Adaptive Clustering and Interactive Visualizations to Support the Selection of Video Clips

Andreas Girgensohn[1], Frank Shipman[2], Lynn Wilcox[1]

[1]FX Palo Alto Laboratory, Inc.
3400 Hillview Avenue
Palo Alto, CA 94304, USA

[2]Department of Computer Science &
Center for the Study of Digital Libraries
Texas A&M University
College Station, TX 77843-3112

{andreasg, wilcox}@fxpal.com, shipman@cs.tamu.edu

ABSTRACT

Although people are capturing more video with their mobile phones, digital cameras, and other devices, they rarely watch all that video. More commonly, users extract a still image from the video to print or a short clip to share with others. We created a novel interface for browsing through a video keyframe hierarchy to find frames or clips. The interface is shown to be more efficient than scrolling linearly through all keyframes. We developed algorithms for selecting quality keyframes and for clustering keyframes hierarchically. At each level of the hierarchy, a single representative keyframe from each cluster is shown. Users can drill down into the most promising cluster and view representative keyframes for the sub-clusters. Our clustering algorithms optimize for short navigation paths to the desired keyframe. A single keyframe is located using a non-temporal clustering algorithm. A video clip is located using one of two temporal clustering algorithms. We evaluated the clustering algorithms using a simulated search task. User feedback provided us with valuable suggestions for improvements to our system.

Categories and Subject Descriptors

H.3.3 [**Information Storage and Retrieval**]: Information Search and Retrieval — *clustering, selection process*. H.5 [**Information Interfaces and Presentation**]: H.5.1 Multimedia Information Systems — *video*. H.5.2 User Interfaces — *GUI*.

General Terms

Algorithms, Design, Human Factors.

Keywords

Video browsing, keyframe selection, adaptive clustering.

1. INTRODUCTION

With the widespread availability of small video recorders in mobile phones, digital cameras, and dedicated camcorders such as Flip Video, user-generated video content is on the rise. The higher resolution and quality of these recorded videos permit additional uses such as the extraction of still images. However, it is difficult to find the best image, or keyframe in the video to print, or the best video clip to share with friends.

The problem of finding a single keyframe or short video clip is magnified for long videos. While fast-forwarding permits moving quickly through a video, it is not efficient for directed skimming or browsing. Timeline sliders [14] that show selected keyframes along a timeline are good for short videos but they do not scale well because a slider's resolution breaks down for long videos. A zoomable timeline can make this easier [7] but it is still difficult to find a particular portion of a video just by scrubbing. Furthermore, scrubbing does not work well for streaming video.

We designed a keyframe-based interface for browsing video on devices with a variety of screen sizes. The interface provides fluid access to different parts of the video by presenting a keyframe hierarchy along a timeline that allows users to quickly zoom in on an area of interest in the video. The algorithm for creating the hierarchy of keyframes adapts both to the user's task and to the repetitiveness of the video.

If the user specifies that he is looking for a single image in the video, the interface is based on a non-temporal visual clustering algorithm that groups keyframes by similarity. If the user is looking for a video clip, the system first analyzes the video to determine whether it is visually repetitive or non-repetitive. Then the system chooses among two temporal-visual clustering algorithms that group the keyframes based on visual similarity while ensuring that the temporal order of the keyframes is preserved and that navigation paths are kept short. User knowledge of the video can be used to locate the desired content in a narrower time interval. The default task is assumed to be finding a video clip because browsing via visual similarity alone is inappropriate for users requiring temporal information.

To evaluate the system we established several metrics to objectively compare the suitability of different algorithms for tasks that users want to perform. We then used a simulated search task to measure the performance.

The next section discusses related work in keyframe selection and browsing interfaces. After this, we present a user interface for navigating video followed by a description of techniques for keyframe selection and video segment clustering. We then evaluate the clustering algorithms and discuss feedback comparing the use of the new interfaces to a scrubbing-based interface. We conclude with a discussion of future work.

2. RELATED WORK

Systems offering keyframe-based interfaces for video access are widespread. They use (1) algorithms for segmenting a video into smaller, meaningful pieces, (2) algorithms for identifying important keyframes from videos and video segments, and (3) visualizations for summarizing video through storyboarding, compositing keyframes, and selective playback. While there are other systems supporting user-generated or home video, the majority of video browsing interfaces are geared towards professionally produced video.

The system most closely related is our own work on the Hitchcock video editor [5]. It displayed piles of keyframes representing video

ICMR'11, April 17-20, Trento, Italy

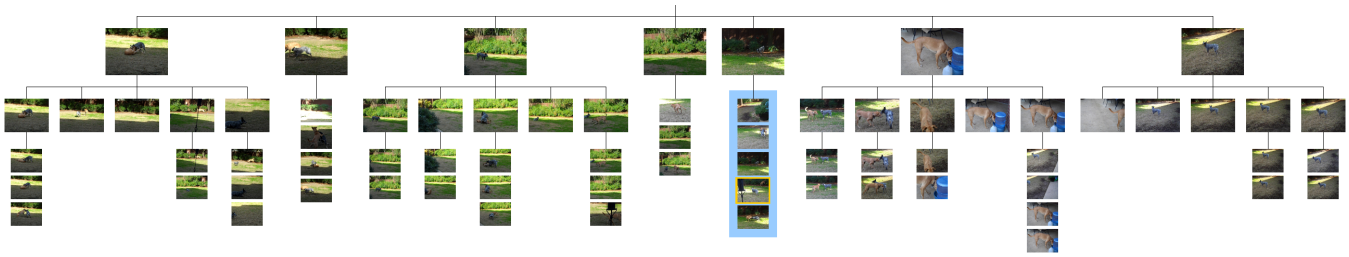

Figure 1. Navigation Tree for a Video.

