Lecture 13: Advanced Image Segmentation

CAP5415-Computer Vision
Lecture 13-Advanced Image Segmentation
(Optimization Problem)

Dr. Ulas Bagci

bagci@ucf.edu
Reminders

• **Oct 18** Guest Lecture (Lecture 17) Dr. Borji
  – Basics of Human Vision System

• **Oct 20** Guest Lecture (Lecture 18) Dr. Gong
  – Deep Learning (CNN, RNN, LSTM, GRU)

• Select your project by 15\textsuperscript{th} of October (Use WebCourse!)
Labeling & Segmentation

• Labeling is a common way for modeling various computer vision problems (e.g. optical flow, image segmentation, stereo matching, etc)
Labeling & Segmentation

- **Labeling** is a common way for modeling various computer vision problems (e.g. optical flow, image segmentation, stereo matching, etc).
- The set of labels can be discrete (as in image segmentation)

\[ L = \{l_1, \ldots, l_m\} \text{ with } L = m \]
Labeling & Segmentation

- **Labeling** is a common way for modeling various computer vision problems (e.g. optical flow, image segmentation, stereo matching, etc)

- The set of labels can be discrete (as in image segmentation)

  \[ L = \{ l_1, \ldots, l_m \} \text{ with } L = m \]

- Or continuous (as in optical flow)

  \[ L \subset \mathbb{R}^n \text{ for } n \geq 1 \]
Labeling is a function

- Labels are assigned to *sites* (pixel locations)
Labeling is a function

- Labels are assigned to *sites* (pixel locations)
- For a given image, we have $|\Omega| = N_{cols} \cdot N_{rows}$
Labeling is a function

- Labels are assigned to *sites* (pixel locations)
- For a given image, we have $|\Omega| = N_{cols} \cdot N_{rows}$
- Identifying a labeling function (with segmentation) is $f : \Omega \rightarrow L$
Labeling is a function

• Labels are assigned to sites (pixel locations)
• For a given image, we have \(|\Omega| = N_{cols} \cdot N_{rows}\)
• Identifying a labeling function (with segmentation) is \(f : \Omega \rightarrow L\)

We aim at calculating a labeling function that minimizes a given (total) error or energy
Labeling is a function

• Labels are assigned to sites (pixel locations)
• For a given image, we have $|\Omega| = N_{cols} \cdot N_{rows}$
• Identifying a labeling function (with segmentation) is $f : \Omega \rightarrow L$

We aim at calculating a labeling function that minimizes a given (total) error or energy

$$E(f) = \sum_{p \in \Omega} [E_{data}(p, f_p) + \sum_{q \in A(p)} E_{smooth}(f_p, f_q)]$$

*A is an adjacency relation between pixel locations*
Energy Function

The role of an energy function in minimization-based vision is twofold:

1. as the quantitative measure of the global quality of the solution and
2. as a guide to the search for a minimal solution.

Correct solution is embedded as the minimum.
Segmentation as an Energy Minimization Problem

• $E_{data}$ assigns non-negative penalties to a pixel location $p$ when assigning a label to this location.
Segmentation as an Energy Minimization Problem

- $E_{data}$ assigns non-negative penalties to a pixel location $p$ when assigning a label to this location.
- $E_{smooth}$ assigns non-negative penalties by comparing the assigned labels $f_p$ and $f_q$ at adjacent positions $p$ and $q$. 
Segmentation as an Energy Minimization Problem

- $E_{data}$ assigns non-negative penalties to a pixel location $p$ when assigning a label to this location.
- $E_{smooth}$ assigns non-negative penalties by comparing the assigned labels $f_p$ and $f_q$ at adjacent positions $p$ and $q$. 
Segmentation as an Energy Minimization Problem

- \( E_{\text{data}} \) assigns non-negative penalties to a pixel location \( p \) when assigning a label to this location.
- \( E_{\text{smooth}} \) assigns non-negative penalties by comparing the assigned labels \( f_p \) and \( f_q \) at adjacent positions \( p \) and \( q \).

