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

Grant S. Roberts1, Lawrence Lechuga1, Timothy Ruesink2

*1Department of Medical Physics, University of Wisconsin-Madison, WI*

*2Department 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 sub-regions 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**.

**Specific Aim 1**: **Reproduce an existing region-growing segmentation algorithm1,2 using a simplified Mumford-Shah functional.**

**Specific Aim 2**: **Develop a novel region-growing algorithm using the same formulation as the previous algorithm but with added flexibility in initialization conditions and increased efficiency in calculating the cost function.**

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.

**Introduction**

Segmentation is an image processing technique that takes advantage of one or more imaging features to partition the image (or set of images) into useful regions of interest. 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).

State hypothesis and purpose

**Theory and Methods**

In regards to **specific aim** 1, this project seeks to reproduce an automatic region-growing based algorithm that was discovered during the literature search process. This algorithm iteratively minimizes a simplified Mumford-Shah cost functional and assumes every segmented regions are piece-wise constant values. The performance of this algorithm will first be tested using a modified Shepp-Logan brain phantom in MATLAB (Mathworks, Natick, MA) with varying degrees of zero-mean noise added (σ = 0, σ = 0.002, and σ = 0.005).

**Results**

**Discussion**

**Conclusion**

**References**

1. Koepfler, G., Lopez, C. & Morel, J. M. A Multiscale Algorithm for Image Segmentation by Variational Method. *SIAM J Numer Anal* **31**, 282–299 (2005).

2. Valentine, R. *Image Segmentation with the Mumford Shah Functional*. (2007).

**Appendix**

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&type=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,totaLength] = approximateFull(regions,trueImage); % create initial segmentation

f\_init = approxImage; % grab initial approximation

L = getAllLengths(regions); % array of region edge lengths

E = cost(approxImage,trueImage,totaLength,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

totaLength = sum(L\_temp); % total length

% [approxTemp,~] = approximateFull(regions,trueImage); % approximate image

approxTemp = approximate(r,proposedAdd,regions,approxImage); % create approximate image

E2 = cost(approxTemp,trueImage,totaLength,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));

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

totaLength = totaLength + 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

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

end