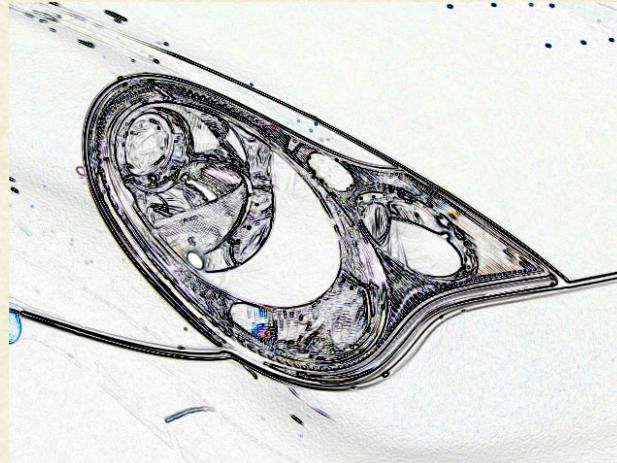




CS7.505: Computer Vision

Spring 2022: Segmentation using MRF

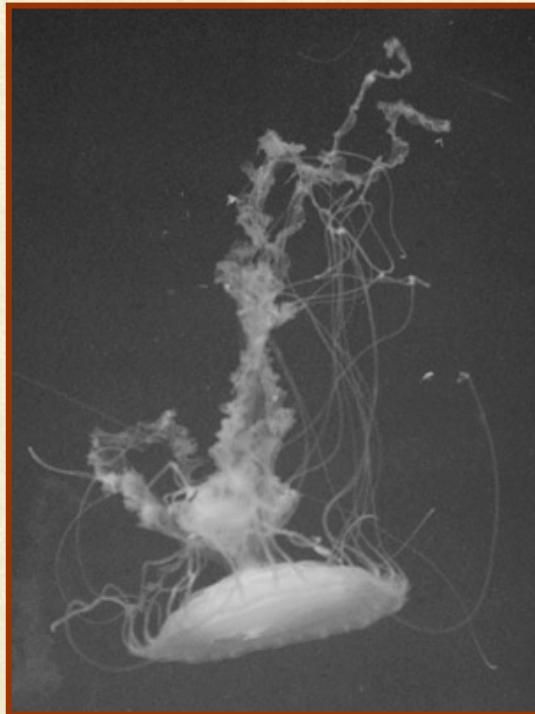


Anoop M. Namboodiri
Biometrics and Secure ID Lab, CVIT,
IIIT Hyderabad

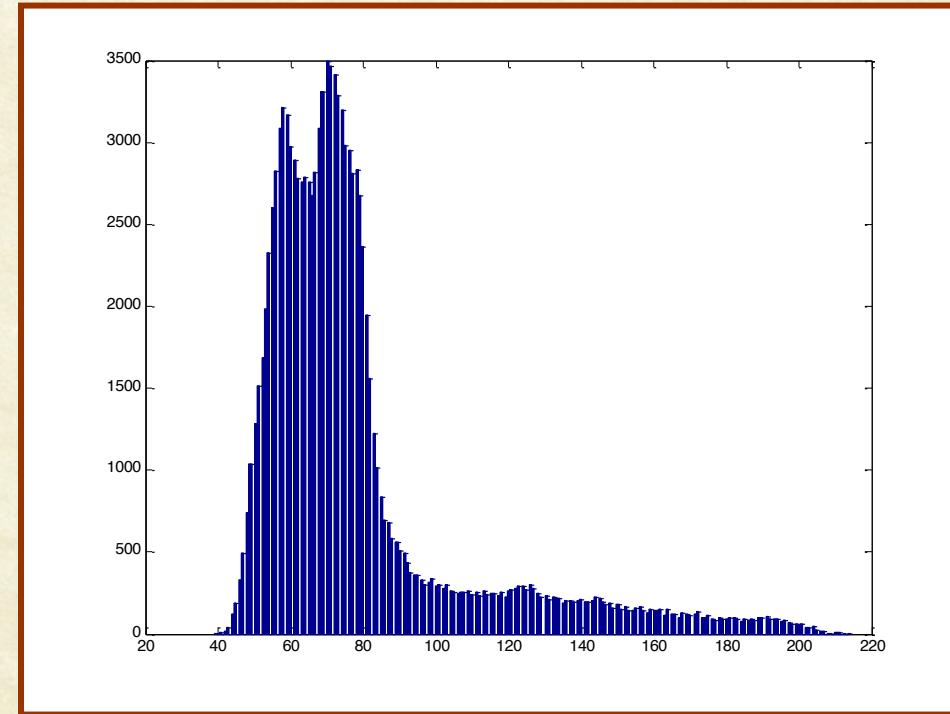


Optimal Thresholding

- 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

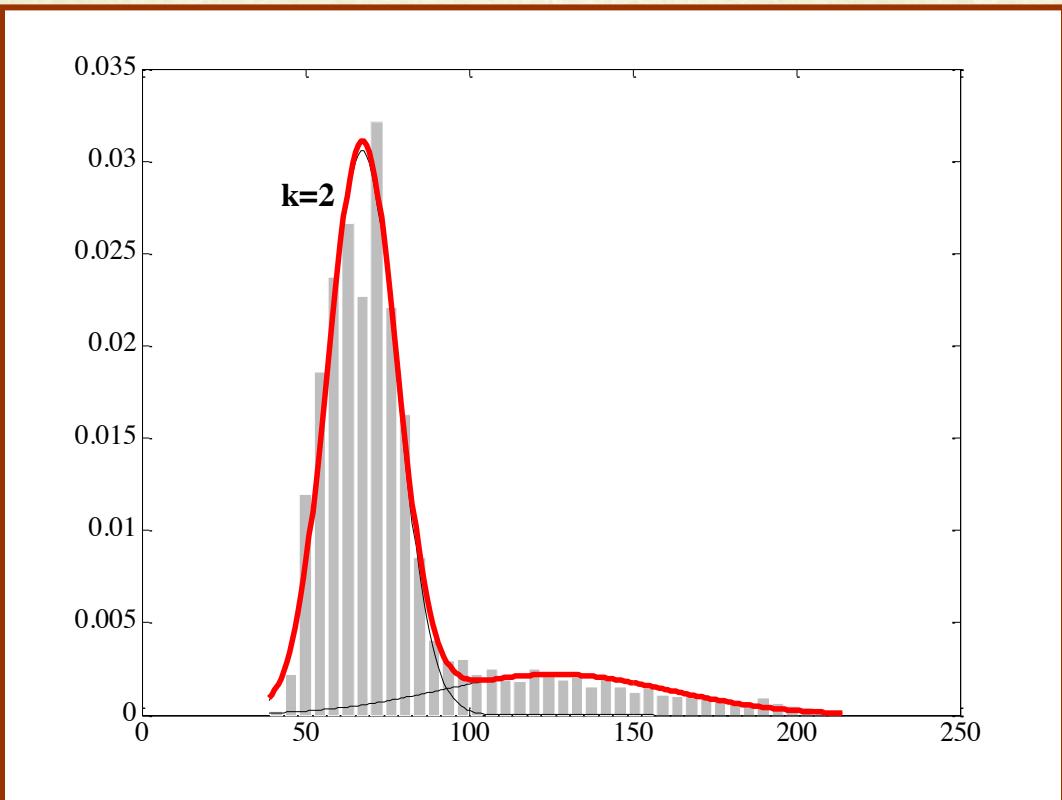
- Obj: $N(\mu_1, \sigma_1) = \frac{1}{\sigma_1 \sqrt{2\pi}} e^{-\frac{(x-\mu_1)^2}{2\sigma_1^2}}$

