Research Article

Semiautonomous Medical Image Segmentation Using Seeded Cellular Automaton Plus Edge Detector

Ryan A. Beasley

Department of Engineering Technology and Industrial Distribution, Texas A&M University, College Station, TX 77843, USA

Correspondence should be addressed to Ryan A. Beasley, beasley@entc.tamu.edu

Received 27 January 2012; Accepted 15 March 2012

Academic Editors: C. Alberola-Lopez, Y. H. Ha, C. S. Lin, C. Sun, and B. Yuan

Copyright © 2012 Ryan A. Beasley. This is an open access article distributed under the Creative Commons Attribution License, which permits unrestricted use, distribution, and reproduction in any medium, provided the original work is properly cited.

Segmentations of medical images are required in a number of medical applications such as quantitative analyses and patient-specific orthotics, yet accurate segmentation without significant user attention remains a challenge. This work presents a novel segmentation algorithm combining the region-growing Seeded Cellular Automata with a boundary term based on an edge-detected image. Both single processor and parallel processor implementations are developed and the algorithm is shown to be suitable for quick segmentations (2.2 s for $256 \times 256 \times 124$ voxel brain MRI) and interactive supervision (2–220 Hz). Furthermore, a method is described for generating appropriate edge-detected images without requiring additional user attention. Experiments demonstrate higher segmentation accuracy for the proposed algorithm compared with both Graphcut and Seeded Cellular Automata, particularly when provided minimal user attention.

1. Introduction

Segmentation, also known as labeling, of medical images is an important task for quantitative analyses, custom intervention planning such as localized radiotherapy, and design of patient-specific tools such as orthotics or jigs for joint replacement. Manually labeling images takes a prohibitive amount of time due to the large number of voxels in most medical images, while autonomous segmentations can fail to reach the required accuracy due to interpatient morphological variability. Supervised segmentation algorithms are a promising solution because they allow the user to guide the segmentation without requiring the user's attention for each voxel.

Popular supervised segmentation algorithms include active contours (snakes) [1], Level Sets [2], intelligent scissors (live wire) [3, 4], Graphcut [5], and to some degree Seeded Cellular Automata (SCA) [6]. For active contours and Level Sets, the user initializes the boundary near the desired contour and the algorithm moves the boundary to a local minimum determined by an energy functional. This approach requires that the user solve two optimization problems: setting the impact of the terms in the functional, and placing the initial boundary such that the results settle

into a desirable minimum [7]. Intelligent scissors connect-the-dots between user-placed boundary points, but either image noise or inaccurate placement of those points can generate inaccurate segmentations. Graphcut uses graph-based operations to separate user-placed “seeds” so as to satisfy the max-flow/min-cut theorem, but demonstrates a bias toward small segmentations and is inherently a two-label approach (with research on K-way heuristic solutions) [8]. Finally, SCA grows regions based on local information making it well suited to parallel hardware [6], and has a constant computation time with respect to the number of labels, but SCA is prone to bleed through at weak boundaries [9].

This paper presents a novel segmentation algorithm that extends the region-growing SCA with a boundary term based on an edge detector. The resulting algorithm demonstrates improved segmentation accuracy, particularly given minimal user attention, while allowing real-time user supervision. In addition to the algorithm, this paper provides an approach for generating the edge-detected image and a description of the code used to run the algorithm on a consumer graphics card (GPU). Validation results compare the proposed algorithm with Graphcut and SCA.

2. Methods

This section describes the proposed algorithm, which is created by combining a region-growing segmentation algorithm (Ford-Bellman), a cell-based dynamic system (Cellular Automata), and an edge image (Canny filter). This section also explores implementation details, investigates how to generate a good edge image, and describes validation experiments.

2.1. Ford-Bellman Region-Growing Algorithm. Region-growing segmentation algorithms expand labels from seeds, which are voxels with *a priori* labels [10, 11]. Typically, such algorithms iteratively determine whether voxels adjacent to a currently labeled region should be added to that region. The Ford-Bellman algorithm (FBA) uses this iterative method to solve the single-source shortest-path problem on a graph of vertices connected by graph edges [12, 13], and can perform a region-growing segmentation.

For FBA, the graph is initialized such that each seed vertex has a distance of zero and a label, each nonseed vertex has an infinite distance and no label, each graph edge between neighboring vertices has a length based on some measure (e.g., absolute difference in image intensities of its endpoints), and all the seed vertices are interconnected with zero-length graph edges. In each iteration, every graph edge with at least one labeled endpoint is tested to see if the sum of the distance of the source (the endpoint with the smaller distance) and the length of the graph edge is less than the distance of the target (the other endpoint). If so, the target's distance becomes that sum and it copies the source's label. Thereby, for iteration number n , each vertex's distance is the minimum sum of graph edge lengths necessary to reach any seed within n graph edge traversals. With sufficient iterations, each voxel's distance is the shortest possible path length to any seed, and each voxel has the label from the nearest seed. Thus, FBA segments the image by growing the seeds into regions based on the graph edge length calculation.

