INTERACTIVE OBJECT SELECTION USING S-T MINIMUM CUT

Ning Xu and Narendra Ahuja

Department of Electrical and Computer Engineering and Beckman Institute
University of Illinois at Urbana-Champaign, Urbana, IL61801, USA

ABSTRACT

In this paper, we present an interactive object selection approach that applies $s-t$ minimum cut on local contour neighborhoods of the object boundary to segment the object from the background. Using a simple interface, users can extract accurate object contours interactively. Our approach has the following properties. 1) Interactivity: The interface is easy to use and accepts flexible freehand inputs. Realtime feedback is provided for user interactive correction. 2) Robustness: The $s-t$ minimum cut is insensitive to image noise since it finds the global minimum within the local contour neighborhood. Further, the approach is valid for contours and objects defined by different features, e.g., color and texture. 3) Controllability: Object contours are fully controllable through interactive correction by the user. Experimental results for a wide range of objects are provided.

1. INTRODUCTION

Object selection is the task of extracting an object from a digital image, e.g., for image/video editing and composition. Manual selection of objects is tedious, time consuming and inaccurate. Therefore, computer vision techniques assisted interactive tools are useful. Three properties are desired in interactive object selection tools. 1) Interactivity: The interface should be easy to use and should provide real time feedback. 2) Robustness: The tool should be able to select objects with complex characteristics, e.g., textured objects and in the presence of noise. 3) Controllability: The user should be able to override the contours extracted by the tool, force them to go through specific image parts/points, and if necessary, supply some parts of the contours manually.

During the last decade, many interactive object selection tools have been developed. The Image Snapping technique [1] assists users by detecting high-contrast edges in the vicinity of user-indicated locations, thus relieving users of the need to trace over the exact boundaries. It is implemented by using gradient descent on blurred versions of feature maps made from the images. The Intelligent Scissors tool [2] uses a dynamic programming based algorithm to search for the optimal contours connecting user specified control points. It generates realtime feedback as the user inputs control points, making it especially suitable for interactive object selection. The dynamic programming algorithm [3, 4] is also utilized in active contour models [5] to extract the globally optimal contour within the neighborhood of the initial contour provided by the user. The ICE method [6] first finds reliable image contours and encodes their location, orientation, blur and asymptotic intensity on both sides. Then the user interactive selects and groups the contours to extract objects. A tool described in [7] uses freehand sketches as input. It first segments the whole image and then selection is made by associating the input sketch with the best matching union of segments. Points, surrounding strokes and rough circumscribing loops are accepted as input sketches. All the approaches mentioned above do not consider the textured objects, which are very common in general images.

In this paper, we present an interactive object selection algorithm, having all three of the desired properties. 1) Our approach accepts flexible freehand inputs, such as control points along an intended contour or strokes along the desired object boundary. 2) Our approach uses the $s-t$ minimum cut to find the optimal contour within a certain neighborhood of the input sketch. The use of the $s-t$ minimum cut has the following advantages. First, graph cuts make our method insensitive to noise since they find the global minimum within the local region. Second, graph cuts make it possible to consider different cues, such as color and texture, within a single framework [8]. We use both color and texture features in our approach and develop an automatic scale selection algorithm. 3) Our approach is well suited for interactive correction by the user. If necessary, the user can even draw a part of the contour manually using the same interactive tool. In addition, since we consider only local regions specified by the user input, our approach is able to provide realtime feedback for user interaction.

As a global minimization method, graph cut has been applied in image segmentation problems [9, 10, 11]. Among these graph cut based segmentation approaches, the method proposed in [11] is most related to our approach. It uses $s-t$ minimum cut to segment an image into two parts with the user interactively identifying the object and background regions. The whole image is represented as a graph where the $s-t$ minimum cut is computed. This might lead to two main disadvantages if object selection is considered to be the application. First, by segmenting the entire image into two parts, the method implicitly assumes that there are only two modes within the input image, which is not true for most of the real images. Thus, the method might require many efforts to identify the object and background regions. Second, segmenting the entire image makes it impossible for realtime feedback. In our proposed approach, we only use $s-t$ minimum cut in a neighborhood area of a local object boundary. This small neighborhood area is reasonable to be modelled as a two-mode region and doing $s-t$ minimum cut within this area ensures realtime feedback. In addition, our proposed approach has much better interactivity and controllability, and considers both color and texture features.

In the next section, we describe our proposed approach in detail. Section 3 presents some experimental results obtained using our algorithm. Section 4 presents concluding remarks.

