Medical Image Segmentation using Region-Growing and a Simplified Mumford-Shah Functional

Grant S. Roberts[1], Lawrence Lechuga[1], Timothy Ruesink[2]
[1]*Department of Medical Physics, University of Wisconsin-Madison, WI*
[2]*Department of Mechanical Engineering, University of Wisconsin-Madison, WI*

Significance and Aims

Segmentation is a vital task in modern medical image processing that serves to reduce data analysis only to specific subregions that are of clinical interest. Advances in medical imaging technologies have provided the medical and scientific communities with larger datasets of ever-increasing image quality. Thus, there is an inherent need for robust segmentation methods that can extract clinically important information *automatically*, as opposed to performing time-consuming *manual* segmentation which is still done today in some clinical applications. The overarching goal of this project is to successfully implement an algorithm that automatically segments medical images using a region-growing algorithm. This will be accomplished through the following **specific aims**.

<u>Specific Aim 1</u>: **Reproduce an existing region-growing segmentation algorithm using a simplified Mumford-Shah functional.**

<u>Specific Aim 2</u>: **Develop a novel region-growing algorithm using the same formulation but with added flexibility in initialization conditions and adjustments to the cost function calculation, in hopes of increasing efficiency and accuracy.**

With respect to expected outcomes, the work proposed in **specific aim 2** is expected to decrease post-processing times and increase segmentation reproducibility, which will in turn positively impact clinical workflows that currently utilize manual segmentation methods.

1. Introduction

Segmentation is an image processing technique that takes advantage of one or more imaging features to partition an image (or set of images) into useful regions of interest. This can analogously be described as assigning labels to pixels in an image, grouping pixels that share similar features. This process is very commonly seen in medical image processing, in which a medical image is decomposed into clinically significant segments, which can make the image easier to visualize or can simplify the data analysis procedure (restricting analysis only to a subset of the image). The degree and accuracy to which an object is segmented depends highly on the task being performed, and is thus highly dependent on the specific application. Some applications involve: quantification of tissue volume, localization of pathology, treatment planning, and statistical analysis of tissue regions[1]. Because of the large diversity in imaging tasks, a large variety of segmentation methods have been developed to better adapt to these disparate imaging task. Segmentation techniques range from manual methods, in which a user manually identifies regions of interest, to more automated methods, in which segmentation is performed unsupervised. Manual methods are often time-consuming and are prone to relatively high inter- and intraoperator variability[2]. Thus, there is an inherent need for robust segmentation methods that can extract clinically important information *automatically*, as opposed to time-consuming *manual* segmentation which is still done today in some clinical applications.

There are several common methods that are used to automatically or semi-automatically segment images, each with varying degrees of accuracy, efficiency, and simplicity. One relatively simple and popular segmentation technique is region-growing, a method that groups similar pixels into larger subregions based on a predefined condition. This method begins with a "seed point" placed in the image, and the region grows based on the predefined condition, which in most cases is a gray-level threshold. Pixels will be advanced outwards from the "seed" and will be added to the region if they are within the range of gray level thresholds. In some cases, manual placement of seeds is not desired, as in fully-automated processing. Several methods exist to allow for automatic placement of seed points[3]. One approach is to subdivide the image initially into a set of arbitrary, disjointed regions and then iteratively merge or split them based on the criteria stated previously[3].

A segmentation can also be performed using variational methods, which is the process of minimizing a variational model (cost function) that suitably describe a practical segmentation problem. This was first described by Geman and Geman[4] in 1984. Variational methods can be solved using a probabilistic approach, often modeled by Markov random fields and optimized with Bayesian estimation (a posteriori estimation)[5]. Another commonly encountered variational model is termed the Pott's model. The formulation is given in equation 1:

$$\text{minimize}\ [\ \nu\|\nabla \mathbf{f}\|_0 + \|\mathbf{f}-\mathbf{g}\|_2^2\ ] \tag{1}$$

where ν is the regularization parameter, the vector **f** is the output segmentation image, **g** is the input image, and $\|\cdot\|_p$ represents the ℓ_p norm. The first term enforces "flatness" of the segmentation and the second term enforces data fidelity, with the regularization parameter ν balancing the two terms. This formulation is quite difficult to minimize because of the ℓ_0 norm in the first term, and an analytical solution for the gradient of the cost function is thus not easily solved. However, this can be solved using advanced techniques such as convex relaxation approach, as noted in many articles[6–8]. Another closely related formulation is termed the Mumford-Shah (MS) model[9], which enforces both boundary length sparsity and data fidelity of the segmentation output. This model was constructed to unify several pre-existing theories regarding variational methods. This model will be more thoroughly discussed in the methods. It should be noted that each of these approaches yield non-convex topologies, which can make the model difficult to minimize, particularly globally. Despite this, generally satisfactory images can be found despite the non-convex nature of the problem, and can be further optimized when multiple segmentation and image processing techniques are used in a multi-stage process[2].

Koepfler at al.[10] presented a novel multi-stage segmentation algorithm that combines both region-growing and variational methods, specifically using the simplified Mumford-Shah (SMS) functional. In this specific paper, *every* pixel in an image is assigned a separate region. If the cost function decreases when the pixels are merged, then the two regions are merged. Alternatively, if the cost function increases when the pixels are merged, they are left as separate regions. This is done iteratively for all regions. Several distinct disadvantages arise from this type of algorithm.

