ENEE731 Project
Normalized Cuts and Image Segmentation

Naotoshi Seo, sonots@umd.edu

November 8, 2006

1 Introduction

Shi and Malik (1997) [1] proposed the Normalized Cuts for image segmentation problem, which is based on Graph Theory. This algorithm treats an image pixel as a node of graph, and considers segmentation as a graph partitioning problem. The Normalized Cuts algorithm measures both the total dissimilarity between the different groups as well as the total similarity within the groups. Amazingly, the optimal solution of splitting points is easily computed by solving a generalized eigenvalue problem.

2 Fundamental Idea

A graph $\mathbf{G} = (\mathbf{V}, \mathbf{E})$ can be partitioned into two disjoint sets, A, B. The degree of dissimilarity between these two pieces can be computed as

$$cut(A, B) = \sum_{u \in A, v \in B} w(u, v). \tag{1}$$

where $w(u, v)$ is the similarity between node u and v. The optimal bipartitioning of a graph is the one that minimizes this cut value. Finding the *minimum cut* is a well-studied problem and there exist efficient algorithms for solving it. However, the minimum cut criteria favors cutting small sets of isolated nodes in the graph, and gives bad partition in some cases such as Fig. 1.

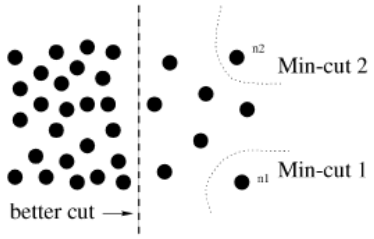


Figure 1: A case where minimum cut gives a bad partition.

Shi an Malik proposed a new measure o disassociation, the *normalized cut* (Ncut):

$$Ncut(A, B) = \frac{cut(A, B)}{assoc(A, V)} + \frac{cut(A, B)}{assoc(B, V)}, \tag{2}$$

where $assoc(A,V) = \sum_{u \in A, t \in V} w(u,t)$ is the total connection from nodes in A to all nodes in the graph and $assoc(B,V)$ is similarly defined.

Let $\mathbf{d}(i) = \sum_j w(i,j)$ be the total connection from the node i to all other nodes. Let $\mathbf{D}$ be an $N \times N$ diagonal matrix with $\mathbf{d}$ on its diagonal, $\mathbf{W}$ be an $N \times N$ symmetric matrix with $\mathbf{W}(i,j) = w(i,j)$. Then it turns out that we can minimize Ncut(A,B) by

$$\min_{A,B} Ncut(A,B) = \min_{\mathbf{y}} \frac{\mathbf{y}^T(\mathbf{D}-\mathbf{W})\mathbf{y}}{\mathbf{y}^T\mathbf{D}\mathbf{y}}. \tag{3}$$

If $\mathbf{y}$ is relaxed to take real values, the above equation can be minimized by solving the generalized eigen value system,

$$(\mathbf{D}-\mathbf{W})\mathbf{y} = \lambda\mathbf{D}\mathbf{y}. \tag{4}$$

Amazingly, the second smallest eigenvector $\mathbf{y}$ gives the solution of the normalized cut problem.

3 Algorithm

Given an image sequence $\mathbf{I}$. Construct a weighted graph $\mathbf{G} = (\mathbf{V}, \mathbf{E})$ whose each node is each pixel of the image $\mathbf{I}$. Let N be the number of nodes (pixels), i.e., $|\mathbf{V}|$.

Step 1

Construct an $N \times N$ symmetric similarity matrix $\mathbf{W}$ as:

$$w_{ij} = exp\frac{-||\mathbf{F}(i)-\mathbf{F}(j)||_2^2}{\sigma_I^2} * \begin{cases} exp\frac{-||\mathbf{X}(i)-\mathbf{X}(j)||_2^2}{\sigma_X^2} & \text{if } ||\mathbf{X}(i)-\mathbf{X}(j)||_2 < r \\ 0 & \text{otherwise} \end{cases} \tag{5}$$

where $\mathbf{X}(i)$ is the spatial location of node i, i.e., the coordinates in the original image $\mathbf{I}$, and $\mathbf{F}(i)$ is a feature vector defined as:

- $\mathbf{F}(i) = 1$ for segmenting point sets,
- $\mathbf{F}(i) = \mathbf{I}(i)$, the intensity value, for segmenting brightness (gray scale) images,
- $\mathbf{F}(i) = [v, u \cdot s \cdot \sin(h), v \cdot s \cdot \cos(h)](i)$, where h, s, v are the HSV values, for color segmentation,
- $\mathbf{F}(i) - [|\mathbf{I} * f_1|, ..., |\mathbf{I} * f_n|](i)$, where the f_i are DOOG filters at various scales and orientations, for texture segmentation.