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

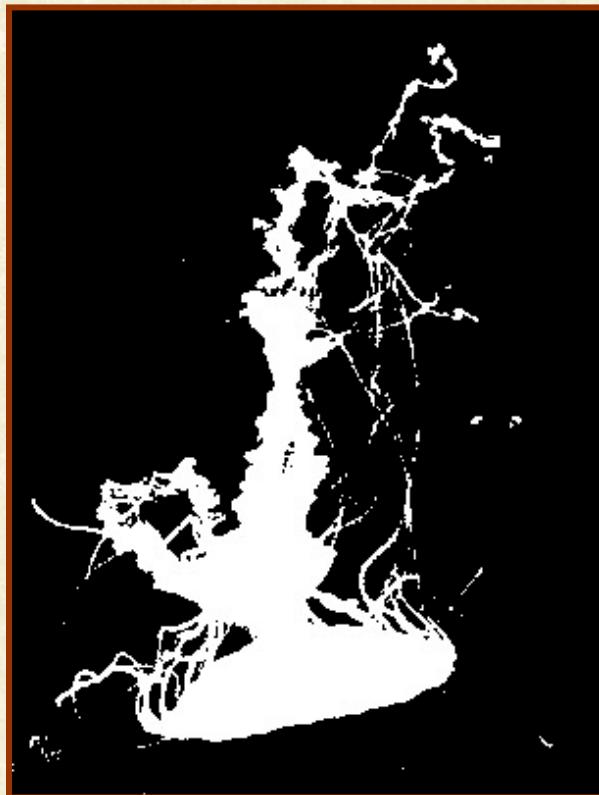
- Initialize μ_1, σ_1, μ_2 , and σ_2 .
- E-Step: Computed the expected pixel label assignments.
This could be either hard or soft assignment.
- M-Step: Computed Maximum-(Log)Likelihood estimates of the parameters:
 $\mu_1, \sigma_1, \mu_2, \sigma_2$
- Repeat the E and M steps until convergence



Optimal Thresholding



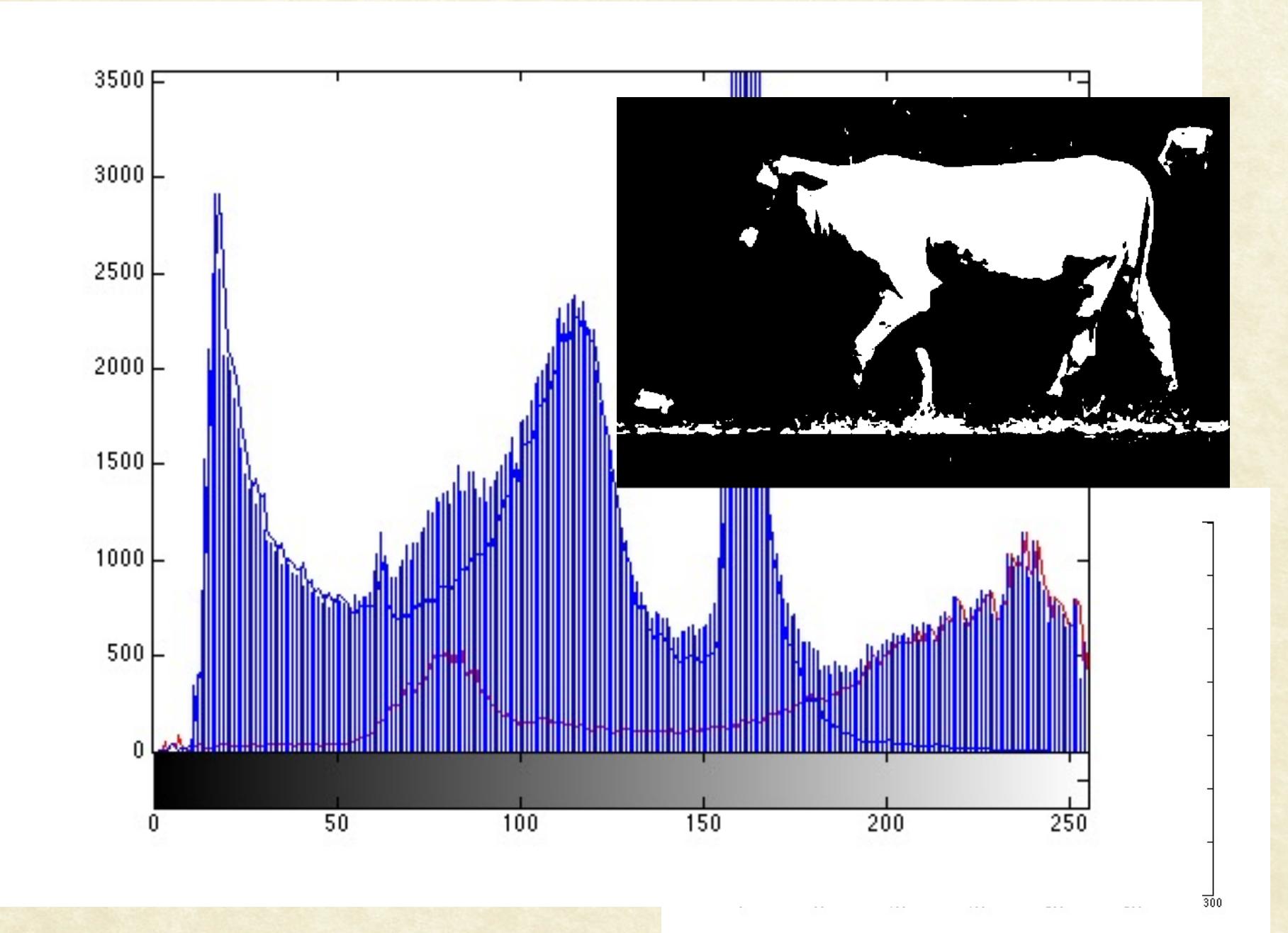
Histogram with bimodal fit



Thresholded ($T=94$)



Is It





Segmentation by Pixel Classification

Two Primary Challenges:

1. How to use object / background properties to decide on pixel label?
 - e.g., Ducks are white and yellow, while background is green and brown
2. How to ensure that regions are continuous regions?
 - Avoid fragmentation of object regions



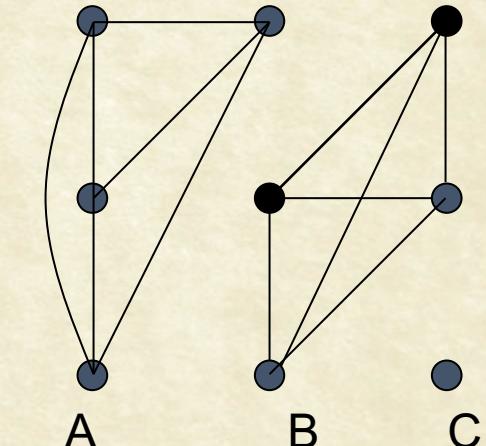
Segmentation as Optimal Labeling

- 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 $Q(f)$.



