Fast, Quality, Segmentation of Large Volumes – Isoperimetric Distance Trees

Leo Grady

Siemens Corporate Research,
Department of Imaging and Visualization,
755 College Rd. East,
Princeton, NJ 08540
Leo.Grady@siemens.com

Abstract. For many medical segmentation tasks, the contrast along most of the boundary of the target object is high, allowing simple thresholding or region growing approaches to provide nearly sufficient solutions for the task. However, the regions recovered by these techniques frequently leak through bottlenecks in which the contrast is low or nonexistent. We propose a new approach based on a novel speed-up of the isoperimetric algorithm [1] that can solve the problem of leaks through a bottleneck. The speed enhancement converts the isoperimetric segmentation algorithm to a fast, linear-time computation by using a tree representation as the underlying graph instead of a standard lattice structure. In this paper, we show how to create an appropriate tree substrate for the segmentation problem and how to use this structure to perform a linear-time computation of the isoperimetric algorithm. This approach is shown to overcome common problems with watershed-based techniques and to provide fast, high-quality results on large datasets.

1 Introduction

Modern medical datasets are often so large that only the most simple, efficient segmentation algorithms may be employed to obtain results in a reasonable amount of time. Consequently, the present body of sophisticated, global segmentation algorithms are typically unsuitable in this context. In practice, thresholding, region growing [2] and watershed [3] algorithms appear to be the only approaches that are feasible under these circumstances.

Often, especially with CT data, simple intensity thresholding (or region growing) is almost sufficient to segment the entire object. However, the problem frequently occurs that the thresholded object is weakly connected to (i.e., touching) another object of equal intensity, leading to the common "leaking" problem associated with region growing. Therefore, an important problem for medical image segmentation is the fast segmentation of a mask into constituent parts via bottleneck detection within the mask. We will refer to this problem as the **mask segmentation** problem, where the mask is assumed to have been given by a simple thresholding or region growing process. Watershed algorithms, based either

A. Leonardis, H. Bischof, and A. Pinz (Eds.): ECCV 2006, Part III, LNCS 3953, pp. 449–462, 2006.

on intensity or the distance transform of the mask, are usually the algorithm of choice for breaking the mask into desired parts. Although this approach can be successful, watershed algorithms have a few common problems: 1) Small amounts of noise in the mask may lead to an overabundance of watershed regions (requiring a subsequent merging procedure), 2) Two objects may be included inside a single watershed region, 3) No measure of segmentation quality is included in the algorithm.

The recent isoperimetric algorithm for graph partitioning [1] has been successfully applied to image segmentation [4] and is specifically designed to use global information to cut a graph (mask) at bottlenecks, while remaining robust to noise, requiring only foreground seeds for initialization and offering a measure of partition quality. Although the isoperimetric algorithm is efficient enough for images or small volumes, ultimately requiring solution to a sparse linear system of equations, the size of medical volumes demands a faster approach. In this paper we show that the linear system associated with the isoperimetric algorithm may be solved in low-constant linear time if the underlying graph is a tree, propose an easily-computable tree from input data (which we call a **distance tree**), and show that our use of the isoperimetric algorithm with distance trees provides a fast, global, high-quality segmentation algorithm that correctly handles situations in which a watershed algorithm fails. For the remainder of this work, we shall refer to the approach of applying the isoperimetric graph partitioning algorithm to the mask-derived distance tree as the IDT (Isoperimetric Distance Tree) algorithm.

This paper is organized as follows: Section 2 gives context for the present algorithm by reviewing previous work. Section 3 recalls the isoperimetric algorithm for graph partitioning, shows how an underlying tree offers a linear-time solution, introduces the distance tree concept and summarizes the IDT algorithm. Section 4 compares the present algorithm to watersheds on several illustrative examples and provides results and runtimes for the IDT algorithm on real-world medical data. Section 5 draws conclusions and outlines future work.

2 Previous Work

The prior literature on segmentation is extremely large. Additionally, the special properties of trees have resulted in their use in many different contexts. Here, we attempt to review only those most relevant previous works in the context of segmentation on large datasets.

Level sets have attracted recent interest in the computer vision literature for general-purpose image segmentation [5]. However, in the context of segmenting large medical volumes, recent approaches still range from minutes to hours [6]. Furthermore, levels set techniques have not, to our knowledge, been applied to our present problem of mask segmentation.

For the task of mask segmentation, there is really only one option that is currently employed on a full resolution mask: the watershed algorithm [3]. Despite the speed of watershed approaches, there are several common problems, as

outlined in the previous section. In Section 4 we show that the proposed IDT algorithm does not suffer from these problems, which maintaining the speed of a watershed approach.

