An implementation of Efficient Graph-Based Image Segmentation

Sahil Narang
Kishore Rathinavel
University of North Carolina, Chapel Hill

Abstract

This report documents an implementation of the paper "Effective Graph-Based Image Segmentation". The method discussed here defines a metric for measuring the evidence of a boundary between two regions using a graph-based representation of the image. Based on the proposed metric, an efficient image segmentation algorithm is developed. Although this algorithm is a greedy algorithm, it respects some global properties of the image. Some important features of the proposed algorithm are that it runs in linear time and that it has the capability to adapt its behavior differently between regions of high-variability and low-variability. In particular, we demonstrate that it ignores details in high-variability regions.

Index Terms

image segmentation, graph algorithm

I. Introduction

A wide range of intermediate and high level computer vision problems would benefit from a reliable segmentation method. It is desirable in most computer vision problems to be able to extract regions which belong to the same object. Some examples where image segmentation could be an important pre-processing step are object detection, recognition tasks, content-based image retrieval etc. In order to build a reliable and useful algorithm which can perform image segmentation, the algorithm needs to satisfy these properties:

1) Capture regions which appear to belong together. This should be consistent with the global appearance of the object
2) High efficiency and run in time linear in the number of pixels in the image.

II. Related work

Image segmentation has been approached from a variety of ways and there exists an extremely large body of literate to summarize and compare with all methods. Here, we describe work that has been done related to the method we describe. This paper was published after the formulation of graph by Shi and Malik [1], Wu and Leahy [2] and spectral methods [3]. The work by Wu and Leahy [2] is biased towards finding small components. This bias was addressed by the *Normalized cuts* by Shi and Malik [1]. However, the normalized cuts criterion yields an NP-hard computational problem. This can be resolved by using some approximations. However, these approximations have not been characterized and are not well-understood. However, all graph-based are too slow for real time implementation. The computational cost for our algorithm is nearly linear with the number of pixels. The execution time is extremely fast and we get a very good segmentation as detailed in the following sections.

III. Our method

In this section, we first introduce the basic concepts of graph-based segmentation, then describe our pairwise region comparison metric, and then describe our algorithm and some properties of it.

A. Graph-based segmentation

Let $G = (V, E)$ be an undirected graph with vertices $v \in V$ and edges $(v_i, v_j) \in E$. Edges are between two vertices always. The set that needs to be segmented is the set of vertices V. Each edge has a corresponding weight $w(v_i, v_j)$ which is a non-negative measure of the dissimilarity between neighboring elements v_i and v_j. Dissimilarity between the edges can be measured as the difference in intensity, color, motion, depth, location or any other local attribute. In our experiments, we define the weight to be the L2 norm between the RGB values of the pixels. The goal of image segmentation is to find a partition of the set V such that each component of the partition is a connected graph $G' = (V, E')$ where $E' \subseteq E$. The driving forces of such a partition are that the elements of each component are similar to other elements belonging to the same components and yet dissimilar to elements belonging to other components. Based on our guideline that the weights measure the amount of dissimilarity between two vertices, we can that the edges between vertices of the same component will have low weights whereas edges between vertices belonging to different components will have high weights.

B. Pairwise region comparison metric

An absolute metric would not take into account the variability of the region. This would result in either merging together separate regions with low variability between them or separating high variability regions into several components. To avoid this, we propose a metric which adapts itself based on the variability of the region under consideration. The metric is based on measuring the dissimilarity between elements along the boundary of two components relative to the measure of dissimilarity of the elements within each component. We define internal difference to be the largest edge weight in the minimum spanning tree of the component. That is,

$$\text{Int(C)} = \max_{e \in MST(C,E)} w(e) \tag{1}$$

where, $MST(C, E)$ is the Minimum Spanning Tree of the sub-graph $G = (C, E)$. A minimum spanning tree is the sub-graph which connects all vertices C and has the least sum total of weights.

