A Genetic Segmentation Algorithm for Image Data Streams and Video

Patrick Chiu **Andreas Girgensohn** **Wolf Polak**

Eleanor Rieffel **Lynn Wilcox** **Forrest H Bennett III**

FX Palo Atlo Laboratory, 3400 Hillview Avenue, Bldg. 4, Palo Alto, CA 94304, *lastname*@pal.xerox.com

Abstract

We describe a genetic segmentation algorithm for image data streams and video. This algorithm operates on segments of a string representation. It is similar to both classical genetic algorithms that operate on bits of a string and genetic grouping algorithms that operate on subsets of a set. It employs a segment fair crossover operation. For evaluating segmentations, we define similarity adjacency functions, which are extremely expensive to optimize with traditional methods. The evolutionary nature of genetic algorithms offers a further advantage by enabling incremental segmentation. Applications include browsing and summarizing video and collections of visually rich documents, plus a way of adapting to user access patterns.

1 INTRODUCTION

Segmenting multimedia data streams is a fundamental problem with many applications. By *segmentation*, we mean breaking up a data stream into meaningful parts. Properly segmented streams can be better organized and reused. They provide points of access that facilitate browsing and retrieval. As more and more multimedia data are created and made available, segmentation algorithms can serve the important function of helping summarize this mass of material.

For video segmentation, the meaningful parts may be shots defined by camera changes, or scenes defined by semantically related shots. This is a difficult segmentation problem for machines (even for humans), because extracting sematics is hard. A poor but sometimes useful method is simple uniform partitioning at fixed time intervals (Mills, Cohen, & Wong, 1992). Resonable results have been obtained with algorithms that look at features such as color histograms. Standard methods involve variations of feature clustering algorithms (e.g.

see Boreczky & Rowe, 1996; Girgensohn & Boreczky, 1999; Uchihashi & Foote, 1999; Yeung & Yeo, 1997; Zhang, Low, Smoliar, & Wu, 1995).

There are several advantages of genetic algorithms over current methods for segmentation such as clustering. First, the genetic mechanism is independent of the prescribed evaluation function and can be tailored to support a variety of characterizations based on heuristics depending on genre, domain, user type, etc. Second, evolutionary algorithms are naturally suited for doing incremental segmentation, which may be applied to streaming media (e.g. video over the Web). Third, it can support dynamically updated segmentation that adapt to usage patterns, like adaptively increasing the likelihood that frequently accessed points will appear as segment boundaries.

In this paper, we will focus on image data streams and video. By *image data streams*, we mean a sequence of images. Of special interest is video, which can be considered as a type of image data stream consisting of frames. This may be produced video or raw video. Examples of produced video are news, movies, and training videos. Examples of raw video are video of meetings, surveillance records, and wearable personal video cameras (Mann, 1996). At a coarser level of granularity, slide shows are also image data streams. Another familiar type of image data stream is a collection of ordered visual documents, such as presentation slides, PowerPoint slides, and note pages with images such as produced by the NoteLook system (Chiu, Kapuskar, Reitmeier, Wilcox, 1999).

The method that we describe in this paper can be applied to non-image data streams. The genetic segmentation algorithm remains the same; what is required are different fitness functions that take into account the appropriate characteristics of that medium and software for processing that medium.

This paper is organized as follows. In Section 2 we define similarity adjacency functions for evaluating segmentations. Section 3 describes our Genetic Segmentation Algorithm (GSA), along with a discussion of the segment fair crossover operation. Section 4

provides some useful variations including incremental segmentation. Section 5 shows two examples demonstrating our technique for summarizing a video of a seminar and a set of note pages with images from a presentation. Section 6 discusses related work and compares our GSA to the classical GA and Genetic Grouping Algorithms (GGA).

2 EVALUATING SEGMENTATIONS

When characterizing and evaluating segmentations of video, the specific applications must be kept in mind. For the purposes of browsing and summarization, we define *similarity adjacency functions* with varying degrees of sophistication. In the simplest form, these functions only take into account image differences. In the more complex forms, information retrieval ideas are used.

