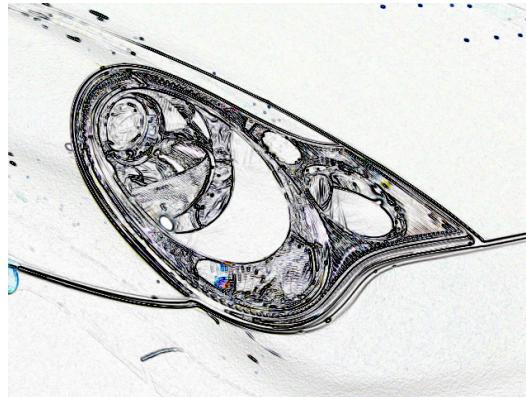


# CSE578: Computer Vision

Spring 2016:  
Image Segmentation



Anoop M. Namboodiri

Center for Visual Information Technology

IIIT Hyderabad, INDIA

# Three “Urges” on seeing a Picture\*

1. **To group** proximate and similar parts of the image into meaningful “regions”.  
Called **segmentation** in computer vision.
2. **To connect to memory** to recollect previously seen “objects”.  
Called **recognition** in computer vision.
3. **To measure** quantitative aspects such as number and sizes of objects, distances to/between them, etc.  
Called **reconstruction** in computer vision.

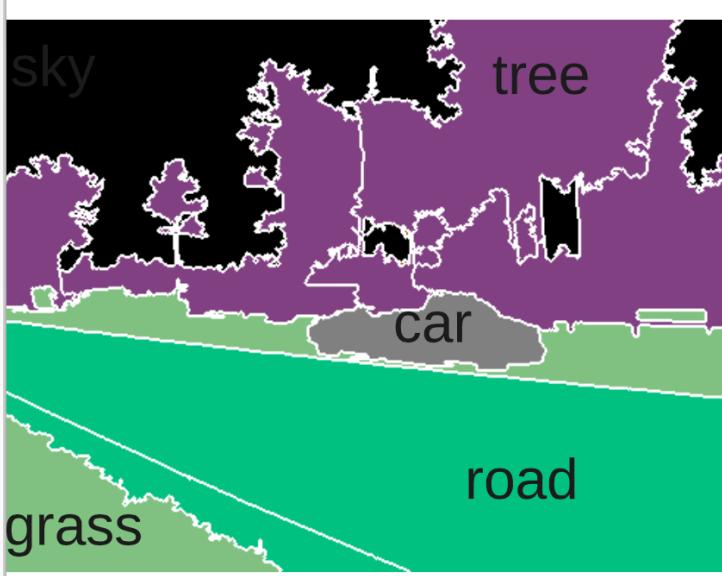
\*Jitendra Malik; Mysore Park, Dec. 2011

# Urge to Group



- We don't see individual pixels (like the computer does!).
- We see groups of pixels together.
- What is the basis for “correct” grouping?

# Urge to Group



- Group similar pixels together as objects.
- Group semantically meaningful pixels together as objects.
- Is appearance similarity the same as semantic similarity?

# Segmentation

- Dividing an image into semantically meaningful regions.



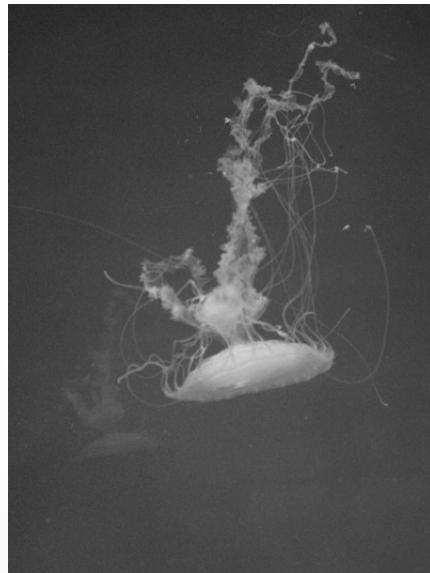
# Types of Segmentation

- Classification-based
  - Label pixels based on region properties
  - Label each pixel based on object models
- Region-based
  - Region growing and splitting
- Boundary-based
  - Find edges in the image and use them as region boundary
- Motion-based
  - Group pixels that have consistent motion (e.g., move in the same direction)

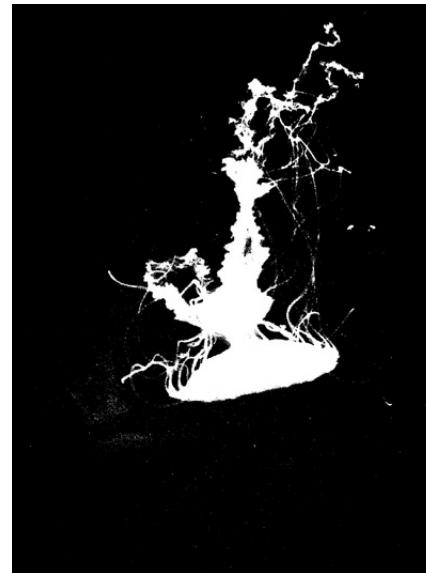
# Thresholding

Decide each pixel to be part of an object or background depending on its gray value

$$t(m, n) = \begin{cases} 1 & \text{if } u(m, n) > T \\ 0 & \text{if } u(m, n) \leq T \end{cases}$$



Original



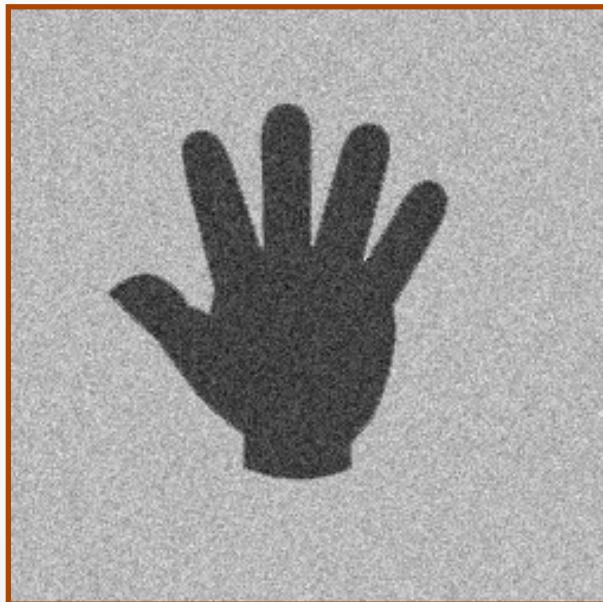
Thresholded (T=95)

# Types of Thresholding

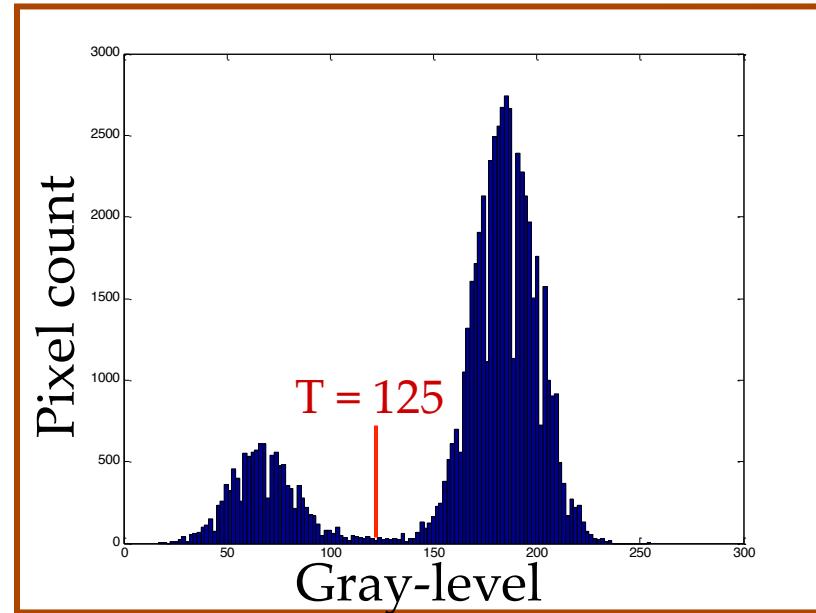
- Global
  - A single threshold is used for the whole image
  - How to determine the threshold?
- Adaptive (Local)
  - Decide the threshold for every pixel depending on its neighborhood
  - How to define the threshold function?

# Histogram

- A count of pixels of each graylevel (or range of graylevels) in an image



Grayscale Image



Histogram

# Thresholded Image



Original



Thresholded ( $T=125$ )

# Automatic Thresholding

1. Select an initial estimate of  $T$
2. Segment the image using  $T$ . Compute the mean gray values of the two regions,  $\mu_1$  and  $\mu_2$
3. Set the new threshold  $T=(\mu_1+\mu_2)/2$
4. Repeat 2 and 3 until  $T$  stabilizes

Assumptions: normal distribution, low noise

# Extensions

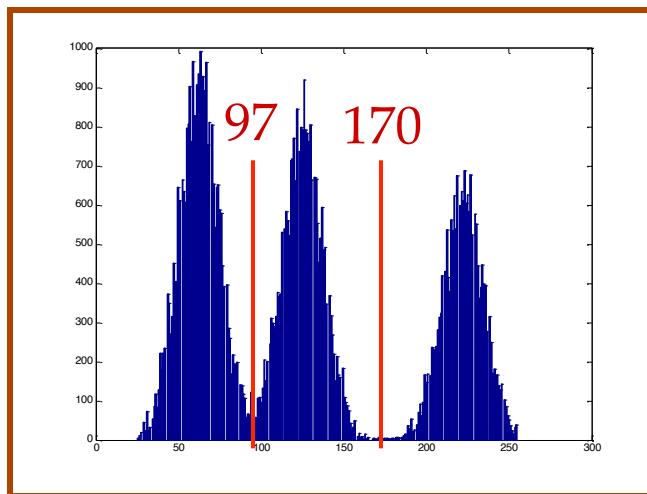
- Multiple Thresholds
  - Find multiple peaks and valleys in the gray level histogram
- Multi-spectral Thresholding
  - In color images, one could use different thresholds for each of the color channels

One might set all the background pixels to black, while leave the foreground at the original value so that the information is not lost.

# Multiple Thresholds



Original



Histogram

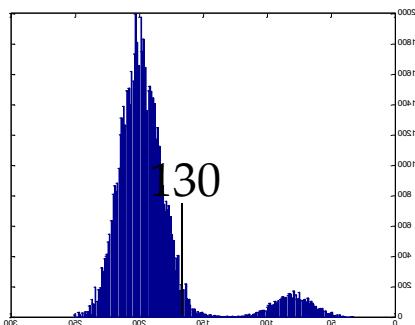
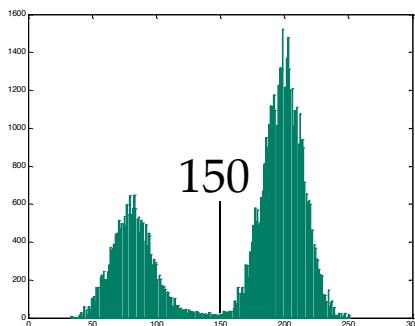
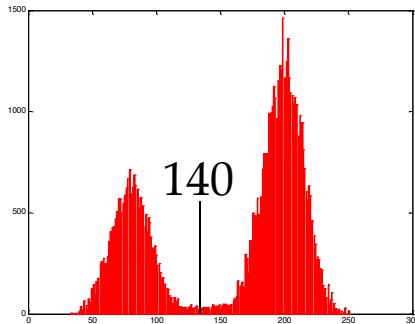


Thresholded

# Multi-spectral Thresholding



Original



Histograms



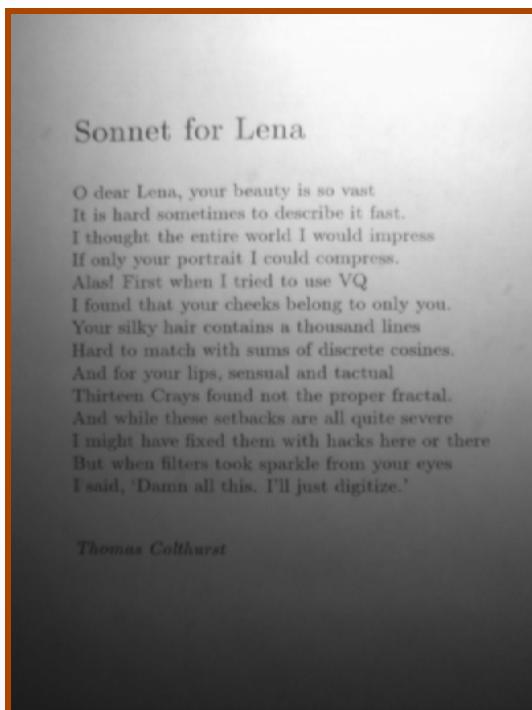
Thresholded

# Otsu's Method

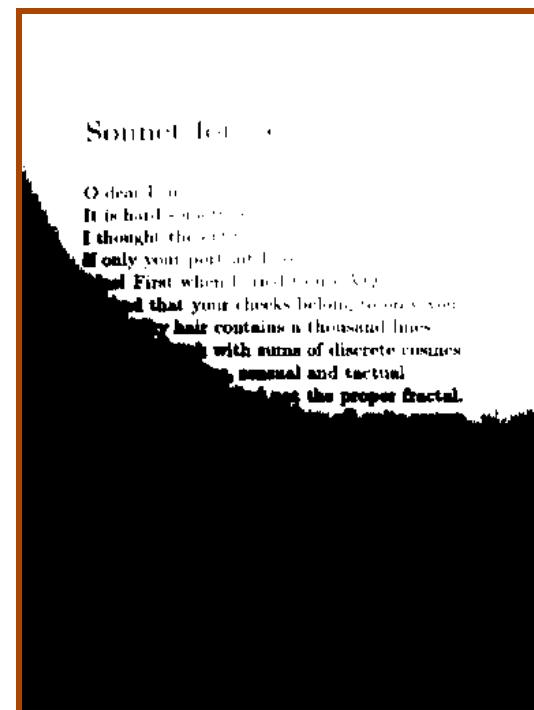
1. Compute histogram and probabilities of each intensity
2. Set up initial  $\omega_i(0)$  and  $\mu_i(0)$  and
3. Step through all possible thresholds  $t = 1..t_{\max}$ 
  1. Update  $\omega_i$  and  $\mu_i$
  2. Compute  $\sigma_b^2(t)$
4. Final threshold corresponds to the maximum  $\sigma_b^2(t)$
5. Compute two maxima (and respective thresholds  $t_1$  and  $t_2$ ) using  $>$  and  $\geq$  (first and last maxima)
6. Desired threshold =  $(t_1 + t_2)/2$

# Adaptive Thresholding

Adaptive thresholding changes the threshold dynamically over the image. This can accommodate strong illumination gradients and shadows



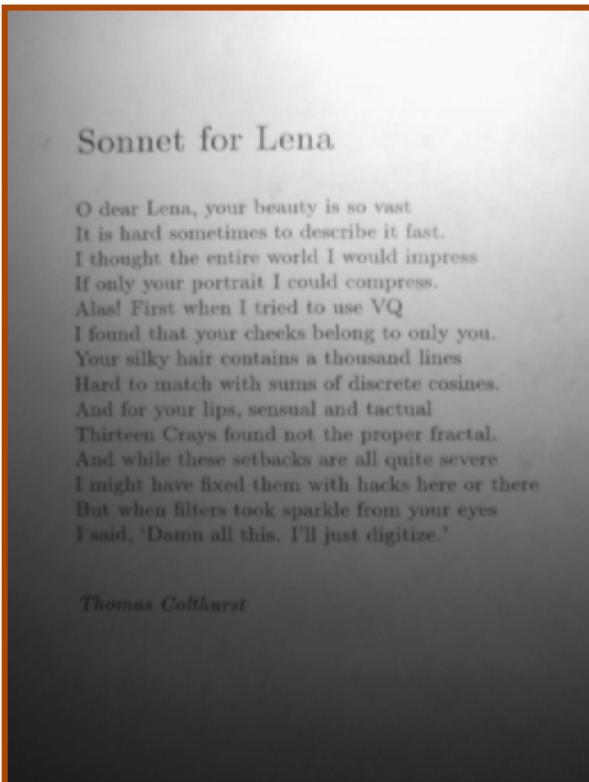
Original



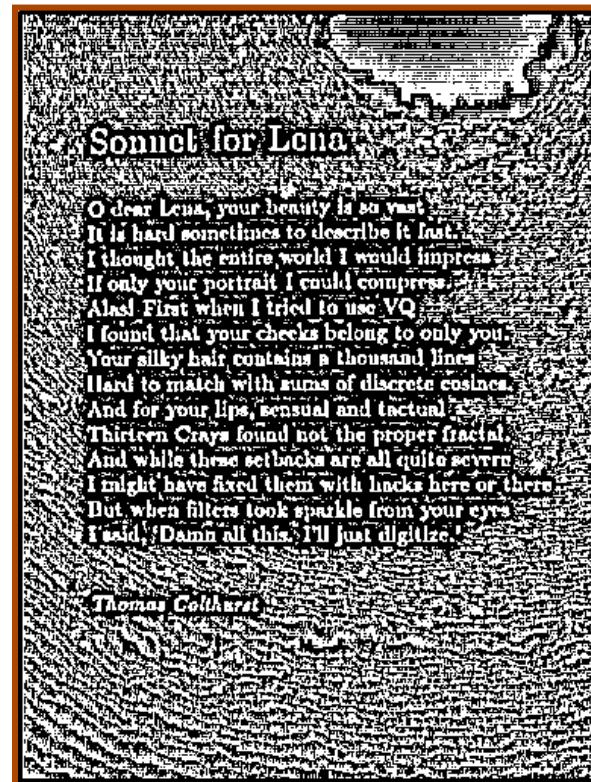
Single Threshold

# Adaptive Thresholding

Set the threshold as mean of pixels (gray values) in a neighborhood (say 7x7)



Original



Adaptive  
Threshold

# Adaptive Thresholding

- Thresholding using Mean-C
  - Set cxc image regions of uniform graylevel to background
- Chow and Kaneko
  1. Apply the mean operator (low pass filter)
  2. Subtract original image from the “mean” mage
  3. Threshold image in step 2
  4. Invert the result

**C.K. Chow and T. Kaneko** Automatic Boundary Detection of the Left Ventricle from Cineangiograms, Comp. Biomed. Res.(5), 1972, pp. 388-410.