FBA is not the fastest shortest-path algorithm, but by taking advantage of its highly parallelizable nature, FBA can segment medical images faster than the fastest-known single-processor approach, Dijkstra's algorithm. Furthermore, unlike Dijkstra's algorithm (see Section 2.4.2), FBA is conducive to interactive user corrections because seeds added during a segmentation propagate normally in later iterations. This work investigates implementing FBA on the current generation of graphics cards (GPUs) and contains an experiment comparing the computational times needed for FBA and Dijkstra's algorithm on typical medical images.

2.2. Seeded Cellular Automata Segmentation Algorithm. A Cellular Automaton (CA) is a discrete dynamic system that iteratively performs local calculations on a grid, for example, Conway's Game of Life [14], making them well suited for efficient implementation on the parallel hardware of General-Purpose Graphics Processing Units (GPGPU). CA has been used to segment images by implementing FBA, each voxel being represented by a cell with the state containing

the voxel's image value, label, and distance. If the graph edge length calculation is a function of just the two endpoint vertices, the FBA calculations are all local and can be used as the CA update rules.

Since this approach needs *a priori* seeds, this method is called Seeded Cellular Automaton (SCA). SCA is a supervised segmentation algorithm that allows additional seeds to be placed during the iterative process to correct the segmentation. Unfortunately, SCA compares poorly with other segmentation algorithms because it demonstrates leakages at weak boundaries, where image gradients due to objects of interest are small with respect to spurious image gradients [9]. Since SCA's calculation for region growth is based on voxel-to-voxel differences, SCA is more likely to place label borders at high image gradients, but the impact of weak boundaries can be insufficient. Thus, SCA requires significant user attention to accurately place label borders at weak boundaries.

2.3. Proposed Segmentation Algorithm, SCAPE. The proposed algorithm, Seeded Cellular Automaton Plus Edge (SCAPE) detector, improves SCA's performance at weak boundaries via a boundary term that impedes region growth through edge-voxels provided by an edge detecting filter (Algorithm 1). Let V be the set of vertices (voxels), with a set of bidirectional graph edges $e \in E \subseteq V \times V$ spanning two neighboring vertices, $v_{e,i}$ and $v_{e,j}$, defined by its neighborhood. A von Neumann neighborhood of range 1 (6-connected in 3D) is used because a larger neighborhood can allow region growth to bypass the single-voxel-wide features generated by some edge detecting filters, for example, the Canny filter. Initialization is via label seeds, $S \subset V$, that may be provided by user scribbles, an atlas, or other technique. Further, let $G \subset V$ be the set of vertices that correspond to edge-voxels.

A typical (e.g., [6]) length for a graph edge, $e \in E$, used in SCA is

$$\text{length}_{\text{SCA}}(e) = \frac{\|g(v_{e,i}) - g(v_{e,j})\|}{l_{\max}}, \quad (1)$$

$$l_{\max} = \operatorname{argmax}_{e \in E} \|g(v_{e,i}) - g(v_{e,j})\|,$$

where $g(\cdot)$ is the intensity, color, or other measure of a voxel. This calculation provides a straightforward measure of the (dis-) similarity between neighboring voxels, and ensures that the graph edge lengths are nonnegative to prevent looping paths. SCAPE's boundary-based term is added onto (1), resulting in line 16 of Algorithm 1, where $\lambda \in \mathbb{R}$ is a constant, and $k(e)$ is the number of endpoints of $e \in E$ that are edge-voxels. Positive values of λ increase the lengths of graph edges connected to edge-voxels, with greater values making it more likely that label borders lie on features in the edge image. However, $\lambda > n$, where n is the number of voxels, will result in the same result as $\lambda = n$, since in either case any loop-less path that does not pass through an edge-voxel will have a shorter distance than any loop-less path that does. Note that boundary smoothness, a common concern in medical image segmentation, can be enforced by adding

```

(1)  $distance(v) \leftarrow 0, \forall v \in S$ 
(2)  $distance(v) \leftarrow \infty, \forall v \in V \setminus S$ 
(3) repeat
(4)   for all  $e \in E$  do
(5)      $converged \leftarrow true$ 
(6)      $source \leftarrow \underset{v \in \{v_{e,i}, v_{e,j}\}}{\operatorname{argmin}} distance(v)$ 
(7)      $target \leftarrow \underset{v \in \{v_{e,i}, v_{e,j}\}}{\operatorname{argmax}} distance(v)$ 
(8)     if  $distance(target) > distance(source) + length(e)$  then
(9)        $converged \leftarrow false$ 
(10)       $distance(target) \leftarrow distance(source) + length(e)$ 
(11)       $label(target) \leftarrow label(source)$ 
(12)     end if
(13)   end for
(14) until  $converged$ 
(15) where
(16)    $length(e) := \|g(v_{e,i}) - g(v_{e,j})\|/l_{\max} + \lambda * k(e),$ 
(17)    $l_{\max} := \max_{e \in E} \|g(v_{e,i}) - g(v_{e,j})\|,$ 
(18)    $k(e) := \begin{cases} 0, & (v_{e,i} \notin G) \wedge (v_{e,j} \notin G) \\ 1, & (v_{e,i} \in G) \oplus (v_{e,j} \in G) \\ 2, & (v_{e,i} \in G) \wedge (v_{e,j} \in G) \end{cases}$ 

```

ALGORITHM 1: Seeded Cellular Automaton Plus Edge detector.

a local smoothness check to Algorithm 1, as in [15], or by presmoothing the image before applying the edge detecting filter.