This optimization model is characterized by local interactions along edges between adjacent pixels, and often called \textbf{MRF (Markov Random Field)} model.
Energy Function-Details

\[ E(f) = \sum_{p \in \Omega} [E_{data}(p, f_p) + \sum_{q \in A(p)} E_{smooth}(f_p, f_q)] \]
Energy Function-Details

\[ E(f) = \sum_{p \in \Omega} \left[ E_{data}(p, f_p) + \sum_{q \in A(p)} E_{smooth}(f_p, f_q) \right] \]

- Sample Data Term: 
  \[ E_{data}(p, f_p) = \Psi(p) \]
  \[ \Psi(p = 0) = -\log P(p \in BG) \]
  \[ \Psi(p = 1) = -\log P(p \in FG) \]
Energy Function-Details

\[ E(f) = \sum_{p \in \Omega} \left[ E_{data}(p, f_p) + \sum_{q \in A(p)} E_{smooth}(f_p, f_q) \right] \]

- Sample Data Term: \( E_{data}(p, f_p) = \Psi(p) \)
  \[ \Psi(p = 0) = -\log P(p \in BG) \]
  \[ \Psi(p = 1) = -\log P(p \in FG) \]

- Sample Smoothness Term:
  \[ \Psi(p, q) = K_{pq} \delta(p \neq q) \text{ where} \]
  \[ K_{pq} = \frac{\exp(-\beta(I_p - I_q)^2/(2\sigma^2))}{||p, q||} \]
Energy Function-Details

\[ E(p) = \sum_{p \in \Omega} \Psi_p(p) + \sum_{p \in \Omega} \sum_{q \in A(p)} \Psi_{pq}(p, q) \]

\[ p^* = \arg\min_{p \in L} E(p) \]
Energy Function-Details

\[ E(p) = \sum_{p \in \Omega} \Psi_p(p) + \sum_{p \in \Omega} \sum_{q \in A(p)} \Psi_{pq}(p, q) \]

\[ p^* = \arg\min_{p \in L} E(p) \]

- To solve this problem, transform the energy functional into min-cut/max-flow problem and solve it!
Energy Function

Unary Cost (data term)

Discontinuity Cost

p
q
Recap: Image as a Graph
Graph Cuts for Optimal Boundary Detection (Boykov ICCV 2001)

• Binary label: foreground vs. background
• User labels some pixels
• Exploit
  – Statistics of known Fg & Bg
  – Smoothness of label
• Turn into discrete graph optimization
  – Graph cut (min cut / max flow)
Each pixel is connected to its neighbors in an undirected graph.

**Goal:** split nodes into two sets $A$ and $B$ based on pixel values, and try to classify neighbors in the same way too!
Each pixel is connected to its neighbors in an undirected graph

**Goal:** split nodes into two sets A and B based on pixel values, and try to classify Neighbors in the same way too!
Graph-Cut

- Each pixel = node
- Add two nodes F & B
- Labeling: link each pixel to either F or B

Desired result
Cost Function: Data term

- Put one edge between each pixel and both F & G
- Weight of edge = minus data term
Cost Function: Smoothness term

- Add an edge between each neighbor pair
- Weight = smoothness term
Min-Cut

• Energy optimization equivalent to graph min cut
• Cut: remove edges to disconnect F from B
• Minimum: minimize sum of cut edge weight
Min-Cut

Graph \((V, E, C)\)

Vertices \(V = \{v_1, v_2 \ldots v_n\}\)

Edges \(E = \{(v_1, v_2) \ldots\}\)

Costs \(C = \{c_{(1, 2)} \ldots\}\)
What is a st-cut?
An st-cut \((S,T)\) divides the nodes between source and sink.

What is the cost of a st-cut?
Sum of cost of all edges going from \(S\) to \(T\)

\[
5 + 2 + 9 = 16
\]
Min-Cut

What is a st-cut?
An st-cut \((S,T)\) divides the nodes between source and sink.

What is the cost of a st-cut?
Sum of cost of all edges going from S to T

What is the st-mincut?
st-cut with the minimum cost

\[
2 + 1 + 4 = 7
\]
How to compute min-cut?

Solve the dual maximum flow problem

Compute the maximum flow between Source and Sink

Constraints
Edges: Flow < Capacity
Nodes: Flow in & Flow out

Min-cut/Max-flow Theorem
In every network, the maximum flow equals the cost of the st-mincut
Max-Flow Algorithms