# Adaptive Thresholding

## Chow & Kaneko Thresholding:

### Sonnet for Lena

O dear Lena, your beauty is so vast  
It is hard sometimes to describe it fast.  
I thought the entire world I would impress  
If only your portrait I could compress.  
Alas! First when I tried to use VQ  
I found that your cheeks belong to only you.  
Your silky hair contains a thousand lines  
Hard to match with sums of discrete cosines.  
And for your lips, sensual and tactual  
Thirteen Crays found not the proper fractal.  
And while these setbacks are all quite severe  
I might have fixed them with hacks here or there  
But when filters took sparkle from your eyes  
I said, 'Damn all this. I'll just digitize.'

*Thomas Colthurst*

Original

Low-pass filtered

### Sonnet for Lena

O dear Lena, your beauty is so vast  
It is hard sometimes to describe it fast.  
I thought the entire world I would impress  
If only your portrait I could compress.  
Alas! First when I tried to use VQ  
I found that your cheeks belong to only you.  
Your silky hair contains a thousand lines  
Hard to match with sums of discrete cosines.  
And for your lips, sensual and tactual  
Thirteen Crays found not the proper fractal.  
And while these setbacks are all quite severe  
I might have fixed them with hacks here or there  
But when filters took sparkle from your eyes  
I said, 'Damn all this. I'll just digitize.'

*Thomas Colthurst*

Difference

# Adaptive Thresholding Results

## Sonnet for Lena

O dear Lena, your beauty is so vast  
It is hard sometimes to describe it fast.  
I thought the entire world I would impress  
If only your portrait I could compress.  
Alas! First when I tried to use VQ  
I found that your cheeks belong to only you.  
Your silky hair contains a thousand lines  
Hard to match with sums of discrete cosines.  
And for your lips, sensual and tactful  
Thirteen Crays found not the proper fractal.  
And while these setbacks are all quite severe  
I might have fixed them with locks here or there  
But when filters took sparkle from your eyes  
I said, 'Damn all this. I'll just digitize.'

*Thomas Culhurst*

Chow & Kaneko Thresholding

## Sonnet for Lena

O dear Lena, your beauty is so vast  
It is hard sometimes to describe it fast.  
I thought the entire world I would impress  
If only your portrait I could compress.  
Alas! First when I tried to use VQ  
I found that your cheeks belong to only you.  
Your silky hair contains a thousand lines  
Hard to match with sums of discrete cosines.  
And for your lips, sensual and tactful  
Thirteen Crays found not the proper fractal.  
And while these setbacks are all quite severe  
I might have fixed them with locks here or there  
But when filters took sparkle from your eyes  
I said, 'Damn all this. I'll just digitize.'

*Thomas Culhurst*

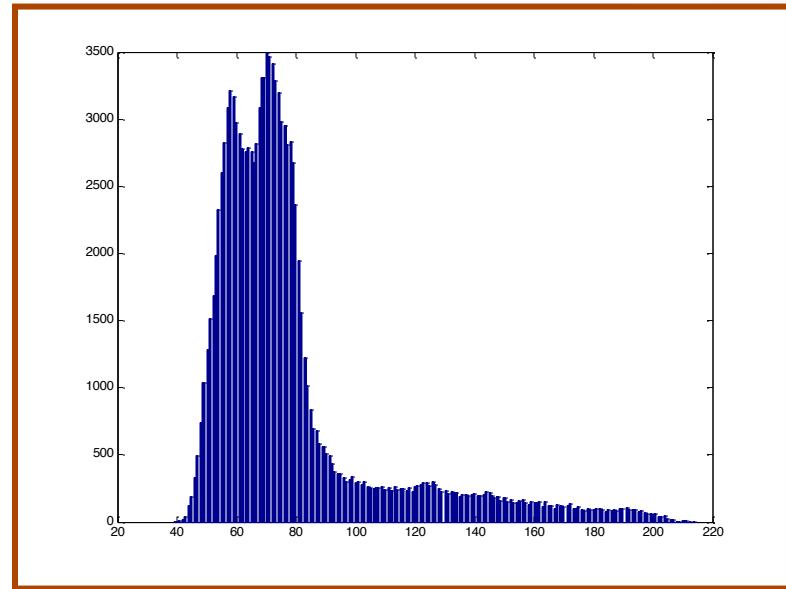
Mean-C (10) Thresholding

# Optimal Thresholding

- The graylevel histogram is approximated using a mixture of two gaussian distributions and set the threshold to minimize the segmentation error



Grayscale Image



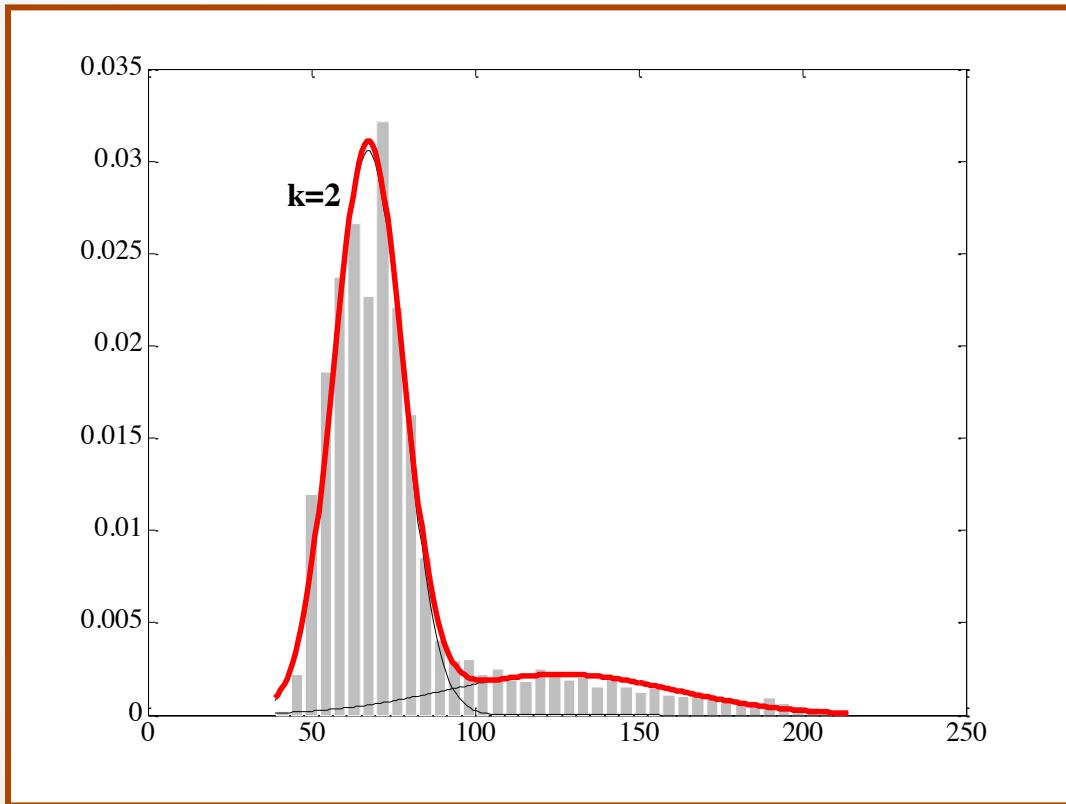
Histogram

# Gaussian Mixture Estimation by EM

- Obj: = 
$$N(\mu_1, \sigma_1) = \frac{1}{\sigma_1 \sqrt{2\pi}} e^{\frac{(x-\mu_1)^2}{2\sigma_1^2}}$$
- Bkg: = 
$$N(\mu_2, \sigma_2) = \frac{1}{\sigma_2 \sqrt{2\pi}} e^{\frac{(x-\mu_2)^2}{2\sigma_2^2}}$$

- Initialize  $\mu_1$ ,  $\sigma_1$ ,  $\mu_2$ , and  $\sigma_2$ .
- E-Step: Computed the expected pixel label assignments. This could be either hard or soft assignment.
- M-Step: Computed Maximum-(Log)Likelihood estimates of the parameters:  $\mu_1, \sigma_1, \mu_2, \sigma_2$
- Repeat the E and M steps until convergence

# Optimal Thresholding

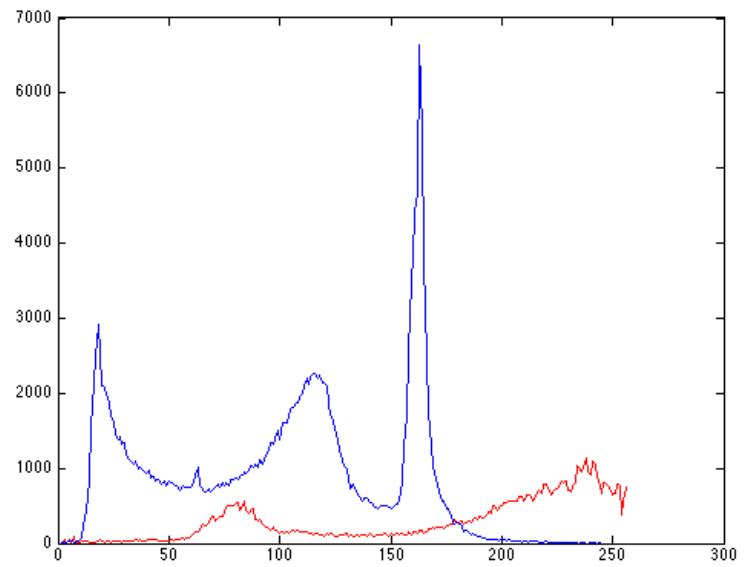
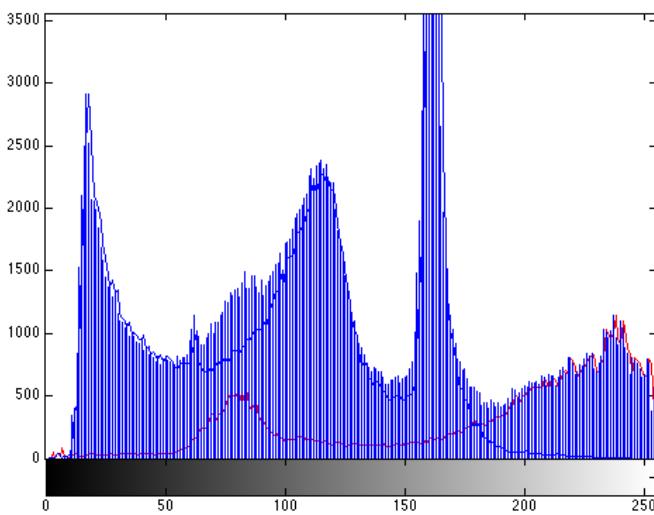
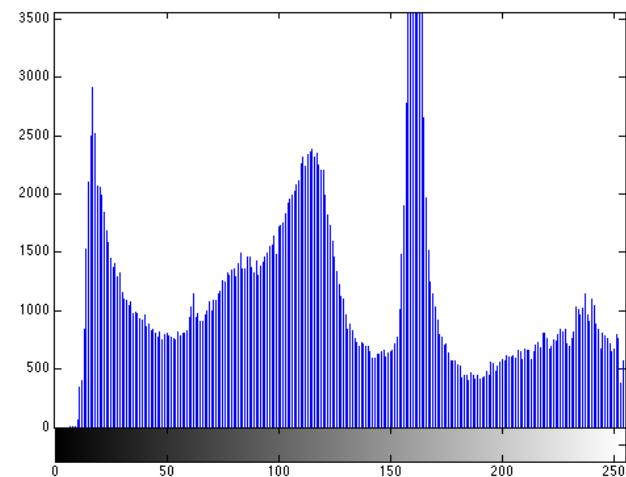


Histogram with bimodal fit



Thresholded ( $T=94$ )

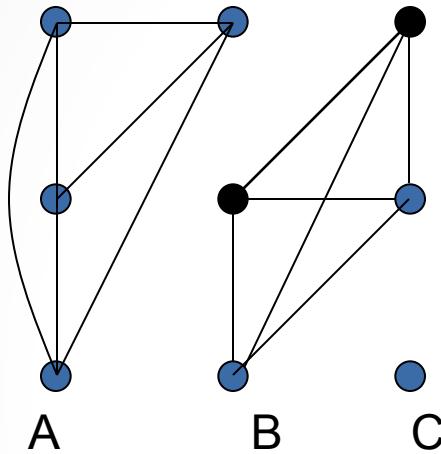
# Is Intensity Histogram Sufficient?



# Segmentation as Optimal Labeling

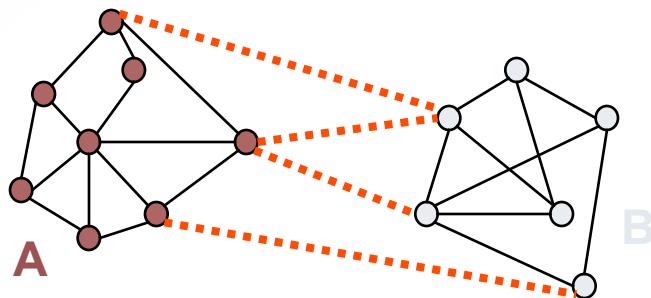
- Model knowledge about the world
- Classify each pixel as belonging to a specific object
  - Independent classification does not work
  - Need to incorporate neighborhood information
- Consider a graph over the image
  - Each node in the graph need to be labeled
  - Edges in the graph represent neighborhood constraints
- Define a cost function,  $Q(f)$ , using the above
- Compute the optimal labeling wrt  $Q(f)$ .

# Segmentation by Graph Cuts



- Break Graph into Segments
  - Delete links that cross between segments
  - Easiest to break links that have low cost (low similarity)
    - similar pixels should be in the same segments
    - dissimilar pixels should be in different segments

# Cuts in a graph



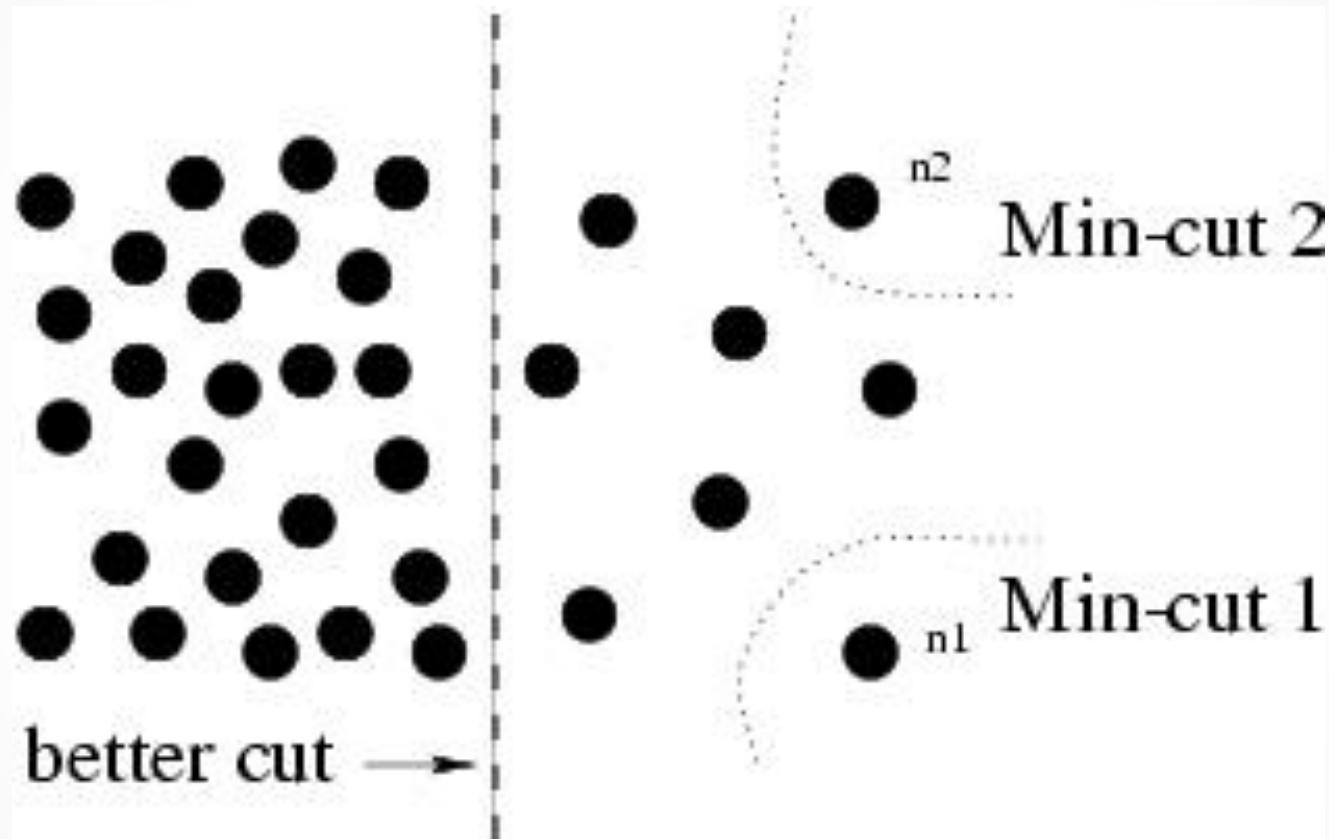
- Link Cut
  - set of links whose removal makes a graph disconnected
  - cost of a cut:

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

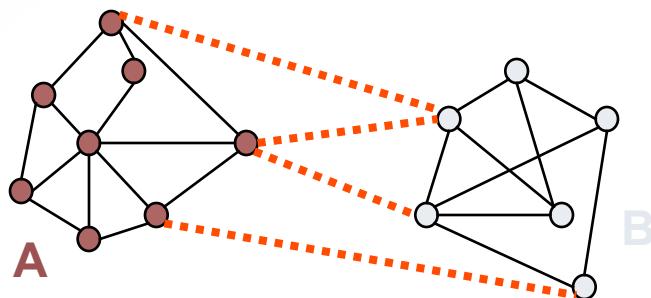
One idea: Find minimum cut

- gives you a segmentation
- fast algorithms exist for doing this

# Min cut is not always the best



# Cuts in a graph



## Normalized Cut

- a cut penalizes large segments
- fix by normalizing for size of segments

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

where  $assoc(A, V) = \sum_{u \in A, t \in V} w(u, t)$

# Recursive normalized cuts

1. Given an image or image sequence, set up a weighted graph:  $G=(V, E)$ 
    - Vertex for each pixel
    - Edge weight for nearby pairs of pixels
  2. Solve for eigenvectors with the smallest eigenvalues:  
 $(\mathbf{D} - \mathbf{W})\mathbf{y} = \lambda \mathbf{D}\mathbf{y}$ , where
    - Use the eigenvector with the second smallest eigenvalue to bipartition the graph
    - Note: this is an approximation
  3. Recursively repartition the segmented parts if necessary
- Details: <http://www.cs.berkeley.edu/~malik/papers/SM-ncut.pdf>

