Agglomerative Connectivity Constrained Clustering for Image Segmentation

Jia Li[*]

Abstract

We consider the problem of clustering under the constraint that data points in the same cluster are connected according to a pre-existed graph. This constraint can be efficiently addressed by an agglomerative clustering approach, which we exploit to construct a new fully automatic segmentation algorithm for color photographs. For image segmentation, if the pixel grid with eight neighbor connectivity is imposed as the graph, each group of pixels generated by this clustering method is ensured to be a geometrically connected region in the image, a desirable trait for many subsequent operations. To achieve scalability for images with large sizes, the segmentation algorithm combines the top-down k-means clustering with the bottom-up agglomerative clustering method. We also find that it is advantageous to conduct clustering at multiple stages through which the similarity measure is adjusted. Experimental results with comparison to other widely used and state-of-the-art segmentation methods show that the new algorithm achieves higher accuracy at much faster speed. A software package is provided for public access.

Keywords: connectivity constrained clustering, agglomerative clustering, image segmentation, k-means

1 Introduction

Research efforts have been devoted to developing clustering techniques for several decades. Motivated by problems in different domains, researchers in multiple disciplines including statistics, computer science, and electrical engineering, have eventually come to face the common abstracted problem of clustering data, and have fairly independently invented various methods, which share principles but differ in treatment. We point readers to [14, 6, 10, 15] as excellent references to a vast body of literature on clustering. With the explosion

[*]Jia Li is an Associate Professor in the Department of Statistics at the Pennsylvania State University. Email: jiali@stat.psu.edu

of data in the current era of information technology as well as the much increased complexity of data, new challenges for clustering continue to emerge in a wide range of applications. Despite the long history of studies on clustering, the topic has sustained interests from researchers in traditional disciplines, and has attracted interests from those in nascent fields, e.g., computational biology.

1.1 Main Approaches to Clustering

Clustering methods can be categorized in several ways. We can contrast them as optimization-based versus model-based. In the former approaches, usually an optimization criterion is proposed to reflect the belief about what constitutes a good clustering result. Then, numerical methods are derived to solve the optimization problem. In model-based approaches, each cluster is assumed to follow a certain parameterized distribution [19, 2, 8] or a mixture of such distributions [16]. The clustering problem is then converted into statistical model estimation. When the mixture model is involved, the EM algorithm [5] is most commonly used to estimate the parameters. In terms of the way clusters are formed, methods can be divided into top-down versus bottom-up, with a hybrid proposed by [4]. In the bottom-up, also called agglomerative, approaches, existing clusters are merged recursively, one pair at a time, starting from clusters containing a single object. Only pairwise distances between objects are needed. There is usually a distance updating scheme that generates distances between clusters when a new cluster is formed by combining an existing pair. Examples of the distance updating scheme include single, complete, average linkage, and Ward's clustering, which greedily minimizes the total variation.

1.2 Clustering with Connectivity Constraints

In some applications, the formed clusters are required to meet certain conditions in addition to the generic requirement that entities within the same cluster are as similar as possible according to a given measure. For instance, in image analysis, clustering is often applied to pixel-level feature vectors such as colors to separate pixels in an image into groups [15, 17]. Segmentation of the image is achieved by taking every group of pixels as one segmented region. The purpose of segmentation often is to build a rough correspondence between segmented regions and physical objects captured in the image. This decomposition of an image into objects facilitates many subsequent analysis, for instance, retrieval and the prediction of semantics [28, 18].

A usual clustering procedure, however, only ensures that pixels in the same cluster are close in the feature vector space. In the spatial plane, the pixels may not be connected geometrically. It is heuristically irrational that several fragmented patches form one object. One would naturally prefer to have each segmented region be spatially connected. There is no obvious way to guarantee this. An ad-hoc method often used is to include the spatial coordinates of pixels as features in clustering, with several issues lingering though. First, it is unclear what is the best practice to scale the coordinates with respect to other features such as color. Second, even when the coordinates are included as features, there is no guarantee that pixels in each final cluster are all connected, although without doubt, they tend to be spatially close.

We thus pose the following problem. Suppose the entities to be clustered are nodes on a graph. Independently from the graph, a measure of similarity is given to any pair of entities. We want to cluster the entities based on the similarity and in the mean time ensure that entities in the same cluster are connected according to the graph. We call this problem *connectivity constrained clustering*, and study the application to fully automatic segmentation of color images. Due to the great challenge of image segmentation, a simple plug-in of the connectivity constrained clustering method is insufficient. Instead, we have to treat the clustering method as a design element for creating the overall segmentation system. Specifically, it is combined with other clustering methods, and is conducted through several stages with an adaptive similarity measure. The integrated algorithm achieves both fast speed and high accuracy.

We point out that the graph in our setup serves as a connectivity constraint and is irrelevant to the similarity measure, a role completely different from that of graph-based clustering methods [29, 25], in which the weighted edges of the graph record values of similarity. Readers can also regard that there are two graphs in the present setting: one given by the similarity between objects that any clustering algorithm employs and is computed from the data and one imposed externally that takes into consideration of neighborhood information.

A related topic to our proposed problem is clustering with side information [27, 30]. There, instead of a graph constraint, pre-given conditions on the similarity of some pairs of points are available. For instance, some pairs are labeled “similar” and some “dissimilar”. The emphasis is on learning a distance metric that is consistent as much as possible with those conditions. The graph constraint we consider here differs intrinsically from a set of pairwise relationships because connectivity can occur through multiple steps, and hence the satisfaction of the constraint depends on all the entities put in the same cluster. Moreover, as a constraint, the connectivity of each cluster is mandatory, while side information is more for consulting and can be approximated. We should also note that requiring each segmented region be spatially connected may not be proper for images in some special domain. For instance, when segmenting microarray images into foreground and background, the foreground can be disconnected spots in a noisy image [22, 23].

1.3 Related Work

Connectivity constrained clustering, also referred to as contiguity-constrained clustering, has been investigated and used in several application areas (see survey [21]). For maximum split clustering with connectivity constraints expressed by a tree graph, Hansen et al.[12] provides an exact algorithm with quadratic complexity in the size of the data set. In the same paper, theoretical results are also given about the computational complexity of an exact algorithm for constraints expressed by a general graph. However, to the best of our knowledge, the agglomerative connectivity constrained clustering in its general form and allowing different linkage schemes has not been used for fully automatic segmentation of color photographs.

Recently, the agglomerative connectivity constrained clustering with average linkage was extended in [20] by defining a dissimilarity between clusters that depends only on dissimilarity between connected data points in the clusters. The paper focuses on the clustering algorithm itself and treats superficially the application to image segmentation. Only one small area of a single MRI grayscale image is shown as an example; and there is no numerical evaluation or comparison with other methods. The dissimilarity between pixels is simply the difference in grayscale, which is known to be too weak for color pictures. Moreover, the dissimilarity between two regions depends only on pixels along the boundary. For general-purpose photographs, this definition of dissimilarity is highly prone to blurred edges that often exist in a photo because of the limited depth of view as well as limitation in resolution. As is shown by some state-of-the-art algorithms [25, 1] and the current work, to achieve good segmentation, much more sophisticated measures are needed for dissimilarity between regions.

In [26], a graph partitioning based clustering method with connectivity constrains is applied to semi-automatic segmentation of images into foreground and background. A user is required to specify which pixels have to be connected with the main object, aka, foreground (assuming a first step segmentation has acquired most of the main object). In addition, the connectivity constraints tackled in that work, different from ours, are meaningful only under the setup of manually assisted segmentation. Although the kind of connectivity constraints we consider here is brought up in [26], it is said to be too difficult for developing heuristic algorithms and is not further studied.

Automatic image segmentation has been extensively researched on (see [13, 3, 9, 24]). We mention in particular the work of Shi and Malik [25] and that of Arbeláez et al. [1], both of which use spectral graph partitioning. The former algorithm is one of the most widely used methods. The latter, developed recently by the same research group, is targeted to address limitations of the former and is shown to achieve more accurate segmentation. We thus compare our method with these two methods because they are popular and represent the state-of-the-art.

The rest of the paper is organized as follows. In Section 2, the connectivity constrained clustering problem is formulated and an agglomerative clustering algorithm is proposed. Section 3 is on the application of the clustering algorithm to image segmentation. Due to the unique challenges of image segmentation, the application is not straightforward. The clustering algorithm serves as a component of a more comprehensive design. In Section 4, we illustrate the steps of the segmentation algorithm and compare results of the new algorithm with those of existing methods. We conclude and discuss future work in Section 5. In the appendices, a key property of the clustering algorithm is proved and details on distance updating are provided.

2 Agglomerative Connectivity Constrained Clustering

Before formulating connectivity constrained clustering, let us provide several graph-related definitions.

1. A *graph* $\mathcal{G}$ is a collection of nodes $\{V_1, ..., V_n\}$ and edges connecting pairs of nodes (V_i, V_j).

2. Two nodes V_i and V_j are called *neighbors* if the edge (V_i, V_j) exists in $\mathcal{G}$.

3. A *path* between V_i and V_j is a sequence of edges (V_i, V_{t_1}), (V_{t_1}, V_{t_2}), (V_{t_2}, V_{t_3}), ..., $(V_{t_{k-1}}, V_{t_k})$, (V_{t_k}, V_j) which are all included in $\mathcal{G}$.

4. A graph $\mathcal{G}$ is *connected* if a path exists in $\mathcal{G}$ between all V_i and V_j, $i \neq j$, where V_i and V_j are nodes in $\mathcal{G}$.

5. A *subgraph* $\mathcal{G}'$ of $\mathcal{G}$ contains a subset of the nodes $\{V_{i_1}, V_{i_2}, ..., V_{i_k}\}$ and inherits any edge $(V_{i_j}, V_{i_{j'}})$, $1 \leq j, j' \leq k$, that exists in $\mathcal{G}$. The subgraph of $\mathcal{G}$ generated from a subset of the nodes $\mathcal{C} = \{V_{i_1}, V_{i_2}, ..., V_{i_k}\}$ is denoted by $\mathcal{G}(\mathcal{C})$.