Minimal/maximal spanning trees have seen extensive use in the computer vision literature since as early as Zahn [7] and Urquhart [8]. Despite the speed of these techniques, they are often insufficient for producing high-quality segmentations of a weakly-connected graph, as illustrated by the authors themselves. Although some papers have used gradient-based minimal spanning trees for segmentation [9] and others have used distance map-based maximal spanning trees for centerline extraction [10] the use of such a tree as the setting for computing a linear-time isoperimetric segmentation is novel. We note that the intense computations associated with finding solutions to Markov Random Fields have also led to approximations defined on trees (Bethe trees) instead of a full lattice [11]. Additionally, the use of quadtrees is ubiquitous in split-and-merge segmentation techniques [12] but, unlike the present approach, the algebraic or topological properties of the tree are typically not of any particular significance.

The recently-developed isoperimetric method of graph partitioning [1] has demonstrated that quality partitions of a graph may be determined quickly and that the partitions are stable with respect to small changes in the graph (mask). Additionally, the same method was also applied to image segmentation, showing quality results [4]. Other methods of graph partitioning have gained prominence in the computer vision literature, most notably the normalized cuts algorithm [13], max-flow/min-cut [14] and the random walker algorithm [15]. However, each of these algorithms is far too computationally expensive to be applied on the full medical image volume, even after thresholding a mask. Additionally, using a tree as the underlying graph is not suitable in any of these algorithms, since normalized cuts would still require an expensive eigenvector computation (albeit somewhat faster on a tree [16]), max-flow/min-cut will cut at the weakest edge in the tree (making it equivalent to Zahn's algorithm [7]) and random walker would simply return a cut such that voxels with a shorter distance to each seed (with respect to the tree) would be classified with the label of that seed.

3 Method

In this section, we review the isoperimetric algorithm of [1], show that the computations may be performed in linear time if the underlying graph is a tree, introduce the distance tree and summarize the entire IDT algorithm.

The isoperimetric algorithm is formulated on a graph where, in the image processing context, each node represents a voxel and edges connect neighboring voxels in a 6-connected lattice. Formally, a **graph** is a pair $G = (V, E)$ with vertices $v \in V$ and edges $e \in E \subseteq V \times V$. An edge, e, spanning two vertices, v_i and v_j, is denoted by e_{ij}. Let $n = |V|$ and $m = |E|$ where $|\cdot|$ denotes cardinality. A **weighted graph** has a value (here assumed to be nonnegative and real) assigned to each edge called a **weight**. The weight of edge e_{ij}, is denoted by $w(e_{ij})$ or w_{ij} and represents the strength of affinity between neighboring voxels.

3.1 Isoperimetric Graph Partitioning

The isoperimetric graph partitioning algorithm of [1] was motivated by the solution to the classical isoperimetric problem, namely: *Given an area of fixed size, what shape has the minimum perimeter?* In $\mathbb{R}^2$, the answer has been known since ancient times to be a circle. However, on an arbitrary manifold, particularly with an unusual metric, the solution is not always obvious. In particular, it is known that the solution to the isoperimetric problem often partitions the manifold at bottleneck points, as exhibited in Cheeger's classic paper on the subject [17].

Unfortunately, on a discrete manifold (represented as a graph), the solution to the isoperimetric problem is known to be NP-Hard [1]. However, one may give a sense of how close a particular partition is to the solution of the isoperimetric problem by defining the **isoperimetric ratio** as the ratio of the perimeter of a node set to the number of nodes in the set and looking for a partition that minimizes this ratio [1].

The isoperimetric algorithm for graph partitioning may be developed by writing the isoperimetric ratio as

$$h_G(x) = \min_x \frac{x^T L x}{x^T r}, \tag{1}$$

subject to $x^T r \leq \frac{n}{2}$, where r is the vector of all ones, x represents a vector indicating node membership in a set $S \subseteq V$, i.e.,

$$x_i = \begin{cases} 0 & \text{if } v_i \in S, \\ 1 & \text{if } v_i \in \overline{S}. \end{cases} \tag{2}$$

The $n \times n$ matrix L is the **Laplacian** matrix [18] of the graph, defined as

$$L_{v_i v_j} = \begin{cases} d_i & \text{if } i = j, \\ -w(e_{ij}) & \text{if } e_{ij} \in E, \\ 0 & \text{otherwise.} \end{cases} \tag{3}$$

where d_i denotes the weighted **degree** of vertex v_i

$$d_i = \sum_{e_{ij}} w(e_{ij}) \ \ \forall\, e_{ij} \in E. \tag{4}$$

The notation $L_{v_i v_j}$ is used to indicate that the matrix L is indexed by vertices v_i and v_j.

With these definitions, the numerator of the ratio in (1) represents the sum of the weights of the edges spanning S and $\overline{S}$, while the denominator gives the cardinality of S.

By relaxing the binary definition of x and minimizing the numerator of (1) with respect to x, given the cardinality constraint $|V| - x^T r = k$, one is left with a singular system of equations. The singularity may be overcome by arbitrarily

assigning one node, v_g, (termed the **ground** in [1] by way of a circuit analogy) to S, resulting in the nonsingular system

$$L_0 x_0 = r_0, \tag{5}$$

where the subscript indicates that the row corresponding to v_g has been removed (or the row and column, in the case of L_0).

