

A Graph-based Approach for Image Segmentation

Thang V. Le¹, Casimir A. Kulikowski¹, and Ilya B. Muchnik²

¹Department of Computer Science, Rutgers University, Piscataway, NJ 08854, USA

²DIMACS, Rutgers University, Piscataway, NJ 08854, USA

Abstract. We present a novel graph-based approach to image segmentation. The objective is to partition images such that nearby pixels with similar colors or grayscale intensities belong to the same segment. A graph representing an image is derived from the similarity between the pixels and partitioned by a computationally efficient graph clustering method, which 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 graph clustering method of normalized cuts shows that our approach is faster and produces segmentations that are in better agreement with visual assessment on original images.

1 Introduction

Image segmentation is a challenging problem in computer vision. Depending on user objectives, different definitions and criteria have been proposed and employed for image segmentation. In this paper, we address the problem of segmenting a grayscale or color image into a set of disjoint regions such that each region is composed of nearby pixels with similar intensities or colors.

We represent an image by a proximity graph in which nodes correspond to image pixels, and edges reflect pairwise similarities between the pixels. Weights of edges are computed by a similarity function based on properties of corresponding pixels such as their location, brightness and color. With this representation, image segmentation 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 . If G is a proximity graph, then 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 the graph clustering problem, 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 weights of edges inside a subgraph is high, while the sum of weights of edges connecting different subgraphs is low. Applying a graph clustering algorithm 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, the set of nodes of each cluster of the proximity graph representing an image correspond to the set of pixels of a segment of that image.

In this paper, we extend a new graph clustering technique [1] to tackle the problem of segmenting grayscale and color images. The clustering approach is called coring method and it works very well if each cluster consists of a region of high density surrounded by a region of low density. We describe the graph representation for images in Section 2. The segmentation algorithm is discussed together with a calibration example for parameter determination in Section 3. Section 4 presents our comparisons with the normalized cut approach tested for efficiency and segmentation performance on images taken from the Berkeley dataset [5].

2 Graph-based Representation for Images

Based on a digital image, we construct a proximity graph $G = (V, E, W)$. Each node of 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 neighboring pixels. For grayscale images, we use the weight function described in [2]:

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

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 color components of the pixels:

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

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 $\pi r^2 \ll |V|$. By using the above weight functions, strong edges exist between nodes whose corresponding pixels are close to each other and have similar intensity/color. 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 their neighbor pixels with dissimilar intensity/color, so corresponding nodes are usually weakly connected to one another. Note that by using the weight functions (1) and (2), we can easily evaluate and validate segmentation results by visual inspection and assessment as pixels with similar intensity or color should be in the same segment.

A good result for a graph-based segmentation method obviously depends on the condition that segments of the image translate into well-separated clusters of the proximity graph. The settings for parameters σ_I , σ_d , and r are therefore important as they determine how images are transformed into proximity graphs. 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, it would then be harder to distinguish two segments that have a similar average intensity/color. The setting of σ_I should depend on the contrast of the image, low contrast images may use a smaller value while high contrast images may use a larger value of σ_I . Parameters σ_d and r specify how spatial information is incorporated into the weight function. They determine the likelihood that neighbor 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. We typically set $\sigma_I = 0.07$, $\sigma_d = 8$, and $r = 11$ for images of size less than 200×300 . In our experiments, these settings are robust as they worked well across a wide range of image types.

3 Image Segmentation Algorithm

As shown in Section 2, an image can be represented by a proximity graph such that segments of the image correspond to clusters of the graph and each cluster has 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 find clusters in proximity graphs of images, we use a new graph clustering method described in [1]. The key idea is that cores of clusters are identifiable by analyzing neighborhood 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, a heuristic allows reducing this to a one-dimensional ordinal sequence of density variation for a set of objects contained in a graph G . The local density at node i of a subgraph $H \subseteq V$ is measured by a function $d(i, H)$:

$$d(i, H) = \sum_{j \in H} w_{ij} . \quad (3)$$

Based on $d(i, H)$, we define the minimum density of H as follows:

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

The node $m = \operatorname{argmin}_{i \in H} d(i, H)$ is called the weakest node of H as it has the minimum local density. The coring method computes the minimum density D value 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 variation 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.

3.1 Image Segmentation Algorithm

In this section, we describe the segmentation algorithm which partitions an image by building and clustering its proximity graph. The algorithm consists of 5 steps.

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.