2.1 PREPROCESSING

For segmenting video, it is a good idea to begin by reducing the number of frames. A video can have thousands of frames, and many adjacent ones are likely to be similar. We reduce the size of the set of images by only looking at those that are not too similar. For a non-video image document stream, this reduction step may be skipped. For a video recorded at 30 frames per second, we first subsample at a lower rate (one frame per half second is reasonable to capture the action in most domains), and call this set of images F. From this set, we pick out the least similar images by measuring their differences with the standard technique of color histograms (e.g. see Boreczky & Rowe, 1996). For any two images i and j we define

$h(i, j) = $ *histogram difference between i and j,*
$dh(i) = h(i-1, i)$ *with* $dh(0) = h(1, 0)$.

The number of elements is reduced by taking only those with *dh* greater than one standard deviation from the mean:

$$F' = \{ j \in F \mid dh(j) > \overline{dh} + \sigma \} .$$

The amount of reduction will vary depending on the genre of the video.

On this reduced set F', define the length of an element i

$\delta(i) = $ *number of frames in F from i*
to the next element in F'.

On F', define $dH(i)$ as we did $dh(i)$

$dH(i) = $ *histogram difference between*
the (i −1)-th and i-th elements of F'.

2.2 EVALUATION FUNCTIONS

There are a multitude of possibilities for an evaluation function. One can come up with a variety of characterizations based on heuristics depending on genre, domain, user type, and so forth. The applications that we have in mind are video summarization and indexing for browsing. We will define some fairly general functions that are based on considerations of image similarity, importance, and precedence.

Naively, one could take the k images with highest $dh(i)$ or $dH(i)$ and use these as the segment boundaries. For browsing and summarization applications in which these k images are the access points to an image data stream, this does not produce a very good segmentation because the most salient images may be similar to each other (even if they are not similar to their immediate neighbors) and too much repetition will occur in the result.

To take into consideration the relative differences among all the selected images, we define *similarity adjacency functions* as follows. Let S_k be a subset of k selected images in F', define

$$f(S_k) = \sum_{i, j \in S_k} \alpha(i, j) h(i, j) , \qquad (1)$$

where $\alpha(i, j)$ is a function for weighting the histogram differences.

For example, one simple way to specify this function is to set $\alpha(i, j) = 1$. A slightly more interesting definition is to put less weight on images that are farther apart by setting $\alpha(i, j) = 1 / |i - j|^2$ for $i \neq j$ and 0 else.

Due to the large cardinality of k-subsets of a set, there is no efficient standard algorithm to optimize (1) even for modest sized sets. One reason for using genetic algorithms is to be able to search this space effectively.

We can also apply information retrieval ideas by weighting each element by its *importance*. One way to define importance is to use a function that factors in the length of an element with its commonality, as in (Uchihashi & Foote, 1999), so that the longer and less common elements have greater importance. Unlike their algorithm, we do this without relying on a clustering of the images. First, we define a set C_i to be those elements similar to i,

$$C_i = \{ j \in F' \mid h(i, j) < \overline{dh} + \sigma \} ,$$

and let $W_i = |C_i| / |F'|$, then we define the *importance* based on length and commonality by

$$\log(\delta(i)) \log(1 / W_i).$$

Departing from (Uchihashi & Foote, 1999), we take the log of the length because $\delta(i)$ can have large variation. In the videos we looked at, the lengths of the elements of F' can differ by a factor of a hundred.

Furthermore, we extend this notion of importance by providing another factor related to the *precedence* of a frame, so that earlier appearing frames are more heavily weighted than later ones in the same similarity class. There are several reasons for using precedence as a criterion. For video, it has been noticed in video playback usage studies (see He, Sanocki, Gupta, & Grudin) that the earlier appearances of an event are accessed more. For images of people or slides, the earlier ones may introduce

or define things that the later ones will refer to. For video from surveillance or wearable personal video cameras, the frames can be processed backwards (or invert our precedence definition) to spotlight the most recent occurrences of interesting events.