Let $\mathbf{d}_i = \sum_j w_{ij}$ be the total connection from node i to all other nodes.
Construct an $N \times N$ diagonal matrix $\mathbf{D}$ with $\mathbf{d}$ on its diagonal.

Step 2

Solve a generalized eigensystem,

$$(\mathbf{D}-\mathbf{W})\mathbf{y} = \lambda\mathbf{D}\mathbf{y}, \tag{6}$$

and get an eigenvector with the second smallest eigenvalue. Fortunately, matlab has a function, eigs, to solve generalized eigensystems.

Step 3

Use the eigenvector to bipartition the graph. In the ideal case, the eigenvector should only take on two discrete values, and the signs tell us exactly how to partition the graph ($\mathbf{A} = \{\mathbf{V}_i | \mathbf{y}_i > 0\}$, $\mathbf{B} = \{\mathbf{V}_i | \mathbf{y}_i <= 0\}$).

However, $\mathbf{y}$ is relaxed to take real values, therefore, we need to choose a splitting point. There are several ways such as

- Take 0
- Take median
- Search a splitting point which results in that Ncut(A, B) is minimized.

The splitting point which minimizes Ncut value also minimizes

$$\frac{\mathbf{y}^T(\mathbf{D}-\mathbf{W})\mathbf{y}}{\mathbf{y}^T\mathbf{D}\mathbf{y}} \tag{7}$$

where $\mathbf{y} = (\mathbf{1}+\mathbf{x}) - b(\mathbf{1}-\mathbf{x})$ where $b = k/(1-k)$ where

$$k = \frac{\sum_{x_i>0}\mathbf{d}_i}{\sum_i \mathbf{d}_i}$$

where $\mathbf{x}$ is an N dimensional indicator vector, $x_i = 1$ if node i is in A and -1, otherwise.

To find the minimal Ncut, we need to try different values of splitting points. The optimal splitting point is generally around the mean value of the obtained eigenvector. Fortunately, matlab has a function, fminsearch, which is suitable for this purpose.

Step 4

Repeat bipartition recursively. Stop if *Ncut* value is larger than a pre-specified threshold value (Large Ncut value means that there is no clear partition point any more). Furthermore, stop if the total number of nodes in the partition (*Area*) is smaller than a pre-specified threshold value (this is another criteria added newly to the paper's algorithm.)

4 Experiments and Results

Gray scale Image

Fig. 2 shows the result of segmentation of a gray scale image. The image was obtained from [4].