6. If (V_i, V_j) exists in $\mathcal{G}$, we write $(V_i, V_j) \in \mathcal{G}$.

The clustering problem in consideration is as follows. A set of objects $\{x_1, x_2, ..., x_n\}$ are to be grouped into multiple categories. A symmetric pairwise distance $D(x_i, x_j)$ is given for every $i \neq j$ (by default $D(x_i, x_i) = 0$). A graph $\mathcal{G}$ with $\{x_1, x_2, ..., x_n\}$ as nodes is specified. It is assumed that $\mathcal{G}$ is connected. Following the generally accepted heuristics about clustering, we aim at dividing the x_i's into groups such that the within group distances between objects are small and the between group distances are large. Moreover, we require that the subgraph generated from x_i's in any cluster is connected. Suppose K clusters are formed, each denoted by $\mathcal{C}_k$, $k = 1, 2, ..., K$. Then $\mathcal{G}(\mathcal{C}_k)$ is connected for any k.

The agglomerative clustering approach attempts to achieve small within clustering distances by recursively merging two existing clusters that yield minimum between-cluster distance. The clustering procedure can be visualized by a tree structure called *dendrogram*. We propose a new way of merging clusters which ensures connectivity of any cluster at any level of the clustering hierarchy. In summary, the algorithm differs from the usual agglomerative clustering without constraints in two ways. First, a graph recording the connectivity between clusters is created recursively after each step of merging. At any particular iteration, the current clusters are treated as nodes in the graph. We denote the graph at iteration p by $\mathcal{G}^{(p)}$. Second, two clusters can be merged only if they are connected according to the graph at the current iteration. Detailed description of the algorithm is as follows.

1. Let the starting clusters $\mathcal{C}_k^{(0)} = \{x_k\}$, $k = 1, 2, ..., n$, and the number of clusters $K = n$. Denote the distance between clusters $\bar{D}(\mathcal{C}_i^{(0)}, \mathcal{C}_j^{(0)}) = D(x_i, x_j)$. Here, notation $\bar{D}$ is used to distinguish the distance between groups of points from the distance between individual points. Construct graph $\mathcal{G}^{(0)}$ on vertices $\{\mathcal{C}_k^{(0)}, k = 1, 2, ..., n\}$. Moreover, $(\mathcal{C}_i^{(0)}, \mathcal{C}_j^{(0)}) \in \mathcal{G}^{(0)}$ if $(x_i, x_j) \in \mathcal{G}$.

2. At iteration p, $p > 0$:

(a) Find the pair $i^* < j^*$ such that $(\mathcal{C}_{i^*}^{(p-1)}, \mathcal{C}_{j^*}^{(p-1)}) \in \mathcal{G}^{(p-1)}$ and

$$\bar{D}(\mathcal{C}_{i^*}^{(p-1)}, \mathcal{C}_{j^*}^{(p-1)}) = \min_{(\mathcal{C}_i^{(p-1)}, \mathcal{C}_j^{(p-1)}) \in \mathcal{G}^{(p-1)}} \bar{D}(\mathcal{C}_i^{(p-1)}, \mathcal{C}_j^{(p-1)}) \, .$$

Note that because $\mathcal{G}$ is connected, $\mathcal{G}$ cannot be divided into mutually disconnected subgraphs. Hence, there always exists i and j such that $(\mathcal{C}_i^{(p-1)}, \mathcal{C}_j^{(p-1)}) \in \mathcal{G}^{(p-1)}$.

(b) Merge $\mathcal{C}_{i^*}^{(p-1)}$ and $\mathcal{C}_{j^*}^{(p-1)}$ into a new cluster: $\mathcal{C}_{i^*}^{(p)} = \mathcal{C}_{i^*}^{(p-1)} \cup \mathcal{C}_{j^*}^{(p-1)}$.

(c) Retain the other clusters: $\mathcal{C}_i^{(p)} = \mathcal{C}_i^{(p-1)}$, for $i < j^*$, $i \neq i^*$, and $\mathcal{C}_i^{(p)} = \mathcal{C}_{i+1}^{(p-1)}$, for $j^* \leq i \leq K-1$. For brevity of notation, let us use $i \uparrow$ to denote the index of the cluster at the previous iteration $p-1$ which becomes cluster $\mathcal{C}_i^{(p)}$. Also, define $i^* \uparrow = i^*$. For $i < i^*$ or $i^* < i < j^*$, $i \uparrow = i$. For $j^* \leq i \leq K-1$, $i \uparrow = i+1$.

(d) Update the graph from $\mathcal{G}^{(p-1)}$ to $\mathcal{G}^{(p)}$. $\mathcal{G}^{(p)}$ is constructed on vertices $\{\mathcal{C}_k^{(p)}, k = 1, 2, ..., K-1\}$. For any $i, j \neq i^*$, $(\mathcal{C}_i^{(p)}, \mathcal{C}_j^{(p)}) \in \mathcal{G}^{(p)}$ if $(\mathcal{C}_{i\uparrow}^{(p-1)}, \mathcal{C}_{j\uparrow}^{(p-1)}) \in \mathcal{G}^{(p-1)}$. If either i or $j = i^*$, and without loss of generality, assume $i = i^*$, then $(\mathcal{C}_{i^*}^{(p)}, \mathcal{C}_j^{(p)}) \in \mathcal{G}^{(p)}$ if $(\mathcal{C}_{i^*}^{(p-1)}, \mathcal{C}_{j\uparrow}^{(p-1)}) \in \mathcal{G}^{(p-1)}$ or $(\mathcal{C}_{j^*}^{(p-1)}, \mathcal{C}_{j\uparrow}^{(p-1)}) \in \mathcal{G}^{(p-1)}$. An illustration of the creation of the new graph will be provided by a figure momentarily.

(e) Update the between-cluster distances. For any $i \neq i^*$ and $j \neq j^*$, $\bar{D}(\mathcal{C}_i^{(p)}, \mathcal{C}_j^{(p)})$ is given because the two clusters are inherited from the previous iteration $p-1$. Compute the distances $\bar{D}(\mathcal{C}_{i^*}^{(p)}, \mathcal{C}_j^{(p)})$, for all $1 \leq j \leq K$, $j \neq i^*$. Details on distance updating will be described shortly.

(f) Reduce the number of clusters: $K - 1 \rightarrow K$.

In Appendix A, we prove that clusters generated by the above algorithm are guaranteed to be connected according to $\mathcal{G}$. The update of the graph $\mathcal{G}^{(p)}$ from $\mathcal{G}^{(p-1)}$, as described in Step 2(d), is illustrated in Figure 1. When two clusters are merged, in the new graph, a third cluster is connected to the merged bigger cluster if it is connected with either of the component clusters in the initial graph. The edge between any pair of unchanged clusters is copied from the initial graph if it exists. The between-cluster distances after merging can be updated in several ways. Some example formulas are provided in Appendix B. We call the algorithm proposed above the *agglomerative connectivity constrained clustering (A3C)*.

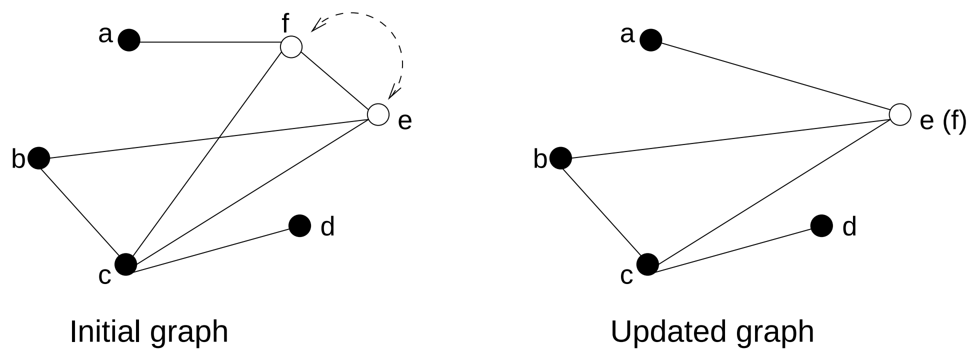

Figure 1: Update of the graph after clusters are merged

3 Image Segmentation

A color image is digitally represented by an array of vectors $v_{i,j}$, $1 \leq i \leq m_r$, $1 \leq j \leq m_c$, where m_r and m_c are the numbers of rows and columns in the image. The vector $v_{i,j} \in \mathcal{R}^3$ contains the values for the Red, Green, and Blue components. In image processing, we often transform the color vector to a color space different than RGB, for instance, LUV, where L is the luminance of the color and U and V are the chromatic components [7]. It is often believed that the LUV space corresponds more directly with human visual perception.

As explained in Section 1, it is preferable to ensure that segmented regions in an image are spatially connected. The A3C algorithm achieves this property by using a graph constructed based on spatial adjacency of pixels. We introduce the following terminologies.

- *8-connected*: Pixels with coordinates (i_1, j_1) and (i_2, j_2) are said to be 8-connected if $|i_1 - i_2| \leq 1$ and $|j_1 - j_2| \leq 1$.

- *Path*: a sequence of pixels $\{(i_1, j_1), (i_2, j_2), ..., (i_n, j_n)\}$ such that for any $1 \leq k < n$, (i_k, j_k) and (i_{k+1}, j_{k+1}) are 8-connected.

- *Region*: a subset of pixels from the image.

- *Connected region*: A region is said to be connected if for any two pixels (i, j) and (i', j') in the region, there exists a path linking the two pixels.

- *Segment*: If an image is divided into several connected regions, each region is called a *segment*.

- *Patch*: If segmentation is performed by agglomerative clustering, to distinguish from the segments obtained at the end, we call a segment acquired in the intermediate steps a *patch*.
- *Cluster*: a group of pixels grouped together by a clustering algorithm. Because A3C satisfies the graph connection constraint, clusters generated by this algorithm are connected regions, and hence may be used interchangeably with patches or segments. However, a cluster generated by other algorithms, e.g., k-means, may not be a connected region.