2. INTERACTIVE OBJECT SELECTION USING $S-T$ MINIMUM CUT

2.1. Overview

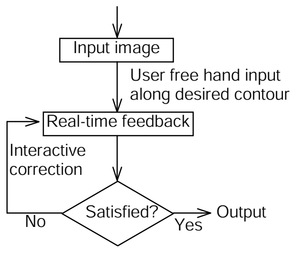

Fig. 1. An overview of our interactive approach.

Fig. 1 presents an overview of our approach. The user interactive correction is carried out iteratively until the result is satisfactory. Given an image with an embedded object to be selected, our approach accepts two kinds of freehand inputs: control points and strokes, as shown in Fig. 2. Two input control points or the two endpoints of a stroke specify a neighborhood around and containing a local segment of the desired object boundary (Fig. 2). The width of the neighborhood, shown hatched in Fig. 2, is under user control. Within the neighborhood, our algorithm finds the optimal contour using the $s-t$ minimum cut. The resulting contours are provided as realtime feedback to the user for interactive and iterative correction.

Fig. 3 presents a simple example illustrating the procedures of selecting an object. In this example, the user inputs two control points, P_1 and P_2, and a stroke $\widehat{P_3P_4}$. The user specifies the width of the neighborhood between

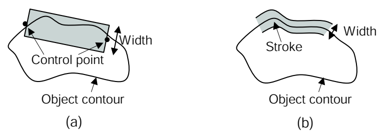

Fig. 2. Two kinds of input. (a) Control points. Two consecutive control points specify a neighborhood around and containing a local segment of the desired contour. The local contour neighborhood is a rectangular window defined by the two control points. (b) Strokes. The stroke width is adjustable so that the user can draw the stroke to specify a neighborhood similar in (a).

P_1 and P_2, and uses default width for the neighborhood defined by the stroke and the regions between the two control points and the stroke, i.e. P_2P_3 andP_4P_1. The two control points and the two endpoints of the stroke separate the entire contour into four parts: P_1P_2, P_2P_3, P_3P_4 and P_4P_1. Our algorithm extract these four parts of object contour successively, as shown in Fig. 4(a)-(h). The obtained contour and the updated control/end points are shown in Fig. 4(h) for user interactive correction. The user can make correction by adding a control point (Fig. 4(i)), or by drawing a stroke (Fig. 4(j)).

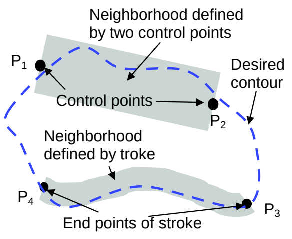

Fig. 3. The user inputs two control points, P_1 and P_2, and a stroke $\widehat{P_3P_4}$. The user specifies the width of the neighborhood between P_1 and P_2, and uses default width for the neighborhood defined by the stroke and the regions between the two control points and the stroke, i.e. P_2P_3 andP_4P_1. The two control points and the two endpoints separate the entire contour into four segments: P_1P_2, P_2P_3, P_3P_4 and P_4P_1.

2.2. Extracting object contour from the user specified neighborhood

The hatched area in Fig. 2 defines a local contour neighborhood. We now describe how to use $s-t$ minimum cut to find an optimal contour within the neighborhood.

2.2.1. Related graph theory

Graph-theoretic description of $s-t$ minimum cut can be found in many graph theory textbooks [12, 13]. The min-

