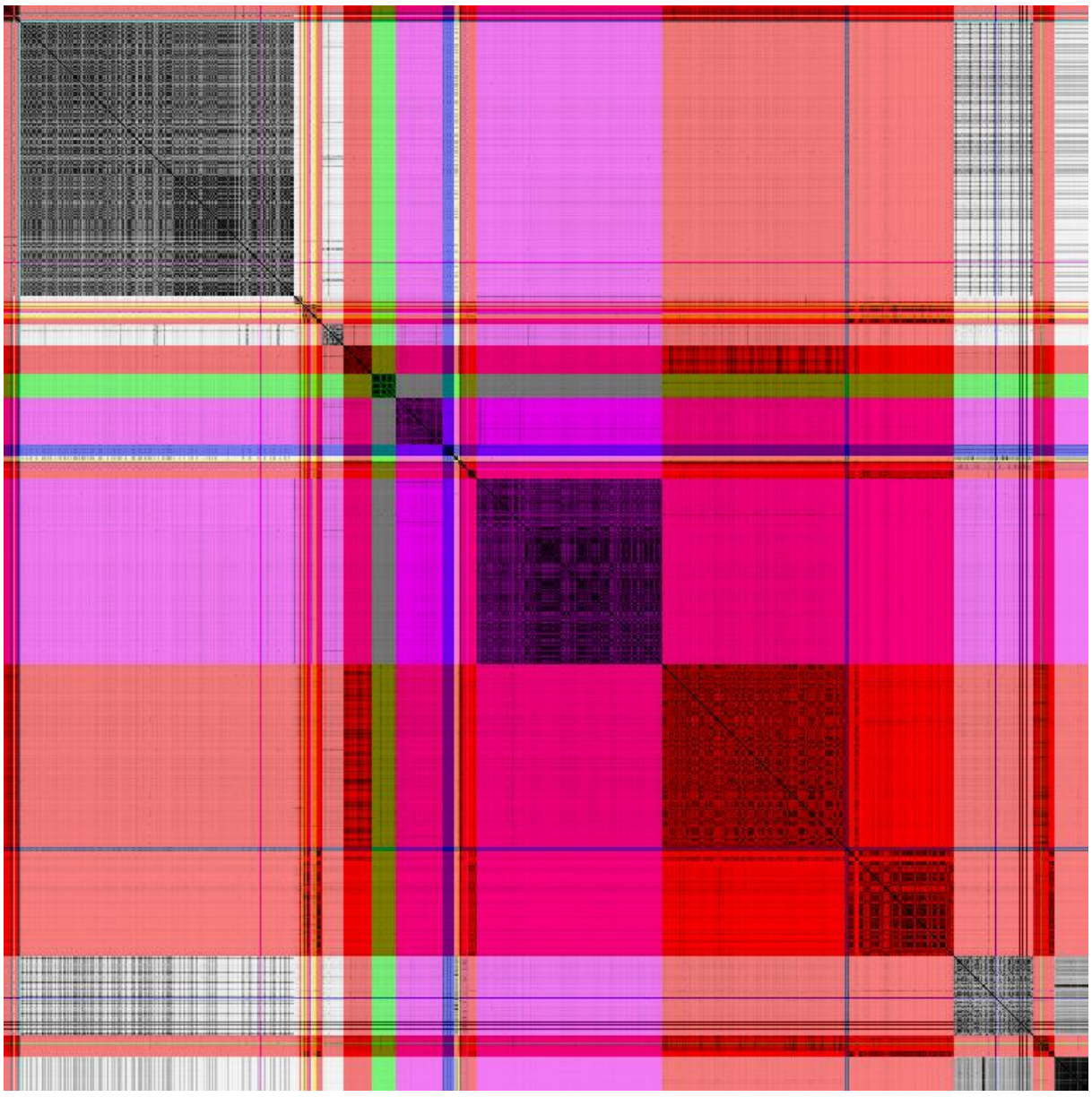


Figure 10: Analysis of similarity in 2004 FST dataset. The x and y axis plot the shots against each other, the color intensity shows the shot dissimilarity calculated with histogram intersection: a darker color means that the shots are more similar. The color of each pixel shows the cluster in which it would be assigned based upon K-means clustering. The ordering of shots in this picture is follows: every shot is grouped with its highest concept, and all concepts are ordered according to the alphabetical list in figure 20.

set all these numbers will have to be available, or constantly being recalculated. This is possible if we use a special computer for this, but since this problem will only get worse when the number of shots increases it is not a solution. For this reason K-means was selected as the clustering algorithm, since it scales much better. K-means scaling is relatively efficient: $O(tkn)$ with n = shots, k = clusters, t = iterations, and usually $k, t << n$

6.4 Forming cluster threads from the created clusters

Now that we have clusters of related shots, we need to add a navigation N to them to create a thread τ. As noted before: this can be done in two ways: based on air time, or based on the closest neighbour algorithm.

Let's look at the first option of sorting threads based on air time. In an experimental setup was created where both a small part of the time thread and a small part of the time-sorted cluster thread are visible for a selected shot.

This resulted in a lot of the same shots visible in both parts of the threads. This is because the shots near the current shot in the time thread have a high likelihood to be in the same cluster thread as the current shot. For example: a summary broadcast of a tennis match usually contains multiple shots after each other. The "tennis" cluster thread also contains these shots, and when the cluster thread is ordered by time then the same ordering will appear as in the time thread. Both threads would be identical for a few consecutive shots.

Since duplicated content in the GUI is not efficient and tends to distract the user the closest neighbour sorting algorithm for clusters as described in section 5.5.2 was chosen.

6.5 The prototype system: The SphereBrowser

Any visualization must at least fulfill the following requirements:

- The interface must be intuitive.
- The interface must be fast. A GUI that is slow will hinder the users' feeling for the underlying dataset, and that will result in less than optimal results for interactive search tasks.
- The interface must be easy to navigate without loosing the context of the dataset.
- The interface must not show too much, nor too little.
- The threads must be shown in context, which means in this case that the surrounding shots in a thread must be shown.
- To convey the storyline of a shot the time thread also has to be shown.

In section 3 it was stated that showing the threads as a spiderweb of connected shots, where arrows between shots represent thread-linked shots, would not be a good thing when there are many shots or threads. Figure 11 illustrates this point: as you can see it is not clear what is visible, why links are made as they are, and how this is to be navigated. Also the timeline, represented by the black arrows in the left image, is not clear at all.

