LECTURE 10: Medical Image Segmentation as an Energy Minimization Problem

Dr. Ulas Bagci
HEC 221, Center for Research in Computer Vision (CRCV), University of Central Florida (UCF), Orlando, FL 32814.

bagci@ucf.edu or bagci@crcv.ucf.edu
Outline

• Energy functional
  – Data and Smoothness terms

• Graph Cut
  – Min cut
  – Max Flow

• Applications in Radiology Images
Manual annotation through expert raters. Shown are image patches with the tumor structures that are annotated in the different modalities (top left) and the final labels for the whole dataset (right). Image patches show from left to right: the whole tumor visible in FLAIR (A), the tumor core visible in T2 (B), the enhancing tumor structures visible in T1c (blue), surrounding the cystic/necrotic components of the core (green) (C). Segmentations are combined to generate the final labels of the tumor structures (D): edema (yellow), non-enhancing solid core (red), necrotic/cystic core (green), enhancing core (blue). Credit: BRATS paper/TMI
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)
- Or continuous (tracking, etc.)

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

- Or continuous (tracking, etc.)

\[ L \subset \mathbb{R}^n \text{ for } n \geq 1 \]
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
  \( f : \Omega \to L \)
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*
Example of Energy Minimization Methods

<table>
<thead>
<tr>
<th>Method</th>
<th>Continuous</th>
<th>Discrete</th>
<th>Global</th>
<th>Local</th>
</tr>
</thead>
<tbody>
<tr>
<td>Newton-Raphson</td>
<td>Yes</td>
<td></td>
<td>Possible</td>
<td>Yes</td>
</tr>
<tr>
<td>Gradient descent</td>
<td>Yes</td>
<td></td>
<td>Possible</td>
<td>Yes</td>
</tr>
<tr>
<td>Conjugate gradient</td>
<td>Yes</td>
<td></td>
<td></td>
<td>Yes</td>
</tr>
<tr>
<td>Proximal gradient</td>
<td>Yes</td>
<td></td>
<td></td>
<td>Yes</td>
</tr>
<tr>
<td>Coordinate descent</td>
<td>Yes</td>
<td></td>
<td></td>
<td>Yes</td>
</tr>
<tr>
<td>Genetic algorithm</td>
<td>Yes</td>
<td></td>
<td></td>
<td>Yes</td>
</tr>
<tr>
<td>Graph cuts</td>
<td></td>
<td>Yes</td>
<td></td>
<td>Yes</td>
</tr>
<tr>
<td>Belief propagation</td>
<td></td>
<td></td>
<td></td>
<td>Yes</td>
</tr>
<tr>
<td>Tree-reweighted message passing</td>
<td></td>
<td>Yes</td>
<td></td>
<td>Yes</td>
</tr>
<tr>
<td>Linear programming</td>
<td>Yes</td>
<td></td>
<td></td>
<td>Yes</td>
</tr>
<tr>
<td>Maximum margin learning</td>
<td>Yes</td>
<td></td>
<td></td>
<td>Yes</td>
</tr>
<tr>
<td>Simulated annealing</td>
<td>Yes</td>
<td></td>
<td></td>
<td>Yes</td>
</tr>
<tr>
<td>Iterated conditional modes</td>
<td>Yes</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Minimal surface</td>
<td>Yes</td>
<td>Yes</td>
<td></td>
<td>Yes</td>
</tr>
</tbody>
</table>

*Note: Continuous, Discrete, Global, and Local: types of optimization.*

Credit: Zhao et al., IJNMBE 15
Energy Function?

• **Penalizing results** which are not compatible with the observed images/volumes
Energy Function?

- **Penalizing results** which are not compatible with the observed images/volumes

Unary (data) cost (inverted)
Energy Function?

- **Penalizing results** which are not compatible with the observed images/volumes
- **Enforcing** spatial coherence.

Pairwise (boundary) cost (inverted)
Energy Function?

• **Penalizing results** which are not compatible with the observed images/volumes
• **Enforcing** spatial coherence.

Pairwise (boundary) cost (inverted)
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 $q$. 

