

Random Walker Segmentation Algorithm for Medical Images

Aznul Qalid Md Sabri

MSc. ViBot, - University of Burgundy, 71200, Le Creusot, France

ABSTRACT

This paper describes the details of the implementation of a semi-automatic segmentation algorithm that takes user defined seeds as input and segments out regions in an image. The main aim is to develop a piece of software for extracting tumors from 3D MRI and PET images of human brains efficiently. The main computational burden in performing the segmentation lies in solving a large sparse linear system. The algorithm was implemented using preconditioned conjugate gradient method that is generalized for images with arbitrary number of dimensions with Modified Incomplete Cholesky factorization identified as the preconditioning method that enhance the convergence rate the most. Using a negative value of alpha, a relaxation parameter in the factorization, proves to be most effective for typical medical images. Initial results had shown that the implementation is efficient for 2D images of reasonable sizes. Cost of storing the large sparse Laplacian matrix which represents the graph (graph theory is used to represent the image) is significantly reduced by only storing $(N+1)$ number of diagonal bands with N being the number of connectivity chosen for the graph. This can be further minimized by only storing the lower triangular part of the Laplacian matrix with $(N/2+1)$ number of diagonal bands since the Laplacian matrix is symmetric. Finally this work had also identified further measures that can be taken to ensure a much more efficient implementation of the segmentation algorithm, which is, to use the multigrid approach. However, due to time constraint, this approach was not explored.

KEY WORDS: Image segmentation, random walks, conjugate gradient, modified incomplete cholesky

phenomenon in which, trivial or minor similarities between the image elements and the seed elements are included in the final segmentation result.

1 INTRODUCTION

This paper documents the implementation of a semi-automatic segmentation algorithm that takes user defined seeds as input and is able to segment out tumors in an image. The algorithm is implemented based on the work of Grady [1]. Grady has also made his code available at [2]. Some of the previous work known is the graph-cut algorithm [3], or the region – growing algorithm.

The main goal is to explore the implementation of a segmentation algorithm that assigns labels based on the probability of each pixel in the image to reach any of the corresponding labels. Another goal of the work is to discover improvements that can be made to the existing algorithm in order to increase the robustness as well as its usability in terms of medical image segmentation.

2 PREVIOUS APPROACHES

It is important to note that the the random walker is a graph based segmentation algorithm, and in this section, some of its predecessors are reviewed.

Intelligent Scissors proposed by Mortensen & Barret [4] introduce a concept known as “live-wire tool” that allows the users to select an optimal boundary interactively by selecting an optimal boundary segment and immediately displaying the minimum cost path from that selected position to the specified seed points in the image.

Interactive Graph Cuts introduced by Boykov & Jolly [5] impose a hard constraint on the segmentation by indicating certain pixels (seeds) that are part of the object or the background, and soft constraints using boundary and region information before applying graph-cuts to find the globally optimal segmentation of the N -dimensional image.

The first two methods are attractive since a general segmentation may be performed given sufficient user interactions, and it is simple to generalize the methods to images with higher dimensionalities. The drawback is that both methods suffer from the “small cut”

GrabCut introduced by Kolmogorov [6] extends the functionality of a conventional graph-cut by introducing an iterative estimation of the color model that alternates between estimation and parameter learning, replacing the one-shot minimum cut estimation algorithm in graph-cut. This method enables a user interface that requires less user interaction, in which, the user only needs to draw a box around the area of the image to be segmented. This means, in initializing the trimap, the user does not have to specify the foreground seed pixels. At the start of the iterative minimization, the foreground region is initialized to an empty set, \emptyset , and the unknown region is initialized to the complement of the background. Although this interface requires less user interaction, it is discovered that the approach is always not sufficient to capture the desired object, and requires further editing using the standard graph-cut method. Although this interface requires less user interaction, it is discovered that the approach is always not sufficient to capture the desired object, and requires further editing using the standard graph-cut method.