Given a real-valued solution to (5), one may convert this solution into a partition by finding the threshold that produces a partitioning with minimal isoperimetric ratio, which requires trying only n thresholds. When trying thresholds in order to measure the isoperimetric ratio of the resulting segmentation, we employ a denominator of $x^T r$ if $x^T r < \frac{n}{2}$ and $\left(n - x^T r\right)$ otherwise. It was proved in [1] that this strategy produces a connected object and shown that the ground node behaves as a specification of the *foreground*, while the background is determined from the thresholding of the solution to (5).

In the present context, we are only interested in the geometry of the graph (mask), and therefore, during the solution to (5) we treat all $w_{ij} = 1$.

3.2 Trees

Although the solution of the linear system in (5) is fast, since the matrix is sparse, symmetric and positive-definite (allowing for the use of such memory efficient methods as conjugate gradients), the enormity of data that comprises current medical volumes demands an even faster approach.

Since it is known that a matrix with a sparsity pattern representing a tree has a zero-fill Gaussian elimination ordering [19], we propose to replace the

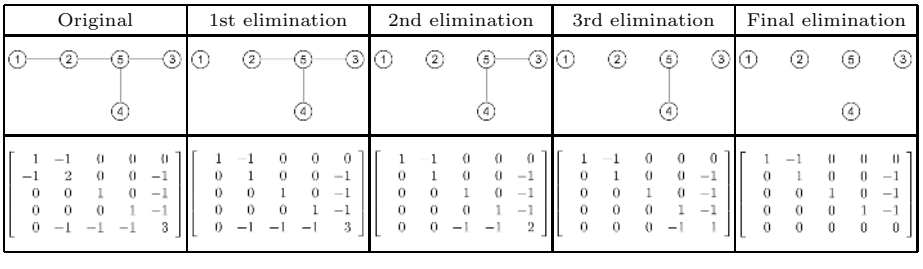


Fig. 1. Gaussian elimination of the Laplacian matrix of a tree with ordering given by the numbers inside the nodes. Note that the resulting Gaussian elimination has the same sparsity structure as the original matrix when a no-fill ordering is used (e.g., as computed by Algorithm 1). This is why we need only compute the no-fill ordering, and not the full Gaussian elimination, in order to solve the linear system required by the isoperimetric algorithm. Note that the Laplacian matrix is singular — the last elimination produces a row of all zeros. Once the graph has been grounded, as in (5), this is no longer a concern e.g., if node 5 were grounded, the elimination would stop after the third elimination and $x_5 = 0$ would be used to recover the remaining values of the solution. Top row: Elimination of the tree — the figures depict the graph represented by the lower triangle of the matrix. Bottom row: Laplacian matrix of the tree after each elimination step.

Algorithm 1. Produce a no-fill ordering of a tree

```
1: void compute_ordering(degree, tree, ground, ordering)
2: k ⇐ 0
3: degree[root] ⇐ 0 {Fixed so that ground is not eliminated}
4: ordering[N − 1] ⇐ ground
5: for each node in the graph do
6:    while degree[current_node] equals 1 do
7:       ordering[k] ⇐ current_node
8:       degree[current_node] ⇐ degree[current_node]−1
9:       current_node ⇐ tree[current_node]
10:      degree[current_node] ⇐ degree[current_node]−1
11:      k ⇐ k + 1
12:   end while
13:   k ⇐ k + 1
14: end for
```

standard lattice edge set with a *tree*. A zero-fill Gaussian elimination ordering means that the system of linear equations may be solved in two passes, with storage equal to n, since all entries in the matrix that were initially zero remain zero during the Gaussian elimination. Specifically, the ordering may be found in linear time by eliminating the nodes with (unweighted) degree of one (i.e., leaf nodes in the tree) and recursively eliminating nodes which subsequently have degree one until a root node is reached. In this case, a convenient root node is the ground. Algorithm 1 accomplishes the ordering in linear time, where the array `tree` contains, for each node, the index of one neighbor (with no edges overrepresented) and the array `degree` contains the degree of each node in the tree. This representation is possible since a tree has $n - 1$ edges (where the root would contain a '0').

Figure 1 illustrates a small tree with corresponding L and elimination. Once the elimination ordering is computed, the system in (5) may be solved by taking a forward pass over the nodes to modify the right hand side (i.e., the elimination of Figure 1) and then a backward pass to compute the solution. Algorithm 2 finds a solution to (5) in linear time, given a tree and an elimination ordering. Note that, as stated above, we assume that all $w_{ij} = 1$ and that the graph geometry (i.e., mask shape) encodes the pertinent information.

Consequently, when the graph is a tree, a low-constant linear time algorithm is available to compute a no-fill Gaussian elimination ordering, solution of (5) and subsequent thresholding to produce a partition. We note also that Branin has shown how to produce an explicit inverse for the Cholesky factors of a grounded Laplacian matrix [20]. Recall that the Cholesky factors are the results of Gaussian elimination for a symmetric, positive-definite matrix, i.e., from an LU matrix decomposition, $L = U = C$ for a symmetric, positive-definite matrix, where C is the Cholesky factor. However, the above procedure is simpler and more memory efficient than explicitly constructing the inverses of the Cholesky factors.