First, this method does not calculate the *global* cost function, but rather the cost of only the two regions being considered. Additionally, initializing the image by assigning every pixel to a different region can become very computationally expensive, particularly in the case for large image matrices.

The purpose of this study is twofold: (1) to successfully reproduce the segmentation algorithm described by Koepfler et al. and to (2) develop a new algorithm to alleviate the drawbacks of the aforementioned algorithm. Both algorithms will be tested on simulated phantoms and clinical images and the results will be compared qualitatively as well as quantitatively, using several common segmentation performance metrics. The hypothesis is that our novel algorithm will both increase the accuracy and efficiency of the previously described algorithm.

2. Theory and Methods

2.1 Mumford-Shah Functional

The Mumford-Shah (MS) functional in the continuous domain can be fully expressed as:

$$E(\mu,\nu) = \mu \iint_{\Omega} (\mathbf{f}-\mathbf{g})^2 dA + \iint_{\Omega\setminus \mathrm{K}} |\nabla \mathbf{f}|^2 dA + \nu[H^{d-1}(\mathrm{K})] \tag{2}$$

where the vector **f** represents the output image (segmentation) and **g** represents the actual image. The first term is the data fidelity term, in which the mean square error between the segmentation and the image is integrated over the entire image set (Ω). Note that individual regions in the segmentation image (Ω_i) are disjointed connected open subsets of a planar domain Ω, each one with a piece-wise smooth *boundary* with K representing the boundary set, such that: $\Omega = \Omega_1 + \Omega_2 + \cdots + \Omega_R + K$. The second term integrates over all non-boundary sets ($\Omega - \mathrm{K} = \sum_{r=1}^{R} \Omega_r$). This second term constrains **f** to not vary considerably over the each segmentation subregion. Lastly, the third term requires that the boundary set be as small as possible. Regularization parameters μ and γ balance the data term and the length of boundaries of all of the segmentation regions respectively. The function H^{d-1} is the d-1 dimensional Hausdorff distance operator. In this paper, only 2D images will be considered. Thus, the sets will be constrained to $\mathbb{R}^2$ coordinate space and $H^{d-1} = H^1$ which simply measures the length of the boundary set K. For simplicity, the 1-dimension Hausdorff operator $H^1(K)$ will be denoted as $L(K)$.

Eq. 2 can be greatly simplified if one assumes that **f** be piece-wise constant over each open set Ω_i. This then causes the gradient of the constant function **f** to be 0, eliminating the second term. Doing so leads to the *simplified* Mumford-Shah (SMS) functional, expressed as:

$$E(\nu) = \int_{\Omega} (\mathbf{f}-\mathbf{g})^2 dx + \nu L(\mathrm{K}) \tag{3}$$

The μ parameter can also be dropped as only one regularization parameter is needed for the two terms. Note that if **f** is assumed to piece-wise constant, the first term will be minimized if **f** is the mean of **g** over the rth region. Thus, the SMS functional can be further simplified and discretized:

$$E(\nu) = \sum_{r=1}^{R}\sum_{i=1}^{N} \left(\mathrm{g}_{\mathrm{i,r}} - mean_{\Omega_r}(\mathrm{g}_{\mathrm{r}})\right)^2 + \nu L(\mathrm{K}) \tag{4}$$

in which the $mean_{\Omega_r}(\mathrm{g}_\mathrm{r}) = \mathrm{f}_{\mathrm{i,r}}$ is the approximated value of the rth subregion. Here, $\mathrm{g}_{\mathrm{i,r}}$ represents the ith pixel in the rth subregion of the input image. The mean squared error will be calculated for each subregion (Ω_r) and will be added to the total length of all non-overlapping boundaries (boundary set K) to give the total energy of the image, denoted by $E(\nu)$. In words, this means our segmentation ($f_{i,r}$) will be approximated using the mean value g_r of that subregion. Recall that $\Omega = \Omega_1 + \cdots + \Omega_R + K$. By combining the each subregion, one can create the total segmented image. Interestingly, equation 4 is very closely related to the Ising model, describing the grouping of discrete magnetic dipole moments of atomic spins to characterize global ferromagnetic behavior. Equation 4 will be the primary equation used to calculate the energy for both algorithms discussed below.

It turns out that this equation is exactly equivalent to the Pott's model. Recall from the introduction the formulation of the Pott's model: minimize $[\, \nu\|\nabla \mathbf{f}\|_0 + \|\mathbf{f} - \mathbf{g}\|_2^2 \,]$. The ℓ_2-norm term is exactly equivalent to the first term in the SMS functional, aside from the absolute value which is redundant since the difference between the segmentation and the input image is squared. Secondly, if one assumes that **f** is piece-wise constant segmentation, the gradient of **f** will only leave edges. The ℓ_0-norm is essentially counting the number of discrete boundary edges, which is precisely what the Hausdorff length is calculating. Lastly, the ν parameter exists in both equations to regularize the equation. Thus, the Pott's model and the SMS functional are equivalent under the assumption that the segmentation image **f** is piece-wise constant.