We do not obtain segmentation by applying A3C directly to the pixels of images. Instead, a multi-stage clustering approach is developed so that both computational efficiency and good segmentation are achieved. First, we apply k-means to over segment an image into small patches homogeneous in color. Because k-means does not guarantee clusters are connected regions, a connected component operation is applied to each cluster of k-means to extract usually multiple patches. In this beginning step, the goal is to produce a sufficiently fine division of the images. K-means is computationally more efficient than the agglomerative clustering approach because there is no need to compute pairwise distances, the complexity of which is in the order of the square of the number of pixels. Next, A3C is applied in two stages using different definitions of pairwise distance. At the first stage, the emphasis is on merging visually similar patches. At the second stage, more sophisticated types of pairwise distances and mechanisms for merging are used in order to achieve a good overall segmentation. The computational cost is nevertheless low because the second stage starts with significantly fewer patches.

As will be seen from the detailed description below, the segmentation algorithm tries to combine various visual cues possibly exploited in human visual perception. Hence, there may seem to be many tuning parameters. However, for all the images we have shown and tested in the experiment section, those parameters are fixed. To apply this algorithm to general-purpose color photographs, the parameters do not need to be tuned. Hence, the segmentation algorithm (software package provided) should be fairly straightforward to use in practice.

3.1 Distances

To start the agglomerative clustering, we need to compute pairwise distances between image patches. When Ward's clustering is used to update the distance after merging two clusters, the initial pairwise distance has to be defined in a specific way so that the merging results in minimum increase of total variation. To update distances by other schemes, there is no special requirement on the definition of the distance. Next, we describe several types of distances between patches based on different pictorial characteristics. How these distances are combined in the clustering algorithm will be explained in Section 3.3 and 3.4.

- **Color**:

  Two versions of color-based distances are defined, one for Ward's clustering, and the other for any other linkage scheme. We denote the distance used by Ward's clustering by D_{cw} and that by other linkage schemes by D_c. Given two patches i and j, let the average LUV color vectors for pixels in the patches be $y_i, y_j \in \mathcal{R}^3$. Let $||\cdot||$ denote the Euclidean distance, and n_i, n_j be the number of pixels in patch i, j respectively. We also refer to n_i as the area or size of the patch. Then, we define

  $$D_{cw}(i,j) = ||y_i - y_j||^2 \cdot \frac{n_i n_j}{n_i + n_j} \ ,$$

  and

  $$D_c(i,j) = ||y_i - y_j||/\sqrt{3} \ .$$

- **Location**: For each patch i, we compute the average coordinates of the pixels in the patch. Let the average horizontal and vertical coordinates of patch i be $z_i \in \mathcal{R}^2$. We define D_l simply as the Euclidean distance between the average coordinates: $D_l(i,j) = ||z_i - z_j||$, and D_{lw} for Ward's clustering: $D_{lw}(i,j) = ||z_i - z_j||^2 \cdot \frac{n_i n_j}{n_i + n_j}$.

- **Edge**: We define a distance to reflect the extent of separation by edges at the boundary between two patches. First, Sobel filter [11] is used to compute the gradient at every pixel in the image using the LUV color components individually. We apply the following two Sobel filters at each pixel to obtain the horizontal and vertical derivatives (in digitized version) g_x and g_y.

  $$\begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix} , \quad \begin{bmatrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix}$$

The gradient is calculated by $\sqrt{g_x^2 + g_y^2}$. The combined gradient is the average of the three gradient values based on LUV components respectively. The rationale to combine gradients in all the LUV components is to increase sensitivity to color variation. One may still perceive an edge if the color changes abruptly, but not the luminance component. For two neighboring patches i and j, we compute the average combined gradient at the boundary pixels located between the two patches. These average gradient values are scaled to the range of $[0, 1]$ and are used as the edge-based distance between patches, denoted by D_e.

- **Balanced partition measure (BPM)**: This measure is designed to encourage segmentation into regions with similar sizes in one image. It is more concerned with the final appearance of segmentation than the closeness of two patches to be merged, and is intended to be used under the scenario that all patches are already substantially different. For instance, in the second stage of applying A3C, when patches resulting from the first stage have already achieved certain level of pairwise dissimilarity, the emphasis is shifted somewhat from merging similar patches to obtaining a visually appealing segmentation result. Motivated by the observation that human eyes tend to recognize large patches of an image at a quick glance, we include BPM as part of the distance between regions. Because the combined distance also incorporates D_c and D_e, the effect of BPM is negligible if the edge and color differences are strong enough to mark out a region.

  Let the proportional sizes of a set of clusters be $\{p_1, p_2, ..., p_k\}$. The p_i's form a discrete distribution with $\sum p_i = 1$. Suppose clusters i and j, $i < j$, are merged and the resulting proportions of the $k-1$ clusters are: $(\{p_1, p_2, ..., p_k\} - \{p_i, p_j\}) \cup \{p_i + p_j\}$. The BPM of clusters i and j is defined as the L2 norm between the distribution given by the new $k-1$ proportions and a uniform distribution. Specifically, if we denote the $k-1$ proportions in $\mathcal{P}(i,j) = (\{p_1, p_2, ..., p_k\} - \{p_i, p_j\}) \cup \{p_i + p_j\}$ by p'_l, $l = 1, ..., k-1$:

$$D_{bpm}(i,j) = \sqrt{\sum_{p'_l \in \mathcal{P}(i,j)} \left(p'_l - \frac{1}{k-1}\right)^2} \, .$$

- **Jaggedness measure**: The jaggedness of the boundary between two neighboring segments is measured and used as a distance. We adopt the heuristic that smoother or more regular boundaries are more likely to be true object boundaries, while jagged boundaries are more likely caused by lighting and surface texture. First, we define a boundary pixel as *jagged* if in its 8-connected neighborhood, which contains a total of 9 pixels including itself, 5 or more pixels are not in the same segment as this pixel. The rationale is as follows. Consider a pixel at the boundary of two segments. If the two segments are separated by a smooth curve, at the refined pixel scale, the smooth curve either passes through the pixel as a straight line or as a line bended at the pixel. Since image pixels locate on a discretized grid, there are only four directions for a line in a 3×3 block to take: horizontal, vertical, 45^o, -45^o. For a bended line with the turning point located at the pixel of interest, there are only three bending angles: 45^o, 90^o, and 135^o. We consider bending angle smaller than 135^o a sharp corner, which a smooth curve should not bear.

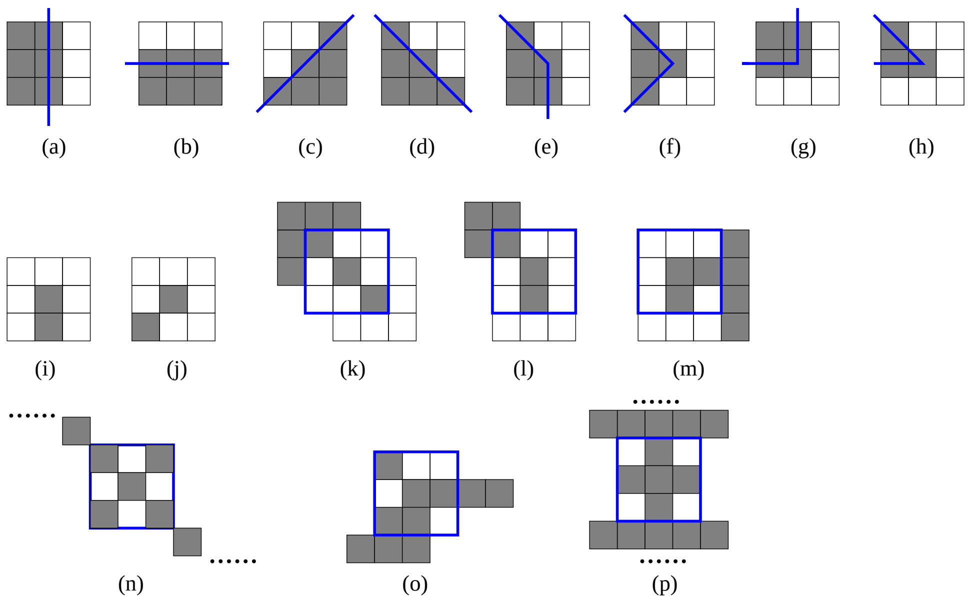

Figure 2: Different separation patterns for a boundary point between two segments indicated by filled and blank squares respectively.

Figure 2 (a)-(d) show the separation of two segments by straight line boundaries in four orientations; Figure 2 (e) shows the one representative case of 135^o bending boundary lines (the other cases can be obtained by reflection); and Figure 2 (f)-(h) show the 45^o and 90^o bending boundary lines. The pixel in consideration is always shown at the center of the 3×3 block, therefore referred to as the center pixel. We see that when the bending line has a sharp angle, the number of pixels in the same segment as the center pixel is smaller than 5, while for cases (a)-(e), that number is at least 5. If the number of pixels in the same segment as the center pixel is only 2, with representative cases shown in Figure 2 (i), (j), the center pixel is an end point of a line protruding into the other segment, an indication of irregular separation between the two segments. In the cases shown from (a)-(j), within the 3×3 blocks, pixels are connected for the neighboring segment shown by the blank squares. In some rare cases, the neighboring segment is disconnected (according to 4-connection) in the 3×3 block, as shown by Figure 2 (k)-(p). To decide whether the center pixel locates on a smooth boundary requires examination of pixels outside the 3×3 block. In (k)-(m), examples are shown that the center pixel resides on a dangling boundary thrusting into the neighboring segment. On the other hand, in (n)-(p), the center pixel is on a thin smooth boundary. To keep the calculation simple, we still use the "≥ 5" criterion for these cases. If the pixels represented by the filled squares in examples (n)-(p) are actually at the end of a dangling boundary, the jaggedness of the boundary can be captured by the non-center pixels.

