Graph Cuts, *Caveat Utilitor*, and Euler's Bridges of Königsberg

Release 0.00

N. J. Tustison, P. Yushkevich, Z. Song, and J. C. Gee

November 14, 2008

Penn Image Computing And Science Laboratory
University of Pennsylvania

Abstract

Graph-based algorithms have enjoyed renewed interest for solving computer vision problems. These algorithms have been the subject of intense interest and research. In order to maintain the ITK library *au courant*, we developed a framework for graph-based methods of energy minimization in ITK which employ energy functions derived within a Markov Random Field (MRF) context. This required not only the implementation of the energy minimization methodology but also the more general requirement of introducing graph-related data structures into ITK which can be used for other graph-based algorithms pertinent to future extensions of the ITK library.

Please note that some of the algorithms described in this paper may be covered by patents and, as such, it is incumbent upon the user to seek licenses before building the binaries which utilize this code. Also note that "research use" is not exempt from acquiring such licenses. The only exemption from patent restrictions is "...amusement, to satisfy idle curiosity, or for strictly philosophical inquiry."[1]

Latest version available at the Insight Journal [http://hdl.handle.net/1926/????]

Contents

[1] We are indebted to Luis Ibanez for the direction provided regarding the various patent issues.

1 Introduction

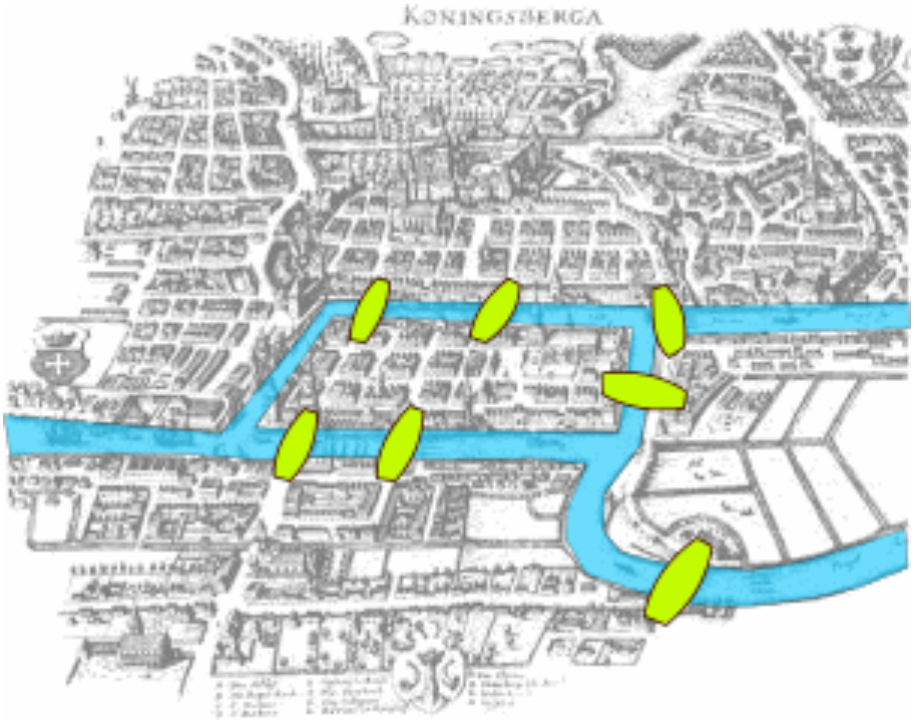


Figure 1: The Famous Königsberg Bridges

Since Euler solved the now famous "Bridges of Königsberg" problem in 1736[2] graph algorithms have been used in various areas of research including computer vision. There has been a recent renewal in graph-based approaches to computer vision problems as attested by special issues on such algorithms (e.g. IEEE-PAMI [3]). Unfortunately there is a conspicuous absence of such algorithms or even a framework to develop such algorithms in the current ITK library. The work described in this paper is meant as a step towards rectifying this glaring deficiency.

Two algorithms of note have been the subject of special attention and research [2, 1]. In this paper we describe the framework we developed for general graph-based methods as well as our implementation of the work found in [2, 1]. These recent algorithms have resurrected interest in such problems first discussed more than a decade ago by Greig *et al.* [5]. The key observation made by these researchers is that solving the well-known min-cut/max-flow problem [4] is equivalent to minimizing certain energy functions. Such algorithms have found useful application in such vision problems as stereo, motion, image restoration, scene reconstruction, and image segmentation [2].

[2] The origins of graph theory finds its historical roots at the confluence of Euler's mathematical prowess and a local fascination with the conundrum presented by the early 18^{th} century topographical characteristics of Königsberg, Prussia (now Kaliningrad, Russia). The puzzle concerned the existence of a path that traversed all seven bridges without repeated traversal (or, in modern parlance, does an Eulerian path exist for the corresponding graph structure?).