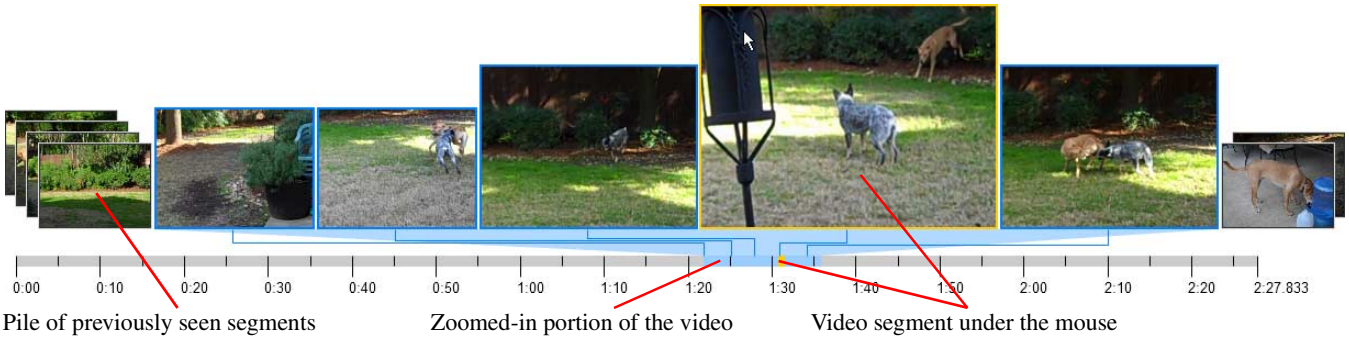

Figure 2. Zoomed-in View of a Segment of a Video with Context piles on the Sides.

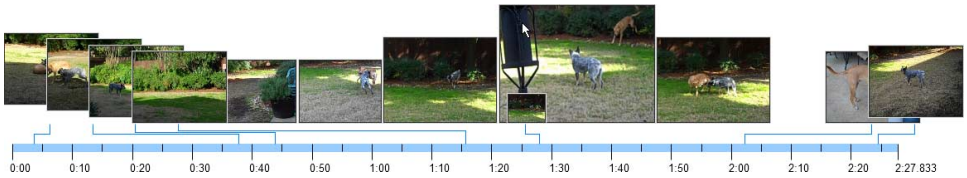

Figure 3. Animation while Zooming out.

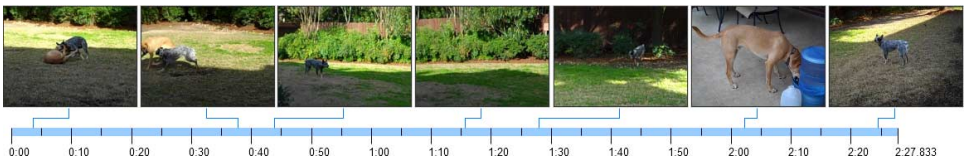

Figure 4. Top-level View of a Video.

sub-shots clustered by visual similarity, either with or without temporal order, without attempting to balance the cluster tree. Users preferred piles in temporal order but disliked deeply nested piles.

Approaches for selecting keyframes frequently segment videos into scenes, shots, and sub-shots, representing either shots or sub-shots with keyframes. Ciocca et al. [2] filter out low-quality and repeated keyframes using pictorial features and a visual attention model before creating a cluster-based hierarchy. Jiang et al. [8] find representative keyframes via an independent component analysis and create a keyframe hierarchy via agglomerative clustering. Ngo et al. [10] cluster video shots by a two-level hierarchy of similar motion and color features to support the retrieval of sports video. Oh et al. [11] present a hierarchical clustering approach that groups shots into a hierarchy of scenes using simple visual features. Guillemot et al. [6] create a three-level hierarchy for a video and found that study participants could locate video frames three times faster than with a streaming video player. Rui et al. [12] describe a similar approach and represent scenes and groups of shots with keyframes in their user interface. Schoeffmann et al. [13] visualize low-level features and frame surrogates along several timelines, enabling users to quickly recognize similar segments. Luo et al. [9] infer the camera operator's intent to select semantically meaningful keyframes. None of these approaches attempt to balance tree structures. Others have combined clustering with balanced trees [1, 4, 17, 18] but only for top-down clustering algorithms and without maintaining temporal order.

In contrast to our work, none of the other approaches attempt to restrict the depth of a cluster tree of visually similar keyframes in temporal order. They either present an uneven natural clustering, present a fixed number of levels with an unlimited number of top-level nodes, or balance a visual cluster tree without maintaining temporal order. Such approaches are included in our evaluation of cluster algorithms below.

Our clustering techniques are very different from standard agglomerative techniques and entail much more than changing a few variables. The combination of temporal and visual aspects in balanced cluster trees has broad applications for video user interfaces. We found that different types of algorithms work best for different types of videos.

3. KEYFRAME BROWSING

We created a user interface that presents a video via hierarchically-organized keyframes. Users can use the keyframes to navigate to an interesting part of the video and then use neighboring keyframes and a timeline to explore that part of the video. In contrast, a large grid of all keyframes would not scale to longer videos with many keyframes, especially when kept to the same screen area. In the evaluation section, we compare a scrollable keyframe grid to the tree representations.

The interface employs a hierarchical cluster tree such as the one shown in Figure 1. The cluster tree is not shown to the user be-

cause its tiny keyframes make it unsuitable for exploration in a constrained display size. Instead, different visualization techniques are used to support the navigation of the tree. In our design of such a navigation interface, the main goals were to make it easy to navigate up and down the tree while providing context for the user's current location in the tree. Thus, as the user navigates through the tree, the keyframes representing segments at higher levels are shrunk and stacked into piles to make room for the new segments. We reduce the tree branching factor below the top level to make the most of the space available at the top level while providing context during the user's descent into the tree.

The visualization shown in Figure 2 consists of a series of keyframes selected as cluster representatives. They are placed above a timeline of the video showing the position and length of the corresponding video segments in the context of the overall video. The zoomed-in segment of Figure 2 is highlighted in Figure 1. When the user moves deeper into the tree, the keyframes of neighboring clusters will form piles at the two ends of the display. The side piles in Figure 2 show the additional six keyframes from the top-level display. Users can navigate horizontally in the tree, moving to sibling sub-clusters, by clicking on the keyframes in these piles. Figures 3 and 4 show subsequent states as the user zooms out from the state depicted in Figure 2. Figure 3 shows the animation of the central keyframes disappearing and keyframes from the side piles sliding in to replace them. Figure 4 shows the top-level view of the video. Overall, this approach provides a visualization somewhat akin to [15] but with a more condensed footprint and without requiring space for all of the keyframes simultaneously.

