Computing Content-Plots for Video[⋆]

Haim Schweitzer

The University of Texas at Dallas, Richardson TX 75083, USA
haim@utdallas.edu

Abstract. The content-plot of a video clip is created by positioning several key frames in two-dimensions and connecting them with lines. It is constructed so that it should be possible to follow the events shown in the video by moving along the lines. Content plots were previously computed by clustering together frames that are contiguous in time. We propose to cluster together frames if they are related by a short chain of similarly looking frames even if they are not adjacent on the time-line. The computational problem can be formulated as a graph clustering problem that we solve by extending the classic k-means technique to graphs. This new graph clustering algorithm is the main technical contribution of this paper.

1 Introduction

Vast amount of data is available in digital form over the Internet and other electronic media. Measured in bytes, digital video dominates. Producing short descriptions of the information in the video data is a major challenge. There is large amount of recent work on digital video parsing and clustering with the apparent motivation of smart video browsing (e.g., [1,2,3,4,5,6]). The general approach is to have the data segmented into video shots, with each shot represented by one or more key frames. The video shots are determined from discontinuities along the time axis.

The goal of this paper is to compute a very compact representation of the visual information in the video data. This may require clustering together frames that do not necessarily belong to the same video shot, and may not be near each other along the time line. Our argument is that in many cases segmentation computed as sequencing along the temporal axis (video shots) does not catch the inherent redundancy in the scene. The following are several example cases:

– The video may be edited, with portions of a single shot being cut and pasted in various locations along the final video time line.
– The video may be taken using more than one camera, with the editor frequently switching between these cameras.

[⋆] This material is based in part upon work supported by the Texas Advanced Research Program under Grant No. 009741-0074-1999 and the Texas Advanced Technology Program under Grant No. 009741-0042-2001

– In a typical situation of several people being interviewed the camera is moving back and forth among them, usually focusing on the one who speaks.

Therefore, the description of the visual-content of the video need not follow the time line, and ideally key frames should be computed by identifying similarities among frames regardless of their location along the time line.

One must distinguish between visual similarity, which can be measured directly from pixel values, and *semantic similarity*, which implies, for example, that the left side of the room is near the right side of the same room even if they look different. The exact definition of semantic similarity and the development of algorithms that can detect it are not discussed here in detail. We propose the following "practical" definition of *semantic similarity* between two frames with respect to the given video data:

> Frame A is considered semantically similar to Frame B if there is a short sequence of frames $f_1 = A, f_2, \ldots, f_k = B$ where f_i is visually similar to f_{i+1}.

This is closely related to the definition of the distance between two nodes on a weighted graph (e.g., [7]). It suggests that the clustering should be performed on a graph created from the frames as nodes, with weights corresponding to their pairwise visual distances.

Our proposed approach for computing content plots performs the analysis in three distinct phases. In the first phase, described in Section 2, the data is analyzed for similarities among all frame pairs. The result is viewed as a graph of frame distances to be processed in the second and third phase. We observe that straightforward computation of distances between all frame pairs has quadratic complexity in the size of the data, which is impractical except for short video clips. When applied to long video movies the frames may have to be sampled. Our proposed algorithm uses projections of the frames into eigenspace representation. Recently developed efficient method for computing such projections (e.g., [8,9]) enable implementation of this phase in linear time.

The input for the second phase is the graph of pairwise frame (dis)similarities. The output is the clustering of the frames and the selection of a small number of key frames. The size of this graph (number of edges) is quadratic in the number of frames. In Section 3 we describe a novel algorithm for clustering this graph by extending the classic k-means algorithm to handle graph data. This is the main technical contribution of this paper.

The goal of the third phase, described in Section 4, is to provide a visual description of the relations between the key frames. This is done by positioning these frames in 2D according to their visual content, and plotting the order in which the clusters are visited according to the time line. We use classic multidimensional scaling technique for this 2-dimensional embedding of the frames, using the distances computed in the first phase.

2 Computing Pairwise Frame Distances

Two frames are considered similar if there is a small translation that puts them in register. The registration error is a measure to their degree of similarity.

