Available online at www.sciencedirect.com
ScienceDirect
Computer Vision and Image Understanding 107 (2007) 210–224
Computer Vision and Image Understanding
www.elsevier.com/locate/cviu

Object segmentation using graph cuts based active contours

Ning Xu [a,*], Narendra Ahuja [b], Ravi Bansal [c]

[a] DMS Lab, Samsung Information Systems America, 3345 Michelson Dr., Suite 250, Irvine, CA 92612, USA
[b] ECE Department, University of Illinois at Urbana-Champaign, Urbana, IL 61801, USA
[c] Department of Psychiatry, Columbia University, New York, NY, USA

Received 16 June 2005; accepted 16 November 2006
Available online 16 January 2007

Abstract

In this paper we present a graph cuts based active contours (GCBAC) approach to object segmentation. GCBAC approach is a combination of the iterative deformation idea of active contours and the optimization tool of graph cuts. It differs from traditional active contours in that it uses graph cuts to iteratively deform the contour and its cost function is defined as the summation of edge weights on the cut. The resulting contour at each iteration is the global optimum within a contour neighborhood (CN) of the previous result. Since this iterative algorithm is shown to converge, the final contour is the global optimum within its own CN. The use of contour neighborhood alleviates the well-known bias of the minimum cut in favor of a shorter boundary. GCBAC approach easily extends to the segmentation of three and higher dimensional objects, and is suitable for interactive correction. Experimental results on selected data sets and performance analysis are provided.

Keywords: Object segmentation; Active contours; Snakes; Graph cut

1. Introduction

A natural and popular formulation of object segmentation is in terms of energy minimization of certain objective functions. For example, the framework of active contours [1,2], also called Snakes, minimizes the contour energy E defined as the sum of external energy and internal energy. The external energy pulls the contours towards desired image features while the internal energy helps achieve smooth boundaries. An early implementation of active contours is based on deforming an initial contour at a number of control points selected along a given initial contour. The deformation is directed towards the object boundary by minimizing the energy E so that its local minimum occurs at the boundary of the object. This implementation has several disadvantages. First, the Snakes find the local minimum nearest to the initial contour and therefore are sensitive to initialization, e.g. when there are a large number of local minima near the initial contour due to image noise or background clutter. Second, the discretization of the contours using a number of control points may lead to problems of uneven spacing and self-crossing as the contours deform. This characteristic may significantly increase the complexity of extensions of the approach to segmentation of 3D objects. Finally, automatic selection of various parameters of the active contours such as the weights in the energy function is still an open problem.

Many approaches have been proposed to improve the robustness and stability of Snakes. In a balloon model, an inflation force is defined on active contours [2]. The active contours then are stopped by a strong edge but move past a spurious edge which is weak relative to the ambient inflation force. The GVF Snakes method introduces an external force called gradient vector flow (GVF) which is computed from diffusion of the gradient vectors of the image [3]. These two Snakes methods change their external forces to achieve a large capture range but they still find a

* Corresponding author.
E-mail address: ningxu01@gmail.com (N. Xu).

1077-3142/$ - see front matter © 2006 Elsevier Inc. All rights reserved.
doi:10.1016/j.cviu.2006.11.004

local minimum. Geodesic active contours [4] find a geodesic curve in a Riemannian space derived from image content. Implicit active shape models embed contours as the zero level set of a higher dimensional function and then solve a partial differential equation of motion [5–9]. These two approaches eliminate the problems with uneven control points and self-crossing and can easily be extended to higher dimensional applications. However, they still have the problem of local minima. Dynamic programming is used to extract the globally optimal contour within a certain region [10,11]. However, these methods do not scale properly from extracting contours to extracting surfaces and still have the self-crossing and uneven spacing problems.

Graph cuts approaches have been recently applied as global optimization methods to the problem of image segmentation [12–15]. The image is represented using an adjacency graph. Each vertex of the graph represents an image pixel, while the edge weight between two vertices represents the similarity between two corresponding pixels. Usually, the cost function to be minimized is the summation of the weights of the edges that are cut. The exact solution can be found in polynomial time. However, this solution has a bias towards cuts with short boundaries, resulting in small regions. The normalized cut approach [16] is aimed at reducing this bias by introducing a cost function called disassociation, but it introduces another bias, towards similar weight partition [17]. The ratio cut [17] normalizes the cost function by the length of the cut. However, the algorithm is slow for planar graphs and NP-hard for non-planar graphs. Interactive graph cuts [18] use *s–t* minimum cut as an optimization method with the user identifying the object and background regions interactively. This method assumes that the desired object contour is the global minimum within the whole image given the user input as constraints. Thus the result of this approach depends on the interactive input and can not break away from it. This method works well when the desired object contour satisfies the assumption; otherwise, the user has to interactively modify the constraints so that the assumption is satisfied and the desired contour is obtained.

In this paper, we present an approach called graph cuts based active contours (GCBAC). This paper is an extension to our previous paper [19]. Compared to the interactive graph cut method of [18], a much more practical assumption is made: that the desired object contour should be the global minimum within its own belt-shaped contour neighborhood (CN) of an *a priori* known size (width). Given an initial contour, the GCBAC algorithm iteratively replaces the contour with the global minimum within the contour's CN until the contour itself is the global minimum within its CN. We use a mechanism similar to the conventional active contours to deform the contour, and at each step of the iterative deformation, we use minimum cut as a global minimization tool to find the global minimum within the current CN, which is obtained by dilating the current contour by *a priori* known size. To define a cost function to be minimized using graph cut, the image within the CN is represented by a (pixel) adjacency graph with edge weights assigned based on the intensity information, and the problem of finding the global minimum contour within this CN is formulated as a multi-source multi-sink *s–t* minimum cut problem on this graph, by treating the pixels on the inner boundary as multiple sources and the pixels on the outer boundary as multiple sinks.

In the next section, we describe our proposed approach in detail. Section 3 presents experimental results and their comparison with the results obtained by other algorithms. Section 4 presents concluding remarks.

2. Our approach

2.1. Related graph theory

2.1.1. Multi-source multi-sink minimum cut

The related theory of graph cut can be found in many text books, e.g. [20–22]. The minimum cut of interest in this paper is required to separate multiple source vertices $\{s_1, s_2, \ldots, s_n\}$ from multiple sink vertices $\{t_1, t_2, \ldots, t_m\}$ with the smallest capacity. This multi-source multi-sink problem can be converted to an ordinary single source single sink *s–t* minimum cut problem. One current method for such conversion is to first add two additional vertices, a super source vertex s and a super sink vertex t, then add a directed edge (s, s_i) with capacity $c(s, s_i) = \infty$ for each $i = 1, 2, \ldots, n$ and add a directed edge (t_j, t) with $c(t_j, t) = \infty$ for each $j = 1, 2, \ldots, m$ [20]. For example, Fig. 1(a) shows a graph with two source vertices $\{s_1, s_2\}$ and three sink vertices $\{t_1, t_2, t_3\}$. Fig. 1(b) shows the converted graph with a single s and a single t. The minimum cut separating s from t in Fig. 1(b) corresponds to the minimum cut in Fig. 1(a) separating $\{s_1, s_2\}$ from $\{t_1, t_2, t_3\}$. However, the above method has the following shortcomings. (1) The converted graph has a larger number of vertices and edges than the original one. This will increase the running time when computing the minimum cut. (2) The use of infinite edge weight has an adverse impact on the time complexity of the excess scaling algorithm [20] that we have used in our implementation, since this algorithm has a running time of $O(nm + n^2 \log U)$, where n is the number of nodes, m is the number of edges, and U is the largest edge weight.

In this paper, we propose another method to convert a multi-source multi-sink min-cut problem to a single source single sink min-cut problem. This method, described in the following subsection, decreases the number of nodes and edges, and does not introduce new edges with infinite edge weight.

2.1.2. Node identification

We propose a simple operation called *node identification* to convert a graph of interest $G(V, E)$ to $G'(V', E')$, by identifying a set of nodes $(V_s = v_1, v_2, \ldots, v_n \subset V$ as a single new node $(v \in V')$. In the converted graph

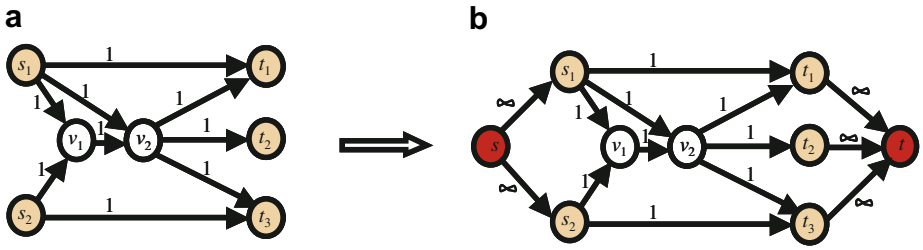

Fig. 1. Converting multi-source multi-sink graph into single source single sink graph. (a) A graph with source vertices s_1, s_2 and sink vertices t_1, t_2, t_3. (b) Converted graph with a super source s and super sink t.