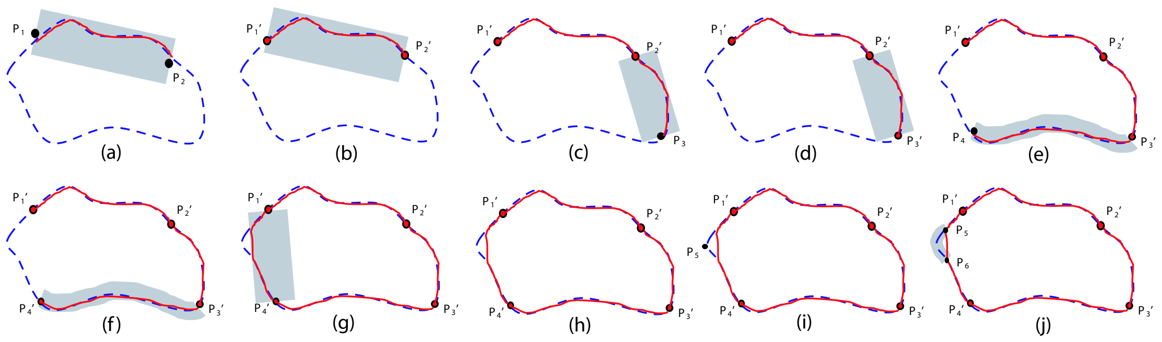

Fig. 4. Extracting an object contour step by step. (a) Obtain a contour within the neighborhood defined by P_1 and P_2. (b) Update control points P_1 and P_2 to P_1' and P_2', respectively. (c) Obtain a contour within the neighborhood defined by P_2' and P_3 (The neighborhood has a default width). (d) Update point P_3 to P_3'. (e) Obtain a contour between P_3' and P_4, within the neighborhood defined by the stroke. (f) Update point P_4 to P_4'. (g) Obtain a contour between P_4' and P_1'. (h) Obtained entire contour and updated control/end points are shown for user interactive correction. (i) The user adds a new control point P_5 to correct the wrong segment between P_4' and P_1'. (j) The user draw a stroke to correct the wrong segment between P_4' and P_1'. The two end points P_5 and P_6 separate the segment $P_4'P_1'$ into three shorter segments. The contour between P_5 and P_6 is to be updated by an optimal contour within the neighborhood defined by the stroke $\widehat{P_5P_6}$.

imum cut considered in this paper is required to separate multiple source nodes from multiple sink nodes. We will use a simple graph operation called *node identification* which merges or identifies a set of nodes $\{V_1, V_2, ..., V_n\}$ into a single new node V, deleting self loops, if any, and merging parallel edges. With the help of this, we can use $s-t$ minimum cut algorithms to solve the multi-source multi-sink minimum cut problem by simply identifying the multiple sources as a single source and multiple sinks as a single sink, respectively.

2.2.2. *Formulation of the problem*

First, the neighborhood in which the optimal contour is to be found has two forms: 1) rectangular windows (between two control points or between a control point and a stroke endpoint), and 2) strokes. We would like to point out that the neighborhood defined using a stroke is essentially the same as a neighborhood defined using two control points. The two end points of the stroke can be regarded as two control points, and the center line of the stroke can be transformed (virtually) into a straight line, Hence, the stroke itself is transformed into a rectangular area. Therefore, without loss of generality, we only consider rectangular areas defined by two control points in detail.

In our algorithm, the problem of extracting object contour within a specified contour neighborhood is formulated as a multi-source multi-sink minimum cut problem. We first represent the specified contour neighborhood, as an 8-connectivity graph $G(V, E)$, which means each vertex $v \in V$ (corresponding to a pixel p within the neighborhood) has edges connecting it with 8 other vertices (corresponding to the 8 neighboring pixels of the pixel p). In order to use the $s-t$ minimum cut algorithm to obtain an optimal contour, the following problems need to be solved: 1) how to define the multiple sources and the multiple sinks, and 2) how to assign the edge-weights in graph G so that multiple segmentation cues, for example, those based on color and texture, can be used.

2.2.3. *Defining sources and sinks*

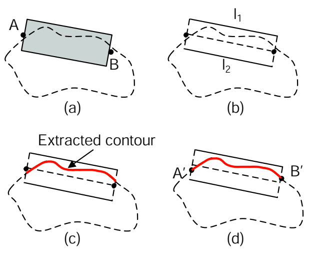

Fig. 5. Extracting contour between two control points. (a) A neighborhood is defined by two user specified control points A and B. (b) The desired contour lies between l_1 and l_2. (c) Extracted object contour within local neighborhood. (d) The end points are now updated to A' and B'.

A neighborhood specified by two control points is shown in Fig. 5(a). Since the desired contour lies within the rectangular window formed by edges l_1 and l_2, we can define the points on l_1 as sources and those on l_2 as sinks, as shown in Fig. 5(b). Then the $s-t$ minimum cut on the resulting graph will yield the optimal contour that separates l_1 and l_2 (Fig. 5(c). We update the control points by moving them onto the contour obtained by the algorithm, as shown in Fig. 5(d).