2.2 Koepfler Algorithm

The Koepfler algorithm[10] was implemented in MATLAB2018b (Mathworks, Natick, MA). This algorithm used the SMS cost function (Eq. 4) and region-growing to perform segmentation. Specifically, the algorithm initialized a region map in which every pixel was labelled as an individual region. For every pixel (starting from the top left pixel), each adjacent (4-connected) pixel was tested by: (1) evaluating the SMS functional for each of the two regions separately and (2) evaluating the functional when the two regions are merged. If merging decreased the cost function, then the two regions were merged. On the other hand, if the cost function increased when the pixels were merged, they were left as separate regions. This was done iteratively for every pixel in the image until that region could not be merged, then the same procedure would be performed on the next candidate region, keeping a track record of the updated region map. The region map could be converted to a segmentation image by averaging the input image values over the area of each unique subregion described by the region map. This algorithm process is outline in Figure 1. Note that the cost is not calculated globally over the whole region, instead it is calculated between just two regions at a time, meaning the cost function *differential* is instead being calculated.

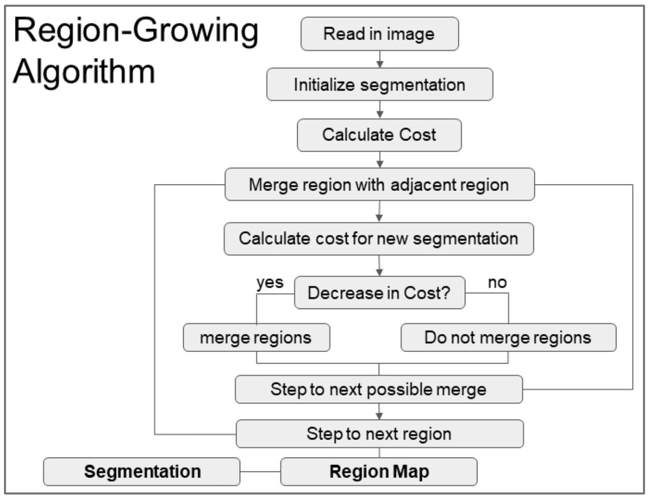

Figure 1: Outline of a generic algorithm used for a variational method region-growing scheme. In this paper, the SMS functional is used as the cost function.

The performance of this algorithm was first tested using a modified Shepp-Logan brain phantom with an image matrix size of 256x256. Three degrees of noise of noise were added to the original Shepp-Logan, producing three test phantoms: $\sigma^2 = 0$, $\sigma^2 = 0.003$, and $\sigma^2 = 0.005$. An edge-preserving bilateral filter was applied to all three images to assist in the denoising. Additionally, one brain image with a prominent brain tumor was acquired from the BRATS-SMIR public database[11], in which no bilateral filter was used. After the algorithm was completed on all of the preprocessed phantom and brain datasets, the segmentation image, region map, total number of regions, computation time, and regularization parameter value were recorded. Several regularization values were tested for each of the four images and an optimal value was chosen qualitatively.

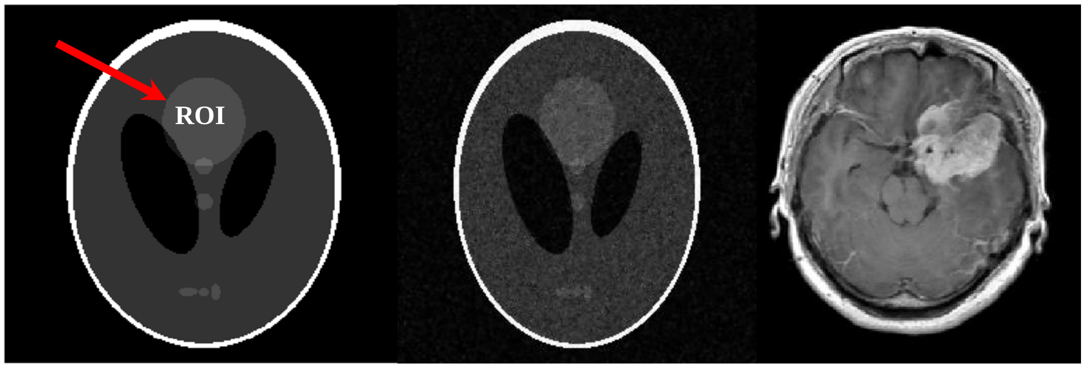

Figure 2: (Left) Original 256x256 modified Shepp-Logan simulation phantom, made in Matlab. The ROI is the region that will be compared with DICE coefficients. (Middle) The same phantom with a noise level of σ^2 = 0.003 artificially added and preprocessed with a bilateral filter, acting to smooth the noise and preserving the edge structures. (Right) Clinical MRI image of a brain with a large brain tumor.

2.3 Our Algorithm

A novel algorithm was developed in MATLAB2018b (provided in Appendix A) to further optimize the region-growing algorithm established by Koepfler. Our algorithm incorporated a calculation of a *global* cost function evaluated over the entire image domain, instead of restricting the calculation to a cost function differential of just two regions being considered. This allowed for a 2D visualization of the descent of the cost function energy as a function of iteration.

Furthermore, the initialization conditions were changed from a strict condition where each pixel was its own region to a more flexible type initialization, intended primarily to increase efficiency and potentially accuracy. To test this, one specific type of initialization was performed. A standard region-growing technique, as described in the introduction, was used to produce an initial segmentation. The initial seed point, like the Koepfler algorithm, began in the top left pixel. Regions were grown based on a threshold condition using a built-in MATLAB command *grayconnected*. This connected adjacent (4-connected) pixels based on a gray-value threshold. Note that this initialization was performed prior to any SMS functional evaluation. For each of the four images, optimal gray-value thresholds were qualitatively selected using a range of different thresholds. Once the optimal threshold was selected, standard region growing mask was created and the same SMS functional procedure was performed, merging two regions if the global cost function decreases or alternatively leaving them as separate regions if the cost function increases.

