**EECE 8395**

**Project 6**

**Graph Cut**

**Tahsin Reasat**

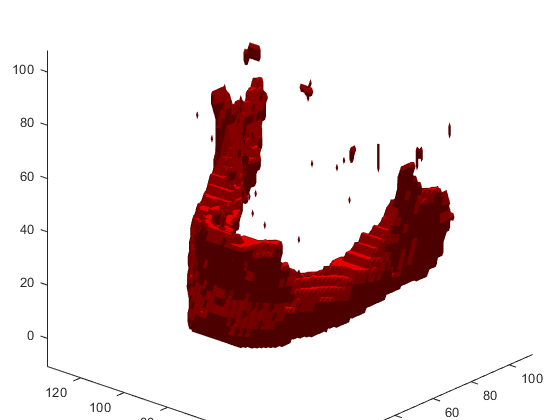
**ID: 000614908**

****



Sigma=0.2, Lambda =0.1, Noise=0.4

Sigma=0.8, Lambda =0.1, Noise=0.4Sigma=0.8, Lambda =0.04, Noise=0.4



Sigma=200, Lambda=0.025, Dice Score: 0.8332

Graphcut2D

function result = Graphcut2D(sigma, lambda, noise,imshow)

if nargin<4

imshow=0;

end

% Initialize all of our helper Hash Tables:

global Parent Tree Active EdgeCaps Edges Edge\_Lens r c Orphans Orphan\_cnt PLengths

r=50;

c=50;

% noise = 0.4;

img = zeros(r,c);

img(15:35,15:35) = 1;

rng('default');

img = img +noise\*randn(r,c);

img(img(:)<0.01)=0.01;

img(img(:)>.99)=.99;

if imshow==1

figure(1); close(1); figure(1); colormap(gray(256));

image(255\*img);

end

% Parameters in our Graph Cut

% sigma = 0.2;

% lambda = 0.1;

Tree = zeros(r\*c+2,1);

Tree(r\*c+1)=1; % s

Tree(r\*c+2)=2; % t

% The first r\*c entries in Tree will contain our final segmentation â€“ 1 for s and 2 for t

Parent = zeros(r\*c+2,1);

Active = zeros(r\*c+2,1);

nodes = (1:r\*c)';

X = mod(nodes-1,r)+1;

Y = floor((nodes-1)/r)+1;

Edges = [X<r,X>1,Y<c,Y>1].\*[nodes+1, nodes-1, nodes+r, nodes-r];

Edge\_Lens = sum(Edges'>0)';

EdgeCaps = zeros(r\*c,6);

EdgeCaps(:,5) = -lambda\*log(1-img(:)');%s

EdgeCaps(:,6) = -lambda\*log(img(:)');%t

EdgeCaps(:,1) = (1-lambda)\*reshape([exp(-((img(1:end-1,:)-img(2:end,:)).^2)/(2\*sigma\*sigma));-ones(1,c)],[r\*c,1]);%x+1

EdgeCaps(:,2) = (1-lambda)\*reshape([-ones(1,c);exp(-((img(1:end-1,:)-img(2:end,:)).^2)/(2\*sigma\*sigma))],[r\*c,1]);%x-1

EdgeCaps(:,3) = (1-lambda)\*reshape([ exp(-((img(:,1:end-1)-img(:,2:end)).^2)/(2\*sigma\*sigma)),-ones(r,1)],[r\*c,1]);%y+1

EdgeCaps(:,4) = (1-lambda)\*reshape([-ones(r,1), exp(-((img(:,1:end-1)-img(:,2:end)).^2)/(2\*sigma\*sigma))],[r\*c,1]);%y-1

for i=1:r\*c

EdgeCaps(i,1:Edge\_Lens(i)) = EdgeCaps(i,Edges(i,1:4)>0);

Edges(i,1:Edge\_Lens(i)) = Edges(i,Edges(i,1:4)>0);

end

% PLengths will help us keep track of the length of the current path to each node â€“ this will help us

% construct shorter paths in the adoption step:

PLengths = zeros(r\*c+2,1);

% We also need Orphans to be a global so that they can be identified in the Augment step and

% addressed in the Adoption step:

Orphans = zeros(1,10000);

Orphan\_cnt = 0;

% Finally, we can actually do the first grow step ourselves in a way that guarantees the initial tree

% assignments match our prior probability distribution function. This reduces a lot of orphan

% adoption operations that would occur if we instead connected all nodes to s initially:

msk = img(:)>0.5;

Active(1:r\*c)=1;

Tree(msk)=1;

Tree(~msk)=2;

Parent(msk)=r\*c+1;

Parent(~msk)=r\*c+2;

% Finally add all the active nodes to the FIFO

FIFOInit(10\*r\*c);

for i=1:length(nodes)

