A Graph-based Clustering Method for Image Segmentation

Thang Le[1], Casimir Kulikowski[1], Ilya Muchnik[2]

[1]Department of Computer Science, Rutgers University
[2]DIMACS, Rutgers University

Abstract. We present a novel graph-based approach to image segmentation which can be applied to either greyscale or color images. The assumption is that nearby pixels with similar colors or greyscale intensities may belong to the same region or segment of the image. A graph representation for an image is derived from the similarity between the pixels, and then partitioned by a computationally efficient graph clustering method, which first identifies representative nodes for each cluster and then expands them to obtain complete clusters of the graph. Experiments with synthetic and natural images are presented. A comparison with the well known normalized cut method shows that our approach can be faster and produces segmentations that are in better agreement with visual assessment of the original images.

1 Introduction

Image segmentation is a challenging problem in computer vision and usually used for finding objects and their boundaries in images. Depending on the objectives, many definitions and criteria have been proposed and employed for image segmentation. Here, we address the problem of segmenting a digital image into a set of disjoint regions such that each region is composed of nearby pixels with similar colors or intensities.

An image can be represented by a proximity graph in which nodes represent image pixels and edges reflect pairwise similarities between the pixels. Weights of edges are computed by a similarity function of properties of corresponding pixels such as location, brightness and color. With this representation, the segmentation task can be solved by graph clustering methods. Let us consider an undirected graph $G = (V, E, W)$, where the set of nodes V represents a set of data objects, the set of edges E represents the relationships between data objects, and W is a symmetric matrix where the entry $w_{ij} \in [0, 1]$ is the weight of the edge between nodes i and j. As G is a proximity graph, the edge weight w_{ij} represents the degree of similarity between the objects corresponding to i and j, a higher value of w_{ij} implies a higher similarity degree between i and j. In graph clustering, a graph is partitioned into subgraphs such that nodes of a subgraph are strongly or densely connected while nodes belonging to different subgraphs are weakly or sparsely connected. In other words, we discover subgraphs such that the sum of edge weights inside subgraphs is high, while the sum of weights of edges connecting different subgraphs is low. Applying a graph clustering method to a proximity graph will partition it into subgraphs, such that each subgraph corresponds to a group of similar objects, which are dissimilar to objects of groups corresponding to other subgraphs. Thus, pixels in a segment of an image will correspond to nodes in a cluster of the proximity graph of that image.

In this paper, we extend and apply the clustering method described in [1] to segmenting greyscale and color images. This graph clustering method works very well if each cluster consists of a region of high density surrounded by a region of low density. It detects nodes in dense regions and cluster cores are then extracted, so nodes can be assigned to the nearest cluster. We describe the graph representation of images in Section 2. Details of the segmentation algorithm are discussed together with calibration examples for parameter determination in Section 3. Section 4 presents our comparisons with the normalized cut approach tested for efficiency and segmentation performance on a representative subset taken from the Berkeley dataset [5].

2 Graph-based Representation for Images

Based on a digital image, we construct a proximity graph $G = (V, E, W)$, where each node in V represents a pixel, the weight w_{ij} of the edge between two nodes corresponding to pixels i and j reflects the likelihood that i and j belong to the same segment in the image. Since we want to group nearby pixels that have a similar intensity/color, the weights of graph edges are computed by a likelihood function based on the location and intensity/color of neighbouring pixels. For greyscale images, we use the weight function described in [2]:

$$w_{ij} = \begin{cases} e^{-\left(\frac{I(i)-I(j)}{\sigma_I}\right)^2 - \left(\frac{dist(i,j)}{\sigma_d}\right)^2} & \text{if } dist(i,j) < r \text{ ,} \\ 0 & \text{otherwise ,} \end{cases} \tag{1}$$

where $I(i) \in [0,1]$ is the intensity of pixel i, $dist(i, j)$ is the Euclidean distance in pixels between i and j. For color images, we replace the difference of intensity in (1) by the normalized Euclidean distance between pixel colors:

$$w_{ij} = \begin{cases} e^{-\left(\frac{\|C(i)-C(j)\|_2}{\sqrt{3}\sigma_I}\right)^2 - \left(\frac{dist(i,j)}{\sigma_d}\right)^2} & \text{if } dist(i,j) < r \text{ ,} \\ 0 & \text{otherwise ,} \end{cases} \tag{2}$$

where $C(i)$ is a vector of three features, namely red, green, and blue color components of pixel i, so $\|C(i) - C(j)\|_2 \in [0, \sqrt{3}]$. There is an edge linking two nodes only if the distance between the corresponding pixels is less than r pixels, so each node is connected to approximately πr^2 other nodes. The proximity graph is very sparse as $\pi r^2 << |V|$. By using these weight functions, strong edges exist between nodes whose corresponding pixels have similar intensity/color and are close to each other. Therefore, pixels inside a segment with homogeneous intensity/color (i.e., the inner or core region of a segment) have their nodes in the proximity graph strongly connected. On the other hand, pixels at boundaries of segments often have neighbour pixels with dissimilar intensity/color, so their corresponding nodes are usually weakly connected to one another. Note that with the weight functions (1) and (2) above, we can easily evaluate and validate segmentation results by visual inspection and assessment assuming that pixels with similar intensity or color should fall into the same segment.

Obviously, a good result for a graph-based segmentation method depends on the condition that segments of the image translate into well-separated clusters of the proximity graph. Therefore, our settings for parameters σ_I, σ_d, and r are important as they decide how images are transformed into proximity graphs. In our experiments, we found a robust set of parameter values which did not require changing across a wide range of image types. Typically, $\sigma_I = 0.07$, $\sigma_d = 8$, and $r = 11$ for images of size less than 200×300. In fact, the value of σ_I is a trade-off between the similarity of intensity/color of pixels in the same segment and the dissimilarity of intensity/color of pixels belonging to different segments. A higher value for σ_I would allow a higher tolerance for differences of pixel intensity/color within each segment. However, then it would be harder to distinguish two segments that have a similar average intensity/color. So the σ_I setting depends on the contrast of the image, low contrast images may use a smaller value of σ_I while high contrast images may use a larger value. Parameters σ_d and r specify how spatial information is incorporated into the weight function. They determine the likelihood that neighbour pixels belong to the same segment. Higher values for σ_d and r make a segment span to greater distances over regions of heterogeneous intensity/color. This is a trade-off between detecting weakly separated segments and not breaking a large segment having some heterogeneous regions inside into smaller parts. We may need to increase values of σ_d and r for larger images because of the likelihood that large images contain larger segments.

3 Image Segmentation Algorithm

As discussed in Section 2, a digital image can be represented by a proximity graph such that segments of the image are represented by clusters on the graph with each cluster having a region of high density (at the core of the corresponding segment) surrounded by a region of low density (at the boundaries of the corresponding segment). To group image pixels, we apply the coring method [1], which is a general clustering method to find clusters in an arbitrary proximity graph. The main idea is that cores of clusters should be obtainable by analysis of neighbourhood relationships between objects in a particular space which reflects object similarities. In most practical problems, direct analysis is unrealistic due to the high dimensionality of the space, but a heuristic allows reducing this to a one-dimensional ordinal sequence of density variation for a set of objects contained in a graph *G*. In the coring method, a local density at node *i* with respect to $H \subseteq V$ is measured by the function $d(i, H)$:

$$d(i,H) = \frac{1}{|H|} \sum_{j \in H} w_{ij} \ . \tag{3}$$

The minimum density of *H* is defined by function $D(H)$:

$$D(H) = \min_{i \in H} d(i,H) \ . \tag{4}$$

The node $m = \operatorname{argmin}_{i \in H} d(i,H)$ is called the weakest node of *H* as it has the minimum density. The method computes the variation of minimum density *D* values while the weakest node is iteratively removed from the graph. If clusters of the graph have a dense core, we can identify nodes belonging to cluster cores by analyzing the sequence of *D* values. Specifically, if there is a significant drop in *D* value after the removal of a node, this node is highly connected with other nodes in a dense region and it is potentially a core node because its elimination drastically reduces the density of the region around it.