After the algorithm was completed, the segmentation image, region map, total number of regions, computation time, and regularization parameter value were recorded. This was done for the same datasets as in the previous algorithm. Several regularization values were tested for each of the four images and an optimal value was chosen qualitatively. Lastly, two additional brain datasets were acquired for demonstration purposes and segmentations will be shown in the results section.

2.4 Metrics

To quantitatively compare both algorithms, computation time, number of initial regions, and number of final images were compared. DICE coefficients were used to compare one specific segmentation region in the phantom to a ground-truth segmentation. Shown in Figure 2 is the modified Shepp-Logan phantom. The segmentation region of interest (ROI) that was compared is illustrated in the figure. The ground-truth can be easily obtained by a simple thresholding procedure performed on the original Shepp-Logan phantom, which is inherently composed of piecewise constant subregions. This region was chosen because it had the most variability between algorithms and because the segmentation of this region was highly dependent on noise. The DICE coefficient is given in Eq. 5:

$$DSC = \frac{2|X \cap Y|}{|X| + |Y|} \tag{5}$$

In this equation, |X| and |Y| represent the number of elements in a test set and ground-truth set, respectively. The equation is essentially defining the amount of overlap between the two segmentations, and is scaled from 0 to 1. In the brain analyses, DICE coefficients were not used because no ground-truth segmentation was obtained. Instead, only qualitative assessment of the brain images was performed.

3. Results

All 4 phantom and brain datasets were successfully segmented for both algorithms. In a qualitative sense, both algorithms produced fairly high quality, comparable, segmentations once the regularization parameter (and region-growing threshold) was optimized. A summarized table for both algorithms is given in Table 1.

Table 1: Quantitative comparison between both algorithms.

	Old Region Growing Algorithm				**New Region Growing Algorithm**			
Phantom	Time (min)	DICE	Initial Regions	Final Regions	Time (min)	DICE	Initial Regions	Final Regions
Phantom (256x256)	6.6	1.000	65536	15	0.07	1.000	15	15
Noisy phantom (σ=0.003)	7.3	0.973	65536	54	0.09	0.972	81	9
Noisy phantom (σ=0.005)	9.7	0.960	65536	407	0.27	0.955	148	7
Brain (256x256)	10.7	-	65536	342	1.6	-	1037	87

Images of the Koepfler algorithm are shown in Figure 3. The computation times were between 6 and 11 minutes for all four images. The DICE coefficient showed excellent agreement to the ground-truth segmentation as all coefficients were above 0.95. The initial regions for each image were 65536 (256^2) as expected. The regularization parameters were much higher for the Koepfler algorithm than for our algorithm, typically around 100-300.

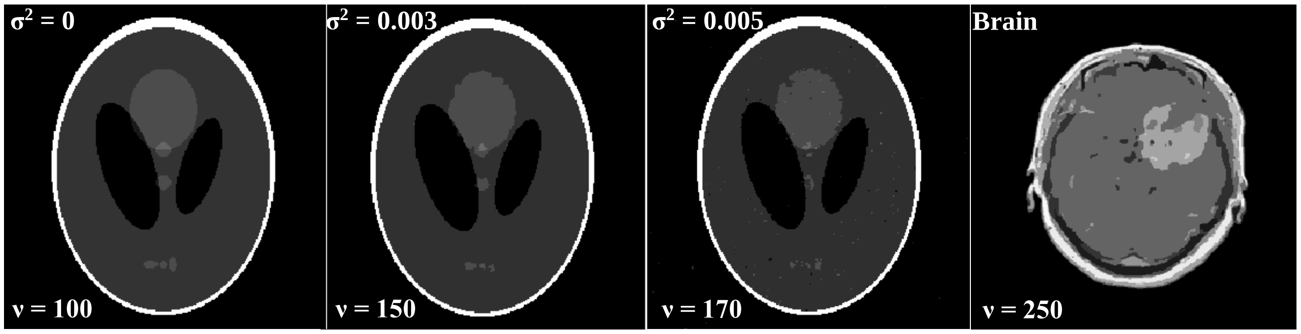

Figure 3: Output for the Koepfler algorithm for all 4 datasets. From left to right, phantoms with noise $\sigma^2 = 0$, $\sigma^2 = 0.003$, $\sigma^2 = 0.005$ and a clinical brain image. Regularization parameters are provided in the bottom left part of the image.

Images of our algorithm are shown in Figure 4. Initial segmentation using a standard region-growing technique was successfully implemented. Figure 5 shows a progression of the standard region-growing used in the initialization procedure. By implementing the initialized segmentation, the number of initial regions was greatly reduced (often over 100-fold). Computation times were reduced drastically, which ranged from 20 seconds to 2 minutes. The DICE coefficient also showed excellent agreement to the ground-truth segmentation, however the coefficients were decreased slightly from the Koepfler algorithm.