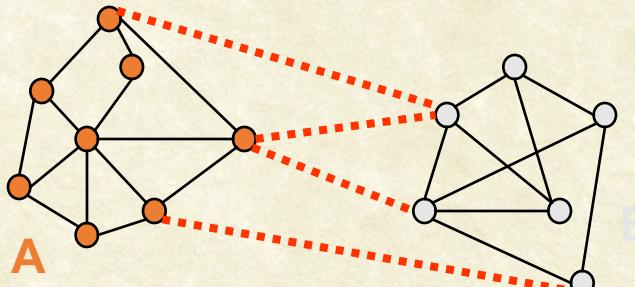
Segmentation by Graph Cuts

- 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

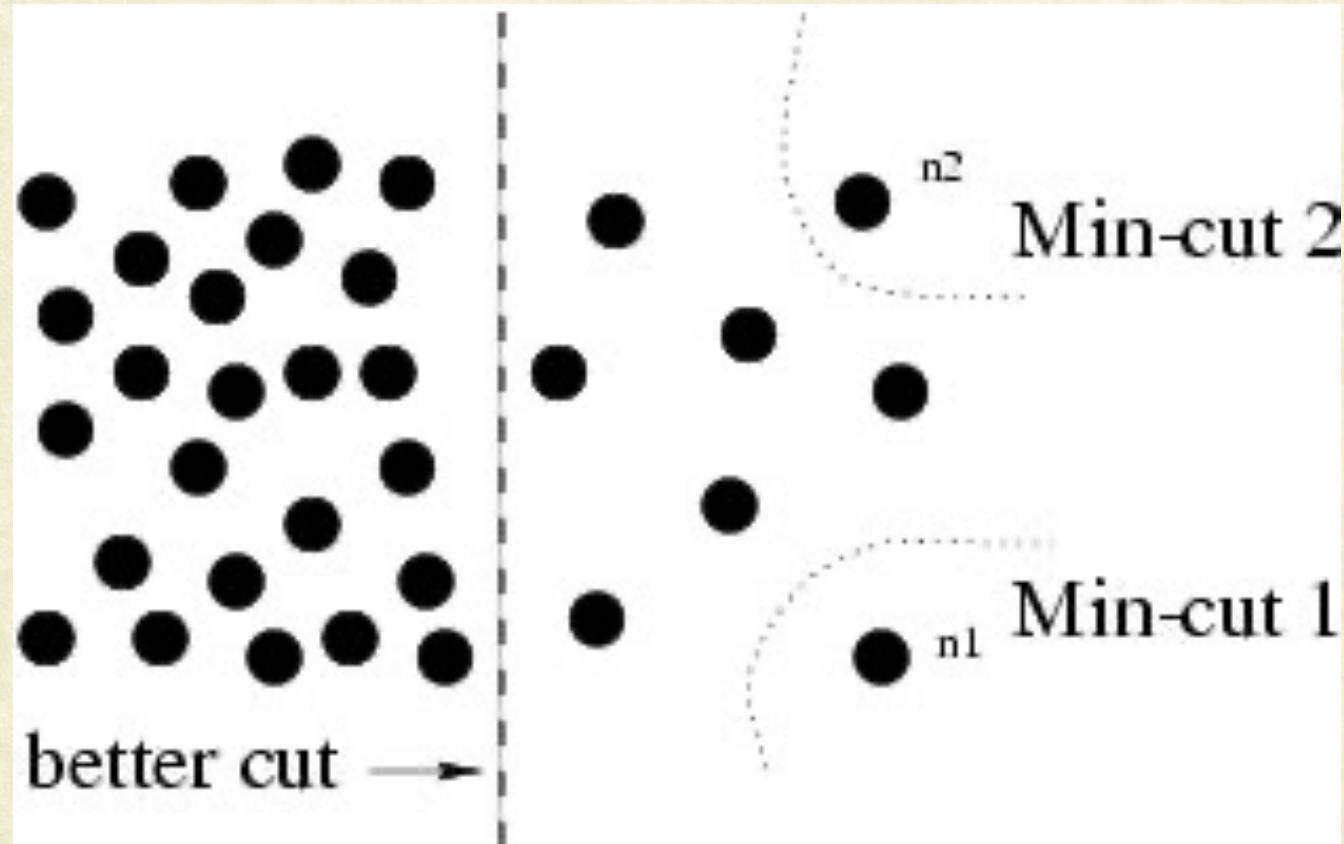


- Link Cut
 - Set of links whose removal makes a graph disconnected
 - Cost of a cut:
- One idea: Find minimum cut
 - Gives you a segmentation
 - Fast algorithms exist for doing this

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

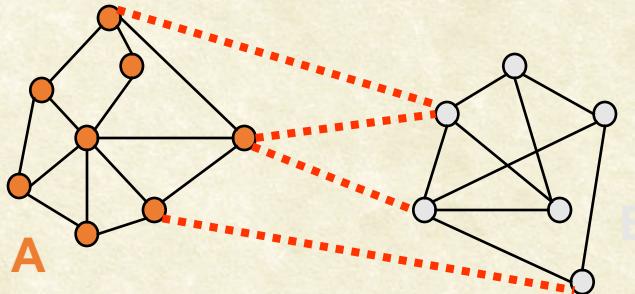


Min cut is not always the best





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



Normalized cuts results





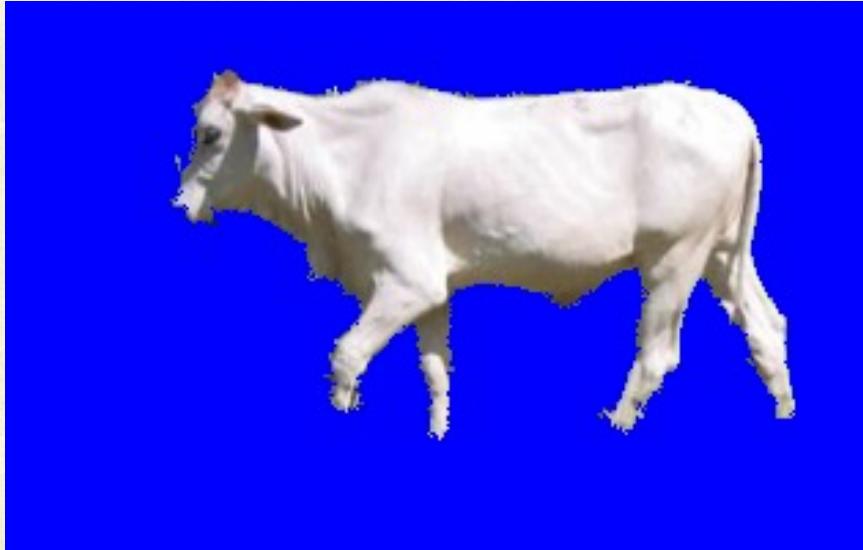
Segmentation by Pixel Classification

Two Primary Challenges:

1. How to use object / background properties to decide on pixel label?
 - e.g., Ducks are white and yellow, while background is green and brown
2. How to ensure that regions are continuous regions?
 - Avoid fragmentation of object regions
3. How to integrate both image and regions properties?



Graphcut for Image Segmentation



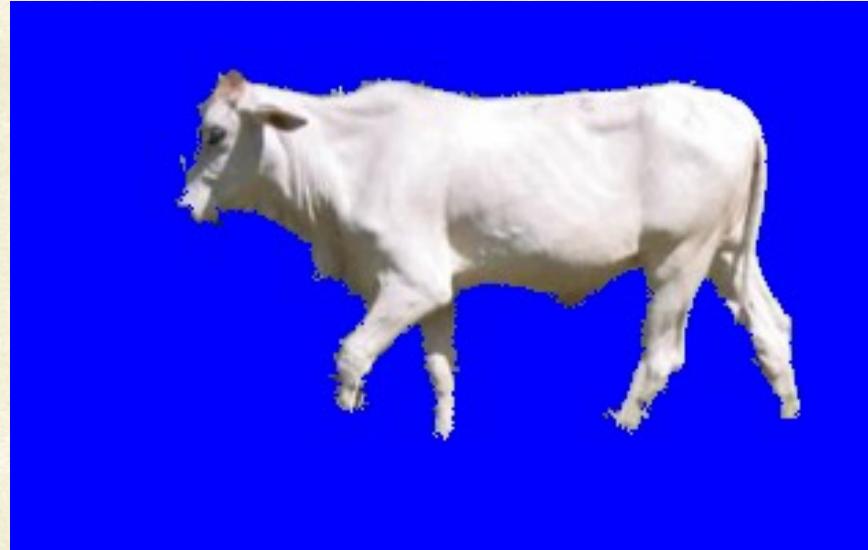
How ?

Cost function Models *our* knowledge about natural images

Optimize cost function to obtain the segmentation



Binary Image Segmentation



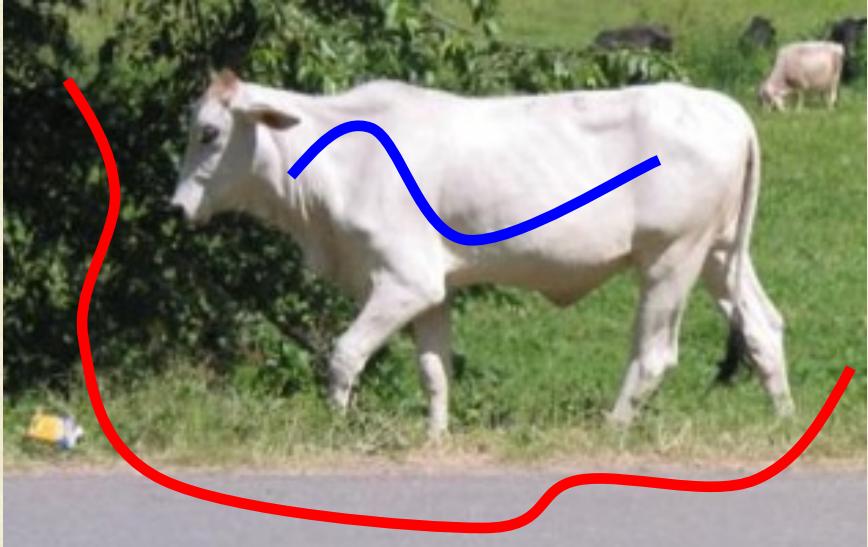
How ?

Cost function Models *our* knowledge about natural images

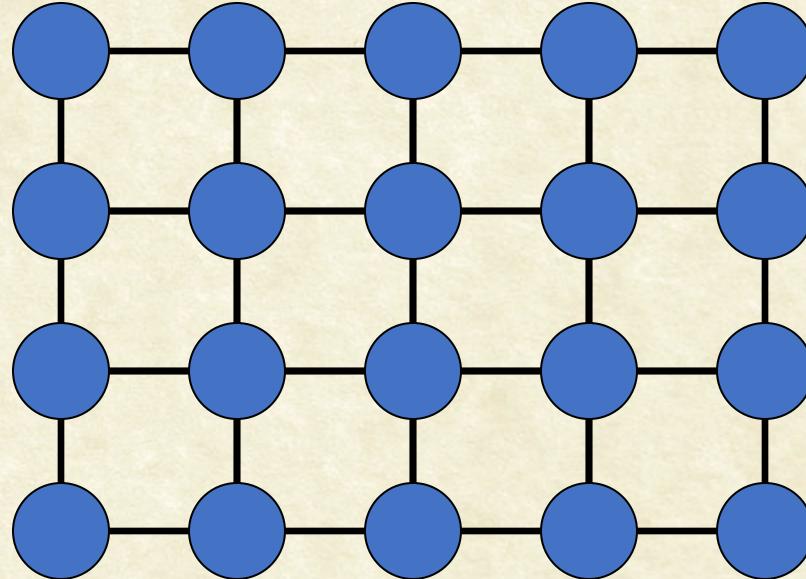
Optimize cost function to obtain the segmentation