# Normalized cuts results



# Graphcut for Image Segmentation



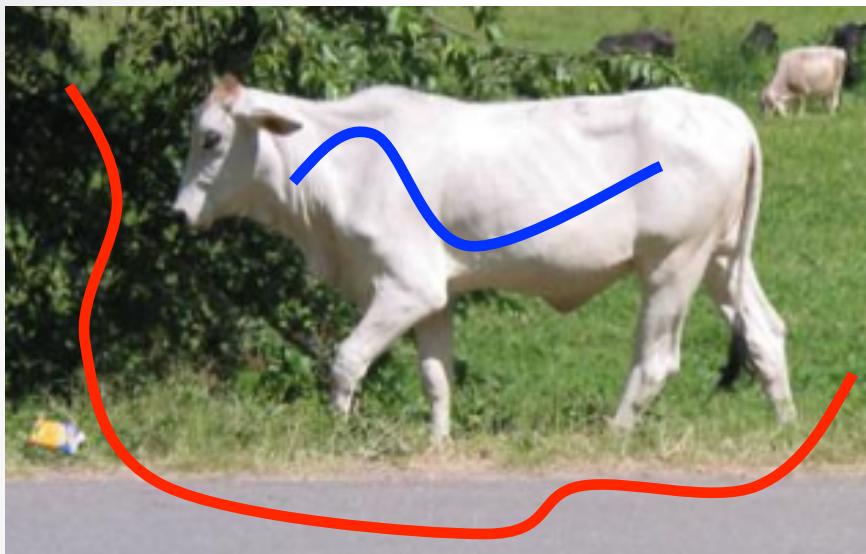
**How ?**

Cost function      Models *our* knowledge about natural images

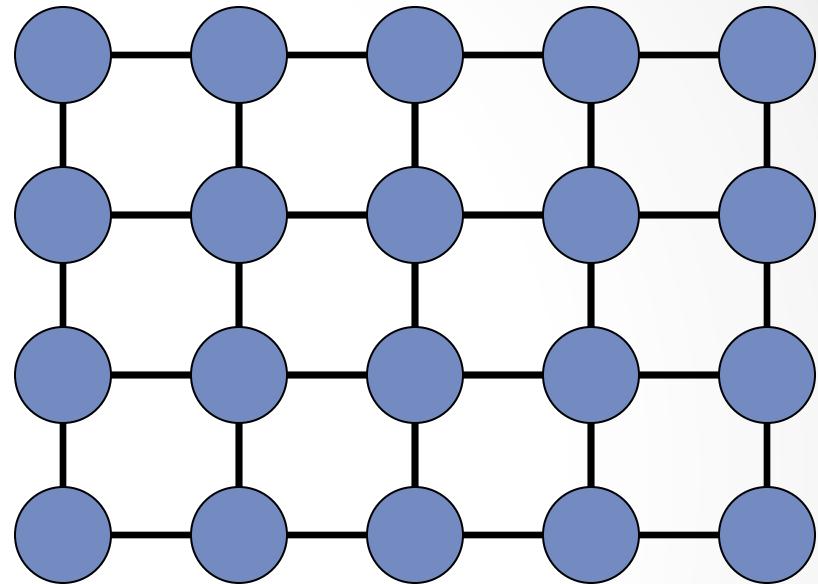
Optimize cost function to obtain the segmentation



# Binary Image Segmentation



Object - white, Background - green/grey



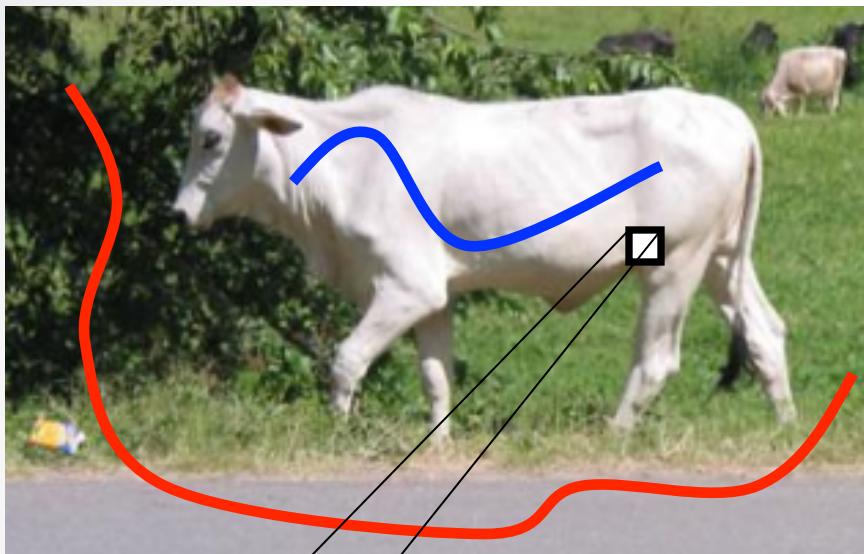
Graph  $G = (V, E)$

Each vertex corresponds to a pixel

Edges define a 4-neighbourhood *grid* graph

Assign a label to each vertex from  $L = \{\text{obj}, \text{bkg}\}$

# Binary Image Segmentation

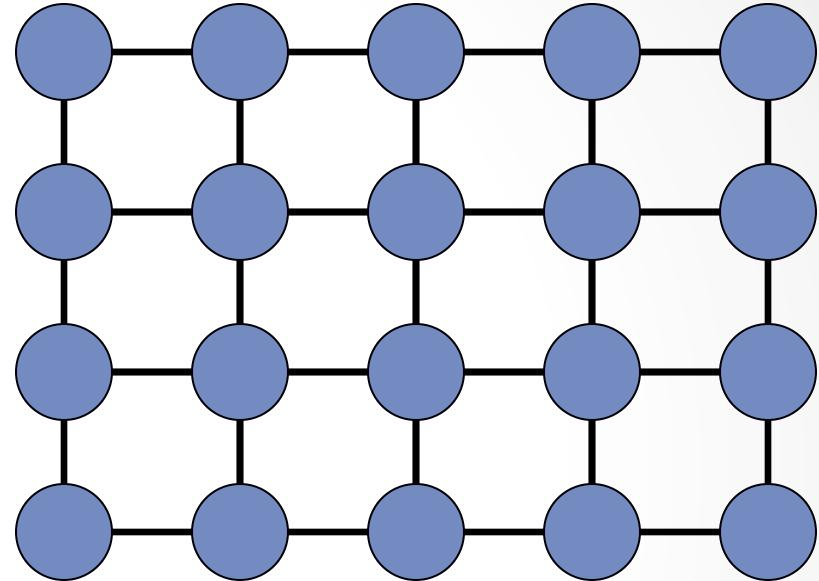


Object - white, Background - green/grey

Cost of a labelling  $f : V \rightarrow L$



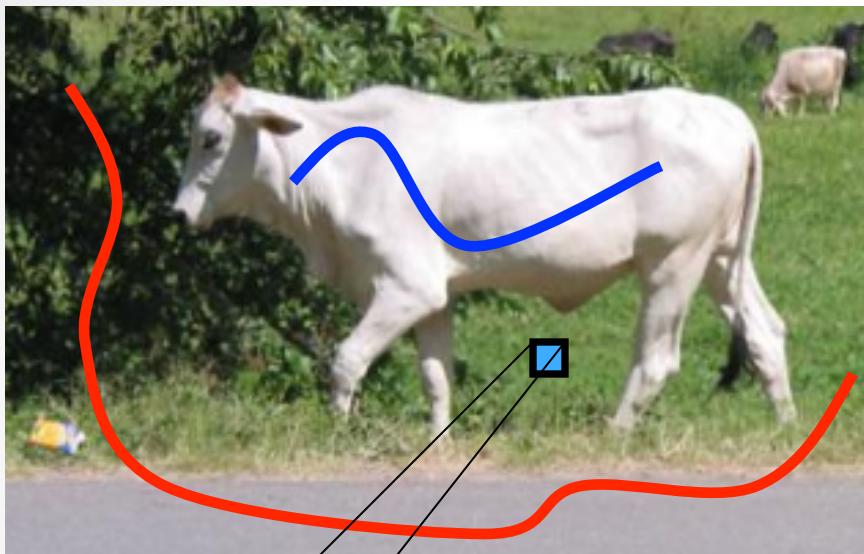
Cost of label ‘obj’ low Cost of label ‘bkg’ high



Graph  $G = (V, E)$

Per Vertex Cost

# Binary Image Segmentation

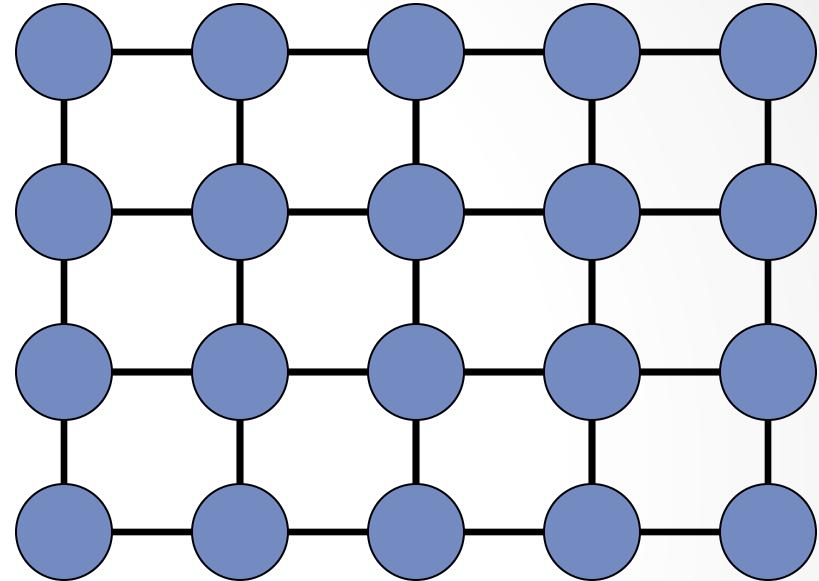


Object - white, Background - green/grey

Cost of a labelling  $f : V \rightarrow L$



Cost of label ‘obj’ high Cost of label ‘bkg’ low

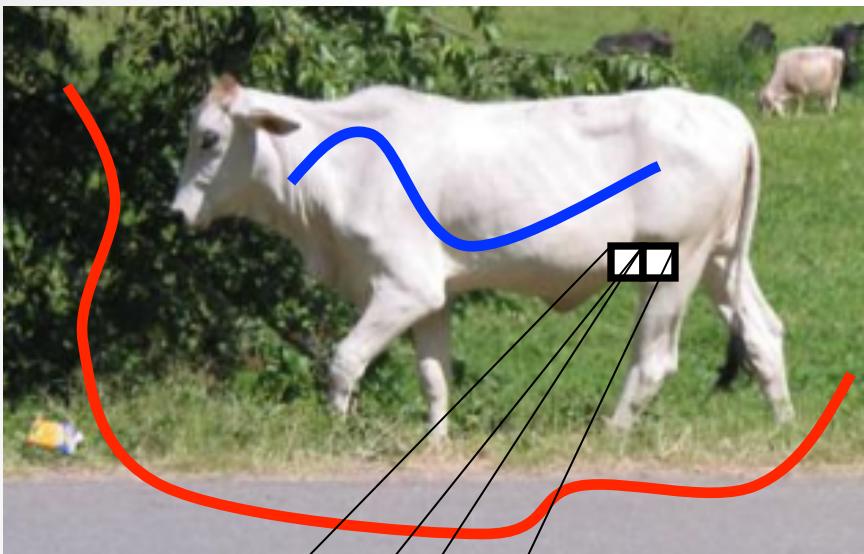


Graph  $G = (V, E)$

Per Vertex Cost

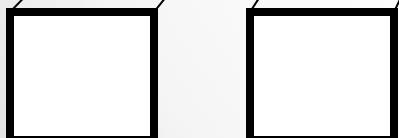
UNARY COST

# Binary Image Segmentation



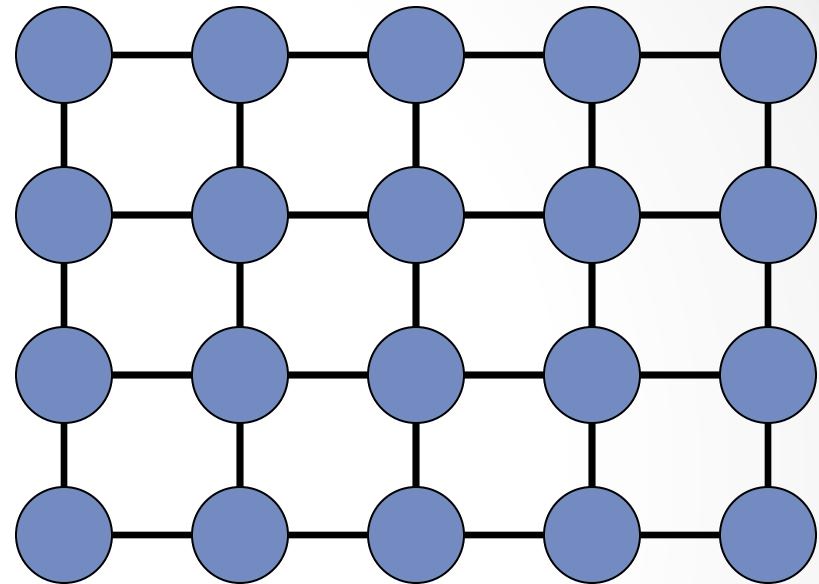
Object - white, Background - green/grey

Cost of a labelling  $f : V \rightarrow L$



Cost of same label low

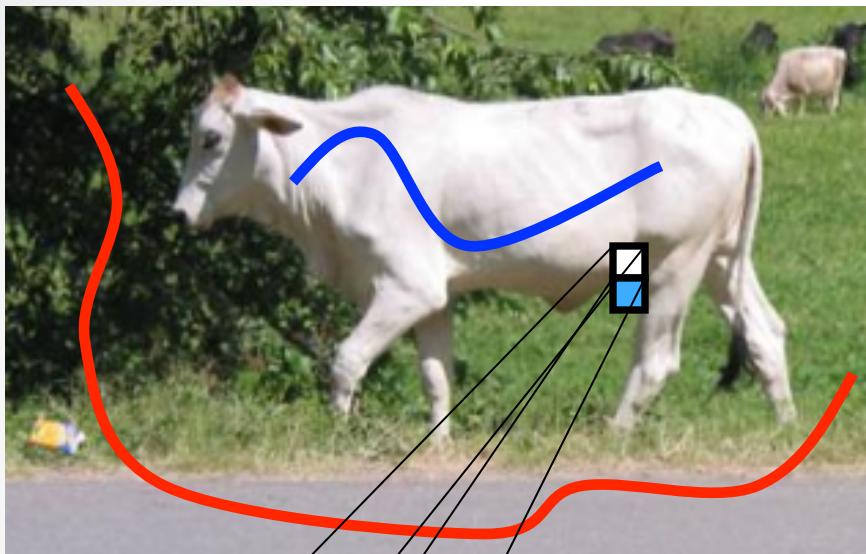
Cost of different labels high



Graph  $G = (V, E)$

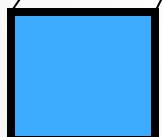
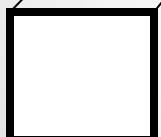
Per Edge Cost