To summarize, when the two segments are separated by smooth curves, zooming in the 3×3 neighborhood of one pixel on the boundary curve, at least 5 neighboring pixels are in the same segment as the former. It is subjective to quantify the level of "jaggedness". We hereby introduce a simple test for declaring a boundary pixel as jagged: 5 or more pixels in its 3×3 neighborhood are in a different segment. To define a measure for the level of jaggedness for two segments denoted by $\mathcal{C}_i$ and $\mathcal{C}_j$, we find the number of boundary points, n_i (n_j), in $\mathcal{C}_i$ ($\mathcal{C}_j$) that are adjacent to $\mathcal{C}_j$ ($\mathcal{C}_i$). Among these $n_i + n_j$ points located at the boundary, we count the number of jagged points, n_{jag}, according to the test described. The measure of jaggedness is

$$D_{jag}(i,j) = 1 - \min\left(\frac{n_{jag}}{2n_1} + \frac{n_{jag}}{2n_2}, 1\right) ,$$

which is also scaled so that the range of values is one.

3.2 Generate the Initial Pixel Patches

The computational complexity of A3C, as of any other pairwise distance based clustering method, is quadratic in the number of objects to be clustered. Because image sizes vary enormously and it is now common for a digital photo to contain millions of pixels, we perform a preparation step to group pixels into relatively homogeneous small patches before applying the A3C algorithm to these patches. Comparing with applying agglomerative clustering directly to the pixels, this strategy reduces the computation by several orders of magnitude. Moreover, it makes clustering less sensitive to the size of an image. A high resolution image with much more pixels than a low resolution image usually does not yield significantly more patches to start with.

The basic idea for generating the patches is to apply k-means clustering to the color vectors of the pixels. We gradually increase the number of clusters in k-means clustering until the resulting total within-cluster distance is below a given threshold. By using a small threshold, the color vectors in the same cluster are forced to be very close to the cluster average. The pixels in the same cluster are not guaranteed to be connected in the image. We apply the connected component operation to find all the connected components for every cluster. Each connected component of pixels in the same cluster becomes a patch.

It is observed that the sensitivity of human vision to color variation depends on the size of the image area looked upon. The same amount of change in color over a larger area is more obvious than over a smaller one due to the blurring of the eyes. When an area is small, this blurring effect makes it difficulty for human eyes to discern variation. Motivated by this observation, we design an iterative procedure to create the patches. Instead of applying k-means once using a single threshold, we employ a decreasing sequence of thresholds $\theta_1 > \theta_2 > \cdots > \theta_l$. A patch acquired at a large θ_i tends to have more color variation. But if it is small enough, it will not be further divided. In the initial iteration, all the pixels in the image are considered. Let the collection of pixels at the beginning be $\mathcal{I}_1$. For $i = 1, ..., l$, repeat the following.

1. Apply k-means with threshold θ_i to color vectors of pixels in $\mathcal{I}_i$. That is, the number of clusters in k-means is the smallest integer such that the resulting average within cluster distance is below θ_i.

2. Find connected components in the image based on the clustering labels generated by k-means. Every connected component becomes a patch.

3. At iteration $i < l$, for every pixel in $\mathcal{I}_i$, if it locates in a patch with size above threshold ζ, it is put in $\mathcal{I}_{i+1}$, otherwise excluded. The excluded patches are recorded in the list of final patches. At iteration $i = l$, all the patches are put in the list.

After the connected components are found, some patches will be of very small sizes and appear like pepper and salt noise in the segmented image. We remove these noisy patches by an iterative merging procedure. Let the threshold for the smallest allowable patch size be δ. We sort the patches in ascending order according to their sizes. Let the sorted patches be $\mathcal{C}^{(0)}_{(1)}$, ..., $\mathcal{C}^{(0)}_{(k_0)}$ with sizes $S(\mathcal{C}^{(0)}_{(1)}) \leq S(\mathcal{C}^{(0)}_{(2)}) \cdots \leq S(\mathcal{C}^{(0)}_{(k_0)})$. Suppose $S(\mathcal{C}^{(0)}_{(1)}) < \delta$. In the tth iteration, starting from $t = 1$, do the following.

1. Find all the patches that are neighbors to $\mathcal{C}^{(t-1)}_{(1)}$ and compute the color based distance D_c between $\mathcal{C}^{(t-1)}_{(1)}$ and each of its neighboring patch. Merge $\mathcal{C}^{(t-1)}_{(1)}$ with the patch yielding the smallest D_c.

2. Sort in ascending order the new set of patches. Denote the sorted new patches by $\mathcal{C}^{(t)}_{(1)}$, ..., $\mathcal{C}^{(t)}_{(k_t)}$. If $S(\mathcal{C}^{(t)}_{(1)}) < \delta$, let $t + 1 \rightarrow t$ and go to step 1. Otherwise, stop.

3.3 First-stage A3C

After the initial patches are obtained, we apply the A3C algorithm to merge patches that are connected with each other. Considering each patch as a node, the constraining graph is constructed based on the spatial adjacency of the patches. Specifically, any two patches are connected if there exist two pixels, one from each patch, that are 8-connected neighbors.

Since the initial clusters taken by the agglomerative clustering algorithm are patches rather than single pixels, the cluster sizes are the numbers of pixels in the patches. For patch i and j, their pairwise distance is a combination of color, edge, and location based distances:

$$D_1(i,j) = \sqrt{\lambda_1^2 D_{cw}(i,j) + \lambda_2^2 D_{lw}(i,j)} + \lambda_3 D_e(i,j) \ .$$

The three types of distances are updated separately. D_{cw} and D_{lw} are updated by the Ward's clustering scheme, while D_e is updated by average linkage. λ_i's are pre-chosen scaling factors adjusting the relative importance of different kinds of distances.

3.4 Second-stage A3C

The first-stage clustering aims at merging patches that are visually similar in a local sense— the choice of merging only depends on the pairwise distances. At the second stage, the pairwise similarity is not the dominant factor in merging because it is assumed that all the patches resulting from the first stage are sufficiently distinct. More emphasis is put on achieving a good overall segmentation. We incorporate the balanced partition measure D_{bpm} into the pairwise distance between patches. Note that $D_{bpm}(i,j)$ depends on the proportional sizes of all the clusters after merging, and is not determined by patch i and j alone. The distance used in the second stage merging is:

$$D_2(i,j) = \lambda_c D_c(i,j) + \lambda_{jag} D_{jag}(i,j) + \lambda_{bpm} D_{bpm}(i,j) + \lambda_e D_e(i,j) \ .$$

Moreover, we directly eliminate patches with too small sizes by merging them with sufficiently large neighboring patches. Suppose the second stage A3C starts with N_1 patches and targets to merge them into N_2 patches. We set two thresholds, $\epsilon_1 = \frac{m_r \cdot m_c}{N_1} \times 5\%$, $\epsilon_2 = \frac{m_r \cdot m_c}{N_2} \times 20\%$, where m_r and m_c are the number of pixels in a row or column of the image. We insert the following steps in the merging procedure of A3C to avoid generating very small patches.

1. For the initial N_1 patches, if any is of size smaller than ϵ_1, merge it with a neighboring patch. Similarly as in agglomerative clustering, the merging is performed recursively with pairwise distances updated after each step. If there are several small patches that need to be merged with neighboring patches, the one with the minimum distance to a neighbor is processed first. Repeat the merging until all the patches are of size above ϵ_1. Suppose N_1' patches are left.

2. Apply A3C to the N_1' patches. After each merging in A3C, check whether the total size of the largest N_2 patches is above $m_r m_c - \epsilon_2$. If so, only perform future merging between the small patches and those among the N_2 largest patches. Otherwise, perform the next step of A3C.

4 Experiments

The images we experimented with are all scaled to 256×384 or 384×256 pixels. To acquire the initial patches, the K-means clustering is applied 4 times with thresholds equally space between 600 and 3600. The number of initial patches created for an image varies widely depending on the amount of details in the image. For instance, for a group of 100 photos of closeup shots of roses, the average number of initial patches is 210, while for a group of 100 photos of harbor scenes, the average is 638. As aforementioned, some patches generated by k-means are very small and are absorbed into bigger patches via a noise removal step described in Section 3.2. In our experiments, we set the threshold for the size of a noisy patch to 16. The average number of noisy patches for the rose group is 663 while for the harbor group is 4410.

4.1 Illustration for Step by Step Segmentation

As explained in Section 3.2, by applying k-means with gradually decreasing thresholds and excluding sufficiently small patches formed along the way from further division, we can reduce the sensitivity to color variation in small areas. Figure 3 shows an example image and its two zoomed-in areas. The segmentation results obtained using our algorithm and one execution of k-means with threshold 600 (same as the smallest threshold used in our algorithm) are compared. As we can see, our algorithm generates better segmentation results, while one execution of k-means creates more noisy patches, many containing only a single pixel.

In Figure 4, the step by step segmentation results for an example image is shown. After the initial step, 447 patches are obtained. The first stage merging reduces the number of segments to 22. In the second stage, the segmentation results obtained with several given numbers of segments are shown.

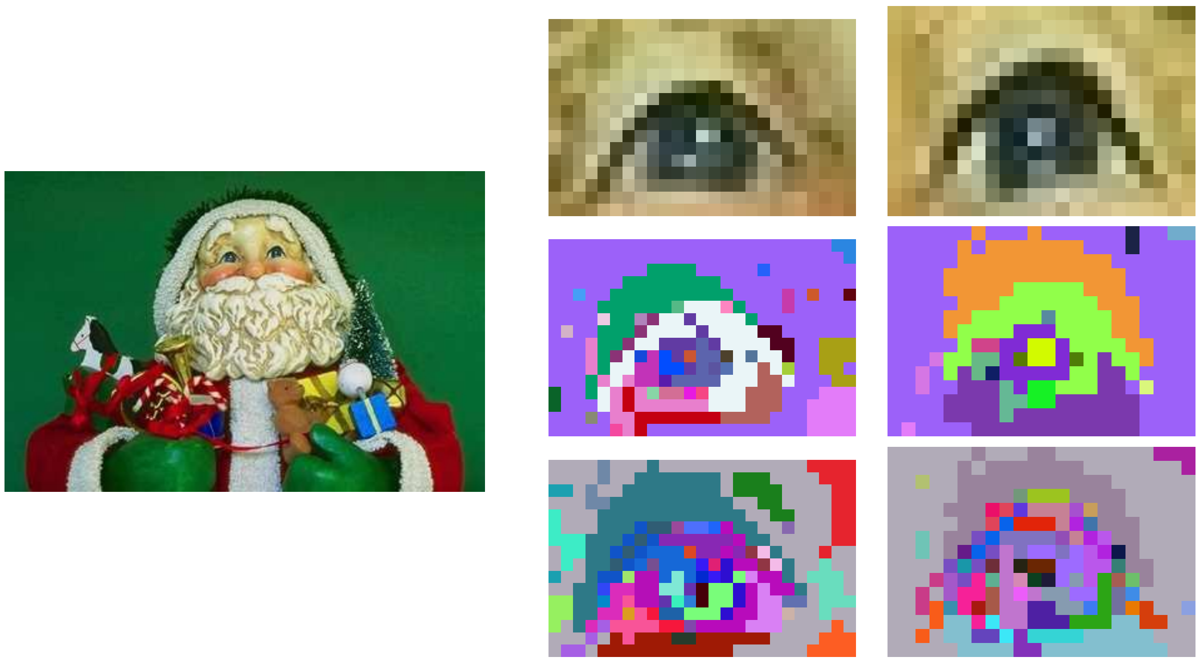

Figure 3: Compare segmentation by multi-iteration k-means and a single pass k-means. The two eyes of the Santa in the original images are zoomed in. The segmentation results by the two methods are compared. The middle row is based on multi-iteration k-means and the bottom row on single pass k-means.