Binary Image Segmentation



Object - white, Background - green/grey



Graph $G = (V, E)$

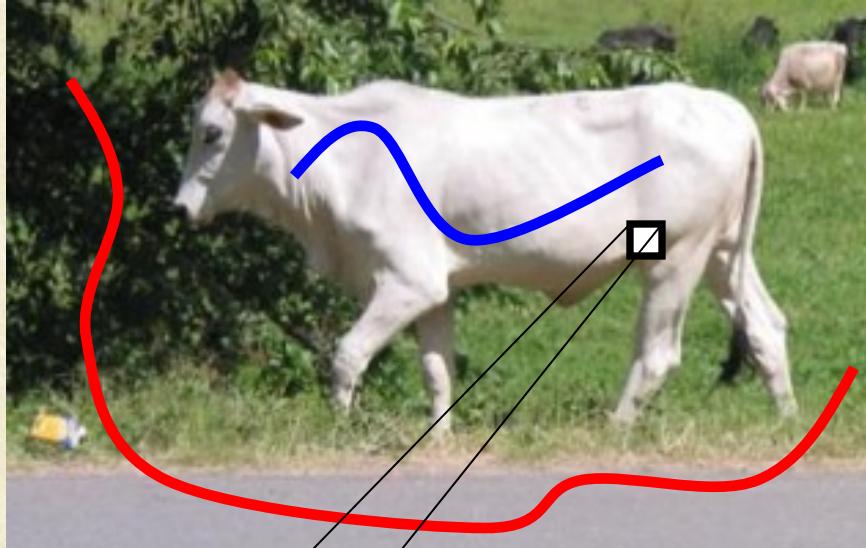
Each vertex corresponds to a pixel

Edges define a 4-neighbourhood *grid* graph

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

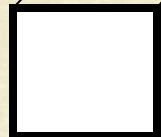


Binary Image Segmentation

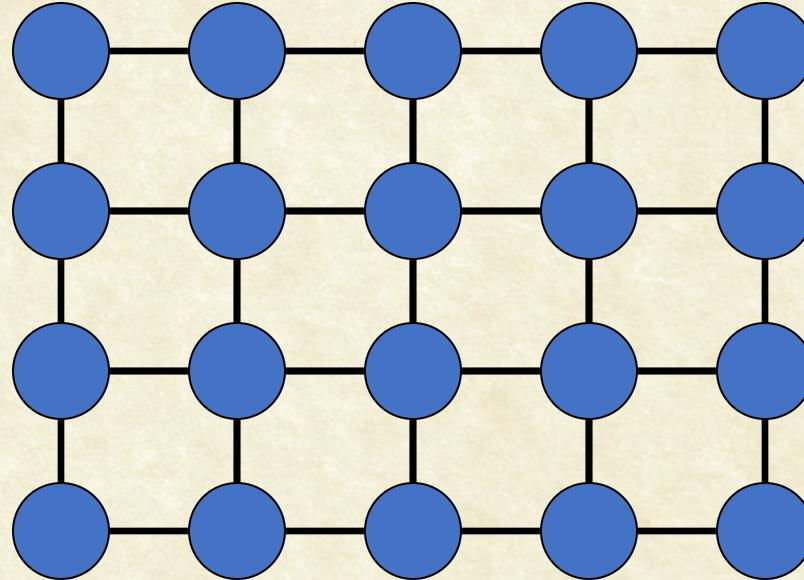


Object - white, Background - green/grey

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



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

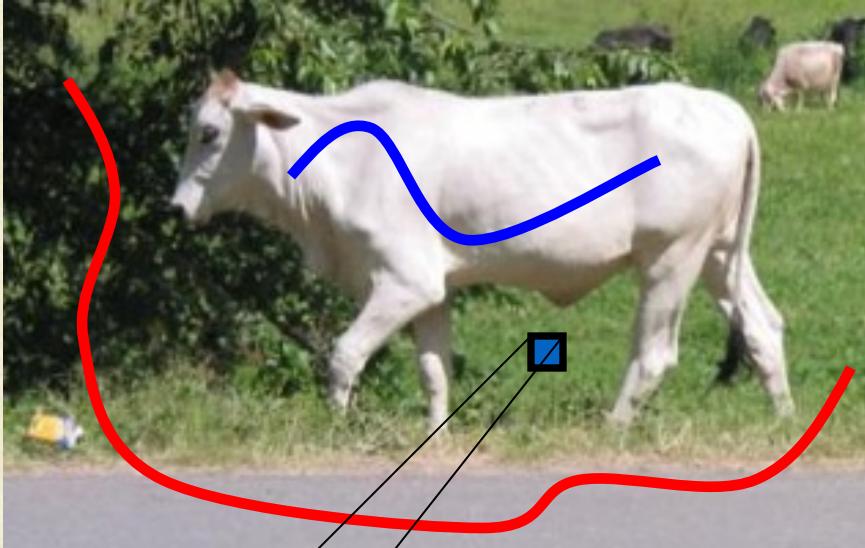


Graph $G = (V, E)$

Per Vertex Cost



Binary Image Segmentation



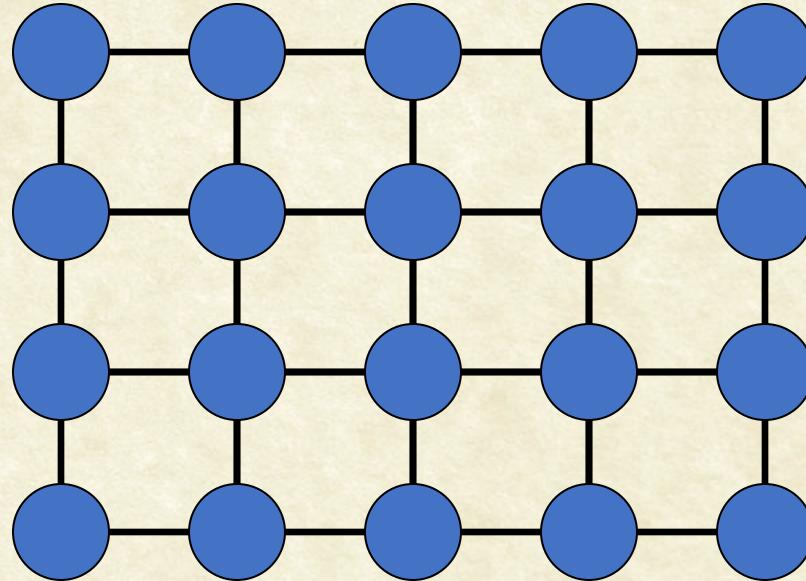
Object - white, Background - green/grey

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