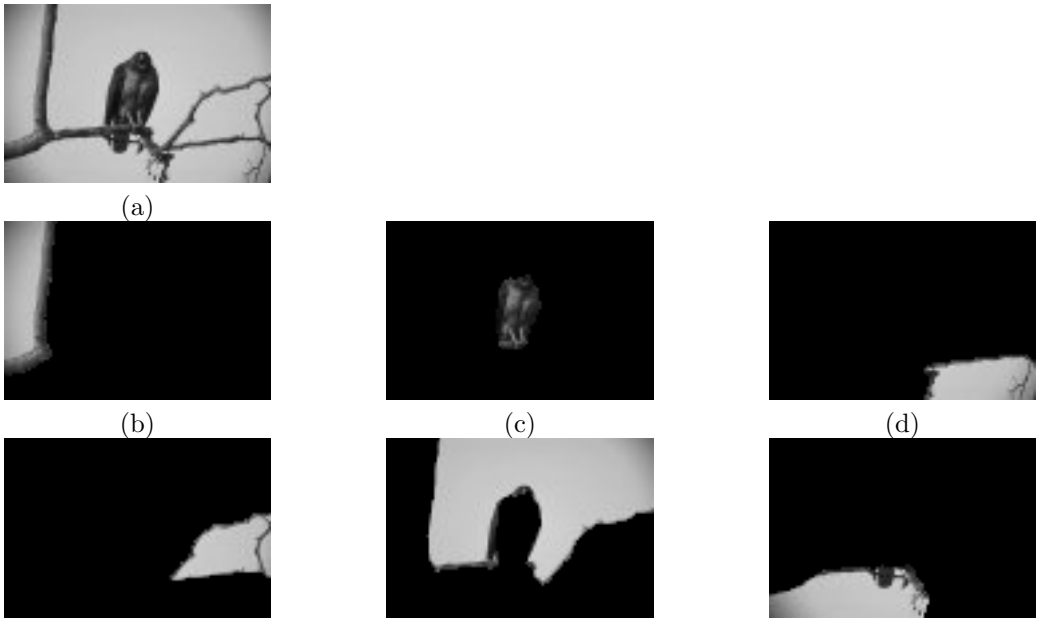


Figure 2: (a) shows the gray scale original image of size 100x67. Image intensity is normalized to lie within 0 and 255. Subplots (b)-(g) show the components of the partition with $Ncut$ value less than 0.14, $Area$ size more than 220. Parameter setting: $\sigma_I = 5.0, \sigma_X = 4.0, r = 1.5$.

Color Image

Fig. 3 shows the result of segmentation of a color image. The image was obtained from [4].

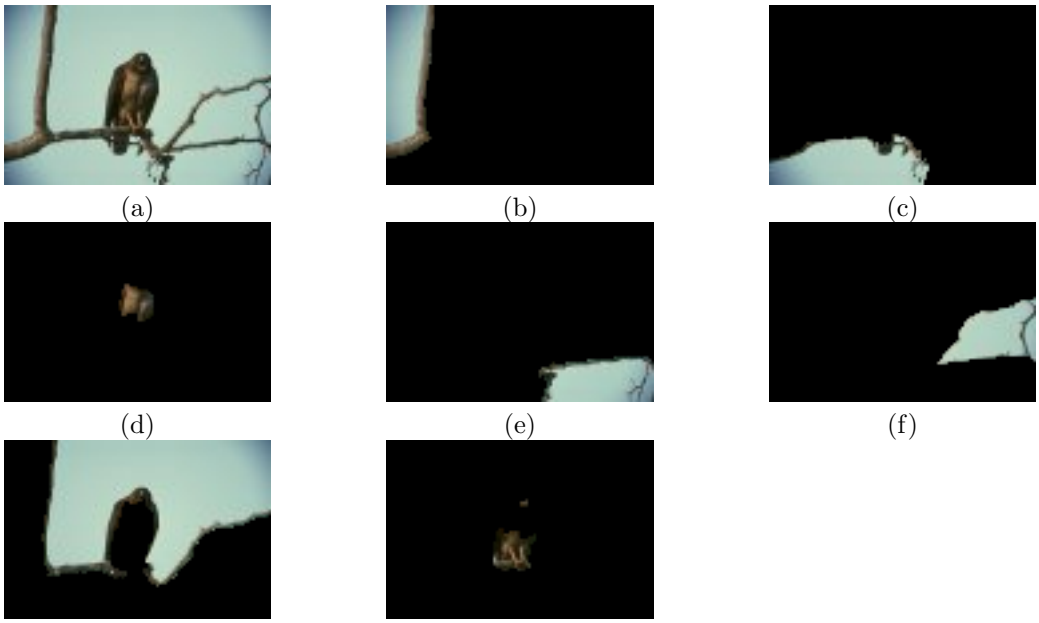
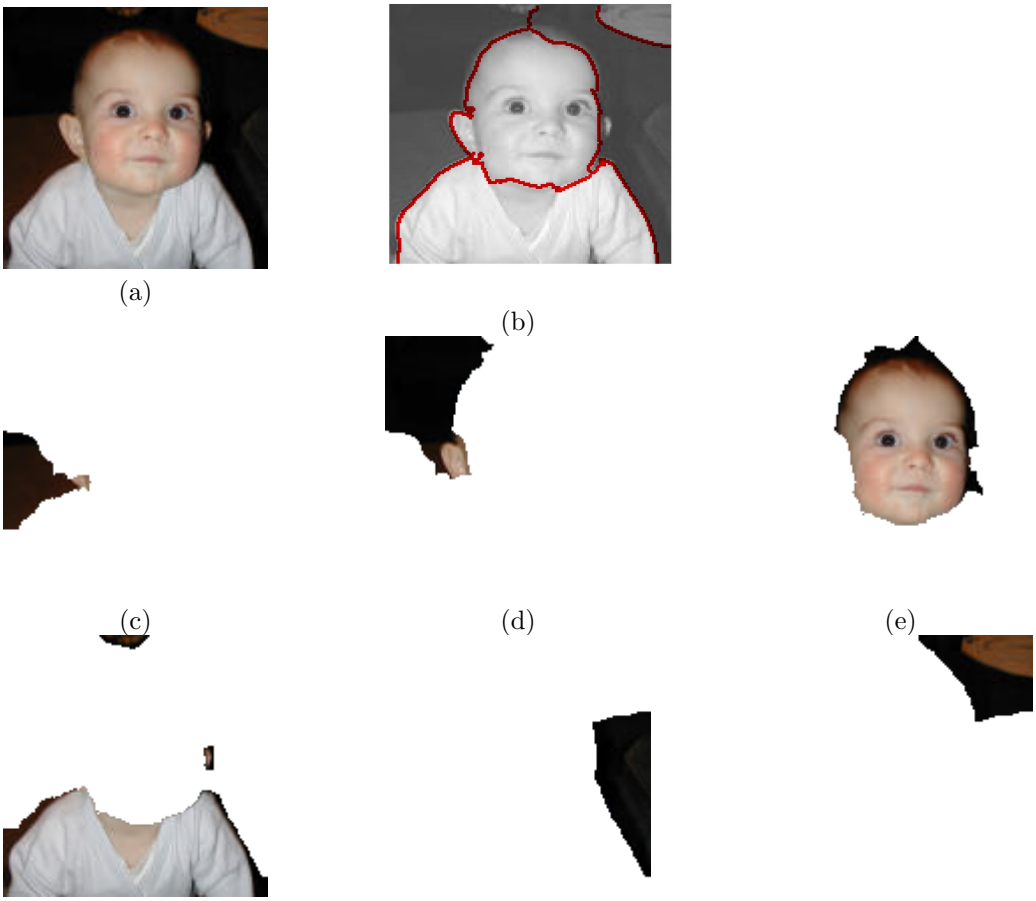