![Image as a graph representation](Image as a graph representation)
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 \( q \).

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

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

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

Example 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

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

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

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

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

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

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

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

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

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

To solve this problem, transform the energy functional into min-cut/max-flow problem and solve it!
Graph Cuts for Optimal Boundary Detection
(Boykov ICCV 2001)

• Binary label: foreground vs. background
• User labels some pixels
• Exploit
  – Statistics of known $F_g$ & $B_g$
  – 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

A 4x4 Image

Dissimilarity Graph of Pixels
Min-Cut

Graph (V, E, C)
- Vertices V = \{v_1, v_2, ..., v_n\}
- Edges E = \{(v_1, v_2), ..., \}
- Costs C = \{c_{(1, 2)}, ..., \}

Vertices:
- Source
- Sink

Edges:
- \(v_1 \rightarrow v_2\): 2
- \(v_1 \rightarrow \text{Sink}\): 5
- \(v_2 \rightarrow \text{Source}\): 9
- \(v_2 \rightarrow \text{Sink}\): 4
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

\[5 + 2 + 9 = 16\]
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

---

Min-cut\Max-flow Theorem

In every network, the maximum flow equals the cost of the st-mincut

---

**Constraints**

Edges: Flow < Capacity

Nodes: Flow in & Flow out
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 = 0 + 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
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

Flow = 2 + 4

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

Flow = 6

Source

\( v_1 \) \( v_2 \)

Sink

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 = 6
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

Flow = 6 + 1

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 = 7
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
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
Another Example - Max Flow

Ford & Fulkerson algorithm (1956)

Find the path from source to sink
While (path exists)

\[
\text{flow} += \text{maximum capacity in the path}
\]

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
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 = 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
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
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
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
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
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
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)

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

$\text{flow} = 15$
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
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.

Individual flows obtained by summing up all paths.

flow = 15
Another Example - Max Flow

Source \( S \) to Sink \( T \) with cost = 18
Another Example - Max Flow

Source $S$

Sink $T$

cost = 23
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

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

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

\[ 3x_1 + 3x_3(1-x_1) + 3x_5(1-x_3) + 3(1-x_5) \]
\[ = \]
\[ 3 + 3x_1(1-x_3) + 3x_3(1-x_5) \]
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 – 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>$O(n m \log U)$</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>$O(n m \log^2 n)$</td>
</tr>
<tr>
<td>1983</td>
<td>Sleator &amp; Tarjan</td>
<td>$O(n m \log n)$</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>$E(n m + n^2 \log^2 n)$</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>$O(n m + n^{8/3} \log n)$</td>
</tr>
<tr>
<td>1992</td>
<td>King et al.</td>
<td>$O(n m + n^2 + \epsilon)$</td>
</tr>
<tr>
<td>1993</td>
<td>Phillips &amp; Westbrook</td>
<td>$O(n m (\log_{m/n} n + \log^{2+\epsilon} n))$</td>
</tr>
<tr>
<td>1994</td>
<td>King et al.</td>
<td>$O(n m \log_{m/(n \log n)} n)$</td>
</tr>
<tr>
<td>1997</td>
<td>Goldberg &amp; Rao</td>
<td>$O(m^{3/2} \log(n^2/m) \log U)$</td>
</tr>
</tbody>
</table>

**Algorithms assume non-negative edge weights**

$n$: #nodes  
$m$: #edges  
$U$: maximum edge weight
Software Packages for Optimization