The requisite form for the aforementioned energy functions is that of a Markov Random Field (MRF) with binary labeling given by

$$E(f) = \sum_{p \in \mathcal{P}} D_p(f_p) + \sum_{\{p,q\} \in \mathcal{N}} V_{p,q}(f_p, f_q), \qquad f_p \in \mathcal{L} = \{\alpha, \beta\} \tag{1}$$

where f is the pixel labeling of the set of image pixels, $\mathcal{P}$. For the energy functions described in [1, 5] the label, f_p, of each pixel p can have one of two labels. We will denote this set of labels arbitrarily as $\mathcal{L} = \{\alpha, \beta\}$.[3] $D_p(f_p)$ is the likelihood measurement of the observed data. $V_{p,q}(f_p, f_q)$ is the interaction measurement (prior probability) between neighboring pixels p and q with respective labelings f_p and f_q (where the pixel neighborhood is denoted by $\mathcal{N}$).

The equivalency between the solution to the well-studied min-cut/max-flow problem and the energy minimization of equation (1) requires the transformation of equation (1) to the graph domain.[4] For our purposes, the set of image pixels, $\mathcal{P}$, is represented as a set of nodes in a constructed graph with two additional *terminal nodes* required to represent the *source* and the *sink* nodes common in network flow problems. Each node that represents an image pixel has a directed edge going from the source node to itself and from itself to the sink node. Associated with each of these directed edges is a weight calculated from the quantity D_p. Additionally, neighboring pixels are linked by edges in the graph. Associated with each edge between neighboring pixels is the edge weight derived from the quantity $V_{p,q}$. An example of such graph construction is illustrated in Figure 2 for a 1-D image consisting of the pixel set $\mathcal{P} = \{p, q, r\}$.

After using any min-cut/max-flow algorithm on the specially constructed graph, one gets a partitioning of the nodes of the graph into two sets — one set pertaining to the sink node and one pertaining to the source node, *e.g.* Figure 2(b). Boykov *et al.* compare different min-cut/max-flow algorithms in [1] including their own algorithm specifically tailored for graphs constructed from images. Experimentally, they found that even though the worst-case complexity of their algorithm was higher than other algorithms, within the domain of computer vision applications, their algorithm performed significantly better than other standard algorithms.

The solution for the binary labeling MRF problem is made simple using min-cut/max-flow algorithms. However, for n-ary labelings, $\mathcal{L} = \{\alpha_1, \alpha_2, \ldots, \alpha_n\}$, finding the minimum of equation (1) is NP-hard [2]. Fortunately, a reasonable approach to this multi-label minimization problem recasts the n-ary problem as a a series of binary sub-problems. Although the solution is not necessarily the global minimum, it is guaranteed to be within a certain factor of the global minimum [2]. This algorithm, known as α-expansion, is given in Table 1. Note that execution of step 3(a) utilizes the min-cut/max-flow algorithm described in [1].

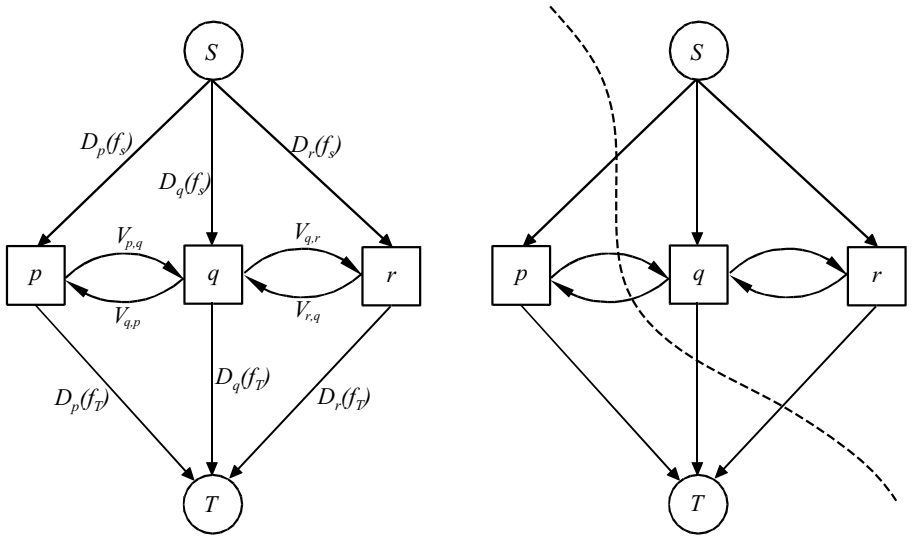
2 Software Implementation

In the previous section we gave a theoretical overview of the energy minimization scheme of Boykov *et al.* and its usage for such applications as image segmentation. In this section we describe the actual software implementation. Below is a list of the classes that were written for the software implementation and potential locations within the ITK directory structure.

- BasicFilters

[3] The case for n labels, *i.e.* $\mathcal{L} = \{\alpha_1, \alpha_2, \ldots, \alpha_n\}$ will be addressed by the α- expansion algorithm [2] discussed later in this paper.

[4] A graph, G, consists of a pair of sets, $G = (V, E)$, where the elements of E are 2-element subsets of V. Usually the elements of V are denoted as *nodes* and the elements of E are the *edges*.