Figure 3: (a) shows the color original image of size 100x67. Each RGB intensity is normalized to lie within 0 and 255. Subplots (b)-(h) show the components of the partition with $Ncut$ value less than 0.21, $Area$ size more than 120. Parameter setting: $\sigma_I = 5.0, \sigma_X = 6.0, r = 1.5$.

Comparison

Fig. 4 shows a comparison with a result of the program provided by Dr. Shi [3]. The input image was also obtained from [3]. The Dr. Shi's program took 77.6793 seconds for computation, but my program took 22.6640 seconds.



(a) (b) (c) (d) (e) (f) (g) (h)

Figure 4: (a) shows the color original image of size 160x160. (b) shows the result by Dr. Shi's program. Subplots (c)-(h) show the result by my program with $Ncut$ value less than 0.04, $Area$ size more than 1000. Parameter setting: $\sigma_I = 5.0, \sigma_X = 6.0, r = 1.5$.

5 Discussion

My program worked reasonably fast for small images (e.g., 100 x 64), but it worked impractically for large images (e.g., 481 x 321). In particular, the calculation of eigen vectors and weight matrix required a lot of time, therefore, more efficient algorithm to compute them, and efficient tricks for memory management is required (they requires much of memory). Of course, we should consider implementing in low level languages such as C and using matlab mex, or running on powerful machines to shorten computation time, too.

My program worked faster than the program provided by Dr. Shi although his program is implemented by C and using matlab mex. It is probably because the Dr. Shi's program is more complicated or my matlab trick for fast computation worked well.

In order to obtain desired results, I had to carefully choose parameters. This is another optimization issue which must be solved in the future.