In practice, we sequentially process the consecutive local neighborhoods, as we have already seen in the example in Fig. 4. A problem there is that how to ensure that the obtained contour in 4(c) go through P_2', which has already been updated, so that the extracted contour segments can be connected seamlessly. To address this problem, the above strategy of defining the sources and sinks within a neighborhood defined by two control points is slightly modified. Three different situations are distinguished according to the characteristics of the two control points of the neighborhood. 1) Neither control points of the rectangle is updated before. This situation occurs only when we consider the first pair of control points, both control points have not been updated before and we will update both of them after we obtain the local contour segment. In this case, we identify one of the long edges as a single source s and the other long edge as a single sink t, as shown in Fig. 6(a). 2) One control point has been updated but the other has not. This situation occurs commonly after the first contour segment is obtained. By identifying sources and sinks as in Fig.6(b) (where the left control point is updated before), the resulting contour is forced to pass through the fixed control points, e.g., the left control point in Fig.6(b). After the optimal contour segment is extracted, the other control point (the right control point in Fig.6(b)) is updated. 3)Both ends have been updated before. This situation occurs only when we are extracting the last segment of the object boundary. In this case, the sources and sinks are defined as shown in Fig.6(c) and the resulting contour has to pass through both endpoints. Thus, the continuity of the final contour is guaranteed because the consecutive contour segments are linked together at the fixed control points.

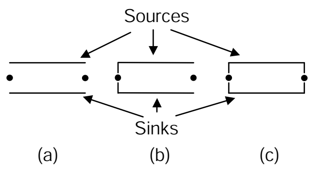

Fig. 6. Three different cases of extracting sources and sinks. (a)Neither, (b) one or (c) both of the two control points have been updated before.

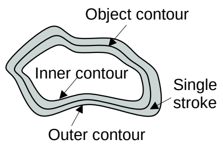

Fig. 7. Using a single stroke as input. The outer contour is identified as a single source and the inner contour is identified as a single sink.

The selection task can also be initialized using only one stroke as in Fig.7. In this case, the vertices corresponding to the outer contour are identified as a single source s and the vertices corresponding to the inner contour are identified as a single sink t.

2.2.4. Defining edge-weights

Edge-weight assignment is very important for graph theory based algorithms. The edge-weight between two vertices is assigned as a measure of similarity between the two corresponding pixels: the higher the edge-weight, the more similar they are. For example, the edge-weights can be assigned inversely proportional to the color differences between pixels. If the region of interest is defined by other criteria, e.g. texture, the edge-weights must be based on texture measures at the two pixels.

Texture based segmentation and specifically what texture features to use is a major question by itself. Here we will not address this problem but only discuss how a given texture feature is used for object contour selection. For this purpose, we will use a simply computed measure of texture scale. We will first estimate the texture scale in terms of gray level variance and then blur the texture of the estimated scale. We will then use this blurred image to assign the edge-weights. Generally, it is difficult to measure the texture scale because we do not know the extent of textured regions and their boundaries. However, in our case, the interactive input specified a neighborhood around the desired object contour, which is assumed to contain two different types of textures. (We will call one of them as the texture of the object, the other as the texture of the background.) Accordingly, we estimate the texture scales for each of the two support regions, i.e., the region around the source pixels and the region around the sink pixels as follows. (1). We apply a set of Gaussian filters with different variances: $\sigma_1, \sigma_2, ..., \sigma_N$ where $\sigma_1 = 0$ and $\sigma_i < \sigma_j$, if $i < j$, at each pixel within this support region. (Here $\sigma_1 = 0$, so the output of the filter is the same as input.) (2). We calculate variance V_i for the region after applying a Gaussian filter with σ_i. (3). If the variance V_1 is smaller than a threshold T_1, we set the scale as $s = 1$, otherwise, we set the scale s to be the smallest number i such that $V_{i-1} - V_i$ is smaller than another threshold T_2. Once we have estimated the scale value for each side, we select one of the two scales to blur the the neighborhood. If there is a considerable difference between the mean values of the two support regions, we select the larger scale because then the object color will be different from the background color after blurring. If the mean value is almost the same, we select the smaller scale because then blurring will homogenize the side with smaller scale while the other side will still have some texture.

As previously explained, from texture analysis perspective, considering only the texture scale and blurring are far from enough to obtain texture features. Many other techniques, for example, Textons [14], can be used for this purpose. However, we have not used elaborate schemes for this purpose for the following reasons. 1. We need provide realtime feedback so the computation time should be small. 2. Considering only the texture scale in our selection tool yields good results for a wide variety of textured objects. 3. Even for the rare cases such as the two regions having the same color distribution, where the above simple features will be most challenged, we could still extract a satisfactory object contour through interactive correction.

2.3. Realtime feedback and interactive correction

The set of control points (including the endpoints of strokes) as well as the estimated contour are shown as overlaps on the input image for user interactive correction. Before we discuss interactive correction, we will discuss how we control the computational complexity to ensure that realtime feedback is available to the user.