Algorithm 2. Given a tree, solve (5)

```
1: solve_system(ordering, diagonal, tree, r, output)
2: {Forward pass}
3: k ⇐ 0
4: for each non-ground node do
5:    r[tree[ordering[k]]] ⇐ r[ordering[k]]/diagonal[ordering[k]]
6:    k ⇐ k + 1
7: end for
8:
9: output[ordering[N−1]] ⇐ r[ordering[N−1]]/diagonal[ordering[N − 1]]
10:
11: {Backward pass}
12: k ⇐ N−2 {Last non-ground node}
13: for each non-ground node do
14:    output[ordering[k]] ⇐ output[tree[ordering[k]]] +
       r[ordering[k]]/diagonal[ordering[k]]
15:    k ⇐ k − 1
16: end for
```

3.3 Distance Trees

In the above section, we have shown that by using a tree as the underlying graph structure (instead of the usual lattice), a very fast, linear-time solution to (5) may be obtained. We take the position that the desired cut will be a solution to the isoperimetric problem (i.e., the cut will minimize the isoperimetric ratio) and therefore we want to select a tree such that a threshold of the solution to (5) will produce the desired cut.

The most important property of a tree, such that the solution will examine the desired cut is: The path within the tree between the foreground point and the remaining voxels in the foreground object do not pass through any voxels in the background. i.e., the foreground is *connected* within the tree. If this condition is satisfied, and the background is also connected within the tree, then the foreground and background are connected with a single edge (since there may be no loops in a tree).

A tree satisfying the above desiderata may be constructed if the following conditions are satisfied: 1) The foreground object is connected, 2) Gradient ascent on the distance map from each node stabilizes at a node in the same set (i.e., foreground nodes stabilize on a foreground node and background nodes stabilize on a background node), 3) The distance value at all neighboring nodes that stabilize to different peaks is smallest along the true foreground/background boundary. The tree may be constructed by assigning to each edge in the lattice the weight

$$w_{ij} = D(v_i) + D(v_j), \tag{6}$$

where $D(v_i)$ denotes the distance map [12] at node v_i, and then compute the *maximal spanning tree* [21]. The above desirable situation may also be restated in terms of the watershed algorithm [3]. If the foreground/background boundary

occurs on the boundary of watershed basins and the height (in terms of D) of the basins separating the foreground/background boundary is larger than the basin boundaries internal to the foreground or background regions then the MST will span the foreground/background with a single edge. We note, however, that the above condition is simply sufficient to produce a tree with a connected foreground, although not necessary. Since a watershed algorithm also requires that the desired boundary lie on a watershed boundary, the isoperimetric algorithm is expected to work whenever a watershed algorithm would work, given a simply connected mask, but may additionally work in more difficult cases.

We term the maximal spanning tree of the image with weights given by (6) as the **distance tree**. We note that, as with a watershed algorithm, it would also be possible to employ different choices of function in (6). With respect to the watershed literature, the most common choices would be a distance map [12] of a masked part of the image, image gradient strength or image intensity. Furthermore, these different choices may be combined via multiplication of their respective weights. For purposes of the mask segmentation problem considered here, we restrict ourselves to distance maps.

3.4 Summary of IDT

The IDT algorithm proposed here may be summarized in the following steps:

1. Obtain a mask from the image data (e.g., via thresholding or region growing).
2. Compute a distance map on the mask.
3. Obtain a problem-specific ground (foreground) point.
4. Compute the maximal spanning tree (using Kruskal, Prim, etc.) on the lattice with edge weights given by (6).
5. Compute a no-fill ordering using Algorithm 1.
6. Solve the system in (5) using Algorithm 2.
7. Check n thresholds of the solution to (5) and choose the one such that the resulting segmentation minimizes the isoperimetric ratio of (1).

All of the above steps have a $\mathcal{O}(n)$ complexity, except for computation of the maximal spanning tree, which has a complexity of $\mathcal{O}(n \log(n))$ (for a lattice). However, as demonstrated in Section 4, the algorithm performs quickly in practice.

We note that several nodes may be used as the ground points, requiring their removal from the L matrix in (5) and fixing their values to zero in the solution procedure of Section 3.2. If desired, background seeds may also be incorporated by only considering thresholds below the x values of the background seeds. Since the x value of the ground will be zero, which will be the smallest x value of any node [1], the threshold will be guaranteed to separate the foreground from the background. Note also that the checking done by the algorithm for partition quality (i.e., the last step in the above summary) is done using the *original* graph (mask), not simply the tree.