To visualize the thread structure a so called SphereBrowser was developed as a part of the MediaMill search engine[18]. The MediaMill system consists of multiple panes, each with a different function. See figure 14. In the first pane the active topic is selected. The second pane allows the user to set up a query. Traditional search is supported by either the ResultBrowser or the CrossBrowser. The GalaxyBrowser shows a similarity based spatial visualization, and uses active learning to automatically

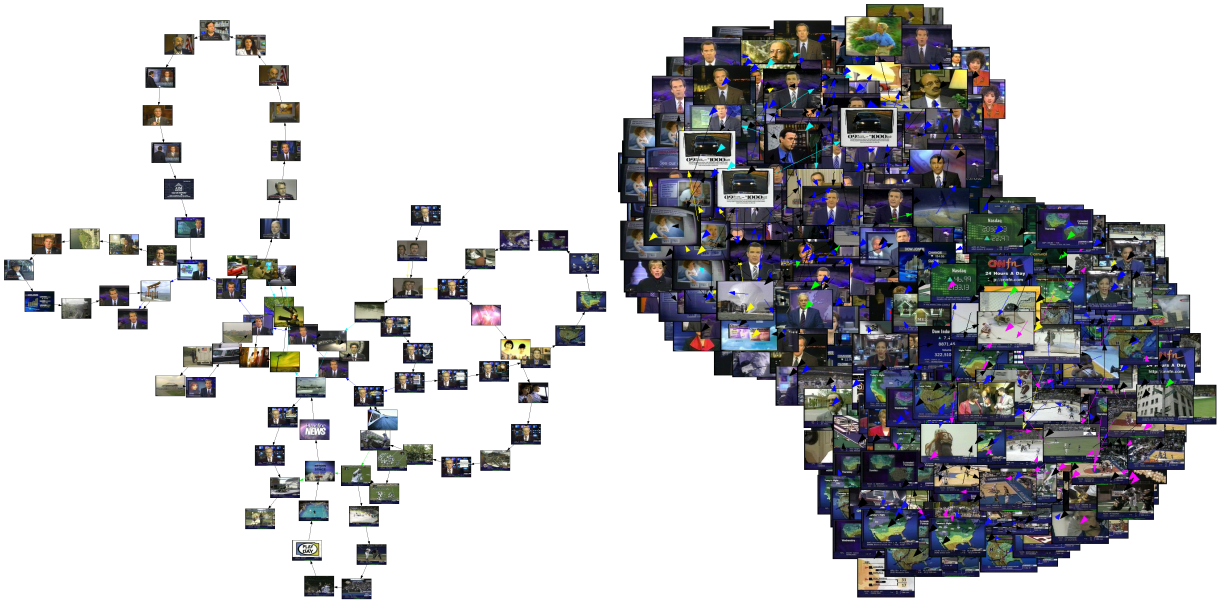


Figure 11: Examples of how threads should not be presented. The left image shows only 3 threads of only the highest 1% of the concepts from the 2004 FST dataset. The right image shows two time threads with a few cluster threads. Each time thread gets twisted into a knot of shots because of all the clusterthread interconnections between both time threads. The visualization has been generated by using graph spring modelling software[17]

Figure 12: Left picture shows a mockup art with the time thread shown horizontal, and relevant parts of the cluster threads shown vertically. The right picture shows how this can be mapped onto a sphere

extend the current selection. On a second monitor a pane is visible with extra frames from the current selected shot from any visualization (figure 15).

The SphereBrowser was developed with the interface requirements stated above in mind. The SphereBrowser GUI gives the user a a spherical layout of nearby shots on the screen, making the center shot bigger than the surrounding shots and thereby giving the user context. When the user jumps from the current shot to any shown shot there is a transition animation so that the browser gives the user the feeling he is looking at one side of a giant turnable sphere of video material. This tries to make the interface more intuitive to use and easy to navigate.

The visualization works as follows. On the horizontal axis the browser shows the time-thread, with related cluster-threads on the vertical axis for each shot. This visualization gives the available context of the current shot to the user. A mockup of this visualization is shown in figure 12. Using the mouse and arrow keys the user can then navigate either through time or through related shots, selecting relevant shots when found. The SphereBrowser also tries to show as much information as possible without leaving the user confused about what he is seeing. For example: the shots pane on the second monitor is updated in real time with extra frames of the center shot to ensure that the user can quickly see other frames of the centered shot, but because this pane is placed on a second monitor this extra information doesn't overlap with the SphereBrowser view itself. Also the browser was written with emphasis on interaction speed, both in datafile loading and in usage. On a modern computer there is no noticeable delay during browsing. The resulting browser is visible in figure 13.

The SphereBrowser is implemented as an extension to the TRECVID search engine, and as such cannot function on its own. The intended use of the SphereBrowser in conjunction with the other parts of the Mediamill search engine is explained in chapter 7.2.