2.3.1. Control of execution time

The most time consuming part of our algorithm is the computation of the $s - t$ minimum cut, which is $O(n^3)$, where n is the number of vertices of the graph. Therefore, the algorithm slows down very quickly as the vertex count increases. To ensure realtime feedback, we limit the vertex number within each graph, i.e. the pixel count within each user specified neighborhood. In our implementation, whenever the neighborhood size exceeds a limit, we simply insert some new control points between the user supplied two end points to divide the neighborhood into several equal-length parts, each of which has a pixel count smaller than the limit.

2.3.2. Interactive correction

The contours extracted by the $s - t$ minimum cut is not guaranteed to be correct. Moreover, user should have the ultimate say about the selection. Therefore, a good interactive correction method is necessary for the user to extract the final contour.

Our algorithm accepts control points and strokes as input for user correction. To use control points as input, the user has two choices: 1) changing an existing control point, or 2) adding a new control point. If the user intends to change an existing control point, he or she clicks within a small neighborhood of the existing control point. Otherwise, to choose adding a new control point between two existing control points, the user clicks on the contour segment between the two control points. Then the user clicks near the desired new position, and this new point replaces the old control point, or becomes a new control point, based on the type of the first clicked point. After this second point is clicked, two neighboring rectangular windows with default widths are shown on the image. The user is able to adjust the widths, and the algorithm extracts the two new contour segments within the two windows. This time, the clicked new control point is assumed to be accurate such that the two new contour segment connected at this point. To use stroke as input to correct the estimated contour, the user first specifies the width of the stroke, then draws a stroke which starts and ends on the estimated contour. The start point and the end point of the stroke then become the new control points, and all the old intermediate control points are discarded. A contour within the new stroke is computed using the same method as before. After each correction, the newly obtained object contour and the new set of control points are shown on top of the input image for further correction, until the estimated contour is satisfactory.

Thus, the user has full control of the final contour. In the extreme case, the user can manually draw the entire object contour using a stroke with a small width.

3. EXPERIMENTAL RESULTS

First, we show some experimental results for extracting optimal contour within a local contour neighborhood defined by two control points. Fig. 8(a) and (b) show two complicated boundaries. Fig. 8(c) and (d) show some synthetic textured objects and background. Fig. 8(e) and (f) show some real images with textured objects.

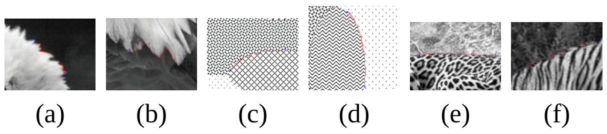

Fig. 8. Extracting optimal contour within a neighborhood defined by two control points. Control points are marked with blue cross, and the obtained contours are marked with red points.

Second, we show an example of using control points as initial input to extract object contour, and for interactive correction. Fig. 9(a) shows a set of initial control points. The selected object is shown in Fig. 9(b). The result is quite satisfactory except one of the feathers in the right corner is not included. Fig. 9(c) shows how the user sequentially inserts two new control points, one at a time, to correct this error. The final result is shown in Fig. 9(d).

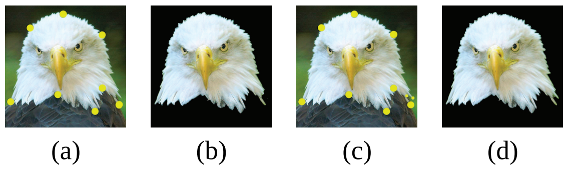

Fig. 9. (a) Using a set of control points as input. Control points are marked in yellow. (b) Result of (a). (c) User adds two more control points, which are marked with yellow points. (d) Final result after correction.