Augmenting Path Based Algorithms

1. Find path from source to sink with positive capacity

2. Push maximum possible flow through this path

3. Repeat until no path can be found

Algorithms assume non-negative capacity
Max-Flow Algorithms

Augmenting Path Based Algorithms

1. Find path from source to sink with positive capacity

2. Push maximum possible flow through this path

3. Repeat until no path can be found

Source

v₁

v₂

Sink

Flow = 0

Algorithms assume non-negative capacity
Max-Flow Algorithms

Augmenting Path Based Algorithms

1. Find path from source to sink with positive capacity

2. Push maximum possible flow through this path

3. Repeat until no path can be found

Algorithms assume non-negative capacity
Max-Flow Algorithms

Augmenting Path Based Algorithms

1. Find path from source to sink with positive capacity
2. Push maximum possible flow through this path
3. Repeat until no path can be found

Algorithms assume non-negative capacity
Max-Flow Algorithms

Flow = 2

Augmenting Path Based Algorithms

1. Find path from source to sink with positive capacity
2. Push maximum possible flow through this path
3. Repeat until no path can be found

Algorithms assume non-negative capacity
Max-Flow Algorithms

Augmenting Path Based Algorithms

1. Find path from source to sink with positive capacity
2. Push maximum possible flow through this path
3. Repeat until no path can be found

Algorithms assume non-negative capacity
Max-Flow Algorithms

Augmenting Path Based Algorithms

1. Find path from source to sink with positive capacity
2. Push maximum possible flow through this path
3. Repeat until no path can be found

Algorithms assume non-negative capacity

Flow = 2 + 4
Max-Flow Algorithms

1. Find path from source to sink with positive capacity
2. Push maximum possible flow through this path
3. Repeat until no path can be found

Algorithms assume non-negative capacity
**Max-Flow Algorithms**

Augmenting Path Based Algorithms

1. Find path from source to sink with positive capacity
2. Push maximum possible flow through this path
3. Repeat until no path can be found

Algorithms assume non-negative capacity
Max-Flow Algorithms

Augmenting Path Based Algorithms

1. Find path from source to sink with positive capacity
2. Push maximum possible flow through this path
3. Repeat until no path can be found

Algorithms assume non-negative capacity
Max-Flow Algorithms

Flow = 7

Augmenting Path Based Algorithms

1. Find path from source to sink with positive capacity
2. Push maximum possible flow through this path
3. Repeat until no path can be found

Algorithms assume non-negative capacity
Max-Flow Algorithms

Augmenting Path Based Algorithms

1. Find path from source to sink with positive capacity
2. Push maximum possible flow through this path
3. Repeat until no path can be found

Algorithms assume non-negative capacity
Ford & Fulkerson algorithm (1956)

Find the path from source to sink
While (path exists)
  flow += maximum capacity in the path
Build the residual graph ("subtract" the flow)
  Find the path in the residual graph
End
Ford & Fulkerson algorithm (1956)

Find the path from source to sink

While (path exists)
  flow += maximum capacity in the path

Build the residual graph ("subtract" the flow)

Find the path in the residual graph

End
Ford & Fulkerson algorithm (1956)

Find the path from source to sink

While (path exists)
  flow += maximum capacity in the path

Build the residual graph ( “subtract” the flow)

Find the path in the residual graph

End

flow = 3
Ford & Fulkerson algorithm (1956)

Find the path from source to sink
While (path exists)
  flow += maximum capacity in the path
  Build the residual graph ("subtract" the flow)
  Find the path in the residual graph
End

flow = 3
Another Example - Max Flow

Ford & Fulkerson algorithm (1956)

Find the path from source to sink
While (path exists)
   flow += maximum capacity in the path
Build the residual graph ("subtract" the flow)
Find the path in the residual graph
End

flow = 3
Another Example-Max Flow

Ford & Fulkerson algorithm (1956)

Find the path from source to sink
While (path exists)
    flow += maximum capacity in the path
Build the residual graph ( “subtract” the flow)
Find the path in the residual graph
End

flow = 3
Ford & Fulkerson algorithm (1956)

Find the path from source to sink
While (path exists)
  flow += maximum capacity in the path
Build the residual graph ("subtract" the flow)
Find the path in the residual graph
End

flow = 6
Another Example - Max Flow

Ford & Fulkerson algorithm (1956)

Find the path from source to sink
While (path exists)
  flow += maximum capacity in the path
Build the residual graph ("subtract" the flow)
  Find the path in the residual graph
End

flow = 6
Another Example-Max Flow

Ford & Fulkerson algorithm (1956)

Find the path from source to sink
While (path exists)
  flow += maximum capacity in the path
  Build the residual graph ("subtract" the flow)
  Find the path in the residual graph
End

flow = 6
Another Example - Max Flow

Ford & Fulkerson algorithm (1956)

