Graph Based Video Segmentation

Abhejit Rajagopal, Rohan Jain, B.S. Manjunath
{*abhejit, rohanjain*} *@ece.ucsb.edu*
Department of Electrical and Computer Engineering, University of California, Santa Barbara

March 14, 2015

Question Sheet

Q1: In 2-3 sentences, describe the main project goal.
A: Our goal was to create a user-friendly library for graph-based video segmentation in Python. Our implementation, referred to as the "Godfather Algorithm", aims to highlight the generic applicability the graph-based method presented originally by Felzenszwalb [1] and extended by Grundmann [2].

Q2: On a scale of 1 (easy) -10 (impossible), quantify if your original goal was reasonable.
A: 5 (possible but difficult). We had to both conceptualize and implement algorithms only discussed (not outlined) in the paper [2].

Q3: On a scale of 1-100, what is your evaluation of the percent work that you were able to complete, with respect to your initial goal.
A: 95, because particular extensions of our code are somewhat inefficient (e.g. Level-1 of the histogram-based hierarchical segmentation occupies more than 50% of the computation time).

Q4: Do you think you were able to achieve the overall base objectives of the project (which is graded for 40% of your course grade) (YES or NO)
A: YES.

Q5: Are you asking for EXTRA CREDIT? (YES or NO)
A: YES (10%).

Q6: If your answer to Q5 is YES, please identify (2-3 sentences) additional work that you did beyond the base objectives that deserve the extra credit. You should elaborate this more in your detailed report.
A: (1) We undertook and finished the video segmentation project, which required us to implement graph-based image segmentation first.
(2) We did not use or even refer to the source code made available by Grundman et al. We felt it was not documented well enough to be read.
(3) We had to implement in-house predicates as they were not talked about in the paper in detail.

Q7: Did you write all of the software for your project? (YES or NO).
A: YES and NO. Our implementation is unique, and we wrote all required software for the project. However, we required other libraries to pre-process input, output, and perform some calculations.

Q8: If your answer is NO, please list ALL the sources you used for your project.
A: We inherited two classes, Node and Forest, from a graph-based image segmentation released in 2012 [3]. Also, we used Numpy (matrix algebra), Scipy (image IO), OpenCV (dense optical flow), and FFMPEG (video IO) [4] [5].

We implemented graph-based video segmentation in Python. Our algorithm, termed the "Godfather Algorithm", is adapted from the method originally proposed by Felzenszwalb for image segmentation [1] and is based on the basic idea that each (n-dimensional) voxel ultimately belongs to a larger voxel group (supervoxel) that is identified by one voxel label, the "Godfather". In short, we use a medley of merging predicates to determine if a particular voxel or group of voxels belongs to a particular family headed by a Godfather.

1 Methodology

Our method follows from the implementation of Grundman et al [2]. We start by over-segmenting the space-time volume, and proceed to the formation of superpixels using various merging predicates. Pixels are initially clustered into groups (identified by their Godfathers) using intensity and spatial similarity (r, g, b, x, y, t). The resulting Godfathers are then clustered using region-based graph approaches. Dense optical flow is used throughout the video to ensure temporal coherence in Godfathers over the spatio-temporal volume. To achieve a segmentation using a modest amount of memory, we break the initial video-volume into windows ranging from 2 to 30 frames and subsequently merge each video-graph using a combination of flow and histogram-based predicates. This final step can exploit parallelism, which can improve the runtime by a linear factor.

1.1 Oversegmentation

This initial step in our framework clusters of Individuals (voxels) into small families. This is done by defining a graph over the windowed video-volume; where nodes represent voxels, and edge weights represent the the similarity between two nodes. Initially, edge weights are calculated solely from differences in spatio-temporal intensities *(Equation 1)*. In order to make this computation efficient, only edges which lie in a 26-connectivity matrix are considered.

$$\begin{gathered} e(i,j) = (i, j, ||f_i - f_j||_2), \\ \text{where } f_k(x,y,t) = (r,g,b) \\ \text{and } f_j \in f_i(x \pm 1, y \pm 1, z \pm 1) \forall (i,j) \end{gathered} \tag{1}$$

The merging predicate used *(Equation 2)* compares the internal difference of the superpixels with the minimum difference between pixels belonging to each superpixel. The internal difference signifies the internal entropy of a region and is calculated by finding the maximum weight encountered in the minimum spanning tree of the graph which spans only the pixels belonging to the particular superpixels.

$$\begin{gathered} Int(A) = \max w(e), \text{ for } e \in A \\ Diff(A,B) = w(e), \text{ for } e\text{between A and B} \\ if\ Int(A) <= Diff(A,B) \\ \text{and } Int(B) <= Diff(A,B): \\ merge(A,B) \end{gathered} \tag{2}$$

1.2 Minimum Region Size

Following this step, we enforce a lower bound on the size of each Family *(Equation 3)*, which is computed as the number of individuals under a Godfather. To accomplish this efficiently, we force neighbouring Godfathers which are below the minimum bound to cluster. However, this should be done in relative terms so that we don't merge Godfathers that are relatively very different.

$$\begin{gathered} \text{if } \#A < threshold \text{ or } \#B < threshold: \\ merge(A,B), \\ \text{where } \# \text{ is the cardinality of a set.} \end{gathered} \tag{3}$$

Once this step is completed, we are left with spatio-temporal regions of meaningful size, allowing us to increase the level of abstraction of our algorithm. The following predicates operate on Families, using region-based properties to further segment the video volume.