# Binary Image Segmentation



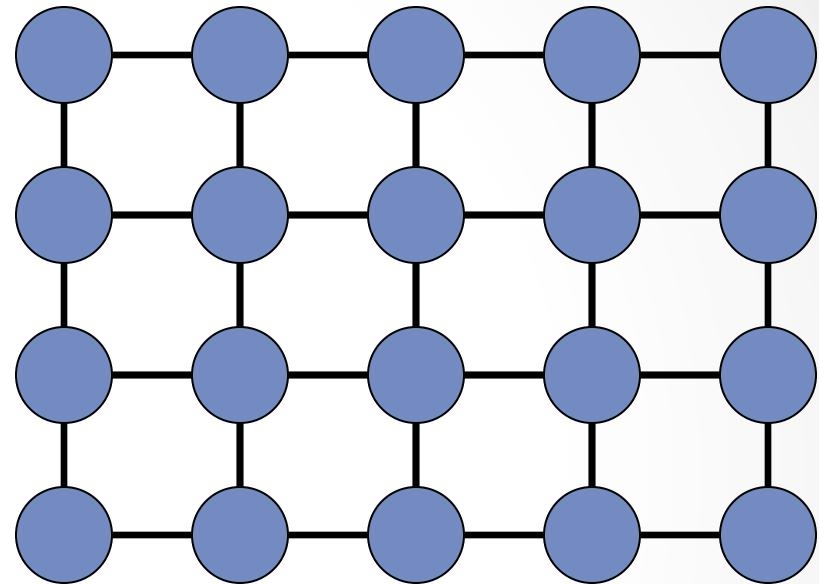
Object - white, Background - green/grey

Cost of a labelling  $f : V \rightarrow L$



Cost of same label high

Cost of different labels low

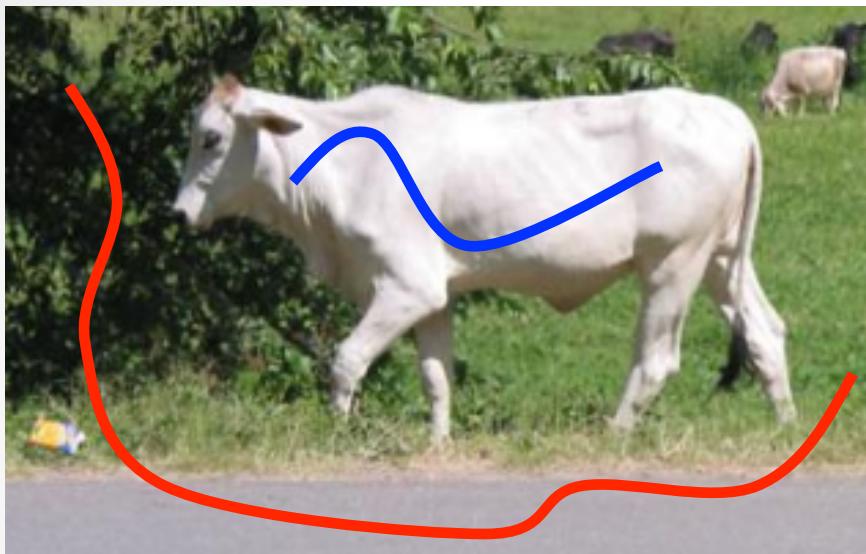


Graph  $G = (V, E)$

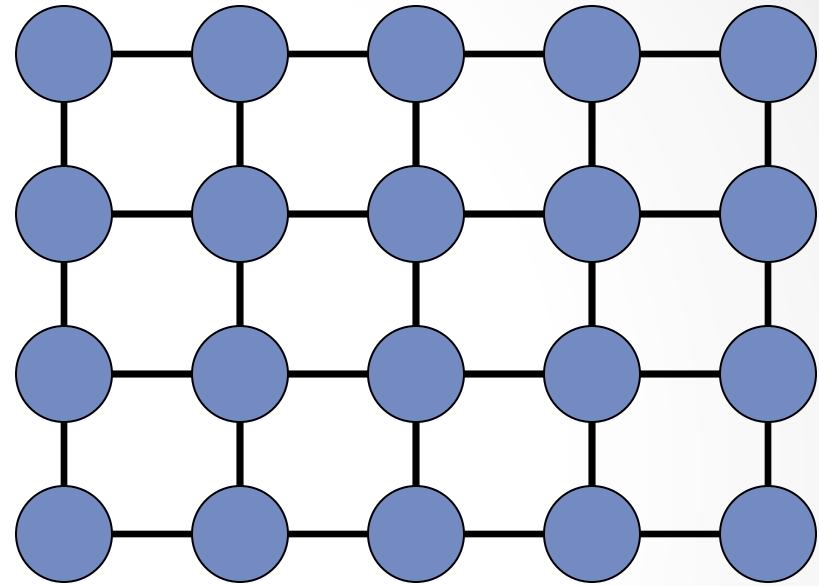
Per Edge Cost

PAIRWISE  
COST

# Binary Image Segmentation



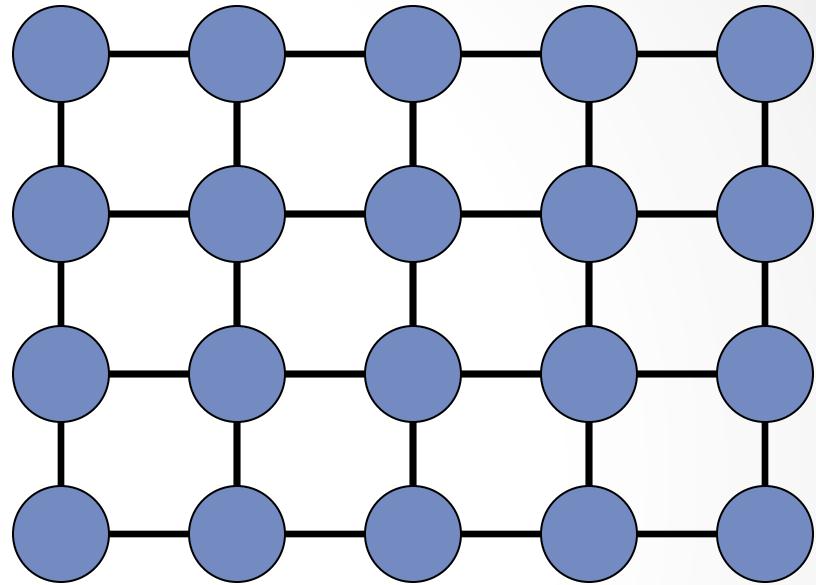
Object - white, Background - green/grey



Graph  $G = (V, E)$

Problem: Find the labelling with minimum cost  $f^*$

# Binary Image Segmentation



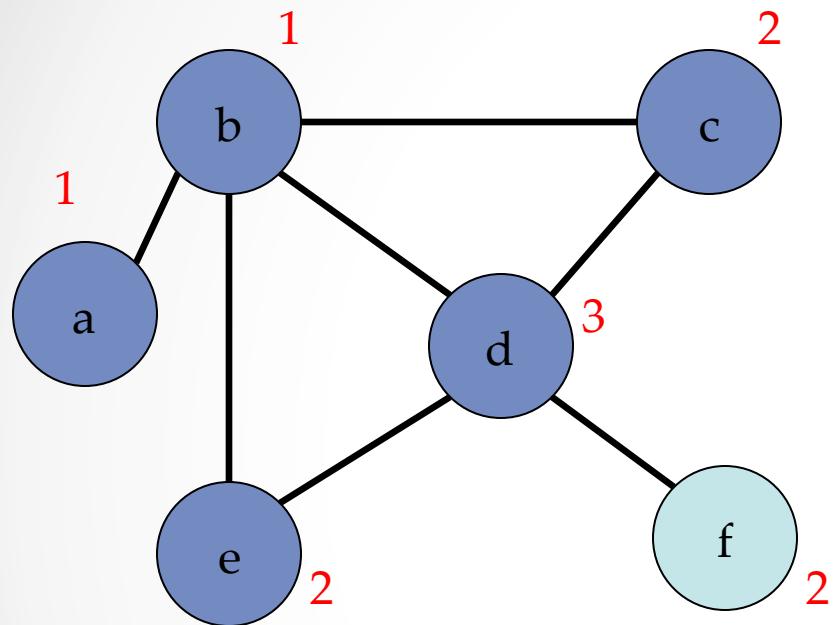
$$\text{Graph } G = (V, E)$$

$$L = \{fg, bg\}$$

Vertex corresponds to a pixel  
Edges define grid graph

Problem: Find the labeling with minimum cost  $f^*$

# The General Problem



Graph  $G = (V, E)$

Discrete label set  $L = \{1, 2, \dots, h\}$

Assign a label to each vertex  
 $f: V \rightarrow L$

Cost of a labelling  $Q(f)$

Unary Cost

Pairwise Cost

Find  $f^* = \arg \min Q(f)$

# Formulation: Energy Function

Label  $l_1$



Label  $l_0$



$V_a$

$D_a$



$V_b$

$D_b$



$V_c$

$D_c$



$V_d$

$D_d$

Random Variables  $V = \{V_a, V_b, \dots\}$

Labels  $L = \{l_0, l_1, \dots\}$  Data  $D$

Labelling  $f: \{a, b, \dots\} \rightarrow \{0, 1, \dots\}$

# Energy Function

Label  $l_1$

2

4

6

3

Label  $l_0$

5

2

3

7

$V_a$

$V_b$

$V_c$

$V_d$

$D_a$

$D_b$

$D_c$

$D_d$

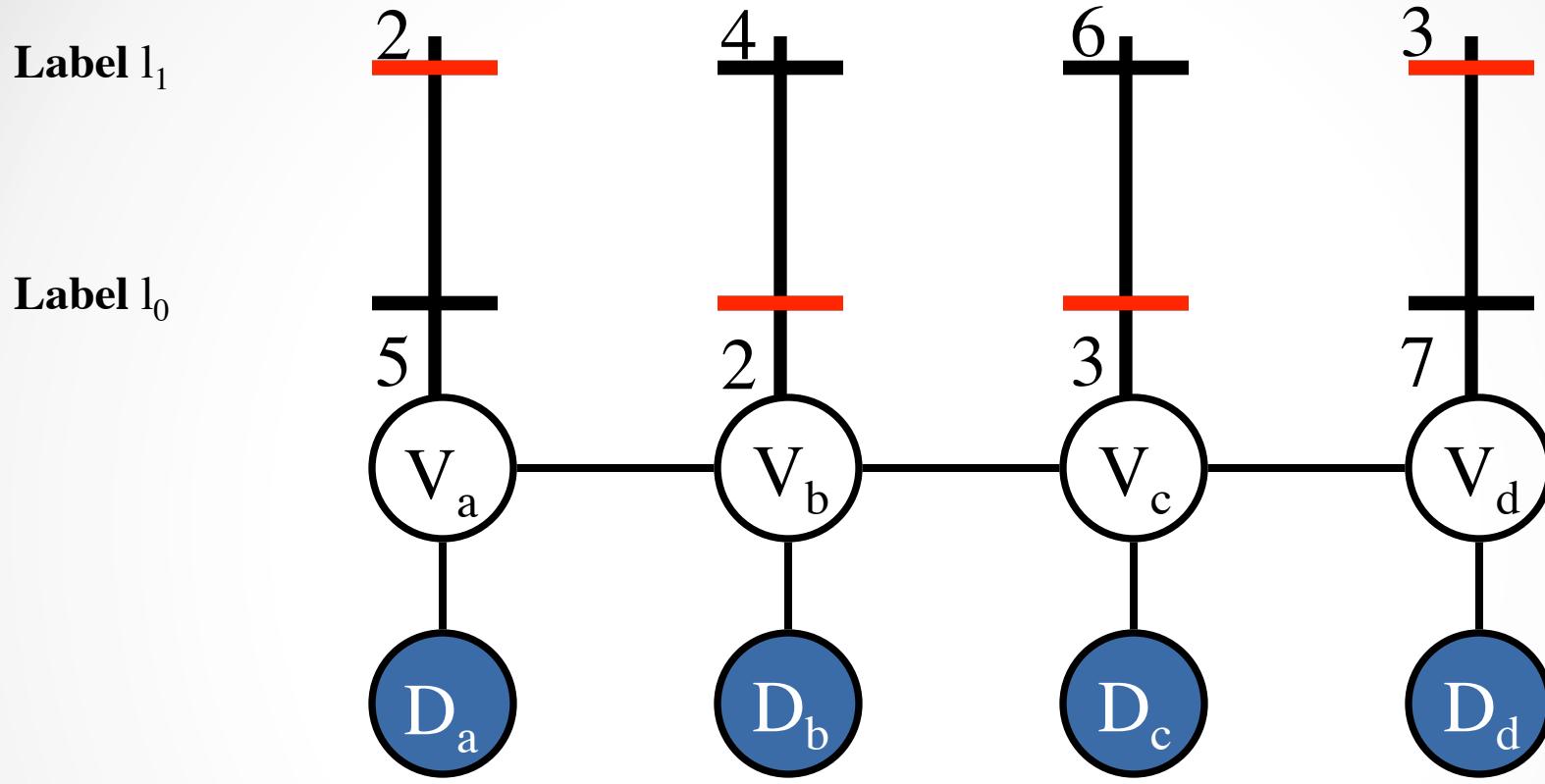
$$Q(f) = \sum_a \theta_{a;f(a)}$$

Unary Potential

Easy to minimize

Neighbourhood

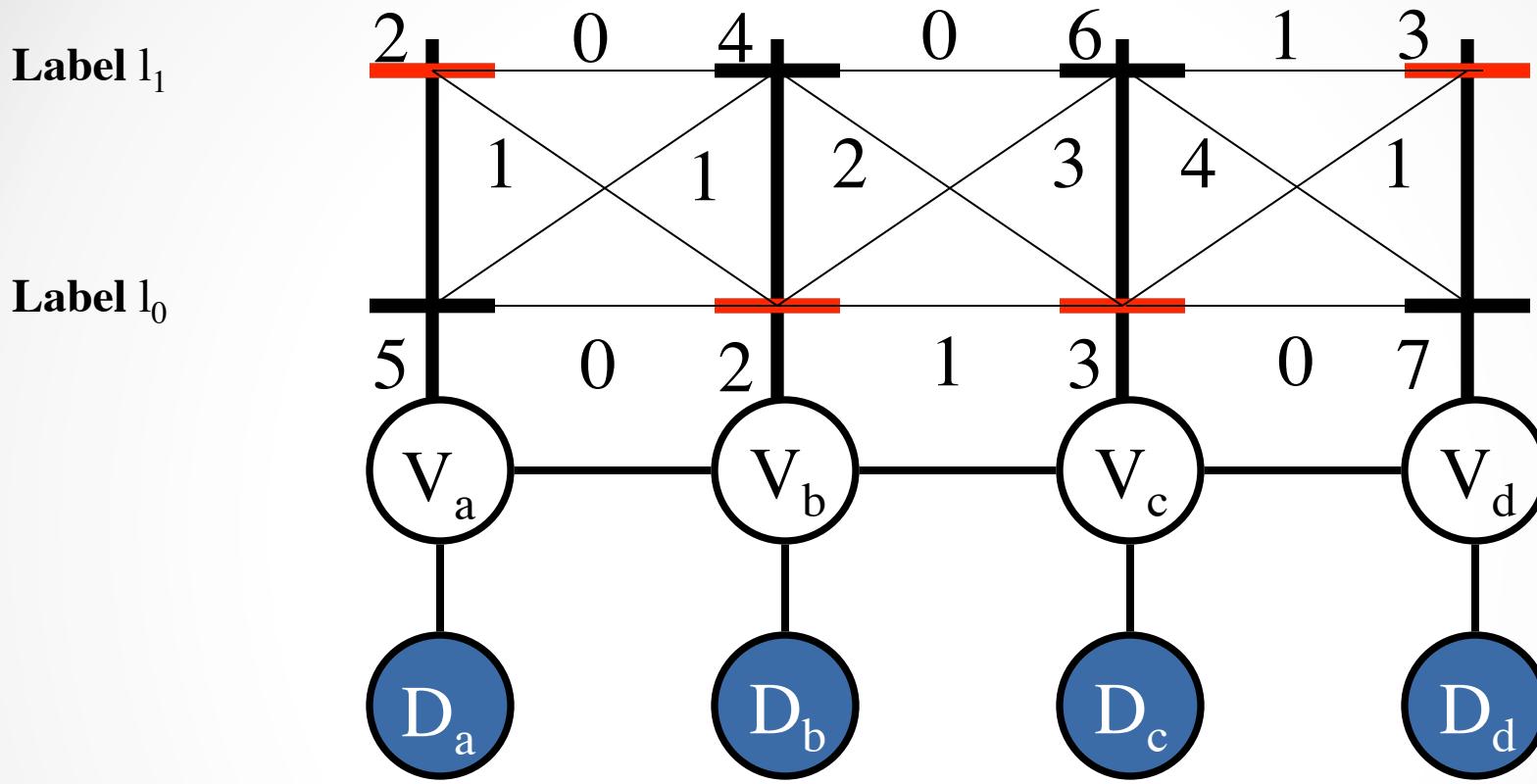
# Energy Function



$E : (a,b) \in E \text{ iff } V_a \text{ and } V_b \text{ are neighbours}$

$$E = \{ (a,b), (b,c), (c,d) \}$$