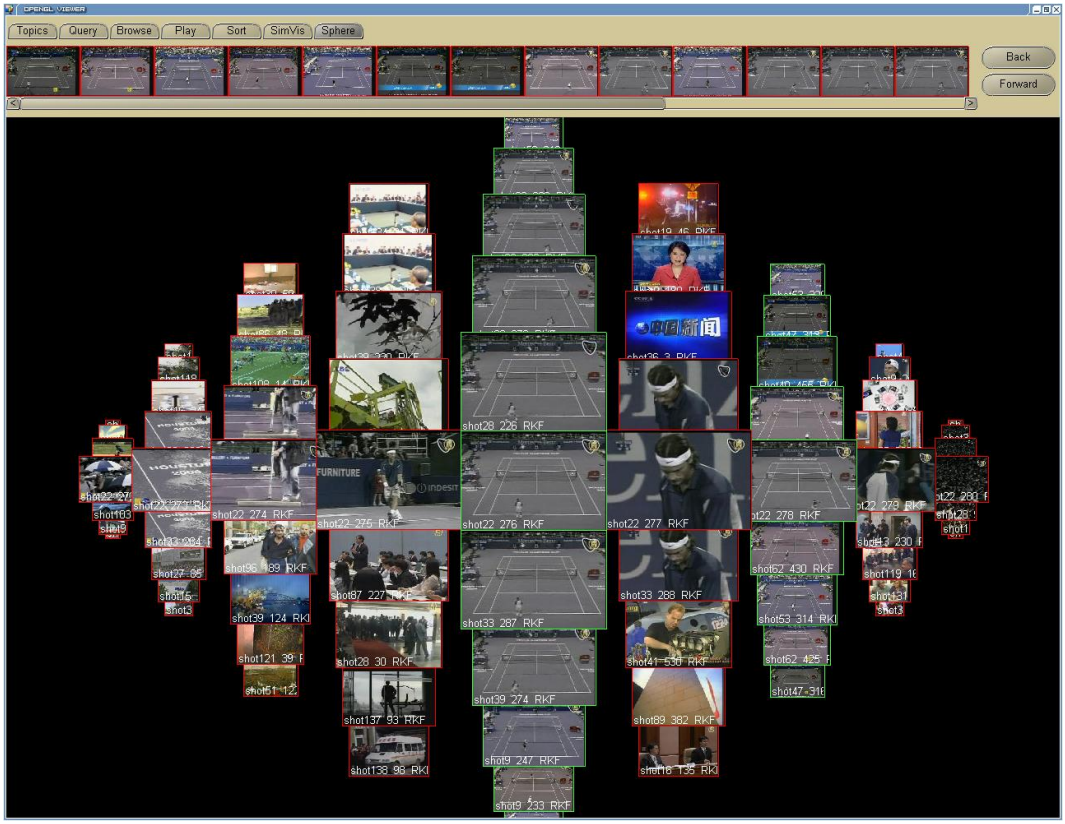


Figure 13: SphereBrowser in action. The top horizontal list shows the current selection. A part of the current time thread is visible in the center, time flows from left to right horizontally. For every shot shown from the time thread a related part from the cluster thread is visible on the vertical axis.

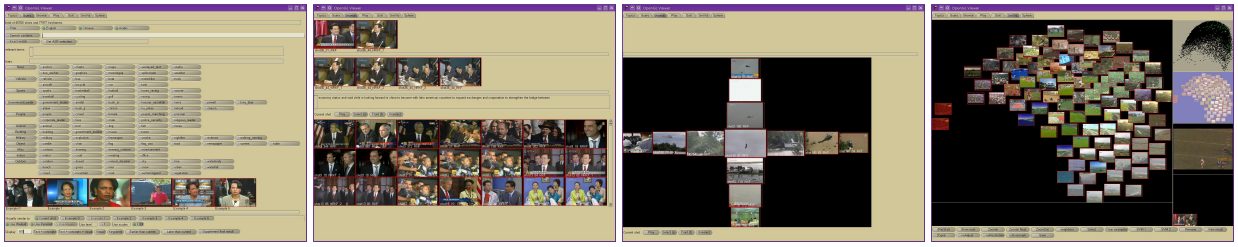


Figure 14: The MediaMill search engine from [18]. From left to right: the query creation pane, the ResultBrowser, the CrossBrowser and the GalaxyBrowser

Figure 15: The MediaMill search system as used during the TRECVID search. The left screen shows the active visualization, in this case the SphereBrowser. The right screen shows frames from the active shot

7 Benchmark evaluation

Now that we have a system to find and display threads in video material to the user, we must evaluate the effectiveness of having these threads. To do this the SphereBrowser system participated in the TRECVID 2005 interactive retrieval benchmark. The purpose of this evaluation is to determine how the SphereBrowser and the accompanying threads perform in comparison with traditional search "*do query - select results*" methods. Also this evaluation gives an impression of how the SphereBrowser will perform in comparison to the current state-of-the-art news video browsing systems.

While this comparison is valid for comparing the SphereBrowser with other systems, the TRECVID evaluation will not directly imply the fitness of the underlying threads themselves. This is because a large part of the SphereBrowser is dependent on both the concept detectors, and the traditional search interface for finding the first results. If we only compare the different MediaMill browser panes the situation would be slightly better, but the number of test subjects is too low, and therefore too influential, to make this comparison.

7.1 TRECVID Video Retrieval track

The NIST TRECVID Video Retrieval track[16] is an independent evaluation track which was created in 2003 from the video evaluation track in the Text REtrieval Conference series. The goal of TRECVID is to promote progress in content-based retrieval from digital video material, by using open, metrics-based evaluation. TRECVID provides an independent evaluation of each participating system. Numerous universities, research institutes and companies participate in TRECVID, with varying degrees of success. An general overview of successful approaches for TRECVID is given in [19]. The TRECVID retrieval track consists of a number of tasks. Each task evaluates a different part of the content-based retrieval problem. The system proposed in this paper is evaluated in the interactive retrieval search task. In the interactive retrieval task the participants are given an objective to find within the entire dataset, and they can use their systems to find as many as possible shots which contain that objective. This process is interactive, so the participant may alter the parameters of their search query during the search. This will typically result in a number of search — select good results iterations. For each topic a fifteen minute time constraint exists from viewing the topic to delivering the results.

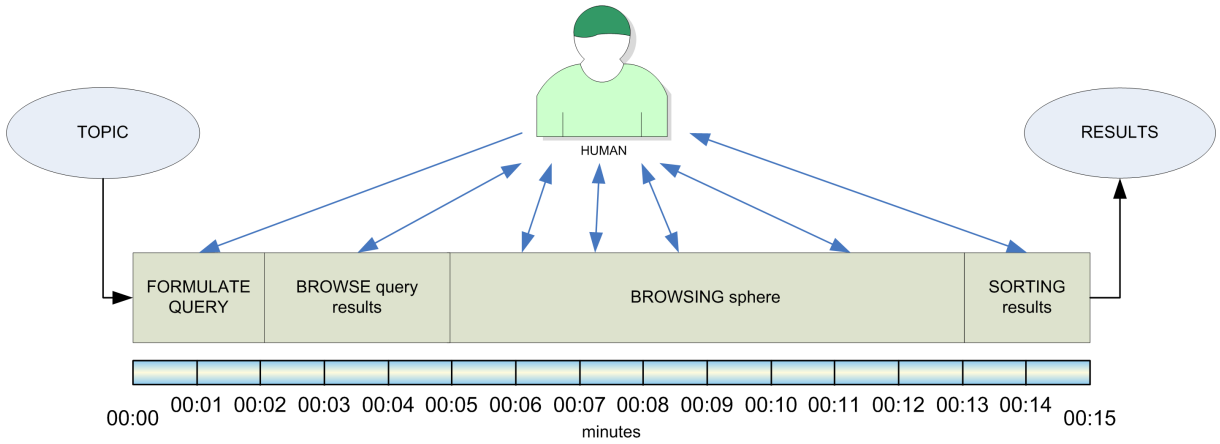
7.2 Proposed method for searching with the SphereBrowser

According to the TRECVID Interactive Retrieval task guidelines a user has fifteen minutes to answer a search query. This time is measured from seeing the search topic to delivering the results. To enable the SphereBrowser to be used in combination with the rest of the system search strategy was formed beforehand. This strategy is described in figure 16. The process is repeated for all topics, with breaks in between.