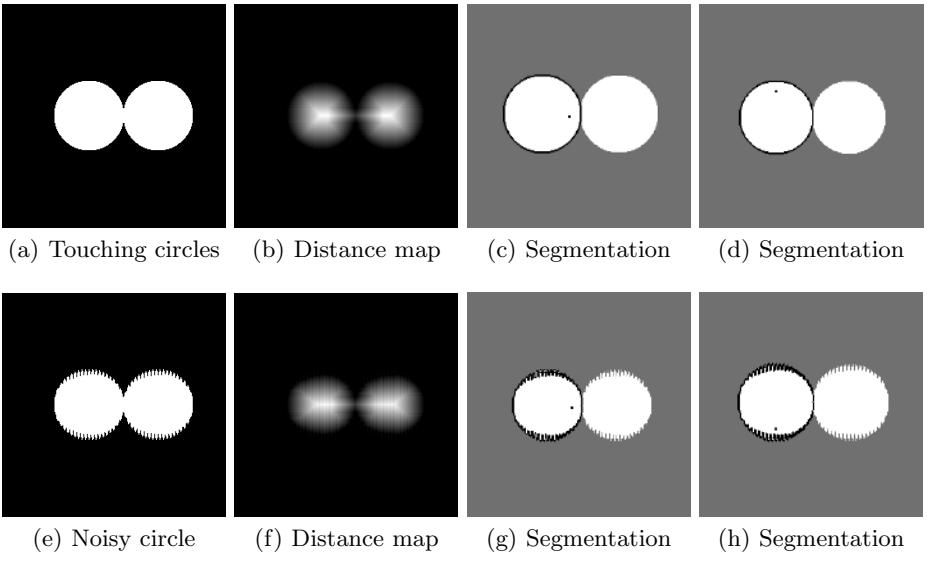
4 Results

In this section, we first show several synthetic examples of mask segmentation problems where a region-growing or watershed algorithm would fail but the IDT algorithm succeeds. Finally, several examples are given with real data.

4.1 Synthetic Examples

In Section 1, several common problems with watershed algorithms were outlined. Specifically, a watershed approach fails if both objects fall in the same watershed region and produces an overabundance of watershed regions in noisy images (requiring an additional merging process).

In Figure 2 the problem of two touching circles is examined. Figure 2 illustrates the distance map and distance tree for two touching circles both with and without noise. Despite the small amount of noise added to the shape and the obviousness of the bottleneck, a watershed approach is left with many watershed regions, requiring an additional merging process to find the correct solution. However, the distance tree is relatively unchanged with noise and



(a) Touching circles (b) Distance map (c) Segmentation (d) Segmentation

(e) Noisy circle (f) Distance map (g) Segmentation (h) Segmentation

Fig. 2. A simple case of two touching circles with noise. Although the distance map of the two circle image in (a) has one watershed basin corresponding to each circle, the small amount of noise associated with figure (e) results in many watershed basins within each circle. Consequently, an additional merging process would need to be employed by a watershed approach in order to obtain the desired segmentation. In contrast, no modification is necessary for the IDT approach. Figures (c,d,g,h) give segmentation results of the IDT algorithm. Each figure shows the user-supplied foreground point represented by a small black dot and the resulting foreground segment outlined in black. Note that no shape assumption was used — The IDT algorithm effectively segments the mask at bottlenecks.

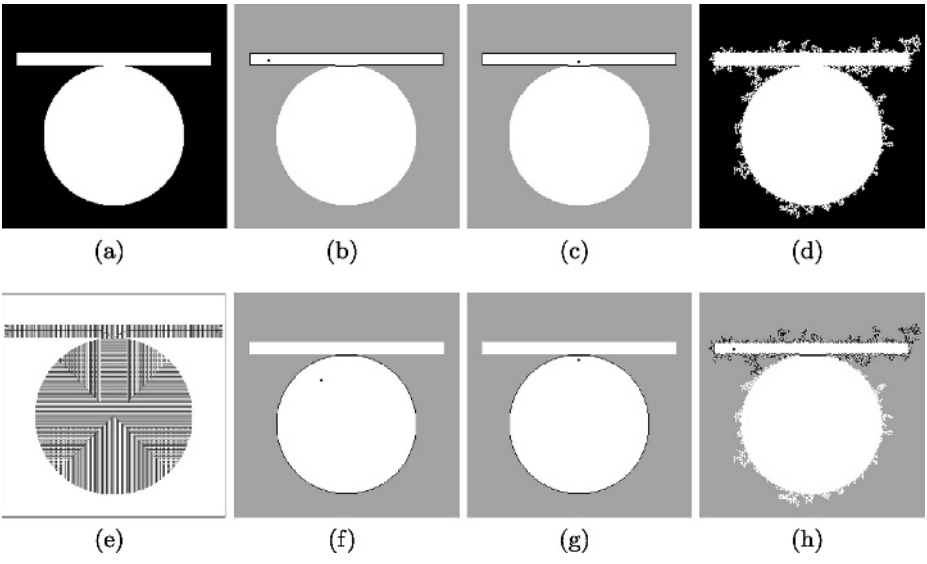


Fig. 3. A thin bar strongly connected to a circle (i.e., the width of the intersection exceeds the width of the bar). A watershed approach is incapable of separating the bar from the circle, since they both fall in the same watershed region. In contrast, the IDT algorithm is capable of finding a correct segmentation with a variety of ground points and noise. a) Original image, b) Distance tree, c–f) Various ground points (black dot) with corresponding segmentation (indicated by black line). Note that placement of the ground node near the desired boundary does not disrupt the segmentation. g) Noise was added to the mask to produce this multiply-connected example of the bar/circle. h) Segmentation of noisy mask.