Find the path from source to sink
While (path exists)
  flow += maximum capacity in the path
Build the residual graph ( "subtract" the flow)
Find the path in the residual graph
End

flow = 6
Another Example - Max Flow

Ford & Fulkerson algorithm (1956)

Find the path from source to sink
While (path exists)
  flow += maximum capacity in the path
Build the residual graph ("subtract" the flow)
Find the path in the residual graph
End

flow = 11
Another Example - Max Flow

Ford & Fulkerson algorithm (1956)

Find the path from source to sink
While (path exists)
    flow += maximum capacity in the path
    Build the residual graph ("subtract" the flow)
    Find the path in the residual graph
End

flow = 11
Ford & Fulkerson algorithm (1956)

Find the path from source to sink
While (path exists)
    flow += maximum capacity in the path
    Build the residual graph ( "subtract" the flow)
    Find the path in the residual graph
End

flow = 11
Another Example - Max Flow

Ford & Fulkerson algorithm (1956)

Find the path from source to sink
While (path exists)
    flow += maximum capacity in the path
    Build the residual graph ( "subtract" the flow)
    Find the path in the residual graph
End

flow = 11
Another Example-Max Flow

Ford & Fulkerson algorithm (1956)

Find the path from source to sink
While (path exists)
  flow += maximum capacity in the path
  Build the residual graph ("subtract" the flow)
  Find the path in the residual graph
End

flow = 13
Another Example - Max Flow

Ford & Fulkerson algorithm (1956)

Find the path from source to sink
While (path exists)
  flow += maximum capacity in the path
Build the residual graph ( “subtract” the flow)
Find the path in the residual graph
End

flow = 13
Ford & Fulkerson algorithm (1956)

Find the path from source to sink
While (path exists)
    flow += maximum capacity in the path
Build the residual graph ("subtract" the flow)
Find the path in the residual graph
End

flow = 13
Another Example-Max Flow

Ford & Fulkerson algorithm (1956)

Find the path from source to sink
While (path exists)
    flow += maximum capacity in the path
    Build the residual graph ("subtract" the flow)
Find the path in the residual graph
End

flow = 13
Another Example - Max Flow

Ford & Fulkerson algorithm (1956)

- Find the path from source to sink
- While (path exists)
  - flow += maximum capacity in the path
  - Build the residual graph ("subtract" the flow)
- Find the path in the residual graph
- End

flow = 15
Another Example-Max Flow

Ford & Fulkerson algorithm (1956)

Find the path from source to sink
While (path exists)
  flow += maximum capacity in the path
Build the residual graph ( "subtract" the flow)
Find the path in the residual graph
End

flow = 15
Ford & Fulkerson algorithm (1956)

Find the path from source to sink
While (path exists)
    flow += maximum capacity in the path
Build the residual graph ( “subtract” the flow)
Find the path in the residual graph
End

flow = 15
Another Example-Max Flow

Ford & Fulkerson algorithm (1956)

Find the path from source to sink
While (path exists)
  flow += maximum capacity in the path
  Build the residual graph ( “subtract” the flow)
  Find the path in the residual graph
End

flow = 15
Ford & Fulkerson algorithm (1956)

Find the path from source to sink

While (path exists)

    flow += maximum capacity in the path

Build the residual graph ( “subtract” the flow)

Find the path in the residual graph

End

flow = 15
Another Example - Max Flow

Ford & Fulkerson algorithm (1956)

Why is the solution globally optimal?

flow = 15
Another Example-Max Flow

Ford & Fulkerson algorithm (1956)

Why is the solution globally optimal?

1. Let $S$ be the set of reachable nodes in the residual graph

flow = 15
Ford & Fulkerson algorithm (1956)

Why is the solution globally optimal?

1. Let $S$ be the set of reachable nodes in the residual graph.
2. The flow from $S$ to $V - S$ equals to the sum of capacities from $S$ to $V - S$.

Another Example-Max Flow
Ford & Fulkerson algorithm (1956)

Why is the solution globally optimal?