7.3 Evaluation Analysis

The topics mentioned in this section and the next sections are shortened versions of the actual topic query. A topic such as *Tennis* (156) refers to the query "*Find shots of tennis players on the court - both players visible at same time*". See table 4 for the complete list.

For each topic a set of at least 1000 results has been sent to TRECVID. This number was almost never obtainable with either one of the MediaMill search engines, so the last part was automatically generated by joining the output of the last executed query with the selected results. Since humans tend to only look at the first few results to decide if a set of results has any value it is important to have good results in the top of the list. When good results are placed somewhere in the middle of the



0m	the user learns about the search topic
0m30s	the user configures a search query using the search pane
2m	the user selects a couple of relevant shots in the browse pane. This should be at least 5 shots, but the maximum number depends on the quality of the query results.
5m	for each selected result the immediate vicinity in the SphereBrowser is visited, and relevant shots are added to the selection.
13m	the user sorts through the results and saves the final set

Figure 16: Typical search overview

category	static	dynamic
general	*map of Iraq* (155) *Flames and smoke* (160) *Banners or signs* (161) *Meeting* (163) *Ship or boat* (164) *Basketball* (165) *Palm trees* (166) *Road with cars* (168) *Military vehicles* (169) *Tall building* (170) *Office setting* (172)	*People shaking hands* (157) *Helicopter* (158) *People entering building* (162) *Airplane taking off* (167)
specific	*Condoleeza Rice* (149) *Iyad Allawi* (150) *Omar Karami* (151) *Hu Juntao* (152) *Tony Blair* (153) *Mahmoud Abbas* (154)	*Tennis* (156) *George Bush* (159) *Soccer* (171)

Table 2: Topics for 2005 divided into categories

list, and bad ones are at the top a human would judge the results as bad. Because of this the Average Precision (eq. 6) is used as an indicator of the quality of a resultset. This metric provides a bounded comparable value between 0 and 1. It is sensitive to the entire ranking, one change changes the AP. It is stable: a small change means a small change in AP, and it favors highly ranked relevant shots. The top ranked shots are the most influential in calculating the AP. For every topic the AP value was returned by TRECVID. To provide an indication of the quality of a system the Mean Average Precision (MAP) can be used. This is the averaged value of the APs from all topics.

$$AP = \frac{1}{R}\sum_{i=1}^{A}\frac{R \cap L^i}{i}\lambda(L_i)$$

R: total nr. of relevant shots
A: answer set
$L^i = l_1, l_2, ..., l_i$: ranked version of answer set A
$R \cap L^i$: nr. of relevant shots in the top i of L
$\lambda(l_i) = 1$ if $l_i \in R$, 0 otherwise (6)

To analyse how the SphereBrowser performs for different types of topics a categorization has been made in the topic list. The same categorization scheme as described in [20] and [21] has been used to divide the topics for 2005 into different categories. This categorization divides topics two times. It splits topics in topics about a general subject, and topics about a specific subject. A general topic is a topic which searches for unnamed objects or groups of objects, such as *Road with cars* (168) or *Helicopter* (158). A specific topic explicitly names something which needs to be found, such as *George Bush* (159) or *Condoleeza Rice* (149). Static topics denote topics in which no explicit action happens (*Palm Trees* (166), *Tony Blair* (153)), and dynamic topics explicitly mention an action which must happen. For example: in *Soccer* (171) it is explicitly mentioned that a soccer match must be made. The full categorization for the 2005 topics is listed in table 2.

7.4 Global evaluation results

This section describes the initial evaluation results from the interactive retrieval task. Also a number of findings were made during the search task itself, and these are discussed here. Based upon the topic categorization done above a more detailed analysis of the SphereBrowser performance is done in section 7.5.

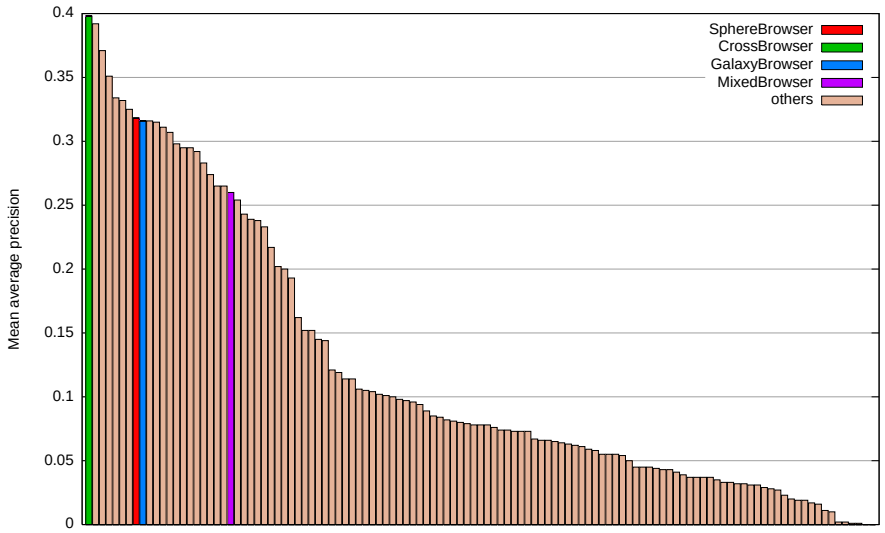


Figure 17: Mean average precision for all topics for all interactive runs from all participating teams. The highlighted bars are the results for each of the MediaMill browsers.

The SphereBrowser performed very well on TRECVID, with a mean average precision (MAP) of 0.331 on all topics. See figure 17 for an indication of how this matches up to the other participating teams. The SphereBrowser is ranked 8th, but as you can see from the figure there at least 4 other contestants who have almost the same mean average precision.

Initial evaluation of the results (see figure 18) reveals two different topic categories. There were topics for which there were multiple cluster threads with good results for that topic, such as *Tennis* (156), *People with banners or signs* (161), *Meeting* (163) and *Tall building* (170). For these topics only the relevant parts of the threads needed to be selected. These types of topics will now be referred to as threadable topics.