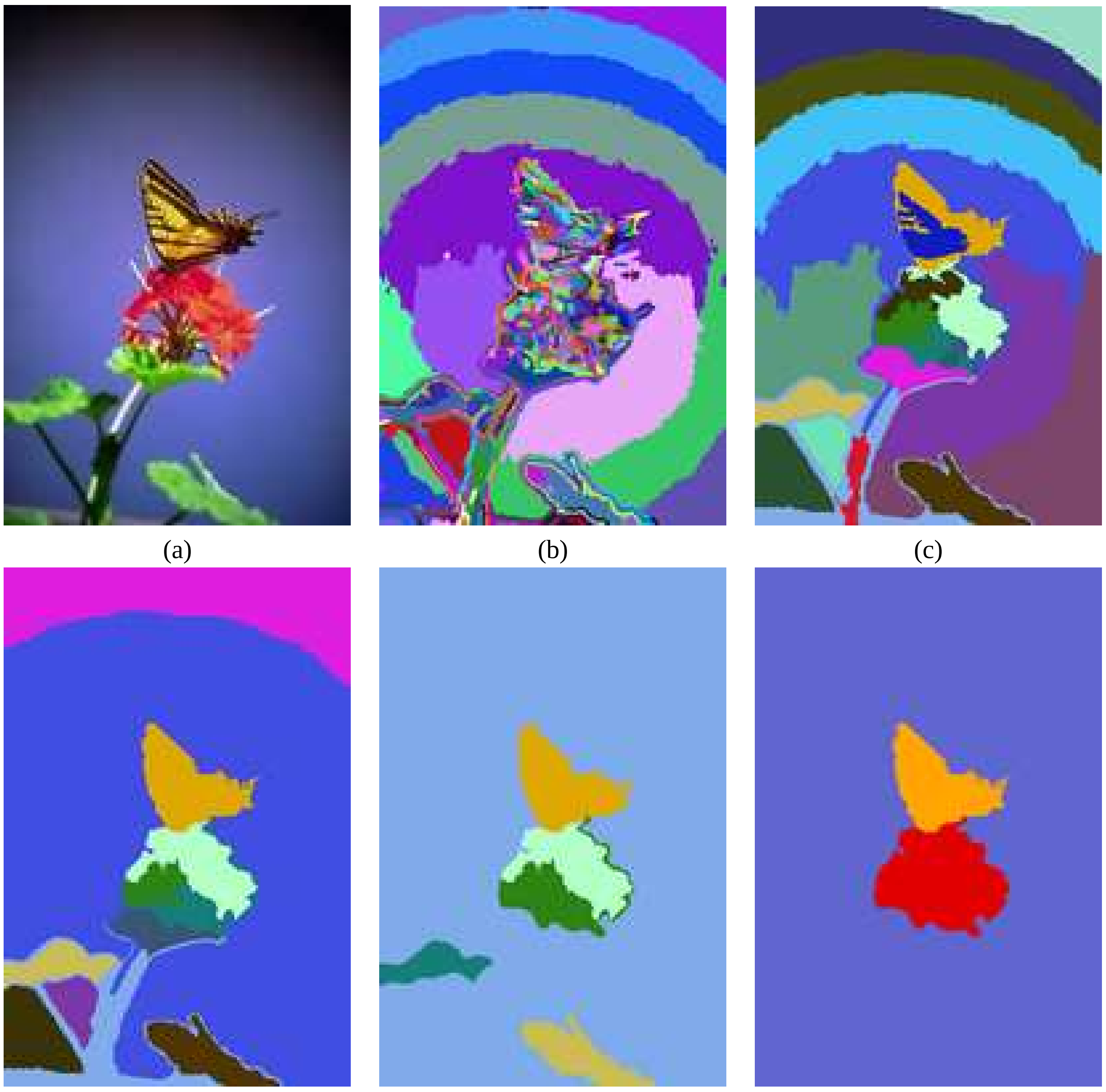
(a) (b) (c) (d) (e) (f)

Figure 4: Segmentation results for an example image. (a): original; (b): segmented patches via the initial step; (c): segmentation after the first stage merging; (d)-(f): results after the second stage merging with 12, 6, and 3 segments obtained respectively.

4.2 Comparison with K-means

We compare our segmentation algorithm to k-means followed by connected component extraction. For brevity, we refer to our algorithm as Multistage A3C (MS-A3C) because the segmentation process includes k-means and A3C conducted through several phases. Results for some example images are provided in Figure 5. We see that for most images, the segments obtained by MS-A3C correspond with objects clearly better than those by k-means. Moreover, k-means based segmentation has the issue that we cannot precisely control the number of connected components obtained at the end. To arrive at the results shown in Figure 5, for every image, we gradually increased the number of clusters in k-means; and at each number, recorded the final number of segments formed after extracting connected components and removing noise. Among these segmentation results, we selected the one with the number of segments closest to the targeted number chosen beforehand for that image.

Figure 6 shows the number of segments generated by extracting connected components based on the clustering result of k-means. The plot on the left shows the number of segments before removing small noisy patches, while that on the right shows the number after. We see that when k-means yields only 2 clusters, for three out of the five example images, the number of segments is above 150. The number of segments increases quickly when the number of clusters in k-means increases gradually from 2 to 10. If the noise removal procedure is applied, the number of segments is drastically reduced. However, the number of segments still grows much faster than the number of clusters. For instance, when the number of clusters in k-means increases one at a time from 2 to 5, the average number of segments obtained for the five images increases from 7.4, to 13, 17.6, and 29.8. In summary, the k-means approach lacks a mechanism for setting the number of segments, even at a moderate granularity. It is not rare to encounter an image for which the minimum number of segments producible by the k-means approach is still large.

4.3 Comparison with Graph Partitioning Methods

We compare our segmentation results with that given by the normalized cut image segmentation algorithm [25], an extremely popular segmentation tool used by researchers, which is referred to in short as Ncut hereafter. The Matlab package provided at http://www.cis.upenn.edu/$\sim$jshi/software/ is used. We also compare our algorithm with a recently developed algorithm, called OWT-UCM [1], the software

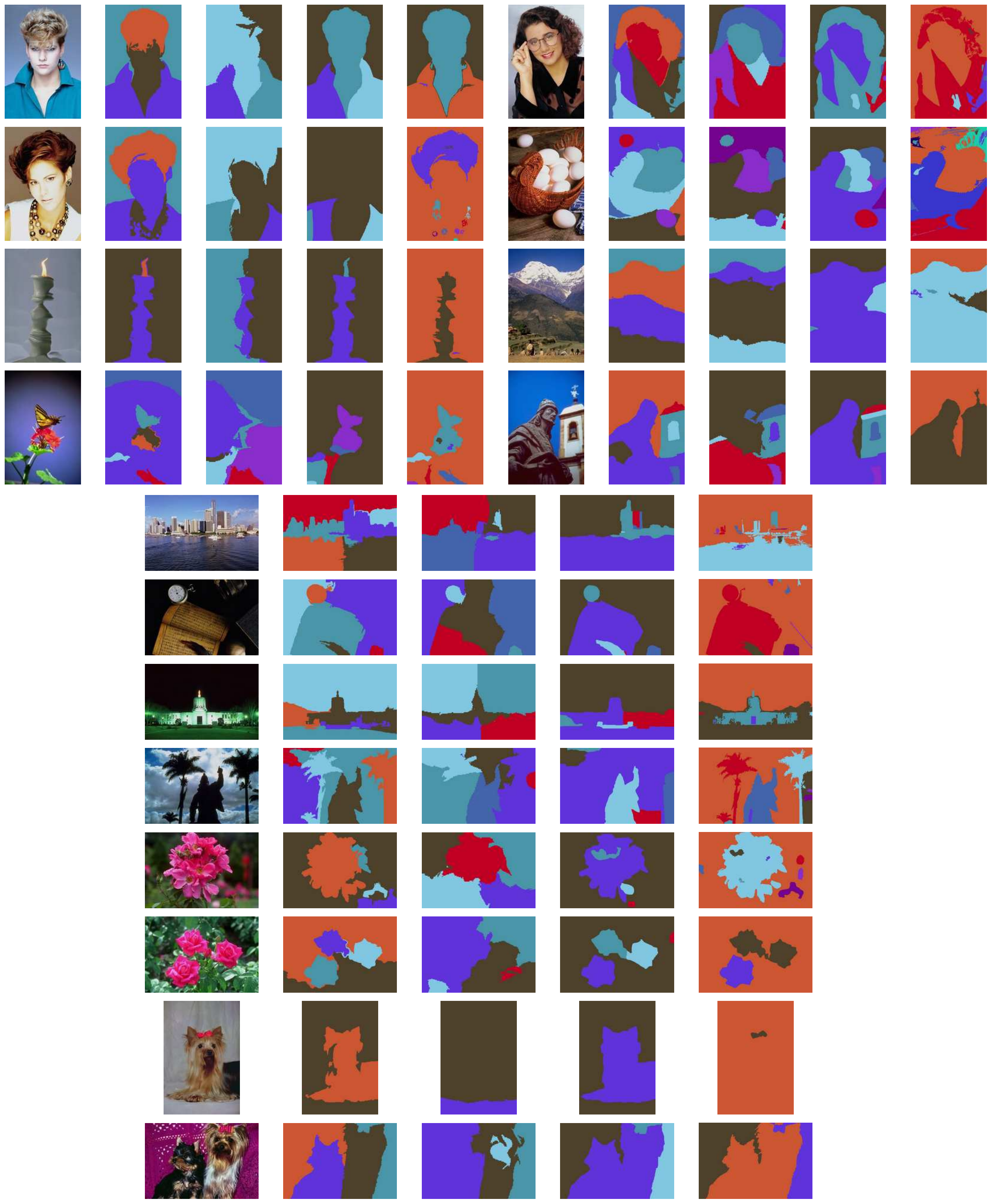

Figure 5: Segmentation result for example images. From left to right: column 1: original image; column 2: segmentation results by MS-A3C; column 3: Ncut; column 4: OWT-UCM; column 5: k-means.