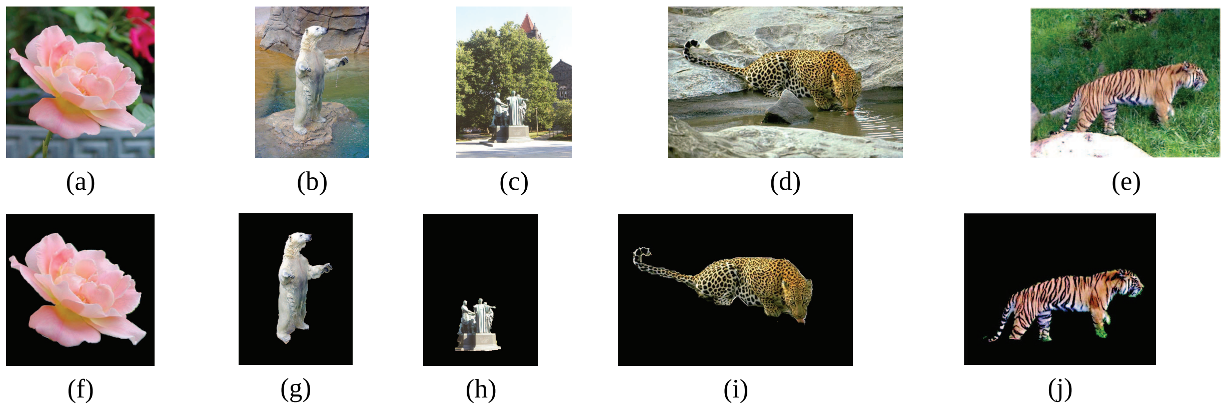

Fig. 10. Examples of selected objects. (a)-(e) are original images and (f)-(j) are corresponding results.

Finally, some other object selection results are shown in Fig. 10.

4. CONCLUSIONS AND DISCUSSION

In this paper, we present an $s - t$ minimum cut based interactive algorithm to extract objects from digital images. The user interface is simple and provides realtime feedback for interactive correction. The final object contours are fully controllable through user interactive correction. We also develop an automatic scale selection algorithm and apply it to define the graph edge-weights, yielding a tool capable of considering texture information.

Although our method is compared with others in section 1, we do not provide experimental comparisons. The reason is that it is difficult to set up a standard for comparing the performance of interactive object selection tools. Also, this is why there is no experimental comparison provided in those classic object selection papers [1, 2, 6, 7].

It would be useful to extend our selection method to include alpha channel estimates for objects with fuzzy boundaries. Other ongoing work is aimed at this extension without sacrificing the realtime nature of the computation.

5. REFERENCES

[1] Michael Gleicher, "Image snapping," *Proceedings of SIGGRAPH*, pp. 183–190, 1995.

[2] Eric N. Mortensen and William A. Barrett, "Intelligent scissors for image composition," *Proceedings of SIGGRAPH*, pp. 191–198, 1995.

[3] Amir Amini et al., "Using dynamic programming for solving variational problems in vision," *IEEE Trans. on PAMI*, vol. 12, no. 9, September 1990.

[4] Davi Geiger, Alok Gupta, Luiz A. Costa, and John Vlontzos, "Dynamic programming for detecting, tracking, and matching deformable contours," *IEEE Trans. on PAMI*, vol. 17, no. 3, March 1995.

[5] M. Kass, A. Witkin, and D. Terzopoulos, "Snakes: Active contour models," *International Journal of Computer Vision*, pp. 321–331, 1988.

[6] James H. Elder and Rick M. Goldberg, "Image editing in the contour domain," *Proceedings IEEE International Conference on Computer Vision and Pattern Recognition*, pp. 374–381, 1998.

[7] K. Tan and N. Ahuja, "Selecting objects with freehand sketches," *The 8th ICCV*, vol. 1, pp. 337–344, July 2001.

[8] Jitendra Malik, Serge Belongie, Thomas K. Leung, and Jianbo Shi, "Contour and texture analysis for image segmentation," *International Journal of Computer Vision*, vol. 43, no. 1, pp. 7–27, 2001.

[9] Zhenyu Wu and Richard Leahy, "An optimal graph theoretic approach to data clustering: Theory and its application to image segmentation," *IEEE Trans. on PAMI*, vol. 15, no. 11, pp. 1101–1113, 1993.

[10] Jianbo Shi and Jitendra Malik, "Normalized cuts and image segmentation," *IEEE CVPR*, June 1997.

[11] Y. Boykov and M. Jolly, "Interactive graph cuts for optimal boundary and region segmentation of objects in n-d images," *The 8th ICCV*, vol. 1, pp. 105–112, July 2001.

[12] Ravindra K. Ahuja, Thomas Magnanti, and James Orlin, *Network Flows: Theory, Algorithms and Applications*, Prentice Hall, 1993.

[13] Thomas Cormen, Charles Leiserson, and Ronald Rivest, *Introduction to Algorithms*, McGraw–Hill Companies, 1990.

[14] S. Zhu, C. Guo, Y. Wu, and Y. Wang, "What are textons?," *Proc. of 7th European Conf. on Computer Vision*, May-June 2002.