$G'(V', E'), V' = V - V_s + v$ and $E' = E - E(V_s) + E_v$, where $E(V_s) = \cup_{u_1 \in V_s}$ or $u_2 \in V_s\{(u_1, u_2)\}$, and $E(v) = \cup_{u \in V} - V_s\{(u, v), (v, u)\}$ with capacity $c'(v, u) = \sum_i c(v_i, u)$ and $c'(u, v) = \sum_i c(u, v_i)$. Simply speaking, node identification identifies a set of nodes $\{v_1, v_2, \ldots, v_n\}$ as a single new node v, deleting self loops, if any, and merging parallel edges with cumulative capacity. An simple example of an undirected graph is shown in Fig. 2.

By using node identification, the graph in Fig. 1(a) can be converted as in Fig. 3. Obviously, the converted graph in Fig. 3(b) (with 4 vertices and 5 edges) is much simpler than the graph in Fig. 1(b) (with 9 vertices and 14 edges), and contains no infinite edge weight. Thus, the computation required for the minimum cut is eased, especially when the number of sources and sinks is large and there are a lot of edges linking vertices within the sources and the sinks.

In terms of *node identification*, we have the following Theorem for the multi-source multi-sink s–t minimum cut problem:

Theorem 1. (Multi-source Multi-sink min-cut). *The minimum cut $MC(G, S, T)$ of graph G which separates a source set $S = \{s_1, s_2, \ldots, s_n\}$ and a sink set $T = \{t_1, t_2, \ldots, t_m\}$ is exactly the s–t minimum cut $MC(G', s, t)$ of the graph G' that results after identifying $s_1, s_2, \ldots, s_n$ as a new source s and identifying $t_1, t_2, \ldots, t_m$ as a new sink t.*

Proof. For each cut $C(G, \hat{S}, \hat{T})$ in the original graph G that separates the vertices in G into $\hat{S}$ and $\hat{T}$, where $S \subseteq \hat{S}$ and $T \subseteq \hat{T}$, there is a one on one mapping to the cut $C(G', \tilde{S}, \tilde{T})$ in the graph G' that separates its vertices into $\tilde{S}$ and $\tilde{T}$, where $s \in \tilde{S}$ and $t \in \tilde{T}$, where $\tilde{S} = \hat{S} + \{s\} - S$, and $\tilde{T} = \hat{T} + \{t\} - T$.

The capacity of $C(G', \tilde{S}, \tilde{T}) = \sum_{u \in \tilde{S}} \sum_{v \in \tilde{T}} c(u, v)$

$$= \sum_{u \in \tilde{S} - \{s\}} \sum_{v \in \tilde{T} - \{t\}} c(u, v) + \sum_{u \in \tilde{S} - \{s\}} c(u, t) + \sum_{v \in \tilde{T} - \{t\}} c(s, v) + c(s, t),$$

and the capacity of the corresponding $C(G, \hat{S}, \hat{T}) = \sum_{u \in \hat{S}} \sum_{v \in \hat{T}} c(u, v)$

$$= \sum_{u \in \hat{S} - S} \sum_{v \in \hat{T} - T} c(u, v) + \sum_{u \in \hat{S} - S} \sum_{v \in T} c(u, v) + \sum_{v \in \hat{T} - t} \sum_{u \in S} c(u, v) + \sum_{u \in S} \sum_{v \in T} c(u, v),$$

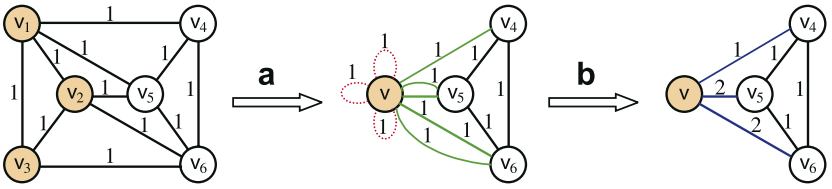

Fig. 2. An example of using node identification to convert an undirected graph. (a) v_1, v_2, v_3 are merged into a new node v. (b) Self loops are deleted and parallel edges are replaced by a single edge.

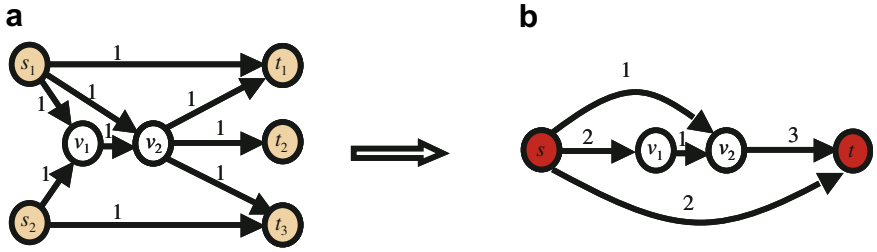

Fig. 3. Using node identification to convert the directed graph in Fig. 1(a). (a) Same graph as in Fig. 1(a). (b) Converted graph with single sources s and single sink t.

with $\tilde{S} - \{s\} = \hat{S} - S$, and $\tilde{T} - \{t\} = \hat{T} - T$.

According to the process of note identification, we have

$$\sum_{u\in\hat{S}-S}\left(\sum_{v\in T} c(u,v)\right) = \sum_{u\in\hat{S}-S} c(u,t),$$

$$\sum_{v\in\hat{T}-T}\left(\sum_{u\in S} c(u,v)\right) = \sum_{v\in\hat{T}-T} c(s,v),$$

$$\sum_{u\in S}\sum_{v\in T} c(u,v) = \sum_{u\in S} c(u,t) = c(s,t),$$

therefore the capacity of the corresponding $C(G,\hat{S},\hat{T})$

$$= \sum_{u\in\hat{S}-S}\sum_{v\in\hat{T}-T} c(u,v) + \sum_{u\in\hat{S}-S} c(u,t) + \sum_{v\in\hat{T}-T} c(s,v) + c(s,t)$$

$$= \sum_{u\in\tilde{S}-\{s\}}\sum_{v\in\tilde{T}-\{t\}} c(u,v) + \sum_{u\in\tilde{S}-\{s\}} c(u,t) + \sum_{v\in\tilde{T}-\{t\}} c(s,v) + c(s,t),$$

which is exactly the capacity of $C(G',\tilde{S},\tilde{T})$. □

With the help of this theorem, 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.

2.2. GCBAC algorithm for object segmentation

2.2.1. Object segmentation using graph cuts

The graph cuts theory discussed above provides us with a method to compute the globally optimal partition of an image by first transforming the image into an edge capacitated graph $G(V,E)$ and then compute the minimum cut. One such transformation is as follows. Each pixel within the image is mapped to a vertex $v \in V$. If two pixels are adjacent, there exists an undirected edge $(u,v) \in E$ between the corresponding vertices u and v. The edge weight $c(u,v)$ is assigned according to some measure of similarity between the two pixels: the higher the edge weight, the more similar they are. The minimum cut on the transformed edge capacitated graph will partition the graph into two parts with minimum capacity, i.e., the summation of the edge weights across the cut is minimized. Therefore, the corresponding contour in the image will partition the image into two segments with the minimum between-segment similarities.

Each contour that partitions the image into two parts S and T corresponds to a cut (S, T) on the graph. However, for object segmentation problem, the cut corresponding to a desired object contour is in general not the global minimum among all possible cuts on the graph (since, for example, the contour of another, smaller object in the same image might correspond to a cut with smaller capacity). As we stated in the previous section, a key assumption of our approach is that the desired segmentation contour is a global minimum within its *a priori* contour neighborhood (CN) of known width. Since there may be many such minima in the image, an initial contour is required to specify a desired segment. The objective then is to find the minimum closest to the specified initial contour. The GCBAC algorithm is designed to find the object contour satisfying the above key assumption automatically, given a proper initial contour. If the key assumption is not satisfied for some object contour, we allow the user to input some hard constraints so that the desired object contour becomes the constrained global minimum within its own CN.

In each automatic iteration step, as shown in Fig. 4, the CN of a contour may be obtained by morphological dilation, from which the inner and outer contours of the CN can be extracted. In the implementation of our method, if some part of the outer contour after dilation is out side of the image boundary, we substitute it with the corresponding part of the image boundary; and if the inner contour does not exist for the given dilation width, we keep reducing the dilation width until we can obtain an inner contour.

2.2.2. Overview of GCBAC algorithm

After the image I is represented as an edge-capacitated adjacency graph G and an initial contour c_0 is given, our algorithm consists of the following steps:

(0) Set the index of current step $i = 0$.
(1) Dilate current contour c_i into its contour neighborhood $CN(c_i)$ with an inner contour IC_i and an outer contour OC_i.
(2) Identify all the vertices corresponding to the inner contour as a single source s_i and identify all the vertices corresponding to the outer contour as a single sink t_i to obtain a new graph G_i.
(3) Compute the *s–t* minimum cut $MC(G_i, s_i, t_i)$ to obtain a new contour $c_{i+1} = \arg\min_{c\in CN(c_i)} E(c)$, where $E(c)$ =capacity of $MC(G_i, s_i, t_i)$.
(4) Terminate the algorithm if a resulting contour $\hat{c}$ reoccurs, otherwise set $i = i + 1$ and return to step 1.

Steps 1–4 of GCBAC algorithm are shown in Fig. 5.