2.4. SCAPE Implementations. FBA is $O(|V||E|)$, so implementing Algorithm 1 by naïvely looping over all the voxels can take a significant amount of time before the segmentation converges [6]. For example, running on a desktop computer (described in Section 2.6), SCAPE took 345 seconds (1229 iterations) to segment a brain MRI with $256 \times 256 \times 124$ voxels. Three types of implementations were investigated to reduce the time needed for a segmentation. First, an attempt was made to accelerate SCAPE on a single processor. Second, an equivalent approach to SCAPE using Dijkstra’s algorithm was implemented on a single processor. Third, SCAPE was implemented on parallel processors.

2.4.1. SCAPE Implementation on CPU. Three improvements were investigated to reduce the time for SCAPE to converge while running on a single processor. First, the calculations were performed using single-buffering instead of the double-buffering that is commonly described for FBA (colloquially, “in-place” updates instead of “ping-pong”) [6, 13]. Using single-buffering, an updated distance propagates through other calculations in the same iteration, reducing the number of iterations required. Second, via Yen’s improvement, the order of the vertices (or edges) was reversed each iteration, allowing the propagation of updates to spread first one direction and then the other [16]. Third, a narrow-band approach was implemented in which the voxels were grouped into $8 \times 8 \times 8$ “tiles” (size chosen heuristically). The calculations for a tile were skipped unless one or more vertices in that tile, or in the orthogonally adjacent (von Neumann neighbor) tiles, were updated in the previous iteration. These

algorithmic changes do not affect the typical inductive proof of correctness for FBA, and extensive tests have resulted in equivalent results with and without these improvements.

2.4.2. Dijkstra’s Algorithm with Edges on CPU. The second implementation tested herein was Dijkstra’s algorithm on a single processor [17]. Like FBA, Dijkstra’s algorithm solves the single-source shortest-path problem, and so can segment images. Furthermore, Dijkstra’s algorithm is asymptotically the fastest known algorithm for graphs with unbounded nonnegative weights, $O(|V| \log |V| + |E|)$, when coded using a priority queue.

The algorithm initially places all vertices in a priority queue based on the shortest known distance to each. Iteratively, the queued vertex with the shortest distance is removed from the queue, and its graph neighbors’ shortest distances and labels are updated (as in Algorithm 1, lines 8–12). Thus, a given edge is used in a single calculation, and as soon as a vertex’s shortest distance is known it leaves the queue and is not needed for further calculations.

The algorithm has two traits that reduce its suitability for interactive segmentation. First, Dijkstra’s algorithm is not conducive to user corrections during a segmentation. Once a vertex is removed from Dijkstra’s priority queue, its distance is not updated unless the algorithm is reinitialized. Second, the algorithm is not clearly parallelizable due to its reliance on the priority queue, though multiple processors can provide some benefit, for example, Delta-Stepping [18].

2.4.3. SCAPE Implementation on GPGPU. SCAPE is well suited for running on parallel hardware, due to the local nature of the SCAPE calculations, and because FBA does not constrain the order that edges or vertices are inspected

[6]. SCAPE, therefore, benefits from running on a General-Purpose Graphics Processing Unit (GPGPU), a highly parallel multiprocessor system. Functions compiled for a GPGPU are called kernels, which perform the same set of operations on all elements in a “stream,” for example, the voxels in an image. Each element is operated on by a single process, a “thread.” Threads are grouped into “blocks” that are automatically assigned to any available processor. The processors simultaneously run the kernel on “warps” of some number of consecutive threads in the block.

The initial step for running SCAPE on a GPGPU is to transfer the data into the GPGPU’s limited memory. Then, two kernels alternate running on the GPGPU. The first kernel performs the calculations to update the vertices’ distances and records which vertices had their distances updated. The second kernel checks if all the distances have stopped changing, which indicates the segmentation is finished. Finally, the data is transferred back out of the GPGPU’s memory.

In the distance-updating kernel, each thread updates the distance of a single vertex using the data from the neighboring vertices. Various approaches were attempted, with the fastest having a block consisting of all the voxels with the same y and z indices. Thus, a given block calculates the new distances for one x -row of voxels, and must load the data for that x -row and the four orthogonally adjacent x -rows (due to the von Neumann neighborhood). Less successful alternatives include shaping the blocks as other rectangular solids, using shared memory, loading the data into CUDA arrays instead of linear arrays, and using textures or surfaces. These calculations must be double buffered because nondeterminate behavior occurs whenever a kernel reads and writes the same data location in multiple threads; single-buffering caused memory corruption. As a result, this kernel requires 13 bytes of memory per voxel. The calculations and memory fetches are skipped if an entire warp only involves vertices unchanged in the previous iteration.

After the distances have been updated, each warp sets a flag bit high if any of its threads updated a distance. Then, the second kernel “reduces” those flag bits (one per warp) using the “or” operator to a single flag bit that is low when the segmentation has converged. For this work running on NVIDIA architecture, this kernel used the reduction code titled “Reduce6” that is provided with the CUDA documentation, and which required over 2 bytes per warp of 32 threads.