<table>
<thead>
<tr>
<th>Software</th>
<th>Brief Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ADMB</td>
<td>A powerful software package for the development of nonlinear optimization frameworks</td>
</tr>
<tr>
<td>CVXOPT</td>
<td>A free software package written in Python for convex optimization</td>
</tr>
<tr>
<td>NLopt</td>
<td>A free/open-source library for nonlinear optimization</td>
</tr>
<tr>
<td>optimx</td>
<td>A free package supporting the optimization of smooth, nonlinear functions with at most box constraints</td>
</tr>
<tr>
<td>pyOpt</td>
<td>A Python-based free package for formulating and solving nonlinear constrained optimization problems</td>
</tr>
<tr>
<td>LINGO</td>
<td>A comprehensive tool for solving linear, nonlinear, integer, and stochastic optimizations</td>
</tr>
<tr>
<td>Maple</td>
<td>A collection of commands for numerically solving optimization problems (e.g., linear, nonlinear, quadratic, continuous, integer, constrained, unconstrained)</td>
</tr>
<tr>
<td>MATLAB</td>
<td>An optimization toolbox consisting of functions for finding parameters that minimize or maximize objectives while satisfying constraints (e.g., linear, nonlinear, quadratic, multiple maxima, multiple minima, non-smooth)</td>
</tr>
<tr>
<td>Mathematica</td>
<td>An optimization package for solving large-scale multivariate constrained and unconstrained, linear and nonlinear, continuous and integer optimizations</td>
</tr>
</tbody>
</table>
Applications Used in Energy Minimization Based Segmentation Methods

<table>
<thead>
<tr>
<th>Medical Image Data Set</th>
</tr>
</thead>
<tbody>
<tr>
<td>2D MRI brain corpus callosum</td>
</tr>
<tr>
<td>2D MRI brain basal ganglia</td>
</tr>
<tr>
<td>2D MRI brain ventricle</td>
</tr>
<tr>
<td>2D cardiac MRI (LV myocardium)</td>
</tr>
<tr>
<td>2D CT bone</td>
</tr>
<tr>
<td>2D MRI femur</td>
</tr>
<tr>
<td>2D ultrasound breast cyst</td>
</tr>
<tr>
<td>2D liver wall</td>
</tr>
<tr>
<td>3D MRI brain cortex</td>
</tr>
<tr>
<td>3D MRI vertebrae</td>
</tr>
<tr>
<td>3D pelvic MRI (prostate gland)</td>
</tr>
<tr>
<td>Multi-dimensional (ND) data</td>
</tr>
</tbody>
</table>
Applications
Interactive Organ Segmentation (Boykov and Jolly, MICCAI 2000)

Segmentation of multiple objects. (a-c): Cardiac MRI. (d): Kidney CE-MR angiography
Interactive Organ Segmentation (Boykov and Jolly, MICCAI 2000)

Segmentation of the right lung in CT. (a): representative 2D slice of original 3D data. (b): segmentation results on the slice in (a). (c-d) 3D visualization of segmentation results.
AUTOMATIC HEART ISOLATION FOR CT CORONARY VISUALIZATION USING GRAPH-CUTS (Funka-Lea et al, ISBI 2006)

- **Isolating the entire heart** allows the **coronary vessels** on the surface of the heart to be easily visualized despite the proximity of surrounding organs such as the ribs and pulmonary blood vessels.
AUTOMATIC HEART ISOLATION FOR CT CORONARY VISUALIZATION USING GRAPH-CUTS (Funka-Lea et al, ISBI 2006)

• **Isolating the entire heart** allows the coronary vessels on the surface of the heart to be easily visualized despite the proximity of surrounding organs such as the ribs and pulmonary blood vessels.

• Numerous techniques have been described for segmenting the **left ventricle of the heart** in images from various types of medical scanners but rarely has the entire heart been segmented.
AUTOMATIC HEART ISOLATION FOR CT CORONARY VISUALIZATION USING GRAPH-CUTS (Funka-Lea et al, ISBI 2006)

- **Isolating the entire heart** allows the coronary vessels on the surface of the heart to be easily visualized despite the proximity of surrounding organs such as the ribs and pulmonary blood vessels.
- Numerous techniques have been described for segmenting the left ventricle of the heart in images from various types of medical scanners but rarely has the entire heart been segmented.
- **Graph-cut formation:**

\[
E(f) = E_{smooth}(f) + E_{data}(f) + E_{blob}(f)
\]
AUTOMATIC HEART ISOLATION FOR CT CORONARY VISUALIZATION USING GRAPH-CUTS (Funka-Lea et al, ISBI 2006)

• **Isolating the entire heart** allows the **coronary vessels** on the surface of the heart to be easily visualized despite the proximity of surrounding organs such as the ribs and pulmonary blood vessels.