Let p denote an image, and let $p(\alpha, \beta)$ be the image translated by α, β. The Horn and Schunck constraint [10] can be used to express $p(\alpha, \beta)$ as a linear function of α, β:

$$p(\alpha, \beta) \approx p - p_x \alpha - p_y \beta$$

where p_x, p_y are the partial derivatives of p in the x and y directions.

Let $\mathrm{p1}, \mathrm{p2}$ be two images. We are looking for symmetric formulas to approximate the translation between p1 and p2. Assume that p1 is translated by α_1, β_1 and that p2 is translated by α_2, β_2 so that the two translated images are similar. This gives:

$$\mathrm{p1}(\alpha_1, \beta_1) \approx \mathrm{p2}(\alpha_2, \beta_2),$$

From the Horn and Schunck linearization we have:

$$\mathrm{p2} - \mathrm{p1} \approx -\alpha_1 \mathrm{p1}_x - \beta_1 \mathrm{p1}_y + \alpha_2 \mathrm{p2}_x + \beta_2 \mathrm{p2}_y$$

In matrix notation this can be written as:

$$d \approx D\phi \tag{1}$$

where $d = \mathrm{p2} - \mathrm{p1}$, $D = (-\mathrm{p1}_x, -\mathrm{p1}_y, \mathrm{p2}_x, \mathrm{p2}_y)$, and $\phi = (\alpha_1, \beta_1, \alpha_2, \beta_2)'$.

To measure how similar are p1 and p2 we compute the error in least squares solution of (1). Let U be an orthonormal basis to the columns of D then:

$$|e|^2 = |d|^2 - |U'd|^2 \tag{2}$$

where the error $|e|$ is defined to be the degree of (dis)similarity between p1 and p2.

2.1 Distances between All Pairs

Now consider the case of m images $\mathrm{p1}, \ldots, \mathrm{pm}$. The goal is to compute the error in (2) between all pairs. Direct approaches are impractical even for moderate values of m, but using eigenspace techniques reduces the complexity enough to enable handling of a few thousands images.

Let $V = (v_1, \ldots, v_k)$ be the k dominant eigenvectors of of the images. Then the formula (2) can be used with the values of d, U computed in the subspace spanned by V.

Algorithm:
1. Compute V as the k dominant eigenvectors of $\mathrm{p1}, \ldots, \mathrm{pm}$.
2. Compute $\hat{\mathrm{pi}} = V'\mathrm{pi}$, $\hat{\mathrm{pi}}_x = V'\mathrm{pi}_x$, $\hat{\mathrm{pi}}_y = V'\mathrm{pi}_y$, for $i = 1, \ldots, m$.

3. For each (i,j) pair:
3a. Compute the derivatives matrix:
$\hat{D} = (-\hat{\mathrm{pi}}_x, -\hat{\mathrm{pi}}_y, \hat{\mathrm{pj}}_x, \hat{\mathrm{pj}}_y)$
3b. Compute $\hat{U}$, an orthonormal basis of $\hat{D}$ (e.g., using Gram Schmidt).
3c. Compute the error from (2) where $\hat{d} = \hat{p_j} - \hat{p_i}$,

Using an iterative technique (e.g., [8,9]) for computing the eigenvectors, Step 1 can be computed in linear time. Step 2 is also linear, and Step 3 is quadratic in m, the number of frames, and k, but not in the number of pixels. In our implementation Step 1 was implemented with the Ritz approximation technique [8], and the value of k was taken as 15. The run time of the algorithm was dominated by Step 1,

3 Graph Clustering by *k*-means

Once the graph of dissimilarities between all frame pairs is computed one can apply standard graph clustering techniques to group the frames. Unfortunately, our experiments with standard techniques that are based on minimum cuts and normalized minimum cuts ([11,12]) proved unsatisfactory. In our experiments these techniques cut out meaningful clusters, but they leave at least one large component. That component may be further split, but this typically results in a large number of clusters. Specifically, on typical video clips we found the minimum cut approach not to provide good clustering when the number of desired clusters is below 20. If one is interested in a smaller number of clusters, say less than 10, the result was unsatisfactory.