2.3. Algorithm details and performance analysis

2.3.1. Connectivity and edge weights

When representing the image by an adjacency graph, either 4-neighborhood or 8-neighborhood can be used to define the pixel adjacency. Fig. 6 shows two contours connecting the top right horizontal edge and bottom left vertical edge of a 4×4 pixel homogeneous region. If we use 4-connectivity, then the two contours in Fig 6 (a) and (b) have the same capacity (=6). However, with 8-connectivity, the same two cuts (second row of Fig. 6) have different capacities, 15 and 11, respectively. As straight contour is preferred in homogenous region, we represent the image as an 8-connected graph G(V, E), which means each vertex $v\in V$ in G, corresponding to a pixel p, has edges connecting it to its 8 neighboring vertices, which correspond to the 8 neighboring pixels of p.

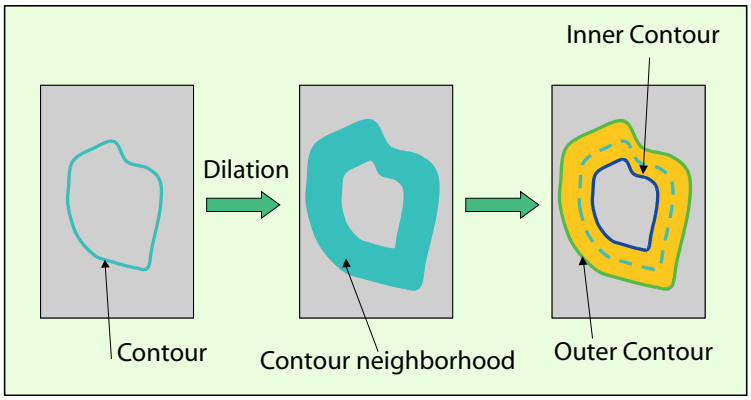

Fig. 4. The CN of a contour may be obtained by morphological dilation. The inner and outer contours of the CN are extracted and treated as the sources and sinks, respectively, in the corresponding graph.

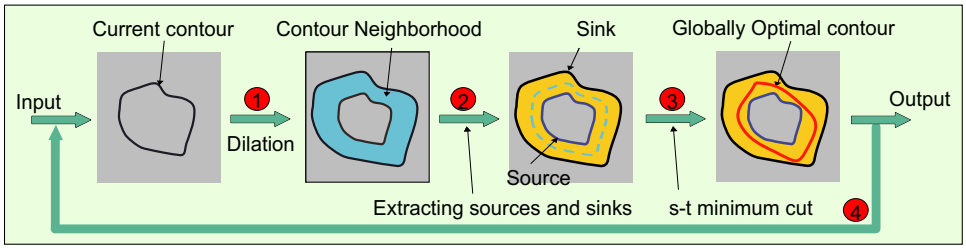

Fig. 5. A schematic of GCBAC algorithm. The input is an initial contour and the output is a segmentation boundary.

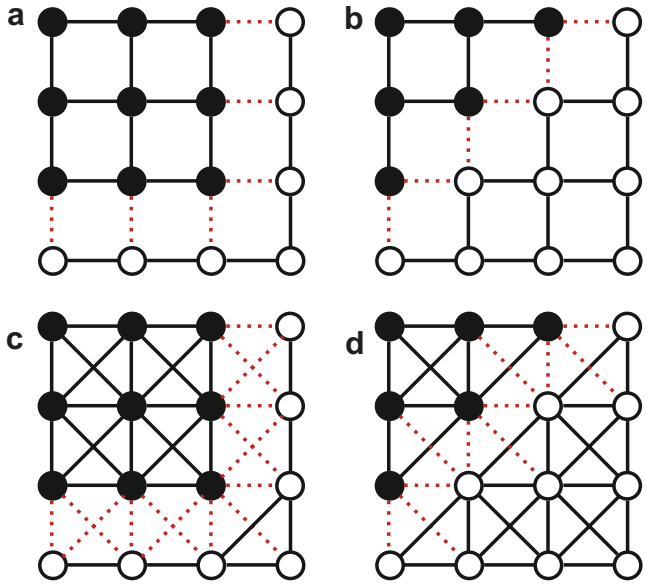

Fig. 6. Four-connectivity and 8-connectivity graphs. Rows correspond to a fixed connectivity and columns correspond to the same contour (cut). The connectivity is same in same row and the cuts are same in same column. The cuts are represented using dotted edges and separated nodes are represented using dark and bright circles.

Besides selecting the type of connectivity, the computation of edge weight which represents the similarity between two pixels in the image is also very important. Usually, the edge weight between two pixels i and j is assigned as $c(i,j) = \exp(-(I(i) - I(j))^2/2\sigma^2)$, where σ is a scale parameter. In addition, other information like edge, color, and texture can also be incorporated in the edge capacity of the graph. However, in this paper, for developing a general purpose method, we only consider the edge gradient information and in our implementation, we empirically choose $c(i,j) = (g(i,j) + g(j,i))^6$, where $g(i,j) = \exp(-\text{grad}_{ij}(i)/\max_k(\text{grad}_{ij}(k)))$, and $\text{grad}_{ij}(k)$ is the magnitude of image pixel intensity gradient at location k in the direction of $i \to j$. The maximum value of $\text{grad}_{ij}(k)$ in the entire image is used to normalize the gradient magnitudes. This weight assignment method leads the active contours to high gradient edges while taking into account directions of the gradients. Other information could be incorporated easily to the edge weight computation for specific applications. For example, if different parts of the object share the same color or texture property and so does the background, the color, texture or region information could be incorporated to improve the performance.

2.3.2. Dilation

The half width of CN, used as the dilation parameter and also referred to as step size in this paper, is selected based on two factors. First, the step size is chosen to be proportional to the size of the object—large for segmenting a large object and small for segmenting a small object. The second factor is the amount of noise in the image. The larger the amount of noise, the larger will be the density of local minima, and hence the greater the chance that a local minimum will be found for given step size. Therefore, the

more the noise, the larger the step size required to make the active contour insensitive to these local minima. If the image is not noisy, both small step size and large step size will work. A small step size will iteratively approach the minimum of interest whereas a larger step size will reach the desired minimum in fewer steps. However, if the step size is too large, the active contour may jump over the object contour of interest, to the contour of another object having smaller energy. Fig. 7 shows the tradeoff in selecting the step size. Fig. 7(a) shows clean data and Fig. 7(b) shows noisy data. In both figures, point A is the desired minimum point, and point B is another local minimum, having smaller energy than point A. If we select a small step size, then for the energy function in Fig. 7(a), the active contours will find point A for a large range of initializations; However, for the energy function in Fig. 7(b), the active contours will probably get stuck at one of those local minima. On the contrary, if we select a very big step size, point B will be found in both cases, which is not desired. In order to make our algorithm applicable to a variety of real and synthetic, 2D, and 3D images, the step size is left as an adjustable parameter for user to select manually according to the size of the desired object and the level of noise in the data. For specific classes of applications, e.g., the problem of lung nodule segmentation from CT images shown in Section 3, it is possible to develop an automatic CN size selection algorithm.

The dilation process in step 1 has several objectives. First, it allows the algorithm to jump over local minima within the CN, whereas the traditional active contours use a CN of just the immediate neighbors of the current contour, as a result of which they do not move away from local minima. Second, if the CN is a homogeneous area, the globally optimal contour found will be the shortest contour that contains the inner contour, as it will cut the least number of edges in the corresponding graph. This property is helpful when the background is simple and the initial contour is much larger than the real object contour in which case the active contours will shrink from the large initial contour to the desired object contour. Third, the dilation helps overcome the well-known bias of the minimum cut in favor of a shorter cut. If we perform minimum cut within a region of interest,the result will be biased toward a smaller segment and, in extremely cases, this would result in a singular point. The use of s–t minimum cut with the inner contour obtained from the dilation process acting as source nodes can prevent this from happening. This is because we regard the nodes corresponding to the extracted inner contour as the source nodes and after the cut, they are always contained in the S part of the resulting s–t minimum cut. Therefore, the resulting contour must be bigger than the inner contour which means the algorithm will never yield a singular point as a result as long as the CN contains an inner contour.

2.3.3. Convergence

The proposed approach is guaranteed to converge by the following theorem:

Theorem 2. (Convergence Theorem). *For a finite data set, the graph cuts based active contours will either converge or oscillate between a sequence of different contours of the same capacity, after a finite number of iterations.*

Proof. A finite data set yields a finite number of possible contours. We denote the total number of different contours within the data set as N. Then after $N+1$ iterations, at least one contour will appear twice. Let contour c_n be the first reoccurance of a previous contour c_m, i.e., $c_n = c_m$, and no contour appears twice before iteration n.

If $n = m+1$, we have $c_{n+1} = \arg\min_{c\in \mathrm{CN}(c_n)} E(c) = \arg\min_{c\in \mathrm{CN}(c_m)} E(c) = c_{m+1} = c_n = c_m$. Therefore from mathematical induction, the algorithm will converge to c_m.

If $n > m+1$, we still have $c_{n+1} = \arg\min_{c\in \mathrm{CN}(c_n)} E(c) = \arg\min_{c\in \mathrm{CN}(c_m)} E(c) = c_{m+1}$, and thus $c_{n+k} = \arg\min_{c\in \mathrm{CN}(c_{n+k-1})} E(c) = \arg\min_{c\in \mathrm{CN}(c_{m+k-1})} E(c) = c_{m+k}$, which means the sequence between c_m and c_n will oscillate.