Figure 2 illustrates that the IDT approach correctly segments the mask, regardless of noise or ground point. We note that this touching circles example strongly resembles the "dumbbell" manifold described by Cheeger [17] for which there is a clear solution to the isoperimetric problem, i.e., an algorithm based on minimizing the isoperimetric ratio should be expected to handle this problem well.

Figure 3 describes an entirely different scenario. Again, two objects (a circle and a bar) are weakly connected, requiring a sophisticated bottleneck detector. However, since the width of the circle/bar connection exceeds the width of the bar, both objects occupy the same watershed basin. Therefore, even a sophisticated basin-merging procedure (necessary for the situation of Figure 2) is incapable of assisting in the present situation, since both the circle and the bar occupy the same basin. However, as shown in Figure 3, such a situation is not a problem for the IDT algorithm. Specifically, choosing a ground (foreground point) in either the circle or the bar will produce the same circle/bar separation.

4.2 Real-World Volumes

The above results on synthetic examples suggest that the IDT algorithm should be expected to produce fast, quality results on large volumes. In this section, we explore the questions of quality and speed for large medical volumes. Segmentation examples were chosen to highlight the versatility of the IDT approach. All computation times reflect the amount of time required to perform all steps of the IDT algorithm given in Section 3.4. Since a Euclidean distance map was unnecessary for our algorithm, we employed a fast L1-metric distance map [22]. The maximal spanning tree was computed using a standard algorithm [21].

We first apply the IDT approach to a set of 2D examples with notoriously unreliable boundaries — ultrasound images. The mask was generated by thresholding out the dark regions and placing a ground point in the desired chamber. Note that no background seeds were placed. The results of the segmentation are displayed in Figure 4. Although the touching circles example of Figure 2

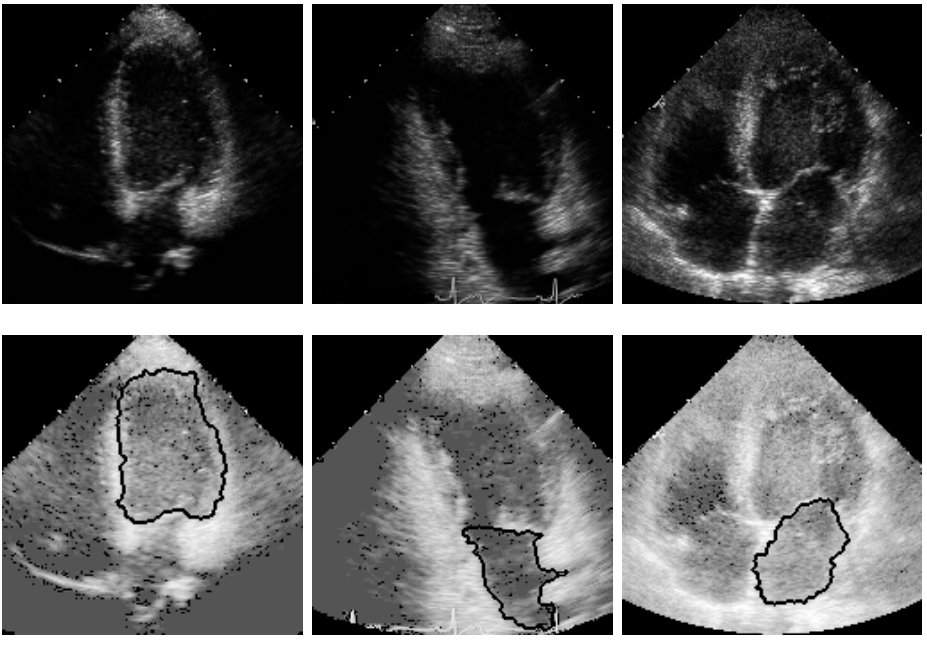


Fig. 4. Application of the IDT segmentation to 2D examples with notoriously unreliable boundaries — ultrasound images. The mask given to the IDT consisted of all pixels with an intensity below a given threshold. The ground (foreground) point was in the center of each object. Note the similarity of this problem to the touching circles example of Figure 2, since the heart chambers are weakly connected to each other through the open valve. All images were 240×320 with approximately 92% of the pixels inside the mask and required approximately 0.3s to process. Note that the top-row represents the images with their original (input) intensities, while the intensities in the bottom row were whitened to enhance the visibility of the contours.