FIFOInsert(nodes(i));

end

% And perform the min-cut:

iter=0;

while (1)

iter=iter+1;

P = Grow();

if ActiveCheck>0

disp('non-active nodes adjacent to free nodes found')

end

if isempty(P)

break;

end

Augment(P);

Adoption()

end

res=Tree(1:r\*c);

if imshow==1

figure(2); close(2); figure(2); colormap(gray(256));

image(reshape(Tree(1:r\*c)\*255/2,[r,c]));

end

Graphcut3D

function result = GraphCut3D(img,hist\_fore,hist\_back,binsize,sigma,lambda,imshow)

if nargin<7

imshow=0;

end

global Parent Tree Active EdgeCaps Edges Edge\_Lens r c d Orphans Orphan\_cnt PLengths;

[r,c,d] = size(img.data);

% Initialize all our helper variables and set Edge capacities

FIFOInit(2\*r\*c\*d);

Tree = zeros(r\*c\*d+2,1);

PLengths = zeros(r\*c\*d+2,1);

PLengths(1:r\*c\*d)=1;

Parent = zeros(r\*c\*d+2,1);

Active = zeros(r\*c\*d+2,1);

nodes = [1:r\*c\*d]';

X = mod(nodes-1,r)+1;

Z = floor((nodes-1)/(r\*c))+1;

Y = floor((nodes-1-(Z- 1)\*r\*c)/r)+1;

Edges = [X<r,X>1,Y<c,Y>1,Z<d,Z>1].\*[nodes+1, nodes-1, nodes+r, nodes-r, nodes+r\*c, nodes-r\*c];

Edge\_Lens = sum(Edges'>0)';

EdgeCaps = zeros(r\*c\*d,8);

D = -ones(r,c,d);

D(1:end-1,:,:) = exp(-((img.data(1:end-1,:,:)-img.data(2:end,:,:)).^2)/(2\*sigma\*sigma));

EdgeCaps(:,1) = (1-lambda)\*reshape(D,[r\*c\*d,1]);%x+1

D = -ones(r,c,d);

D(2:end,:,:) = exp(-((img.data(1:end-1,:,:)-img.data(2:end,:,:)).^2)/(2\*sigma\*sigma));

EdgeCaps(:,2) = (1-lambda)\*reshape(D,[r\*c\*d,1]);%x-1

D = -ones(r,c,d);

D(:,1:end-1,:) = exp(-((img.data(:,1:end-1,:)-img.data(:,2:end,:)).^2)/(2\*sigma\*sigma));

EdgeCaps(:,3) = (1-lambda)\*reshape(D,[r\*c\*d,1]);%y+1

D = -ones(r,c,d);

D(:,2:end,:) = exp(-((img.data(:,1:end-1,:)-img.data(:,2:end,:)).^2)/(2\*sigma\*sigma));

EdgeCaps(:,4) = (1-lambda)\*reshape(D,[r\*c\*d,1]);%y-1

D = -ones(r,c,d);

D(:,:,1:end-1) = exp(-((img.data(:,:,1:end-1)-img.data(:,:,2:end)).^2)/(2\*sigma\*sigma));

EdgeCaps(:,5) = (1-lambda)\*reshape(D,[r\*c\*d,1]);%z+1

D = -ones(r,c,d);

D(:,:,2:end) = exp(-((img.data(:,:,1:end-1)-img.data(:,:,2:end)).^2)/(2\*sigma\*sigma));

EdgeCaps(:,6) = (1-lambda)\*reshape(D,[r\*c\*d,1]);%z-1

EdgeCaps(:,7) = -lambda\*log(hist\_back(floor((img.data(:)'-mn)/binsize)+1));%s

EdgeCaps(:,8) = -lambda\*log(hist\_fore(floor((img.data(:)'-mn)/binsize)+1));%t

K = max(sum((EdgeCaps(:,1:6).\*(EdgeCaps(:,1:6)>0))'))+1;

for i=1:size(fore,1)

for j=max([1,floor(fore(i,1)-fore(i,4))]):min([img.dim(1),ceil(fore(i,1)+fore(i,4))])

for k=max([1,floor(fore(i,2)-fore(i,4))]):min([img.dim(2),ceil(fore(i,2)+fore(i,4))])

if (j- fore(i,1))\*(j-fore(i,1))+(k-fore(i,2))\*(k-fore(i,2))<fore(i,4)\*fore(i,4)

EdgeCaps(j+r\*((k-1)+c\*(fore(i,3)-1)),7) = K;

EdgeCaps(j+r\*((k-1)+c\*(fore(i,3)-1)),8) = 0;

end

end

end

end

for i=1:size(back,1)

for j=max([1,floor(back(i,1)-back(i,4))]):min([img.dim(1),ceil(back(i,1)+back(i,4))])

for k=max([1,floor(back(i,2)-back(i,4))]):min([img.dim(2),ceil(back(i,2)+back(i,4))])

if (j- back(i,1))\*(j-back(i,1))+(k-back(i,2))\*(k-back(i,2))<back(i,4)\*back(i,4)