When the data belongs to a Euclidean space, a classic clustering technique that provides *pivots* to characterize each cluster is k-means. A straightforward implementation of k-means cannot be used to compute clusters in graphs. In this section we extend k-means to graph data. To the best of our knowledge this has never been done before.

3.1 Classic *k*-means

Let $x_1, \ldots, x_n$ be vectors in a multidimensional Euclidean space. The classic k-means clustering algorithm (e.g, [13,14,15]) computes a partitioning of these points into disjoint subset $S_1, \ldots, S_k$ so that the following error is minimized:

$$E = \sum_{j=1}^{k} \sum_{i \in S_j} |x_i - \mu_j|^2 \tag{3}$$

The vectors $\mu_1, \ldots, \mu_k$ are called *pivots*. In signal processing the error E is sometimes called the "quantization error". The minimization of E is achieved by iterating two steps. In the first step the partition is given, and each pivotal vector μ_j is computed as the mean of the vectors in the group S_j. In the second step the pivotal vectors are given. The group S_j is computed as the vectors closer to μ_j than to any other pivotal vector. The proof that both steps reduce the error E is straightforward. We need the following observation which is part of the proof:

> μ_j, the mean (centroid) of $x_i \in S_j$, is the vector y that minimizes the error term $\sum_{i \in S_j} |x_i - y|^2$.

3.2 *k*-means on Graphs

Let G be a graph of n nodes $V = \{v_1, \dots, v_n\}$. Since distance is defined only between graph nodes, it is natural to take the pivots $\mu_1, \dots, \mu_k$ as k nodes in V, and take $d(v_i, \mu_j)$ as the given distance between v_i and μ_j. This gives the following analog to the error (3):

$$F = \sum_{j=1}^{k} \sum_{i \in S_j} d(v_i, \mu_j) \tag{4}$$

As in the case of classic k-means it is easy to see that F is being reduced by each one of the following steps: In the first step the partition is given, and each pivotal node μ_j is computed as the minimizer of $\sum_{i \in S_j} d(x_i - y)$. In the second step the pivotal nodes are given. The group S_j is computed as the nodes closer to μ_j than to any other pivotal node. The proof that both steps reduce the error F is straightforward.

Unfortunately, the non Euclidean nature of the node distances causes problems with this straightforward generalization of k-means. In particular, there is no guarantee that the pivotal node μ_j is a member of S_j, or that the k pivotal vectors are distinct. However, these problems disappear if the distance measure d is chosen as the *shortest path distance* between nodes. Algorithms for efficient computation of shortest path are well known. See, e.g., [7]. This gives the following algorithm for Graph k-means Clustering:

```
Algorithm:
Input: A graph $G$ with $n$ nodes (images) $x_1, \dots, x_n$, and distances specified
    between all pairs of nodes.
Output: the partitioning $S_1, \dots, S_k$, and the pivotal images (nodes) $\mu_1, \dots, \mu_k$.
0. For each pair of nodes $v_i, v_j$, compute the shortest path distance $d(v_i, v_j)$.
1. Start with an initial choice of pivots $\mu_j$, $j = 1..k$.
2. Iterate to convergence the following $k$-means steps:
    a. Compute a new partition.
       $i \in S_\alpha$ if $d(x_i, \mu_\alpha) = \min_j d(x_i, \mu_j)$.
    b. Compute new pivots.
       Define: $T_{ij} = \sum_j d(x_i, x_j)$.
       Choose $\mu_j$ as the node $x_t$ satisfying:
       $T_{tj} = \min_i T_{ij}$.
```

It is easy to verify that each iteration of the algorithm has quadratic complexity in the number of nodes, which is acceptable in our model. In our experiments the convergence was very fast, typically in less than 5 iterations.

4 Content Plots of Video Clips

Running the k-means algorithm of Section 3 produces k key frames. When k is chosen to be small one can easily plot these frames in 2D, and connect them according to the time-line. Specifically, the key frame of Cluster j_1 is connected with a line to the key frame of Cluster j_2 if there are two frames adjacent on the time line, where one belongs to j_1 and the other to j_2.