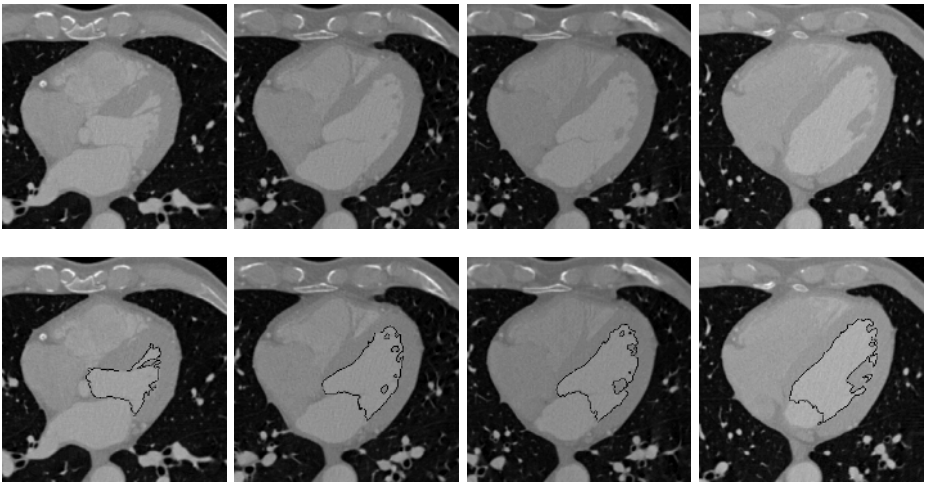
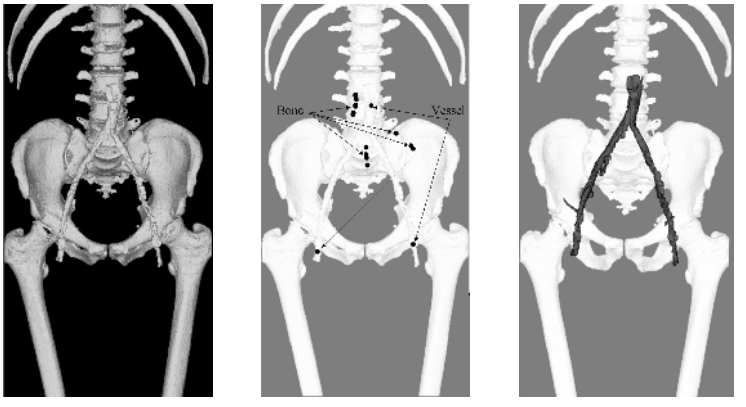


Fig. 5. The IDT algorithm applied to left ventricle segmentation of a mask produced by thresholding the volume to separate the blood pool from the background. A single ground point was placed inside the left ventricle and the IDT algorithm was run in 3D. No background seeds were given. Note that, with the open valve to the left atrium (and the aorta), this problem resembles the touching circles scenario of Figure 2. Top row: Slices from the original volume. Bottom row: Segmentation (outlined in black). The time required to perform the segmentation on this $256 \times 256 \times 181$ volume with 1,593,054 voxels inside the mask was 7.37s. Note that the papillary muscles were excluded from the blood pool mask.

was synthetic, this type of bottleneck detection is also the essence of these ultrasound segmentations since the heart chambers are weakly connected to each other through open valves and noisy boundaries.

The CT volume of Figure 5 is also suited to this segmentation approach. By thresholding the volume at the level of the blood pool, a mask was produced such that each chamber was weakly connected to each other by open or thin valves. Consequently, it was possible to apply the IDT to the mask with a ground point inside the desired chamber (in this case, the left ventricle). Figure 5 shows several slices of the segmentation of the left ventricle, using a ground (foreground) point inside the chamber. As with the ultrasound data, the mask contains weakly connected objects (i.e., the left ventricle blood pool is weakly connected to the atrium and the aorta through the thin valves, which are frequently open), making this task analogous to the touching circles example, for which the IDT was shown to behave robustly. Note that no background seeds were necessary to produce these segmentations.

Although it is possible to threshold bone in CT data, calcified blood vessels frequently also cross threshold and, more importantly, are pressed close against the bone. Consequently, the separation of bone from vessel inside the mask is a challenging task. However, as was shown in the bar/circle synthetic example of Figure 3, the IDT approach is capable of separating two tightly pressed



(a) Input (b) Seeds (c) Vessel

Fig. 6. The IDT algorithm applied to interactive vessel/bone separation. This problem is difficult because bone and vessel have similar intensities and strongly touch each other (i.e., have a weak boundary) in several places, as in the bar/circle separation problem of Figure 3. (a) Original 3D mask obtained by thresholding. (b) User-specified seeds (ground) of vessel (foreground) and bone (background). (c) Resulting vessel segmentation shaded in gray.

structures, even in noisy masks and when the point of contact is larger than the bar (vessel). Figure 6 shows the results of applying the IDT interactively to separate a blood vessel from bone. In this case, multiple grounds (i.e., vessel seeds) were placed and, additionally, background seeds (i.e., bone seeds) were also used to constrain the thresholding procedure.