The Random walker algorithm tries to capture the advantages of its predecessors while at the same time overcoming the disadvantages. It tries to capture the benefit of an interactive segmentation algorithm, in which it is able to locate weak or missing boundaries, be robust to noises in the image, identify multiple objects simultaneously (i.e. segmentation based on multiple labels) and avoid small or trivial solutions, i.e. it is able to avoid “small cut” phenomenon.

3 METHODOLOGY

In the following subsections the theoretical aspects of the algorithm are discussed in detail before proceeding with the implementation details.

3.1.1 OVERVIEW OF THE RANDOM WALKER ALGORITHM

Random Walker is initialized with a set of voxels taking one of a predefined set of labels. The probability that a random walker starting at a given unlabeled voxel will first reach a voxel of a particular label is computed, based on solving a so-called Dirichlet problem [8]. A Dirichlet integral may be defined as

$$D[u] = \frac{1}{2} \int_{\Omega} |\nabla u|^2 d\Omega \quad (1)$$

for a field u and region Ω . Meanwhile, a harmonic function is a function that satisfies the Laplace equation

$$\nabla^2 u = 0 \quad (2)$$

A Dirichlet problem is defined as the problem of finding a harmonic function subject to its boundary values. The harmonic function that satisfies the boundary conditions minimizes the Dirichlet integral.

In the case of the random walker algorithm, the Laplacian matrix is used to model the relationship between each of the nodes on the graph. This matrix is then used to model the harmonic function that satisfies the boundary condition (build the linear system), thus, solving the Dirichlet problem.

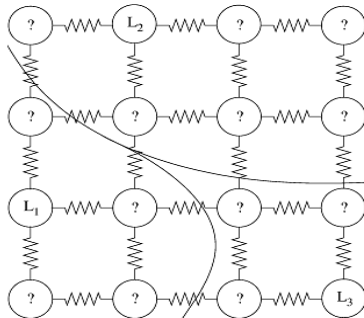


Fig. 1: An example of a “circuit” with three different labels

Referring to Fig. 1, using a circuit theory as an analogy, edge weights are resistors between each of the nodes (pixels) and model the influence of nodes upon each-other. They are stored in a Laplacian matrix, a matrix representation of the relationship between the nodes (the edge weights), along with a diagonal band containing the degree (number of edges incident on the node) of each node. The steady state potential at each node in figure 1 represents a probability of the node belonging to a particular label. To get the probability (steady state potential) of each node belonging to each of the label, the linear system is solved once per label with the seed nodes of other labels grounded (i.e. the potentials of other seed nodes are fixed to zero). The probabilities for each label is then super positioned, with the maximum probability value for each node corresponds to the label it belongs to.

For a $K * K$ Laplacian matrix, the number of elements stored is, $K * (N + 1)$, where K is the number of rows in the matrix (and number of voxels in the image), and N is the connectivity to local neighbors.

Once the Laplacian matrices have been obtained, the probabilities may be computed using standard linear algebra.

The sequence of steps followed by the Random Walker algorithm is:

- Input: Image with the dimension D , and a set of labeled seed points.
- Build Laplacian matrix based on the computed edge weights

- Partition Laplacian matrix, $L = \begin{pmatrix} L_M & B \\ B^T & L_U \end{pmatrix}$. Subscript M

and U stands for marked (labeled) and unmarked (unlabeled) pixels. The B matrix is part of the Laplacian matrix corresponding to the labels.

- Setup the linear system, $L_U X = -B^T M$
 - L_U : Edge weights for the unlabeled pixels ($n * n$)
 - M : Set of labels for the seed points ($l * l$), l is the number of labels.
 - $-B^T$: Edge weights corresponding to the labels ($n * l$)
 - X : the probability for each pixel being a member of the labels ($n * l$)
- Solve the linear system

The terms and symbols used above, as well as the steps will be clarified in the following subsections.