There are several visual cues that indicate the current state of the interface to the user. Each keyframe is connected with a line to the timeline to indicate its position in the video. The part of the timeline corresponding to the currently shown cluster is marked in blue. For longer videos, a fish-eye view for the timeline would be beneficial where the blue portion would use a larger share of the overall width. A trapezoid connects the timeline segment to the keyframes. When hovering with the mouse over a keyframe, the keyframe and the timeline portion corresponding to the video segment at that time are highlighted in yellow.

A mouse wheel facilitates navigation up and down the segment tree. This exploits the user's familiarity with using a mouse wheel for zooming in on-line maps, virtual worlds and other environments. Transitions are animated showing keyframes moving to make space for new keyframes to help the user understand the relationships between the pre- and post-navigation states of the interface. When the mouse moves over a keyframe, it and its neighbors are enlarged to provide the user a more detailed view while more distant keyframes are shrunk to make room. As this occurs often and the change in the visualization is not confusing, it is animated over ¼ second. In contrast, the animations where keyframes for segments are being added or removed (e.g., Figure 3) are more likely to result in user confusion and are animated over ¾ second. Shorter animations use linear speed while longer animations use linear acceleration to produce a more natural feel.

The user interface is implemented for web browsers. The extracted keyframes can also be used for streaming video services such as YouTube where scrubbing would not work. Image animation is implemented via JavaScript manipulation of the Document Object Model. The timeline is drawn in an HTML5 Canvas so that the lines and the trapezoid shown in Figure 2 can be animated while images change size and position. Videos are processed during check-in and relevant data such as the clip boundaries and the distance matrix between keyframes are stored in a data file. This allows for the fast creation of cluster trees with different branching factors and algorithms in response to the user resizing the browser window. The segmentation tree is sent to the browser as a JSON document.

4. SEGMENTATION AND CLUSTERING

When selecting images or clips from a video, it is important to be able to efficiently browse through the video. A typical approach is to segment video into shots and to show shot images. Usually, shots assume that frames are similar. Other researchers have created hierarchies of scenes, shots, and sub-shots [6, 12]. Underlying many such approaches is an infrastructure to segment a video and to create a cluster tree of the segments based on similarity. While there are a wide variety of different similarity models, many create highly unbalanced cluster trees due to the distribution of differences (inverse of computed similarity) that they generate.

Such clusters are appropriate for many automated uses but are problematic for interactive uses. They require, on average, much more navigation to find a specific keyframe than is required for a more balanced tree. We have explored the use of balanced tree algorithms to provide access to the video segments. While such balanced trees reduce the navigational distance required to access a video, there is the danger of creating less intuitive clusters of video content. As a result, more backtracking through the tree is required to navigate the video.

4.1 Segmentation and Keyframe Selection

Our techniques are intended for single camera takes and not for production video with transitions. While video takes are usually only a few minutes long, longer takes would only lead to deeper hierarchies in our system but not affect the quality otherwise.

For finding video segment boundaries, we use a 10-second wide Gaussian checkerboard kernel that we move across the self-similarity matrix of frames in the video sampled at 5 fps [3]. The self-similarity matrix is composed of the color correlogram similarities between frames. The amount of correlation with the kernel provides the strength of a segment boundary. Color correlograms can detect changes in a scene even if the camera does not move so that our approach is suitable for a video camera on a tripod. To ensure a fairly uniform temporal distribution of keyframes, we pick them independent of segment boundaries and at a lower sampling rate. We select the strongest boundary between every pair of adjacent keyframes as the distance for the temporal clustering algorithms.

Home videos frequently include motion blur that lasts for several frames. When selecting keyframes, our algorithm inspects all frames in the video and picks the least blurry frames at an average rate of 2 fps. We adapted an approach for determining the blurriness of a photo [16] using the strength of the top 10% edges and the entropy of the edge histogram. This techniques avoids blurry frames. In addition, we skip keyframes that are too dark.

4.2 Clustering of Keyframes

For browsing video along a timeline, it makes sense to combine only temporally adjacent video clips. In contrast, a cluster hierarchy without a temporal constraint produces clusters containing keyframes from different parts of the timeline. While such a hierarchy can produce good results for locating a single image in the video, it does not support locating a short clip containing several keyframes or browsing the whole video.

For clustering keyframes, we started with looking at three extremes. First, we used complete-link hierarchical agglomerative clustering[1] solely based on visual similarity without consideration of temporal order (*visual*, Figure 8). This approach is most similar to those described in the literature. Second, we clustered by visual similarity while only merging temporally adjacent clusters (*tempo-*

[1] http://nlp.stanford.edu/IR-book/completelink.html

ral-visual, Figure 7) as in [5]. Unfortunately, the requirement of keeping keyframes in temporal order while maintaining visual similarity can cause very unbalanced cluster trees. Third, we created a complete tree[1] in temporal order without any clustering (*temporal-complete*, Figure 9). By definition, this produces the most balanced tree but the sub-trees do not represent any groupings beyond temporal adjacency. Figures 5 to 8 show examples of the results produced by these algorithms as well as those described below.

We present two new clustering algorithms: *backtrack-balanced* and *split-balanced*. They generate temporally contiguous clusters while also producing fairly balanced trees with average navigation path lengths slightly longer than in balanced trees. Both algorithms relax the constraint that the most visually similar keyframes in a time interval should be in the same cluster.

Others have combined clustering with balanced trees but only for top-down clustering algorithms and without maintaining temporal order [1, 4 17, 18]. Our approach for clustering by similarity is novel in two respects. First, it maintains temporal order and produces a navigation tree with short paths. Second, the technique can be applied to browsing video on devices with a variety of screen sizes.