Inputs: Image I ; Parameters σ_l , σ_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 grayscale image, by (2) if I is a color image.

return

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

3.1.2 Compute the sequence of density variation for G . The sequence of density variation for a graph G is constructed iteratively. At each iteration t , we determine the minimum density D_t by (3) and (4) and the weakest pixel M_t . The procedure produces the sequences of D_t values and M_t pixels.

Input: Proximity graph $G = (V, E, W)$.

Output: Sequences of D_t and M_t .

$n \leftarrow |V|$

$H \leftarrow V$

for $t = \langle 1, 2, \dots, n-1, n \rangle$

$D_t \leftarrow \min_{i \in H} \sum_{j \in H} w_{ij}$

$M_t \leftarrow \operatorname{argmin}_{i \in H} \sum_{j \in H} w_{ij}$

$H \leftarrow H - M_t$

return

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

3.1.3 Identify a set of core pixels. Based on the sequences of D_t and M_t , core pixels are identified using two parameters $\delta \in [0,1)$ and $\beta \in \mathbb{N}$. The role of δ is to specify the minimum rate of decrease for the D values of the core pixels, and β is to control the minimum size of a group of successive core pixels in the M_t sequence.

Inputs: Sequences of D_t and M_t ; Parameters δ, β .

Output: A set of core pixels.

Compute the local rates of decrease R_t on the D_t sequence: $R_t = (D_t - D_{t+1})/D_t$.

Sort the list of positive R_t in ascending order.

$\alpha \leftarrow$ The value of R_t at the position δ relative to the beginning on the sorted list of positive R_t .

for $t = \langle 1, 2, \dots, n-1, n \rangle$

M_t is extracted as a core pixel if it is among a set of at least β successive M_t that have the corresponding $R_t > \alpha$.

return

The procedure scans the sequences of D_t and M_t to find core pixels satisfying the conditions, so it runs in $O(|V| + |V_p| \log |V_p|)$, where V_p is the set of pixels that have a positive rate of decrease, note that $|V_p| \ll |V|$.

3.1.4 Partition the set of core pixels. The set of core pixels produced by the last step is partition into groups; each group is considered 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 smaller than θ .

Find connected components of the core graph; each component represents the core of a segment.

return

Weak edges are removed to disconnect neighboring core groups which happen to reside within r pixels from each other. We fix the threshold θ at a small value of 0.1. Breadth-first or depth-first search is used to find connected components in the graph of 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 $|V_c| \ll |V|$ and $|E_c| \ll |E|$.