3.1.2 BUILD A LAPLACIAN MATRIX BASED ON EDGE WEIGHTS

A Laplacian matrix is a matrix representing a graph. A graph, G , consists of a set of vertices, V , and edges, E , i.e. $G = \{V, E\}$. Explicitly, the vertices are the graph’s representation of the image’s pixels and the edges are the representation of each pixel’s relationship with its surrounding neighbors (edge weights). Typical relationships considered are 4 or 8 connectivity for 2D images, and 6 or 26 connectivity for 3D images.

The Laplacian matrix can be defined as

$$L_{ij} = \begin{cases} d_i & \text{if } i=j \\ -w_{ij} & \text{if } v_i \text{ and } v_j \text{ are adjacent nodes} \\ 0 & \text{otherwise} \end{cases} \quad (3)$$

Referring to (3), L_{ij} refers to each entry in the Laplacian matrix. d_i refers to the degree of vertex i , i.e. the number of connection between that particular vertex, v_i and its neighboring vertices. w_{ij} refers to the edge weight at the ij -th position. The result of (3) is a sparse, symmetric and positive semi-definite Laplacian matrix.

A Gaussian weighting function is used to select edge weights:

$$\omega_{ij} = \exp(-\beta(g_i - g_j)^2) \quad (4)$$

Where both g_i and g_j are the image intensity values at pixel i and j . (4) can be modified to apply texture information, filter coefficients or other image features.

3.1.3 SET UP THE LINEAR SYSTEM

If rows are interchanged as necessary, the Laplacian matrix may be partitioned into four sub matrices:

$$\begin{pmatrix} L_M & B \\ B^T & L_U \end{pmatrix} \quad (5)$$

in which, the subscripts M and U stand for “marked”, (labeled), and “unmarked”, (unlabeled), pixels. The B matrix is part of the Laplacian matrix corresponding to the labels.

The Laplacian matrix (5) defines a system of linear equations, which may be used to solve the following system (solving the Dirichlet problem):

$$L_U X = -B^T M \quad (6)$$

In the system of equations defined by (6), X is the probability that a voxel belongs to a particular label. X has as many columns as there are labels. Only the sub-matrix L_U is needed because the solution for the marked nodes is already known.

M is the boundary condition for the Dirichlet problem, containing the labels for each of the seed points defined by (7). M has as many columns as there are labels, and as many rows as there are seeds. The set of labels for the seed points is defined as a function, $Q(v_j) = s$.

$$m_j^s = \begin{cases} 1, & \text{if } Q(v_j) = s \\ 0, & \text{if } Q(v_j) \neq s \end{cases} \quad (7)$$

In (7), m_j^s refers to each of the entry in M for the j -th column of every row for each seed pixels, s .

Using the probability obtained by solving the system of linear equations, each pixel in the image is then assigned to its corresponding labels for which it has the highest probability.

3.2 IMPLEMENTATION DETAILS

The sparse nature of the linear system defined in (3), may be exploited to improve computational and storage efficiency. Even with these performance gains, the size of L_U in (5) makes it impractical for segmenting 3D images without further optimizations. The demonstration program provided by Grady at [2], supplies a relatively small test image (256*256), which is atypical of real-world problems. Since the supplied image is small, solutions are obtained within a reasonable time using MATLAB's backslash direct solver. The remainder of this section explores a number of methods for further optimization.

3.2.1 GENERATING THE EDGE WEIGHTS

A predefined function, "makeweights.m" (available within the "Graph Analysis Toolbox" [2]), by the author of the paper is used to compute the edge weights. This function computes weights for a point and edge list based upon element values and Euclidean distance. The distances are then normalized between [0, 1], before the Gaussian weights are computed as mentioned in (4).

3.2.2 SETTING UP THE LINEAR SYSTEM

Equation (6) has to be solved to obtain the final probability of each pixel belonging to a label. Prior to that, the linear system needs to be set up. This is done by first building the Laplacian matrix based on (3). Using (3), $N+1$ diagonal bands are produced in the Laplacian matrix, where N is the connectivity that had been chosen.