5 Conclusion

The challenge of quickly segmenting regions of interest within large medical volumes frequently forces the use of less-sophisticated segmentation algorithms. Often, thresholding or region growing approaches are nearly sufficient for the required segmentation task, except that one is left with a weakly connected mask that a smart bottleneck detector is capable of parsing. Since watershed algorithms are used almost exclusively in this setting, and there are significant concerns with this approach, we have proposed a new algorithm based on operating the recent, sophisticated isoperimetric algorithm [1] on a tree derived from the mask geometry. This approach is shown to be fast, widely-applicable, high-quality, robust to noise and initialization, require specification of only a foreground seed and is not bound by the limitations of the watershed algorithm.

Further work includes investigation of other useful tree structures (e.g., based on functions other than a distance map), determining the suitability of graph structures with a level of connectivity between the lattice and the tree and domain-specific applications of this general approach to mask segmentation.

References

1. L. Grady and E. L. Schwartz, "The isoperimetric algorithm for graph partitioning," *SIAM Journal on Scientific Computing*, 2006, in press.
2. C. R. Brice and C. L. Fennema, "Scene analysis using regions," *Artificial Intelligence*, vol. 1, no. 3, pp. 205–226, 1970.
3. J. Roerdink and A. Meijster, "The watershed transform: definitions, algorithms, and parallellization strategies," *Fund. Informaticae*, vol. 41, pp. 187–228, 2000.
4. L. Grady and E. L. Schwartz, "Isoperimetric graph partitioning for image segmentation," *IEEE Trans. on Pat. Anal. and Mach. Int.*, vol. 28, no. 3, pp. 469–475, March 2006.
5. J. A. Sethian, *Level Set Methods and Fast Marching Methods.* Cambridge University Press, 1999.
6. K. Krissian and C.-F. Westin, "Fast sub-voxel re-initialization of the distance map for level set methods," *Pattern Rec. Let.*, vol. 26, no. 10, pp. 1532–1542, July 2005.
7. C. Zahn, "Graph theoretical methods for detecting and describing Gestalt clusters," *IEEE Transactions on Computation*, vol. 20, pp. 68–86, 1971.
8. R. Urquhart, "Graph theoretical clustering based on limited neighborhood sets," *Pattern Recognition*, vol. 15, no. 3, pp. 173–187, 1982.
9. P. F. Felzenszwalb and D. P. Huttenlocher, "Efficient graph-based image segmentation," *Int. J. of Computer Vision*, vol. 59, no. 2, pp. 167–181, Sept. 2004.
10. M. Wan, Z. Liang, Q. Ke, L. Hong, I. Bitter, and A. Kaufman, "Automatic centerline extraction for virtual colonoscopy," *IEEE Trans. on Medical Imaging*, vol. 21, no. 12, pp. 1450–1460, December 2002.
11. C.-H. Wu and P. C. Doerschuk, "Tree approximations to Markov random fields," *IEEE Trans. on Pat. Anal. and Mach. Int.*, vol. 17, no. 4, pp. 391–402, April 1995.
12. A. Jain, *Fundamentals of Digital Image Processing.* Prentice-Hall, Inc., 1989.
13. J. Shi and J. Malik, "Normalized cuts and image segmentation," *IEEE Trans. on Pat. Anal. and Mach. Int.*, vol. 22, no. 8, pp. 888–905, Aug. 2000.
14. Y. Boykov and M.-P. Jolly, "*Interactive graph cuts* for optimal boundary & region segmentation of objects in N-D images," in *Proc. of ICCV 2001*, 2001, pp. 105–112.
15. L. Grady and G. Funka-Lea, "Multi-label image segmentation for medical applications based on graph-theoretic electrical potentials," in *Computer Vision and Mathematical Methods in Medical and Biomedical Image Analysis, ECCV 2004*, no. LNCS3117. Prague, Czech Republic: Springer, May 2004, pp. 230–245.
16. P. E. John and G. Schild, "Calculating the characteristic polynomial and the eigenvectors of a tree," *MATCH*, vol. 34, pp. 217–237, Oct. 1996.
17. J. Cheeger, "A lower bound for the smallest eigenvalue of the Laplacian," in *Problems in Analysis*, R. Gunning, Ed. Princeton, NJ: Princeton University Press, 1970, pp. 195–199.
18. R. Merris, "Laplacian matrices of graphs: A survey," *Linear Algebra and its Applications*, vol. 197, 198, pp. 143–176, 1994.
19. K. Gremban, "Combinatorial preconditioners for sparse, symmetric diagonally dominant linear systems," Ph.D. dissertation, Carnegie Mellon University, Pittsburgh, PA, October 1996.
20. F. H. Branin, Jr., "The inverse of the incidence matrix of a tree and the formulation of the algebraic-first-order differential equations of an RLC network," *IEEE Transactions on Circuit Theory*, vol. 10, no. 4, pp. 543–544, 1963.
21. A. Gibbons, *Algorithmic Graph Theory.* Cambridge University Press, 1989.
22. A. Rosenfeld and J. L. Pfaltz, "Sequential operations in digital picture processing," *J. of the Assoc. for Computing Machinery*, vol. 13, no. 4, pp. 471–494, Oct. 1966.