The electric version of this document and source codes is available at `http://note.sonots.com/index.php?SciSoftware%2FNcutImageSegmentation`

Bibliography

[1] Jianbo Shi and Jitendra Malik, "Normalized Cuts and Image Segmentation," *IEEE Transactions on PAMI*, Vol. 22, No. 8, Aug. 2000. `http://www.cs.berkeley.edu/~malik/papers/SM-ncut.pdf`

[2] Jianbo Shi, "Graph Based Image Segmentation Tutorial", *CVPR*, June. 2004. `http://www.cis.upenn.edu/~jshi/GraphTutorial/`

[3] Jianbo Shi, "MATLAB Normalized Cuts Segmentation Code", 2006 (present). `http://www.cis.upenn.edu/~jshi/software/`

[4] D. Martin and C. Fowlkes and D. Tal and J. Malik, "A Database of Human Segmented Natural Images and its Application to Evaluating Segmentation Algorithms and Measuring Ecological Statistics", *Proc. 8th Int'l Conf. Computer Vision*, vol. 2, pp. 416-423, July 2001. `http://www.eecs.berkeley.edu/Research/Projects/CS/vision/bsds/`

6 Appendix: Source Code

NcutSegment.m (main)

```
% NcutSegment: Normalized Cuts and Image Segmentation
%
%  segI = NcutSegment(I, SI, SX, r, sNcut, sArea)
%
%  Input and output arguments ([]'s are optional):
%   I  (matrix) of size RxCxD: Image of size RxC. D is 3 for color, 1 for
%      gray scale.
%   SI (scalar): coefficient used to compute similarity (weight) matrix
%   SX (scalar): coefficient used to ...
%   r  (scalar): coefficient used to ..., Definition of neighboorhood.
%   sNcut (scalar): The smallest Ncut value (threshold) to keep
%      partitioning recursively.  (Hint: smaller Ncut value means
%      better partitioning. )
%   sArea  (scalar): The smallest size of area (threshold) which is accepted
%      as a segment.
%
%   segI (cell of matrices): segmented images.
%
% Author : Naotoshi Seo, sonots@umd.edu
% Date   : Oct, 2006
function segI = NcutSegment(I, SI, SX, r, sNcut, sArea)
 [nRow, nCol, c] = size(I);
 N = nRow * nCol;
 V = reshape(I, N, c); % connect up-to-down way. Vertices of Graph

 % Step 1. Compute weight matrix W, and D
 W = computeW(I, SI, SX, r);

 % Step 5. recursively repartition
 seg = (1:N)'; % the first segment has whole nodes. [1 2 3 ... N]'
 [sub ids ncuts] = Segment(seg, W, sNcut, sArea, 'ROOT');

 % Convert node ids into images
 for i=1:length(sub)
     subV = zeros(N, c); %ones(N, c) * 255;
     subV(sub{i}, :) = V(sub{i}, :);
     segI{i} = uint8(reshape(subV, nRow, nCol, c));
     fprintf('%s. Ncut = %f\n', ids{i}, ncuts{i});
 end
```