Since the Laplacian matrix is symmetric, only the upper triangular part of the matrix needs to be stored. Fig. 2 represents a Laplacian matrix where color indicates the value at each location. Most of the values in the matrix are zero. Hence, each band in the matrix may be stored as a column-wise vector. The zero elements outside of the bands in the matrix need not be stored. Although only the upper triangular part of the matrix need be stored, requiring three vectors for each band in the upper part, all 5 (in 2D) diagonal bands were stored to check for errors.

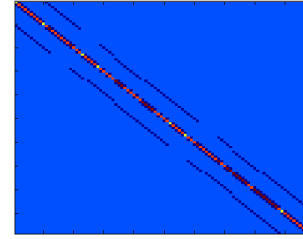


Fig. 2: A visual representation of a laplacian matrix for a 10x10 image, with 4 connectivity

3.3 SOLVING THE LINEAR SYSTEM

Due to the large sparse nature of the linear system, direct solvers are too inefficient for practical use.

Under conditions where the A matrix size is not large, e.g. for a 256x256 image, A (i.e. L_U from (5)) can be decomposed into its lower triangular part, H_L , and upper triangular part, H_U , efficiently.

Using both this triangular matrices, the linear system can be solved directly with the following two solves:

$$A * x = b; \quad // \text{To solve for } x \quad (8)$$

$$H_L * y = b; \quad // \text{First solve for } y, \text{ a "dummy" vector} \quad (9)$$

$$H_U * x = y; \quad // \text{Next, solve for } x \quad (10)$$

Note that the use of the two triangular matrices allows the problem to be solved efficiently using forward and backward substitution [9]. Since the Laplacian matrix is symmetric, Cholesky decomposition [12] can be used to factorize the matrix, where, $H_U = H_L^T$.

This approach (direct solver) might be suitable for 2D images of reasonable sizes. For a typical 2D medical image, e.g. a slice of a CT scanned image with the size of 256^2 (65536 pixels), the Laplacian matrix (from (3)) needs to model the interaction between 65536 different nodes. The dimension of the Laplacian matrix will then be 65536^2 , which can still be handled efficiently using direct solvers.

However, often there is a need to segment 3D images, e.g. for a typical PET image of the size $128 * 128 * 100$ pixels (1,638,400 pixels). In this case, the size of the Laplacian matrix (size of the linear system) will be extremely large (the Laplacian matrix needs to model the interaction between millions of nodes) and is beyond the capability of any direct solving method as it will take an impractical amount of time to both factorize the matrix as well as to solve it using direct solvers.

Referring to Fig. 3, H_U contains fill-in values in between the original diagonal bands (red circle in figure 4) of the A matrix. Fill-in values are new non zero values in the Cholesky factor that do not exist in the original A matrix. This might not seem significant in small matrices, however, if the size of the matrix is large, the number of non-zero values in H_U , can be significantly larger than the number of non-zero values in the original A matrix.

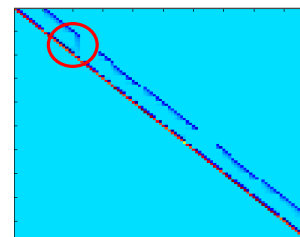


Fig. 3: A visual representation of an upper triangular matrix, H_U , cholesky factor of A from $A * x = b$

Thus, solving the system is computationally expensive for two reasons. Firstly, it is expensive to factorize the A matrix since zero values in the A matrix need to be checked, whether there are any fill-in values at that specific location. Secondly, the structure of the sparse matrix (containing only diagonal bands) can no longer be exploited, and a full matrix needs to be stored and traversed during computation.

3.3.1 USAGE OF ITERATIVE SOLVER

The *conjugate gradient method* has been chosen as the iterative solver of choice for the linear system due to the fact that it is an algorithm which is developed with the original purpose of handling symmetric–positive definite matrices [10]. Moreover, the algorithm is simple to implement, and is widely used. Both [11] and [12] supply a number of algorithms along with pseudo-code that handle large sparse linear system, including the conjugate gradient method.