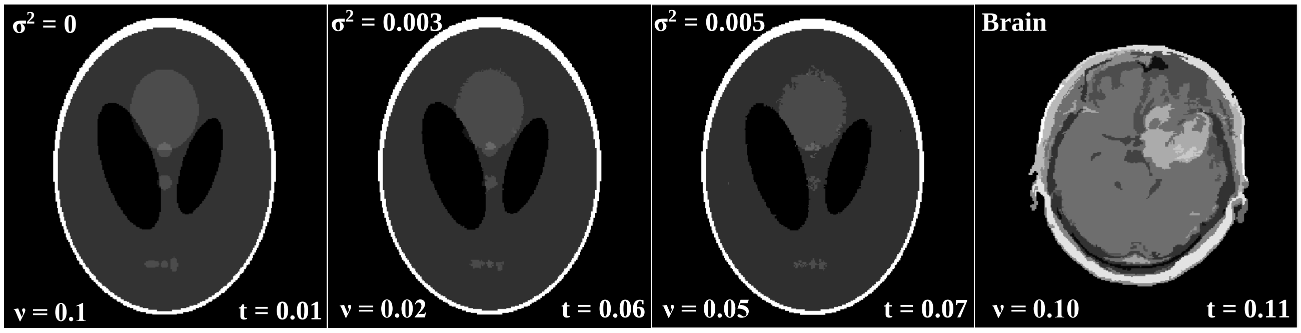

Figure 4: Output for our algorithm for all 4 datasets. From left to right, phantoms with noise $\sigma^2 = 0$, $\sigma^2 = 0.003$, $\sigma^2 = 0.005$ and a clinical brain image. Regularization parameters are provided in the bottom left part of the image and initialization gray-level threshold levels are provided in the bottom right part of the image.

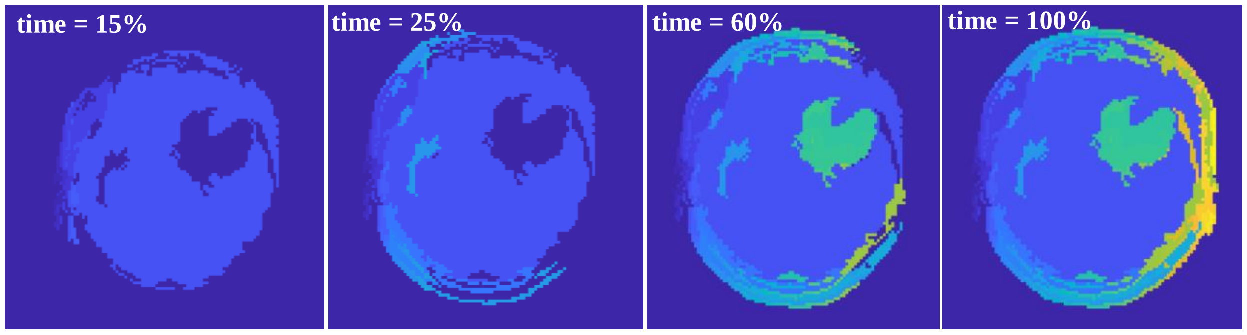

Figure 5: Progression of a standard region-growing applied to a brain image. This procedure was used as initialization for our algorithm. Images from left to right indicate the progression, with the far right image representing the final initialization (initial regions = 420).

Finally, the two additional brain segmentations are shown in Figure 6. Additionally, a plot of the progression of the SMS cost function as a function of iteration is shown in Figure 7.

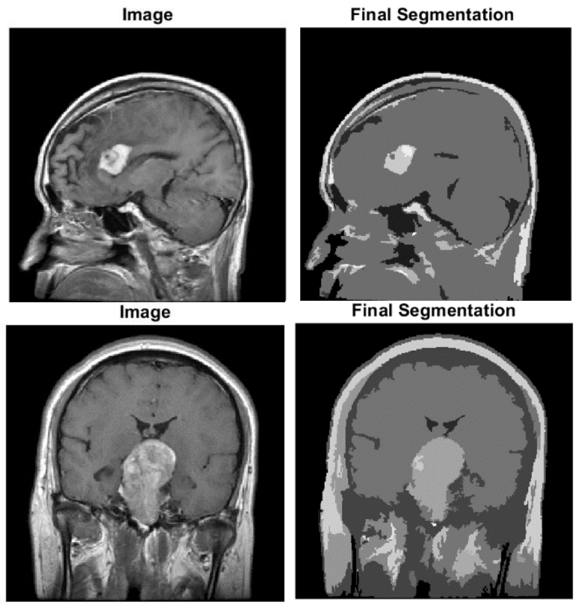

Figure 6: Brain MRI images of 2 patients with large intracranial tumors. The segmentation qualitatively performed well over the tumors of interest. Segmentation for these images were performed using our segmentation algorithm.

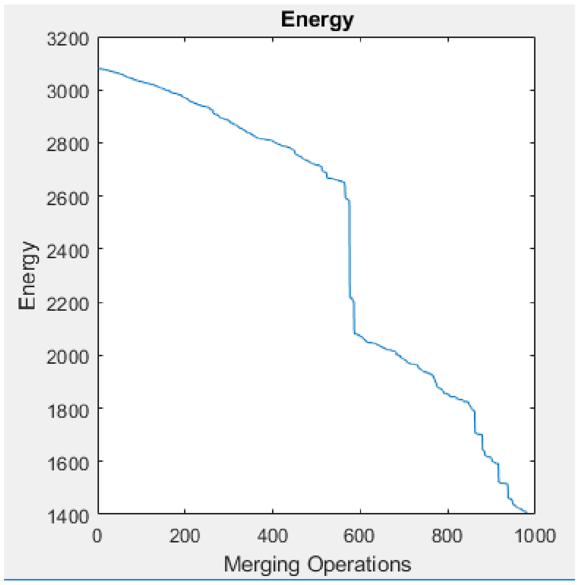

Figure 7: SMS energy as a function of merging operation (iterations). Note the drastic decrease in energy for some iterations. These cases are the result of a large region merging.