Let $B_i = \{ j \in C_i \mid i \leq j \}$, we define the precedence factor by

$$P_i = |B_i| / |C_i|.$$

Putting together the factors for length, commonality, and precedence, we obtain the importance

$$I_i = P_i \log(\delta(i)) \log(1/W_i).$$

We put this into the evaluation function (1) by weighting each term with the average importance, i.e. in (1) we set

$$\alpha(i, j) = (I_i + I_j) / |i - j|^2 \text{ for } i \neq j \text{ and } 0 \text{ else.}$$

The evaluation function now reads

$$f(S_k) = \sum_{\substack{i, j \in S_k \\ i \neq j}} h(i, j) \frac{(I_i + I_j)}{|i - j|^2} \qquad (2)$$

Qualitatively, the effect of this similarity adjacency function is making more nearby images more dissimilar and permits a certain amount of repetition in the overall summary to capture the rhythm of the video.

Again, we emphasize that *any well-defined evaluation function may be used to characterize the desirable properties of segmentations and will work with the genetic mechanism of the algorithm.*

3   GENETIC SEGMENTATION ALGORITHM

First, we describe the input and output of our algorithm. The input is a video and an integer k for the desired number of segment boundaries. We used these boundary images as access points for indexing and summarization. The output is a sequence of k boundary images, plus their importance scores. A variation with varying k is described below. The importance scores may be used for layout purposes (e.g. see Uchihashi & Foote, 1999).

Our Genetic Segmentation Algorithm (GSA) can be described by specifying the encoding, fitness function, crossover and mutation operations. For more details on the basics of genetic algorithms, refer to (Goldberg, 1989). To run the algorithm, a population of individuals is randomly generated, and the evolution process is performed iteratively one generation at a time. In the end, the individual with the highest fitness is decoded to obtain a sequence of images for the segmentation.

3.1   ENCODING

For the encoding, we take a string of `0`'s and `1`'s like a classical GA as in (Goldberg, 1989). This string is called a *chromosome*. The video data stream structure lends itself to be divided into contiguous segments, so a string is sufficient. In contrast, the Genetic Grouping Algorithm (GGA) from (Falkanauer, 1991, 1998) uses sets.

The bit position of a chromosome string is an index for an element of the image data stream, e.g. a video frame in F', read left to right. The length of the string is the number of images $|F'|$. We use `1`'s to denote the segment boundaries; e.g. `00100010010` breaks up into the segments `00`, `1000`, `100`, `10`. In terms of the frames, the corresponding segments for $F' = \{ i_0, i_1, ..., i_{10} \}$ are $\{i_0, i_1\}$, $\{i_2, i_3, i_4, i_5\}$, $\{ i_0, i_0, i_8\}$, $\{ i_9, i_{10}\}$. The number of segments or `1`'s is set to be a fixed constant; this is given by the input specification of how many boundary images are desired.[1]

3.2   FITNESS FUNCTION

For the fitness function, we take the similarity adjacency function (2). Any well-defined evaluation function may also be used.

3.3   CROSSOVER AND MUTATION

The genetic mechanism works by randomly selecting pairs of individual chromosomes to reproduce for the next generation. The probability of a chromosome being selected is proportional to its fitness function value relative to the other chromosomes in the same generation. To reproduce, a crossover procedure is defined. In the classical GA, two chromosome strings reproduce by selecting a random bit for the crossing site, and the strings are sliced at the site and the two tail pieces are swapped and rejoined with the head pieces to produce two progenies. On the other hand, with GGA the chromosomes are not strings but subsets, and randomly selected subsets are recombined.

The stream structure allows our GSA to use a string structure like the classical GA, but instead of crossing at any bit, we cross only at segment boundaries; this is not unlike how groups are crossed in the GGA. What we do is to randomly select a segment, i.e. an index $i \in S_k$, with equal probability for each index. This index is used as the crossing site. The chromosome strings are crossed like the classical GA, plus an additional step to alter the resulting strings so that they have exactly k `1`'s in order to maintain the fixed number of segments.

Reducing the number of segments in a string is easy. We merge the partial piece sliced by the crossover procedure with an adjacent segment; this way, the segment boundaries coming from the earlier generation are preserved. Adjacent segments are then merged together until k `1`'s remain.

Increasing the number of segments in a string requires introducing new boundaries not inherited from earlier

---

[1] An alternative encoding is to set the leftmost bit to `1` for all segments.

generations. One way to do this is to pick a segment near the crossing site and split it at its weakest point, say the point with smallest *dH*. Alternatively, to reduce the amount of computation, we can use a mutation process to split the segments, which means randomly selecting a place to split. We use the latter for the work described in this paper.