• Numerous techniques have been described for segmenting the **left ventricle of the heart** in images from various types of medical scanners but rarely has the entire heart been segmented.

• **Graph-cut formation:**

\[ E(f) = E_{smooth}(f) + E_{data}(f) + E_{blob}(f) \]
AUTOMATIC HEART ISOLATION FOR CT CORONARY VISUALIZATION USING GRAPH-CUTS (Funka-Lea et al, ISBI 2006)

• **Isolating the entire heart** allows the coronary vessels on the surface of the heart to be easily visualized despite the proximity of surrounding organs such as the ribs and pulmonary blood vessels.

• Numerous techniques have been described for segmenting the **left ventricle of the heart** in images from various types of medical scanners but rarely has the entire heart been segmented.

• **Graph-cut formation:**

\[ E(f) = E_{smooth}(f) + E_{data}(f) + E_{blob}(f) \]

\[ E_{blob}(f) = \sum_{p,q \in N} B_{pq}(f(p), f(q); C) \]  

(C is center)
AUTOMATIC HEART ISOLATION FOR CT CORONARY VISUALIZATION USING GRAPH-CUTS (Funka-Lea et al, ISBI 2006)

- **Isolating the entire heart** allows the coronary vessels on the surface of the heart to be easily visualized despite the proximity of surrounding organs such as the ribs and pulmonary blood vessels.

- Numerous techniques have been described for segmenting the **left ventricle of the heart** in images from various types of medical scanners but rarely has the entire heart been segmented.

- **Graph-cut formation:**

\[
E(f) = E_{smooth}(f) + E_{data}(f) + E_{blob}(f)
\]

\[
E_{blob}(f) = \sum_{p,q \in N} B_{pq}(f(p), f(q); C)
\]

\[
B_{pq}(f(p), f(q)) = \cos^2(pq, pC) \quad \text{If } \cos(.) < 0, \quad 0 \text{ otherwise.}
\]
AUTOMATIC HEART ISOLATION FOR CT CORONARY VISUALIZATION USING GRAPH-CUTS (Funka-Lea et al, ISBI 2006)

**Top Left:**
A balloon is expanded within the heart. The heart wall pushes the balloon toward the heart center as the balloon grows.

**Top right:**
volume rendering of original heart volume.

**Bottom left:**
heart cropped based on segmentation mask.

**Bottom right:**
volume rendering after automatic heart isolation algorithm.

• Proposed a method for simultaneously segmenting longitudinal magnetic resonance (MR) images

• Proposed a method for simultaneously segmenting longitudinal magnetic resonance (MR) images
  – 3D MRI + time component (longitudinal)

• Proposed a method for simultaneously segmenting longitudinal magnetic resonance (MR) images
  – 3D MRI + time component (longitudinal)
• A 4D graph is used to represent the longitudinal data:
  – edges are weighted based on spatial and intensity priors and connect spatially and temporally neighboring voxels represented by vertices in the graph

• Proposed a method for simultaneously segmenting longitudinal magnetic resonance (MR) images
  – 3D MRI + time component (longitudinal)
• A 4D graph is used to represent the longitudinal data:
  – edges are weighted based on spatial and intensity priors and connect spatially and temporally neighboring voxels represented by vertices in the graph.
  – Solving the min-cut/max-flow problem on this graph yields the segmentation for all time-points in a single step

• Proposed a method for simultaneously segmenting longitudinal magnetic resonance (MR) images
  – 3D MRI + time component (longitudinal)

• A 4D graph is used to represent the longitudinal data:
  – edges are weighted based on spatial and intensity priors and connect spatially and temporally neighboring voxels represented by vertices in the graph.
  – Solving the min-cut/max-flow problem on this graph yields the segmentation for all time-points in a single step

• Time-series image can be considered as a single (4D) image, with the following dimensions: x,y,z and t

• Proposed a method for simultaneously segmenting longitudinal magnetic resonance (MR) images
  – 3D MRI + time component (longitudinal)