(a) (b)

Figure 2: (a) Graph construction for a 1-D image with pixel set $\mathcal{P} = \{p,q,r\}$. The source and sink node are labeled S and T, respectively. In this example, a pixel's immediate neighbors comprise the pixel neighborhood. For the examples shown in this paper, the edge weights between pixels are symmetric such that $V_{p,q} = V_{q,p}$. (b) Hypothetical minimum cut for this 3-pixel example. Pixel p belongs to the sink terminal, T, while the set $\{q,r\}$ belongs to the source terminal, S.

Table 1: Algorithmic synopsis of the α-expansion algorithm given in [2].

1. Start with an arbitrary labeling f.
2. Set **success** $:= 0$.
3. For each label $\alpha \in \mathcal{L}$
   (a) Find $\hat{f} = \arg\min E(f')$ among f' within one α-expansion of f.
   (b) If $E(\hat{f}) < E(f)$, set $f := f'$ and **success** $:= 1$.
4. If **success** $= 1$ go to step 2.
5. Return f.

- `itkBoykovAlphaExpansionMRFImageFilter.h`
- `itkBoykovAlphaExpansionMRFImageFilter.txx`
- `itkBoykovMinCutGraphFilter.h`
- `itkBoykovMinCutGraphFilter.txx`
- `itkGraphToGraphFilter.h`
- `itkGraphToGraphFilter.txx`

  - itkGraphToImageFilter.h
  - itkGraphToImageFilter.txx
  - itkImageToGraphFilter.h
  - itkImageToGraphFilter.txx

- Common
  - itkBoykovGraphTraits.h
  - itkDefaultGraphTraits.h
  - itkImageGraphTraits.h
  - itkBoykovImageToGraphFunctor.h
  - itkBoykovImageToGraphFunctor.txx
  - itkDefaultImageToGraphFunctor.h
  - itkDefaultImageToGraphFunctor.txx
  - itkGraph.h
  - itkGraph.txx
  - itkGraphDuplicator.h
  - itkGraphDuplicator.txx
  - itkGraphSource.h
  - itkGraphSource.txx
  - itkInPlaceGraphFilter.h
  - itkInPlaceGraphFilter.txx

- Testing
  - itkGraphTest.cxx
  - itkBoykovAlphaExpansionFilterTest.cxx

- Examples
  - BoykovGraphCutFilter.cxx

The inheritance relationship between these newly implemented classes and existing classes is given in Figures 3, 5, and 4. Note that the graph class and associated filter class hierarchies have analogical equivalencies to the mesh and image-related classes. For example, just as the ImageSource and MeshSource classes are base classes for any filter producing their corresponding data types as outputs, GraphSource is the base class for any filter which produces a graph.

In order to further elucidate the functionality and the relationships between classes, we first describe the graph framework and then revisit the α-expansion algorithm discussed in Table 1 to give the reader a sense of the usage of the associated classes.

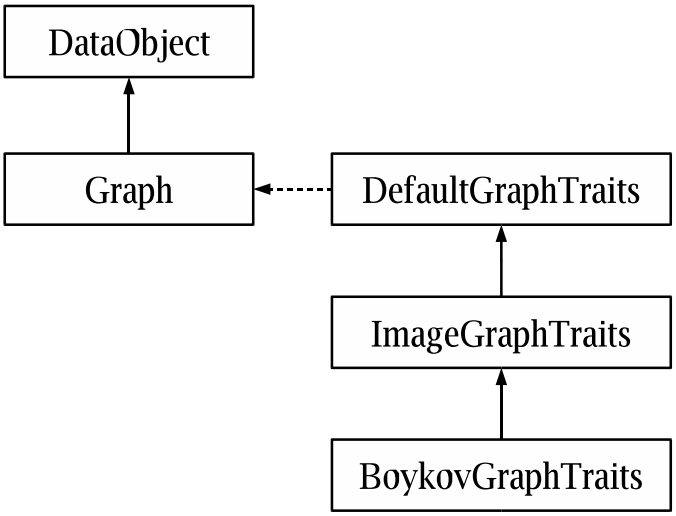


Figure 3: Illustration of the hierarchy for the filter classes for graphs and graph-related data structures. The graph class is templated over a class which defines the graph traits. The graph traits classes define the node and the edges.

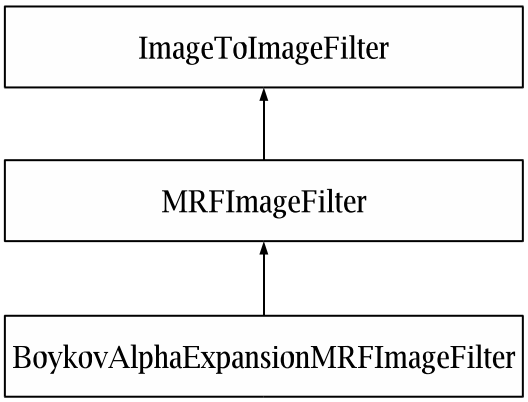


Figure 4: Illustration of the hierarchy for the α-expansion filter algorithm. Note that it is derived from the image filter class associated with MRFs.

2.1 Graph Framework

As mentioned previously, a graph, G, consists of a pair of sets, $G = (V, E)$, where the elements of E (edges) are 2-element subsets of V (nodes). We envision graphs to be on par with the image and mesh types as principle data types in ITK. The class hierarchy for the graph class is given in Figure 3.