computeW.m

```
% computeW: Compute a similarity (weight) matrix
%
%  W = computeW(I, SI, SX, r)
%
%  Input and output arguments ([]'s are optional):
%   I  (matrix) of size RxCxD: Image of size RxC. D is 3 for color, 1 for
%   gray scale.
%   SI (scalar): coefficient
%   SX (scalar): coefficient
%   r  (scalar): coefficient. Definition of neighboorhood.
%   W (matrix) of size (RxC)x(RxC). The computed similarity (weight)
%   matrix. w_{ij} is similarity between node i and j.
%   Hint: V = reshape(I, nRow * nCol, c) merges matrix into up-down way
%
% Author : Naotoshi Seo, sonots@umd.edu
% Date   : Oct, 2006
function W = computeW(I, SI, SX, r);
 [nRow, nCol, c] = size(I);
 N = nRow * nCol;
 W = sparse(N,N);

 % Feature Vectors
 if c == 3
     F = F3(I);
 else
     F = F2(I);
 end
 F = reshape(F, N, 1, c); % col vector
 % Spatial Location, e.g.,
 % [1 1] [1 2] [1 2]
 % [2 1] [2 2] [2 3]
 X = cat(3, repmat((1:nRow)', 1, nCol), repmat((1:nCol), nRow, 1));
 X = reshape(X, N, 1, 2); % col vector

 % Future Work: Reduce computation to half. It can be done
 % because W is symmetric mat
 for ic=1:nCol
     for ir=1:nRow
        % matlab tricks for fast computation (Avoid 'for' loops as much as
        % possible, instead use repmat.)

        % This range satisfies |X(i) - X(j)| <= r (block distance)
        jc = (ic - floor(r)) : (ic + floor(r)); % vector
        jr = ((ir - floor(r)) :(ir + floor(r)))';
        jc = jc(jc >= 1 & jc <= nCol);
        jr = jr(jr >= 1 & jr <= nRow);
        jN = length(jc) * length(jr);

        % index at vertex. V(i)
        i = ir + (ic - 1) * nRow;
```

```
        j = repmat(jr, 1, length(jc)) + repmat((jc -1) * nRow, length(jr), 1);
        j = reshape(j, length(jc) * length(jr), 1); % a col vector

        % spatial location distance (disimilarity)
        XJ = X(j, 1, :);
        XI = repmat(X(i, 1, :), length(j), 1);
        DX = XI - XJ;
        DX = sum(DX .* DX, 3); % squared euclid distance
        %DX = sum(abs(DX), 3); % block distance
        % square (block) reagion may work better for skew lines than circle
        %(euclid) reagion.

        % |X(i) - X(j)| <= r (already satisfied if block distance measurement)
        constraint = find(sqrt(DX) <= r);
        j = j(constraint);
        DX = DX(constraint);

        % feature vector disimilarity
        FJ = F(j, 1, :);
        FI = repmat(F(i, 1, :), length(j), 1);
        DF = FI - FJ;
        DF = sum(DF .* DF, 3); % squared euclid distance
        %DF = sum(abs(DF), 3); % block distance

        % Hint: W(i, j) is a col vector even if j is a matrix
        W(i, j) = exp(-DF / (SI*SI)) .* exp(-DX / (SX*SX)); % for squared distance
        %W(i, j) = exp(-DF / SI) .* exp(-DX / SX);
      end
  end
end


% F1 - F4: Compute a feature vector F. See 4 EXPERIMENTS
%
%  F = F1(I) % for point sets
%  F = F2(I) % intensity
%  F = F3(I) % hsv, for color
%  F = F4(I) % DOOG
%
%  Input and output arguments ([]'s are optional):
%   I (scalar or vector). The image.
%   F (scalar or vector). The computed feature vector F
%
% Author : Naotoshi Seo
% Date   : Oct, 2006
function F = F1(I);
 % for point sets
 F = (I == 0);
end
function F = F2(I);
 % intensity, for gray scale
```

```
 F = I;
end
function F = F3(I);
 % hsv, for color
 F = I; % raw RGB
 % Below hsv resulted in errors at eigs(). eigs returns erros so often.
%  F = rgb2hsv(double(I)); % V = [0, 255] with double, V = [0, 1] without double
%  % any fast way in matlab?
%  [nRow nCol c] = size(I);
%  for i=1:nRow
%      for j=1:nCol
%          HSV = reshape(F(i, j, :), 3, 1);
%          h = HSV(1); s = HSV(2); v = HSV(3);
%          F(i, j, :) = [v v*s*sin(h) v*s*cos(h)];
%      end
%  end
end
function F = F4(I);
 % DOOG, for texture
 % Future
end

% After all, I did not use this
% coord: Convert vertex index into spatial coordinates
%
%  X = coordinate(n, nRow, nCol)
%
%  Input and output arguments ([]'s are optional):
%   n (scalar or col vector). The vertex index of graph
%   nRow (scalar). The # of rows in original image
%   nCol (scalar). The # of cols in original image
%   X (2 cols vector). The spatial (image) coordinates whose 1st col
%   expresses row, and 2nd col expressed col.
%
% Hint: reshape() connects into column way (up-to-down)
%
% Author : Naotoshi Seo
% Date   : Oct, 2006
function X = coord(i, nRow, nCol);
 i = i - 1; % let me start from 0 to make mod easy
 row = mod(i, nRow);
 col = floor(i / nRow);
 % matlab index starts from 1
 row = row + 1;
 col = col + 1;
 X = [row col];
end
```