Another selection method was found in queries such as *Airplane takeoff* (167) and *Office setting* (172). Here there were only a limited number of consecutive valid shots visible in each thread, but because of the combination of both time and cluster threads in the SphereBrowser there was often another valid but not yet selected shot visible. For these queries selection was done by hopping as fast as possible from one valid result to another. Lets refer to these topics as hoppable topic.

Also a number of topics were not answerable by the SphereBrowser at all because of lack of nearby valid shots in any thread. These include all person X topics, such as *Omar Karami* (151), *Tony Blair* (153) and *Condoleeza Rice* (149). For these topics the time spent in the normal search/result browser had to be extended before results could be found.

During the actual search task a number of findings were made. These are discussed here:

- The animation between transitions is good for providing feedback of what is happening with the system, but when the system is used in a search it tends to slow the system down too much. This is especially true for hoppable topics,

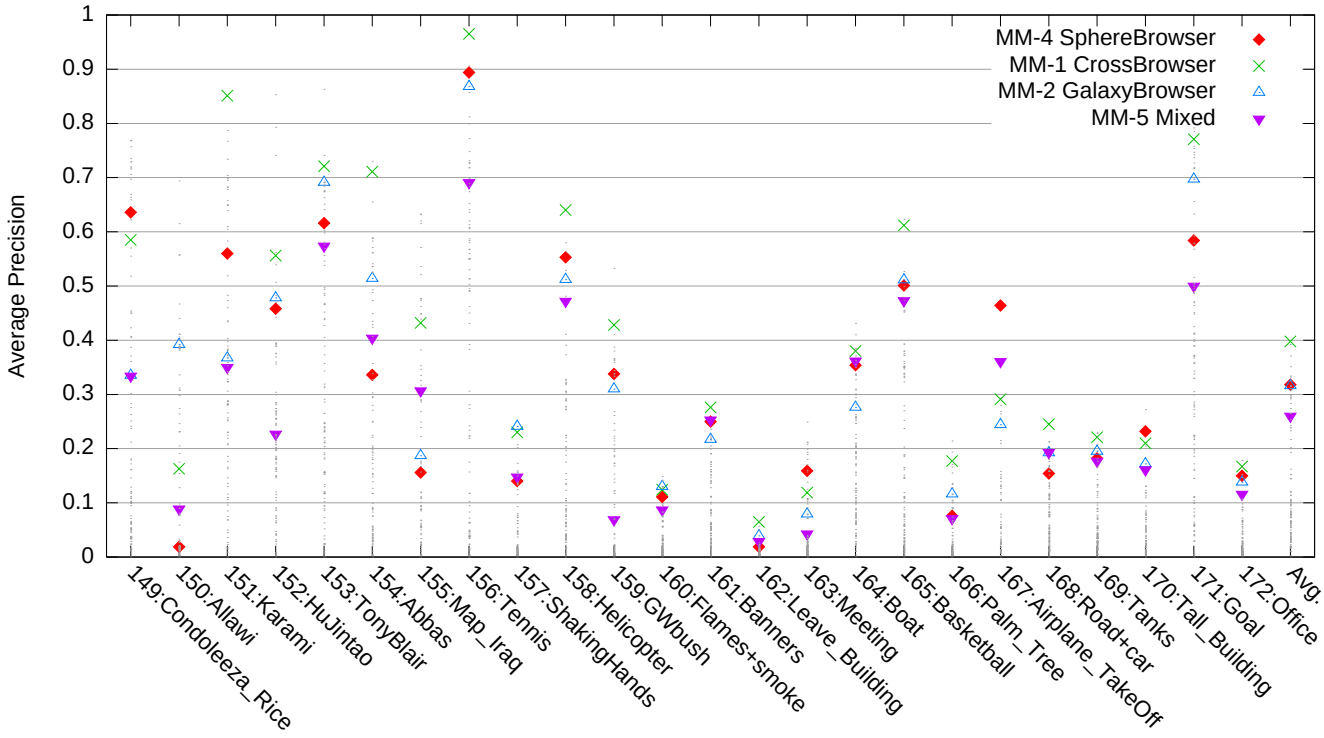


Figure 18: Average Precision compared per topic for the MediaMill runs. Gray dots denote scores of other teams

because the number of transitions is the highest here.

- For some topics the number of selected results with traditional browsing were exhausted in the SphereBrowser without finding enough new leads to follow up on. This especially happened with the person X type queries. In this case a switch back to the traditional browser had to be made to select more valid results. This is a problem because switching between the SphereBrowser view and the normal browsing view is very expensive in time: the user takes a lot of time to adapt to the different interfaces. Also the selected shots in the SphereBrowser are not removed from the traditional search pane, which leads to a lot of duplicate selection attempts. The opposite also holds: when there are a lot of good results in both the traditional browser and in the SphereBrowser continuously switching between both panes to get the best of both worlds is not good either. A decision has to be made based upon a quick initial view of both resultsets as to which browser to stick to and exhaust all the possibilities, even when there seem to be no more.
- For threadable topics the bulk of the selected results was often selected in just a few minutes. For example *Tennis* (156) only took 4 minutes, 2 of which were spent setting up the initial query and selecting a few results, the remaining two were used in selecting very large consecutive portions of a single thread which consisted of mostly tennis. However because TRECVID does not yet take unused minutes into account during the evaluation the remaining time was used to try to find more results in other threads.
- Finally the SphereBrowser works better for queries for which traditional search does only have a limited set of results. This especially happens with hoppable topics. Because of the hopping effect the SphereBrowser keeps on finding relevant shots, while in the traditional browser the query has to be altered and reprocessed to get more results, and this takes more time.