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. Therefore in this expanding step, we scan the M_t sequence in the backwards direction to go 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 .

```

 $n \leftarrow |V|$ 
 $S \leftarrow \{\text{cores of segments}\}$ 
 $L \leftarrow \{\}$ 
for  $t = \langle n, n-1, \dots, 2, 1 \rangle$ 
  if  $M_t$  is not a core pixel then
     $m1 \leftarrow \max_{C \in S} \text{average}_{i \in C, i \notin L, w_{M_i} > 0} w_{M_i}$ 
     $s \leftarrow \text{argmax}_{C \in S} \text{average}_{i \in C, i \notin L, w_{M_i} > 0} w_{M_i}$ 
     $m2 \leftarrow \max_{C \in S} \text{average}_{i \in C, i \notin L, w_{M_i} > 0} w_{M_i}$ 
    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, max2 is a function for determining the second maximum value of a set. The similarity between a pixel M_t and a segment $C \in S$ is computed by $\text{average}_{i \in C, i \notin L, w_{M_i} > 0} w_{M_i}$, where L is the set of low-confident pixels created with the condition $m2 > \lambda * m1$. We typically fix λ value at 0.5, so 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, a new segment will be created to accommodate it. This situation may occur if the values of parameters δ and β are so high 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 step is $O(|E|)$.

3.2 Experimental Study for Determining Parameter Settings

Fig. 1 (a) is a grayscale 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. So this number of cores matches the number of regions in the original image. Expanding these cores gives us the expected segmentation as in Fig. 1 (e) where different segments are shown by different intensities. 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 [4] as in Fig. 5. Here, along with the original images, we show 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.

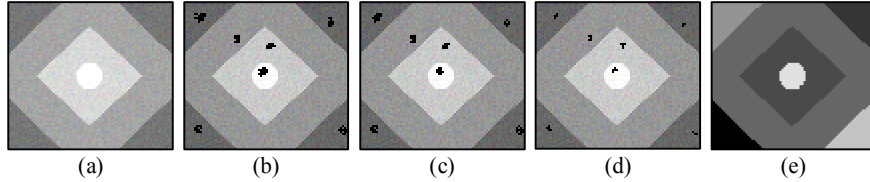


Fig. 1. (a) is a grayscale image. (b), (c), and (d) show core pixels when $\beta = 3$, $\delta = 90\%$, 96% , and 99% , respectively. The segmentation result (e) is the same for any settings

Since the number of nodes is finite, there are a limited number of possible settings for β and δ . Parameter β is an integer with the role of eliminating noisy pixels in the set of pixels of high decreasing rates of D values. Usually we set β value to 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 δ on the set of core pixels. The cores of segments shrink if we increase δ , and they are enlarged if δ is decreased. In general, we set δ to a value greater than 90% because for noisy images, setting δ too low may produce core pixels which are not very reliable. For images that contain a mixture of segments with different sizes, by increasing or decreasing δ , we can obtain a coarse or fine-grained segmentation. In Fig. 2, (a) shows a grayscale 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 demonstrates the point that core pixels are arranged in an order such that the pixels having higher rates of decrease in their D values belong to the cores of stronger segments.



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

4 Comparisons with the Normalized Cut Method [2]

Spectral clustering in general and normalized cut method in particular are well known graph clustering approaches [2, 6, 7]. Application of normalized cut to image segmentation has been shown in [2] and an implementation of this method developed by its authors is available on the web at www.cis.upenn.edu/~jshi/software. The method basic idea is to partition a graph $G = (V, E, W)$ into k subgraphs A_1, A_2, \dots, A_k based on the minimum k -way normalized cut, which is defined by:

$$Ncut_k = \sum_{i=1}^k \frac{\sum_{u \in A_i, v \in V - A_i} W_{uv}}{\sum_{u \in A_i, v \in V} W_{uv}}. \quad (5)$$

Finding exact minimum normalized cut is NP-hard, 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 it is true of many clustering methods, the number of clusters k has to be pre-specified. This is problematic for image segmentation because the number of segments is highly variable depending on the scene in images. The number of segments of 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 (5) make the method favor cutting the graph into clusters of similar sizes. As a result, small clusters are omitted very easily while large clusters are often split up into smaller parts. In