The *backtrack-balanced* algorithm performs bottom-up clustering and backtracks when clusters become too large. The *split-balanced* algorithm uses a top-down approach to recursively split the video into clusters. It produces shorter average paths and more balanced video segment durations. Figures 5 and 6 show how the two algorithms cluster the same video. A downside of the *split-balanced* algorithm is that for visually repetitive videos, it produces trees with longer search paths. An example would be a home video where the camera moves back and forth between filming activity in the kitchen and the family room. To leverage the strengths of the *split-balanced* algorithm without its weaknesses, we determine the visual repetitiveness of the video. For videos that are not very repetitive, our system applies the *split-balanced* algorithm and for others the *backtrack-balanced* algorithm.

Backtrack-balanced Clustering Algorithm

Agglomerative hierarchical clustering is well-suited for creating clustering trees that maintain temporal order. Rather than considering all pairs of clusters when deciding which cluster to merge next, only pairs of clusters that are temporally adjacent are considered.

Our algorithm uses a bottom-up clustering approach while limiting the number of cluster elements. The algorithm takes as inputs the tree branching factor b (the number of keyframes to be displayed on a screen), the list of keyframes in the video in temporal order, and a distance function to determine cluster distances. The maximum number of leaves in a balanced tree with height h and branching factor b is b^h. To keep the cluster tree balanced, no sub-cluster may contain more than b^{h-1} elements.

The algorithm clusters all keyframes into b clusters, only merging clusters that have temporally adjacent elements. Merged clusters that would have more than b^{h-1} elements are skipped. The algorithm then recursively clusters the elements of the sub-clusters, starting with the smallest k such that b^k is not smaller than the number of cluster elements and increasing k up to h-1 in case of failures. If sub-clustering a cluster produces more than b sub-clusters because no more sub-clusters can be merged, the algorithm marks the cluster as undesirable and backtracks to the point where the undesirable cluster was created. If backtracking happens at the root of the tree, the algorithm starts over with an increased overall tree height. In that case, the longest navigation path will be longer than that in a truly balanced tree.

[1] http://www.itl.nist.gov/div897/sqg/dads/HTML/completetree.html

When comparing Figure 6 and Figure 7, one can see that the right sides of the cluster trees produced by the *backtrack-balanced* and *temporal-visual* algorithms are identical. The *backtrack-balanced* algorithm rearranged the left side to reduce the maximum depth.

Split-balanced Clustering Algorithm

The *split-balanced* algorithm performs top-down clustering by recursively splitting clusters until sub-clusters do not contain more keyframes than the branching factor. The algorithm maintains two constraints. First, the duration of each sub-cluster is kept between $^1/_3$ and 3 times the average duration to make sure that all clusters have a good representation in the timeline. Second, sub-clusters are not assigned more keyframes than can be accommodated with the remaining tree height. To avoid overconstraining, each level of the tree is given an additional padding of 8%. For example, with a branching factor of 4, a tree with a remaining height of 2 does not get assigned more that 15 keyframes (4^2 - 8%) and a tree of height 3 does not get more than 54 keyframes (4^3 - 15.4%). The padding on average results in trees that are slightly taller than truly balanced trees.

To determine the best split for a video sequence, the same number of keyframes are initially assigned to each sub-cluster. The algorithm then keeps picking the boundary with the least strength and replaces it with the strongest possible boundary between the neighboring boundaries that satisfies the constraints of cluster duration and number of cluster members. Once no better boundaries can be found, the sub-clusters are clustered recursively.

Clearly, this algorithm cannot always produce a cluster tree that matches a natural clustering. For example, with a branching factor of 4 and natural clusters of 2, 7, and 6 frames, respectively (see the right of Figure 5), frames from different clusters have to be combined. Still, it keeps the height of the cluster tree small and produces satisfactory results in many cases.

A comparison of Figures 5 and 6 shows that the *split-balanced* algorithm produces a more compact tree at the expense of combining unrelated keyframes (see the second column from the right). The *split-balanced* algorithm performs better than the *backtrack-balanced* algorithm for non-repetitive videos in our test collection.

4.3 Adapting to the Video Repetitiveness

As described, visual repetition in the video plays a large role in determining which clustering algorithm performs better. Thus, we heuristically decide which clustering algorithm to use based on whether a video appears to have repeated scenery. A good metric for selecting among the approaches is the amount of temporal contiguity in non-temporal clusters. We define temporal-visual cohesiveness as the degree to which visually similar keyframes gather in the same temporal region. A video with high cohesiveness would have little visual repetition because the visually similar frames would all be in close temporal proximity.

To determine the temporal-visual cohesiveness of a collection of keyframes extracted from a video, we first create a hierarchical agglomerative cluster tree solely based on visual similarity. We then split the tree into a number of clusters proportional to the duration of the video (e.g., 10 seconds per cluster). Next, we place the members of each cluster in temporal order and compute the mean gap between temporally adjacent keyframes across all clusters. We normalize the mean gap by dividing it by the mean keyframe interval for the whole video. We do not compute any inter-cluster distances. If visual and temporal properties are correlated, only keyframes from the same time in the video will be in the same visual cluster and the normalized mean gap will be close to 1.0. For a random distribution of visual features across the video, the normalized mean gap will approach the number of clusters.

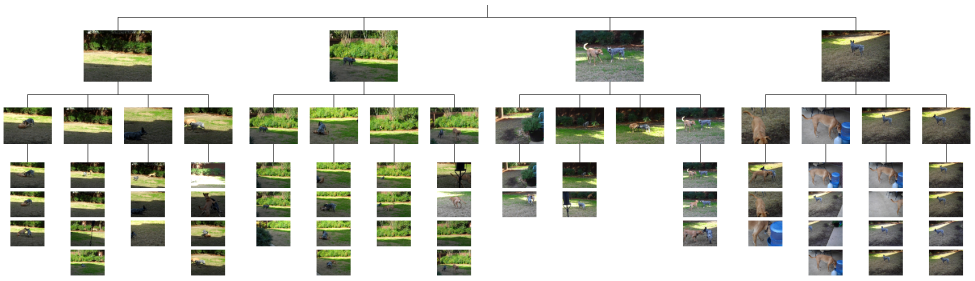

Figure 5. Split-balanced cluster tree.