This produces a similar description to the one generated in [3]. In this section we discuss how to automate this process. The key idea is to use the same graph of frame distances computed in Section 2 to position the frames according to their visual content.

Let $\mu_1, \ldots, \mu_k$ be the k frames, and let W be their $k \times k$ distance matrix. The frames can be positioned in 2D so that their distances approximate the values in W. The computation of this embedding is known as *Multidimensional Scaling.* The classic solution (see, e.g., [16]) is given by the following simple algorithm.

1. Compute the matrix B as:
$B = -1/2(I - \mathbf{1}\mathbf{1}'/n)W(I - \mathbf{1}\mathbf{1}'/n)$, where "$\mathbf{1}$" is the vector $(1, \ldots, 1)$.
2. Compute eigenvector factorization of B. Let v_1, v_2 be the two eigenvectors corresponding to λ_1, λ_2, the two largest eigenvalues. The 2D coordinates of μ_j are: $(v_1(j)\sqrt{\lambda_1}, v_2(j)\sqrt{\lambda_2})$.

5 Experimental Results

We describe experimental results with several video clips that were downloaded from the Internet. In each case we show the pairwise distances computed using the algorithm of Section 2, the shortest path distances, and the visual content plot generated from the key frames using the MDS technique discussed in Section 4.

In the first example the input is a video clip downloaded from MTV web site. The video was created with several cameras, that move around the singer. It starts with the first camera showing the video shot characterized by Frame 2, then the camera switches to a closeup as shown in scene characterized by Frame 1, then there is a switch back to the scene characterized by Frame 2, and then a different camera shows the scene in Frame 5, etc.

The second example shows James Baker reading an announcement in a news conference after the last US presidential elections. It was downloaded from the AP News web site. There is very little "action" here, and the scene repeats itself, which translates into repetitive patterns in the distance matrix.

In the first two examples the algorithm computes 6 clusters, which we found to be sufficient in most cases. In the last example we demonstrate what happens when the number of clusters grows, even though the visual-content does not require so many key frames. The clip described in this example is the trailer of the "scream" movie. Most of the clip shows the actress talking over the phone. The k-means algorithm produces multiple clusters, but the MDS algorithm positions