1.3 Dense Optical Flow

To increase the temporal coherence of Families within a window, we use dense optical flow to identify edges between voxels that lie along a backward flow vector. Although an aggressive approach would consider edges along the backward flow vectors of each of the 8-connected neighbors of a voxel belonging to a frame, we disregard these for ease of calculation. Optical flow assumes that a supervoxel (Family) maintains its visual identity on each frame. Accordingly, we only create edges between Families which contain voxels (Individuals) which are connected along the computed flow vector *(Equation 4)*. Our procedure implicitly makes two simple assumptions:

(1) The pixel intensities of an object do not change between consecutive frames.

(2) Neighbouring pixels have similar motion.

$$e(i,j) = (G(i), G(j), w(G(i), G(j)), \quad (4)$$
where G(k) identifies the Family
headed by the Godfather of k

Once we have identified edges for a graph based on dense optical flow vectors, we apply region-based merging predicates to define the weights, as explained in the following section.

1.4 Region-based Histogram

Each node in the graph now represents a Family, or a group of pixels, identified by a pixel label called the "Godfather". Once possible edges between Families (or equivalently, between Godfathers) have been identified, their edge weights can be calculated using a distance metric based on color-intensity histograms. A popular approach to this characterization is to compute the the chi-squared difference between neighboring histograms *(Equation 5)*. In order to merge a region with another neighboring region, this chi-squared histogram difference is compared to the standard deviation within in a region, which can also be derived from the histogram.

In our work, we experimented with several histogram comparison methods, including the dot-product of normalized histograms *(Equation 6)* and our own method, which we call the *Difference of Histogram (DoH) (Equation 7)*, which is analogous to the L1-norm of the histogram similarity. The DoH region metric computes the histogram of the difference between histograms, and applies an inverse weighting on the bins. In the most rudimentary implementation, the first bin [0] can be selected. Essentially, this measures the number of histogram bins which differ negligibly.

$$w(A,B) = \sum_{k=1}^{n} \frac{(hist(A)_k - hist(B)_k)^2}{hist(A)_k + hist(B)_k} \quad (5)$$

$$w(A,B) = hist(A) \cdot hist(B) \quad (6)$$

$$w(A,B) = hist(hist(A) - hist(B))[0] \quad (7)$$

An interesting observation here is that neighbouring regions can be identified from the spatial information of the previous stage of oversegmentation. Specifically, the unresolved edges from the initial oversegmentation reflect all (and only) possible neighboring regions. Utilizing this information would reduce the computational complexity of our region-based graph segmentation implementation, which is currently the bottleneck in our framework.

2 Computational Complexity

In this section, we analyze the complexity of our merging routines. The time and memory complexity is largely dependent on the data structures used to depict the final annotation. Trees and hash tables are preferable to matrices, but in either case the update time must be short when sorting a merging a large number of nodes in the video-graph.

2.1 Oversegmentation

Our implementation uses a structured graph using a collection of sorted trees. As such, have a search complexity of O(nlogn), and a memory complexity of O(n), where n is the number of pixels in a window. These data structures are used for Predicates 1-3, (oversegmentation, minimum-size, and pixel-based dense optical flow)

2.2 Region-based segmentation

Our implementation employs a 3-dimensional component matrix for identifying Godfathers, and a companion hashtable (Python dictionary) for identifying Families. As such, our search is extremely fast, having a time complexity of $O(1)$, although our memory complexity is $O(n)$. In this respect, a linear memory complexity cannot be avoided since eventually every pixel must be represented in memory. However, our implementation performs updates very slow (worst case $O(n^2)$), which significantly degrades our performance compared to Grundmann's C++ code. This can be improved by combining the oversegmentation forest (collection of trees) with a companion hashtable. The search complexity will then be $O(1)$, memory complexity of $O(n)$, and update complexity of $O(2n)$.

3 Results

The referenced hierarchical graph-based video segmentation paper [2] takes on an average 23 minutes to segment a 10 second video. We used the same videos to test our method and achieved video segmentation of a 4-5 second clip in 90 minutes. The

initial oversegmentation and region merging based on optical flow completes in under 40 minutes, while the region-based segmentation occupies more than 50 percent of the time.

Our segmentation results are also perceptually worse than the results presented by Grundmann. Even so, we still manage to segment a variety of fine details, as can be seen the the following frames of the movie "No Country for Old Men".

4 Scope of this Work and Possible Improvements

One of the challenges associated with video segmentation is that the target method should be agnostic to the kind of video it is segmenting. Parameters should therefore be translated as features in the video. In the current setup however, parameters are manually varied to get the perceptually best segmentation.