For any $i \geqslant 0$, as $c_{i+1} = \arg\min_{c\in \mathrm{CN}(c_i)} E(c)$, and $c_i \in \mathrm{CN}(c_i)$, we have $E(c_{i+1}) \leqslant E(c_i)$. For the recurrence contour sequence c_i, $i = m, m+1, \ldots, n$, we have $E(c_m) \geqslant E(c_m+1) \geqslant \ldots \geqslant E(c_n)$. However, as $E(c_m) = E(c_n)$, we have $E(c_k) = E(c_m)$ for any $k, n \geqslant k > m$. Therefore, the recurrence contour sequence has the same capacity.

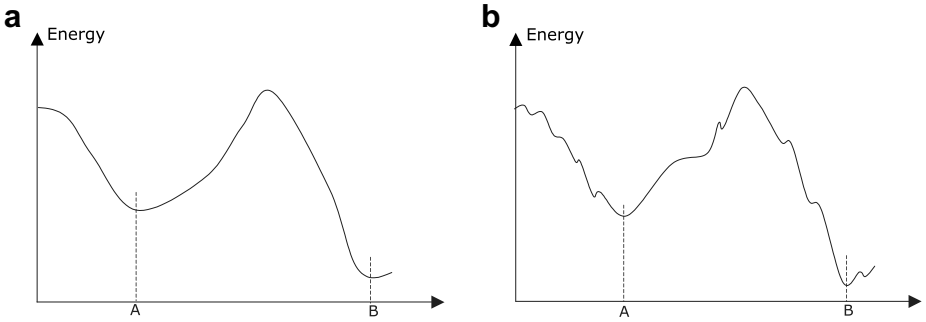

Fig. 7. Step size (CN width) selection. The energy function in (a) is much smoother than the energy function in (b). Point A is the desired minimum while point B is not desired even though it has a smaller energy value. Step size should be selected so as to avoid local minima but not miss the desired solution A in favor of a smaller-energy solution B.

From above, we know that the GCBAC algorithm will either converge or oscillate between a sequence of different contours of the same capacity. □

So if a contour $\hat{c}$ reoccurs, the algorithm terminates and give this $\hat{c}$ as output.

2.3.4. Initialization and interactive correction

Given a proper step size, as we discussed previously, the performance of our algorithm is quite insensitive to the position of the initial contour, compared to traditional active contours. This is because the algorithm has the ability to jump over the local minima within the contour neighborhood at each iteration and is designed to find a segmentation contour which is a global minimum within its own CN. If an object contour is a global minimum within its CN of size h, we can always put an initial contour within its CN of size $h/2$. Therefore, if we dilate the initial contour with a step size of $h/2$, the object contour will lie within the dilated region and thus be found by our algorithm in its first iteration. This is illustrated in Fig. 8 via minimization of a one-dimensional energy function, where point O is the desired global minimum within a neighborhood of radius h. Let $|OA| = |OB| = h$ and $|OC| = |OD| = h/2$. Then we can put the initial point anywhere between point C and D (in the shaded region), and set the step size to $h/2$, the desired minimum point O can be found in the first iteration.

Since in general there are multiple such global minima, e.g., corresponding to different object, in an image, the initial contour still needs to be close to the desired object contour so that the algorithm will not converge to an incorrect minimum.

The initial contour can be provided either by the user or by another algorithm. For example, in the lung nodule segmentation problem discussed in the experimental results section, we accept a polygon manually drawn by the user as the initial contour in the 2D case. For the 3D nodule segmentation problem, the user "blows-a-sphere" from a given location within the nodule to specify the initial "contour". Alternatively, an object detection algorithm might also be used to obtain the initial contours.

If the final contour is not satisfactory, the user can correct it interactively by specifying some control points, one at a time. In our implementation, the user can click a sequence of points that the boundary must pass through and the algorithm updates the source set and the sink set accordingly. Three situations are distinguished based on the user clicked point as follows:

1. The input point lies between the inner contour and the outer contour.
2. The input point lies inside the inner contour.
3. The input point lies outside the outer contour.

In each case, the algorithm updates the sources and sinks by stretching the inner contour and outer contour towards the user specified input points such that the input point is contained in a narrow gap between the stretched inner and outer contours. The contour resulting from this iteration is therefore forced to pass through the input point (Fig. 9). In each iteration thereafter, this input point would lie in the dilated CN and will be treated as an input point of situation 1. Therefore, the final contour found after all iterations will also pass through this input point.

2.3.5. Contour discontinuities and image noise

In some instances, the object boundary may be discontinuous and filling the discontinuities may be difficult. For example, Fig. 10(a) shows a elliptical object in front of a gray background and adjacent to another white object with a commonly shared boundary. Such a situation could arise in a lung CT image where the elliptical white object represents a small nodule attached on the lung wall, big white object. A similar situation arises in the subjective contour problems, e.g., for a broken triangle shown in Fig. 10(b). Our algorithm will compute such boundary discontinuities using a path of the smallest capacity, which in

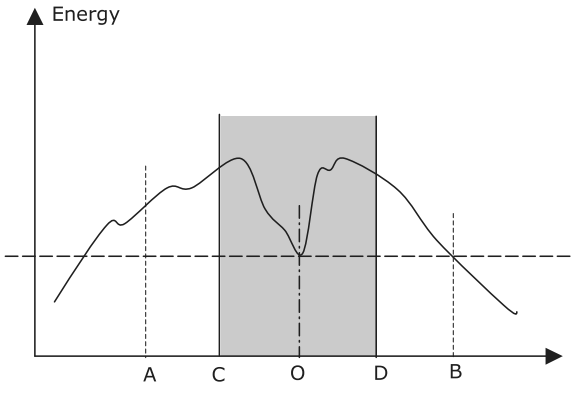

Fig. 8. The worst case of initialization in a one dimensional analogous. Point O is the desired minimum.

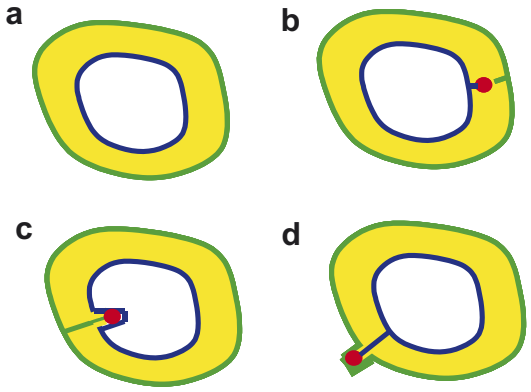

Fig. 9. Updating sources and sinks to incorporate an input point interactively supplied by the user (b–d) correspond to the three different locations of the point. The shaded region indicates contour neighborhood (CN), the dark curve indicates the sources and the lighter curve indicates sinks.

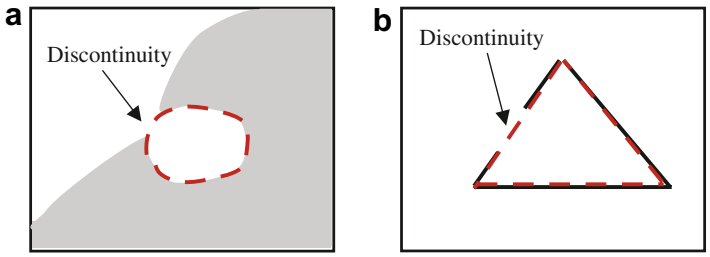

Fig. 10. Two examples of object boundary discontinuities: (a) an elliptical and (b) a triangle. The dashed contours show the desired object contour.

general is the shortest path across the discontinuity. The real object boundary across discontinuity might be different, but in the absence of any such information and using only the image colors, the smallest capacity (shortest path) is a reasonable way to complete the boundary.

However, if the desired shortest path across the discontinuity is not a minimum within its CN, our algorithm will miss it. For example, the desired object contour shown in Fig. 11(a) is not global minimum within its own CN; the contour shown in Fig. 11(b) has a smaller energy. Therefore, our algorithm will prefer shorter contours such as in Fig. 11(b), and will not converge to the desired object contour.

2.3.6. Extension to higher dimensional problems

Our GCBAC algorithm easily extends to three or higher dimensional segmentation problems. In three-dimensional problems, the 3D data can be represented by an edge capacitated adjacency graph with either 26-connectivity or 8 connectivity. Given an initial surface, we perform 3D dilation to obtain a local surface neighborhood with an inner surface and an outer surface. The vertices corresponding to these two surfaces are identified as a single source and a single sink, respectively. All the other steps of the GCBAC algorithm are exactly the same and the result is 3D object surface.

2.4. Comparison of performance with traditional active contours

The proposed GCBAC algorithm uses the optimization tool of graph cuts in each iteration to deform the active contours and differs fundamentally from traditional active contours. In the following subsections, we compare GCBAC with traditional active contours in terms of properties other than those properties of GCBAC mentioned above.

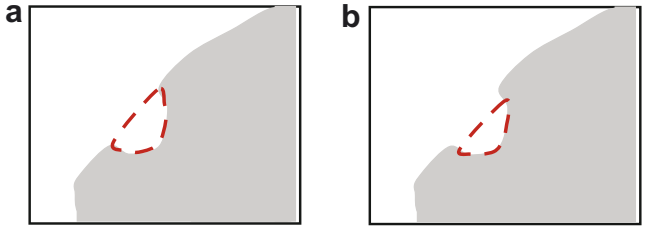

Fig. 11. The dashed contour in (a) shows the desired object contour. The dashed contour in (b) shows a contour with smaller energy as both the length of the contour across the discontinuity and the length of the contour along the desired object contour are shorter than those in (a).