2.5. Methods for Determining a Suitable Edge Image. SCAPE’s segmentations are dependent on the edge image used as the boundary based term. Ideally, the edge image would have edge-voxels everywhere the user desires the label borders, but nowhere else, so that the regions would grow to meet at those borders. Placing the edge-voxels manually would amount to a manual segmentation, so a method requiring less user attention is needed. This work uses the Canny edge detecting filter [19], but any method for creating an edge-detected image could be considered, including using an automatic segmentation algorithm (e.g., watershed) to generate boundaries. Preliminary experiments have demonstrated the

need for an edge image’s localized response; replacing the edge image with a gradient image affects the region growth all around the maximum gradient, such that the resulting label boundaries do not lie on the maximum gradients.

2.5.1. Canny Edge Detecting Filter. The Canny filter is commonly available on image analysis software, and the exact form is considered optimal in its detection, localization, and minimal response to edges. The Canny image must be computed prior to use in the CA, because the filter uses nonlocal information. First, an intermediate image is created by clamping the intensities to a given range (equivalent to a window/level transform), which reduces spurious edge-voxels. Then, the Canny filter consists of a Gaussian blur, gradient magnitude calculation, edge detection from hysteresis thresholding of gradient magnitudes, and edge thinning using the gradient direction. Therefore, five parameters need to be selected: two for the clamp operation (minimum intensity, maximum intensity) and three for the Canny filter (sigma for the blur, low-gradient threshold, high-gradient threshold).

2.5.2. Parameter Search. Various methods can be used to choose the parameters, including having the user heuristically select them, employing typical values, or using a nonlinear search. This section describes an approach to search the edge detecting filter’s parameter space and find an edge image that results in an accurate segmentation via SCAPE. The benefit of such a search is that, given the image and seeds, a large number of parameter sets can be searched automatically, finding a suitable edge image without additional user attention. This section also presents specific improvements to the search for use with SCAPE.

Beasley [20] describes an algorithm that takes a seeded image and searches for an edge image that best separates the seeds according to:

$$\text{cost}_{\text{search}} = \sum_{v \in V} \left(10^{-\text{dist}_{v,2}/d_{0.1}} - 10^{-\text{dist}_{v,1}/d_{0.1}} \right), \quad (2)$$

where $d_{0.1} \in \mathbb{R} > 0$ is a constant that determines the degree to which longer distances are penalized, $\text{dist}_{v,1}$ is the shortest distance from v to the nearest seed of any label, and $\text{dist}_{v,2}$ is the shortest distance from v to the nearest seed with a different label. The term for a given voxel is most negative when the distance to the nearest seed is small and the distance to the second-nearest seed is large. Thus, a search based on this cost function tends toward parameters that result in edge-voxels separating seeds and tends away from parameters that cause extraneous edge-voxels. The previous work used graph edge lengths:

$$\text{length}_{\text{previous}}(e) = 1 + \lambda * k(e), \quad (3)$$

designed to best separate the seeds. The search consisted of a random sampling of the five-dimensional parameter space followed by a simplex search refinement. Unfortunately, the previous work is not sufficient for the needs of SCAPE, as it takes many hours to select an edge image that does not result in accurate SCAPE segmentations.

The following four improvements to that nonlinear search increase the utility of the resulting edge images for use in SCAPE. First, the cost function was adjusted by using the same edge-length calculation as in Algorithm 1 line 16, instead of assuming that graph edges not connected to an edge-voxel are unit length as in (3). This modification ensures the edge image search uses the same graph as the segmentation, biases this modified search to find an edge image that best separates the labels, and means that the SCAPE GPGPU implementation can be used to quickly calculate the distances needed in (2) by running it once for each label. Second, the seed voxels are not used in the cost function calculation. Since seed voxels are always distance zero to the nearest label, their inclusion would bias the cost function toward more edge-voxels, increasing the distance to their second-nearest label.

Third, the search space is reduced to four dimensions by fixing the low-gradient threshold in the Canny filter to three standard deviations below the mean gradient magnitude. The low-gradient threshold is used in the Canny filter's hysteresis approach as the minimum gradient magnitude for a voxel to be an edge-voxel if the gradient direction is the same as an adjacent edge-voxel. In practice, the impact of the high-gradient threshold overwhelms that of the low-gradient threshold, and setting the low gradient threshold to a very low number typically has minimal effect because image noise makes the gradient directions erratic for voxels with low gradients. This change had a minimal effect on the resulting edge images, but significantly decreased the computation time.

Fourth, to further reduce the search time, the random search of the parameter space was replaced by a more effective combination of grid- and binary-based search. This change was motivated by the observation that "good" minimum and maximum intensity parameters remove intensity variations unrelated to features separating the seeds, making those parameters relatively independent of the other two parameters. Therefore, a binary divide-and-conquer search is performed over the minimum and maximum intensity parameters. Then, for each pair of minimum and maximum intensity parameters, an evenly spaced grid of sigma and high-gradient threshold parameters is examined. Throughout this search, the five best parameter sets are kept for initializing the simplex refinement performed as the last step.

This search finds parameters such that the Canny filter creates an edge image best separating the regions grown from the seeds. Since SCAPE and the parameter search use the same shortest-path calculation on the same graph, the search effectively optimizes the segmentation based on the seeds, the image features, the framework for creating the edge image, and the segmentation algorithm. A typical workflow would be acquiring the image, manually placing seeds, using the parameter search to get an edge image, segmenting the image with SCAPE, and then interactively correcting the segmentation by adding additional seeds.