This basic class consists of two containers, viz. an edge container and a node container. The definition of the node types and the edges types are given in the graph traits class over which the graph class is templated (similar to the relationship between the mesh class and the mesh traits classes). Also defined within the class are simple iterators for the nodes and edges.

Due to the simplicity of the graph class and the ability to template over specifically tailored nodes and edges, *i.e.* graph traits, we foresee a simple integration with any future graph-based methods implemented in ITK.

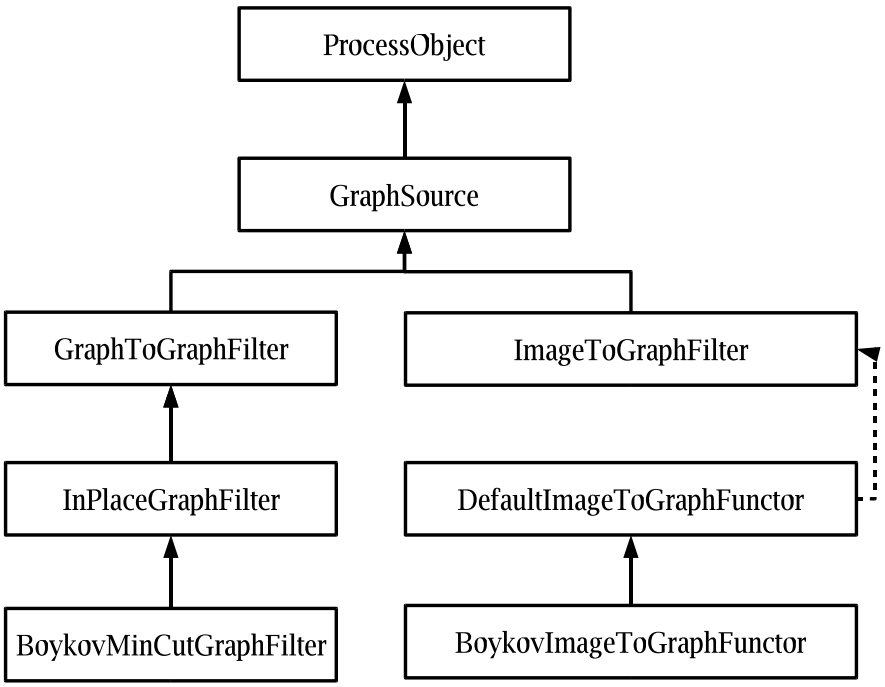


Figure 5: Illustration of the hierarchy for the graph-related filter classes.

2.2 α-Expansion Redux

We first described the α-expansion algorithm in Table 1. We revisit that algorithm to elucidate the associated class structure. Shown in Figure 6 is a flow diagram illustrating the usage of the `BoykovAlphaExpansionFilter` class. The user-provided input includes the image to be segmented and the selection of the number of labels, n. For each label $i = 1, 2, \ldots, n$, the user provides the likelihood image $\Pr(I_p|\text{label } i)$ and sets the pixel index container for label i (optional). Also necessary is the functor (*i.e.* `ImageToGraphFunctor`) which essentially describes how to build a graph from the given image. After execution of the `BoykovAlphaExpansionFilter`, the output consists of an integer image where each pixel corresponds to one of the labels $i_p \in \{1, 2, \ldots, n\}$.

Shown in Figure 8(b) are sample results derived from the 2-D brain slice shown in Figure 8(a). The probability images that were supplied as input were derived by using the mean of each pixel type in the original image and using the Gaussian models given in equations (2), (3), and (4). The three labels csf, gm, and wm correspond to CSF, gray matter, and white matter, respectively. The mean pixel value of each of these three labels is denoted by μ.

$$\Pr(I_p|csf) = \frac{1}{\sqrt{2\pi}\sigma_{csf}} \exp\left(-\frac{(I_p - \mu_{csf})^2}{2\sigma_{csf}^2}\right) \tag{2}$$

$$\Pr(I_p|gm) = \frac{1}{\sqrt{2\pi}\sigma_{gm}} \exp\left(-\frac{(I_p - \mu_{gm})^2}{2\sigma_{gm}^2}\right) \tag{3}$$