7.5 Evaluation results per topic category

run		topic category				average
		general static	general dynamic	specific static	specific dynamic	
MM-1 CrossBrowser	MAP	0.269	0.306	0.598	0.721	0.414
MM-2 GalaxyBrowser	MAP	0.201	0.259	0.462	0.525	0.329
MM-5 MixedBrowser	MAP	0.203	0.253	0.330	0.420	0.270
MM average	MAP	0.221	0.278	0.457	0.593	0.336
all interactive runs averaged	MAP	0.134	0.117	0.356	0.383	0.218
MM-4 SphereBrowser	MAP	0.211	0.294	0.438	0.605	0.331
compared ranking R		9th	2nd	23th	5th	8th
user feeling		good	good	bad	good	good
most common browse method		hopping	both	hopping	threading	both

Table 3: Topic category evaluation

When the topics are split up into categories we see the following:

- The scores for specific topics are on average much higher than of those for general topics.
- The SphereBrowser performed badly for Specific Static topics, $R = 23$. This can be explained because most of these topics are ”Find person X” topics, and a named person usually gets named in the audio track or his name is visible on screen. This means that keyword searches for the named person will retrieve shots with this person, whereas

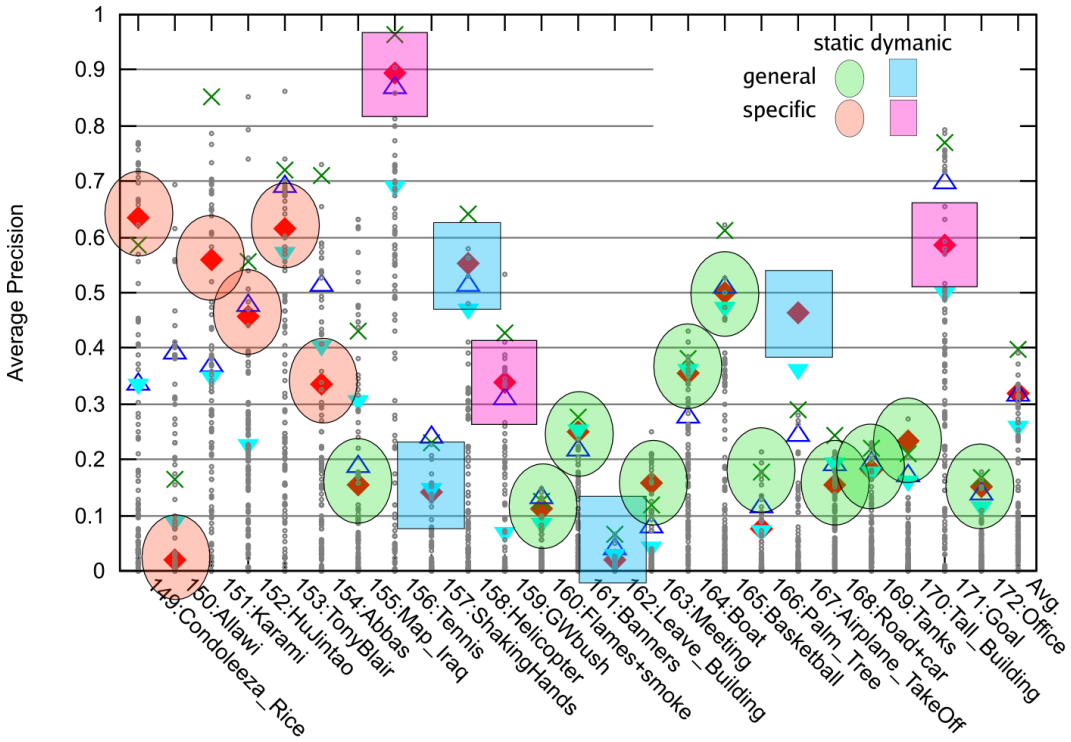


Figure 19: Average Precision compared per topic for the MediaMill runs. The markings denote the topic category as listed in 2.

the SphereBrowser can only rely on the effectiveness of the underlying treads and thereby the effectiveness of the underlying concept detectors.

- General Dynamic topics seem to be the strong point of the SphereBrowser, $R = 2$. This is odd since the averaged MAP of all interactive runs indicates that this is the worst performing category of topics. This is not significant however since there are only four general dynamic topics. If we look at general topics as a whole then we see that the SphereBrowser performs better on generic topics than most other systems. This can be explained by looking at the general topics themselves. A lot of generic topics are difficult to describe with keywords. For example: searching with the keywords "*palm tree*" will not work, since palm trees are rarely the center of attention, and thus are rarely mentionned explicitly. They usually are accidentaly visible in the shot. Because of this systems have to rely on their visual similarity detectors and vegetation detectors for finding palm trees. The SphereBrowser has an advantage here because hopping though threads does not need keywords at all, and the concept detectors do their work very well.

8 Conclusion

The SphereBrowser performs quite well in the TRECVID benchmark. However it is currently undetermined to what extend this is caused by the quality of the underlying concept detectors. The MediaMill teams all received high scores: this shows the clear advantage of using the MediaMill concept detectors. However because every interface wasn't extensively tested, every pane was tested by its own developer, comparisons between the interfaces lead to comparisons between the users. For a more objective evaluation it would be good to test with multiple users per browse pane. Also the underlying threads work, but they are definitely not optimal. More extensive research into cluster threads is necessary to optimize them. For visually similar topics entire threads can be relevant and are quickly selected. For generic topics the SphereBrowser performs very well, since the SphereBrowser it isn't dependent on keyword searching to be able to find valid results. Most generic topics require a method from jumping from a valid result to some other valid result. In other systems this is often done by finding visually similar shots and selecting them. The SphereBrowser allows to select conceptually similar scenes from the time neighbourhood of each valid result. This increases the number of found results enormeously. However there is still much work to be done before the SphereBrowser will actually bridge the semantic gap.

8.1 Future research

This section provides a list of things which need attention if the SphereBrowser is to be used in next years TRECVID competition.

Clustering and threading engine

The clustering and threading engine performs adequately, though only shots of visually similar topics actually get contained in a thread of their own. This should be extended to generic topics so more threading and less hopping is required. A solution for Person X topics must be developed. A possible solution for this is to extend the number of different threads to other modalities than time and semantic concepts. Currently every shot is placed within a time thread, and within a concept-detector based cluster thread. If we can extend this so that every shot is also placed in, for example, a text-thread, a person-thread, or a visual-thread, then the user has more control over which threads to see. For example: person X topics would use the person-thread. Here is a list of possible threads in other modalities:

- **Text threads:** threads created from the available text of every shot. Shots with very specific keywords, like the name of a person, get joined together in a thread. A problem here is that not every shot has text, and text can be quite worthless when shots originate from an other language than English. In combination with the other modalities this is however not a problem: where the text is good we should use it. Where the text is bad we can use something else.
- **Person threads:** a special version of the text thread in which a named entity detector retrieves the names of people from the text. A variant of this would be the **organization** and **location** thread.
- **Visual threads:** threads created from visual information only, this would provide an even better visually similar browse method for every shot. Currently the MediaMill engine does not have an easy to use visual similarity browser, and this could be it.
- **Auditory threads:** threads created from typical sounds in shots, such as cheering, explosions, a certain voice, or music.