As mentioned earlier, a more rigorous implementation of optical flow is required to deal with temporal incoherence.

Though we did use in-house predicates, we have not made comparisons with the state-of-the-art predicates and features. This could very well be an area of improvement for our algorithm.

A major point of concern for us is the time of execution of the Godfather algorithm when the size of regions increase. We have addressed these concerns in the Computational Complexity section, but ultimately our implementation will require a shared-memory architecture, so predicates may be evaluated in parallel.

5 Teaming

The authors contributed equally to this work. Rohan's experience with image processing complemented Abhejit's aptitude in scientific computing and software development. Each module of the project was worked on in parallel and one's work was scrutinized in detail by the other. Both team members learned a lot by working as a team, and plan to continue their collaboration in order to rival the state of the art work done by Grundmann et al [2].



Figure 1: *Clip from "No Country for Old Men" (a) Original Frame, (b) Oversegmentation, (c) Dense Optical Flow, (d) Region-based Segmentation*

References

[1] Felzenszwalb, Pedro F., and Daniel P. Huttenlocher. "Efficient graph-based image segmentation." International Journal of Computer Vision 59.2 (2004): 167-181.

[2] Grundmann, Matthias, et al. "Efficient hierar-

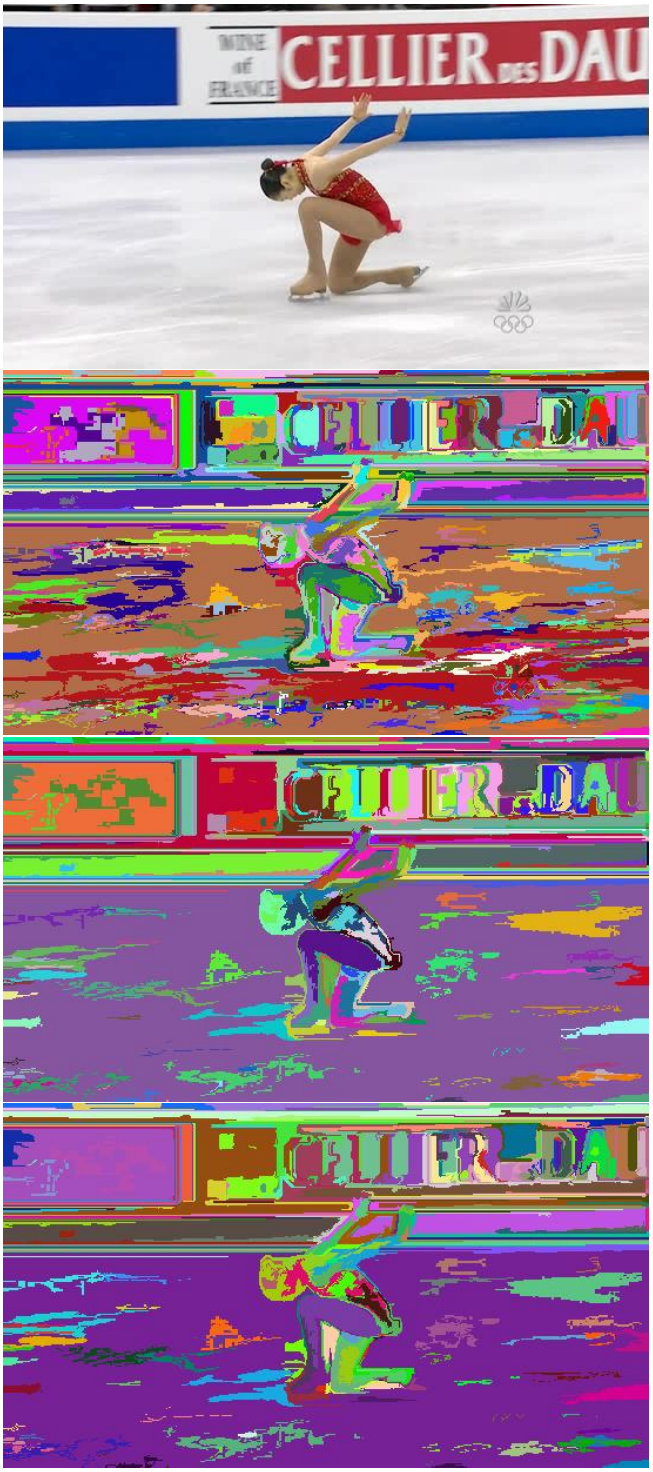


Figure 2: *Clip of athlete Yunakim (a) Original Frame, (b) Oversegmentation, (c) Dense Optical Flow, (d) Region-based Segmentation.*

chical graph-based video segmentation." Computer Vision and Pattern Recognition (CVPR), 2010 IEEE Conference on. IEEE, 2010.

[3] A python implementation of the algorithm described in the paper Efficient Graph-Based Image Segmentation by Luis Gabriel. https://github.com/luisgabriel/image-segmentation

[4] OpenCV documentation on Optical Flow.

[5] Documentation of Scikit-Image, Python Imaging Library, FFMPEG.