In [1], the local density at a node is defined with the normalized term $|H|$ as shown in (3). This term is necessary if the graph *G* is densely connected because the sum $\sum_{j \in H} w_{ij}$ for any node *i* depends on most of the nodes of the graph, in other words, this sum tends to decrease if we remove some nodes from the graph. The normalized term makes the local density estimation more accurate when parts of the graph are removed. However, in cases where graph *G* is sparse, which almost always applies when *G* is the proximity graph of an image, each node is connected to only a small and fixed number of neighbours. For any node *i*, the sum $\sum_{j \in H} w_{ij}$ depends on only several neighbour nodes and therefore this sum for most of the nodes does not change when we remove parts of the graph. Thus, here we eliminate this normalized term and estimate the local density at node *i* of *H* by:

$$d(i,H) = \sum_{j \in H} w_{ij} \ . \tag{5}$$

3.1 Image Segmentation Algorithm

In this section, we present the segmentation algorithm which partitions an image by building and clustering its proximity graph. Algorithm steps are outlined in the following procedure.

Input: An image *I*.
Output: Segmentation of the image *I*.
1. Build a proximity graph *G* for the input image *I*.
2. Compute the sequence of density variation for *G*.
3. Identify a set of core pixels based on the sequence of density variation.
4. Partition the set of core pixels into groups.
5. Expand the groups of core pixels to get the image segmentation.

3.1.1 Build a proximity graph *G* for the input image

We construct the proximity graph of an image as described in Section 2.

```
**Inputs:** Image $I$; Parameters $\sigma_I$, $\sigma_d$, and $r$.
**Output:** Proximity graph $G$.
Create a node in $G$ for each pixel of $I$.
for each node $v$ of $G$
    Create edges in $G$ connecting $v$ with nodes corresponding to pixels locating
    within a distance $r$ from the pixel of $v$, edge weights are computed by (1) if $I$
    is a greyscale image, by (2) if $I$ is a color image.
return
```

Each pixel is connected to its neighbours within a radius of r pixels. So each node of the proximity graph has about πr^2 incident edges, and $2|E| \approx \pi r^2|V|$. The time complexity to build this graph is $O(|E|)$ which is equal to $O(|V|)$ because of the sparseness of the graph.

3.1.2 Compute the sequence of density variation for *G*

Given a graph *G*, the sequence of density variation is constructed iteratively. At each iteration t, we compute the minimum density D_t by formulas (4, 5) and determine the weakest pixel M_t. The procedure returns the sequence of D_t values and the sequence of corresponding M_t pixels.

```
Input: Proximity graph $G = (V, E, W)$.
Output: Sequences of $D_t$s and $M_t$s.
$n \leftarrow |V|$
$H \leftarrow V$
for $t = < 1, 2, ..., n-1, n >$
    $D_t \leftarrow \min_{i \in H} \Sigma_{j \in H} w_{ij}$
    $M_t \leftarrow \mathrm{argmin}_{i \in H} \Sigma_{j \in H} w_{ij}$
    $H \leftarrow H - M_t$
return
```

The procedure computes the local density for every node and then iterates $|V|$ times. At each iteration, we find the pixel with the minimum density, then remove it and update the densities for its neighbours on the graph. Using Fibonacci heaps, we can quickly extract the minimum value and decrease key values for neighbouring pixels. The amortized cost of this step is $O(|E| + |V| \log |V|)$.

3.1.3 Identify a set of core nodes

Using the sequences of D_ts and M_ts, core pixels are identified based on two particular parameters $\delta \in [0,1)$ and $\beta \in \mathbf{N}$. The role of δ is to specify the minimum rate of decrease for the D values of the core pixels, and the role of β is to control the minimum size of a group of successive core pixels in the M_t sequence.

Inputs: Sequences of D_ts and M_ts; Parameters δ, β.

```
Output: A set of core pixels.
Compute the local rates of decrease $R_t$ in the $D_t$ sequence: $R_t = (D_t - D_{t+1})/D_t$.
Sort the list of positive $R_t$s in ascending order.
$\alpha \leftarrow$ The value of $R_t$ at the position $\delta$ relative to the beginning on the sorted list of positive $R_t$s.
for $t = < 1, 2, ..., n-1, n >$
    $M_t$ is extracted as a core pixel if it is among a set of at least $\beta$ successive $M_t$s that have corresponding $R_t$s $> \alpha$.
return
```

The procedure scans the sequences of D_ts and M_ts to find core pixels satisfying the conditions, so its time complexity is $O(|V| + |V_p| \log |V_p|)$, where V_p is the set of pixels that have a positive rate of decrease, $|V_p| << |V|$.