2.4.1. Cost function

The proposed GCBAC approach has a much simpler energy function than the traditional active contours. A cost function often used by the traditional active contours is $E = E_{\text{internal}} + E_{\text{external}} = \alpha E_{\text{tension}} + \beta E_{\text{rigidity}} + E_{\text{ext}}$, where α and β are weighting parameters that control the curve's tension and rigidity, respectively. The internal energy is designed to hold the curve together and to keep it from bending excessively. This allows control of the object shape, but requires careful adjustment of the weights for different kinds of object boundaries, or even at different stages of deformation. In our approach, no internal energy is explicitly needed. The graph cuts guarantee the continuity of the resulting contour. The absence of explicit internal energy makes it easy to extend our algorithm to segment 3D and even higher-dimensional objects; analogous generalizations of the traditional active contours to higher dimensions are difficult.

The external energy in our approach is derived from the image and represented by edge weights on the corresponding graph. Through different assignments of edge weights, e.g., considering similarity of color and texture between pixels, or selecting different connectivities, e.g., constructing an edge between each pair of vertices, the cut based methods are able to take more image information into account than captured in the external energy in traditional active contours approaches.

2.4.2. Final contour

Since the final contour found by our algorithm is globally optimal within its own CN, it avoids local minima. On the other hand, the traditional active contours yield a locally optimal contour as minimum is found only in the immediate neighborhood. Also, since minimum cut is computed on a graph where each vertex corresponds to a pixel in the corresponding image, the resulting contour does not have problems of uneven spacing would occur if control points were used for representing the contour. Further, the final contour provided by the traditional active

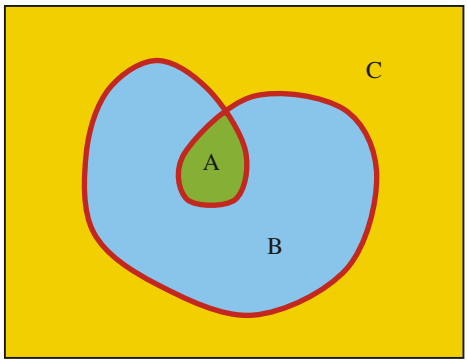

Fig. 12. The contours found by traditional active contour algorithms may have self-crossings. Here the result may be three segments (A, B, C) instead of the desired two.

contours algorithm may contain undesired self-crossings, (Fig. 12). This situation will not occur in our approach as the algorithm will partition the graph into exactly two parts defined by a minimum capacity cut.

3. Experimental results

In this section, we present our experimental results and compare them with some other object segmentation methods. Different aspects of the performance discussed in Section 2 are evaluated and validated in different subsections.

3.1. Insensitivity to initialization

As discussed in Section 2.3.4, our algorithm is not sensitive to the position of initial contour since the algorithm finds a minimum within its own contour neighborhood. Fig. 13 shows a synthetic image containing a gray, diamond-shaped object constructed to demonstrate the insensitivity of the algorithm to the initial contour. Different rows correspond to the use of different initial contours. The leftmost image in each row shows the initial contour and the rightmost image shows the final contour. The middle two images depict the intermediate stages. Fig. 14 shows an application to lung nodule segmentation. The images are extracted from a CT volume of a patient's lung. Four different initializations are shown as light contours, and the corresponding final contours are shown darker.

3.2. Discontinuities and noise

Fig. 15 demonstrates that the GCBAC algorithm is able to handle large edge discontinuities on object boundary as well as noise in the data, as discussed in Section 2.3.5. The left image in the first row of Fig. 15 shows a synthetic image of a triangle-shaped object with large gaps in the boundary. The left image in the second row shows an image with scattered noise points. The left image in the last row is corrupted by additive Gaussian noise. The rightmost image of each row shows the final contour. The middle two images depict the intermediate stages.

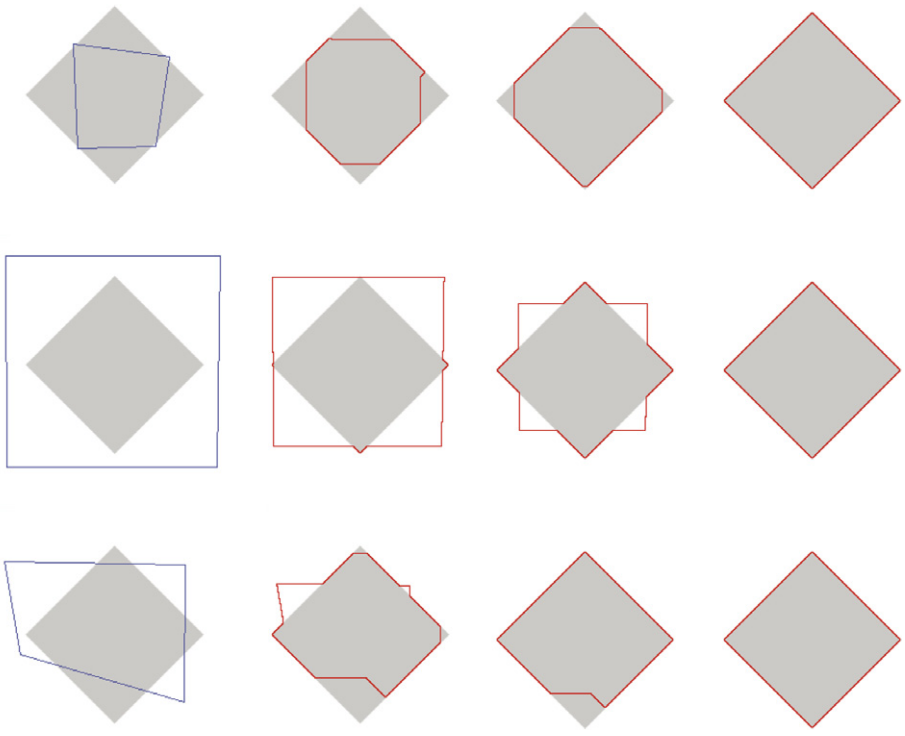

Fig. 13. Experimental results on synthetic images show that the proposed approach is insensitive to initial contours. The leftmost image in each row shows the initial contour and the rightmost image shows the final contour. The intervening images show how the active contour deforms.

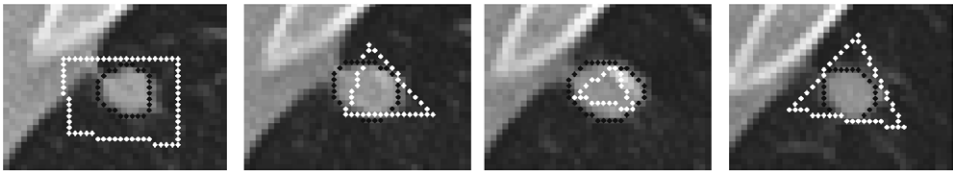

Fig. 14. GCBAC used for lung nodule segmentation. The light polygon in each image shows the initial contour and the darker contour shows the final contour.

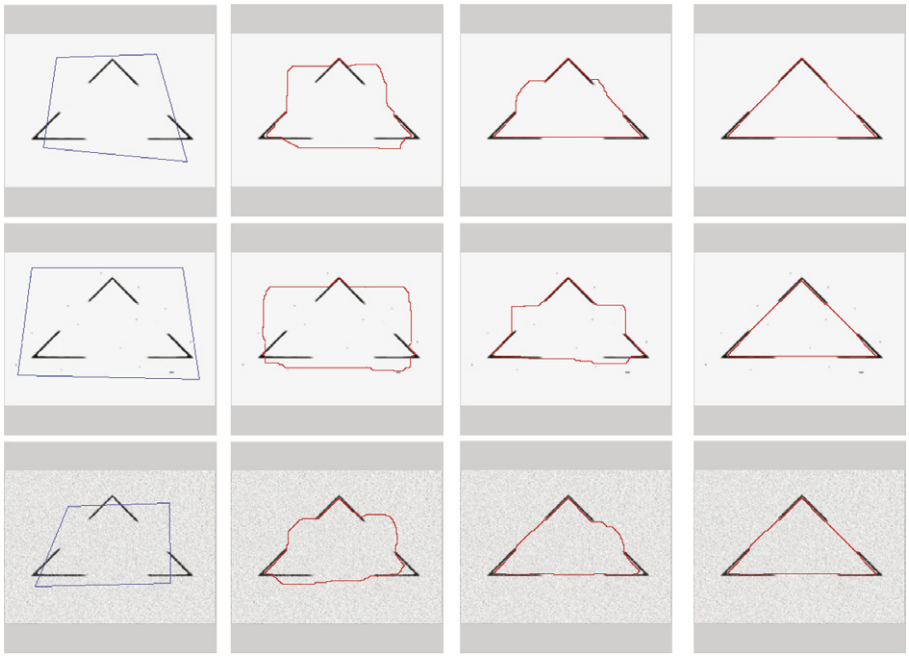

Fig. 15. Experiments with an image containing edge discontinuities (first row), isolated noise (middle row), and additive noise (last row). The leftmost column shows initial contours and the rightmost column shows final contours. The intervening columns show how the active contours deform.