1. Let $S$ be the set of reachable nodes in the residual graph
2. The flow from $S$ to $V - S$ equals to the sum of capacities from $S$ to $V - S$
3. The flow from any $A$ to $V - A$ is upper bounded by the sum of capacities from $A$ to $V - A$
4. The solution is globally optimal
Another Example - Max Flow

S

source

sink

cost = 18

S

source

sink

T

T
Another Example-Max Flow

source

S

cost = 23

cost = 23

sink

T

Lecture 13: Advanced Image Segmentation
Another Example—Max Flow

\[ C(x) = 5x_1 + 9x_2 + 4x_3 + 3x_3(1-x_1) + 2x_1(1-x_3) + 3x_3(1-x_1) + 2x_2(1-x_3) + 5x_3(1-x_2) + 2x_4(1-x_1) + 1x_5(1-x_1) + 6x_5(1-x_3) + 5x_6(1-x_3) + 1x_3(1-x_6) + 3x_6(1-x_2) + 2x_4(1-x_5) + 3x_6(1-x_5) + 6(1-x_4) + 8(1-x_5) + 5(1-x_6) \]
Another Example - Max Flow

\[ C(x) = 2x_1 + 9x_2 + 4x_3 + 2x_1(1-x_3) \]
\[ + 3x_3(1-x_1) + 2x_2(1-x_3) + 5x_3(1-x_2) + 2x_4(1-x_1) \]
\[ + 1x_5(1-x_1) + 3x_5(1-x_3) + 5x_6(1-x_3) + 1x_3(1-x_6) \]
\[ + 3x_6(1-x_2) + 2x_4(1-x_5) + 3x_6(1-x_5) + 6(1-x_4) \]
\[ + 5(1-x_5) + 5(1-x_6) \]
\[ + 3x_1 + 3x_3(1-x_1) + 3x_5(1-x_3) + 3(1-x_5) \]
C(x) = 2x_1 + 9x_2 + 4x_3 + 2x_1(1-x_3) 
+ 3x_3(1-x_1) + 2x_2(1-x_3) + 5x_3(1-x_2) + 2x_4(1-x_1) 
+ 1x_5(1-x_1) + 3x_5(1-x_3) + 5x_6(1-x_3) + 1x_3(1-x_6) 
+ 3x_6(1-x_2) + 2x_4(1-x_5) + 3x_6(1-x_5) + 6(1-x_4) 
+ 5(1-x_5) + 5(1-x_6) 
+ 3x_1 + 3x_3(1-x_1) + 3x_5(1-x_3) + 3(1-x_5) 

\[ C(x) = 3 + 3x_1(1-x_3) + 3x_3(1-x_5) \]
Another Example - Max Flow

\[ C(x) = 3 + 2x_1 + 6x_2 + 4x_3 + 5x_1(1-x_3) + 3x_3(1-x_1) + 2x_2(1-x_3) + 5x_3(1-x_2) + 2x_4(1-x_1) + 1x_5(1-x_1) + 3x_5(1-x_3) + 5x_6(1-x_3) + 1x_3(1-x_6) + 2x_5(1-x_4) + 6(1-x_4) + 2(1-x_5) + 5(1-x_6) + 3x_3(1-x_5) + 3x_2 + 3x_6(1-x_2) + 3x_5(1-x_6) + 3(1-x_5) \]

\[
3x_2 + 3x_6(1-x_2) + 3x_5(1-x_6) + 3(1-x_5) = 3 + 3x_2(1-x_6) + 3x_6(1-x_5)
\]
Another Example - Max Flow

\[ C(x) = 6 + 2x_1 + 6x_2 + 4x_3 + 5x_1(1-x_3) + 3x_3(1-x_1) + 2x_2(1-x_3) + 5x_3(1-x_2) + 2x_4(1-x_1) + 1x_5(1-x_1) + 3x_5(1-x_3) + 5x_6(1-x_3) + 1x_3(1-x_6) + 2x_5(1-x_4) + 6(1-x_4) + 3x_2(1-x_6) + 3x_6(1-x_5) + 2(1-x_5) + 5(1-x_6) + 3x_3(1-x_5) \]
Another Example - Max Flow

\[ C(x) = 15 + 1x_2 + 4x_3 + 5x_1(1-x_3) \]
\[ + 3x_3(1-x_1) + 7x_2(1-x_3) + 2x_1(1-x_4) \]
\[ + 1x_5(1-x_1) + 6x_3(1-x_6) + 6x_3(1-x_5) \]
\[ + 2x_5(1-x_4) + 4(1-x_4) + 3x_2(1-x_6) + 3x_6(1-x_5) \]

- All coefficients positive
- Must be global minimum

\[ S \text{ – set of reachable nodes from } s \]
# History of Max-Flow Algorithms