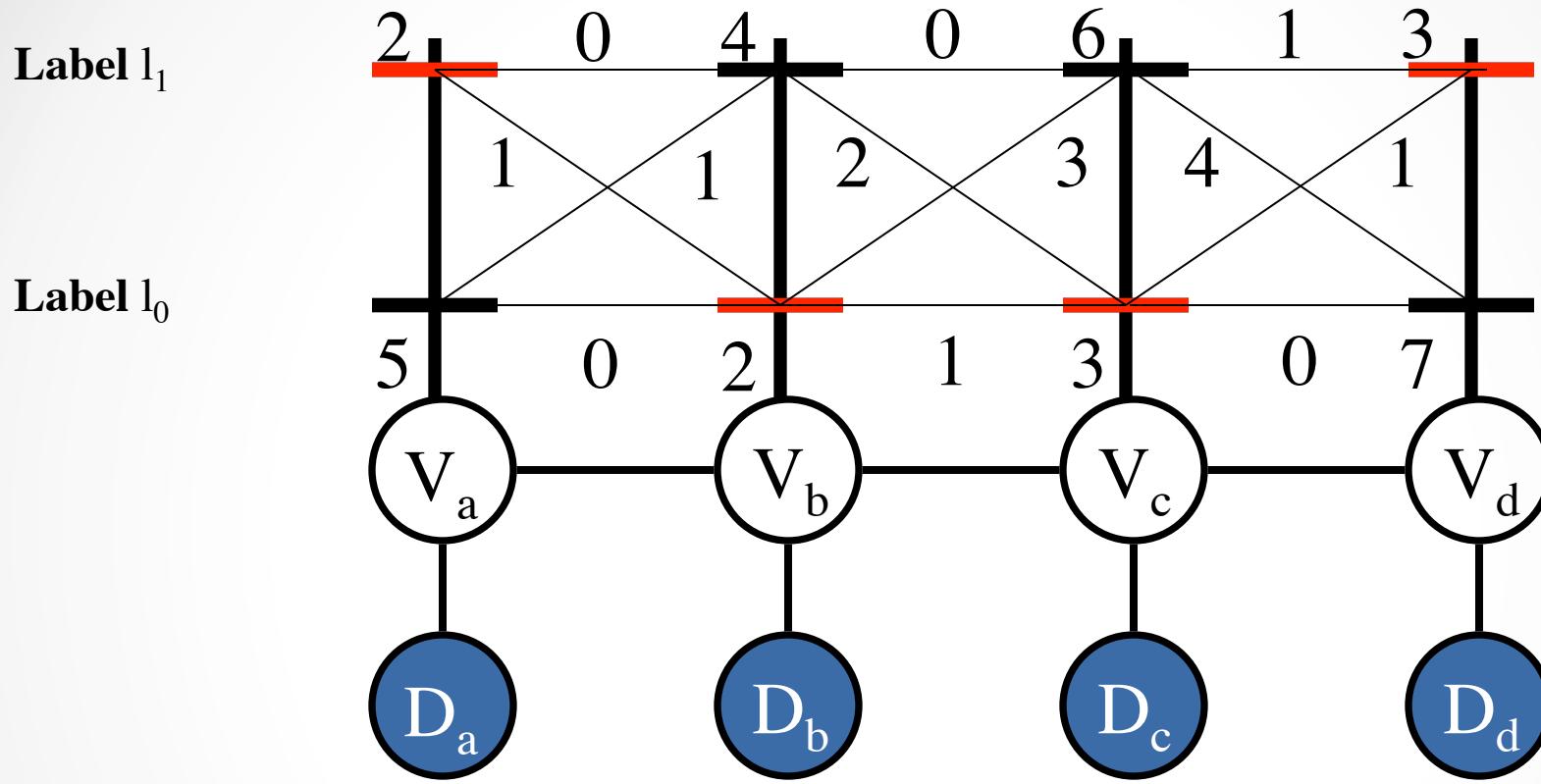
# Energy Function



Pairwise Potential

$$Q(f) = \sum_a \theta_{a;f(a)} + \sum_{(a,b)} \theta_{ab;f(a)f(b)}$$

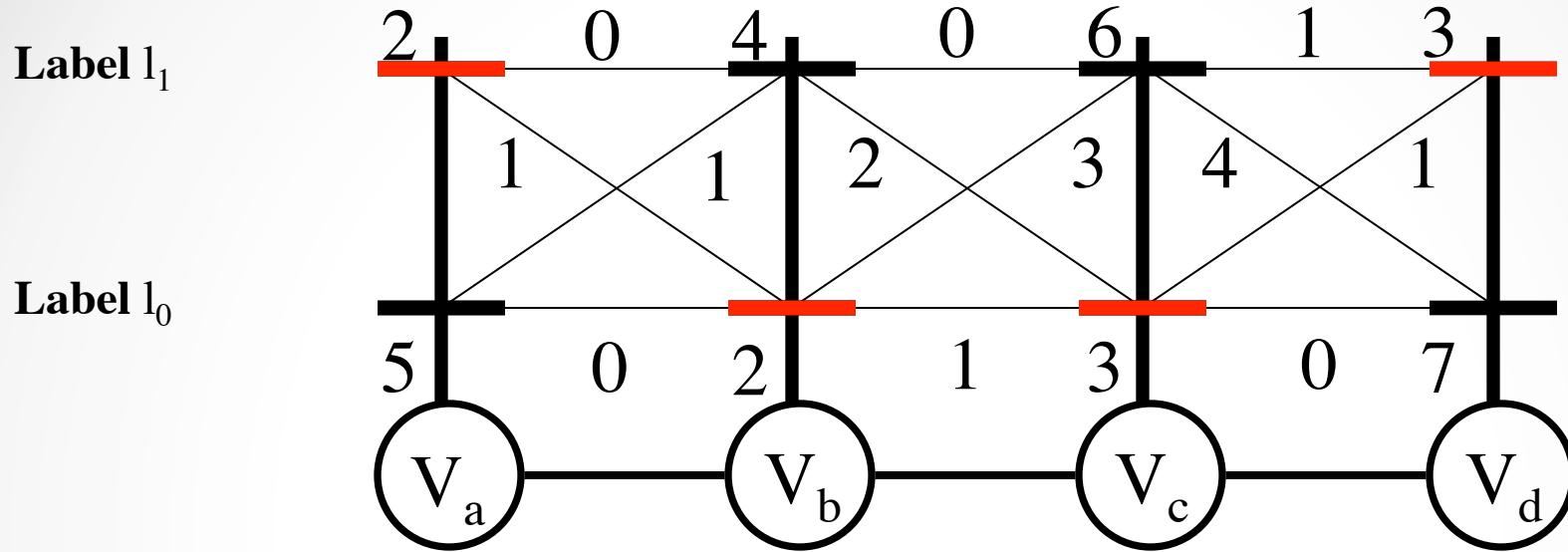
# Energy Function



$$Q(f; \theta) = \sum_a \theta_{a;f(a)} + \sum_{(a,b)} \theta_{ab;f(a)f(b)}$$

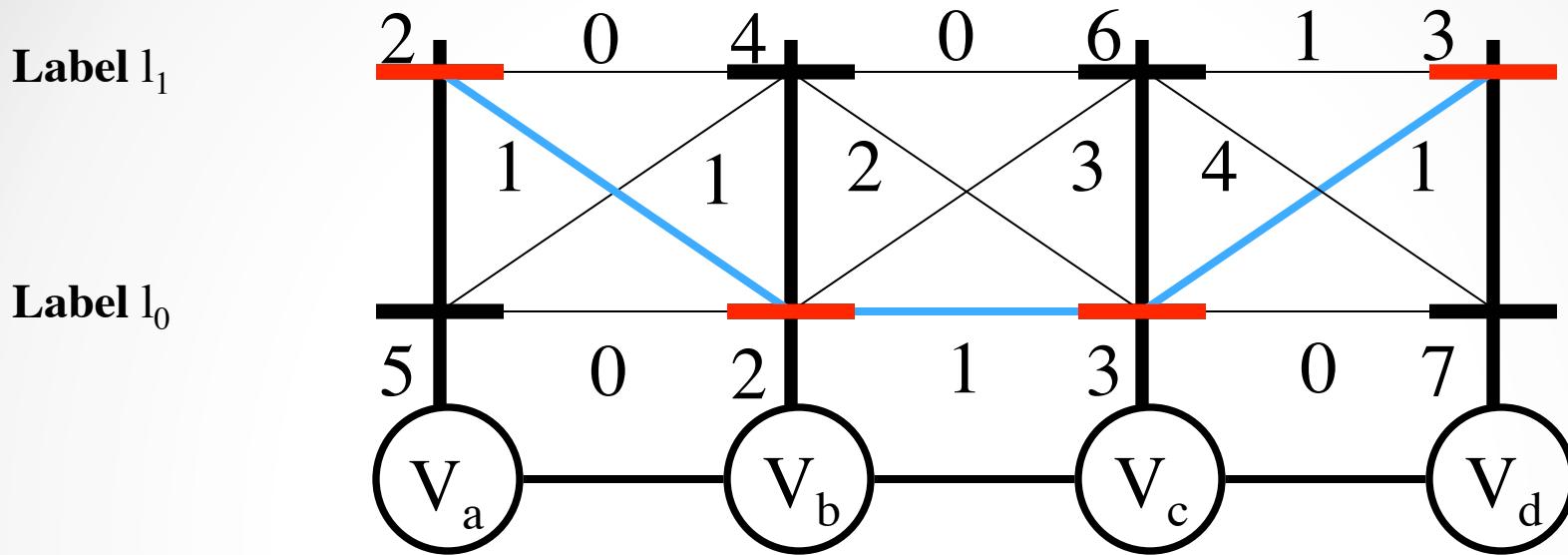
Parameter

# MAP Estimation



$$Q(f; \theta) = \sum_a \theta_{a;f(a)} + \sum_{(a,b)} \theta_{ab;f(a)f(b)}$$

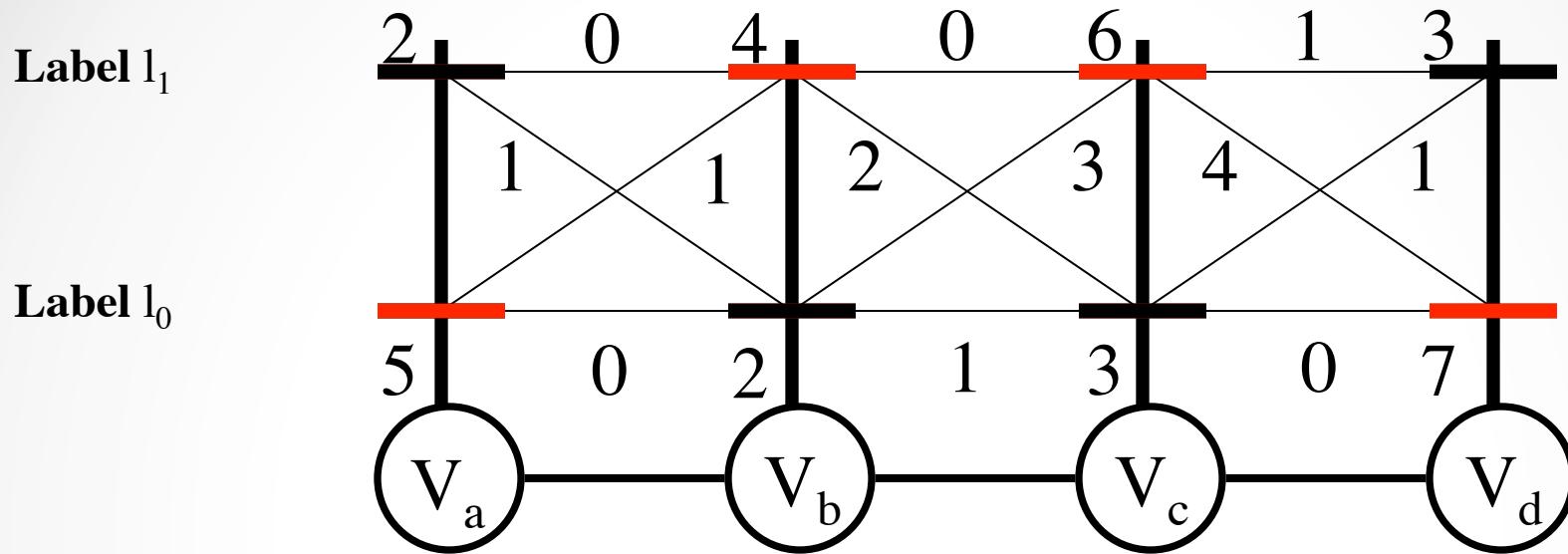
# MAP Estimation



$$Q(f; \theta) = \sum_a \theta_{a;f(a)} + \sum_{(a,b)} \theta_{ab;f(a)f(b)}$$

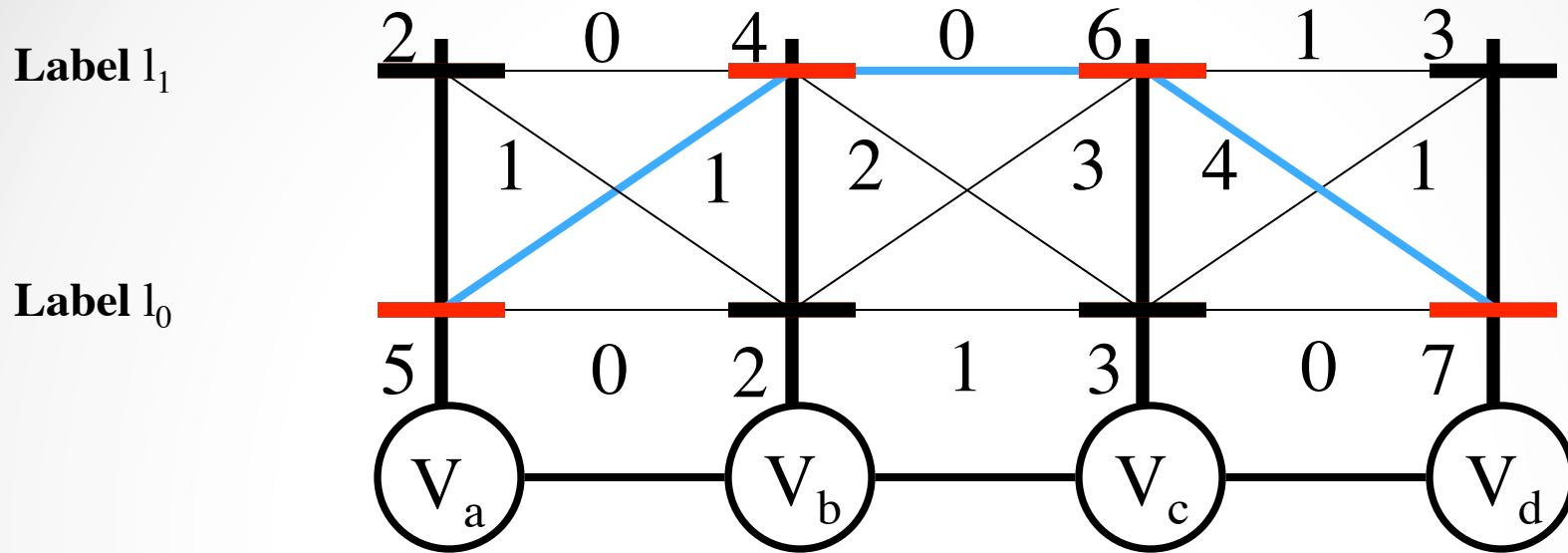
$$2 + 1 + 2 + 1 + 3 + 1 + 3 = 13$$

# MAP Estimation



$$Q(f; \theta) = \sum_a \theta_{a;f(a)} + \sum_{(a,b)} \theta_{ab;f(a)f(b)}$$

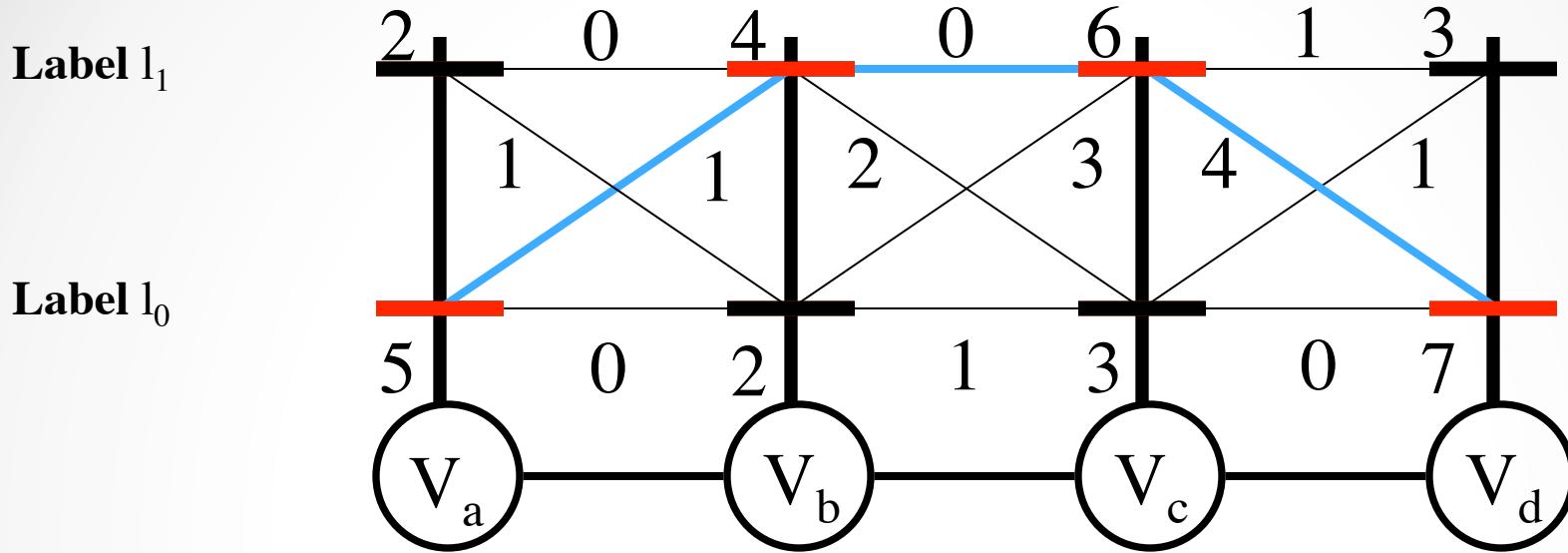
# MAP Estimation



$$Q(f; \theta) = \sum_a \theta_{a;f(a)} + \sum_{(a,b)} \theta_{ab;f(a)f(b)}$$

$$5 + 1 + 4 + 0 + 6 + 4 + 7 = 27$$

# MAP Estimation



$$q^* = \min Q(f; \theta) = Q(f^*; \theta)$$

$$Q(f; \theta) = \sum_a \theta_{a; f(a)} + \sum_{(a,b)} \theta_{ab; f(a)f(b)}$$

$$f^* = \arg \min Q(f; \theta)$$

# MAP Estimation

16 possible labellings

$$f^* = \{1, 0, 0, 1\}$$

$$q^* = 13$$

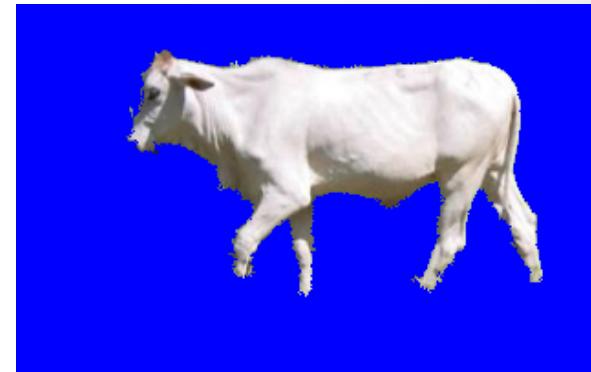
$f(a)$	$f(b)$	$f(c)$	$f(d)$	$Q(f; \theta)$
0	0	0	0	18
0	0	0	1	15
0	0	1	0	27
0	0	1	1	20
0	1	0	0	22
0	1	0	1	19
0	1	1	0	27
0.	1	1	1	20

$f(a)$	$f(b)$	$f(c)$	$f(d)$	$Q(f; \theta)$
1	0	0	0	16
1	0	0	1	13
1	0	1	0	25
1	0	1	1	18
1	1	0	0	18
1	1	0	1	15
1	1	1	0	23
1	1	1	1	16

# Computational Complexity

Segmentation

$$2^{|V|}$$



$$|V| = \text{number of pixels} \approx 320 * 480 = 153600$$

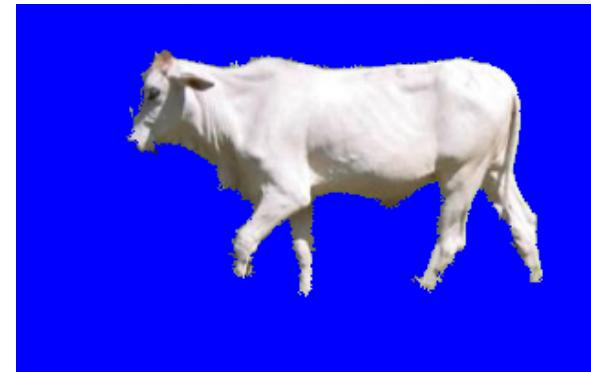
Can we do better than brute-force?

MAP Estimation is NP-hard !!