Fig. 16 shows an application to lung nodule segmentation. The nodules have discontinuities on their boundaries because of the blood vessels. Initializations are shown in light contours, and the corresponding final contours are shown darker.

3.3. Higher dimensional problems

As discussed in Section 2.3.6, our GCBAC algorithm easily extends to three or higher dimensional problems. Fig. 17 shows the application of GCBAC to 3D objects. A synthetic object is shown in Fig. 17(a). The initial boundary is a sphere as shown in Fig. 17(b). Fig. 17(c) through (e) show intermediate results. The final surface is shown in Fig. 17(f).

Fig. 18 shows the application of GCBAC to 3D lung nodule segmentation in a CT volume data set. The nodule is attached to a few blood vessels. Our initialization here is a sphere near the object boundary. This sphere is blown from a point inside the nodule and is bounded by the nodule. The estimated surface is shown slice by slice in

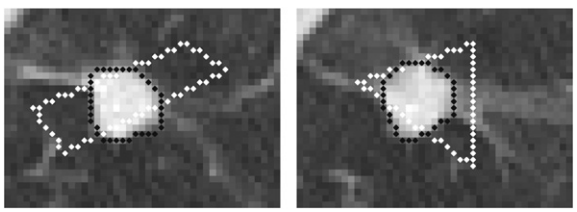

Fig. 16. Application to segmentation of lung nodules having boundaries with discontinuities. The light polygon in each image shows the initial contour and the final contour is shown darker.

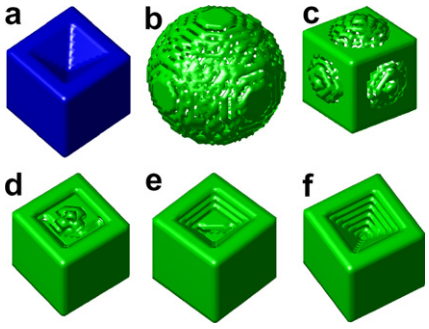

Fig. 17. Experimental results for a synthetic 3D object. (a) 3D object. (b) Initial surface. (c–e) Resulting intermediate surfaces. (f) Final result.

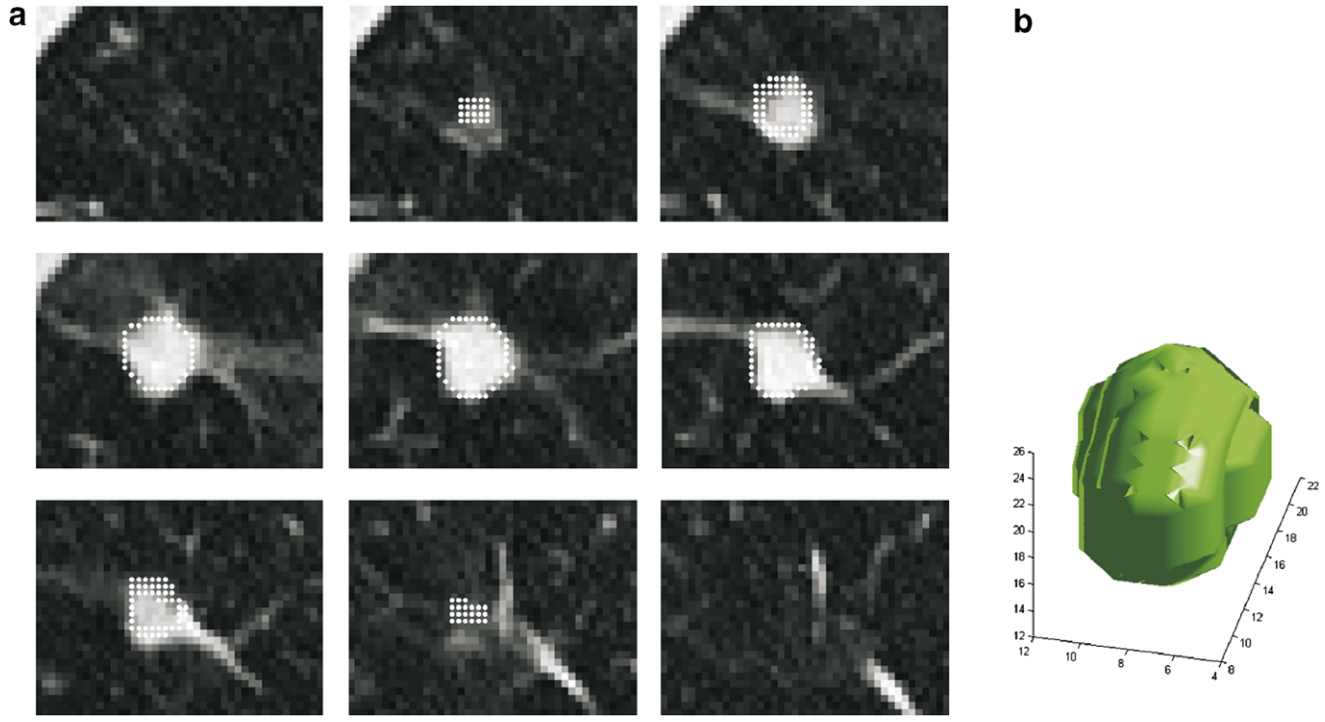

Fig. 18. Segmentation of a 3D nodule in 3D CT volume data. (a) Estimated surface of the nodule shown slice by slice to validate the result. (b) Rendered 3D nodule surface.

Fig. 18(a) and the reconstructed surface of the lung nodule is shown in Fig. 18(b).

3.4. Interactive corrections

If the final contour is not satisfactory, the user can correct it interactively by inputting some control points, one at a time. The first row of Fig. 19 shows the interactive correction procedure applied to a triangle with gaps in its boundary and an initial contour of Fig. 19(a). The resulting contour without correction is shown in Fig. 19(b). However, this is not the desired result. By accepting a user clicked point, marked with circle in Fig. 19(c), our algorithm continues and obtains a satisfactory result shown in Fig. 19(d). Fig. 19(e)–(h) show our results on a real image. Fig. 19(e) is the original image with an initial boundary. Fig. 19(f) shows the resulting boundary without correction. Since the resulting pepper contour is not satisfactory, the user clicks two points one after another, marked as small circles, to guide the active contours to the desired results. Fig. 19(g) shows the result after first correction and Fig. 19(h) shows the final result after both corrections.

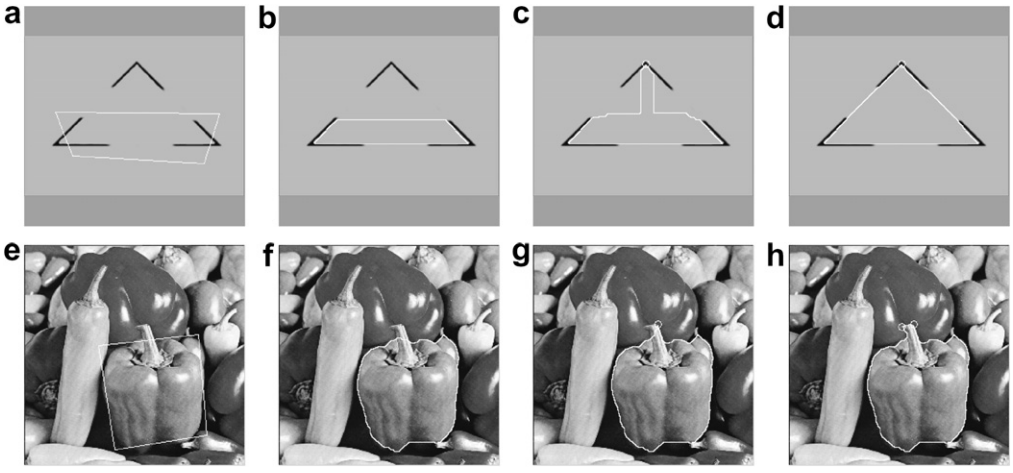

Fig. 19. Interactive corrections. The leftmost image in each row shows the initial contour. The circles are the correction points clicked by the user to guide the deformation of the active contours. The rightmost image shows the final results.

3.5. Changes in contour topology

Our algorithm is capable of changing the topology of the initial contour during deformation. A simple example is shown in Fig. 20. Changing topology is desired in some instances but not in others [23].

3.6. Comparisons with GVF Snakes and Interactive Graph Cuts

We compare our results with those obtained using the traditional active contours, specifically, the GVF Snakes, which have a large capture range and can move into boundary concavities. We use the GVF implementation available at http://iacl.ece.jhu.edu/projects/gvf/. We use the following parameters, suggested by the author of the GVF Snakes: $\mu = 0.1$, iteration count $= 200$ (for computing the gradient vector field), $\alpha = 0.05$, $\beta = 0$, $\gamma = 1$, $\kappa = 0.6$, $D_{\min} = 2$, and $D_{\max} = 4$. Fig. 21 shows that the GVF Snakes are prone to local optimum problem. The two images are the same as the images shown in the last two rows of Fig. 13.