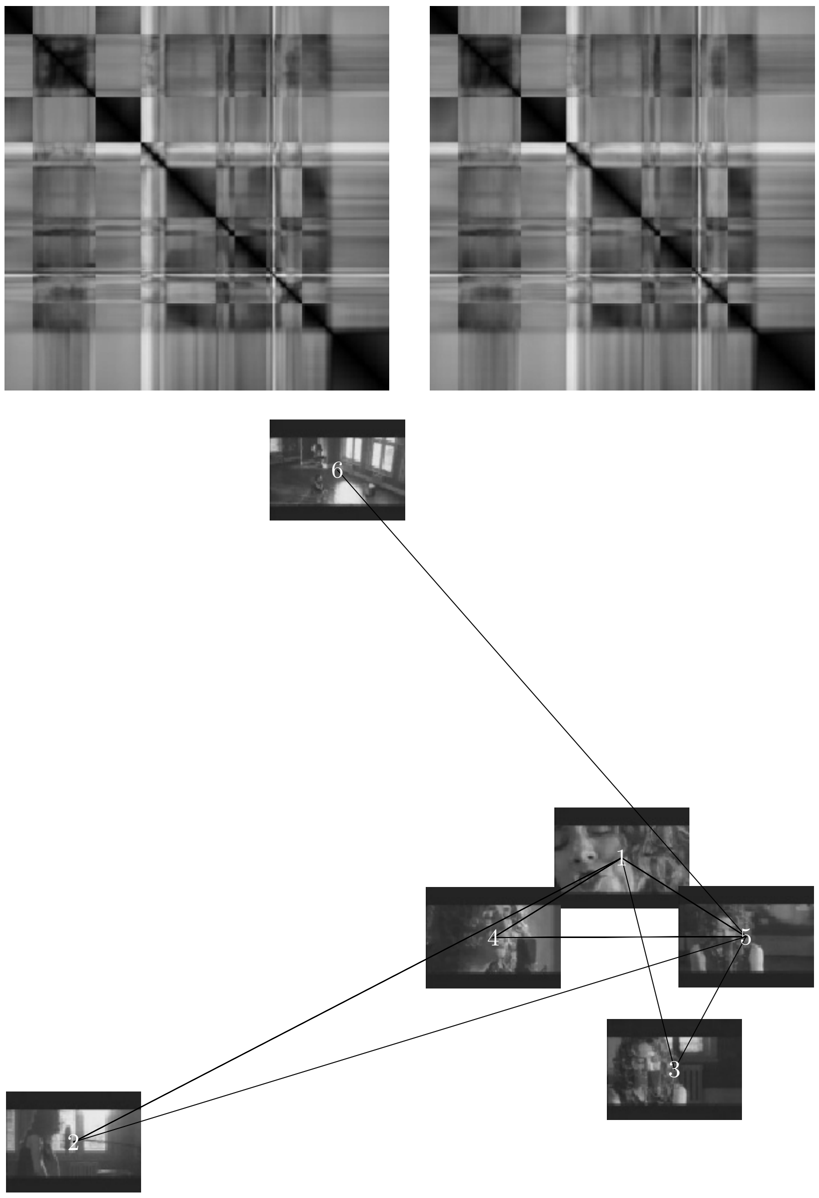

Fig. 1. Results obtained from an MTV clip of Cheryl Crow. The clip has 450 frames. The top image shows the distances computed between all pairs. The the left image is the computed distances, and the right image shows the shortest path distances. The bottom is the results obtained for 6 clusters.