Fig. 3, (b) shows the segmentation of the image in (a) by a two-way normalized cut. The cut partitions the image into two segments similar in size. It fails to find the central oval as one segment because of the disparity between the number of pixels inside and outside the oval. In Fig. 4, (g), (h), (i), (j), (k), and (l) show the segmentations for image (a) by normalized cuts with $k = 2, 3, 4, 5, 6$ and 7 , respectively. Clearly, for any k , the image is partitioned into segments of roughly similar sizes. Moreover, varying parameter k dramatically changes the segmentation result.

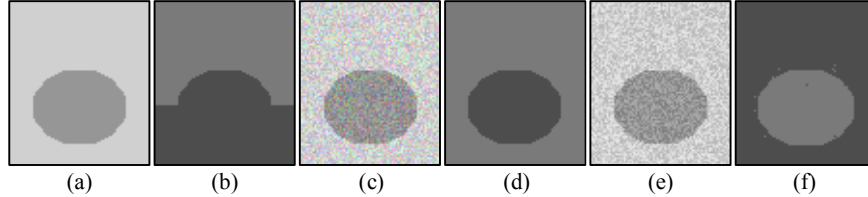


Fig. 3. (b) shows the segmentation result by the normalized cut method on the image in (a). Images (c) and (e) are created by adding different kinds of noise to (a). (d) and (f) show the segmentations by our method on (c) and (e), respectively

Based on the same proximity graphs, our algorithm produces better results. In Fig. 4, (b), (c), (d), (e), and (f) are our segmentations for image (a) with $\beta = 3$ and $\delta = 99\%$, 97% , 95% , 93% , and 90% , respectively. Note that lowering δ yields finer segmentation and 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 runs much faster than the normalized cut method on the same proximity graphs. In Tab. 1, we show the 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. The running times of 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 $105 \cdot 10^3$ nodes and $18 \cdot 10^6$ edges, the normalized cut method fails to execute because of an ‘Out of memory’ error. In addition, an 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 having to re-compute the cut from scratch.

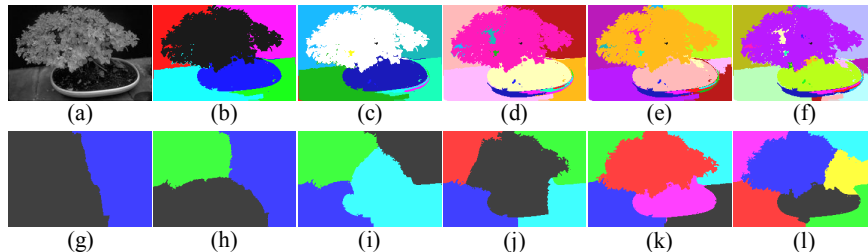


Fig. 4. Effects of parameters on segmentation results. (a) is a grayscale image from the Berkeley dataset. (b), (c), (d), (e), and (f) show segmentations by our method with $\delta = 99\%$, 97% , 95% , 93% , and 90% , respectively. (g), (h), (i), (j), (k), and (l) show segmentations of the normalized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

5 Conclusions

We have developed and evaluated a graph-based method for image segmentation. Based on proximity graphs, we partition images into segments of nearby pixels with similar intensities or colors. The method is simple and very fast. Because of exploiting the pixels in cores of segments which are generally stable and reliable, the method is robust to noise. As in Fig. 3, by adding noise to (a) we obtain images (c) and

(e), the segmentations of these noisy images shown in (d) and (f) remain stable. In addition to speed and robustness, an advantage is that parameters can be adjusted to yield a range of results from a coarse to fine-grained segmentation. The current weight functions (1) and (2) do not take into account texture information. It is possible to apply the method to texture segmentation by additionally incorporating statistical texture information around each image pixel into the weight functions. This should help refine the segmentation results for images that have texture.

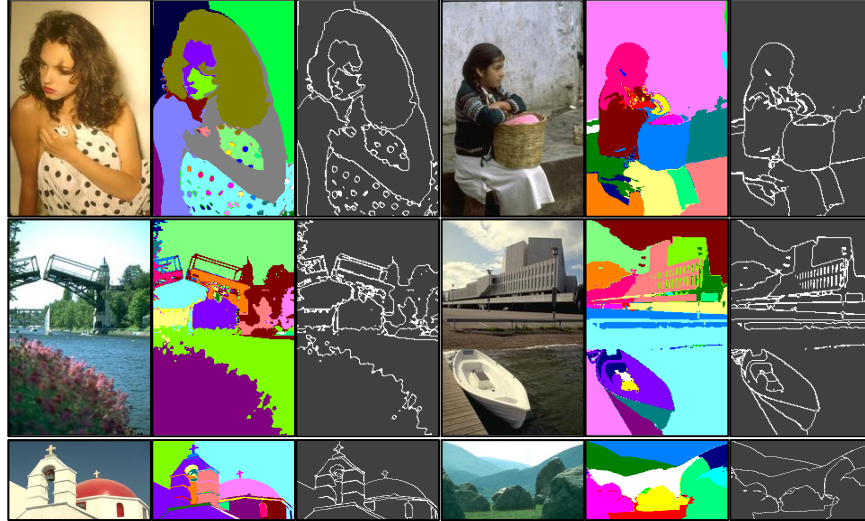


Fig. 5. Segmentations and derived boundaries on images from the Berkeley dataset. The same parameter settings $\beta = 3$, $\delta = 97\%$ are used for all the images

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

Proximity graph sizes		Normalized cut (seconds)	Coring method (seconds)
#nodes	#edges		
$30 \cdot 10^3$	$4.7 \cdot 10^6$	9	0.5
$45 \cdot 10^3$	$7.1 \cdot 10^6$	16	0.7
$60 \cdot 10^3$	$10.7 \cdot 10^6$	30	1.0
$75 \cdot 10^3$	$12.2 \cdot 10^6$	60	1.2
$90 \cdot 10^3$	$14.8 \cdot 10^6$	106	1.4
$105 \cdot 10^3$	$17.6 \cdot 10^6$	146	1.6
$120 \cdot 10^3$	$18.7 \cdot 10^6$	n/a	1.8
$500 \cdot 10^3$	$50 \cdot 10^6$	n/a	5.2
$1000 \cdot 10^3$	$70 \cdot 10^6$	n/a	7.5

References

1. Le, T., Kulikowski, C., Muchnik, I.: Coring method for clustering a graph. Proceedings of the 19th International Conference on Pattern Recognition (2008)
2. Shi, J., Malik, J.: Normalized cuts and image segmentation. IEEE Trans. on Pattern Analysis and Machine Intelligence (2000) 888-905
3. 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
4. Berkeley segmentation dataset, <http://www.cs.berkeley.edu/projects/vision/bsds/>
5. 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. of the 8th International Conference on Computer Vision (2001) 416-423
6. Luxburg, U.: A tutorial on spectral clustering. Statistics and Computing (2007) 395-416
7. 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