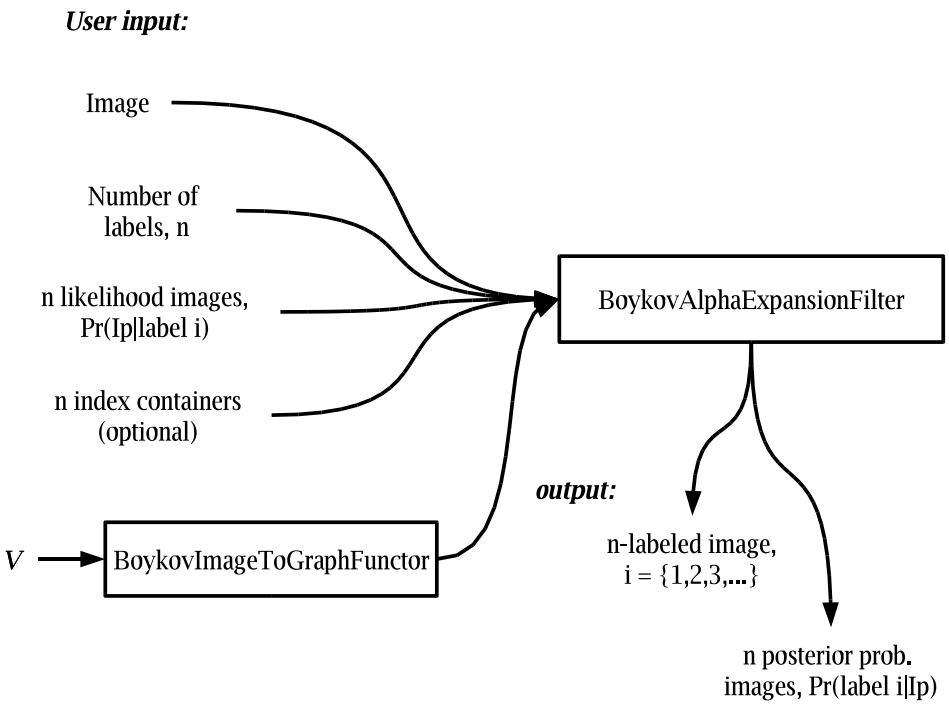


Figure 6: Flowchart of the functioning of the α-expansion algorithm.

$$\Pr(I_p|wm) = \frac{1}{\sqrt{2\pi}\sigma_{wm}} \exp\left(-\frac{(I_p-\mu_{wm})^2}{2\sigma_{wm}^2}\right) \tag{4}$$

While such a simplistic approach leads to plausible results, more sophisticated methodologies can certainly be employed. For example, the user has the option of specifying sets of pixels which correspond to a given label. In this way, the user can hard-constrain a pixel to be of a certain label (see Figure 10). From such a sampling of pixel types, it is possible to use non-parametric techniques, *e.g.* Parzen windowing, to construct the likelihood images.

2.3 Boykov Min-Cut/Max-Flow Filter

As mentioned previously, we can find an approximate solution to the multi-label problem by dividing this problem into a set of binary sub-problems. This is step 3(a) of Table 1. This requires the min-cut/max-flow algorithm discussed previously. For each iteration of step 3 of Table 1, a label is chosen. Using the `ImageToGraphFilter` class, a graph exhibiting the specific form described in [2] is created from the input images using the image-to-graph functor. This graph is then labeled using the min-cut/max-flow algorithm from which the a new image labeling is derived. The flowchart for this subprocess is given in Figure 7.

3 Sample Usage

We include the file `itkBoykovGraphCutFilterTest.cxx` which produces the 2-D and 3-D results illustrated in the following section. Consistent with Figure 6 the input consists of an input image to be segmented,

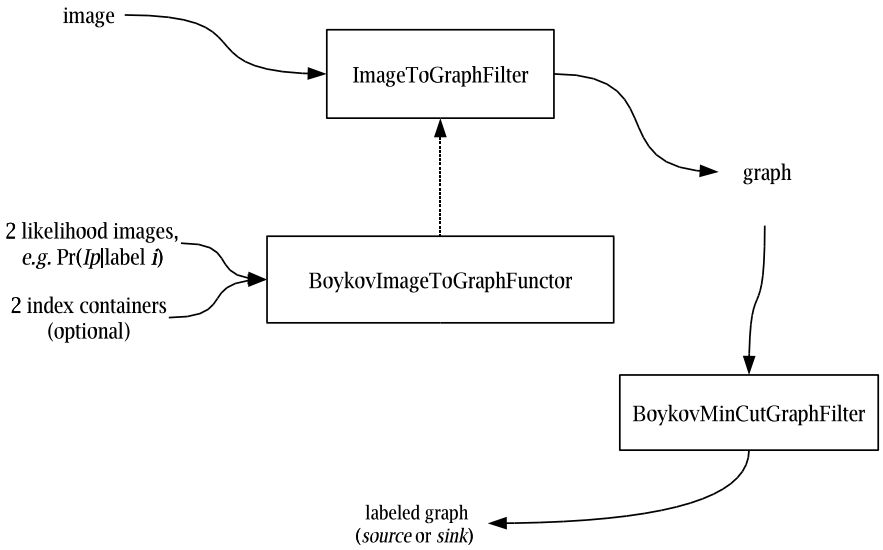


Figure 7: Flowchart describing the min-cut/max-flow algorithm used in step 3(a) of Table 1.

and n likelihood images—one for each class. We then specify the BoykovImageToGraphFunctor as follows:

```
198   typedef itk::BoykovGraphTraits<short, ImageDimension> GraphTraitsType;
199   typedef itk::Graph<GraphTraitsType>                    GraphType;
200
201   typedef itk::BoykovImageToGraphFunctor
202     <LikelihoodImageType, GraphType> FunctorType;
203   typename FunctorType::Pointer BoykovFunctor = FunctorType::New();
204   BoykovFunctor->SetLambda( 10.0 );
205   BoykovFunctor->SetSigma( 10.0 );
206   BoykovFunctor->SetExcludeBackground( exclude );
207   BoykovFunctor->SetBackgroundValue( 0 );
208   BoykovFunctor->SetRadius( 1 );
209   BoykovFunctor->ActivateAllNeighbors();
210
211 //  BoykovFunctor->ActivateIndex( 1 );
212 //  BoykovFunctor->ActivateIndex( 3 );
213 //  BoykovFunctor->ActivateIndex( 4 );
214 //  BoykovFunctor->ActivateIndex( 5 );
215 //  BoykovFunctor->ActivateIndex( 7 );
```