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

UNARY COST

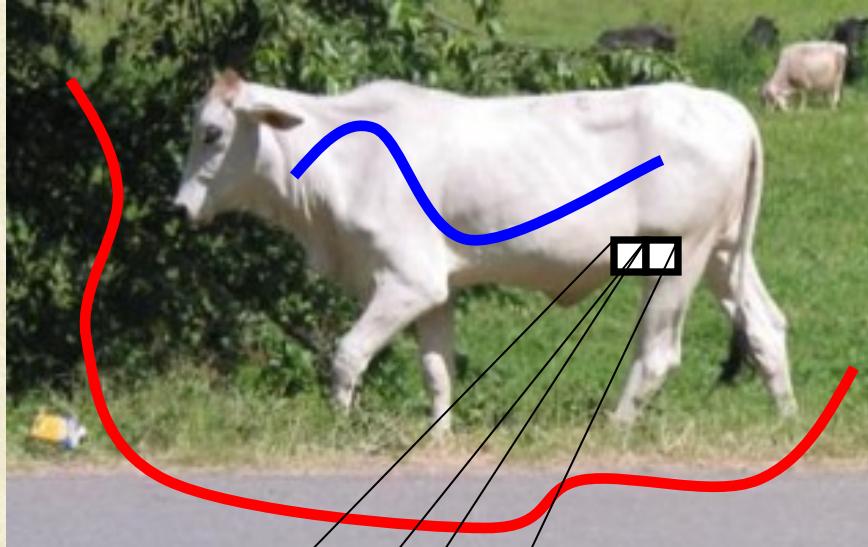


Graph $G = (V, E)$

Per Vertex Cost



Binary Image Segmentation



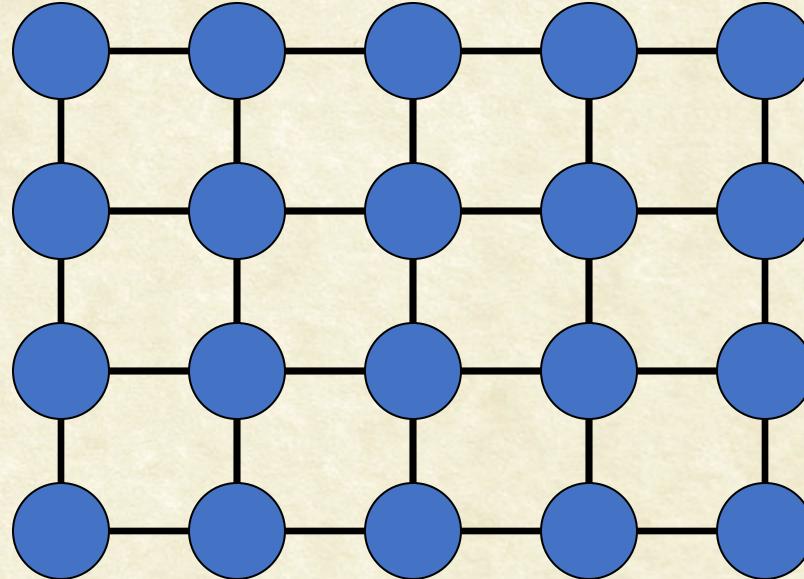
Object - white, Background - green/grey

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



Cost of same label low

Cost of different labels high

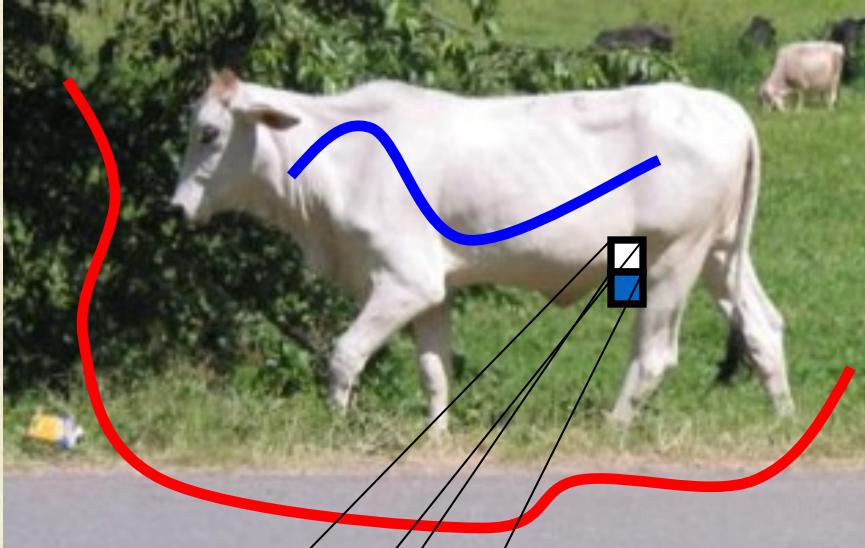


Graph $G = (V, E)$

Per Edge Cost

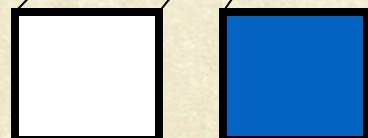


Binary Image Segmentation



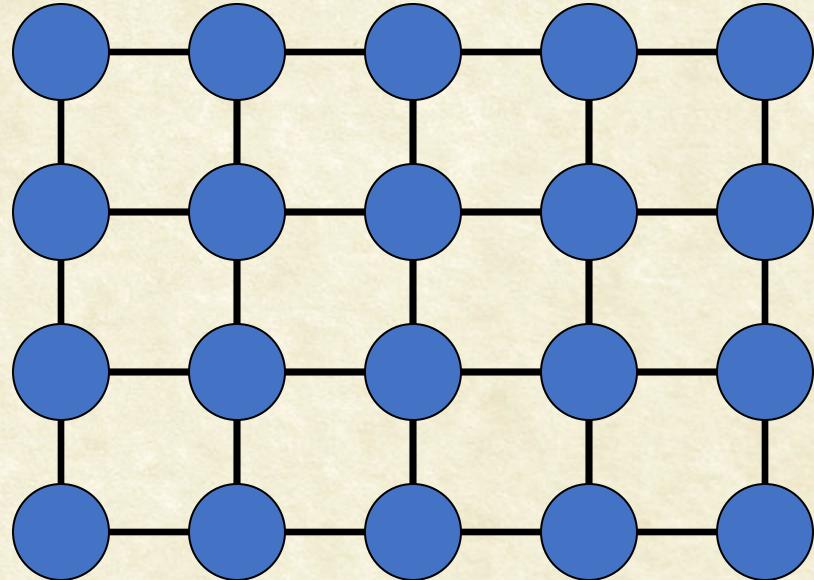
Object - white, Background - green/grey

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



Cost of same label high

Cost of different labels low



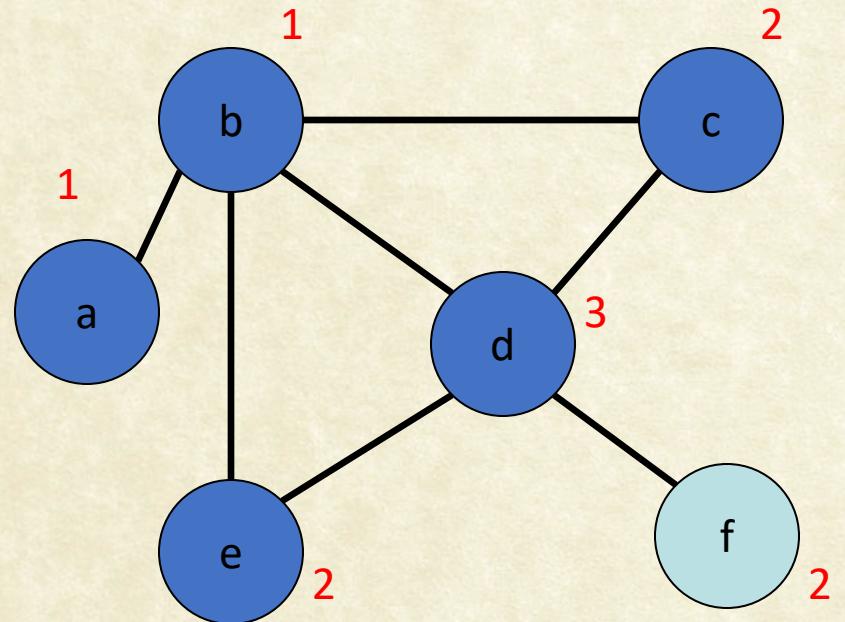
Graph $G = (V, E)$

Per Edge Cost

PAIRWISE
COST



