Efficient Graph-based Image Segmentation

TianChen Jin, Weiting Lin, Steven Munn

Friday, March 13, 2015

1 Questions

1. In 2-3 sentences, describe the main project goal.

   Our project implements *Efficient Graph-based segmentation* by Felzenszwalb **??**, first in C/C++, then as a mex wrapper for use with MATLAB, and finally, in Android for a mobile application. Our main goals for the implementation are accuracy and speed relative to the author's original algorithm and implementation. To this end, we plan to perform benchmark tests and comparisons for a quantitative analysis.

2. On a scale of 1 (easy) -10 (impossible), quantify if your original goal was reasonable.

   7: the goal presented some challenges, especially for efficient implementation, but it was reasonable for a team of 3. This is why we decided to extend it and add more features. Specifically, we implemented a minimum size for the components, experimented with building the graph in a 5-D feature space, and we tried different color spaces for the old algorithm and the 5-D space one.

3. 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.

   120% with the extensions and experiments we added, we did more than we had set out to do originally.

4. Do you think you were able to achieve the overall base objectives of the project (which is graded for 40

   Yes, absolutely.

5. Are you asking for EXTRA CREDIT? (YES or NO)

   Yes.

6. 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.

   Our extra credit objective was to implement and adapt the same algorithm on an Android phone. This involves rewriting code that will work in the context of Android's limited memory availability and library system that isn't compatible with some of the tools we had originally used.

7. Did you write all of the software for your project? (YES or NO).

   Yes, except for the benchmark comparisons. And, we used 3rd party libraries detailed below.

8. If your answer is NO, please list ALL the sources you used for your project.

   We used OpenCV for reading and writing files, for Guassian blur, and for approximate nearest neighbor search. And, we used BOOST for the exact k-nearest neighbor search.

   For comparison we used the authors' available at,

`http://cs.brown.edu/~pff/segment/`

For a presegmened image dataset we used the image available at,

`http://www.wisdom.weizmann.ac.il/~vision/Seg_Evaluation_DB/dl.html`

2 Final Technical Report

2.1 Introduction

Image segmentation is a well studied problem and their are many existing methods. These range from simple pixel manipulation (thresholding or watershed for example) to more global methods like mean-shift and graph cut. Among these methods, graph-based techniques are among the most popular methods these days due to their efficiency and segmentation accuracy. The method described in Felzenszwalb *et al.* **??** starts off with every pixel as a component. Then it iterates through all edges in ascending order based on pixel "differences" (defined in certain way), and merges pixels based on the edge weight. In a high-level sense, this graph-based method maximizes inter-component distance and minimizes intra-component distance.

2.2 Implementation Outline

For efficiency, we write the core components of our code in C++. At a high-level, there are two main time consuming stages: building the graph and segmenting it. To build the graph we iterate through the 8 nearest-neighbor pixels to produce a list of edges called the adjency list and a list of weights computed using a measure for the difference between pixels. We store the adjancency list in a preallocated integer array and the weights in a preallocated double array. Our choice of data structure is due to the fact the next function will sort these lists in increasing weight order.

For the segmentation, we use disjoint sets, as mentioned in the paper, to store the different clusters. Our implementation of this data structure uses path compression to find the set of each pixel efficiently once the segmentation is done. The code first sorts the adjancency list according to edge weights, then initializes the pixels to their own set. It iterates through the entire adjency list and performs a comparison to decide whether or not to merge.

Producing the colored output image is actually a significant step as well though (with regards to time consumption). The coloring function must iterate through each pixel and find the set in which it belongs. We use an unordered map, or hash table, to store the color values corresponding to each set and set the pixel colors accordingly.

We present a more detailed breakdown of the code's performance in the benchmark section; however, in is important to note that the segmentation step is significantly more time consuming that the other steps, as expected. We can build the graph in less than 1.5 seconds for a 3187 by 2015 pixel image. The segmentation takes less that 6 seconds, and the coloring less than 1.

2.3 Segmentation Algorithm

One of the major difference between our algorithm and the authors' is that we impose a minimum compenent size right at the beginning of the segmentation process whereas they impose it as a post-processing condition. This leads to less noisy components (small compenents with no sematic information) in the final result. The reason is that our method seeds compenents in noisy regions and facilitates component expansion where it would otherwise not occur.

Since one of the paper's main objectives is to collect noisy regions into components to account for textures, we believe our minimum size pre-condition is an improvement. The benchmark provides a quatitative backing for this assertion.

References

[1] Felzenszwalb, P. F., & Huttenlocher, D. P. (2004). *Efficient graph-based image segmentation.* International Journal of Computer Vision, 59(2), 167-181.