4. Discussion and Conclusion

Qualitatively, both algorithms performed quite well, but performance decreased rapidly in the presence of noise. Our novel algorithm did decrease computation times drastically, however the DICE coefficients decreased slightly, indicating that our algorithm was not as accurate as the Koepfler algorithm. In terms of overall effectiveness of the algorithm, this decrease was almost negligible and does not negate the drastic decrease in computation times. For both algorithms, many regularization parameters were tested to find the ideal ν value. In other words, the final segmented image was highly dependent on the regularization parameter. Decreasing the computation time allowed for much quicker testing of suitable ν value, and would be much more practical in a clinical setting where time may be limited. Finding a more rigorous, analytical method for find the best regularization parameter would be useful in saving time and increasing performance.

It should be noted that noise greatly affected the segmented image, and quickly degrades the outcome even with the addition of slight noise. Filtering the images to smooth the noise did create better segmentations. Testing with different initializations may prove to yield even more accurate results, particularly initializations of a stochastic nature, such as genetic algorithms. Due to the deterministic nature of our initialization (starting with the top-left pixel and using a set threshold), it is likely that solutions are constrained to local minima due to the high dimensionality and non-convexity of the problem. We were in fact successful in implementing a genetic segmentation algorithm, however, due to time limits, we could not use this as an initialization. It is anticipated that stochastic algorithms may help find more accurate solutions by “jumping out of local minima”. Another point to make about initialization is that if one sets a high gray-value threshold for the standard region-growing initialization (allowing many pixels to be grouped together) one is greatly restricting the number of possible solutions. This is because the overall amount of regions decrease when the threshold increases, leading to less possible solutions. One of the downsides to our algorithm is that another parameter needs to be optimized, namely the gray-value threshold.

In conclusion, region growing using the simplified Mumford-Shah functional is a practical tool for basic image segmentation when used with proper initialization. However, performance degrades rapidly in the presence of noise. The Koepfler algorithm successfully performed high quality segmentations, but the computation time was relatively high. Our algorithm decreased this computation time drastically by initializing a segmentation using a standard region-growing technique. Our algorithm preserved the quality of the segmentation based on quantitative DICE comparison. These results suggest an improvement upon the Koepfler algorithm, however, more rigorous quantitative comparison are needed to confirm this.

References

1. Pham, D. L., Xu, C. & Prince, J. L. Current methods in medical image segmentation. *Annu Rev Biomed Eng* **2**, 315–337 (2000).
2. Despotović, I., Goossens, B. & Philips, W. MRI Segmentation of the Human Brain: Challenges, Methods, and Applications. *Comput Math Methods Med* **2015**, 1–23 (2015).
3. Gonzalez, R. C. & Woods, R. E. *Digital Image Processing*. (2002).
4. Geman, S. & Geman, D. Stochastic Relaxation, Gibbs Distributions, and the Bayesian Restoration of Images. *IEEE Trans Pattern Anal Mach Intell* **6**, 721–741 (1984).
5. Kato, Z. & Zerubia, J. Markov Random Fields in Image Segmentation. in *Foundations and Trends in Signal Processing* **5**, 1–15 (2011).
6. Pock, T., Cremers, D., Bischof, H. & Chambolle, A. An algorithm for minimizing the Mumford-Shah functional. in *Proceedings of the IEEE International Conference on Computer Vision* 1133–1140 (2009). doi:10.1109/ICCV.2009.5459348
7. Chambolle, A., Cremers, D. & Pock, T. A Convex Approach to Minimal Partitions. *SIAM J Imaging Sci* **5**, 1113–1158 (2011).
8. Pock, T. & Chambolle, A. A convex relaxation approach for computing minimal partitions. in *Proceedings of the IEEE International Conference on Computer Vision* 810–817 (2009).
9. Mumford, D. & Shah, J. Optimal approximations by piecewise smooth functions and associated variational problems. *Commun Pure Appl Math* **42**, 577–685 (1989).
10. Koepfler, G., Lopez, C. & Morel, J. M. A Multiscale Algorithm for Image Segmentation by Variational Method. *SIAM J Numer Anal* **31**, 282–299 (2005).
11. Kistler, M., Bonaretti, S., Pfahrer, M., Niklaus, R. & Buchler, P. The virtual skeleton database: an open access repository for biomedical research and collaboration. *JMIR* **15**, (2013).