EdgeCaps(j+r\*((k-1)+c\*(back(i,3)-1)),7) = 0;

EdgeCaps(j+r\*((k-1)+c\*(back(i,3)-1)),8) = K;

end

end

end

end

% Clean up our edges at our border voxels:

for i=1:r\*c\*d

EdgeCaps(i,1:Edge\_Lens(i)) = EdgeCaps(i,Edges(i,1:6)>0);

Edges(i,1:Edge\_Lens(i)) = Edges(i,Edges(i,1:6)>0);

end

% Now we can initialize our search tree

msk = EdgeCaps(:,7)>EdgeCaps(:,8);

FIFOInsert(nodes);

Active(1:r\*c\*d)=1;

Tree(msk)=1;

Tree(~msk)=2;

Tree(r\*c\*d+1)=1;

Tree(r\*c\*d+2)=2;

Parent(msk)=r\*c\*d+1;

Parent(~msk)=r\*c\*d+2;

Orphans = zeros(1,10000);

Orphan\_cnt = 0;

iter=0;

tic

while (1)

iter=iter+1;

P = Grow();

if isempty(P)

break;

end

Augment(P);

Adoption();

if mod(iter,5000)==0

disp(['iteration: ' num2str(iter)])

CutCheck()

end

end

toc

result = reshape(Tree(1:r\*c\*d),[r,c,d]);

if imshow==1

figure();

msh = isosurface(res,1.5);

msh.vertices = msh.vertices.\*repmat(img.voxsz,[length(msh.vertices),1]);

DisplayMesh(msh,[1,0,0],1)

end

Grow

function P=Grow()

global Parent Tree Active EdgeCaps Edges Edge\_Lens PLengths

P=[];

while FIFOLen

p=FIFOPeek;

if Active(p)==1

if p<=length(Edge\_Lens)

neibs=[Edges(p,1:Edge\_Lens(p)) length(Edge\_Lens)+1 length(Edge\_Lens)+2];

else

nodes=1:length(Edge\_Lens);

if p==length(Edge\_Lens)+1

neibs=nodes(EdgeCaps(1:length(Edge\_Lens),end-1)>0);

end

if p==length(Edge\_Lens)+2

neibs=nodes(EdgeCaps(1:length(Edge\_Lens),end)>0);

end

end

for i = 1:length(neibs)

q=neibs(i);

cap=EdgeFunc(p,q);

if cap>0

% Add free nodes

if Tree(q)==0

Active(q)=1;

Tree(q)=Tree(p);

Parent(q)=p;

PLengths(q)=PLengths(q)+1;

FIFOInsert(q)

end

% Return path

if Tree(q)~=0

if Tree(q)~=Tree(p)

P=TracePath(p,q);

return

end

end

end

end

end

FIFOPop;

Active(p)=0;

end

Augment

function Augment(P)

global Parent Orphans Orphan\_cnt Tree PLengths

% find bottleneck capacity

bot\_cap=1e6;

for i\_node=2:length(P)

p=P(i\_node-1);

q=P(i\_node);

cap=EdgeFunc(p,q);

if bot\_cap>cap

bot\_cap=cap;

end

end

%% push flow

for i\_node=2:length(P)

p=P(i\_node-1);

q=P(i\_node);

cap\_res=PushFlow(p,q,bot\_cap);

if cap\_res==0

if Tree(p)==1 && Tree(q)==1

Parent(q)=0;

PLengths(q)=0;

Orphan\_cnt=Orphan\_cnt+1;

Orphans(Orphan\_cnt)=q;

end

if Tree(p)==2 && Tree(q)==2

Parent(p)=0;

PLengths(p)=0;

Orphan\_cnt=Orphan\_cnt+1;

Orphans(Orphan\_cnt)=p;

end

end

end

Adoption

function Adoption

global Parent Tree Active Edges Edge\_Lens Orphans Orphan\_cnt PLengths

while Orphan\_cnt

p=Orphans(Orphan\_cnt);

Orphan\_cnt=Orphan\_cnt-1;

neibs=[Edges(p,1:Edge\_Lens(p)) length(Edge\_Lens)+1 length(Edge\_Lens)+2];

% sort neibs according to PLengths

[~,ind]=sort(PLengths(neibs));

neibs=neibs(ind);

for i=1:length(neibs)

q=neibs(i);

% check if path to tree exists

current\_node=q;

while Parent(current\_node)~=0

current\_node=Parent(current\_node);

end

if current\_node>length(Edge\_Lens)

path2tree=1;

else

path2tree=0;

end

% get capacity

cap=EdgeFunc(p,q);

if Tree(p)==Tree(q) && cap>0 && path2tree

Parent(p)=q;