2.6. Experimental Setup. Experiments were performed to evaluate SCAPE's segmentations, the different implementations of SCAPE, and the different approaches to determining

the edge images. The data used in these experiments are 40 T1-weighted MRI images of normal brains from the Segmentation Validation Engine (SVE) [21]. The images are $256 \times 256 \times 124$ voxels with an intensity ranging 0–4095 (Figure 1). SVE also scored any segmentations in comparison with a "gold standard" for a skull-stripping segmentation, in which the brain tissue is separated from nonbrain tissue. The segmentations were compared to the gold standard using the Jaccard index, the number of true positive brain voxels divided by the sum of the true positive, false positive, and false negative. The test system for these experiments was an Intel Core i7-2600 CPU @ 3.40 GHz with 12 GB RAM, 64-bit Windows 7, and a consumer graphics card (NVIDIA GeForce GTX 580, 1536 MB memory, compute capability 2.0, CUDA compiler version 4.0).

These experiments used $\lambda = 4$ in Algorithm 1 line 16, so any graph edge with an edge-voxel for an endpoint was at least four times longer than any without. This value for λ was heuristically chosen, with tests showing negligible effect for greater values. Similarly, for this work $d_{0.1} = 1$ in (2), heuristically chosen with tests of other values showing negligible effect. These constants are different from the values used in [20] because the graph edge lengths are calculated differently.

When the following experiments used the parameter search (Section 2.5.2), two methods were compared. The first, referred to as the short search, had settings chosen to keep the time required for the search around half an hour: the binary search was stopped when the difference between the minimum and maximum intensities would have dropped below 500, two values were tested for sigma (0.15 and 0.5), and the high-gradient threshold was tested at five values (the mean gradient magnitude plus/minus 0, 0.75, and 1.5 standard deviations). The long parameter search was designed to take around four hours of search time: the binary search over the minimum and maximum intensity parameters was allowed to continue until the cost function stopped improving, three sigma values were tested (0.2, 0.75, 1.5), and the high-gradient threshold was tested at seven values (the mean gradient magnitude plus/minus 0, 0.5, 1, and 2 standard deviations). Furthermore, before the simplex in the long search, a finer grid-based search was done around each of the five parameter sets, using the same minimum and maximum intensity thresholds but varying sigma and the high-gradient threshold each by 0.2 down to the nominal value minus 0.8, and by 0.2 up to the nominal value plus 0.8. This second-stage search thereby performed up to 80 additional searches in the area of each of the results from the coarse search, and then the simplex was initialized with the five parameter sets with the best results overall.

2.6.1. Experiment on Algorithm Accuracy and Speed. First, SCAPE was compared with SCA and Graphcut. For each image, three seed images were created by manually placing seeds: seeds on a single axial slice (1-slice), seeds on three orthogonal slices (3-slice), and seeds on five slices in each of the three orthogonal directions (15-slice). Then, a segmentation was performed on each image by each algorithm

FIGURE 1: Typical data used for validation: an axial slice (a) from one of the 40 T1-weighted MRI volumes from the Segmentation Validation Engine, with the three-slice seeds overlaid ((b), brain in red, nonbrain in blue), and a 3D rendering of a resulting SCAPE segmentation (c).

for each set of seeds, a total of 360 segmentations. The SCAPE segmentations used an edge image found using the short parameter search, and the GPGPU implementation. The SCA implementation was similar to the SCAPE implementation, but without the edge image calculation terms. The Graphcut implementation was on the CPU, provided by Boykov and Kolmogorov [5], and used a Gaussian weighting function:

$$w(e) = \exp\left(-\beta * \left(\frac{(g(v_{e,i}) - g(v_{e,j}))^2}{l_{\max}^2}\right)\right), \quad (4)$$

where $\beta = 150$ (based on [9, 22], though varying β had minimal effect). For all three algorithms, the voxel intensity was used for $g(\cdot)$.

2.6.2. Experiment on SCAPE Implementation Timings. Second, the computation time and update rate were evaluated for the different SCAPE implementations. The three implementations segmented an SVE image, as well as the same image upscaled to $512 \times 512 \times 426$ (one of the the maximum volume sizes that could be successfully segmented on the GPGPU, along with $484 \times 484 \times 484$ and $1024 \times 1024 \times 106$). Blank edge images and the 3-slice seeds were used. Dijkstra's algorithm was provided by the Boost Graph Library [23].

2.6.3. Experiment on Timing and Accuracy of Edge Parameter Selection Methods. In the third experiment, different approaches to determining the edge images used by SCAPE were evaluated by user time, computation time, and segmentation accuracy. Four approaches to determining edge images were investigated: manual selection, the short parameter search, the long parameter search, and fixed parameters. For the manual selection case, for each of the 40 SVE images, the author selected the parameters creating the edge image that seemed to best separate the brain from the surrounding tissue. For the two parameter searches, each search was performed on each image for each of the three sets of seeds.

For the fixed parameters test the same values, the mean value of the manually chosen parameters were used to create all 40 edge images.