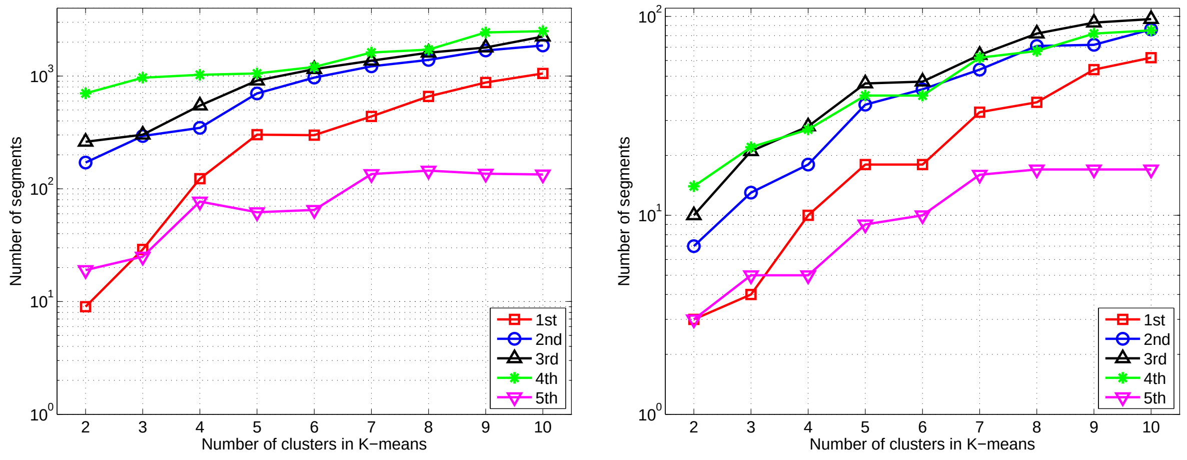

Figure 6: Number of segments generated based on k-means clustering using different numbers of clusters for five example images. Left: before removing noisy patches, the number ranges between 9 and 3000. Right: after removing noisy patches, the number ranges between 3 and 100.

provided at http://www.eecs.berkeley.edu/Research/Projects/CS/vision/grouping/. The gpb detector in the OWT-UCM software is used to obtain the contours. The OWT-UCM algorithm aims at improving the segmentation accuracy of Ncut, but not the computational speed. Although Ncut does not directly enforce connected regions, in practice, it is very rare for disconnected regions to appear. One possible reason is that location proximity between pixels is incorporated into the similarity measure. Hence, as we will see from the experimental results, the main advantages of our algorithm over Ncut are higher accuracy and faster computation. Although OWT-UCM also generates segments by first over-segmenting the image and then iteratively merging adjacent regions, the over-segmentation step itself is based on spectral graph partitioning and is thus computationally intensive. Instead of exploiting over-segmentation to achieve scalability as in our algorithm, over-segmentation in OWT-UCM is utilized to avoid breaking a large region incorrectly into smaller pieces, as is often done by Ncut. The region merging in OWT-UCM at the second step exploits a graph-based method, where the graph records the spatial adjacency of regions. This is different from our A3C algorithm with several types of linkage schemes.

The Ncut algorithm requires a pre-given number of segments. In our algorithm, a user can either specify the number of segments, or let the algorithm automatically choose the number of segments. If not specified, the number of segments is set to be one third of the number of patches created after the first stage merging. For many images we experimented with, the automatically chosen number of segments is reasonable, and we accepted those numbers. For a few dozens of images with close-up shots of roses and dogs, the automatically chosen numbers tend to be large because the close-up objects in the images have great variation within themselves. We thus manually selected the number of segments. For the rose images, we set the number of segments to 5. For the dog images, if the background is relatively homogeneous, we set the number of segments to the number of dogs in the picture plus 1 (to account for the background); otherwise, a few more segments are added. For the Ncut segmentation results, we used the same number of segments for every image as that in our algorithm. For the OWT-UCM method, the number of segments cannot be directly specified. It is determined by a threshold applied to the strength levels of contours. In our experiments, we exhaustively searched through all the possible values of the threshold and recorded the resulting number of segments. We chose the result with the number of segments matching that used by our algorithm. For more than 95% of the images we experimented with, the number of segments can be exactly matched. For the rest of the images, we let the number of segments obtained by OWT-UCM be larger than that of our algorithm by one.

Segmentation results for 20 example images are compared in Figure 5. We see that in general, the MS-A3C and OWT-UCM algorithms generate segments that follow object boundaries more faithfully than Ncut. Ncut focuses on achieving a good global separation, and often ignores the boundaries of objects. As a result, segments generated often contain fragments of multiple objects or objects and background. The boundaries by OWT-UCM tend to be smoother than those by MS-A3C. However, OWT-UCM appears to be more likely to combine several objects into one region and in the mean time to generate tiny regions of little importance in the images.

To numerically compare the three algorithms, we manually assessed the segmentation results on 220 images. It is well known that evaluation of segmentation is inevitably subjective because of the lack of ground truth. We adopt two strategies in our scoring scheme, which is significantly more objective than eyeballing the results. First, a score for segmentation quality is given to every segmented region rather than a whole image so that the evaluation process is broken into more manageable smaller tasks. Second, every segment is categorized into seven types with clear definitions. We believe that categorization can be conducted more decisively than assigning numerical scores according to subjective impression. Scores can then be given to each category to yield a numerical assessment for each image. One also has the freedom to vary the scores to better suit his own judgment of quality without repeating the manual evaluation process. Definitions for the seven types of segmented regions are described below.

1. Type a: The segment accurately corresponds to an object. The accuracy is up to the allowed resolution of the image. For instance, in Ncut, because the images are scaled down, the segmentation boundaries appear crude when scaled back to the original resolution. We still consider a segment accurate as long as the boundary roughly follows the object boundary.

2. Type b: The segment is a portion of an object, but close to the entity. For instance, a flower with a small portion (e.g., visually below 20%) of the petals missing.

3. Type c: The segment is a portion of an object, but not close to the entity.

4. Type d: The majority (visually above 80%) of the segment is one complete object.

5. Type e: The segment is a combination of several objects.

6. Type f: The segment contains a complete object and parts of other objects.

7. Type g: The segment contains parts from several different objects or background. This type is considered the worst scenario for a segment.

For the 220 images, the total number of segments in each type is listed Table 1. The table shows that for MS-A3C, type a and c dominate, with considerably more type a, while for Ncut, type a, c, and g dominate, with considerably more type c than g and a, and more type g than a. OWT-UCM performs much closer to MS-A3C than Ncut. The number of type g segments is nearly the same as that by MS-A3C. As with MS-A3C, type a and c also dominate. However, the number of type a segments by OWT-UCM is smaller than that by MS-A3C, while the number of type c by OWT-UCM is larger than by MS-A3C.

To summarize the results, we assign a score for each type. The higher the score is, the better the segment. One set of scores we use for type a-g are: $(a, 5)$, $(b, 4)$, $(c, 3)$, $(d, 4)$, $(e, 3)$, $(f, 2)$, $(g, 1)$. The average scores for the images under this score set (set 1) are shown in Table 1. If we simplify the scores and assign 5 to a, 1 to g, and 3 to every other types (set 2), the average scores vary slightly, as shown also in Table 1. MS-A3C achieves on average one point higher than Ncut under both sets of scores, and slightly higher scores than

OWT-UCM. The histograms for the scores (under score set 1) of the 220 images using the three algorithms are compared in Figure 7.

Type	a	b	c	d	e	f	g	Ave. (score set 1)	Ave. (score set 2)
MS-A3C	487	99	376	71	31	47	33	3.91	3.82
Ncut	209	42	472	46	26	81	267	2.84	2.85
OWT-UCM	407	57	447	51	84	70	35	3.71	3.69

Table 1: Segmentation results for MS-A3C, Ncut, OWT-UCM

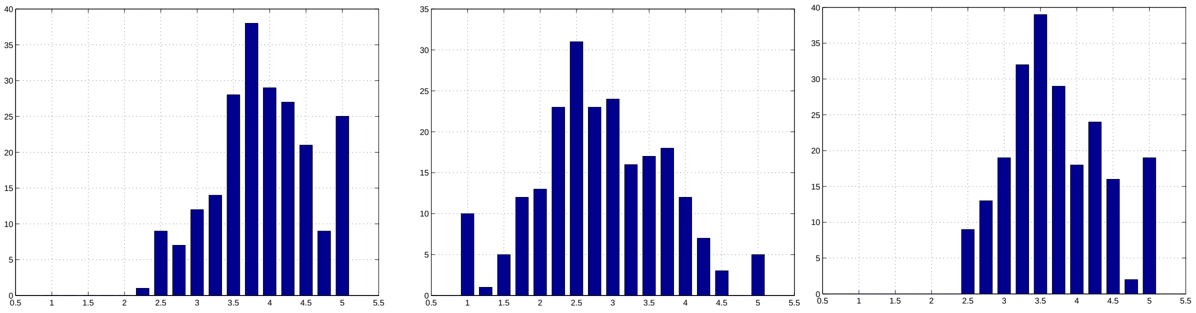

Figure 7: Histograms for the scores of the 220 images. Left: MS-A3C. Middle: Ncut. Right: OWT-UCM.