# Computational Complexity

Segmentation

$$2^{|V|}$$

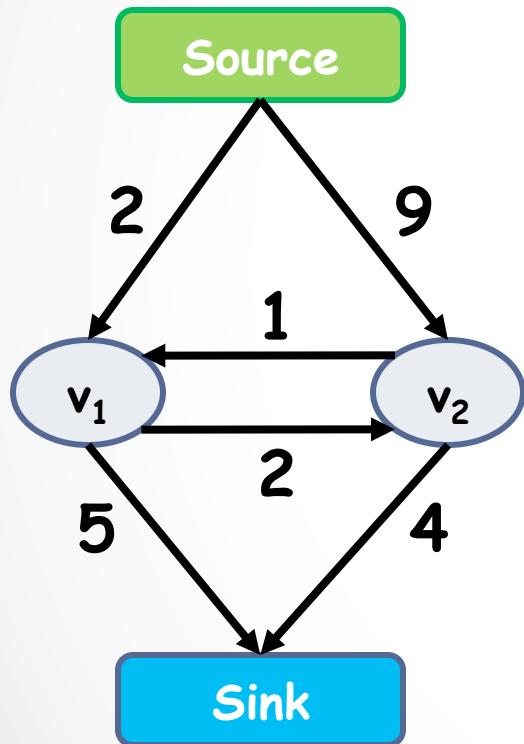


$$|V| = \text{number of pixels} \approx 320 * 480 = 153600$$

Exact algorithms do exist for special cases

Good approximate algorithms for general case

# The st-Mincut Problem



**Graph ( $V, E, C$ )**

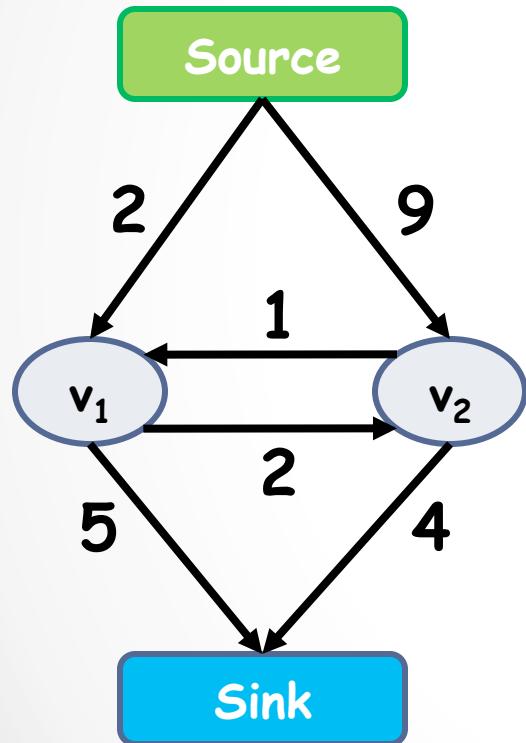
Vertices  $V = \{v_1, v_2 \dots v_n\}$

Edges  $E = \{(v_1, v_2) \dots\}$

Costs  $C = \{c_{(1, 2)} \dots\}$

# The Solution: st-Mincut

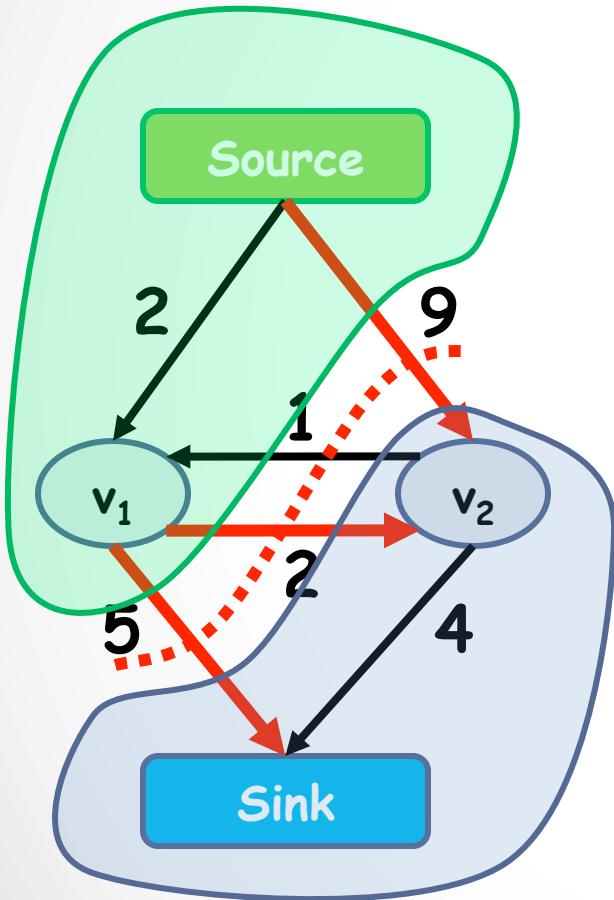
What is an st-cut?



# The st-Mincut Problem

What is an st-cut?

An st-cut ( $S, T$ ) divides the nodes between source and sink.



What is the cost of an st-cut?

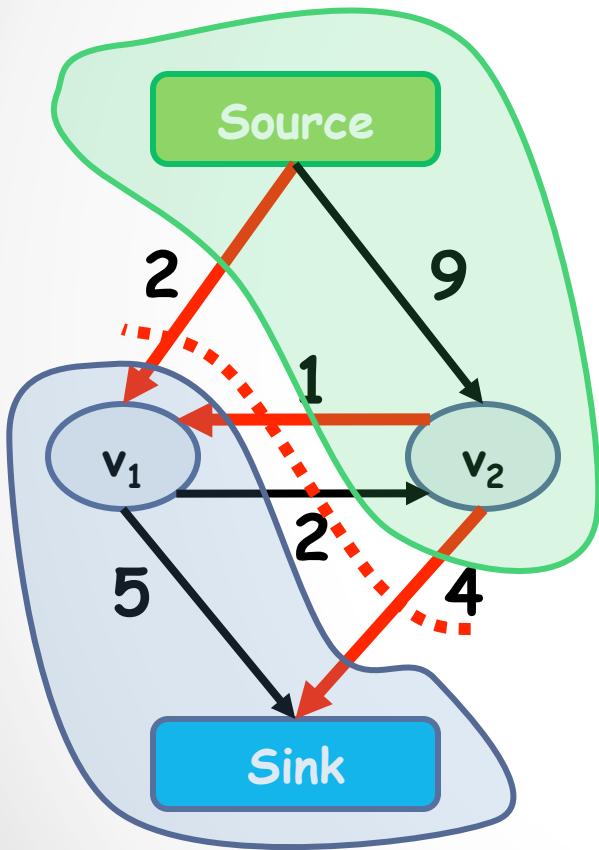
Sum of cost of all edges going from S to T

$$5 + 2 + 9 = 16$$

# The st-Mincut Problem

What is an st-cut?

An st-cut ( $S, T$ ) divides the nodes between source and sink.



What is the cost of an st-cut?

Sum of cost of all edges going from S to T

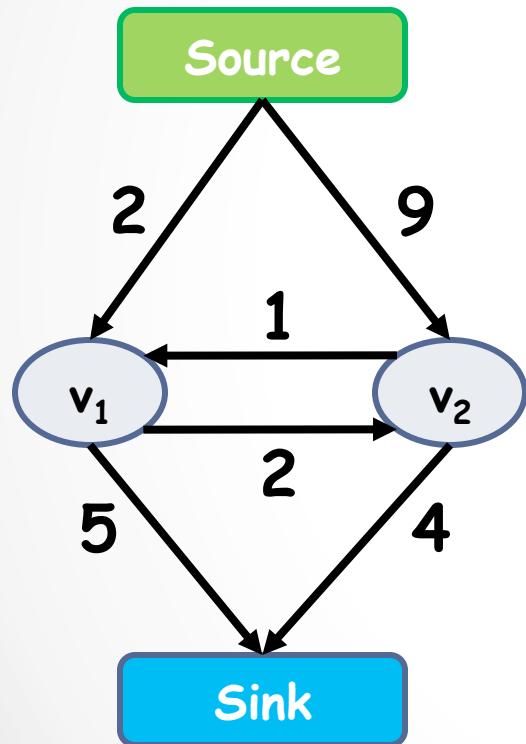
What is the st-mincut?

st-cut with the minimum cost

$$2 + 1 + 4 = 7$$

# How to Compute the st-Mincut?

Solve the dual maximum flow problem  
Compute the maximum flow between  
Source and Sink