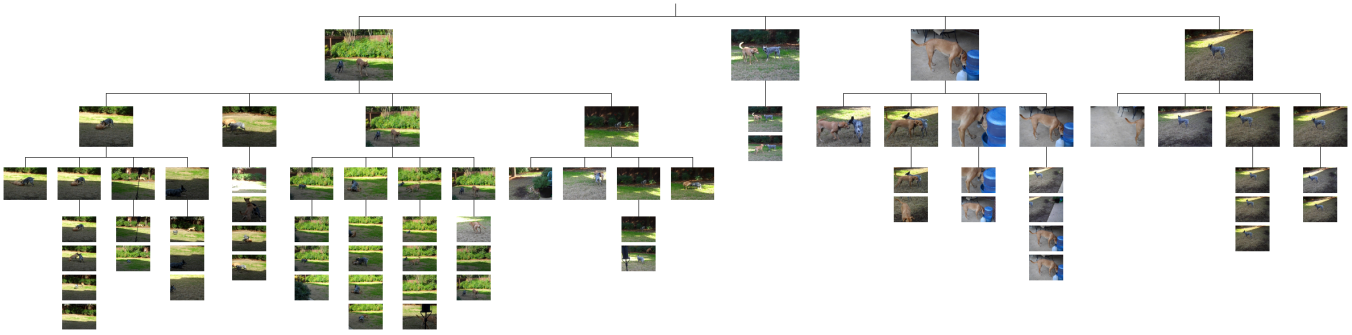

Figure 6. Backtrack-balanced cluster tree.

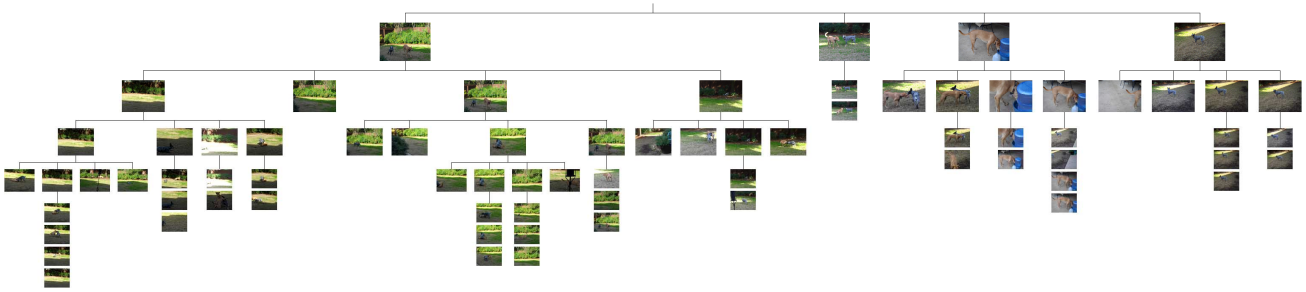

Figure 7. Temporal-visual cluster tree.

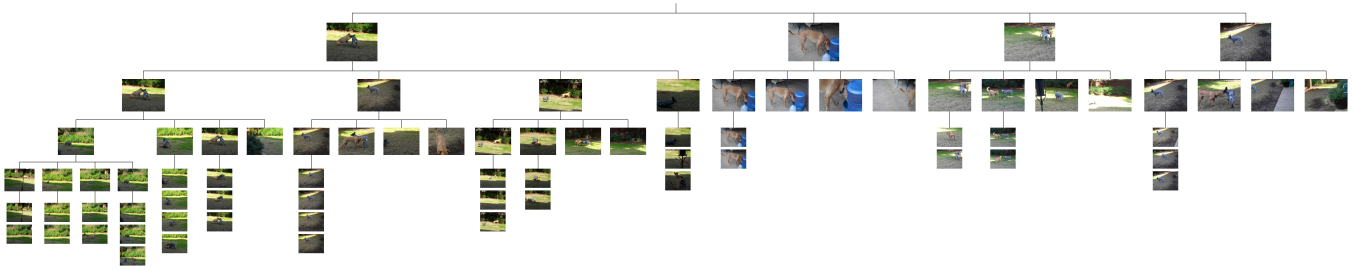

Figure 8. Visual cluster tree (no temporal order).

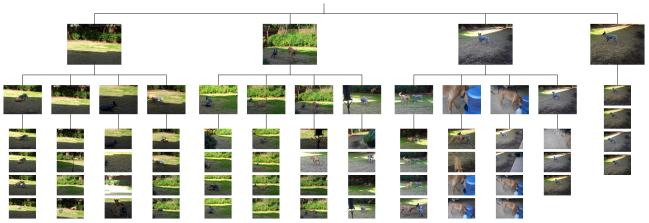

Figure 9. Temporal-complete tree (not clustered).

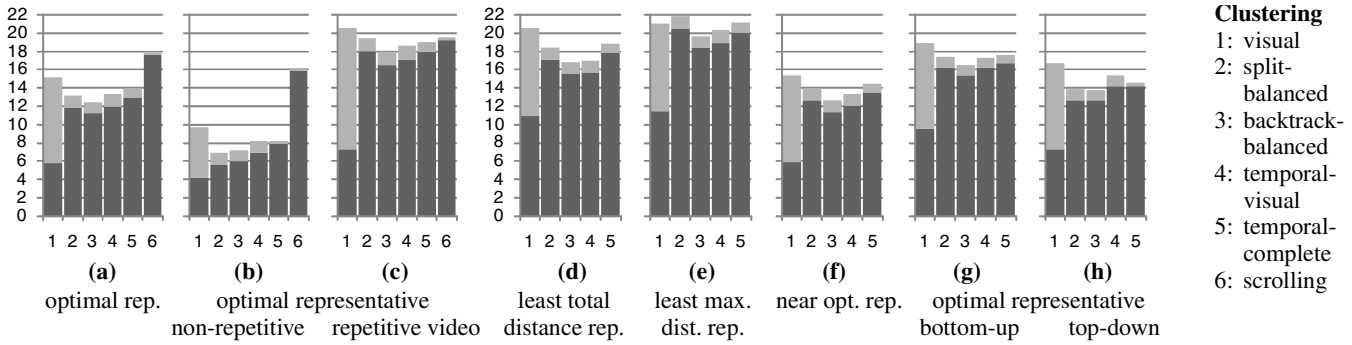

Figure 10. Average navigation distances to a matching frame (dark gray) and its temporal neighbor (light gray).