3.1.4 Partition the set of core pixels into groups

This step partitions the set of core pixels produced by the previous step into groups, each of which is then considered to be the core of a segment.

```
Input: The set of core pixels.
Output: Cores of segments.
In the graph induced by the core pixels:
Remove edges whose weights are less than $\theta$.
Find connected components of the graph, each component represents the core of a segment.
return
```

We remove weak edges to separate neighboring core groups which happen to reside within r pixels from each other. The threshold θ is set to a small value (0.1). One can use a breadth-first or depth-first search algorithm to find connected components in the graph induced by the core pixels. Its time complexity is $O(|V_C| + |E_C|)$, where V_C and E_C are the sets of core pixels and incident edges, respectively. Note that $|V_C| << V|$ and $|E_C| << |E|$.

3.1.5 Expand core groups to find image segmentation

The sequence of density variation progresses from low to high density regions on the proximity graph. So now in this expanding step, we scan the M_t sequence in the backwards direction to move from dense to sparse regions of the graph. While scanning the sequence, we assign pixels to the most similar segment.

```
Inputs: Proximity graph $G = (V, E, W)$; $M_t$ sequence; Cores of segments.
Output: Segmentation $S$.
n $\leftarrow$ |V|
$S \leftarrow$ {cores of segments}
$L \leftarrow$ {}
for $t = < n, n-1, ..., 2, 1 >$
```

```
    if $M_t$ is not a core pixel then
        $m1 \leftarrow \max_{C \in S} \text{average}_{i \in C,\, i \notin L,\, w_{M_t i} > 0}\, w_{M_t i}$
        $s \leftarrow \text{argmax}_{C \in S} \text{average}_{i \in C,\, i \notin L,\, w_{M_t i} > 0}\, w_{M_t i}$
        $m2 \leftarrow \text{max2}_{C \in S} \text{average}_{i \in C,\, i \notin L,\, w_{M_t i} > 0}\, w_{M_t i}$
        if $m1 > 0$ then
            Add $M_t$ to segment $s$ of $S$.
            if $m2 > \lambda * m1$ then
                $L \leftarrow L \cup \{M_t\}$
            end if
        else
            Add a new segment containing $M_t$ to $S$.
        end if
    end if
return
```

In the above procedure, function max2 returns the second maximum value of a set. The similarity between a pixel M_t and a segment $C \in S$ is computed by $\text{average}_{i \in C,\, i \notin L,\, w_{M_t i} > 0}\, w_{M_t i}$, where L is the set of low-confident pixels determined by the condition $m2 > \lambda * m1$. We typically set λ to 0.5, therefore a pixel belongs to L if its similarity with the second nearest segment is greater than half of its similarity with the nearest segment. Note that if a pixel is not connected to any known segments, then a new segment is created to accommodate it. This situation may occur if δ and β are set to so high values that step 3 excludes all the representatives of some strong segments from the set of core pixels. With the graph adjacency-list representation, the time complexity of this expanding step is $O(|E|)$.

3.2 Experimental Study for Determining Parameter Settings

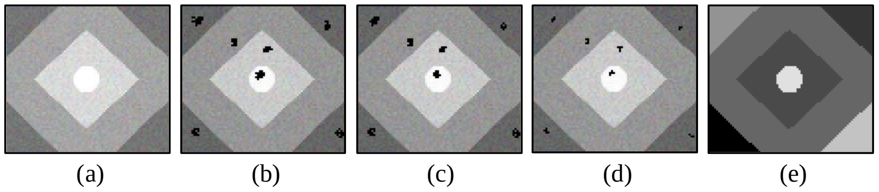

Fig. 1. (a) is a greyscale image. (b), (c), and (d) show core pixels when $\beta = 3$, $\delta = 90\%$, 96%, and 99%, respectively. (e) shows the segmentation result