Generally, mutation by random flipping of bits in the string is not a good idea for doing segmentation because it makes the segments rather unstable. Hence, for the basic version of GSA we do not do additional mutation beyond its use for increasing the number of segments in the crossover procedure.

We provide an example to illustrate. The following strings have 4 segments with segment boundaries on the left of the 1's:

```
00010010010
01000100100
```

Crossing at the point after the second segment of the first string, at site 6, we obtain

```
000100 | 00100
010001 | 10010
```

In the first string, a random bit ($i = 2$) is mutated to 1, increase the number of segments to 4. In the second string, the third 1 is flipped to decrease the number of segments to 4. The final results are:

```
00110000100
01000100010
```

Having described the encoding, fitness function, crossover and mutation (as part of crossover) operations, the genetic segmentation algorithm is specified.

3.4 SEGMENT FAIR CROSSOVER

The key difference between crossover in the classical GA and crossover in our GSA is that in the classical GA the crossover site is chosen with equal probability for each bit, whereas in our GSA the crossover site is chosen with equal probability for each segment. We call this *segment fair crossover*.

When we line up two parent chromosomes to perform a crossover, there will be short and long substrings where the alleles at contiguous loci in both parents are zero, like this:

```
loci number:  a b c         d
parent 1: ...10000000000000001000...
parent 2: ...00001000000000000001...
```

No matter which crossover site is selected in [a, b] and [c, d], the resulting progenies will be the same. In general, when there are k non-overlapping 1's in the two parent chromosomes, then there are only $2k + 1$ different crossover results possible. By choosing the crossover site with equal probability for each segment, our segment fair crossover uniformly samples from the $2k + 1$ possible

crossover results. The classical GA deviates from this uniform sampling of possible crossover results, because it is less and less likely to select a crossover site the shorter the intersecting substring of 0's.

Similarly, Faulkanauer's GGA could be described as a *group fair crossover*.

4 VARIATIONS

We now describe several variations of the basic GSA and similarity adjacency functions for handing incremental segmentation, and varying the number k of selected boundary images in the segmentation. When these images are used for browsing and summarization, we show a way to adapt segmentations dynamically to user access patterns.

4.1 INCREMENTAL SEGMENTATION

Because the algorithm is evolutionary, it is highly suitable for incremental segmentation. Streaming video and databases of accumulating image collections are examples where incremental segmentation and summarization can be useful. Basically, the system maintains a population of segmentations and lets it evolve as new video images are added. The good image segment boundaries that have been found are more likely to survive. For each generation, the individual with the highest fitness is used to determine the segmentation.

Between generations, new images are added. First, they are preprocessed as in Section 2.1 by keeping a running average of *dH*. To keep the chromosome length bounded when new images are added, old ones can be removed by throwing out the ones with low importance or low *dH*. This works because in equation (2), *epistasis* (how the bit positions combine to affect the fitness function) is well behaved. It is clear by looking at the equation that dropping lowly rated images has little effect on the fitness. To keep k fixed, if a chromosome loses a bit position marked by a 1, one of the new bit positions is randomly set to 1. Most of the new bit positions are set to 0, but occasionally (say with probability one over the length of the chromosome) a bit position is set to 1, and a random segment is merged to keep k fixed.

4.2 VARYING *K*

We now describe a way to vary k, the number of segment boundary images. We do this by normalizing the evaluation function (2) to a prescribed target k_0:

$$g(S_k) = f(S_k) / (1 + | k - k_0 |),$$

and simplifying the crossover so as not to keep k constant. This way, the number of images will not be exactly k_0, but some number around k_0 that provides a potentially better segmentation.

4.3 ADAPTING TO USER ACCESS PATTERNS

The segmentation can be dynamically updated to reflect user access patterns. The most frequently accessed images by users are weighted more heavily in the importance term of equation (2), and the segmentation is incrementally updated. The update schedule may be daily, weekly, or longer.