For our test collection of videos, the median of the normalized mean gap is 2.1. As the standard deviation was less predictive of algorithm performance, we use the inverse of the mean gap as the measure of temporal-visual cohesiveness. We use the above median as the cutoff value for selecting among the two algorithms.

5. EVALUATION OF THE ALGORITHMS

We evaluated our clustering approach using a variety of metrics. Our objective is to identify clustering methods that reduce the overall navigation requirements while preserving the similarity of segments within clusters. We believe that this is the first time that metrics based on a model of user navigation behavior have been used to assess how well suited a particular clustering approach is for navigation tasks. We applied these metrics to a collection of 14 unedited home videos.

5.1 Evaluation Metrics

The metrics for locating an image or a video clip consider the number of steps it would take to perform tasks such as locating an individual keyframe or a short video clip. We also measure the uniformity of segment durations along the timeline with a preference for uniform distributions to better support timeline browsing.

Locating an Image

The metric for finding a target image is the number of steps needed to traverse the cluster tree nodes. At each cluster, we choose the sub-cluster with the visually most similar representative to the target image. The navigation ends at a node whose keyframe is the same as the target image. At a branch containing only non-matching leaf nodes, backtracking to the parent cluster is required followed by the selection of the sub-cluster with the next most similar representative. Backtrack steps are included in the metric.

We also compare the tree navigation to a scrollable display of all keyframes where as many keyframes are visible at a time as in the tree browsing interface. We measure how many display groups have to be stepped through on average.

Locating a Video Clip

The metric for finding a short video clip matching a target image includes the metric for locating an image. To determine the endpoints of a video clip, we need to examine the temporal neighbors of the matching leaf node. Thus, we count the number of steps required to visit the nearest temporal neighbor. For temporal trees, the temporal neighbor is often in the same cluster so that no additional steps are needed. Otherwise, paths are short in such trees.

As a temporal neighbor cannot be located directly in a non-temporal tree, we perform a visual search for the temporal neighbor, again using the target image. Rather than starting that search at the root, we first perform a visual search among the siblings of the leaf node found in the first step. If that search is unsuccessful, we continue with the siblings of its ancestors until the temporal neighbor is found. As before, this counts the steps up and down the tree.

Uniformity of Timeline Segments

As a measure of non-uniformity of cluster durations, we compute the sum of normalized absolute differences from the uniform cluster durations. If all clusters have the same duration, this measure is zero. For example, with a branching factor of 4, the uniform normalized cluster duration is 0.25. With normalized cluster durations of 0.6, 0.2, 0.15, and 0.05, respectively, the non-uniformity measure is 0.7 ((0.6 - 0.25) + (0.25 - 0.2) + (0.25 - 0.15) + (0.25 - 0.05)). The maximum value of this measure for b clusters is $2 * (b - 1) / b$. As the *visual* algorithm does not produce temporally contiguous clusters, this measure is not applicable for it.

5.2 Evaluation Results

For the 14 home videos, we extracted keyframes at approximately 3 fps. To avoid effects due to different durations, we truncated longer videos to 3 minutes. We evenly subsampled those keyframes to between 48 and 256 frames. For the subsampled frames, we kept the strongest segment boundary between neighboring frames. We applied all clustering algorithms to each set of frames with a tree branching factor of 4. The *temporal-visual* algorithm generated trees up to a height of 15 and the *visual* algorithm up to a height of 8. None of the other algorithms exceeded a tree height of 4.

Locating an Image

As indicated by the dark bars in Figure 10a, the *visual* algorithm performed best. This was expected because there are no temporal constraints in this task. When looking just at keyframe sets with a temporal-visual cohesiveness above the median (Figure 10b), the *split-balanced* algorithm performs best among the temporal algorithms. The average path for the *visual* algorithm (4.1) is only slightly longer than the average distance from the root of a complete tree (3.5) indicating that less backtracking takes place. At the other end of the temporal-visual cohesiveness spectrum (Figure 10c), the *split-balanced* algorithm performs worst.

Scrolling on average needs half as many steps as the number of display groups. As expected, all trees perform better than scrolling (see Figure 10a). As backtracking in a tree requires a complete traversal of a sub-tree, scrolling can perform better than trees with ineffective clusters.

If the user has prior knowledge of the video, the approximate time in the video can be used. When restricting the search to sub-trees that overlap a one-minute interval centered around the frame to be located, the performance of the temporal trees approaches that of the *visual* algorithm with 10% longer paths for the balanced clustering algorithms. With the less optimal representative keyframes described in the next section, the temporal algorithms outperform the *visual* algorithm by a good margin in this situation.

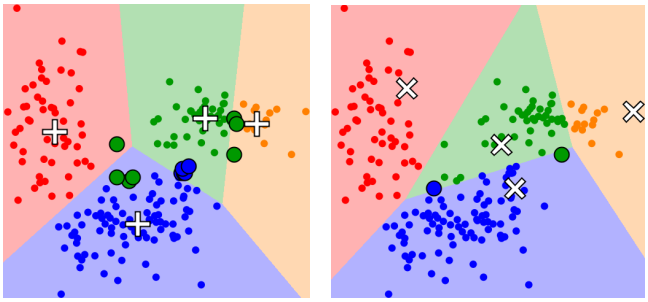

Figure 11. Navigation partitioning with least total distance (+) and optimal (x) representatives indicating wrong clusters with larger dots (11 vs. 2).