SphereBrowser

The SphereBrowser itself also can be improved upon. The list below gives a few suggestions for this.

- Time-based logging capability to determine when which result was selected from which thread at which time. This metric can be used to determine the fitness of threads, and also give an indication of how much time is actually required for certain topics.
- The different search interfaces have to look more like each other in terms of extra windows and button mappings. The current differences lead to a long adaptation time every time the user switches from one pane to another. Also the already selected items in another pane should be completely removed from the current pane results, not just only colored differently. This is especially true when switching from the SphereBrowser to the CrossBrowser or the ResultsBrowser.
- The current SphereBrowser visualization only gives a mockup of a sphere. Further research is needed to determine if a real spherical view would give the user a better feeling for the dataset.

8.2 Final words

Semantic threads allow users to find results when normal keyword searching isn't sufficient. In combination with the SphereBrowser this can be done quickly and efficiently.

References

[1] "The topic detection and tracking evaluation project (tdt-2004)," http://www.nist.gov/speech/tests/tdt/tdt2004/.

[2] Arnold W.M. Smeulders, Marcel Worring, Simone Santini, Amarnath Gupta, and Ramesh Jain, "Content based image retrieval at the end of the early years," in *IEEE Transactions on Pattern Analysis and Machine Intelligence*, December 2000, vol. 22, pp. 1349–1380.

[3] J. Allan, S. Harding, D. Fisher, A. Bolivar, S. Guzman-Lara, and P. Amstutz, "Taking topic detection from evaluation to practice," in *Proceedings of the 38th Hawaii International Conference on System Sciences*, Amherst, MA, USA, 2005.

[4] John. R. Kender and Milind R. Naphade, "Ontology design for video using semantic threads," in *IEEE ICME*, Amsterdam, The Netherlands, 2005.

[5] Chaomei Chen, "Bridging the gap: The use of pathfinder networks in visual navigation," *JVLC*, vol. 9, no. 3, pp. 267–286, 1998.

[6] Giang P. Nguyen and Marcel Worring, "Similarity based visualization of image collections," in *Seventh DELOS on audio-visual content and information visualization in digital libraries*, May 2005.

[7] Giang P. Nguyen and Marcel Worring, "Scenario optimization for interactive category search," in *MIR '05: Proceedings of the 7th ACM SIGMM international workshop on Multimedia information retrieval*, New York, NY, USA, 2005, pp. 159–166, ACM Press.

[8] M. Rautianen, T. Ojala, and T. Seppänen, "Cluster-temporal video browsing with semantic filtering," in *Proc. Advanced Concepts for Intelligent Vision Systems*, Belgium, 2003, pp. 116 – 123.

[9] M. Rautianen, T. Ojala, and T. Seppänen, "Cluster-temporal browsing of large news video databases," in *IEEE ICME*, Taipei, Taiwan, 2004, pp. 2:751–754.

[10] D. Heesch, P. Howarth, J. Magalhães, A. May, M. Pickering, A. Yavlinksy, and S. Rüger, "Video retrieval using search and browsing," in *Proc. TRECVid*, Gaithersburg, MD, 2004.

[11] "Text retrieval conference," http://trec.nist.gov/.

[12] Cees G. M. Snoek, Marcel Worring, Jan-Mark Geusebroek, Dennis C. Koelma, Frank J. Seinstra, and Arnold W. M. Smeulders, "The semantic pathfinder: Using an authoring metaphor for generic multimedia indexing," *IEEE TPAMI*, vol. 28, no. 10, pp. 1678–1689, 2006.

[13] F.J. Seinstra, C.G.M. Snoek, D. Koelma, J. Geusebroek, and M. Worring, "User transparent parallel processing of the 2004 nist trecvid data set," in *Proceedings of the 19th International Parallel and Distributed Processing Symposium*, Denver, USA, April 2005.

[14] "Wordnet," http://wordnet.princeton.edu/.

[15] M.J. Swain and D.H. Ballard, "Color indexing," *International Journal of Computer Vision*, vol. 7, no. 1, pp. 11–32, 1991.

[16] "Nist trecvid video retrieval evaluation," http://www-nlpir.nist.gov/projects/trecvid/.

[17] "Graphviz," http://www.graphviz.org/.

[18] Cees G. M. Snoek, Marcel Worring, Jan van Gemert, Jan-Mark Geusebroek, Dennis Koelma, Giang P. Nguyen, Ork de Rooij, and Frank Seinstra, "Mediamill: exploring news video archives based on learned semantics," in *ACM Multimedia*, New York, NY, USA, 2005, pp. 225–226.

[19] A.G. Hauptmann and M.G. Christel, "Successful approaches in the trec video retrieval evaluations," in *Proceedings of ACM Multimedia*, New York, USA, Oktober 10-16 2004, pp. 668 – 675.

[20] Laura Hollink, Giang P. Nguyen, Dennis Koelma, A. Th. Schreiber, and Marcel Worring, "User strategies in video retrieval: a case study," in *Proceedings of the International Conference on Image and Video Retrieval (CIVR)*, Dublin, Ireland, July 2004.

[21] Laura Hollink, A. Th. Schreiber, B. Wielinga, and Marcel Worring, "Classification of user image descriptions," in *International Journal of Human Computer Studies*, 2004, pp. 601–626.

A TRECVID Topics 2005

Nr	Topic question
0149	Find shots of Condoleeza Rice
0150	Find shots of Iyad Allawi, the former prime minister of Iraq
0151	Find shots of Omar Karami, the former prime minister of Lebannon
0152	Find shots of Hu Jintao, president of the People's Republic of China
0153	Find shots of Tony Blair
0154	Find shots of Mahmoud Abbas, also known as Abu Mazen, prime minister of the Palestinian Authority
0155	Find shots of a graphic map of Iraq, location of Bagdhad marked - not a weather map,
0156	Find shots of tennis players on the court - both players visible at same time
0157	Find shots of people shaking hands
0158	Find shots of a helicopter in flight
0159	Find shots of George W. Bush entering or leaving a vehicle (e.g., car, van, airplane, helicopter, etc) (he and vehicle both visible at the same time)
0160	Find shots of something (e.g., vehicle, aircraft, building, etc) on fire with flames and smoke visible
0161	Find shots of people with banners or signs
0162	Find shots of one or more people entering or leaving a building
0163	Find shots of a meeting with a large table and more than two people
0164	Find shots of a ship or boat
0165	Find shots of basketball players on the court
0166	Find shots of one or more palm trees
0167	Find shots of an airplane taking off
0168	Find shots of a road with one or more cars
0169	Find shots of one or more tanks or other military vehicles
0170	Find shots of a tall building (with more than 5 floors above the ground)
0171	Find shots of a goal being made in a soccer match
0172	Find shots of an office setting, i.e., one or more desks/tables and one or more computers and one or more people