Segment.m

```
% Segment: segmentation. This function is called recursively (Step 5)
%
%  [sub ids ncuts] = Segment(seg, W, sNcut, sArea, id)
%
%  Input and output arguments ([]'s are optional):
%   seg (vector) of size N: A segment to be partitioned. Each element has a
%      node index of V (global segment).
%   W (matrix) of size NxN: A weight matrix of the segment
%   sNcut (scalar): The smallest Ncut value (threshold) to keep
%      partitioning recursively. If the Ncut value of a segment is larger than
%      this value, stop partitioning.
%   sArea  (scalar): The smallest size of area (threshold) which is accepted
%      as a segment.
%   id (string): label of the segment (for debuggin)
%
%   sub (cell of vectors): Segments which was partitioned. Each cell
%      expresses each segment.
%   [ids] (cell of string): A stack of labels of each segment (for debugging)
%   [ncuts] (cell of scalars): A stack of Ncut values of each segment
%
% Author : Naotoshi Seo, sonots@umd.edu
% Date   : Oct, 2006
function [sub ids ncuts] = Segment(seg, W, sNcut, sArea, id)
 % Compute D
 N = length(W);
 d = sum(W, 2);
 D = spdiags(d, 0, N, N); % diagonal matrix

 % Step 2 and 3. Solve generalized eigensystem (D -W)*S = S*D*U (12).
 % (13) is not necessary thanks to smart matlab.
 % Get the 2 smallests ('sm')
 warning off; % let me stop warning
 [U,S] = eigs(D-W, D, 2, 'sm');
 % 2nd smallest (1st smallest has all same value elements, and useless)
 U2 = U(:, 2);
 % T = abs(U2); m = max(T); T = T / m * 255; imshow(uint8(reshape(T, 15, 20)));

 % Step 3. Refer 3.1 Example 3.
 % (1). Bipartition the graph at 0 (hopefully, eigenvector can be
 % splitted by + and -). % This did not work well.
 %A = find(U2 > 0);
 %B = find(U2 <= 0);
 % (2). Bipartition the graph at median value.
 %t = median(U2);
 %A = find(U2 > t);
 %B = find(U2 <= t);
 % (3). Bipartition the graph at point that Ncut is minimized.
 t = mean(U2);
 t = fminsearch('Ncut', t, [], U2, W, D);
 A = find(U2 > t);
```

```
 B = find(U2 <= t);

 % Step 4. Decide if the current partition should be divided
 % if either of partition is too small, stop recursion.
 % if Ncut is larger than threshold, stop recursion.
 ncut = Ncut(t, U2, W, D);
 if (length(A) < sArea || length(B) < sArea) || ncut > sNcut
    sub{1}   = seg;
    ids{1}   = id; % for debugging
    ncuts{1} = ncut; % for duebuggin
    return;
 end

 % sub segments of A
 [sub ids ncuts] = Segment(seg(A), W(A, A), sNcut, sArea, [id '-A']);
 % seg(A): node index at V. A is index at the segment, seg
 % W(A, A); % weight matrix in segment A

 % sub segments of B
 [subB idsB ncutsB] = Segment(seg(B), W(B, B), sNcut, sArea, [id '-B']);

 % concatenate cell arrays
 sub   = [sub subB];
 ids   = [ids idsB];
 ncuts = [ncuts ncutsB];
end
```

Ncut.m

```
% Ncut: 2.1 Computing the Optimal Partition Ncut. eq (5)
%
%  ncut = Ncut(T, U2, D, W);
%
%  Input and output arguments ([]'s are optional):
%   t (scalar): splitting point (threshold)
%   U2 (vector) of size P: The 2nd smallest eigenvector computed at step 2.
%   W (matrix) of size PxP: The weight matrix
%   D (diagonal matrix) of size PxP:
%   ncut (scalar): The value calculated at the right term of eq (5). This can
%   be used to find minimum Ncut.
%
% Author : Naotoshi Seo, sonots@umd.edu
% Date   : Oct, 2006
function ncut = Ncut(t, U2, W, D);
    x = (U2 > t);
    x = (2 * x) - 1; % convert [1 0 0 1 0]' to [1 -1 -1 1 -1]' to follow paper's way
    d = diag(D);
    k = sum(d(x > 0)) / sum(d);
    b = k / (1 - k);
    y = (1 + x) - b * (1 - x);
    ncut = (y' * (D - W) * y) / ( y' * D * y );
end
```

demo.m

```
function demo
 SI =5; SX =6; r = 1.5; sNcut = 0.14; sArea = 220;
 I = imread('s42049.jpg');
 segI = NcutSegment(I, SI, SX, r, sNcut, sArea);
 % show
 for i=1:length(segI)
     figure; imshow(segI{i});
     imwrite(segI{i}, sprintf('s42049-%d.png', i));
 end
end
```