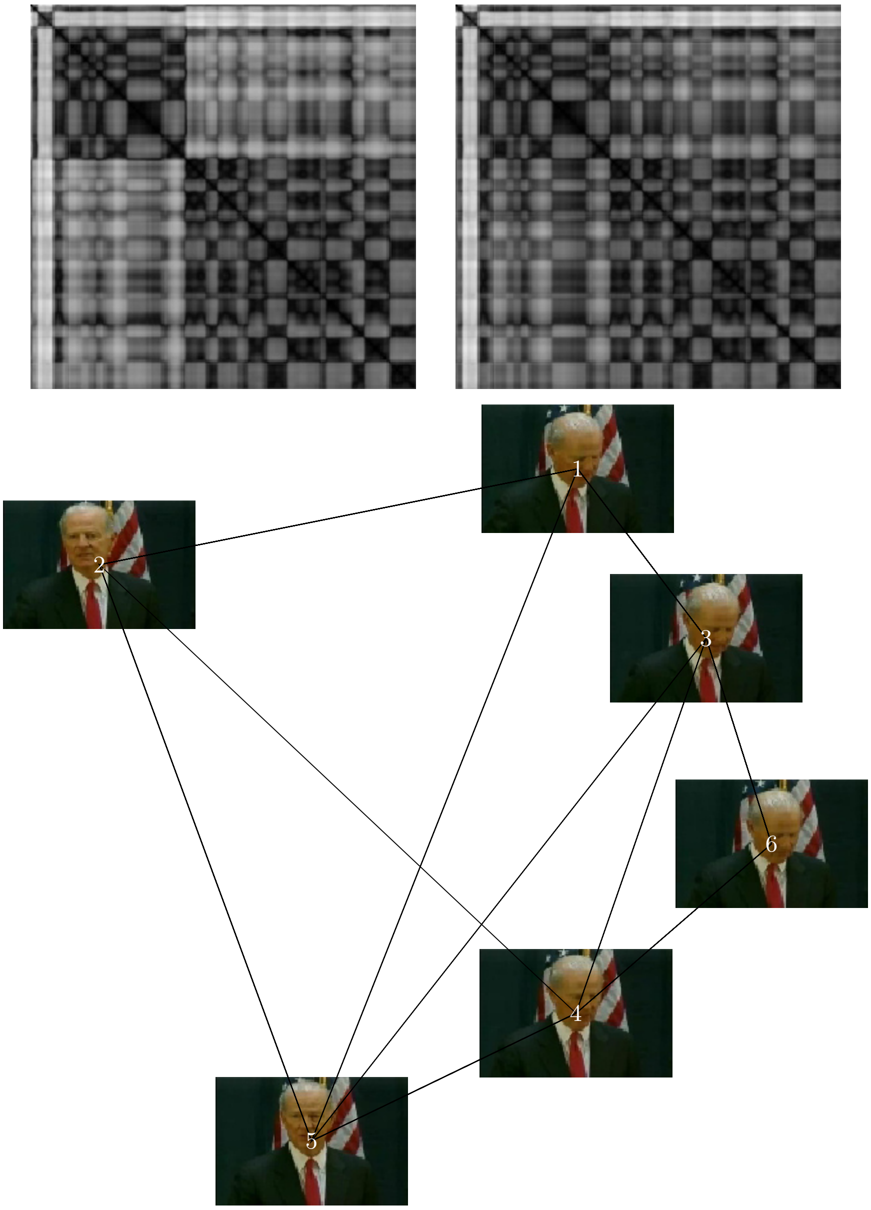

Fig. 2. Results obtained from a recent clip of a news conference of James Baker, downloaded from AP News Wire. The clip has 693 frames. The top image shows the distances computed between all pairs. The the left image is the computed distances, and the right image shows the shortest path distances. The bottom is the results obtained for 6 clusters.