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 $Q(f)$

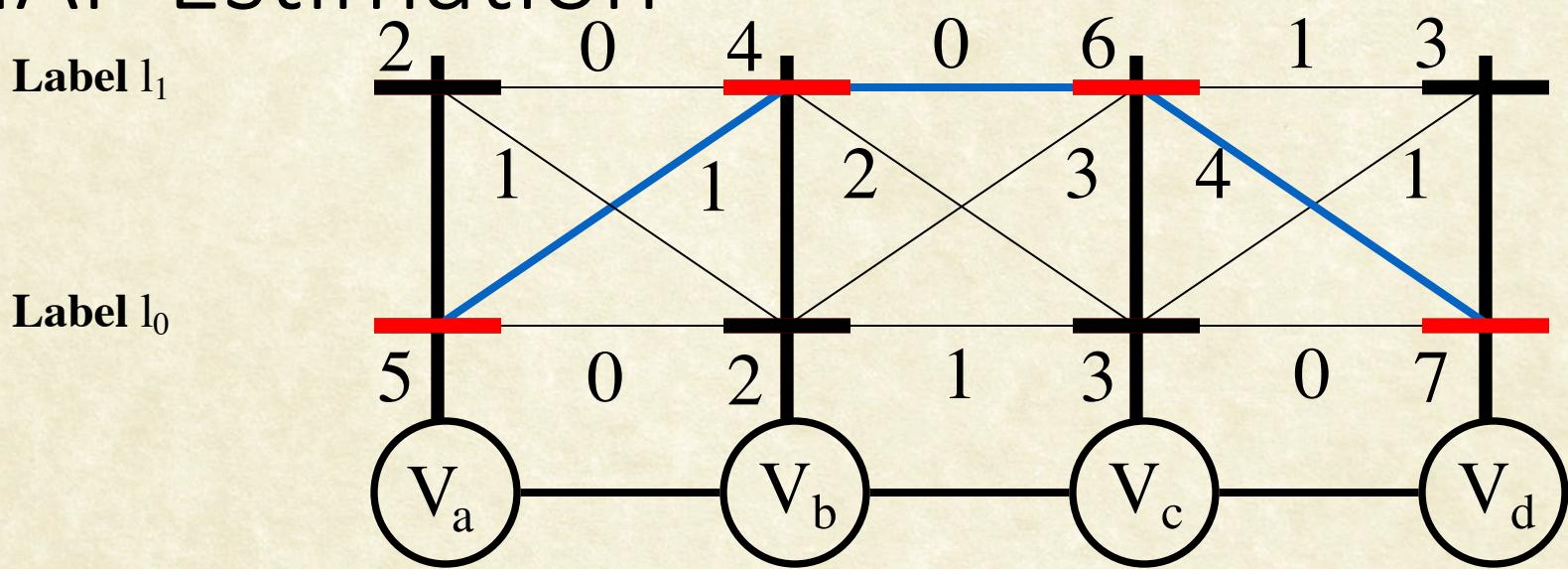
Unary Cost

Pairwise Cost

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



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

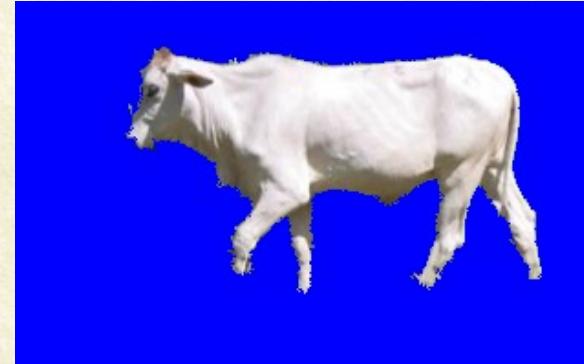
$f(a)$	$f(b)$	$f(c)$	$f(d)$	$Q(f; \theta)$
1	0	0	0	16
1	0	0	1	13
1	0	1	0	25
1	0	1	1	18
1	1	0	0	18
1	1	0	1	15
1	1	1	0	23
1	1	1	1	16



Computational Complexity

Segmentation

$$2^{|V|}$$



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

Can we do better than brute-force?

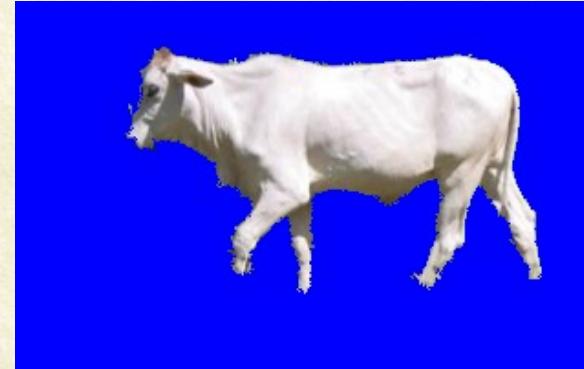
MAP Estimation is NP-hard !!



Computational Complexity

Segmentation

$$2^{|V|}$$



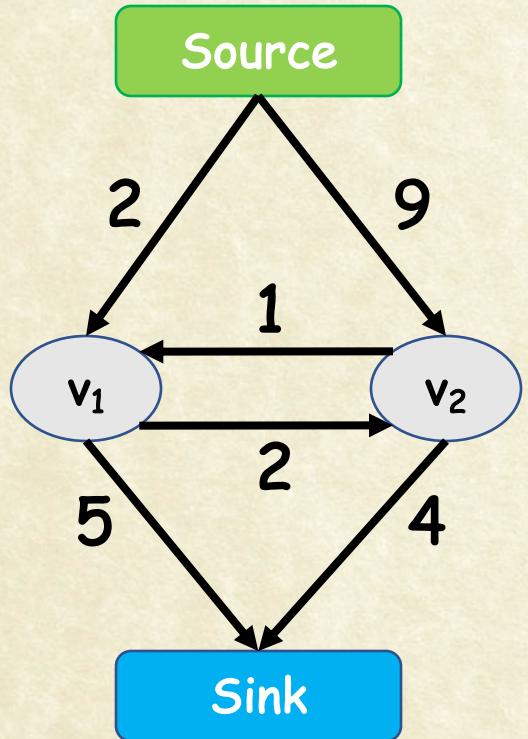
$$|V| = \text{number of pixels} \approx 320 * 480 = 153600$$