We also compare our method with GVF Snakes and Interactive Graph Cuts [18] for the object segmentation problem in real images. We still use the above mentioned parameters for GVF Snakes. Regarding the interactive graph cuts approach, in order to make a fair comparison, we use the same way to assign edge weights and use a similar initialization. We dilate the initial contour so that the desired object contour lies within the dilated area, and mark the pixels on the inner and outer contours as the object and background pixels. Figs. 22–24 show the object segmentation results of the GVF Snakes, the Interactive Graph Cuts, and our approach, respectively, for three objects. Note that the initial contour provided for GVF Snakes as shown in the top row of Fig. 24(a) is very accurate. Also note that given the object and background pixels (as shown in the top row of the middle column of each figure), the global minimization results (as shown in the bottom row of the middle column of each figure) are not corresponding to the desired object contour. The Interactive Graph Cuts approach requires more interactive steps to achieve the desired results. However, as seen in the GCBAC results, these desired object contours are the global minima within their own contour neighborhoods, and are automatically obtained given the initial contours.

3.7. Running time analysis

Several polynomial-time algorithms for graph cut are described and compared in [22]. An experimental comparison of several different max-flow min-cut algorithms is provided in [24]. Based on the running time comparison and the comments in [22], we implemented the excess scaling preflow-push algorithm to solve the s–t minimum cut problem. Without using sophisticated data structures, the algorithm achieves the running time of $O(nm + n^2 \log U)$, where n is the number of nodes, m is the number of edges, and U is the largest edge weight. However, the simple topology of the graph used in GCBAC makes the algorithm run much faster. We performed the minimum cut on graphs having different numbers of nodes after node identification. Fig. 25 shows the observed running time. By obtaining the best polynomial fit to the observed data,

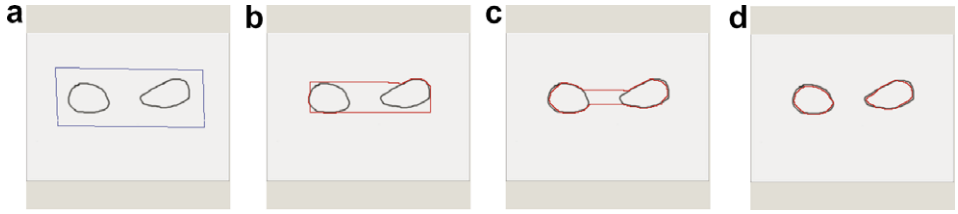

Fig. 20. A simple example of topology changes in the active contour during deformation. (a) Original image and initial contour. (b, c) Intermediate results. (d) Final contour with a different topology.

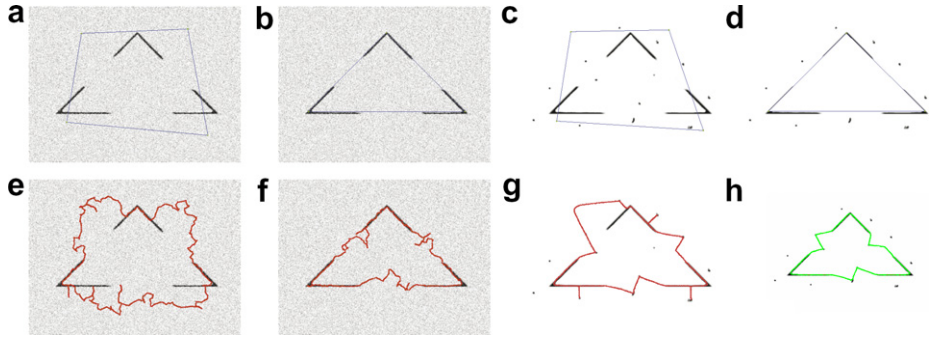

Fig. 21. Results of GVF Snakes. The first row shows the initial contours and the second row shows their corresponding final results.

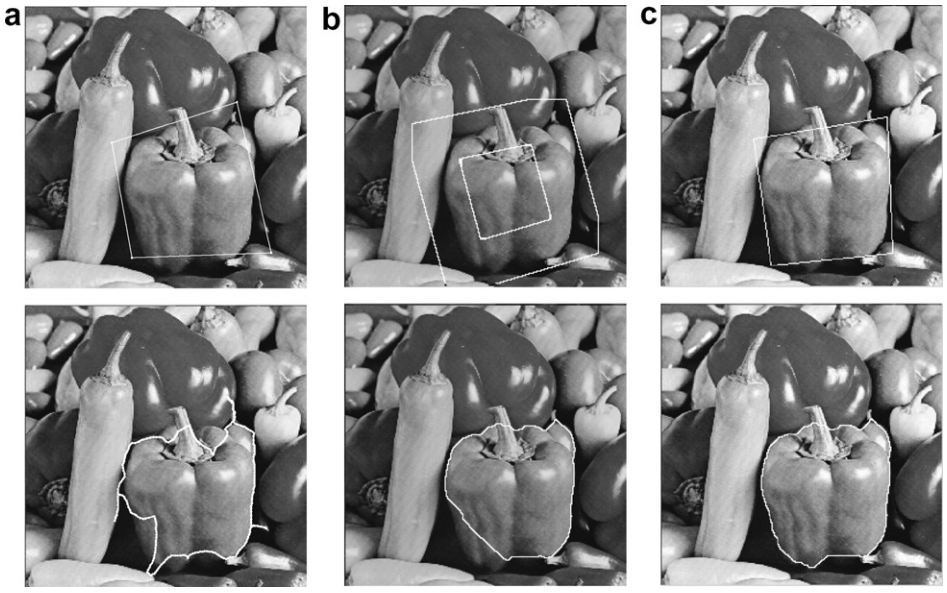

Fig. 22. Comparison with GVF Snakes and Interactive Graph Cuts for pepper image. From left to right, the three images in the top row show the initialization for GVF Snakes, Interactive Graph Cuts, and GCBAC, respectively. The three images in the bottom row show the corresponding results of the three methods.

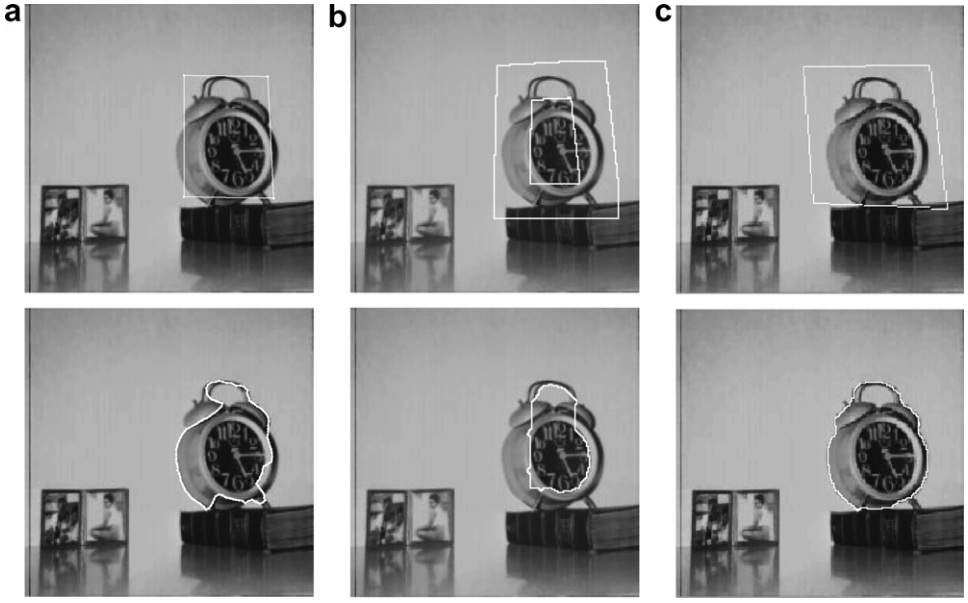

Fig. 23. Comparison with GVF Snakes and Interactive Graph Cuts for clock image. From left to right, the three images in the top row show the initialization for GVF Snakes, Interactive Graph Cuts, and GCBAC, respectively. The three images in the bottom row show the corresponding results of the three methods.

we estimate that the minimum cut algorithm used in GCBAC has running time of $O(n^{1.2})$. In Fig. 25, the running time is shown as a function of the number n of nodes.

3.8. Demonstration

We implemented our GCBAC approach using mixture code of Matlab and C, which compiles in windows machines. A demonstration and the associated source code are available online at http://vision.ai.uiuc.edu/~ningxu/projects/gcbac.html. A screen snap of our demo program is shown in Fig. 26.