The call on line 208 specifies that all the neighbors within the specified radius be connected to the center voxel. However for a sparser neighborhood structure, one can specify the neighborhood connections individually as shown in the commented lines 209-213. We then specify the `BoykovAlphaExpansionMRFImageFilter` and input the likelihood images.

```
217   typedef itk::BoykovAlphaExpansionMRFImageFilter
218     <LikelihoodImageType, GraphTraitsType> FilterType;
219   typename FilterType::Pointer filter = FilterType::New();
220   filter->SetRandomizeInitialLabeling( false );
221   filter->SetNumberOfClasses( Nlabels );
222   filter->SetInput( input );
```

```
223   filter->SetImageToGraphFunctor( BoykovFunctor );
224
225   /**
226    * The zeroth label is reserved for the background.
227    */
228   filter->SetLikelihoodImage( 1, label1_image );
229   filter->SetLikelihoodImage( 2, label2_image );
230   if( Nlabels > 2 )
231     {
232     filter->SetLikelihoodImage( 3, label3_image );
233     if( Nlabels > 3 )
234       {
235       filter->SetLikelihoodImage( 4, label4_image );
236       }
237     }
```

One can also assign specific indices to be of a certain label. This is done

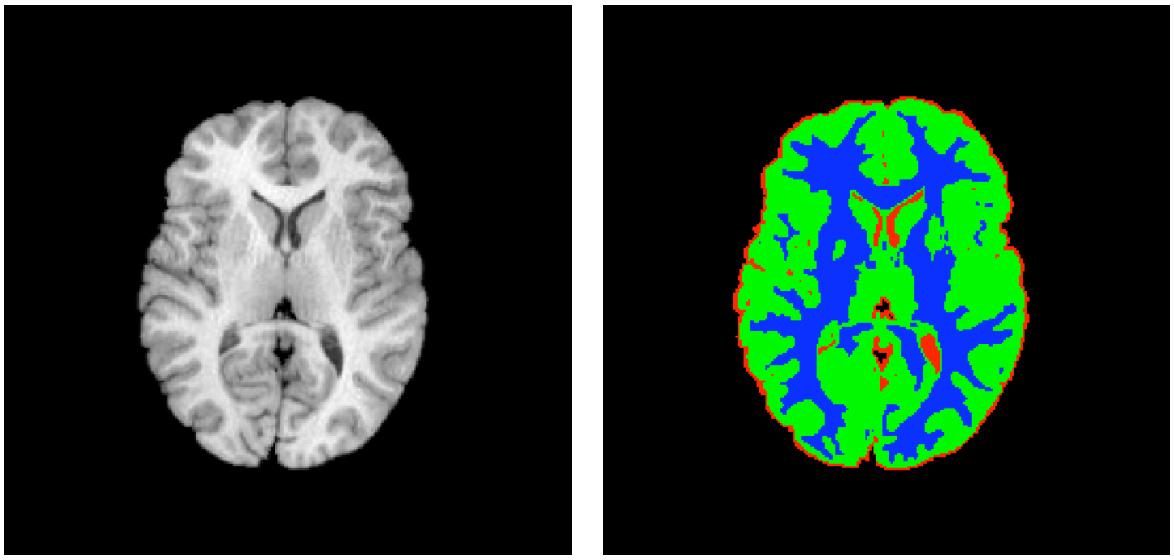
```
243     IndexType index;
244     typename FilterType::IndexContainerType label1_indices;
245     typename FilterType::IndexContainerType label2_indices;
246
247     for( unsigned int i = 32; i <= 72; i++ )
248       {
249       for( unsigned int j = 86; j <= 124; j++ )
250         {
251         index[0] = i;
252         index[1] = j;
253         label2_indices.push_back( index );
254         }
255       }
256
257     for( unsigned int i = 5; i <= 30; i++ )
258       {
259       for( unsigned int j = 5; j <= 30; j++ )
260         {
261         index[0] = i;
262         index[1] = j;
263         label1_indices.push_back( index );
264         }
265       }
266
267     filter->SetIndexContainer( 1, label1_indices );
268     filter->SetIndexContainer( 2, label2_indices );
```

4 Sample Results

We give a selection of a few of the results that we have derived from the methodology described in this report. The images were produced using ITK-SNAP. Note that all the likelihood images have been specified by the naive approach of sampling a set of images pixels from a particular region and modeling the likelihood as a normal distribution. Also note that we did not attempt to optimize the parameter choices.