3. Results

Manually drawing the three sets of seeds took 14 ± 2 s (mean $\pm$ standard deviation) for the 1-slice set, 53 ± 7 s for 3-slice, and 440 ± 44 s for 15-slice.

3.1. Comparison of Algorithm Accuracy and Speed. SCAPE's segmentations were more accurate than those of either SCA or Graphcut (Figure 2; for Jaccard index results from 15-slice seeds: SCAPE 0.915 ± 0.013 , SCA 0.910 ± 0.014 , Graphcut 0.893 ± 0.021 ; paired t -tests $P < 0.05$). SCAPE's advantage over the next best algorithm, SCA, increased to 0.047 for 3-slice seeds and then to 0.076 for 1-slice seeds, though SCAPE's overall accuracy dropped (0.883 ± 0.030 and 0.861 ± 0.054 resp.). The segmentations performed by Graphcut were similar in accuracy to SCA and SCAPE for the 15-slice seeds, but the accuracy fell significantly as the number of seed slices was reduced, down to 0.152 ± 0.016 for 1-slice seeds. Inspection of the Graphcut 1-slice seed segmentations showed each to be labeled predominantly as brain or predominantly as nonbrain, a consequence of Graphcut's boundary-length bias. Many of the Graphcut 3-slice seed segmentations suffer from the same issue, while the others are on par with the SCA segmentation accuracies. The main effects (i.e., seeds, algorithms) and interaction effects were significantly different based on two-way ANOVA, $P < 0.0001$ each.

SCA on the GPGPU took the least time to perform the segmentations, 1.7 ± 0.4 s. The SCAPE segmentation on the GPGPU took 2.2 ± 0.5 s, the extra time being used for data transfer of the edge image and additional calculations due to the edge image. Graphcut on the CPU took 155 ± 222 s. SCA's timing varied by 10% difference between the three sets of seeds (not shown), as did SCAPE's, but Graphcut's variation in timing was an order of magnitude between the

FIGURE 2: Comparison of segmentation accuracy for 40 brain MRIs, across three algorithms and three sets of seeds (for which seeds were placed on 1, 3, or 15 image slices). The proposed algorithm, SCAPE, generated segmentations that match the gold standard better than those of Seeded Cellular Automata (SCA) and Graphcut, particularly when minimal seeds were provided. Error bars show standard deviations.

three sets of seeds (1-slice 23 ± 9.3 s, 3-slice 413 ± 220 s, 15-slice 31 ± 17 s). The SCAPE algorithm also requires an edge image, in this case determined by the short parameter search on the 3-slice seeds, for which the computation time is covered in Section 3.3

3.2. Comparison of SCAPE Implementation Timings. The GPGPU implementation was the fastest SCAPE implementation, segmenting the $256 \times 256 \times 124$ image in 2.4 s and 1229 iterations (37 s and 1241 iterations for the $512 \times 512 \times 426$), approximately 2 ms (29 ms) per iteration on average (Table 1). Of that time, copying the data from the CPU to the GPGPU and then back took 27 ms (356 ms). By altering the algorithm to perform the segmentation in groups of 50 iterations separated by there-and-back transfers of the data, the implementation was made more responsive (i.e., higher update rates) at the expense of additional time spent transferring data, resulting in an update rate of 8 Hz (0.6 Hz) and a 33% increase in segmentation time (3.1 s and 49 s resp.).

Altogether, the three improvements described for speeding up SCAPE on the CPU cut the segmentation time by over 90%. For the $256 \times 256 \times 124$ image, the segmentations took 334 s (1229 iterations) without any improvements. The single-buffering improvement reduced that to 186 s (714 iterations), while single-buffering and Yen's improvement together reduced the time to 122 s (219 iterations), and finally including the narrow-band tiles improvement resulted in 28 s (219 iterations). For the $512 \times 512 \times 426$ volume, the segmentation time went from 4975 s (1241 iterations) with no improvements to 189 s (124 iterations) with all improvements. The resulting labels were identical in all cases.

For the CPU implementation with all the improvements, the minimum update rate during the initial segmentation of the $256 \times 256 \times 124$ image was 2 Hz, with an average of 8 Hz ($512 \times 512 \times 426$ image: 0.13 Hz minimum, 0.66 Hz average). To investigate interactive ease-of-use, an initial segmentation of the smaller image was performed and then additional seeds were added. The average update rate for these corrections varied with the number of affected voxels, from 16 Hz for 2.7 million voxels changed, to 220 Hz for 3.4 thousand voxels.

The Dijkstra implementation took 12 s to segment the $256 \times 256 \times 124$ image, five times as long as SCAPE on the GPGPU, but less than half the time as SCAPE on the CPU. Unfortunately, the Dijkstra implementation was unable to load the $512 \times 512 \times 426$ image due to memory constraints. The Boost Graph Library implementation for Dijkstra's algorithm creates a separate graph data structure, so presumably a Dijkstra's algorithm implementation working directly on the array of image intensities would be successful for the larger image size.