• A 4D graph is used to represent the longitudinal data:
  – edges are weighted based on spatial and intensity priors and connect spatially and temporally neighboring voxels represented by vertices in the graph.
  – Solving the min-cut/max-flow problem on this graph yields the segmentation for all time-points in a single step

• Time-series image can be considered as a single (4D) image, with the following dimensions: \( x, y, z \) and \( t \)

\[
\begin{align*}
\mathbb{p}_{x, y, z} & \quad \text{6 spatial neighbors in 3D} \\
\mathbb{p}_{x, y, z, t-1} & \quad \text{and} \\
\mathbb{p}_{x, y, z, t+1} & \quad \text{And 2 temporal neighbors}
\end{align*}
\]

(a) Right hippocampus segmentation (baseline), (b) follow up segmentation (12 months)
Integrated Graph Cuts for Brain Image Segmentation (Song et al, MICCAI 2006)

- In addition to image intensity, **tissue priors and local boundary information** are integrated into the edge weight metrics in the graph.
Integrated Graph Cuts for Brain Image Segmentation (Song et al, MICCAI 2006)

- In addition to image intensity, **tissue priors and local boundary information** are integrated into the edge weight metrics in the graph.

Example of the graph with three terminals for brain MRI tissue segmentation of gray matter (GM), white matter (WM), and cerebrospinal fluid (CSF). The set of nodes \( V \) includes all voxels and terminals. The set of edges \( E \) includes all \( n \)-links and \( t \)-links.

- \( n \)-links: voxel-to-voxel edges
- \( t \)-links: voxel-to-terminal edges
Integrated Graph Cuts for Brain Image Segmentation (Song et al, MICCAI 2006)

\[ E(f) = \sum_{p \in \mathcal{P}} (\lambda \gamma D_p(f_p) + (1 - \gamma)E_A(f_p)) + \gamma \sum_{\{p, q\} \in \mathcal{N}} V_{p,q}(f_p, f_q). \]
Integrated Graph Cuts for Brain Image Segmentation (Song et al, MICCAI 2006)

\[
E(f) = \sum_{p \in \mathcal{P}} (\lambda \gamma D_p(f_p) + (1 - \gamma) E_A(f_p)) + \gamma \sum_{\{p,q\} \in \mathcal{N}} V_{p,q}(f_p, f_q)
\]

- Regularization term
- Data term
- Atlas term
- Pairwise term
Integrated Graph Cuts for Brain Image Segmentation (Song et al, MICCAI 2006)

(T2, segmentation results)
GC + Appearance Model

Medical Image Segmentation by Combining Graph Cuts and Oriented Active Appearance Models

Xinjian Chen, Jayaram K. Udupa, Ulas Bagci, Ying Zhuge, and Jianhua Yao
Medical Image Segmentation by Combining Graph Cuts and Oriented Active Appearance Models

Xinjian Chen, Jayaram K. Udupa, Ulas Bagci, Ying Zhuge, and Jianhua Yao
Hierarchical Scale-Based Multiobject Recognition of 3-D Anatomical Structures

Ulas Bagci, Member, IEEE, Xinjian Chen, and Jayaram K. Udupa*, Fellow, IEEE
Regions used for calculating Dice score, sensitivity, specificity, and robust Hausdorff score. Region $T_1$ is the true lesion area (outline blue), $T_0$ is the remaining normal area. $P_1$ is the area that is predicted to be lesion by—for example—an algorithm (outlined red), and $P_0$ is predicted to be normal. $P_1$ has some overlap with $T_1$ in the right lateral part of the lesion, corresponding to the area referred to as $P_1 \land T_1$ in the definition of the Dice score. (Credits: BRATS paper)
Summary

- Data and Smoothness Terms -> Graph based segmentation methods
- Additional terms can(should) be added into segmentation formulation based observation/need and problem definition
- Problems formulated as a MRF task can be solved by max-flow/min-cut
Slide Credits and References

- Fredo Durand
- M. Tappen
- R. Szeliski
- [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.