Fig. 1 (a) is a greyscale image consisting of 7 regions of nearby pixels with similar intensities. To illustrate the role of core pixels for finding a segmentation of the image, Fig. 1 (b) shows the location of core pixels with $\beta = 3$, $\delta = 90\%$. The set of core pixels consists of 7 connected components representing the cores of 7 segments. Expanding these cores gives us the expected segmentation as shown in Fig. 1 (e) where different segments are labeled by different colors. These results demonstrate an exact match between the number of regions in the original image and the number of core groups found for these parameter settings. It is interesting that the same parameter settings work well for segmenting a wide range of more complex natural images, such as those from the widely referenced Berkeley dataset [6] as shown in Fig. 5. Here, along with the original images, we illustrate the segmentations and the derived boundaries. By visual assessment, these results satisfy our original objective of grouping nearby pixels with similar color into the same segment.

Since the number of nodes is finite, there are a limited number of possible settings for β and δ. Parameter β is an integer and its role is to eliminate some isolated noisy pixels at the top of the list of pixels with high rates of decrease in *D* values. Usually we fix β to be 3 or 4. The main parameter of the clustering method is δ, changing δ can result in increasing or decreasing the number of segments. In (b), (c), and (d) of Fig. 1, we illustrate the effect of parameter δ on the set of core pixels. The cores of segments shrink when we increase δ, conversely they are enlarged if δ is decreased. Yet the segmentation results remain the same as shown in (e). In general, we avoid setting δ to a value lower than 90% because it may produce core pixels that are not very reliable, especially in noisy images.

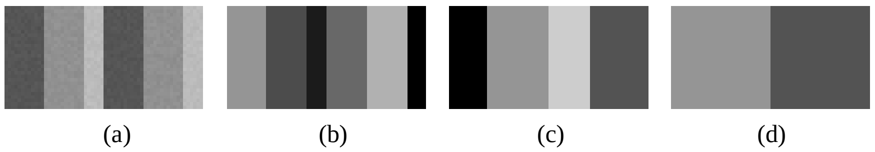

Fig. 2. (a) is a greyscale image. (b), (c), and (d) shows segmentations of the image with δ = 97%, 98%, and 99%, respectively

In images which contain a mixture of segments of different sizes or levels, by increasing or decreasing δ, we can obtain a coarse or fine-grained segmentation. In Fig. 2, (a) shows a greyscale image, (b), (c), and (d) show segmentation results of (a) with different values of δ. Higher δ produces coarser segmentation while lower δ yields finer segmentation. This illustrates the point that core pixels are arranged in an order such that the pixels having high rates of decrease in their *D* values belong to the cores of strong segments.

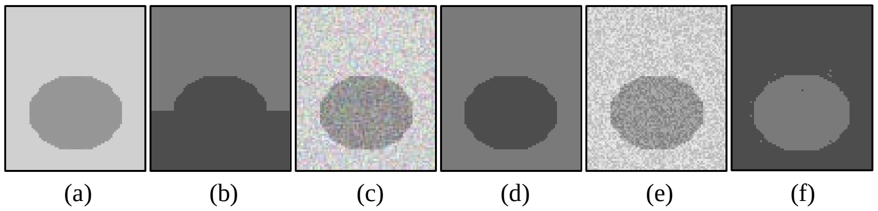

Fig. 3. (b) is the segmentation by the normalized cut on the image in (a). Images (c) and (e) are created by adding noise to (a). (d) and (f) show the segmentations by our method on (c) and (e), respectively

4 Comparisons with the Normalized Cut Method [2]

Spectral clustering methods in general and normalized cuts in particular are well known graph clustering approaches [2, 7, 8]. Application of normalized cut to image segmentation has been shown in [2]. An implementation of this method developed by its authors is available on the web [3]. The method basic idea is to partition a graph $G = (V, E, W)$ into k subgraphs $A_1, A_2, \ldots, A_k$ based on the minimum k-way normalized cut, which is defined by:

$$Ncut_k = \sum_{i=1}^{k} \frac{\sum_{u\in A_i,\, v\in V-A_i} w_{uv}}{\sum_{u\in A_i,\, v\in V} w_{uv}} \,. \tag{6}$$