3.3. Comparison of Timing and Accuracy of Edge Parameter Selection Methods. Of the four methods tested for determining an edge image for use in SCAPE, both parameter searches generated edge images resulting in the most accurate segmentations, demonstrating that the proposed search method is suitable for this purpose (Figure 3). The more thorough parameter search resulted in the overall highest quality (Jaccard index of 0.915 ± 0.013 for 15-slice seeds) and the best result for 3-slice seeds (0.879 ± 0.034), while the faster search was the top performer for 1-slice seeds (0.861 ± 0.054), but the two factors were not significantly different ($P = 0.65$ based on two-way ANOVA). Having the short search outperform the long search reinforces the knowledge that the cost function in (2) finds suitable edge images based on the seeds, but does not directly optimize the final segmentation. Based on all the approaches, the main (i.e., seeds, algorithms) and interaction effects were significantly different based on two-way ANOVA, $P < 0.0001$ each.

Manual parameter selection resulted in relatively low quality segmentations (no greater than 0.876 ± 0.034) suggesting that manually setting the parameters based on inspection of the generated edge images can be a difficult task for a structure with a surface as complex as the brain's. When using fixed parameters equal to the mean values from the manually selected parameters, relatively high quality segmentations resulted when 15-slice seeds were provided (0.908 ± 0.019), but not for 3-slice (0.851 ± 0.032) or 1-slice (0.820 ± 0.061) seeds. Other fixed parameter sets, for example, the means of the long search's parameters, fared no better, presumably because the MRIs intensities have high interpatient variability.

In addition to any user time or computation needed to generate the seeds, additional time may be spent determining a suitable edge image for SCAPE. The parameter searches required no additional user time but did take off-line computation time: 33 ± 5 m per image for the short search and 250 ± 51 m for the long search. The fixed parameters

TABLE 1: Comparison of segmentation time and number of iterations for different implementations of SCAPE. The parallel hardware (GPGPU) implementation has faster computation times than the single processor implementations of both FBA and Dijkstra’s algorithm. Altogether, the improvements to the CPU implementation of SCAPE reduce the segmentation time by 90%.

Image size	CPU no improve		CPU all improve		Dijkstra	GPGPU	
	time (s)	iter.	time (s)	iter.	time (s)	time (s)	iter.
$256 \times 256 \times 124$	334	1229	28	219	12	2.4	1229
$512 \times 512 \times 426$	4975	1241	189	124	*	37	1241

* Dijkstra implementation failed due to memory constraints. The test system had 12 GB of RAM.

FIGURE 3: Comparison of segmentation accuracy resulting from SCAPE using edge images from four different methods and three sets of seeds. The long and short parameter searches both determined edge images that resulted in high quality SCAPE segmentations. Manually selecting parameters resulted in relatively low quality segmentations. Using the same “fixed” parameters for each edge image did reasonably well with more seeds. Quality measured by comparing with a gold standard using Jaccard index, for which the scale is shown ranging from 0.7 to 0.95 to show more detail. Error bars show standard deviations.

took no additional time once the parameters were chosen. Finally, the manual selection took 145 ± 83 s of user attention per image, but no additional computation time.

4. Discussion

This paper presents a novel segmentation algorithm, Seeded Cellular Automata Plus Edge detector (SCAPE), combining a region-growing algorithm with edge-detected images to improve behavior at weak borders. This combination reduces the amount of user interaction necessary for quality segmentations, while allowing real-time user guidance. Experimental results from a typical MRI segmentation task demonstrated higher segmentation accuracy compared with Seeded Cellular Automata and a typical implementation of Graphcut. Furthermore, three software implementations of SCAPE were examined, with a version running on a consumer graphics card performing segmentations of $256 \times 256 \times 124$ images in 2.2 s on average, and a single-processor

version having an update rate suitable for interactive user corrections (2–220 Hz, same image size).

SCAPE requires an edge-detected image to use as a boundary term, and various methods were investigated for generating edge images using the Canny edge detecting filter. The presented parameter search resulted in high-quality segmentations and required no additional user time, but took half an hour or more of computation time. A suggested work-flow would be placing seeds on some number of image slices (e.g., three slices taking less than a minute of manual scribbles), running the short parameter search for half an hour of offline computation, and finally user correction of the segmentation (using the interactive CPU implementation) until the segmentation is accurate. For modalities in which the edge image parameters for a segmentation task may be consistent, such as CT, fixed parameters can be used to bypass the need for the parameter search. Alternatively, other segmentation tasks have responded well to manual selection of the parameters, though the task demonstrated in this paper was not so amenable.

Determining good edge images for SCAPE is an open area of investigation. Other edge image filters should be explored, for example, the SUSAN filter, as should a parallel hardware implementation of the Canny filter to reduce the necessary computation time. Further investigations of the edge image parameter search are also needed, as the edge images found by the long parameter search had better costs but did not generate more accurate segmentations in comparison with the short-parameter search.

Further comparisons should be performed between SCAPE and the popular supervised segmentation algorithms, for example, Level Sets, Graphcut, Random Walker. Though Graphcut on the CPU took an order of magnitude more time in this experiment, if Graphcut was implemented on parallel hardware the segmentation times could be expected to be dropped by 90%, bringing it into the same range as SCAPE on parallel hardware. Also warranted is a more thorough investigation of the relationship between minimal user attention, the graph edge lengths used for Graphcut, and Graphcut’s boundary-length bias.