Ncut and OWT-UCM are both computationally more intensive than MS-A3C. For OWT-UCM, the majority of the computation time is spent on over-segmenting an image in the first step. For Ncut, to acquire the segmentation of an image in a relatively short time, images are scaled to a size with the maximum dimension no greater than 160 pixels. For OWT-UCM and MS-A3C, the original images with size 256x384 (or 384x256) are used. The average running time for Ncut to segment any of the 220 shrunken images is 21.8 seconds on a PC with 2.8 GHz CPU cycles. The average running time for OWT-UCM to segment an original image is 308.7 seconds on 2.8 GHz CPU. MS-A3C completes the segmentation of an original image in 8 seconds on average on a PC with 3.4 GHz CPU cycles. If we convert the CPU cycles to the equivalence of 2.8 GHz, the average segmentation time is roughly 9.8 seconds. In all the cases, the time to load the image into the computer is excluded. For OWT-UCM, the time to search for a proper threshold that yields a desired number of segments is excluded although the time is negligible comparing with that for over-segmentation. We see that Ncut requires more than twice of the time to segment an image shrunken more than half both horizontally and vertically than MS-A3C, while OWT-UCM, operating on images with the original sizes, requires more than thirty times longer time.

5 Conclusions and Discussion

In this paper, we have developed a new image segmentation algorithm that combines the strengths of k-means and A3C clustering to achieve fast and accurate segmentation. A software package for this algorithm is provided at *http://www.stat.psu.edu/∼jiali/msa3c*. A set of features that take into account color variation, edge separation, and global characteristics are developed using various statistics computed from a neighborhood of pixels. Experimental results show that MS-A3C has apparent advantages over existing methods such as k-means, Ncut, and OWT-UCM in terms of segmentation accuracy, computational complexity, and flexibility with setting the number of segments.

For a task as challenging as image segmentation, a direct plug-in of a certain clustering method is insufficient to meet the demands of both speed and quality. Human visual perception of images remains largely a mystery. The method developed here attempts to capture multiple lines of heuristics about the process. The ideas may appear intuitive rather than theoretical, but this simply reflects the complexity of the human cognition. The current work aims primarily on achieving good segmentation rather than theories about clustering and is thus in this sense an application kind (albeit a very difficult one). As an application, our work exemplifies the power of integrating several clustering methods into the design of a coherent system. In another word, clustering methods are viewed as elements for design, which are to be fitted into a system rather than to be selected as a system.

It is found that distortion of an image due to compression, for instance, the JPEG compression scheme, causes jagged boundaries or regions with thin dangling parts. These problems can be alleviated by performing segmentation on resolution reduced images. Artifacts resulting from compression are suppressed at the lower resolution due to the smoothing effect. The idea of smoothing before segmenting noisy images has been investigated in the literature, e.g., segmentation of microarray images [22, 23]. We found that applying our algorithm to smoothed images at the original resolution actually yields worse results. The reason is that the edges are obscured after smoothing. However, by shrinking the image to a smaller size, sensitivity to artifacts is reduced without suffering the loss of prominent edges. When we convert the segmentation result on the shrunken image back to the original size, a simple expansion will lead to blocky boundaries. Therefore, we developed a method that attains refined boundaries at the higher resolution by comparing pixels along the boundaries with pixels inside the segmented regions. If segmentation is performed on an image subsampled several resolutions down, the boundary refinement is iteratively conducted through the resolutions, each iteration raising the resolution of the segmented regions by only one level. This technique of gradually refining segmented regions acquired in a lower resolution can also be used for the mere purpose of smoothing the segmentation boundaries. Suppose segmentation is obtained at a high resolution. We can subsample the segmented regions to a lower resolution. Because of jagged boundaries, small portions of a region may be broken away from the main part. These small broken pieces can be removed. Then when we convert the segmentation back to the original resolution using the aforementioned technique, we obtain regions with smoother boundaries. In our segmentation software package, we implemented the function to segment an image at a lower resolution and then to convert the result back to the original resolution as well as the function to smooth segmentation boundaries. We do not show examples in this paper due to the limitation of space, but the functions are provided in the software package which is available to the public.

One direction of future work is to enhance the computational efficiency of the algorithm. The amount of computation required by any basic bottom-up clustering method grows quadratically with the number of objects to be clustered, hindering scalability. A top-down clustering approach is advantageous in this sense because not all the pairwise distances between objects are needed. The computational complexity is usually linear in both the number of objects and the number of clusters. The computational load of A3C may be lower than quadratic although it is a bottom-up approach because the constraining graph may be sparse, allowing only certain pairs of objects to be grouped. However, this is only true under the assumption that the graph construction itself is not included in the computation, which may not always be so in practice. In our current work, to reduce computation, we use k-means, a top-down approach, to generate an initial set of objects before applying A3C. The number of patches resulting from k-means is much smaller than the number of pixels in an image. As a general strategy to reduce computation, we can also consider dividing the original set of objects into several groups using a top-down approach, and then restricting the agglomerative clustering to objects within the same group or objects within a few similar groups. After a certain number of merging steps, the number of clusters left will be reduced to a value small enough for applying agglomerative clustering at the full scale.

Appendix A

We now prove that the agglomerative algorithm in Section 2 generates clusters satisfying the graph connectivity constraint. Let $\{x_1, ..., x_n\}$ be the set of objects to be clustered. The graph imposed on x_i's is $\mathcal{G}$. The A3C algorithm generates a sequence of clustering results. Initially, every object is one cluster $\mathcal{C}_k^{(0)} = \{x_k\}$, $k = 1, 2, ..., n$. At each iteration, the number of clusters reduces by one due to the merge of two clusters. Let the clusters formed at iteration p be $\{\mathcal{C}_1^{(p)}, ..., \mathcal{C}_{n-p}^{(p)}\}$, $p = 1, 2, ..., n-1$. A graph $\mathcal{G}^{(p)}$ is formed on the nodes $\mathcal{C}_k^{(p)}$'s recursively according to the description in the algorithm.

We first prove the following lemma.

Lemma: If $(\mathcal{C}_k^{(p)}, \mathcal{C}_{k'}^{(p)}) \in \mathcal{G}^{(p)}$, then there exist $x_i \in \mathcal{C}_k^{(p)}$ and $x_j \in \mathcal{C}_{k'}^{(p)}$ such that $(x_i, x_j) \in \mathcal{G}$.

Proof: We prove by induction. When $p = 0$, since $\mathcal{C}_k^{(0)} = \{x_k\}$, this is true by construction. Suppose the statement is true for p. We prove it is also true for $p+1$.

Consider $\mathcal{C}_k^{(p+1)}$ and $\mathcal{C}_{k'}^{(p+1)}$ with $(\mathcal{C}_k^{(p+1)}, \mathcal{C}_{k'}^{(p+1)}) \in \mathcal{G}^{(p+1)}$. Either of the following two cases occurs.

1. If $\mathcal{C}_k^{(p+1)} = \mathcal{C}_{k\uparrow}^{(p)}$ and $\mathcal{C}_{k'}^{(p+1)} = \mathcal{C}_{k'\uparrow}^{(p)}$, that is, both clusters are inherited directly from the previous iteration, then $(\mathcal{C}_{k\uparrow}^{(p)}, \mathcal{C}_{k'\uparrow}^{(p)}) \in \mathcal{G}^{(p)}$ according to the construction of $\mathcal{G}^{(p+1)}$ from $\mathcal{G}^{(p)}$. Hence, there exist $x_i \in \mathcal{C}_{k\uparrow}^{(p)} = \mathcal{C}_k^{(p+1)}$ and $x_j \in \mathcal{C}_{k'\uparrow}^{(p)} = \mathcal{C}_{k'}^{(p+1)}$ such that $(x_i, x_j) \in \mathcal{G}$.

2. Now consider the other case. Without loss of generality, assume $\mathcal{C}_k^{(p+1)}$ is merged from $\mathcal{C}_k^{(p)}$ and $\mathcal{C}_l^{(p)}$, $k < l$, while $\mathcal{C}_{k'}^{(p+1)} = \mathcal{C}_{k'\uparrow}^{(p)}$. If $(\mathcal{C}_k^{(p+1)}, \mathcal{C}_{k'}^{(p+1)}) \in \mathcal{G}^{(p+1)}$, then $(\mathcal{C}_k^{(p)}, \mathcal{C}_{k'\uparrow}^{(p)}) \in \mathcal{G}^{(p)}$ or $(\mathcal{C}_l^{(p)}, \mathcal{C}_{k'\uparrow}^{(p)}) \in \mathcal{G}^{(p)}$.

   (a) If $(\mathcal{C}_k^{(p)}, \mathcal{C}_{k'\uparrow}^{(p)}) \in \mathcal{G}^{(p)}$, then there exist $x_i \in \mathcal{C}_k^{(p)} \subset \mathcal{C}_k^{(p+1)}$ and $x_j \in \mathcal{C}_{k'\uparrow}^{(p)} = \mathcal{C}_{k'}^{(p+1)}$ such that $(x_i, x_j) \in \mathcal{G}$.

   (b) If $(\mathcal{C}_l^{(p)}, \mathcal{C}_{k'\uparrow}^{(p)}) \in \mathcal{G}^{(p)}$, then there exist $x_i \in \mathcal{C}_l^{(p)} \subset \mathcal{C}_k^{(p+1)}$ and $x_j \in \mathcal{C}_{k'\uparrow}^{(p)} = \mathcal{C}_{k'}^{(p+1)}$ such that $(x_i, x_j) \in \mathcal{G}$.

   In summary, if $(\mathcal{C}_k^{(p+1)}, \mathcal{C}_{k'}^{(p+1)}) \in \mathcal{G}^{(p+1)}$, then there exist $x_i \in \mathcal{C}_k^{(p+1)}$ and $x_j \in \mathcal{C}_{k'}^{(p+1)}$ such that $(x_i, x_j) \in \mathcal{G}$.