Finding the exact minimum normalized cut is a NP-hard problem, so an approximate solution is estimated using eigenvectors of the normalized Laplacian matrix $L = I - D^{-1}W$, where D is the diagonal matrix of vertex degrees. The complexity of solving this approximation is relatively high, about $O(|V|^{2.5})$ for sparse graphs [2]. As is true of many clustering methods, the normalized cut method requires that the number of clusters k be pre-specified, which is especially problematic for image segmentation, since the number of segments is highly variable depending on the scene in the images. The number of segments in an image is not something that we would think about in the first place, but rather is a natural result of the perception process. Yet a more significant problem with minimizing normalized cuts is that the normalizing factors in the criterion function (6) make the cut favor clusters of similar sizes. As a result, small clusters are very easily omitted while large clusters are often split up into small parts. In Fig. 3, (b) shows the segmentation of the image in (a) by the two-way normalized cut. The cut partitions the image into two similar size segments. It fails to find the oval as one segment because of the disparity between the number of pixels inside and outside the oval. (g), (h), (i), (j), (k), and (l) of Fig. 4 show the segmentations for the images in (a) by normalized cuts with k = 2, 3, 4, 5, 6 and 7, respectively. Clearly changing k dramatically changes the segmentation, and for any k, the image is always partitioned into segments of roughly similar sizes.

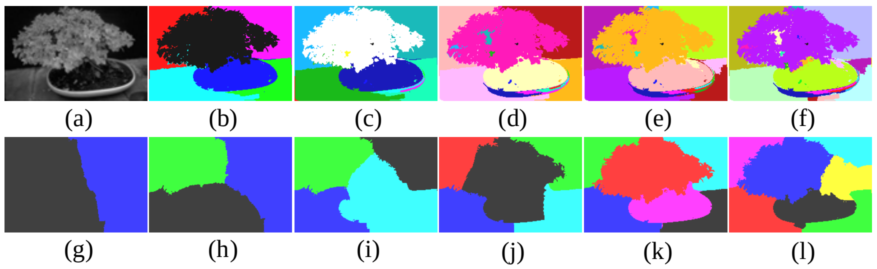

Fig. 4. Effects of parameters on segmentation results. Greyscale images in (a) are taken from the Berkeley dataset. (b), (c), (d), (e), and (f) show the segmentations by our method with δ=99%, 97%, 95%, 93%, and 90%, respectively. (g), (h), (i), (j), (k), and (l) show the segmentations of the normalized cut method on the same proximity graph with k = 2, 3, 4, 5, 6 and 7, respectively

Based on the same proximity graphs, our segmentation algorithm shows better results and performance. In Fig. 4, (b), (c), (d), (e), and (f) are our segmentations for the images in (a) with $\beta = 3$ and $\delta = 99\%, 97\%, 95\%, 93\%$, and 90%, respectively, where lowering δ yields finer segmentation. Note that the results are very consistent in their sensitivity to perturbations of the different parameter settings. In terms of speed, our implementation has a total time complexity of $O(|E| + |V| \log |V|)$. Experiments show that it is much faster than the normalized cut method implemented to run on the same proximity graphs. In Tab. 1, we show the approximate execution times on different proximity graphs of the two methods using a PC with a CPU of Core 2 Duo 2.4GHz and 2GB RAM. It can be seen that the running time of the coring method is roughly linear to the number of edges of the proximity graphs. Running times shown for the normalized cut are the average time for partitioning the graph into 2, 3, and 4 segments. For the cases that the graphs contain more than $105*10^3$ nodes and $17.6*10^6$ edges, the normalized cut method fails to execute because of an 'out of memory' error. An additional advantage of the coring method is that we can change β or δ and quickly obtain a new result by re-executing fast steps 3, 4 and 5 of the method. In contrast, for the normalized cut method, changing k will result in re-computing the cut from scratch.