We define the difference between two components to be the minimum edge weight connecting the two components. That is,

$$Dif(C_i, C_j) = \min_{v_i \in C_i, v_j \in C_j, (v_i, v_j) \in E} w(v_i, v_j) \tag{2}$$

If there is no edge connecting C_i and C_j, we let $Dif(C_i, C_j) = \infty$. While this error metric could be made more robust to outliers by using some statistical measure like median, mean or quantile, it makes the problem NP-hard. The proof for this claim is provided in the paper by Felzenszwalb and HuttenLocher [4] which we omit here. The criteria for a merge between two components is to check if the difference between the two components is lesser than the internal difference of each of the components by a threshold function. In particular we compare $Dif(C_i, C_j)$ and:

$$MInt(C_i, C_j) = min(Int(C_i) + \tau(C_i), Int(C_j) + \tau(C_j)) \tag{3}$$

If the former is lesser than the later, the components are merged, otherwise, the components are not merged and we conclude that there is a strong evidence of a boundary between the two components. The threshold function is defined as:

$$\tau(C) = k/|C| \tag{4}$$

The threshold function above implies that for small components, we require a strong evidence for a boundary. A large k prefers larger components and vice versa. Instead of defining the $\tau(C)$ based on some constants and cardinality of the components, it is possible to define $\tau(C)$ based on prior information to favor some desired shape.

C. Algorithm

We first form the graph $G = (V, E)$ by considering all pixels to form the set V and a choose a neighborhood around each pixel to find the edges around that pixel. Generally, we could choose a 4-neighborhood or a 8-neighborhood. For the rest of the paper, we use a 8-neighborhood. The weights of the edges are defined to be the absolute intensity differences between the pixels forming the edge.

The Algorithm 1 returns a segmentation that is neither too fine nor too coarse. For a proof of this property, refer to the paper by Felzenszwalb and HuttenLocher [4].

Algorithm 1 Graph-based Image segmentation

Input: $G = (V, E)$ and $w(v_i, v_j) \forall v_i, v_j \in V$ and $v_i \neq v_j$

1: Sort E into $E' = (0_1, ..., o_m)$ by non-decreasing edge weight
2: Start with a segmentation S^0 where each vertex v_i is in a component by itself
3: Let $o_q = (v_i, v_j)$. Repeat step 3 for q = 1,...,m to find S^q given S^{q-1}
4: **if** v_i and v_j are in disjoint components of S^{q-1} **then**
5: **if** $w(o_q)$ is lesser than $MInt(C_i, C_j)$ where $v_i \in C_i$ and $v_j \in C_j$ **then**
6: Merge C_i and C_j
7: **end if**
8: **end if**