Let a_i be the number of times an image i has been accessed or viewed by users, then we define the *access frequency* factor by $A_i = 1 + log (1 + a_i)$. The following is then used for the importance in equation (2):

$$I_i = A_i P_i \log(\delta(i))\log(1/W_i).$$

5 EXAMPLES

We illustrate with two examples by applying the GSA to summarize an hour-long seminar video and a set of note pages containing annotated slide images and pictures. These examples were computed *after* the algorithm had been developed—the algorithm was not tuned to them.

We have used a fairly simple implementation of the algorithm and fitness function in these examples to demonstrate that it works on real data and that it is a promising technique. More extensive testing on a large corpus of data would be required to establish the efficacy of the GSA.

5.1 PARAMETERS

This algorithm works with a small number of parameters, both in the fitness function definition and in the genetic mechanism. We ran the GSA as a generational fitness proportional GA. Crossover is used for 100% of the genetic operations. Mutation was introduced only during crossover segment splits, as defined in section 3.3. Elitism was not used. Fitness function (2) was employed. The algorithm was implemented in C++. Hardware was a normal single processor Pentium 350.

5.2 VIDEO OF A SEMINAR

The GSA is applied to the video with $k = 5$ and 11, with population size of 2000 run over 100 generations. The length of the video is about an hour, and subsampling and reduction gave $|F| = 7765$ and $|F'| = 149$. Our hardware and software were by no means optimal, but to give some indication of performance, the preprocessing (described in section 2.1) took about 2 minutes and the GSA took about 3 minutes.

For images used in the summary, we take the boundary images plus the first frame of the first segment. We choose 6 and 12 images because these numbers have several divisors and can produce different layouts (e.g. 6 gives 1x6, 2x3, 3x2, 6x1). The results are shown in Figures 1−2. Figure 3 illustrates the whole video sampled at every 5 seconds.

The three topics of the seminar talk were "Active Messenger," "comMotion," and "Nomadic Radio," and the video images of the three slides introducing these topics were selected along with two pictures of the speaker and a picture of the room. The result is remarkably good and it would be difficult for a person to select a much better set of representative images for a summary.

For $k = 5$, GSA found the global maximum, which we checked by brute force computation. For moderately larger values of k, the combinatorial explosion in (2) makes brute force infeasible. While it may be possible come up with a tractable algorithm to optimize this specific function, the advantage with genetic algorithms is that the evaluation function can be tailored to focus on whatever features are desirable.

5.3 NOTE PAGES WITH IMAGES

In this example, we demonstrate the GSA on a stream of visual documents. Our data are note pages from the FXPAL NoteLook system (Chiu, Kapuskar, Reitmeier, Wilcox, 1999), which contains a variety of visual images: presentation slide images, pictures, and ink annotations. One can also apply the GSA to simpler image data streams such as collections of presentations slides (e.g. PowerPoint slides).

As in the above example, we choose $k = 5$ and 11. The GSA is applied to the note pages with a population of 1000 run over 100 generations. The results are shown in Figures 4−5, with the full set of note pages shown in Figure 6.

6 RELATED WORK

We have done some experiments with a classical GA (as described in Goldberg, 1989), with crossover at random bit positions as opposed to segment boundaries, to optimize similarity adjacency functions and they failed to converge. We have not tried using Falkanauer's Genetic Grouping Algorithm, but since it operates on sets rather than string segments, it is less natural than the GSA for the structure of data streams.

Other work using genetic algorithms/programming for image analysis has been done (e.g. Poli, 1996; Tackett, 1993), but these mainly analyze the features of a fixed image and are not aimed at segmentation of an image data stream or video.

7 CONCLUSION

We described a genetic segmentation algorithm for image data streams and video that employs a segment fair crossover operation. We performed experiments to demonstrate that it works on real data. As video analysis is a relatively new field without an established corpus for testing, more experience with the GSA and standard algorithms like clustering will be required to gauge their

respective strengths and weaknesses. There are some notable advantages with the GSA: it offers more tailorability in the evaluation function and enables incremental segmentation.

Acknowledgments

References

Boreczky, J.S. and Rowe, L.A. (1996) Comparison of video shot boundary detection techniques. *Storage and Retrieval for Still Images and Video Databases IV, Proceedings of SPIE 2670*, pp. 170-179.