Table 4: Topics for TRECVID 2005 benchmark

B Available concept detectors for 2004

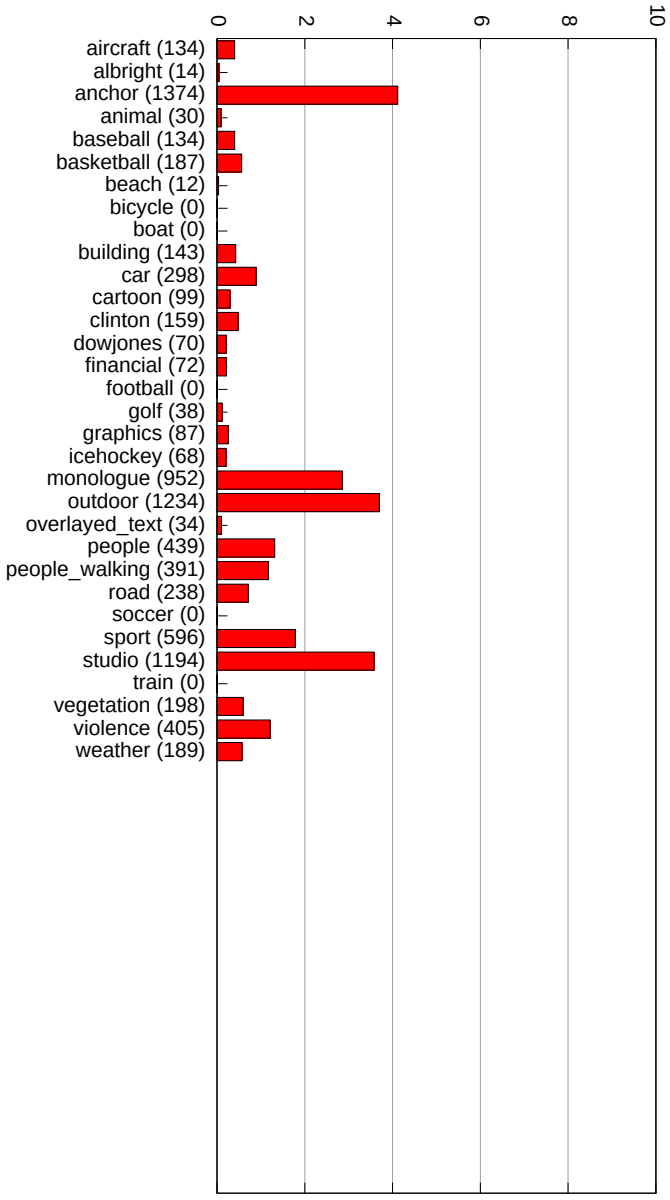


Figure 20: Concepts in 2004 FST dataset. For each concept the % of shots (from the total 33367 shots) that contain that concept is also shown. For this graph the treshold is set at 0.5. Note that the scale only goes to 10%

C Available concept detectors for 2005

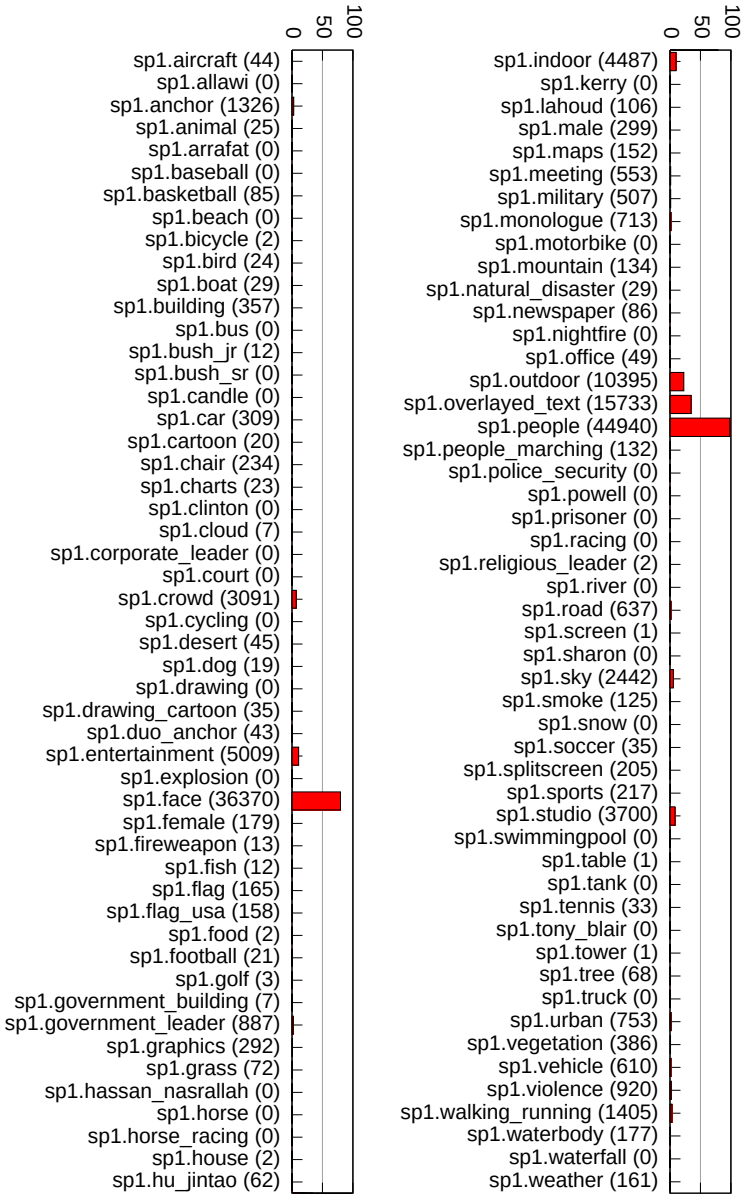


Figure 21: Concepts in 2005 FST dataset. For each concept the % of shots (from the total of 45765 shots) that contain that concept is also shown. For this graph the threshold is set at 0.5