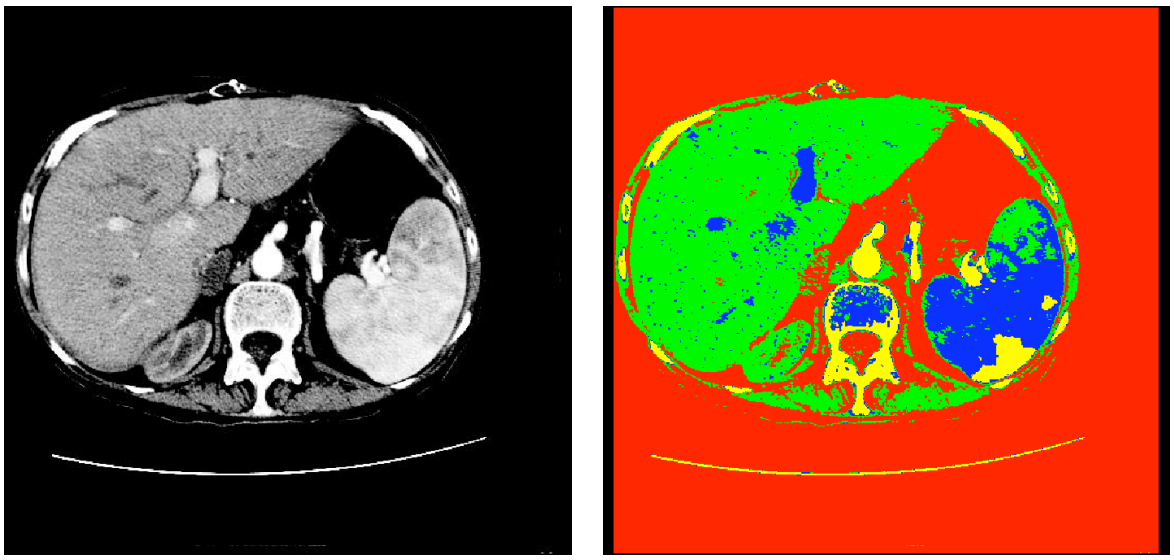
Acknowledgments

This work was supported by the National Library of Medicine under grant 467-MZ-402069.



(a) (b)

Figure 8: Results of the α-expansion algorithm (Table 1) from a 2-D brain slice for $\mathcal{L} = \{1,2,3\}$. Note that the background pixels have a value of 0. Here the labels correspond to CSF, gray matter, and white matter, respectively.

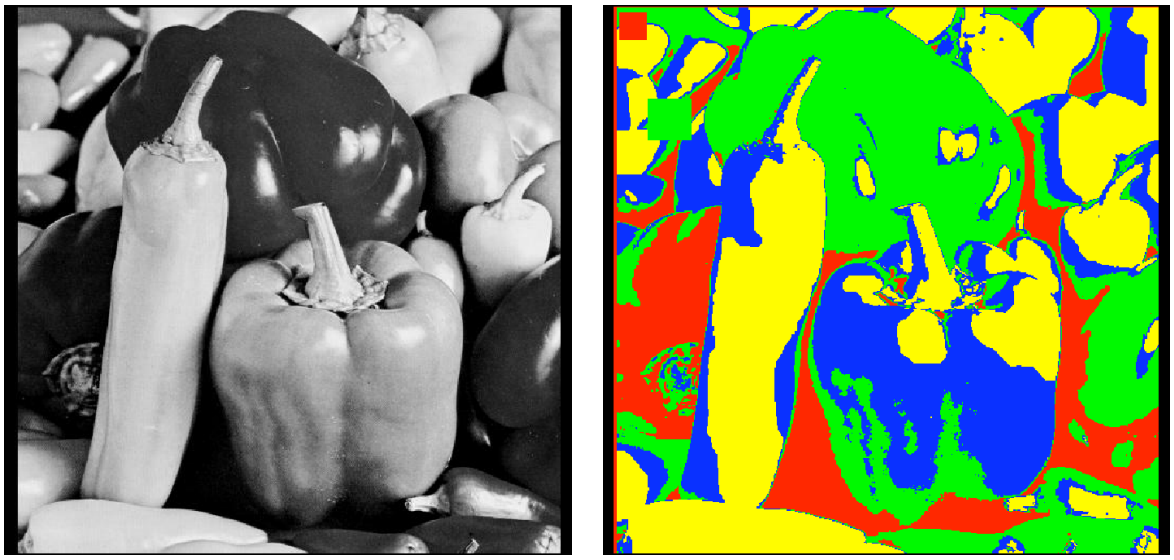


(a) (b)

Figure 9: Results from a CT scan where $n = 4$. The different labelings correspond to background, liver, spleen, and bone. (a) The original image. (b) Segmented image.