4. Conclusions and discussion

We have presented a graph cuts based active contours (GCBAC) approach for object segmentation problem. Given an initial contour, the problem of finding the desired segmentation contour is formulated as that of finding the closest contour that is a global minimum within its contour

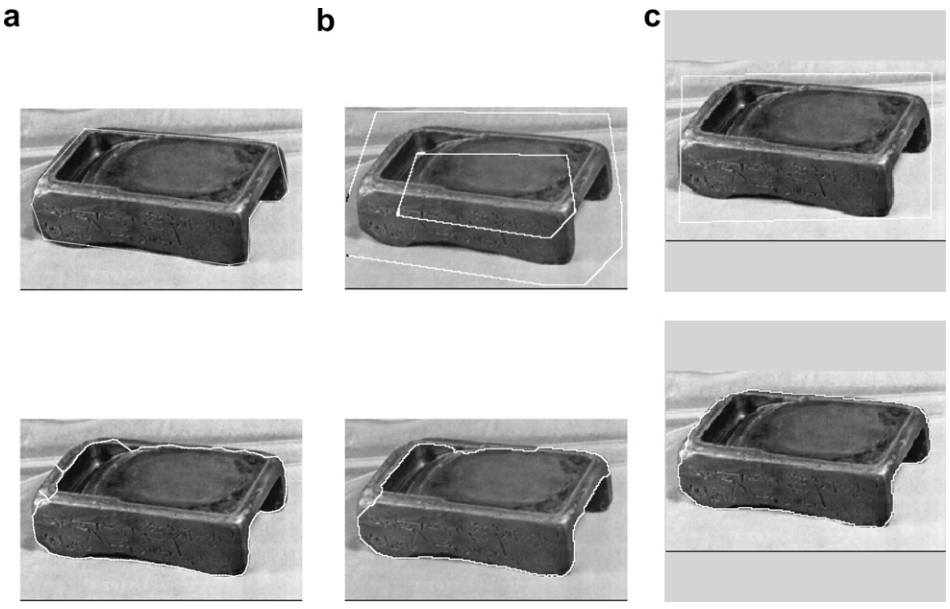

Fig. 24. Comparison with GVF Snakes and Interactive Graph Cuts for pepper image. From left to right, the three images in the top row show the initialization for GVF Snakes, Interactive Graph Cuts, and GCBAC, respectively. The three images in the bottom row show the corresponding results of the three methods. Note that the initial contour for GVF in the third row is very accurate.

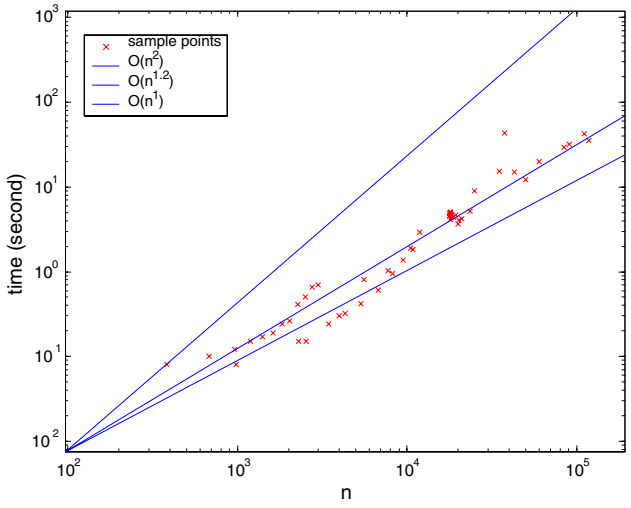

Fig. 25. The observed running time of the minimum cut algorithm used in our approach is $O(n^{1.2})$. The performance levels of $O(n)$ and $O(n^2)$ are shown, respectively, as references.

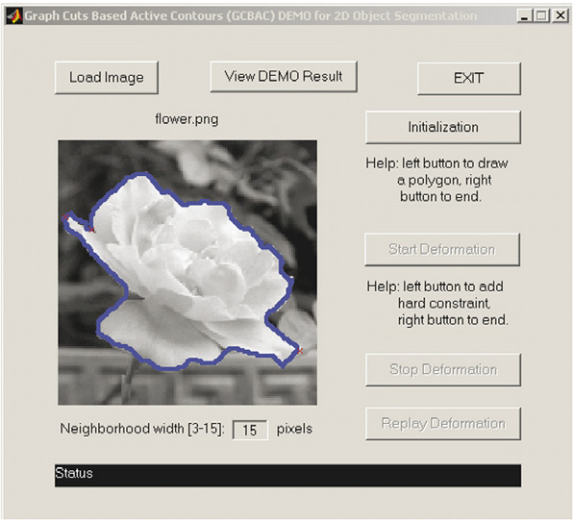

Fig. 26. Screen snap of the demo program.

neighborhood (CN). To use GCBAC algorithm, we need to construct the graph with appropriate pixel connectivity and edge weights. An example of the need for appropriate topology selection is illustrated by the fact that the 8-connectivity graph may result in smoother curves than the 4-connectivity graph, but the 8-connectivity graph still has limitations in homogeneous image areas. It will be interesting to devise a strategy for constructing the graph such that in homogeneous areas a cut of shorter length has a strictly smaller capacity. Usually similarity of pixel intensities is used to assign edge weight; however, in our implementation, we compute edge weight using gradient information. In the future, we could also incorporate texture information to segment textured objects by using similarity of texture between pixels to assign edge weight. Other future problems include how to extend our algorithm to segment multiple objects, and how to determine when our algorithm should allow changes in topology.

Acknowledgments

This work is partly supported by National Science Foundation under Grant ECS-0225523, and is partly completed at Siemens Corporate Research, Inc. Their support is gratefully acknowledged.

References

[1] M. Kass, A. Witkin, D. Terzopoulos, Snakes: active contour models, International Journal of Computer Vision 1 (4) (1988) 321–331.

[2] L.D. Cohen, On active contour models and balloons, computer vision, graphics, Image Processing: Image Understanding 53 (2) (1991) 211–218.

[3] C. Xu, J. Prince, Snakes, shapes, and gradient vector flow, IEEE Transaction on Image Processing 7 (3) (1998) 359–369.

[4] V. Caselles, R. Kimmel, G. Sapiro, Geodesic active contours, in: Proceedings of IEEE International Conference on Computer Vision, 1995, pp. 694–699.

[5] S. Osher, J. Sethian, Fronts propagating with curvature dependent speed: algorithms based on Hamilton–Jacobi formulations, Journal of Computational Physics 79 (1988) 12–49.

[6] R. Malladi, J.A. Sethian, B.C. Vemuri, Shape modeling with front propagation: a level set approach, IEEE Transactions on Pattern Analysis and Machine Intelligence 17 (2) (1995) 158–175.

[7] J. Sethian, Level Set Methods and Fast Marching Methods, Cambridge University Press, Cambridge, UK, 1999.

[8] O. Faugeras, R. Keriven, Variational principles, surface evolution, PDE's, level set methods, and the stereo problem, IEEE Transactions on Image Processing 7 (3) (1998) 336–344.

[9] N. Paragios, O. Mellina-Gottardo, and V. Ramesh, Gradient vector flow fast geodesic active contours, in: Proceedings of IEEE International Conference on Computer Vision, vol. 1, July 2001, pp. 67–73.

[10] A. Amini, T. Weymouth, R. Jain, Using dynamic programming for solving variational problems in vision, IEEE Transactions on Pattern Analysis and Machine Intelligence 12 (9) (1990) 855–867.

[11] D. Geiger, A. Gupta, L.A. Costa, J. Vlontzos, Dynamic programming for detecting, tracking, and matching deformable contours, IEEE Transactions on Pattern Analysis and Machine Intelligence 17 (3) (1995) 294–302.

[12] Z. Wu, R. Leahy, An optimal graph theoretic approach to data clustering: theory and its application to image segmentation, IEEE Transactions on Pattern Analysis and Machine Intelligence 15 (11) (1993) 1101–1113.

[13] H. Ishikawa, D. Geiger, Segmentation by grouping junctions, in: Proceedings of IEEE International Conference on Computer Vision and Pattern Recognition, June 1998, pp. 125–131.

[14] D. Forsyth, J. Ponce, Computer Vision: A Modern Approach. Upper Saddle River, Prentice-Hall, New Jersey, 2002.

[15] V. Kolmogorov, R. Zabih, What energy functions can be minimized via graph cuts? in: Proceedings of European Conference on Computer Vision, 2002, pp. 65–81.

[16] J. Shi, J. Malik, Normalized cuts and image segmentation, in: Proceedings of IEEE International Conference on Computer Vision and Pattern Recognition, June 1997, pp. 731–737.

[17] S. Wang, J. Siskind, Image segmentation with ratio cut, IEEE Transactions on Pattern Analysis and Machine Intelligence 25 (6) (2003) 675–690.

[18] Y. Boykov, M. Jolly, Interactive graph cuts for optimal boundary and region segmentation of objects in n-d images, in: Proceedings of IEEE International Conference on Computer Vision, vol. 1, July 2001, pp. 105–112.

[19] N. Xu, R. Bansal, N. Ahuja, Object segmentation using graph cuts based active contours, in: IEEE International Conference on Computer Vision and Pattern Recognition, June 2003, pp. 46–53.

[20] T. Cormen, C. Leiserson, R. Rivest, Introduction to Algorithms, McGraw–Hill Companies, 1990.

[21] L. Ford, D. Fulkerson, Flows in Networks Princeton, Princeton University Press, New Jersey, 1962.

[22] R.K. Ahuja, T. Magnanti, J. Orlin, Network Flows: Theory, Algorithms and Applications Upper Saddle River, Prentice Hall, New Jersey, 1993.

[23] X. Han, C. Xu, J. Prince, A topology preserving deformable model using level sets, in: Proceedings of IEEE Conference on Computer Vision and Pattern Recognition, vol. 2, December 2001, pp. 765–770.

[24] Y. Boykov, V. Kolmogorov, An experimental comparison of min-cut/max-flow algorithms for energy minimization in vision, in: Proceedings of International Workshop on Energy Minimization Methods in Computer Vision and Pattern Recognition, September 2001, pp. 359–374.