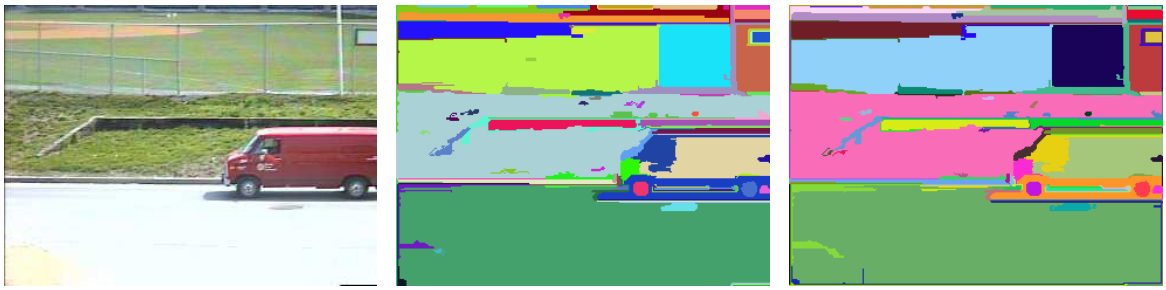
Output: S^m

(a) Scene (b) Author's implementation (c) Our implementation

Fig. 1. Scene 1

IV. Implementation Details

In this section we describe the date structure we use for the implementation of the algorithm, then describe the metric we use to calculate weights. The implementation of graphs and components is best done by the disjoint-set forest with union by rank and path compression [5]. Both these strategies, union by rank and path compression are aimed at minimizing the parsing time from any node to the root node. In a tree data structure, each node holds a reference to its parent node. In a disjoint-set forest, each set is represented by the root of set's tree. A merging operation is an operation that combines the trees of 2 sets into 1 tree. This raises an interesting question of which tree's node now becomes the root node of both sets. We use the terms depth and rank interchangeably here. Keeping in mind that the parsing time from any node to the root node mainly depends upon the depth of the tree, we should combine the trees in such a way that the resulting tree has the minimum depth possible. So, the tree with the smaller depth gets added under the root of the deeper tree. If both trees have equal depth, then the choice of root node doesn't matter and pick either one. Whenever we traverse from a node to its root node, we would pass through a certain order of nodes. All these nodes belong to the



(a) Scene (b) Author's implementation (c) Our implementation

Fig. 2. Scene 3

(a) Scene (b) Author's implementation (c) Our implementation

Fig. 3. Scene 4

same root and hence, they could all be attached to the node for the purposes of our algorithm. This practice is called path compression. Step 1 of the algorithm 1 can be done in linear time for integer weights by counting sort and in $O(Slocum)$ for floating values. Steps 2 to 7 of the algorithm 1 take $O(m(m))$ time, where α is the very slowly-growing inverse Jermain's function.

V. Results

In order to reduce the effects of noise, we use a Gaussian filter to smooth the image. We always use a Gaussian with $\sigma = 0.8$ which does not reduce any edge details but removes artifacts. Fig. 1 shows a synthetic since for image segmentation, the author's implementation and our implementation. Looking at the synthetic scene, most people would say that the scene should be divided into 3 regions - the left half of the image, the constant intensity portion of the right half of the image, and the region of high variability in the right half of the image. Our algorithm demonstrates that if you adaptively ignore details in regions of high variability, then it is possible to segment out such regions. Fig. 2 shows an outdoor scene. The largest components found by the algorithm are 3 of the grassy areas behind the fence, the grassy slope, the van, and the roadway. Note that the van is not uniform in color in the scene but these are treated as internal variations and merged together to form 1 coherent region. Fig. 3 shows 2 baseball players. In their paper, their segmentation is very similar to our implementation's results. But, for some unknown reason, their code seems to work better now and is able to segment the grass separately from the back wall in the region under the player.

VI. Conclusions

In this paper, we introduced a greedy algorithm for image segmentation which respects the global properties of the image. We used a novel metric to compare 2 components to decide whether or not they should be merged. This metric adapts to regions in such a way that it ignores details in regions of high variability and uses details in regions of low variability. This algorithm is nearly linear in time - in particular it is $O(nlogn$ where n is the number of pixels in the image.

References

[1] J. Shi and J. Malik, "Normalized cuts and image segmentation," *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, vol. 22, no. 8, pp. 888–905, 2000.

[2] Z. Wu and R. Leahy, "An optimal graph theoretic approach to data clustering: Theory and its application to image segmentation," *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, vol. 15, no. 11, pp. 1101–1113, 1993.

[3] Y. Weiss, "Segmentation using eigenvectors: a unifying view," in *Computer vision, 1999. The proceedings of the seventh IEEE international conference on*, vol. 2. IEEE, 1999, pp. 975–982.

[4] P. F. Felzenszwalb and D. P. Huttenlocher, "Efficient graph-based image segmentation," *International Journal of Computer Vision*, vol. 59, no. 2, pp. 167–181, 2004.

[5] T. H. Cormen, C. E. Leiserson, R. L. Rivest, C. Stein *et al.*, *Introduction to algorithms*. MIT press Cambridge, 2001, vol. 2.