Chiu, P., Kapuskar, A., Reitmeier, S., and Wilcox, L. NoteLook: Taking notes in meetings with digital video and ink. *Proceedings of ACM Multimedia '99*. ACM, New York, pp. 149-158.

Falkanauer, E. (1998) *Genetic Algorithms and Grouping Problems*. Wiley.

Falkanauer, E. (1991) A genetic algorithm for grouping. *Proceedings of the Fifth International Symposium on Applied Stochastic Models and Data Analysis,* Gutierrez, R. and Valderrama, M.M. (eds.), 1991. World Scientific Publishing Co., Singapore, pp. 198-206.

Girgensohn, A., and Boreczky, J. (1999) Time-constrained keyframe selection technique. *Proceedings of the 1999 IEEE International Conference on Multimedia Computing and Systems*. IEEE Computer Society, vol. 1, pp. 756-761.

Goldberg, D.E. (1989) *Genetic Algorithms in Search, Optimization, and Machine Learning.* Addison-Wesley.

He, L., Sanocki, E., Gupta, A., Grudin, J. (1999) Auto-summarization of audio-video presentations. *Proceedings of ACM Multimedia '99.* ACM Press, pp. 489-498.

Mann, S. (1996) 'Smart Clothing': Wearable multimedia computing and 'personal imaging' to restore the technological balance between people and their environments. *Proceedings of ACM Multimedia '96.* ACM Press, pp. 163-174.

Mills, M., Cohen, J., and Wong, Y.Y. (1992) A magnifier tool for video data. *Proceedings of CHI '92.* ACM Press, pp. 93-98.

Poli, R. (1996) Genetic programming for image analysis. *Proceedings of Genetic Programming 1996.* MIT Press, pp. 363-368.

Tackett, W.A. (1993) Genetic programming for feature discovery and image discrimination. *Proceedings of the 5th International Conference on Genetic Algorithms*, pp. 303-309.

Uchihashi, S. and Foote, J. (1999) Summarizing video using a shot importance measure and frame-packing algorithm. *Proceedings of ICASSP '99*, vol. 6, pp. 3041-3044.

Yeung, M.M. and Yeo, B-L. (1997) Video visualization for compact presentation and fast browsing of pictorial content. *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 7, no. 5, pp. 771-785.

Zhang, H.J., Low, C.Y., Smoliar, S.W., and Wu, J.H. (1995) Video parsing, retrieval and browsing: An integrated and content-based solution. *Proceedings of ACM Multimedia '95.* ACM Press, pp. 15-24.
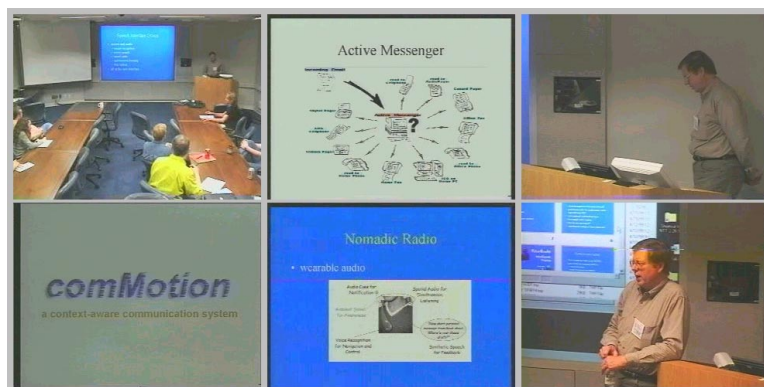

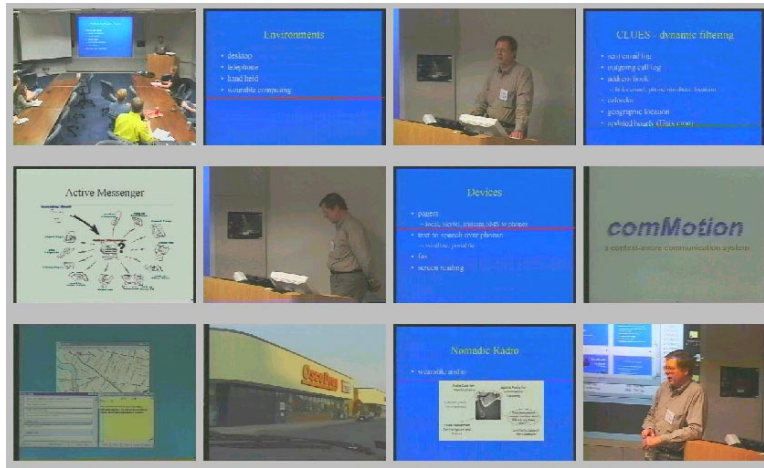Figure 1. Video segmented by GSA with $k = 5$