Locating a Video Clip

In temporal trees, the distance from the matching leaf node to its temporal neighbor is short (see the light bars in Figure 10a). The more complicated algorithm of locating the temporal neighbor in the tree created by the *visual* algorithm puts that algorithm last. When looking at the performance of this algorithm more closely, we observed that 20% of the neighbors accounted for most of this increase because they required an ascension of two or more tree levels with an average path of 38.4. In those cases, the first found frame may not provide a good visual sample for its temporal neighbor.

Uniformity of Timeline Segments

To measure the uniformity of the cluster durations, we compute the average of the non-uniformity measure weighted by the duration of the respective video sequence. The *temporal-complete* algorithm is most uniform (0.34), followed by *split-balanced* (0.37), *backtrack-balanced* (0.48), and *temporal-visual* (0.74). By replacing the complete tree in the *temporal-complete* algorithm with a balanced tree that maximizes the uniformness of cluster durations, one could further reduce its non-uniformity at a small expense of navigation path lengths.

5.3 Representative Keyframes

The selection of representative keyframes for each cluster turned out to be more important than we expected. Our first approach was to pick the frame with the least total maximum distance to the other frames in the cluster (Figure 10d), an approximation of the cluster centroid. We unsuccessfully tried to improve the discriminative power by favoring representative frames that are dissimilar to keyframes in sibling clusters. Picking the frame with the least maximum distance to the other frames in the cluster was also counterproductive (Figure 10e).

To determine why the algorithms performed so badly, we checked how often the search path went down the wrong sub-tree because the sample image was more similar to the representative frame of that sub-tree. The *visual* algorithm performed best here but still took a wrong turn at the top-level in 10.3% of the paths with an average detour of 58.9. This does not include additional detours at lower levels. As expected, the temporal-complete algorithm performed worst with 33.3% wrong paths at the top-level. Descending into the wrong cluster has a major impact on search performance because the whole sub-tree has to be traversed before backtracking to the top-level.

To overcome these performance issues, we conduct an exhaustive search on all permutations of representative keyframes (see Figure 10a). This leads to a much better performance with 2.6% wrong paths at the top-level for the visual-algorithm. Figure 11 illustrates the effects of different representative frames with 11 versus 2 wrong clusters.

With n total keyframes and b branches, up to $(n/b)^b$ permutations have to be checked (16 million for n=256 and b=4). With four branches, this makes the algorithm about 500 times slower than the least total distance algorithm for our data set. Furthermore, permutations increase quickly with larger branching factors.

To deal with this combinatorial explosion, we decided to put an upper limit on the number of permutations. For example, with a limit of 160,000 permutations, four clusters can each provide 20 candidates or three can provide 10 candidates each and the fourth can use the remaining 160. For clusters exceeding the limit, we compute as many sub-clusters as the limit and select the centroid for each of them. The comparison of Figures 10a and 10f shows that this approach is near-optimal with only 3% longer paths.

We received user feedback that not seeing the representative frame of a cluster anywhere in the expanded cluster is confusing. We initially used a bottom-up approach where the representative frame for a parent segment is selected among those of the children. We also employed a top-down approach where the representative frame for a cluster is also assigned to the sub-cluster containing that frame. Representative frames for other clusters are selected as before. The comparison of Figures 10g and 10h shows that the top-down approach produces on average 19% shorter paths.

5.4 Discussion

These results provide clear guidance for how to select a clustering algorithm appropriate both for the user's task and for the repetitiveness of the video. For the first task of finding a single image, visual clustering is optimal. If the user has prior knowledge of the video to narrow the time interval for the search, temporal clustering algorithms can approach or even exceed the performance of visual clustering. For the second task of finding a video clip, the b*acktrack-balanced* algorithm is optimal for repetitive video and the *split-balanced* algorithm is optimal for non-repetitive video. As visually repetitive videos can be easily detected, the visualization appropriate for a particular video can be chosen. Neither the non-balanced temporal-visual algorithm nor the non-visual complete tree was preferable for any task or type of video. A scrolling interface using the same screen space and keyframe sizes performs much worse than tree representations.

The selection of representative keyframes had a surprisingly large impact on the overall search performance. Limiting the number of permutations in an exhaustive search for the optimal representatives produced good results. We found that the users' desire to maintain context by reusing keyframes that were used in higher levels of the tree lead to less optimal trees with longer search paths. By using a top-down approach for selecting representative frames, the performance penalty was reduced.

6. USER FEEDBACK

We invited two people external to our research laboratory to use the browsing user interface in order to gather feedback on its intuitiveness and effectiveness. We first played two videos for them, each 1-2 minutes in length and described a clip we wanted them to find in each one. One video was of a race, and the participants were asked to find a clip of the runners as they reached the finish line. The other video was of a child's birthday party, and we asked them to locate the clip where the child tries to blow out the last candle on the cake. We scrubbed the video and showed them they could do it that way, but then asked them to try our interface. The specific task was to show us the first keyframe of the clip and the last one. Each participant used the interface for one video while the other participant watched. They both found the requested clips in a minute or two — the tasks were perceived as really easy.

Comments indicated that the interface needs to provide more feedback with regards to the user's location within the cluster tree. While the keyframe border already turned red for leaf nodes, they suggested that the same color should be used for the corresponding timeline segment. They also wanted to have more indication of how deep they were in the tree. Currently, only the top and the bottom of the tree offer visual cues. That affected their task because they found a keyframe from the target frame without realizing that it might not be the beginning of the target clip.

Other comments indicated that they were unsure about aspects of the visualization and the interaction with the scroll wheel. In particular, they were confused about the piles of keyframes on the sides and had to experiment with the piles to understand their function. This happened after they completed their tasks because they did not notice or need the piles to complete the tasks. Also, they wanted to first click on an image and then use the scroll wheel. It appeared that they did not trust that the keyframe was selected on mouse-over.