## Constraints

Edges: Flow < Capacity

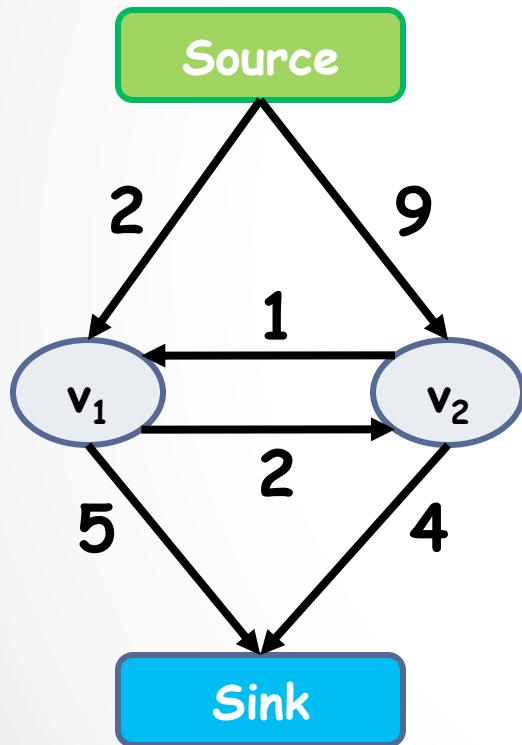
Nodes: Flow in = Flow out

## Min-cut/Maxflow Theorem

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

# Maxflow Algorithms

Flow = 0



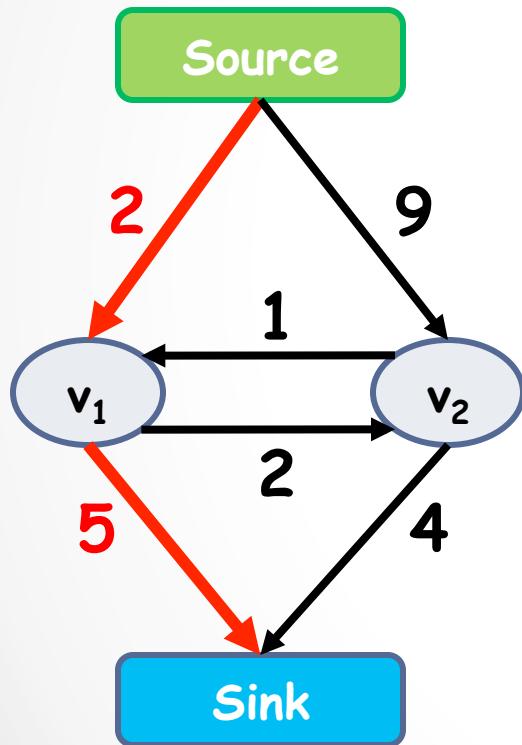
## Augmenting Path Based Algorithms

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

# Maxflow Algorithms

Flow = 0



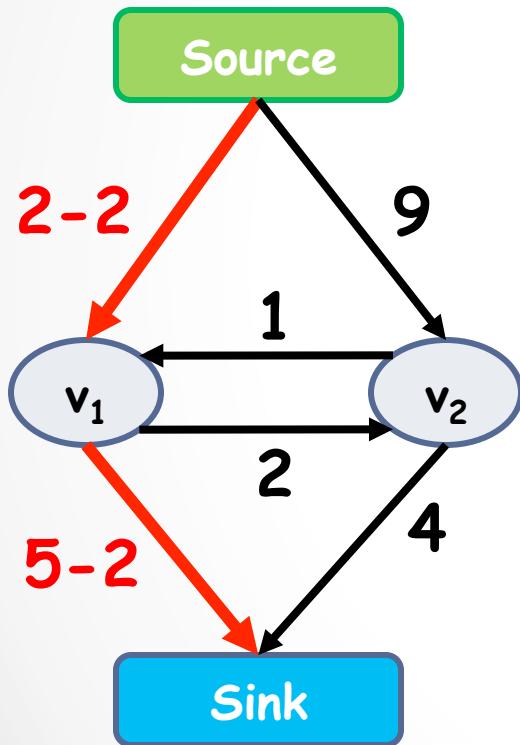
## Augmenting Path Based Algorithms

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

# Maxflow Algorithms

Flow = 0 + 2



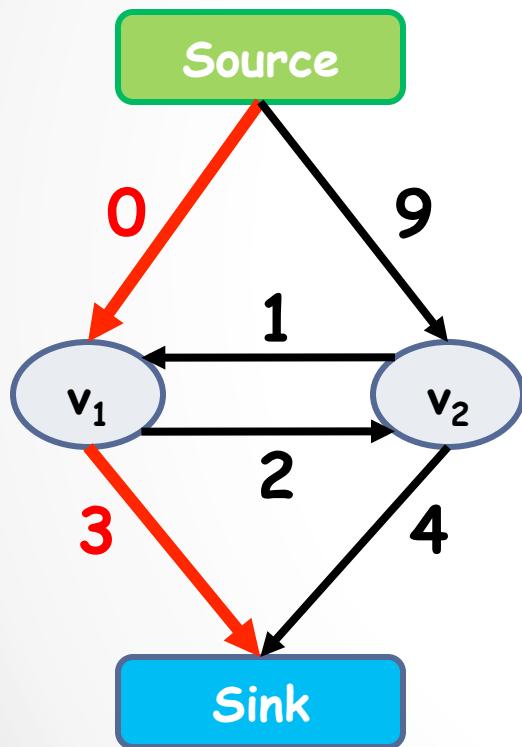
## Augmenting Path Based Algorithms

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

# Maxflow Algorithms

Flow = 2



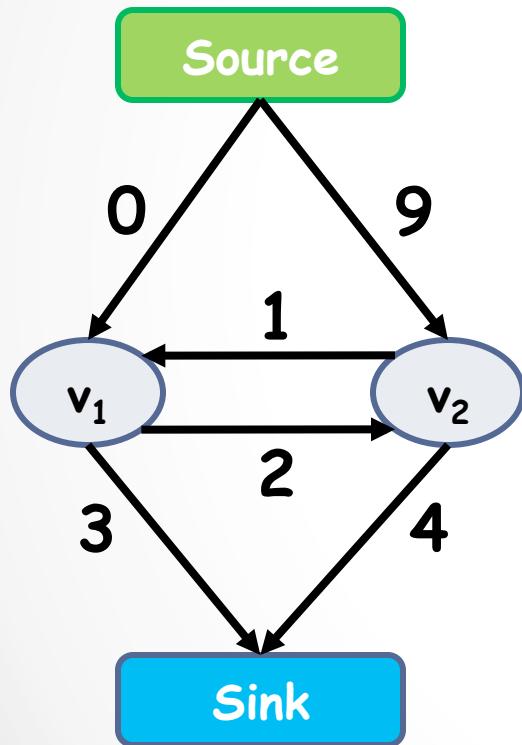
## Augmenting Path Based Algorithms

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

# Maxflow Algorithms

Flow = 2



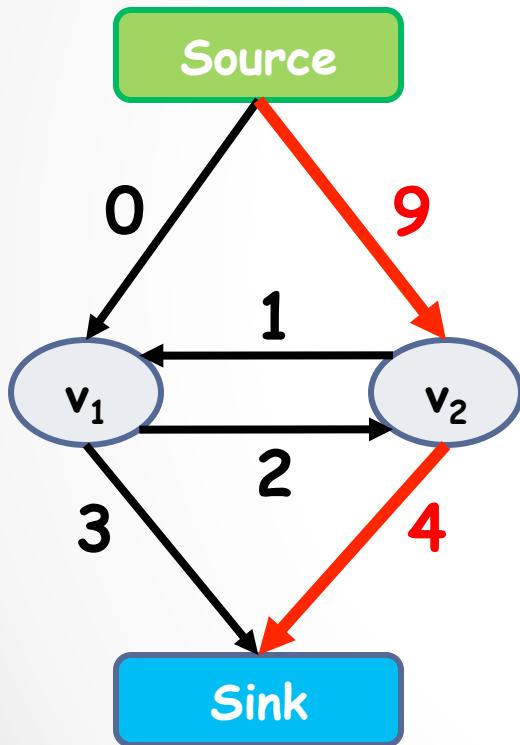
## Augmenting Path Based Algorithms

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

# Maxflow Algorithms

Flow = 2



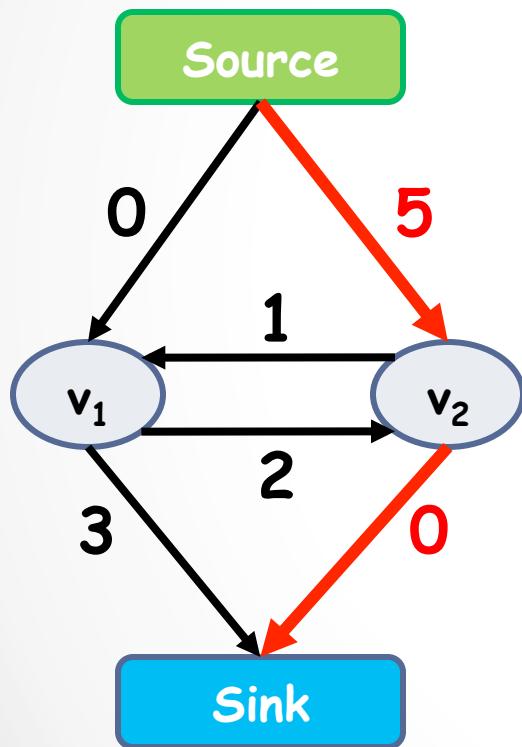
## Augmenting Path Based Algorithms

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

# Maxflow Algorithms

Flow = 2 + 4



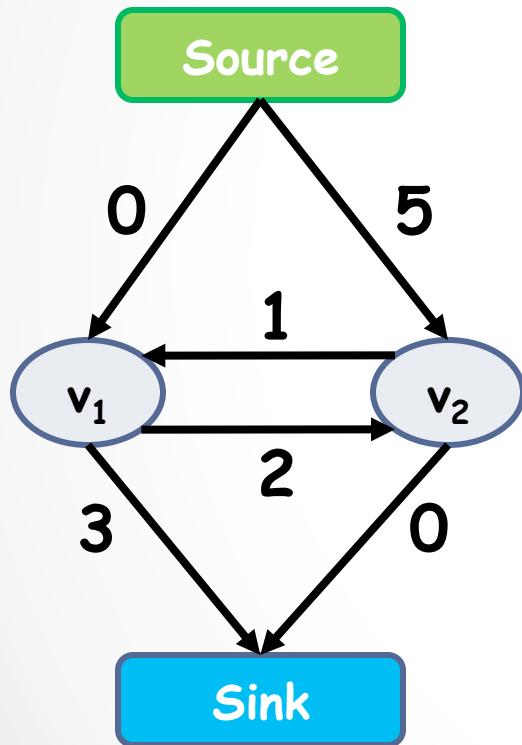
## Augmenting Path Based Algorithms

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

# Maxflow Algorithms

Flow = 6



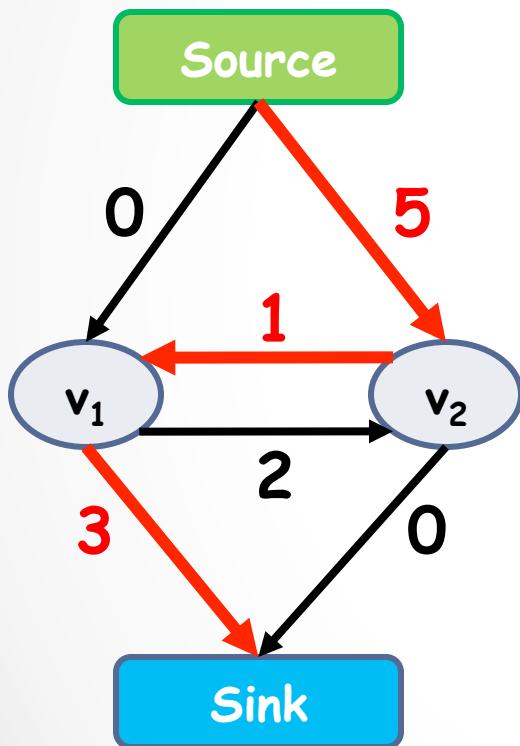
## Augmenting Path Based Algorithms

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

# Maxflow Algorithms

Flow = 6



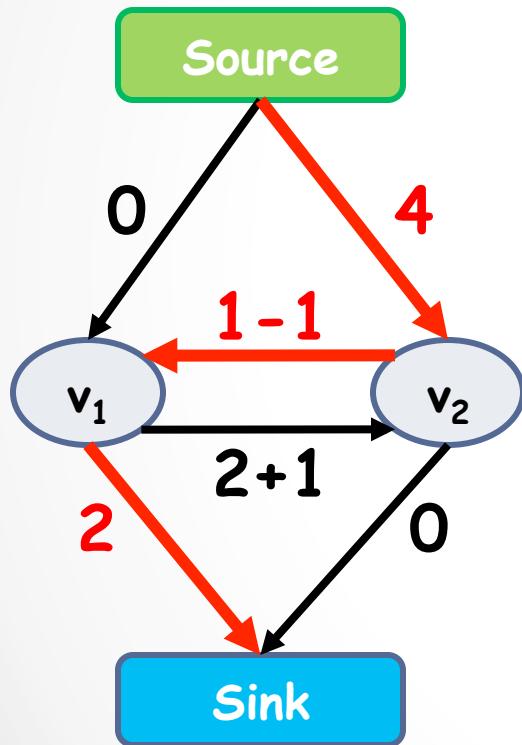
## Augmenting Path Based Algorithms

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

# Maxflow Algorithms

Flow = 6 + 1



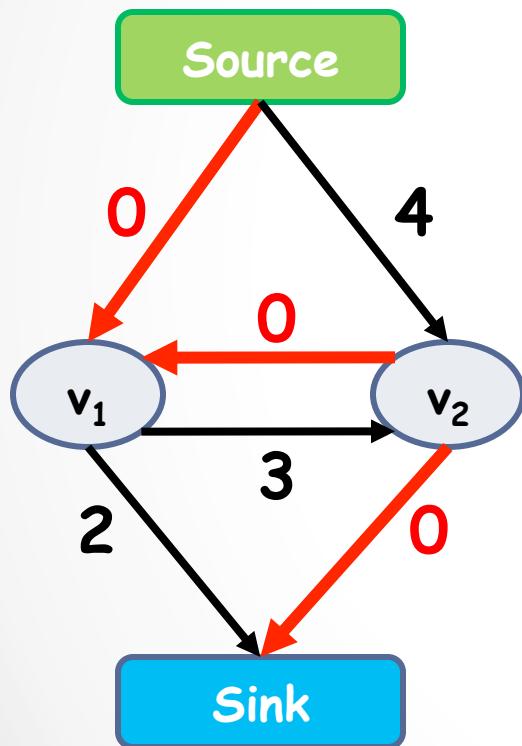
## Augmenting Path Based Algorithms

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

# Maxflow Algorithms

Flow = 7



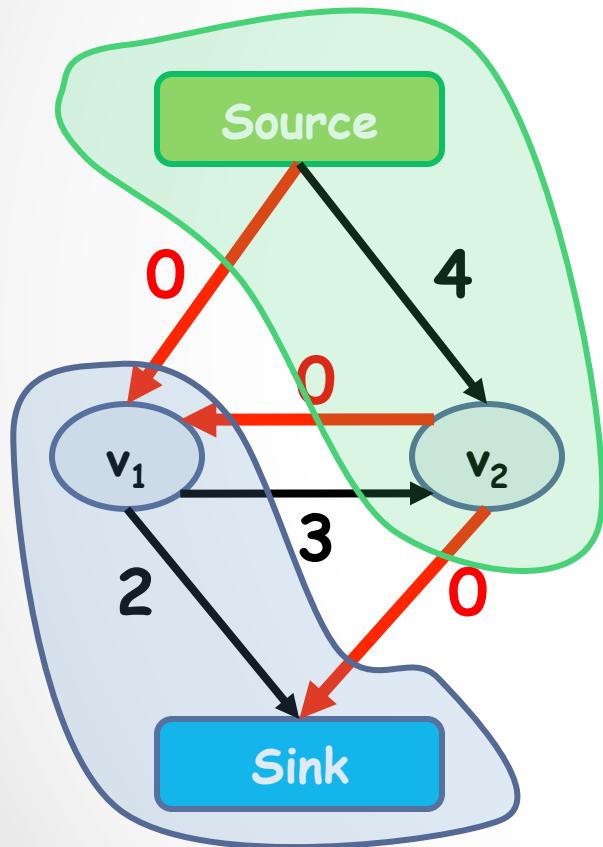
## Augmenting Path Based Algorithms

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

# Maxflow Algorithms

Flow = 7



## Augmenting Path Based Algorithms

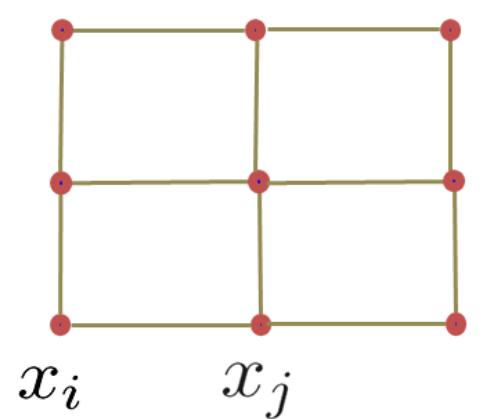
1. Find path from source to sink with positive capacity
2. Push maximum possible flow through this path
3. Repeat until no path can be found

Algorithms assume non-negative capacity

# Maxflow in Computer Vision

- Specialized algorithms for vision problems

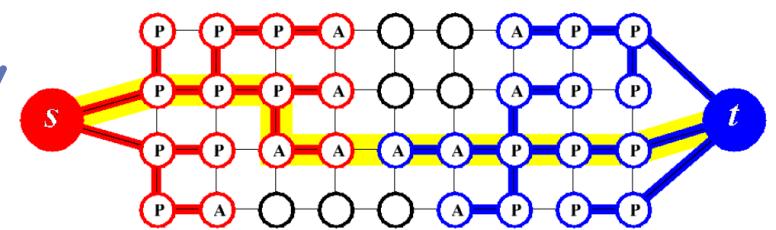
- Grid graphs
- Low connectivity ( $m \sim O(n)$ )



- Dual search tree augmenting path algorithm

[Boykov and Kolmogorov PAMI 2004]

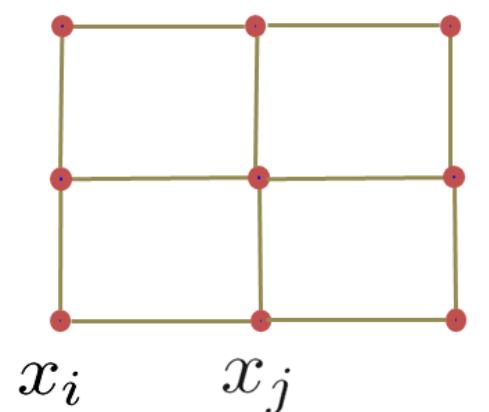
- Finds approximate shortest augmenting paths efficiently
- High worst-case time complexity
- Empirically outperforms other algorithms on vision problems



# Maxflow in Computer Vision

- Specialized algorithms for vision problems

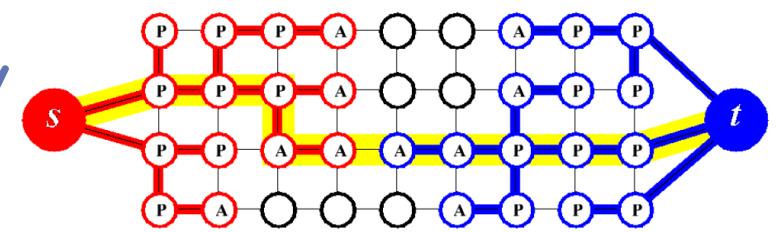
- Grid graphs
- Low connectivity ( $m \sim O(n)$ )



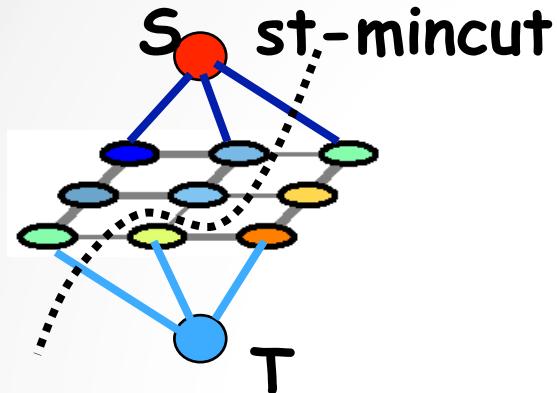
- Dual search tree augmenting path algorithm

[Boykov and Kolmogorov PAMI 2004]

- Finds approximate shortest augmenting paths efficiently
- High worst-case time complexity
- Empirically outperforms other algorithms on vision problems
- Efficient code available on the web  
<http://pub.ist.ac.at/~vnk/software.html>



# St-mincut and Energy Minimization



Minimizing a Quadratic Pseudoboolean function  $E(x)$

Functions of boolean variables

$$E: \{0, 1\}^n \rightarrow \mathbb{R}$$

$$E(x) = \sum_i c_i x_i + \sum_{i,j} c_{ij} x_i (1 - x_j)$$

$$c_{ij} \geq 0$$

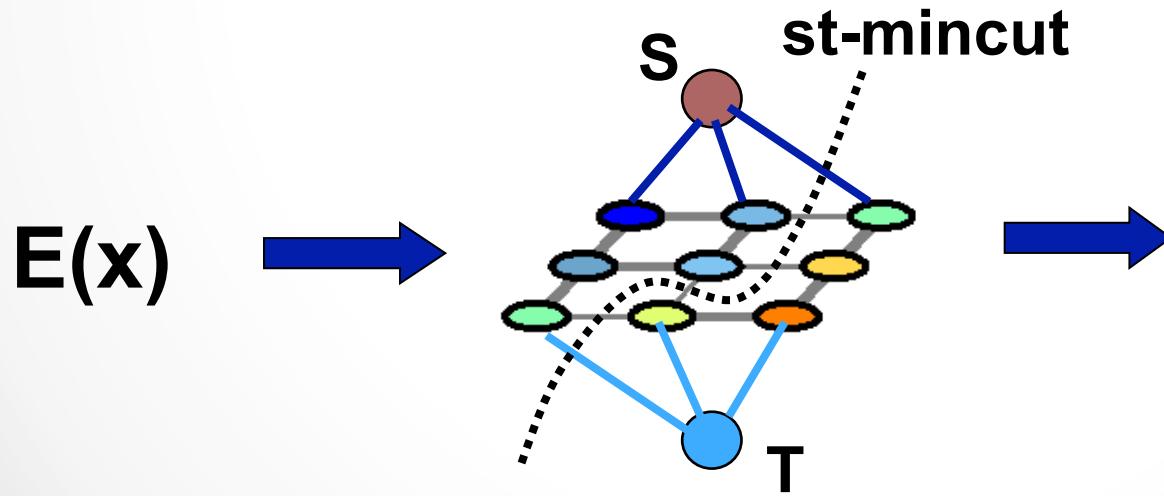


Polynomial time st-mincut algorithms require non-negative edge weights

# How does this Work?

Construct a graph such that:

1. Any st-cut corresponds to an assignment of  $x$
2. The cost of the cut is equal to the energy of  $x : E(x)$



Solution

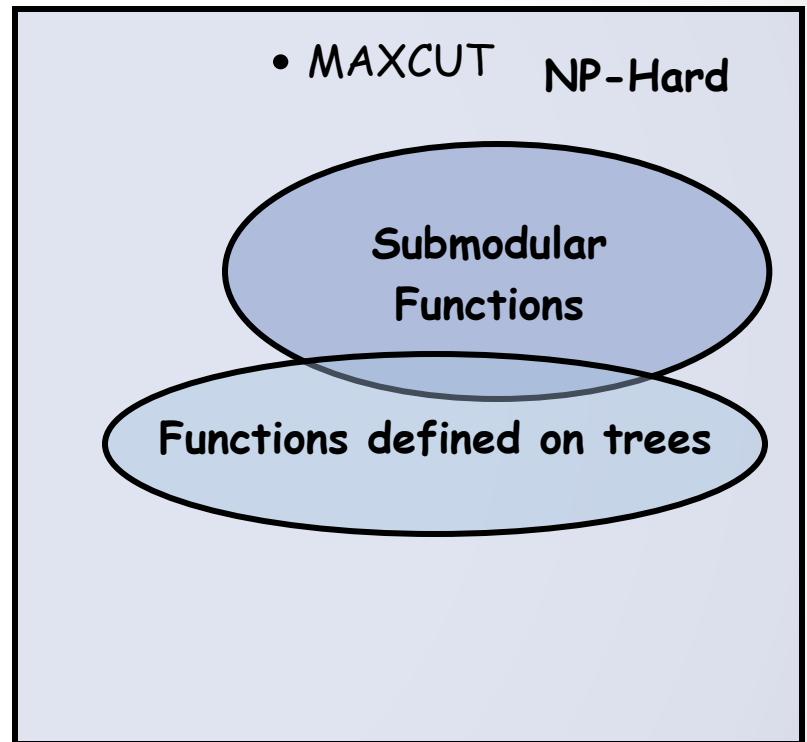
# Minimizing Energy Functions

- **General Energy Functions**

- NP-hard to minimize
- Only approximate minimization possible

- **Easy energy functions**

- Solvable in polynomial time
- Submodular  $\sim O(n^6)$



Space of Function  
Minimization Problems

# GrabCut

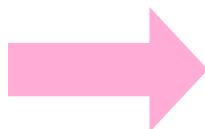
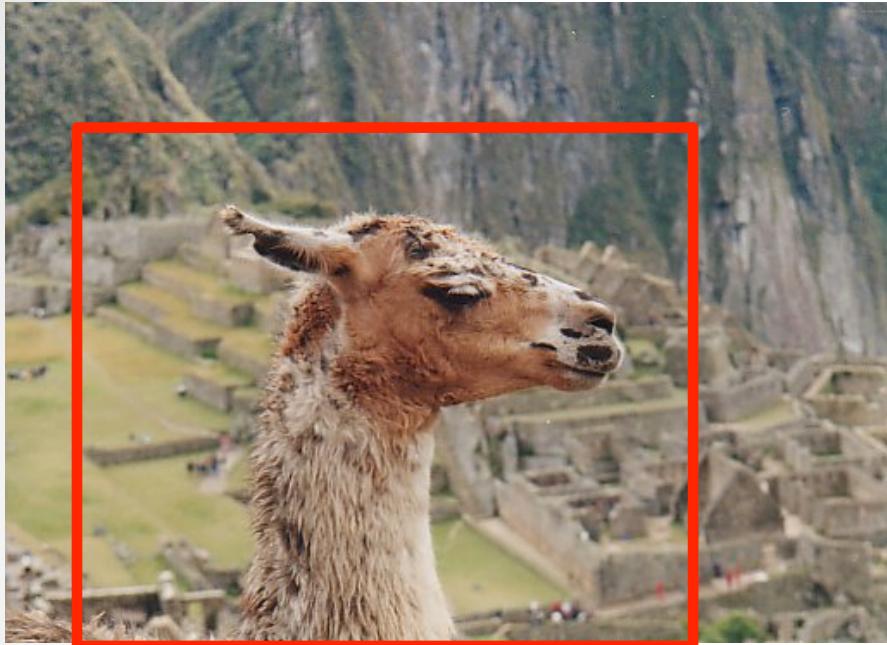
## Interactive Foreground Extraction using Iterated Graph Cuts

Carsten Rother

Vladimir Kolmogorov

Andrew Blake

# The Problem

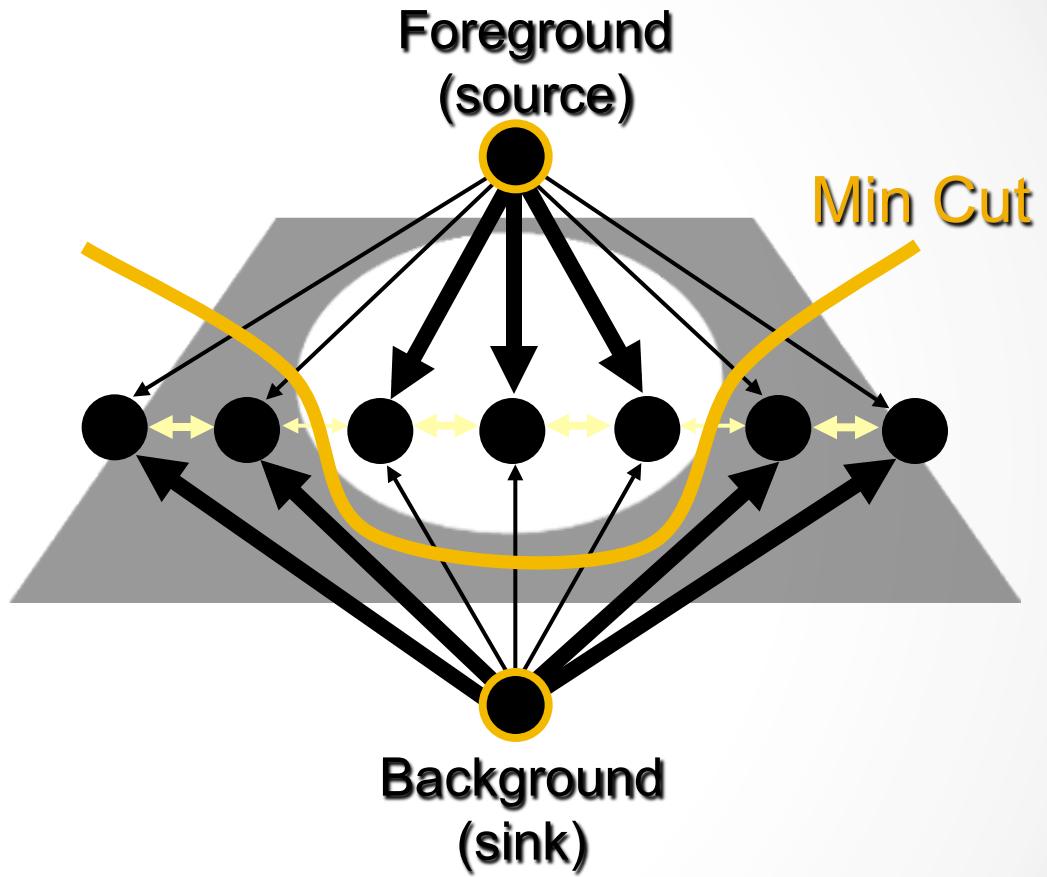
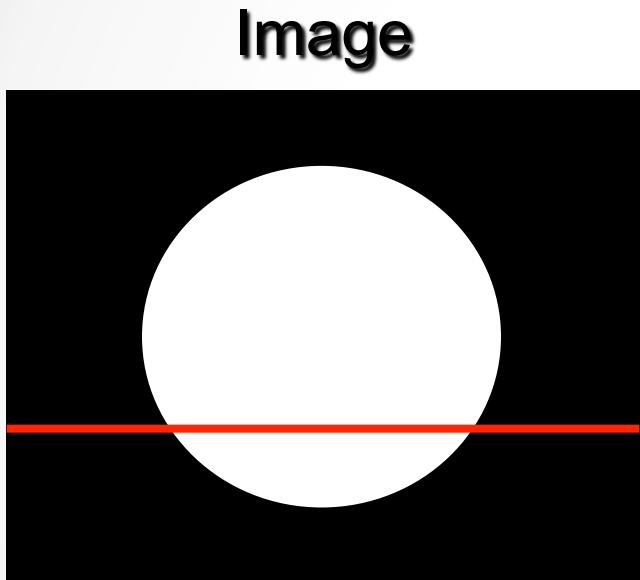


Fast &  
Accurate ?