Using conjugate gradient alone is impractical, because the number of iterations required to solve the linear system is proportional to the number of unknowns. However, there is a method to enhance the convergence rate of an iterative solver. For detailed explanation of the conjugate gradient, please refer to [10].

3.3.2 PRECONDITIONED CONJUGATE GRADIENT METHOD

In order to enhance the convergence rate of the iterative solver, the preconditioned conjugate gradient method (PCG) is used. This is done by using a “preconditioner” in the standard conjugate gradient method.

In using a preconditioner, instead of solving $Ax = b$, both the left preconditioned side and the right preconditioned side are altered by applying a “preconditioner” matrix, M .

$$M^T A X = M^T B \quad (27)$$

By improving the condition number of the coefficient matrix ($M^T A$), the accuracy of the solution of the linear system is improved by reducing the effect of the coefficient matrix on the final solution. This reduces the number of iteration for the solver to converge to a user required tolerance.

However, the preconditioner is not applied directly on A . Referring to Fig. 4, while using PCG, there is a need to perform a preconditioner solve for each iteration. The following is the pseudo-code of the preconditioned conjugate gradient algorithm taken from [13] which is similar to the standard conjugate gradient method with an addition of performing a preconditioner solve at every iteration.

This solving step is similar to the steps mentioned in (8), (9) and (10). This means that computing the preconditioner solve must be efficient since it is done at every iteration. Several different types of preconditioners are detailed in the following subsections.

3.3.3 Jacobi Preconditioner

A popular choice for a preconditioner is a Jacobi preconditioner. The Jacobi preconditioner takes the form where it is a diagonal matrix containing the diagonal elements of the A matrix. Using a Jacobi “preconditioner”, the preconditioner is simply the pseudo-inverse of a diagonal. Using a Jacobi preconditioner, the number of iterations needed is reduced by half.

This is still not efficient enough for the needs of the application. Thus, more sophisticated preconditioning methods are investigated.

3.3.4 Incomplete Cholesky Preconditioner, IC(0)

Cholesky factorization [9] decompose the A matrix into its lower triangular factor, H_L , where the upper triangular factor, H_U , is simply the transpose of H_L .

In incomplete Cholesky factorization, the fill-in values are discarded based on a specified set of positions, \mathcal{S} , in the original A matrix (i.e. L_U from (5)). The set \mathcal{S} is usually taken to encompass all positions (i, j) where $a_{i,j} \neq 0$. A position that is zero in A but non-zero in Cholesky factorization is called a fill position. All fill positions that are outside \mathcal{S} , are said to be “discarded”. Often, \mathcal{S} is chosen to coincide with the set of non-zero positions in A , discarding all fill-ins. This type of factorization is called the incomplete Cholesky factorization level zero (IC(0)).

The algorithm to perform this factorization is taken from [9]. Based on [9], using this method the convergence rate is supposedly to improve to $O(N^{1/2})$.

More importantly, since this factorization method produces a triangular factor that has the same sparse structure of the A matrix, the preconditioner solve needed for every iteration in PCG can be solved efficiently using forward and backward substitution.

```

r0 = b - Ax0 %initial residual
z0 = M-1r0
p0 = z0 %initial conjugate direction
k = 0
for k = 0, 1, 2...n
    αk = rkTrk / pkTApk %updating the search parameter
    xk+1 = xk + αkpk %updating the solution
    rk+1 = rk - αkApk %updating the residual
    (if rk+1 is sufficiently small, then exit loop)
    zk+1 = M-1rk+1 %preconditioner solve
    βk+1 = rk+1Tzk+1 / rkTzk
    pk+1 = zk+1 + βkpk %updating the conjugate direction
    k = k + 1
end

```

Fig. 4: Pseudo-code of the preconditioned conjugate gradient method

Next, a slightly more sophisticated version of the Cholesky factorization is investigated in order to view its effects on the convergence rate.