Appendix A

Main Script

```
%% Region Growing Segmentation Algorithm
%
% Authors:      Tim Ruesink (truesink@wisc.edu)
%               Lawrence Lechuga (llechuga@wisc.edu)
%               Grant Roberts (gsroberts@wisc.edu)
% Institution:  University of Wisconsin - Madison
% Department:   Mechanical Engineering and Medical Physics
% Last Update:  05/01/2019
% Built for:    MATLAB 2018
%
% Performs region-growing (RG) using a simplified Mumford-Shah functional.
% Initialization is done using region-growing ('grayconnect' function)
% based on a given pixel-value tolerance (tol). Further segmentation is
% done by proposing a merging of two neighboring regions and evaulating if
% this proposed merge decreases the simplified MS functional of the image.
% This process is done iteratively for every initial region. Regularization
% parameter (nu) controls # of regions by weighting region length term.
% Simplified MS: norm(trueImage-approxImage) + length(regions)
%   where: approxImage is a piecewise-constant approximation of true image
%   based on average pixel value in region, and length of regions is the
%   total perimeter length of all regions.
%
% Inspired by: Georges Koepfler and Russell Valentine
% https://coldstonelabs.org/files/science/math/Intro-MS-Valentine.pdf
%
http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.53.1631&rep=rep1&typ
e=pdf

%% Import Data
% Reads in PNG, MAT, or DICOM files
file = 'brain8.mat'; % Name of image file in current directory
if sum(contains(file,'png'))
    rawImage = imread(file);            % read png file
elseif sum(contains(file,'mat'))
    rawImage = load(file);              % load mat file
    rawImage = struct2array(rawImage);
elseif sum(contains(file,'dcm'))
    rawImage = dicomread(file);         % import dcm file
else
    disp('Could not recognize file type. Only reads PNG, MAT, and DICOM
files');
end

%% Assign variables and initialize matrices
nu = 0.1; % regularization parameter
tol = max(rawImage(:))*0.1; % initial RG tolerance, merge pixels w/in tol
trueImage = double(rawImage); % input image

%% Initialize Segmentation
regions = init_seg(trueImage,tol); % initial set of regions
```

```
[approxImage,totalLength] = approximateFull(regions,trueImage); % create
initial segmentation
f_init = approxImage; % grab initial approximation
L = getAllLengths(regions); % array of region edge lengths
E = cost(approxImage,trueImage,totalLength,nu); % compute the cost (mumford
shah)
%% Merge Regions
labels = unique(regions); % get region labels
N_regions = length(labels); % number of regions in initial segementation
L_temp = L;
k = 1;
fig2 = figure(2);
fig2.Position = [50 700 1400 400];
for r = 1:N_regions
    if ismember(r,labels) % if this region still exists
        added = [];
        for p = 1:N_regions
            added_p = added; % used to test if no regions are added (break)
            [regions_temp,added] = merge(regions, r, added); % merge adjacent
regions
            proposedAdd = added(end); % proposed region addition
            L_temp(r) = edgeLength(r,regions_temp); % set new individual
region length
            L_temp(proposedAdd) = 0; % set merged region length to zero
            totalLength = sum(L_temp); % total length
            % [approxTemp,~] = approximateFull(regions,trueImage); %
approximate image
            approxTemp = approximate(r,proposedAdd,regions,approxImage); %
create approximate image
            E2 = cost(approxTemp,trueImage,totalLength,nu); % calculate cost
            if (E2-E(k) < 0) % does the cost decrease by merging the new
region?
                k = k+1;
                regions = regions_temp; % if so, lets change our regions
                L = L_temp;
                E(k) = E2; % add result to energy array
                labels = unique(regions);
                approxImage = approxTemp;

                % Updated regions and energy figures
                subplot(1,3,1); imagesc(regions); title('Regions')
                subplot(1,3,2); plot(1:1:length(E),E); title('Energy');
xlabel('Merging Operations'); ylabel('Energy')
                subplot(1,3,3); bar(length(labels));
set(gca,'xticklabel','Number of Regions')
                drawnow

                % Display outputs
                disp(['Iteration: ',num2str(r),'. Region
',num2str(added(end)), ' added to region ', num2str(r),'. attempted merge:
',num2str((p))])
            elseif size(added_p) == size(added)
                disp(['Iteration: ',num2str(r),'. There are no more adjacent
regions next to region ', num2str(r),'. attempted merge: ',num2str((p))])
                break
            else
```

```
                disp(['Iteration: ',num2str(r),'. Region
',num2str(added(end)), ' NOT added to region ', num2str(r),'. attempted
merge: ',num2str((p))])
                L_temp = L;
            end
        end
    end
end
f_final = approxImage;
regions_final = regions;
N_regions = length(labels);

figure;
subplot(1,3,1); imshow(trueImage,[]); title('Image')
subplot(1,3,2); imshow(f_init,[]); title('Initalization')
subplot(1,3,3); imshow(f_final,[]); title('Final Segmentation')
```