Table 1. Execution times of the normalized cut and coring methods for clustering graphs

Proximity graphs		Normalized cut (seconds)	Coring method (seconds)
#nodes (10^3)	#edges (10^6)		
30	4.7	9.3	0.5
45	7.1	16	0.7
60	10.7	30	1.0
75	12.2	60	1.2
90	14.8	106	1.4
105	17.6	146	1.6
120	18.7	NA	1.8
500	50	NA	5.2
1000	70	NA	7.5

5 Conclusion

We have developed and evaluated a graph-based clustering method for image segmentation. Using proximity graphs, we partition images into segments of nearby pixels with similar intensities or colors. The method is simple and fast. It is also stable and robust to noise because it identifies and uses the pixels in cores of segments. For example in Fig. 3, we add noise to (a) and get images (c) and (e), for which the segmentation results remain accurate as shown in (d) and (f). In addition to speed and robustness, an advantage is that parameters can be adjusted to yield a range of results from a coarse to very fine-grained segmentation. The weight functions (1) and (2) do not take into account texture information. It would be possible to apply the method to texture segmentation by additionally incorporating statistical texture information around each image pixel into the weight functions, which should help refine the segmentation results for cases where texture is an important feature.

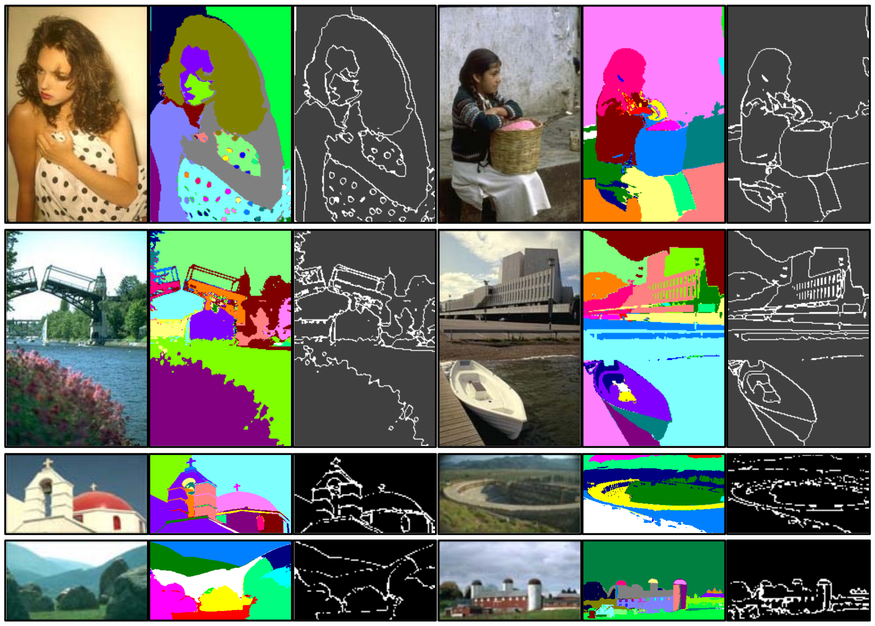

Fig. 5. Segmentations and derived boundaries on images from the Berkeley dataset. The same parameter settings β=3, δ=97% are used for all the image segmentations

References

1. Le, T., Kulikowski, C., Muchnik, I.: Coring method for clustering a graph, DIMACS Technical Report (2008)
2. Shi, J., Malik, J.: Normalized cuts and image segmentation. IEEE Trans. on Pattern Analysis and Machine Intelligence (2000) 888-905
3. Shi, J.: Normalized cut image segmentation code, http://www.cis.upenn.edu/~jshi/software
4. Charikar, M.: Greedy approximation algorithms for finding dense components in a graph. Volume 1913 of Lecture Notes in Computer Science, Springer-Verlag (2000) 84-95
5. Berkeley segmentation dataset, http://www.cs.berkeley.edu/projects/vision/bsds/
6. Martin, D., Fowlkes, C., Tal, D., Malik, J.: A database of human segmented natural images and its application to evaluating segmentation algorithms and measuring ecological statistics. Proc. 8th International Conference on Computer Vision (2001) 416-423
7. Luxburg, U.: A tutorial on spectral clustering. Statistics and Computing (2007) 395-416
8. Kannan, R., Vempala, S., Vetta, A.: On Clusterings: Good, Bad and Spectral. Proc. 41st Annual Symposium on the Foundation of Computer Science (2000) 367-380