PLengths(p)=PLengths(q)+1;

break

% Active() state of p remains unchanged

end

end

if Parent(p)==0 % no valid neighbours

neibs=[Edges(p,1:Edge\_Lens(p)) length(Edge\_Lens)+1 length(Edge\_Lens)+2];

for i=1:length(neibs)

q=neibs(i);

if Parent(q)==p

Orphan\_cnt=Orphan\_cnt+1;

Orphans(Orphan\_cnt)=q;

Parent(q)=0;

PLengths(q)=0;

end

if EdgeFunc(p,q)>0 && Tree(q)~=0

FIFOInsert(q)

Active(q)=1;

end

end

Tree(p)=0;

Active(p)=0;

Parent(p)=0;

PLengths(p)=0;

end

end

FIFO Functions

function FIFOInit(initlen)

global fifo

if nargin<1

initlen=1000;

end

fifo=struct;

fifo.start=1;

fifo.stop=0;

fifo.q=zeros(1,initlen);

function FIFOInsert(node)

global fifo

for i=1:length(node)

fifo.stop=fifo.stop+1;

fifo.q(fifo.stop)=node(i);

end

function len=FIFOLen

global fifo

if fifo.start > fifo.stop

len=0;

else

len=length(fifo.q(fifo.start:fifo.stop));

end

function node=FIFOPeek()

global fifo

if fifo.start > fifo.stop

node=[];

else

node=fifo.q(fifo.start);

end

function node=FIFOPop()

global fifo

if fifo.start > fifo.stop

node=[];

else

node=fifo.q(fifo.start);

fifo.start=fifo.start+1;

end

Helper Functions

function cap\_res=PushFlow(p,q,bot\_cap)

% returns residual capacity between nodes after a max flow operation

global Edges EdgeCaps Edge\_Lens

np=min(p,q);

nq=max(p,q);

if nq>length(Edge\_Lens)

if nq==length(Edge\_Lens)+1

EdgeCaps(np,end-1)=EdgeCaps(np,end-1)-bot\_cap;

cap\_res=EdgeCaps(np,end-1);

end

if nq==length(Edge\_Lens)+2

EdgeCaps(np,end)=EdgeCaps(np,end)-bot\_cap;

cap\_res=EdgeCaps(np,end);

end

else

EdgeCaps(np,Edges(np,1:Edge\_Lens(np))==nq)=EdgeCaps(np,Edges(np,1:Edge\_Lens(np))==nq)-bot\_cap;

cap\_res=EdgeCaps(np,Edges(np,1:Edge\_Lens(np))==nq);

end

function cap=EdgeFunc(p,q)

% returns capacity between two nodes

global Edges EdgeCaps Edge\_Lens

np=min(p,q);

nq=max(p,q);

if nq>length(Edge\_Lens)

if nq==length(Edge\_Lens)+1

cap=EdgeCaps(np,end-1);

end

if nq==length(Edge\_Lens)+2

cap=EdgeCaps(np,end);

end

else

cap=EdgeCaps(np,Edges(np,1:Edge\_Lens(np))==nq);

end

function totcap = ActiveCheck

global EdgeCaps Edges Edge\_Lens Tree Active

totcap=0;

for i=1:length(Edge\_Lens)

if Tree(i)~=0 && Active(i)==0

for j=1:Edge\_Lens(i)

if Tree(Edges(i,j))==0

p = min([i,Edges(i,j)]);

q = max([i,Edges(i,j)]);

for k=1:Edge\_Lens(p)

if Edges(p,k)==q

totcap = totcap + EdgeCaps(p,k);

break;

end

end

end

end

end

end

if sum(Tree(1:length(Edge\_Lens))==0)>0

if Active(length(Edge\_Lens)+1)==0

totcap = totcap + sum(EdgeCaps(Tree(1:length(Edge\_Lens))==0,end-1));

end

if Active(length(Edge\_Lens)+2)==0

totcap = totcap + sum(EdgeCaps(Tree(1:length(Edge\_Lens))==0,end));

end

end

function totcap=CutCheck

global EdgeCaps Edges Edge\_Lens Tree

totcap=0;

for i=1:length(Edge\_Lens)

for j=1:Edge\_Lens(i)

if Tree(Edges(i,j))~=Tree(i)

p = min([i,Edges(i,j)]);

q = max([i,Edges(i,j)]);

for k=1:Edge\_Lens(p)

if Edges(p,k)==q

totcap = totcap + EdgeCaps(p,k);

break;

end

end

end

end

if Tree(i)~=1

totcap = totcap + EdgeCaps(i,end-1);

end

if Tree(i)~=2

totcap = totcap + EdgeCaps(i,end);

end

end