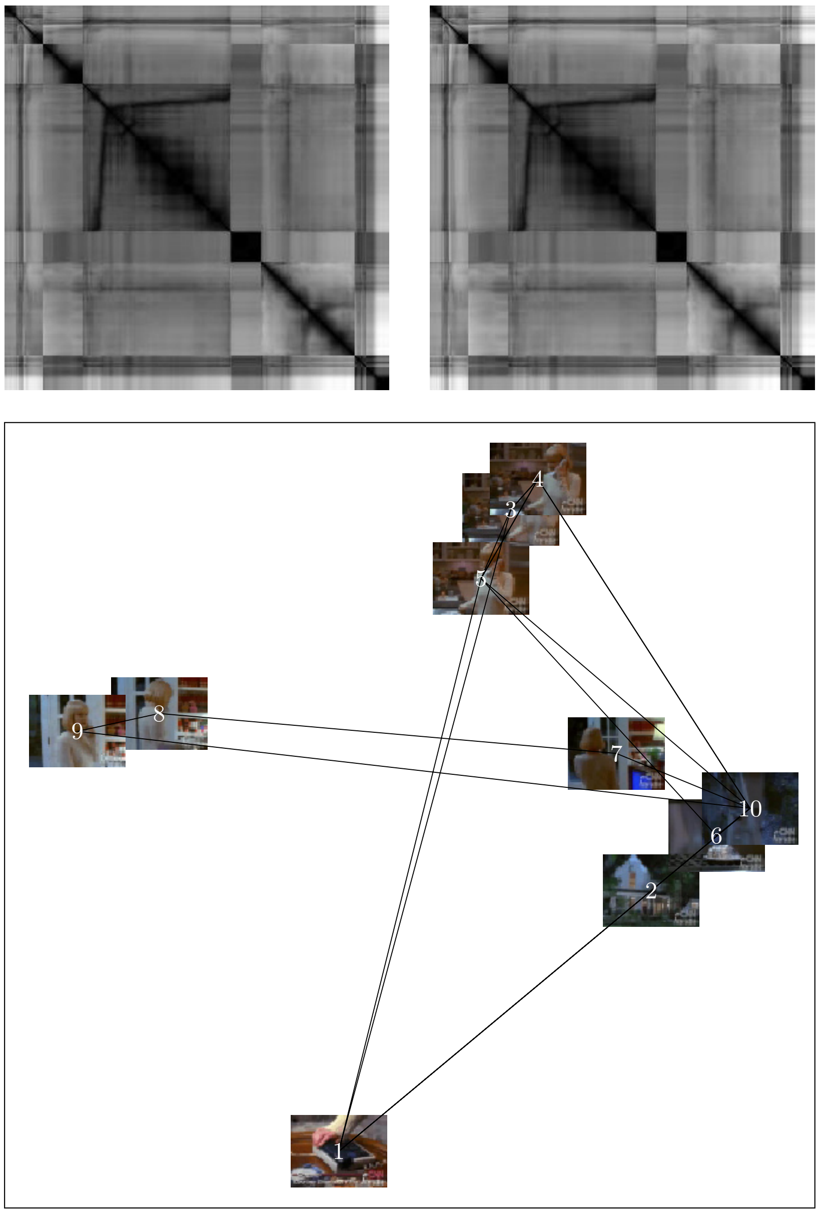

Fig. 3. Results obtained from a video clip of the trailer to the *scream* movie, downloaded from CNN web site. The clip has 450 frames. The top image shows the distances computed between all pairs. The the left image is the computed distances, and the right image shows the shortest path distances. The bottom is the results obtained for 10 clusters.

them at similar locations in the 2D plot so that the relation between these scenes can be clearly visualized.

6 Concluding Remarks

A technique for clustering and analysis of video clips was presented. It extracts a small number of key frames without any information from the time-line. These frames are automatically positioned in 2D according to their visual-content, providing a compact description of the visual information in the clip.

The basic idea is to compute a graph of pairwise frame distances and then cluster this graph into "semantically similar" frames. The main technical result of this paper is the clustering algorithm. We develop an extension of the classic k-means algorithm to graphs, replacing centroids with key frames.

References

1. Xiong, W., Lee, J.C.M.: Efficient scene change detection and camera motion annotation for video classification. Computer Vision and Image Understanding: CVIU **71** (1998) 166–181
2. Zhang, H., Kankanhalli, A., Smoliar, S.: Automatic partitioning of full-motion video. ACM Multimedia Systems **1** (1993) 10–28
3. Yeung, M., Yeo, B., Liu, B.: Segmentation of video by clustering and graph analysis. Computer Vision and Image Understanding: CVIU **71** (1998) 94–109
4. Bolle, R.M., Yeo, B.L., Yeung, M.M.: Video query: Research directions. IBM Journal of Research and Development **42** (1998) 233–252
5. Dimitrova, N., Golshani, F.: Motion recovery for video content classification. ACM Transactions on Information Systems **13** (1995) 408–439
6. Yeo, B.L., Yeung, M.M.: Retrieving and visualizing video. Communications of the ACM **40** (1997) 43–52
7. Cormen, T.H., Leiserson, C.E., Rivest, R.L.: Introduction to algorithms. 6th edn. MIT Press and McGraw-Hill Book Company (1992)
8. Schweitzer, H.: A distributed algorithm for content based indexing of images by projections on Ritz primary images. Data Mining and Knowledge Discovery **1** (1997) 375–390
9. Chandrasekaran, S., Manjunath, B.S., Wang, Y.F., Winkeler, J., Zhang, H.: An eigenspace update algorithm for image analysis. Graphical models and image processing: GMIP **59** (1997) 321–332
10. Horn, B.K., Schunck, B.G.: Determining optical flow. Artificial Intelligence **17** (1981) 185–203
11. Wu, Z., Leahy, R.: An optimal graph theoretic approach to data clustering: Theory and its application to image segmentation. IEEE Transactions on Pattern Analysis and Machine Intelligence **15** (1993) 1101–1113
12. Shi, J., Malik, J.: Normalized cuts and image segmentation. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR'97). (1997) 731–737
13. Fukunaga, K.: Introduction to Statistical Pattern Recognition. Second edn. Academic Press, New York (1990)

14. Jain, A.K., Dubes, R.C.: Algorithms for Clustering Data. Prentice-Hall, Englewood Cliffs, NJ (1988)
15. Marroquin, J.L., Girosi, F.: Some extensions of the K-means algorithm for image segmentation and pattern classification. Technical Report AIM-1390, Massachusetts Institute of Technology (1993)
16. Cox, T.F., Cox, M.A.: Multidimensional Scaling. Chapman & Hall (1994)