Participants comments also identified applications for the overall system in viewing sports video. One participant's team currently watches videos of their water polo games while the coach critiques them. She said this system would be a good way to get to a specific play in the game, show a still image of a key incident in the play, and find out what happened just before and after. Participants also proposed the idea of browsing a video to locate the best still image of a person's face for printing and sharing. They both indicated that the interface was fun and attractive.

Overall, the feedback shows that both participants easily understood the overall metaphor of navigating to get shorter or longer segments of the video. It also showed that the interaction with the cluster tree would benefit from more visual feedback about the current state and what will happen when the user clicks on an image or uses the scroll wheel.

7. CONCLUSIONS

In this paper, we looked at approaches for providing video access through a hierarchy of keyframes. We presented a user interface suitable for quickly browsing the structure of a video to locate images or short video clips of interest. It provides fluid access to the video by displaying a keyframe hierarchy along a timeline supporting the zooming-in on areas of interest. Keyframes are put in balanced tree to avoid long navigation paths. The system adapts to both the task and the video. If the user wants a single image in the video, the interface is based on a non-temporal visual clustering algorithm. If the user wants a video clip, the system first analyzes the video to determine whether it is repetitive or non-repetitive and then chooses among two temporal-visual clustering algorithms.

We introduced and used several metrics to assess the performance of these algorithms. We presented heuristics for picking the algorithm best suited for a particular video. In addition to using a good clustering algorithm, it is also very important to pick representative frames that discriminate among clusters. The novelty of our approach is clustering by similarity while maintaining temporal order and producing a navigation tree with short paths. Adapting the selection of the particular clustering algorithm to the user's task and the repetitiveness of the video is another novel aspect of our work. While temporal-visual clustering in general supports video browsing well, we found that it performs poorly if video is too visually repetitive. As we can detect that, we offer an alternative interface in such cases.

Early user comments were favorable. We conducted a preliminary user evaluation and received valuable suggestions for improvements to our systems. The performance advantages of our techniques indicate that our approach should compare favorably to other techniques for accessing video clips.

8. REFERENCES

[1] C.-H. An, K. Berry, and A. Cosby. Fractal image compression by improved balanced tree clustering *Proc. of SPIE, Vol. 3164, Applications of Digital Image Processing XX,* 555-564, 1997.

[2] G. Ciocca and R. Schettini. Hierarchical Browsing of Video Key Frames. *Lecture Notes in Computer Science,* Vol. 4425, Advances in Information Retrieval, 691-694, 2007.

[3] M. Cooper and J. Foote. Scene Boundary Detection Via Video Self-Similarity Analysis. *Proc. of Int. Conf. on Image Processing,* 378-381, 2001.

[4] S. Geva. K-tree: a height balanced tree structured vector quantizer. *Proc. of 2000 IEEE Signal Processing Society Workshop.* Vol. 1, 271-280, 2000.

[5] A. Girgensohn, S. Bly, F. Shipman, J. Boreczky, and L. Wilcox. Home Video Editing Made Easy — Balancing Automation and User Control. *Proc. of INTERACT '01,* IOS Press, 464-471, 2001.

[6] M. Guillemot and P. Wellner. A Hierarchical Keyframe User Interface for Browsing Video over the Internet. *Proc. of INTERACT '03*, 769-776, 2003.

[7] W. Hürst, G. Götz, and P. Jarvers. Advanced User Interfaces for Dynamic Video Browsing, *Proc. of ACM Multimedia*, 742-743, 2004.

[8] J. Jiang and X.-P. Zhang. A New Hierarchical Key Frame Tree-Based Video Representation Method Using Independent Component Analysis. *Advanced Intelligent Computing Theories and Applications. With Aspects of Artificial Intelligence. Lecture Notes in Computer Science*, Vol. 6216/2010, 132-139, 2010.

[9] J. Luo, C. Papin, and K. Costello. Towards Extracting Semantically Meaningful Key Frames From Personal Video Clips: From Humans to Computers. *IEEE Transactions on Circuits and Systems for Video Technology*, 19(2), 289-301, 2009.

[10] C.-W. Ngo, T.-C. Pong, and H.-J. Zhang. On Clustering and Retrieval of Video Shots. *Proc. of ACM Multimedia,* 51-60, 2001.

[11] J.H. Oh and K.A. Hua. Efficient and Cost-effective Techniques for Browsing and Indexing Large Video Databases. *Proc. of ACM Conf. on Management of Data*, 415-426, 2000.

[12] Y. Rui, T. Huang, and S. Mehratra. Constructing Table-of-Contents for Videos. *ACM Multimedia Systems,* 7(5):359-368, 1999.

[13] K. Schoeffmann, M. Taschwer, and L. Boeszoermenyi. The Video Explorer — A Tool for Navigation and Searching within a Single Video based on Fast Content Analysis. *Proc. of ACM Conf. on Multimedia Systems,* 247-258, 2010.

[14] F. Shipman, A. Girgensohn, and L. Wilcox. Hypervideo Expression: Experiences with Hyper-Hitchcock. *Proc. of ACM Conf. on Hypertext and Hypermedia,* 217-226, 2005.

[15] K. Wittenburg, C. Forlines, T. Lanning, A. Esenther, S. Harada, and T. Miyachi. Rapid Serial Visual Presentation Techniques for Consumer Digital Video Devices. *Proc. of ACM UIST*, 115-124, 2003.

[16] J. Xiao, X. Zhang, P. Cheatle, Y. Gao, C.B. Atkins. Mixed-Initiative Photo Collage Authoring. *Proc. of ACM Multimedia,* 509-518, 2008.

[17] T. Zhang, R. Ramakrishnan, and M. Livny. BIRCH: An Efficient Data Clustering Method for Very Large Databases. *Proc. of ACM Conf. on Management of Data,* 103-114. 1996.

[18] D. Zhong, H. Zhang and S.-F. Chang. Clustering Methods for Video Browsing and Annotation. *Proc of SPIE Conf. on Storage and Retrieval for Image and Video Databases*, 1997.