Figure 2. Video segmented by GSA with $k = 11$

Figure 3. Video of an hour-long seminar sampled every 5 seconds. In our computation with GSA, the video was sampled every 0.5 second—these are too numerous to be shown here. The 6 highlighted images correspond to the result in Figure 1 (within 10 frames).
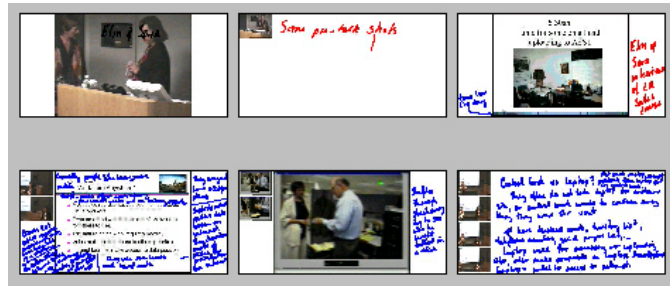
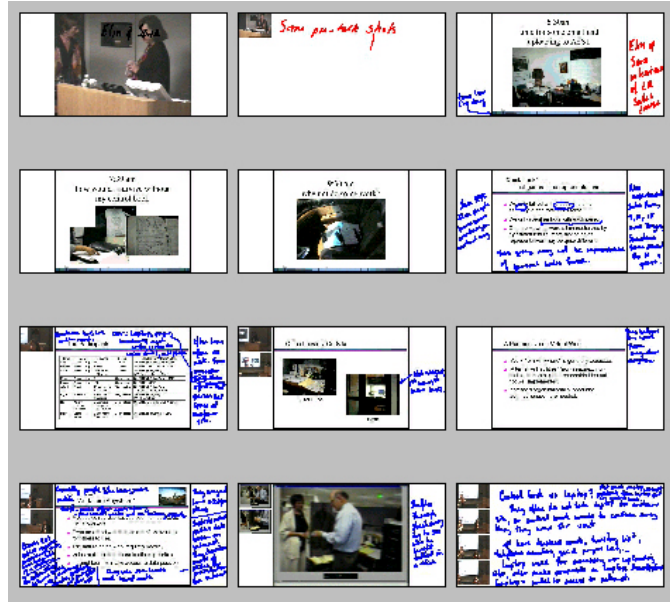Figure 4. Note pages with images segmented by GSA with $k = 5$.

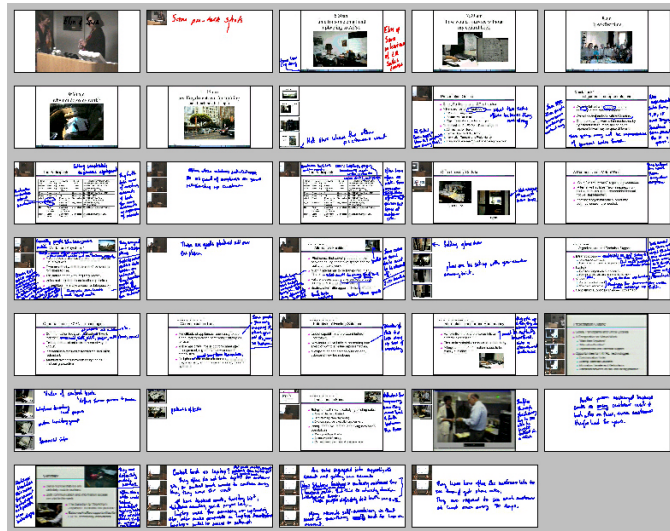Figure 5. Note pages with images segmented by GSA with $k = 11$.

Figure 6. Full set of note pages with images from an hour-long presentation.