A main thrust of this work is the tradeoff between user attention and offline computation time. The more time a user spends placing seeds, the more likely any of the algorithms will provide an accurate segmentation. A decrease in user attention (placing seeds or correcting initial segmentations) can be offset by an increase in computation

time, for example, using SCAPE and an offline parameter search. Further investigation of minimal attention for accurate segmentations could significantly further clinical use of image segmentations, particularly investigation of supervised algorithms such as the one proposed in this work. In particular, a study is planned on user-guided corrections following an initial segmentation, testing the increase in segmentation accuracy as a function of user attention.

Acknowledgment

This work grew out of image analysis advice and software framework provided by Dr. Christopher Wagner and Ortho-run, Ltd.

References

- [1] M. Kass, A. Witkin, and D. Terzopoulos, "Snakes: active contour models," *International Journal of Computer Vision*, vol. 1, no. 4, pp. 321–331, 1988.
- [2] D. Cremers, M. Rousson, and R. Deriche, "A review of statistical approaches to level set segmentation: integrating color, texture, motion and shape," *International Journal of Computer Vision*, vol. 72, no. 2, pp. 195–215, 2007.
- [3] E. N. Mortensen and W. A. Barrett, "Interactive segmentation with intelligent scissors," *Graphical Models and Image Processing*, vol. 60, no. 5, pp. 349–384, 1998.
- [4] A. X. Falcao, J. K. Udupa, S. Samarasekera, S. Sharma, B. E. Hirsch, and R. D. A. Lotufo, "User-steered image segmentation paradigms: live wire and live lane," *Graphical Models and Image Processing*, vol. 60, no. 4, pp. 233–260, 1998.
- [5] Y. Boykov and V. Kolmogorov, "An experimental comparison of min-cut/max-flow algorithms for energy minimization in vision," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 26, no. 9, pp. 1124–1137, 2004.
- [6] C. Kauffmann and N. Piché, "Seeded ND medical image segmentation by cellular automaton on GPU," *International Journal of Computer Assisted Radiology and Surgery*, vol. 5, no. 3, pp. 251–262, 2010.
- [7] S. K. Weeratunga and C. Kamath, "An investigation of implicit active contours for scientific image segmentation," in *Visual Communications and Image Processing 2004*, vol. 5308 of *Proceedings of SPIE*, pp. 210–221, January 2004.
- [8] A. K. Sinop and L. Grady, "A seeded image segmentation framework unifying graph cuts and random Walker which yields a new algorithm," in *Proceedings of the 11th IEEE International Conference on Computer Vision (ICCV '07)*, pp. 1–8, October 2007.
- [9] E. Moschidis and J. Graham, "A systematic performance evaluation of interactive image segmentation methods based on simulated user interaction," in *Proceedings of the 7th IEEE International Symposium on Biomedical Imaging: From Nano to Macro (ISBI '10)*, pp. 928–931, April 2010.
- [10] R. Adams and L. Bischof, "Seeded region growing," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 16, no. 6, pp. 641–647, 1994.
- [11] J. Freixenet, X. Munoz, D. Raba, J. Mart, and X. Cuf, "Yet another survey on image segmentation: region and boundary information integration," *Computer Vision-ECCV*, vol. 2002, pp. 21–25, 2002.
- [12] S. Even, *Graph Algorithms*, vol. 606, Computer Science Press, Rockville, Md, USA, 1979.
- [13] R. Bellman, "On a routing problem," *Quarterly of Applied Mathematics*, vol. 16, no. 1, pp. 87–90, 1958.
- [14] M. Gardner, "Mathematical games: the fantastic combinations of John Conway's new solitaire game 'life,'" *Scientific American*, vol. 223, no. 4, pp. 120–123, 1970.
- [15] V. Vezhnevets and V. Konouchine, "Growcut: interactive multi-label ND image segmentation by cellular automata," in *Proceedings of the Graphicon*, pp. 150–156, 2005.
- [16] J. Yen, "Algorithm for finding shortest routes from all source nodes to a given destination in general networks," *Quarterly of Applied Mathematics*, vol. 27, no. 4, pp. 526–530, 1970.
- [17] E. W. Dijkstra, "A note on two problems in connexion with graphs," *Numerische Mathematik*, vol. 1, no. 1, pp. 269–271, 1959.
- [18] U. Meyer and P. Sanders, "Δ-stepping: a parallelizable shortest path algorithm," *Journal of Algorithms*, vol. 49, no. 1, pp. 114–152, 2003.
- [19] J. Canny, "A computational approach to edge detection," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 8, no. 6, pp. 679–698, 1986.
- [20] R. Beasley, "Finding the best edge image," in *Proceedings of the International Conference on Image Processing, Computer Vision, and Pattern Recognition*, pp. 504–510, July 2011.
- [21] D. W. Shattuck, G. Prasad, M. Mirza, K. L. Narr, and A. W. Toga, "Online resource for validation of brain segmentation methods," *NeuroImage*, vol. 45, no. 2, pp. 431–439, 2009.
- [22] L. Grady and M. Jolly, "Weights and topology: a study of the effects of graph construction on 3D image segmentation," in *Proceedings of the Medical Image Computing and Computer-Assisted Intervention (MICCAI '08)*, pp. 153–161, 2008.
- [23] J. Siek, L. Lee, and A. Lumsdaine, *Boost Graph Library: User Guide and Reference Manual*, Addison-Wesley Professional, 2001.