Ancillary Functions

```
%% Initial Segmentation
% Description:
%   Basic region-growing segmentation utilizing 'grayconnected' function
%   to connect regions of like pixels. Also fills in small holes.
% Returns:
%   regions = basic region segmentation (initialization)
% Arguments:
%   trueImage = input image (double)
%   tol = threshold for "like pixels" (see 'grayconnected')
% Dependencies:
%   NONE

function [regions] = init_seg(trueImage,tol)

R_num = 0; % initialize region number
mask = ones(size(trueImage)); % matrix for regions available for merging
regions = zeros(size(trueImage)); % initialize region matrix

while (sum(mask(:)) > 0)
    R_num = R_num + 1; % update region number
    [Rx, Ry] = find(mask == 1,1); % find first instance of new region
    temp_mask = grayconnected(trueImage,Rx,Ry,tol); % connect like pixels
    largeRegions = bwareaopen(~temp_mask, 10); % fill in holes < size 10
    temp_mask = ~largeRegions; % invert to turn region of interest to 1's
    regions(temp_mask) = R_num; % assign this region to a specific region #
    trueImage(temp_mask) = NaN; % exclude region in trueImage for next
grayconnected iter
    mask = mask - temp_mask; % remove already segmented region from mask
    imagesc(regions); axis square; title('Unique Region Labels');
    drawnow;
end

end


%% Approximate Image
% Description:
%   Approximates an image based on regions and true image.
%   Image approximation is done by averaging true image pixel values in
%   each region.
% Returns:
%   approxImage = image approximation (double)
%   totaLength = sum of total length of each region
% Arguments:
%   regions = image of distinct region labels (double)
%   trueImage = input image (double)
% Dependencies:
%   edgeLength.m

function [approxImage,totaLength] = approximateFull(regions,trueImage)

totaLength = 0;
approxImage = zeros(size(regions));
```

```matlab
labels = (unique(regions)); % unique region numbers (labels)
N_regions = length(labels); % number of total distinct regions

for r = 1:N_regions
    areaR = (regions == labels(r)); % returns 1's in the current region, and
zeros elsewhere
    approxImage = approxImage + mean(trueImage(areaR))*areaR; % add
approximation of region r to f
    totalLength = totalLength + edgeLength(labels(r),regions); % iteratively
add region lengths
end

end


%% Get Each Region Edge Length
% Description:
%   Calculates discrete edge length (perimeter) of each region.
%   Note this does not count outside edge of image as a border
% Returns:
%   L = array of region length
% Arguments:
%   regions = image of distinct region labels (double)
% Dependencies:
%   NONE

function L = getAllLengths(regions)

[m,n] = size(regions);
labels = unique(regions); % unique region numbers (labels)
N_regions = length(labels); % number of total distinct regions
L = zeros(N_regions,1); % initialize length array
for r = 1:N_regions
    for i = 1:m
        for j = 1:n
            if regions(i,j) == r
                if (i > 1 && regions(i-1,j) ~= r) % if there is a border b/w
pixel of interest and next pixel
                    L(r) = L(r) + 1; % add 1 to the length
                end
                if (i < m && regions(i+1,j) ~= r)
                    L(r) = L(r) + 1;
                end
                if (j > 1 && regions(i,j-1) ~= r)
                    L(r) = L(r) + 1;
                end
                if (j < n && regions(i,j+1) ~= r)
                    L(r) = L(r) + 1;
                end
            end
        end
    end
end

end
```

```
%% Cost Function
% Description:
%   Calculates cost function (simplified mumford-shah functional) value.
%   Simplified MS: L2-norm(trueImage-approxImage) + length(regions)
%       where: approxImage is a piecewise-constant approx of true image
%       based on average pixel value in region, and length of regions is
%       total perimeter length of all regions.
% Returns:
%   E = cost function value
% Arguments:
%   approxImage = approximation of true image (double)
%   trueImage = input image (double)
%   totaLength = sum of length of all regions
%   nu = regularization parameter
% Dependencies:
%   NONE

function E = cost(approxImage,trueImage,totaLength,nu)

data_term = norm(approxImage-trueImage); % data consistency (L2-norm)
edge_term = nu*totaLength; % regularization limiting length of region edges
E = data_term + edge_term; % cost function value

end


%% Edge Length
% Description:
%   Calculates discrete edge length (perimeter) of a region.
% Returns:
%   regions = new map of distinct region labels
%   added = new proposed region to merge
% Arguments:
%   regions = image of distinct region labels (double)
%   R = region of interest
%   added = regions which have already been tested
% Dependencies:
%   NONE

function [regions,added] = merge(regions, R, added)

flag = 0;
[m,n] = size(regions);
for i = 1:m
    for j = 1:n % iterate left to right
        if (regions(i,j) == R)
            if (i > 1 && regions(i-1,j) ~= R && sum(eq(regions(i-
1,j),added))==0) % check row above
                % if there is a border between the region of interest
                added = [added, regions(i-1,j)]; % region of proposed merge
                mask = (regions == regions(i-1,j)); % create mask of region
to be added
                regions(mask) = R; % merge regions
                flag = 1;
                break
            end
            if (i < m && regions(i+1,j) ~= R &&
sum(eq(regions(i+1,j),added))==0) % check row below
                added = [added, regions(i+1,j)];
                mask = (regions == regions(i+1,j));
                regions(mask) = R;
                flag = 1;
                break
            end
            if (j > 1 && regions(i,j-1) ~= R && sum(eq(regions(i,j-
1),added))==0) % check column to the left
                added = [added, regions(i,j-1)];
                mask = (regions == regions(i,j-1));
                regions(mask) = R;
                flag = 1;
                break
            end
            if (j < n && regions(i,j+1) ~= R &&
sum(eq(regions(i,j+1),added))==0) % check column to the right
                added = [added, regions(i,j+1)];
                mask = (regions == regions(i,j+1));
                regions(mask) = R;
                flag = 1;
                break
            end
        end
    end
    if (flag == 1) % break if a proposal is found
        break
    end
end

end


%% Approximate Image
% Description:
%   Approximates an image based on regions and true image.
%   Image approximation is done by averaging true image pixel values in
%   each region. Time is saved by importing an already approximated image,
%   only merging new regions.
% Returns:
%   approxImage = image approximation (double)
% Arguments:
%   r = current region
%   proposedAdd = region of proposed merge
%   regions = image of distinct region labels (double)
%   approxImage = initial image approximation (double)
% Dependencies:
%   NONE

function approxImage = approximate(r,proposedAdd,regions,approxImage)

maskR = regions==r; % mask area of current region
maskP = regions==proposedAdd; % mask area of proposed region
mask = maskR | maskP; % create mask of combined region
newMean = mean(approxImage(mask)); % calculate mean of combined region
```

```matlab
approxImage(mask) = newMean; % make new approximate image

end
```