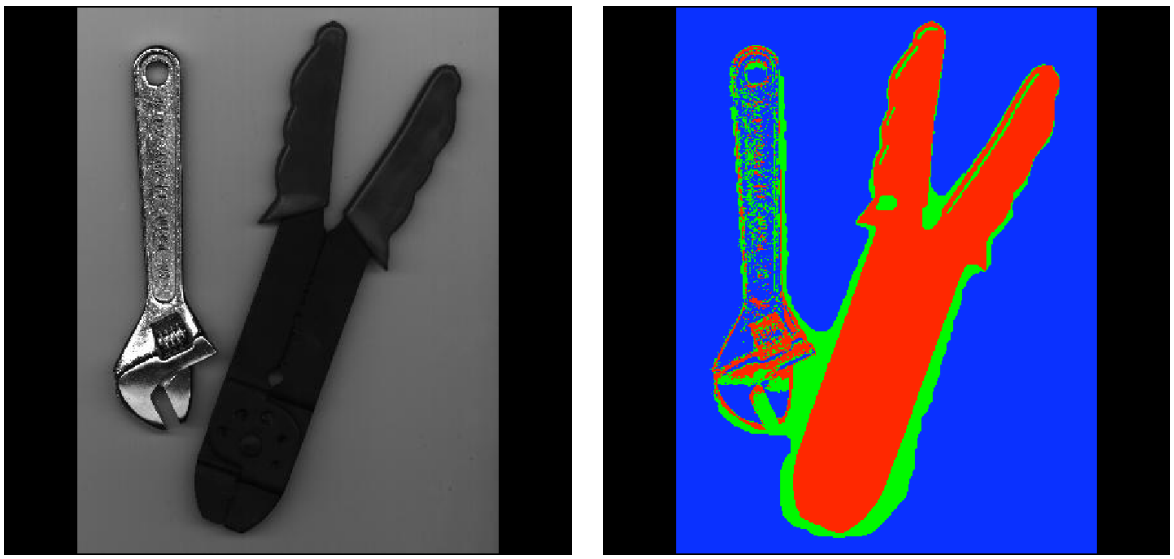
References

[1] Yuri Boykov and Vladimir Kolmogorov. An experimental comparison of min-cut/max-flow algorithms for energy minimization in vision. *IEEE Trans Pattern Anal Mach Intell*, 26(9):1124–37, Sep 2004. 1, 1, 1



(a) (b)

Figure 10: Two resulting images derived from an image of an assortment peppers ($n = 4$). (a) The original image. (b) Segmented image two sets of pixels corresponding to label 1 and label 2 shown as squares in the upper right corner of the image.

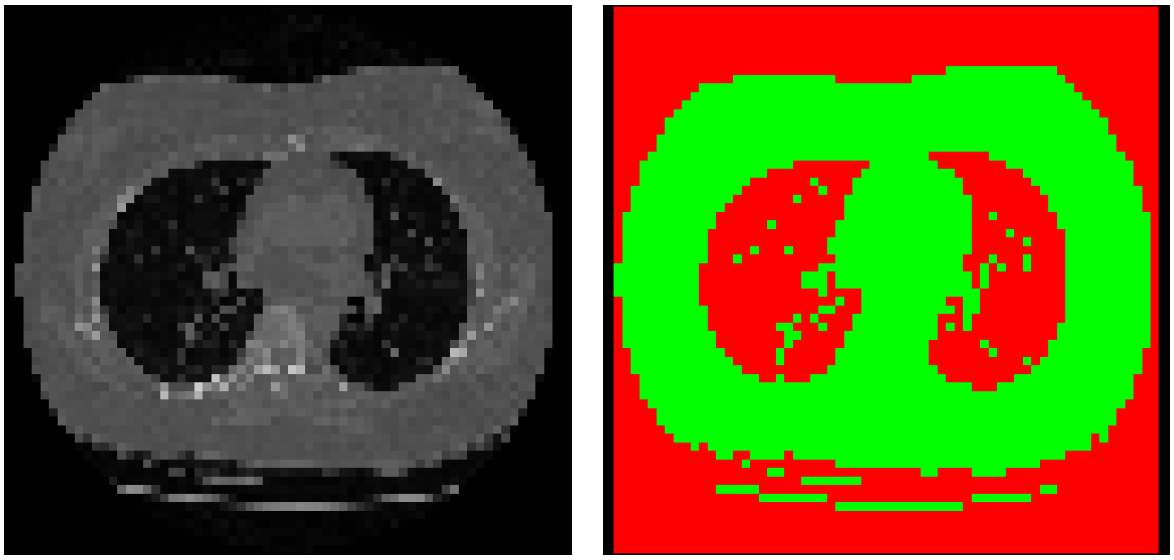


(a) (b)

Figure 11: Results from an image of tools ($n = 3$). (a) The original image. (b) Segmented image. Note that the mean of the wrench is very close to the mean pixel value of the background. This is manifested in the resulting image.

[2] Yuri Boykov, Olga Veksler, and Ramin Zabih. Fast approximate energy minimization via graph cuts. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 23(11):1222–1239, 2001. 1, 1, 3, 1, 2.3

[3] Sven Dickinson, Marcello Pelillo, and Ramin Zabih. Introduction to the special section on graph algorithms in computer vision. *IEEE Transactions on Pattern Analysis and Machine Intelligence*,



(a) (b)

Figure 12: 3-D results from a CT lung image ($n = 2$). (a) The original image. (b) Segmented image. This is manifested in the resulting image.

23(10):1049–1052, 2001. 1

[4] L. R. Ford and D. R. Fulkerson. Maximal flow through a network. *Canadian Journal of Mathematics*, 8:399–404, 1956. 1

[5] D. M. Greig, B. T. Porteous, and A. H. Seheult. Exact maximum a posteriori estimation for binary images. *J. R. Statist. Soc. B*, 51(2):271–279, 1989. 1, 1