3.3.5 Modified Incomplete Cholesky Preconditioner (MIC(0))

There are variations of modified incomplete factorization as mentioned in [12]. However, a factorization method that is able to retain the original sparse structure of the A matrix had been chosen due to reasons stated in the previous sections.

This can be accomplished using a method known as modified incomplete Cholesky level zero (note that zero stands for no fill-ins).

The algorithm is similar to the normal incomplete Cholesky factorization; however, instead of simply discarding the fill-in values, the values are subtracted from the corresponding diagonal elements. The effect of doing this is to model to a certain extent the effect of complete factorization where all of the fill values are retained, and a perfect factorization of the A matrix is accomplished.

A slight variation of the modified incomplete factorization is to introduce a relaxation parameter, α . Here the fill value is multiplied with α , before being subtracted from the diagonal elements. This has the effect of controlling the modification made on the diagonal elements. For more details on the effect of α on the factorization, refer to [12].

4 EXPERIMENTAL RESULTS

The experiment is divided into *two* parts. The first is done to determine the suitable factorization method for the matrix A , to be paired with the preconditioned conjugate method (PCG) in order to solve (6).

The second is done to model the effect of alpha, α , the relaxation parameter in the modified incomplete Cholesky factorization, on the convergence rate of PCG.

The stopping condition of the PCG code for both parts is the norm of the residual to a specified tolerance, which is chosen to be 1×10^{-5} .

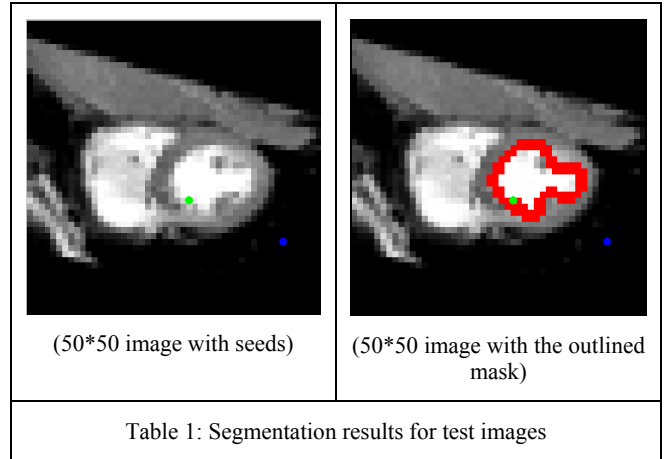
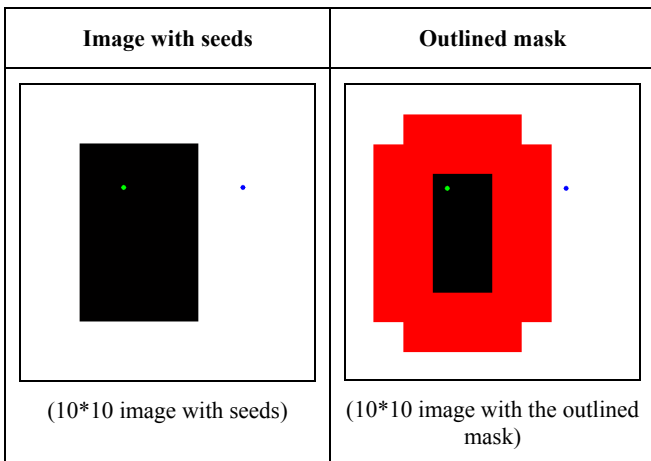
The terms (abbreviations) used in the following subsections are explained as follows:

- PCG : Preconditioned Conjugate Gradient Method
- IC(0) : Incomplete Cholesky Level Zero, refer section 3.3.4.
- MIC(0) : Modified Incomplete Cholesky Level Zero, refer section 3.3.5.
- alpha, α : a relaxation parameter used along with MIC(0), refer section 3.3.5.
- Iter. : Iterations

4.1 DETERMINING THE SUITABLE FACTORIZATION METHOD

In this part the experimental result using all the preconditioning methods discussed above is presented. To get the experimental results efficiently, the problem is tested using two images.