Hence, we have proved that the statement is true for $p+1$, and thus true for any $p = 1, ..., n-1$.

Next, we prove the following proposition.

Proposition: For any $p = 1, ..., n-1$, $1 \leq k \leq n-p$, $\mathcal{C}_k^{(p)}$ is connected according to $\mathcal{G}$.

Proof: We again prove by induction. When $p = 0$, since $\mathcal{C}_k^{(0)}$ contains only x_k, $\mathcal{C}_k^{(0)}$ is connected by construction. Suppose the proposition holds for p. We prove it also holds for $p+1$. The following two cases apply to $\mathcal{C}_k^{(p+1)}$.

1. If $\mathcal{C}_k^{(p+1)} = \mathcal{C}_{k\uparrow}^{(p)}$, that is, $\mathcal{C}_k^{(p+1)}$ is inherited from the previous iteration, by the assumption, $\mathcal{C}_k^{(p+1)}$ is connected according to $\mathcal{G}$.

2. If $\mathcal{C}_k^{(p+1)}$ is merged from $\mathcal{C}_k^{(p)}$ and $\mathcal{C}_l^{(p)}$, $k < l$, then

   (a) Both $\mathcal{C}_k^{(p)}$ and $\mathcal{C}_l^{(p)}$ are connected according to $\mathcal{G}$.

   (b) $(\mathcal{C}_k^{(p)}, \mathcal{C}_l^{(p)}) \in \mathcal{G}^{(p)}$ since the two cannot be merged otherwise.

   According to the lemma, there exist $x_{i'} \in \mathcal{C}_k^{(p)}$ and $x_{j'} \in \mathcal{C}_l^{(p)}$ such that $(x_{i'}, x_{j'}) \in \mathcal{G}$. Consider any $x_i, x_j \in \mathcal{C}_k^{(p+1)}$. Either of the following two cases occurs.

   (a) Both x_i and x_j belong to $\mathcal{C}_k^{(p)}$ (or $\mathcal{C}_l^{(p)}$). Because the proposition holds at p, x_i and x_j are connected in $\mathcal{G}$.

   (b) x_i belongs to $\mathcal{C}_k^{(p)}$, while x_j belongs to $\mathcal{C}_l^{(p)}$ (or the other way around without loss of generality). Since $\mathcal{C}_k^{(p)}$ is connected, there exists a path between x_i and $x_{i'}$. Similarly, since $\mathcal{C}_l^{(p)}$ is connected, there exists a path between x_j and $x_{j'}$. Because $(x_{i'}, x_{j'}) \in \mathcal{G}$, a path between x_i and x_j can be constructed from x_i to $x_{i'}$, to $x_{j'}$, to x_j. Thus x_i and x_j are connected by $\mathcal{G}$. Therefore, $\mathcal{C}_k^{(p+1)}$ is connected by $\mathcal{G}$.

In summary, we have proved that $\mathcal{C}_k^{(p)}$ is connected according to $\mathcal{G}$, for any $p = 1, ..., n-1$, $1 \leq k \leq n-p$.

Appendix B

Suppose cluster $\mathcal{C}_r$ and $\mathcal{C}_s$ are merged into a new cluster $\mathcal{C}_t$. Let $\mathcal{C}_k$ be any other cluster. The following list provides several schemes of updating the distance $D(\mathcal{C}_t, \mathcal{C}_k)$ from $D(\mathcal{C}_r, \mathcal{C}_k)$ and $D(\mathcal{C}_s, \mathcal{C}_k)$. Let the size of cluster $\mathcal{C}_r$, $\mathcal{C}_s$, $\mathcal{C}_k$ be n_r, n_s, n_k correspondingly.

- *Single linkage*:

$$D(\mathcal{C}_t, \mathcal{C}_k) = \min(D(\mathcal{C}_r, \mathcal{C}_k), D(\mathcal{C}_s, \mathcal{C}_k))$$

- *Complete linkage*:

$$D(\mathcal{C}_t, \mathcal{C}_k) = \max(D(\mathcal{C}_r, \mathcal{C}_k), D(\mathcal{C}_s, \mathcal{C}_k))$$

- *Average linkage*:

$$D(\mathcal{C}_t, \mathcal{C}_k) = \frac{n_r}{n_r + n_s} D(\mathcal{C}_r, \mathcal{C}_k) + \frac{n_s}{n_r + n_s} D(\mathcal{C}_s, \mathcal{C}_k)$$

- *Ward's clustering:*

$$D(\mathcal{C}_t, \mathcal{C}_k) = \frac{n_r + n_k}{n_r + n_s + n_k} D(\mathcal{C}_r, \mathcal{C}_k) + \frac{n_s + n_k}{n_r + n_s + n_k} D(\mathcal{C}_s, \mathcal{C}_k) - \frac{n_k}{n_r + n_s + n_k} D(\mathcal{C}_r, \mathcal{C}_s)$$

References

[1] P. Arbeláez, M. Maire, C. Fowlkes, and J. Malik, "From contours to regions: an empirical evaluation," *CVPR*, pp.2294-2301, 2009.

[2] J. D. Banfield and A. E. Raftery, "Model-based Gaussian and non-Gaussian clustering," *Biometrics*, 49:803-821, 1993.

[3] H. D. Cheng, X. H. Jiang, Y. Sun, and J. Wang, "Color image segmentation: advances and prospects," *Pattern Recognition*, 34(12):2259-2281, 2001.

[4] H. Chipman and R. Tibshirani, "Hybrid hierarchical clustering with applications to microarray data," *Biostatistics*, 7(2):286-301,2006.

[5] A. P. Dempster, N. M. Laird, and D. B. Rubin, "Maximum likelihood from incomplete data via the EM algorithm," *Journal Royal Statistics Society*, 39(1):1-21, 1977.

[6] B. S. Everitt, S. Landau, and M. Leese, *Cluster Analysis*, Oxford University Press US, 2001.

[7] M. D. Fairchild, *Color Appearance Models*, 2nd Ed., Wiley, 2005.

[8] C. Fraley and A. E. Raftery, “Model-based clustering, discriminant analysis, and density estimation,” *Journal of the American Statistical Association*, 97:611-631, 2002.

[9] J. Freixenet, X. Munoz,, D. Raba, J. Marti, and X. Cufi, “Yet another survey on image segmentation: region and boundary information integration,” *Lecture Notes in Computer Science,, vol. 2352, Proc. ECCV,* pp. 21-25, 2002.

[10] A. Gersho and R. M. Gray, *Vector Quantization and Signal Compression*, Kluwer Academic Publishers, 1992.

[11] R. C. Gonzalez, R. E. Woods, *Digital Image Processing*, 2nd Ed., Prentice Hall, 2002.

[12] P. Hansen, B. Jaumard, C. Meyer, B. Simeone, V. Doring, “Maximum split clustering under connectivity constraints,” *Journal of Classification*, vol. 20, pp. 143-180, 2003.

[13] R. M. Haralick, L. G.. Shapiro, “ Image segmentation techniques,” *Applications of Artificial Intelligence II.* , 548:2-9, 1985.

[14] T. Hastie, R. Tibshirani, and J. Friedman. The Elements of Statistical Learning. Springer-Verlag, 2001.

[15] A. K. Jain and R. C. Dubes, *Algorithms for Clustering Data*, Prentice-Hall, Inc., NJ, USA, 1988.

[16] J. Li, “Clustering based on a multi-layer mixture model,” *Journal of Computational and Graphical Statistics*, 14(3):547-568, 2005.

[17] J. Li, R. M. Gray, *Image Segmentation and Compression Using Hidden Markov Models (monograph),* Springer, 2000.

[18] J. Li, J. Z. Wang, “Real-time computerized annotation of pictures,” *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 30(6):985-1002, 2008.

[19] G. J. McLachlan and D. Peel, *Finite Mixture Models*, New York: Wiley, 2000.

[20] E. R. C. Morales, and Y. Y. Mendizabal, “Building and assessing a constrained clustering hierarchical algorithm,” *Lecture Notes in Computer Science (LNCS),* vol. 5197, pp. 211-218, 2008.

[21] F. Murtagh, “A survey of algorithms for contiguity-constrained clustering and related problems,” *The Computer Journal*, vol. 28, no. 1, pp. 82-88, 1985.

[22] P. Qiu, and J. Sun, “Local smoothing image segmentation for spotted microarray images,” *JASA*, vol. 102, pp. 1129-1144, 2007.

[23] P. Qiu, and J. Sun, “Using conventional edge detectors and post-smoothing for segmentation of spotted microarray images,” *Journal of Computational and Graphical Statistics*, vol. 18, no. 1, pp. 147–164, 2009.

[24] J. C. Russ, *The Image Processing Handbook*, 5th ed., CRC, 2006.

[25] J. Shi and J. Malik, “Normalized cuts and image segmentation,” *IEEE Trans. Pattern Analysis and Machine Intelligence*, vol. 22, no. 8, pp. 888-905, 2000.

[26] S. Vicente, V. Kolmogorov, and C. Rother, “Graph cut based image segmentation with connectivity priors,” *CVPR*, pp. 1-8, Anchorage, AK, 2008.

[27] K. Wagstaff, C. Cardi, S. Rogers, and S. Schroedl, “Constrained k-means clustering with background knowledge,” *Proc. Int. Conf. Machine Learning*, 2001.

[28] J. Z. Wang, J. Li, G. Wiederhold, “SIMPLIcity: Semantics-sensitive integrated matching for picture libraries,” *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 23(9):947-963, 2001.

[29] Z. Wu, R. Leahy, “An optimal graph theoretic approach to data clustering: theory and its application to image segmentation,” *IEEE Trans. Pattern Analysis and Machine Intelligence*, 15(11):1101-1113, 1993.

[30] E.P. Xing, A.Y. Ng, M.I. Jordan, and S. Russell, “Distance metric learning, with application to clustering with side-information,” *Advances in Neural Information Processing Systems 16 (NIPS2002)*, 521-528, 2002.