- Less user input: only rectangle
- Handle color
- Extract matte as post-process

# Use Basic Graph Cut for Segmentation



**Cut:** separating source and sink; Energy: collection of edges

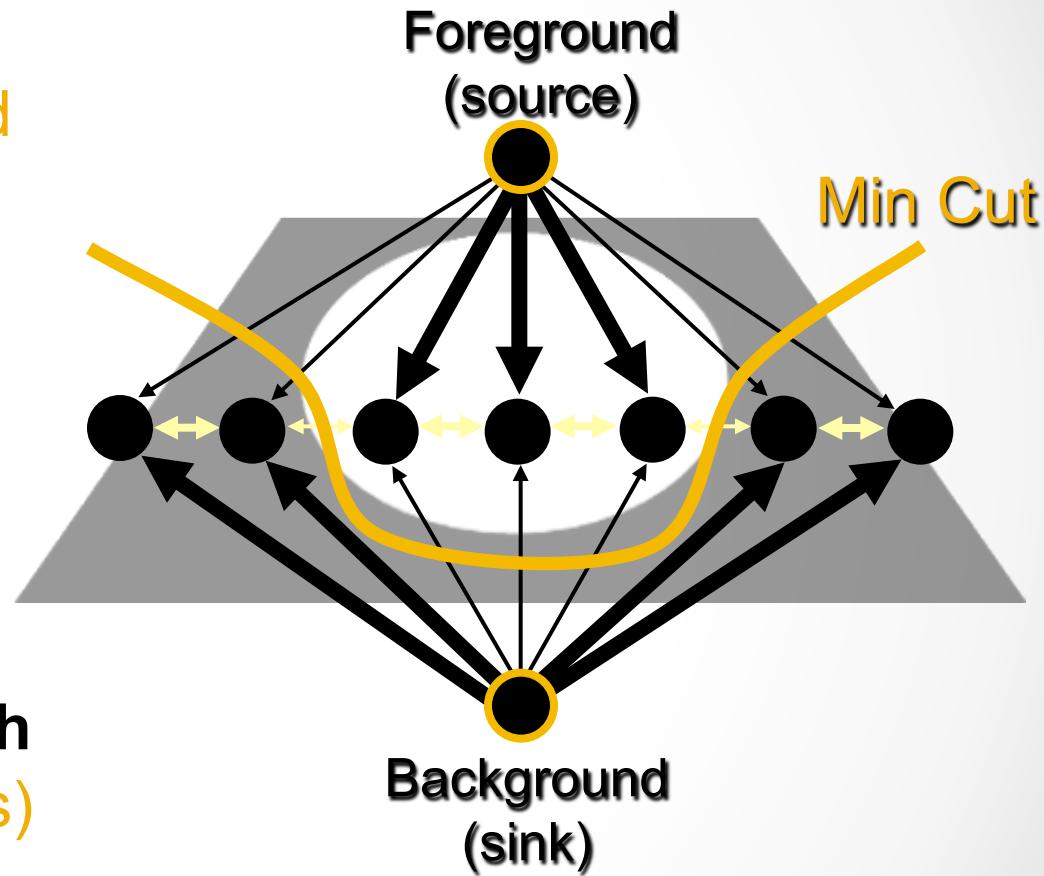
**Min Cut:** Global minimal energy in polynomial time

# Graph Cuts for Foreground Extraction

Assume we know foreground is **white** and background is **black**

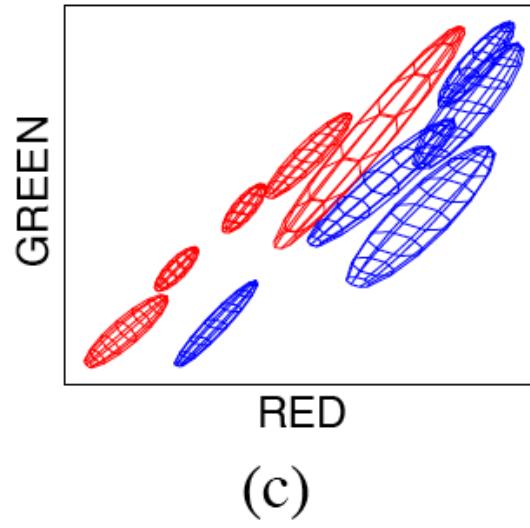
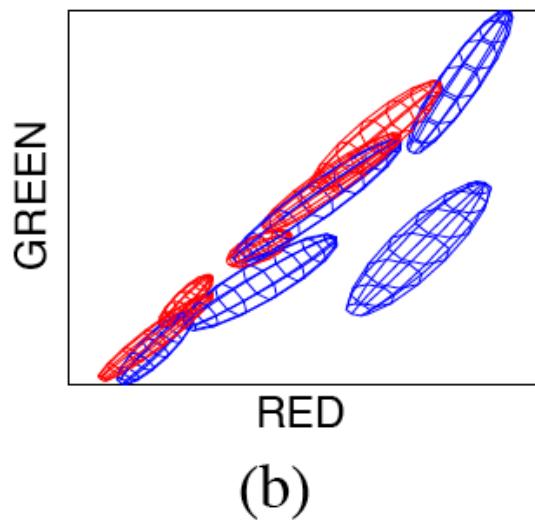
Data term = **whiteness**  
(cost of assigning label)

Regularization = **color match**  
(cost of separating neighbors)



# Color Data Term

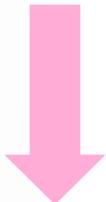
- Model 3D color histogram with Gaussians
  - Because brute force histogram would be sparse
  - Gaussian Mixture Model (GMM)
  - Just means histogram = sum of Gaussians
    - They advise 5 Gaussians



# Iterated Graph Cuts



User Initialisation



Learn foreground  
color model



Graph cuts to  
infer the  
foreground

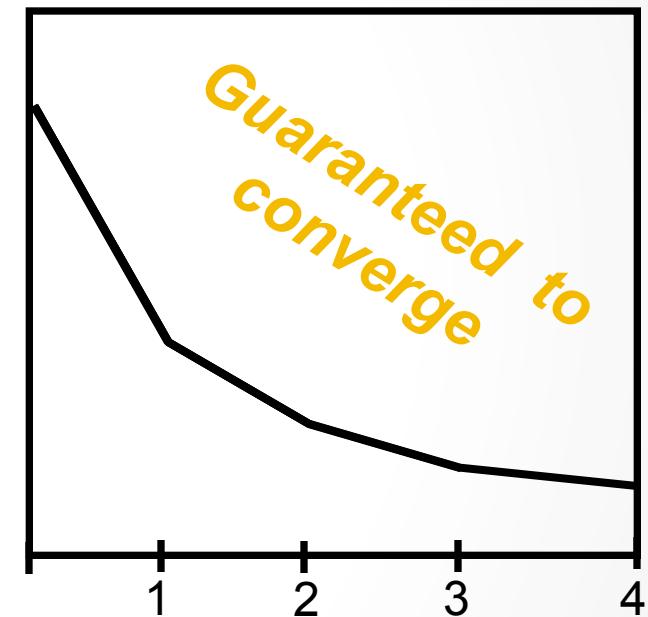
# Grabcut: Iterative approach

- Initialize
  - Background with rectangle boundary pixels
  - Foreground with the interior of rectangle
- Iterate until convergence
  - Compute color probabilities (GMM) of each region
  - Perform graphcut segmentation
- Apply matting at boundary
- Potentially, user edits to correct mistakes

# Iterated Graph Cuts

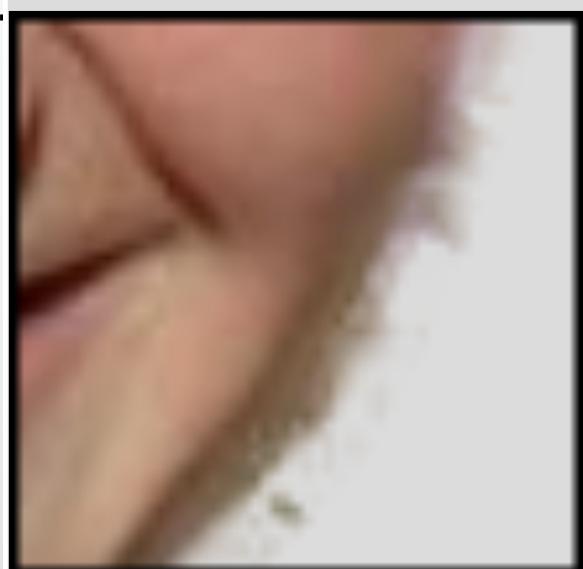
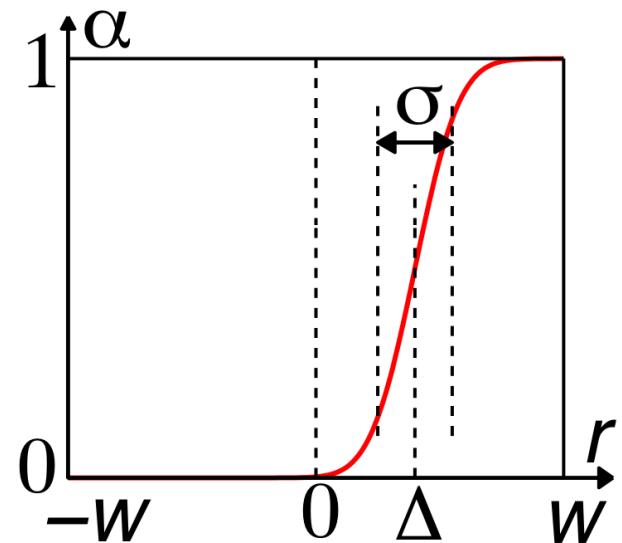
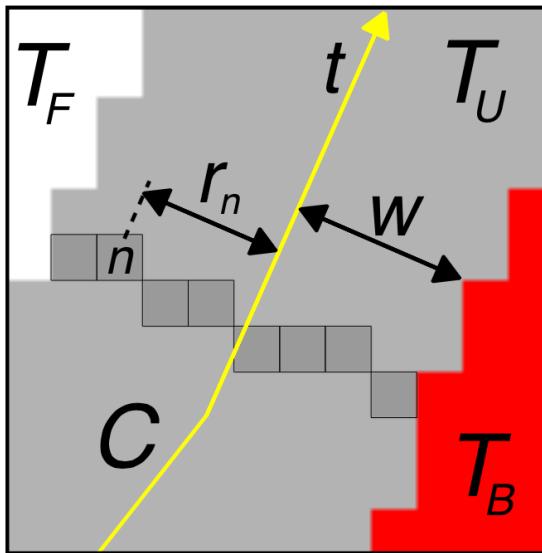
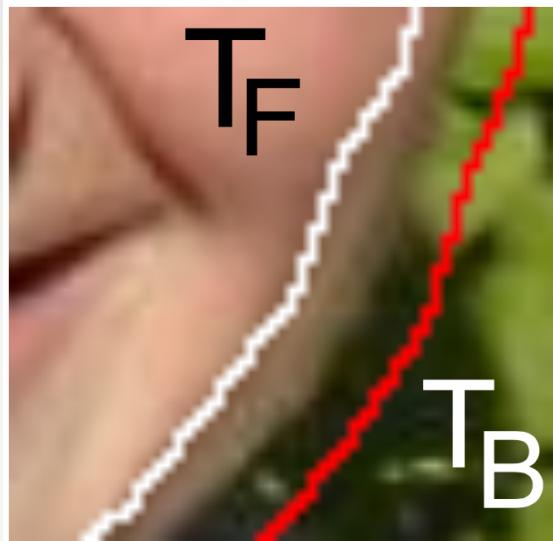


Result

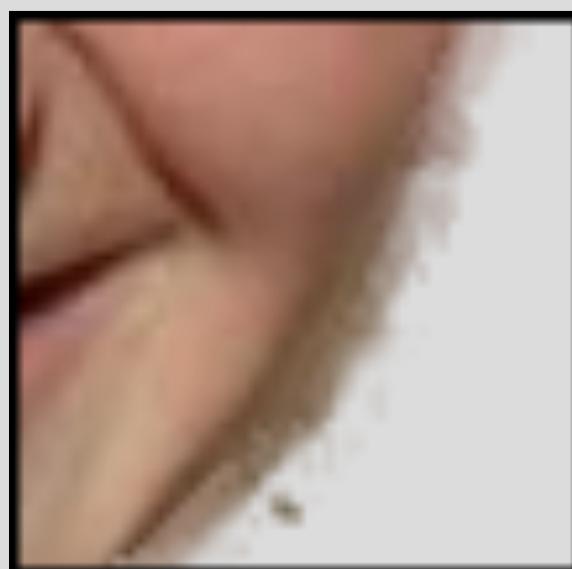


Energy after each Iteration

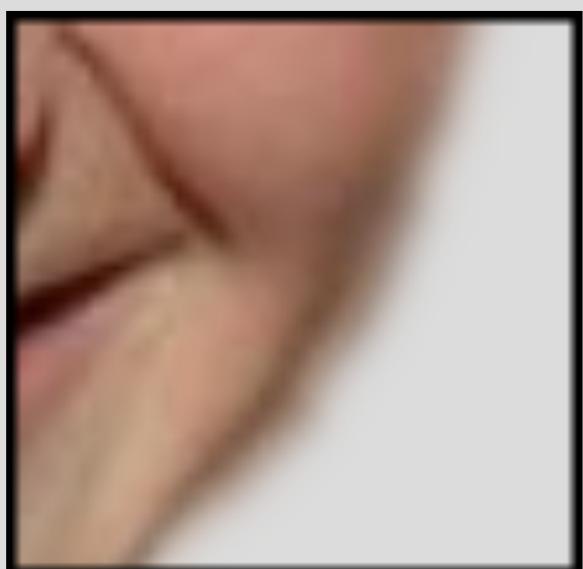
# Border Matting



Knockout 2

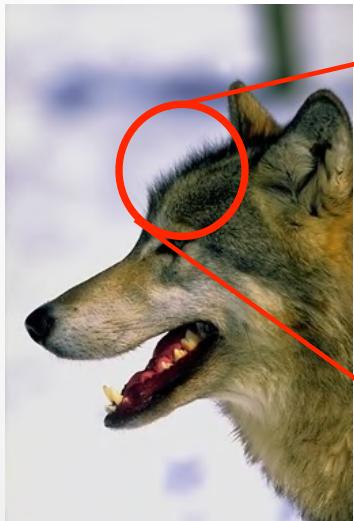


Bayes Matte



GrabCut

# Border Matting Results



# Examples: Moderate Complexity



... GrabCut completes automatically

# User Interaction



Automatic



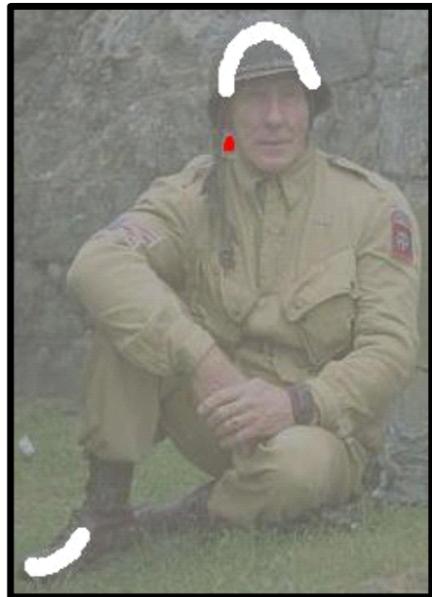
Segmentation



User



Interaction



Automatic



Segmentation



# Examples: Difficult Ones

Camouflage &  
Low Contrast

Initial  
Rectangle



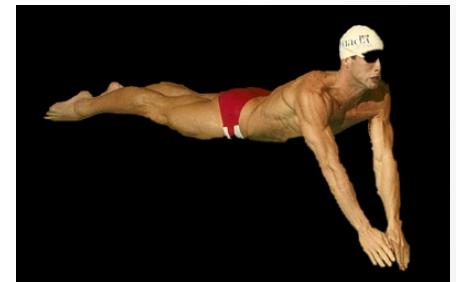
Initial  
Result



Fine structure



No telepathy



# GraphCut vs. GrabCut

Boykov and Jolly (2001)

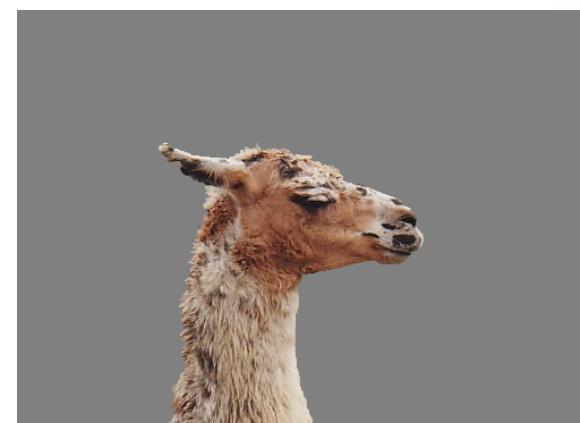
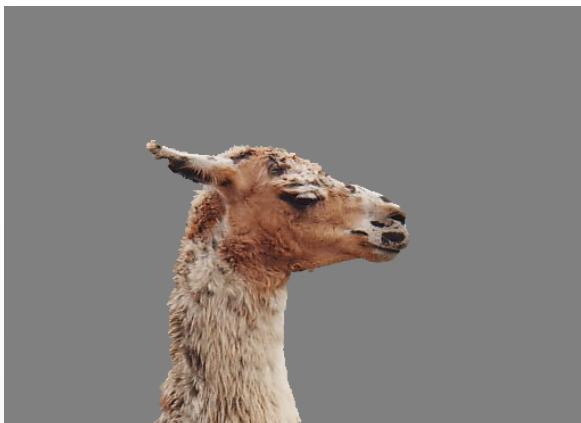
User  
Input



GrabCut



Result



Error Rate: 0.72%

Error Rate: 0.72%