First, the images and its segmentation results are presented.



Next, the number of iterations as well as the computation time taken to perform the segmentation using each of the preconditioning methods discussed is presented.

Preconditioner/ Image	10*10 Image	50*50 Image
Diagonal	#Iter. : 71 #Time: 0.023s	#Iter. : 4881 #Time: 13.87s
IC(0)	#Iter. : 27 #Time: 0.014s	#Iter. : 634 #Time: 1.88s
MIC(0) ($\alpha = 1$)	#Iter. : 8 #Time: 0.003s	#Iter. : 399 #Time: 0.92s
MIC(0) ($\alpha = 0.95$)	#Iter. : 21 #Time: 0.018s	#Iter. : 833 #Time: 2.46s
MIC(0) ($\alpha = 0.90$)	#Iter. : 21 #Time: 0.018s	#Iter. : 739 #Time: 2.02s
MIC(0) ($\alpha = -1$)	#Iter. : 25 #Time: 0.018s	#Iter. : 624 #Time: 1.85s
MIC(0) ($\alpha = -0.95$)	#Iter. : 25 #Time: 0.017s	#Iter. : 659 #Time: 1.88s
MIC(0) ($\alpha = -0.90$)	#Iter. : 28 #Time: 0.016s	#Iter. : 659 #Time: 1.88s
Table 2: Number of iterations and the computation time for different preconditioning methods		

4.2 Modeling the effect of alpha, α , with MIC(0) on the convergence rate of PCG

Operating System

Windows Vista Home Premium Service Pack 1

Testing machine specification:

Dell Inspiron 1525, Intel(R) Core2 Duo CPU T7250 @ 2.00 GHz, 4.00 GB (RAM)

Language and compiler:

C++, running on CodeBlocks IDE, GNU GCC Compiler version 3.4.5 (mingw-vista special)

Testing data

A typical medical image, axial view of a slice of CT scanned image with 256*256 pixels.

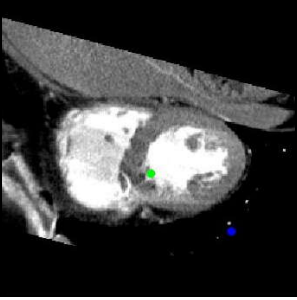
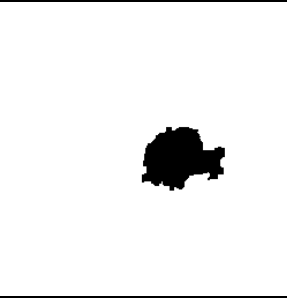
Image with seeds	Output Mask
 (256*256 image with seeds)	 (256*256 output mask image)

Table 3: Input image with its output mask

Value of alpha, α	Number of iterations taken for PCG to converge, with time
-0.95	#Iter. : 2941 #Time: 217s
-0.97	#Iter. : 2929 #Time: 219s
-0.99	#Iter. : 2928 #Time: 214s
-1.0	#Iter. : 2929 #Time: 218s
1.0	#Iter. : 3742 #Time: 268s
0.99	#Iter. : 3275 #Time: 259s
0.97	#Iter. : 3385 #Time: 261s
0.95	#Iter. : 3410 #Time: 261s

Table 4: Number of iterations and computation time for different values of alpha

5 RESULTS AND CONCLUSIONS

The results of the first part of the experiment shows that the “MIC(0) alpha = 1” is the best preconditioning method for solving the linear system. For the second part of the experiment, it is important to note that undersized images have less of an effect on the weights, while large images introduce more variation in the weights calculated using (4). Therefore, when the weights are fairly uniform as in the case of the first part of the experiment, alpha = 1, produces better convergence rate. However, for strongly varying weights (as one might expect from an image), using a negative value of alpha can be more effective. This is proven by the results obtained from the second part of the experiment.