## Augmenting Path and Push-Relabel

<table>
<thead>
<tr>
<th>year</th>
<th>discoverer(s)</th>
<th>bound</th>
</tr>
</thead>
<tbody>
<tr>
<td>1951</td>
<td>Dantzig</td>
<td>$O(n^2 m U)$</td>
</tr>
<tr>
<td>1955</td>
<td>Ford &amp; Fulkerson</td>
<td>$O(m^2 U)$</td>
</tr>
<tr>
<td>1970</td>
<td>Dinitz</td>
<td>$O(n^2 m)$</td>
</tr>
<tr>
<td>1972</td>
<td>Edmonds &amp; Karp</td>
<td>$O(m^2 \log U)$</td>
</tr>
<tr>
<td>1973</td>
<td>Dinitz</td>
<td>$O(nm \log U)$</td>
</tr>
<tr>
<td>1974</td>
<td>Karzanov</td>
<td>$O(n^3)$</td>
</tr>
<tr>
<td>1977</td>
<td>Cherkassky</td>
<td>$O(n^2 m^{1/2})$</td>
</tr>
<tr>
<td>1980</td>
<td>Galil &amp; Naamad</td>
<td>$O(nm \log^2 n)$</td>
</tr>
<tr>
<td>1983</td>
<td>Sleator &amp; Tarjan</td>
<td>$O(nm \log n)$</td>
</tr>
<tr>
<td>1986</td>
<td>Goldberg &amp; Tarjan</td>
<td>$O(n m \log(n^2/m))$</td>
</tr>
<tr>
<td>1987</td>
<td>Ahuja &amp; Orlin</td>
<td>$O(n m + n^2 \log U)$</td>
</tr>
<tr>
<td>1987</td>
<td>Ahuja et al.</td>
<td>$O(n m \log(n \sqrt{\log U/m}))$</td>
</tr>
<tr>
<td>1989</td>
<td>Cheriyan &amp; Hagerup</td>
<td>$E(nm + n^2 \log^2 n)$</td>
</tr>
<tr>
<td>1990</td>
<td>Cheriyan et al.</td>
<td>$O(n^3 / \log n)$</td>
</tr>
<tr>
<td>1990</td>
<td>Alon</td>
<td>$O(nm + n^{8/3} \log n)$</td>
</tr>
<tr>
<td>1992</td>
<td>King et al.</td>
<td>$O(nm + n^{2+\epsilon})$</td>
</tr>
<tr>
<td>1993</td>
<td>Phillips &amp; Westbrook</td>
<td>$O(nm(\log_{m/n} n + \log^{2+\epsilon} n))$</td>
</tr>
<tr>
<td>1994</td>
<td>King et al.</td>
<td>$O(nm \log_{m/(n \log n)} n)$</td>
</tr>
<tr>
<td>1997</td>
<td>Goldberg &amp; Rao</td>
<td>$O(m^{3/2} \log(n^2/m) \log U)$</td>
</tr>
<tr>
<td></td>
<td></td>
<td>$O(n^{2/3} m \log(n^2/m) \log U)$</td>
</tr>
</tbody>
</table>

$n$: #nodes  
$m$: #edges  
$U$: maximum edge weight

Algorithms assume non-negative edge weights
Example Segmentation in a Video

$E(x) \rightarrow \text{Flow} \rightarrow \text{Global Optimum}$
Segmentation Example
Segmentation Example
Segmentation Example
Lecture 13: Advanced Image Segmentation

Optimality?

Seeds

GrabCut (iterative GC With box prior)
Vascular Surface Segmentation via GC
Slide Credits and References

- Fredo Durand of MIT
- M. Tappen of Amazon
- R. Szelisky, Univ. of Washington/Seattle
- [http://www.csd.uwo.ca/faculty/yuri/Abstracts/eccv06-tutorial.html](http://www.csd.uwo.ca/faculty/yuri/Abstracts/eccv06-tutorial.html)
- J. Malcolm, Graph Cut in Tensor Scale
- [http://www.cse.yorku.ca/~aaw/Wang/MaxFlowStart.htm](http://www.cse.yorku.ca/~aaw/Wang/MaxFlowStart.htm)
- [http://research.microsoft.com/vision/cambridge/i3l/segmentation/GrabCut.htm](http://research.microsoft.com/vision/cambridge/i3l/segmentation/GrabCut.htm)
- [http://www.cc.gatech.edu/cpl/projects/graphcuttextures/](http://www.cc.gatech.edu/cpl/projects/graphcuttextures/)
- P. Kumar, Oxford University.