Since the main computational burden involved in solving the random walker problem lies in solving a large sparse linear system, using a direct solver is not efficient, since this will cost an impractical amount of time. Therefore the preconditioned conjugate gradient method is implemented and it is generalized for images with arbitrary number of dimensions. The Modified Incomplete Cholesky factorization [12] along with a negative value for the relaxation parameter had been identified as producing the most desirable results.

The cost of storing the Laplacian matrix which is used to represent the graph can be minimized by only storing $(N+1)$ diagonal

bands, with N being the number of connectivity chosen for the graph. This can be further minimized by only storing the lower triangular part of the Laplacian matrix with $(N/2+1)$ diagonal bands, since the Laplacian matrix is symmetric. Thus, the burden of storing the whole sparse Laplacian matrix is reduced significantly.

It is frequently mentioned in the literature that the “multigrid” algorithm is at the moment the fastest iterative solving method [13] for large linear systems. Moreover, Grady has developed a version of “multigrid” specifically tailored to the random walker problem [14]. A “multigrid” implementation of the random walker algorithm will be explored in future work.

It is also possible to solve the problem by iteratively expanding the region under consideration. Instead of solving the whole linear system, a small portion of the labelled image is taken, and the random walker algorithm is performed on that particular portion. The segmented (labelled) portion is then used as seed pixels, and the random walker algorithm is performed again on a larger area. This is repeated until the whole image is segmented.

Finally, once the desired efficiency is obtained in the segmentation, the use of multiple modalities to improve the segmentation’s performance will be examined.

6 REFERENCES

1. Leo Grady, “Random Walks for Image Segmentation,” IEEE Trans. On Pattern Analysis and Machine Intelligence, Vol. 28, No. 11, pp. 1768-1783, Nov., 2006.
2. <http://cns-web.bu.edu/~lgrady/>
3. M. X. Goemans, D. P. Williamson, “Improved approximation algorithms for maximum cut and satisfiability problems using semidefinite programming,” Journal of the ACM, 42, 6 (Nov. 1995), 1115-1145
4. E. Mortensen, W. Barrett, “Interactive Segmentation with Intelligent Scissors,” Graphical Models in Image Processing, Vol. 60, no. 5, pp. 349-384, 1998.
5. Y. Boykov, M.-P. Jolly, “Interactive Graph Cuts for Optimal Boundary and Region Segmentation of Objects in N-D Images,” Proc. International Conference Computer Vision, pp. 105-112, 2001.
6. C. Rother, V. Kolmogorov, A. Blake, “Grab Cut—Interactive Foreground Extraction Using Iterated Graph Cuts,” ACM Trans. Graphics, vol. 23, no. 3, pp. 309-314, 2004.
7. J. Shi, J. Malik, “Normalized Cuts and Image Segmentation,” IEEE Trans. Pattern Analysis and Machine Intelligence, vol. 22, no. 8, pp. 888-905, Aug. 2000.
8. S. Axler, P. Gorkin, K. Voss, “The Dirichlet problem on quadratic surfaces,” Mathematics of Computation 73 (2004), 637-651.
9. G. Golub and C. Van Loan, “Matrix Computations”, 3rd ed. The Johns Hopkins University Press, 1996.
10. Jonathan Richard Shewchuk, An Introduction to the Conjugate Gradient Method Without the Agonizing Pain, August 1994
11. D. Watkins, “Fundamentals of Matrix Computations,” p. 84.
12. R. Barrett, M. Berry, T. F. Chan, J. Demmel, J. M. Donato, J. Dongarra, V. Eijkhout, R. Pozo, C. Romine and H. Van der Vorst, “Templates for The Solution of Linear Systems: Building blocks for Iterative Methods”.
13. Restrepo, Juan. "The Conjugate Gradient Method." 12 Apr. 2003. Mathematics Department, University of Arizona. 29 May 2009
<http://www.physics.arizona.edu/~restrepo/475A/Notes/sou_rcea/node74.html>.