Exact algorithms do exist for special cases

Good approximate algorithms for general case



The st-Mincut Problem



Graph (V, E, C)

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

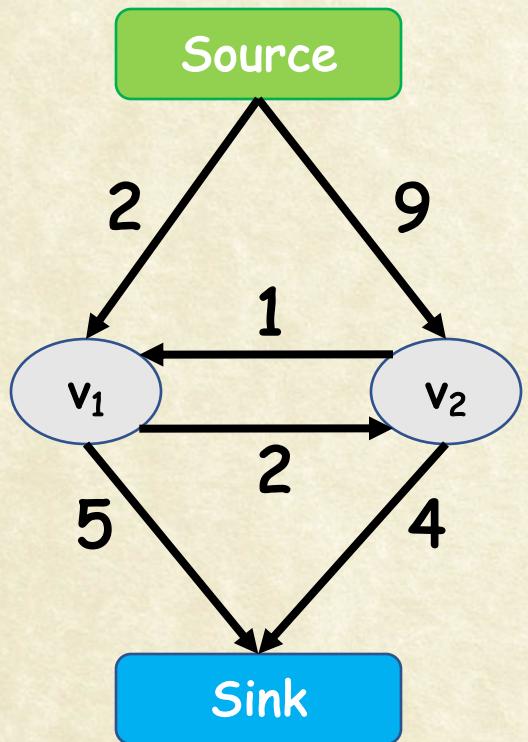
Edges $E = \{(v_1, v_2) \dots\}$

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



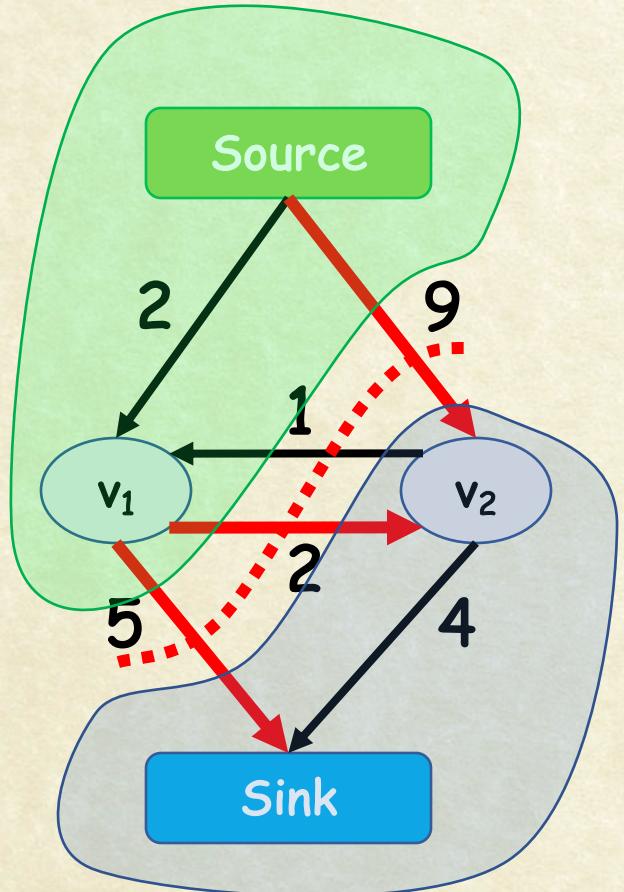
The Solution: st-Mincut

What is an st-cut?





The st-Mincut Problem



$$5 + 2 + 9 = 16$$

What is an st-cut?

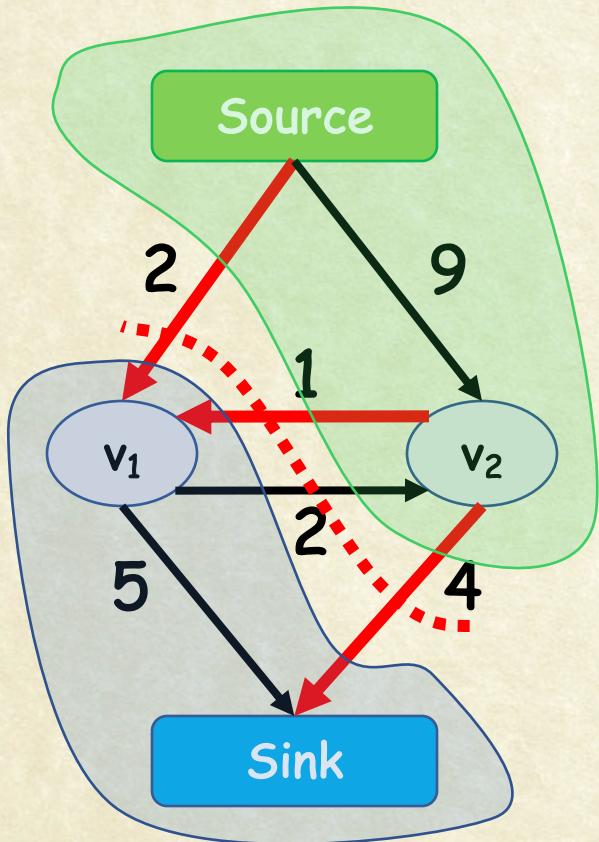
An st-cut (S, T) divides the nodes between source and sink.

What is the cost of an st-cut?

Sum of cost of all edges going from S to T



The st-Mincut Problem



$$2 + 1 + 4 = 7$$

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

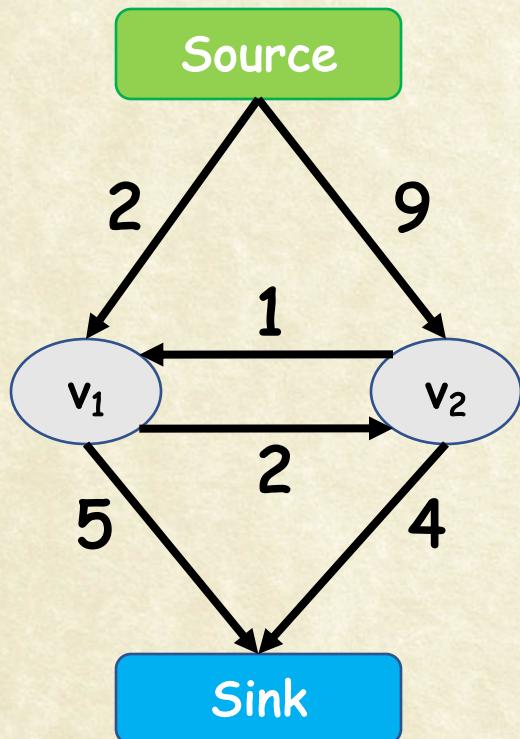
What is the st-mincut?

st-cut with the minimum cost



How to Compute the st-Mincut?

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



Constraints

Edges: Flow < Capacity

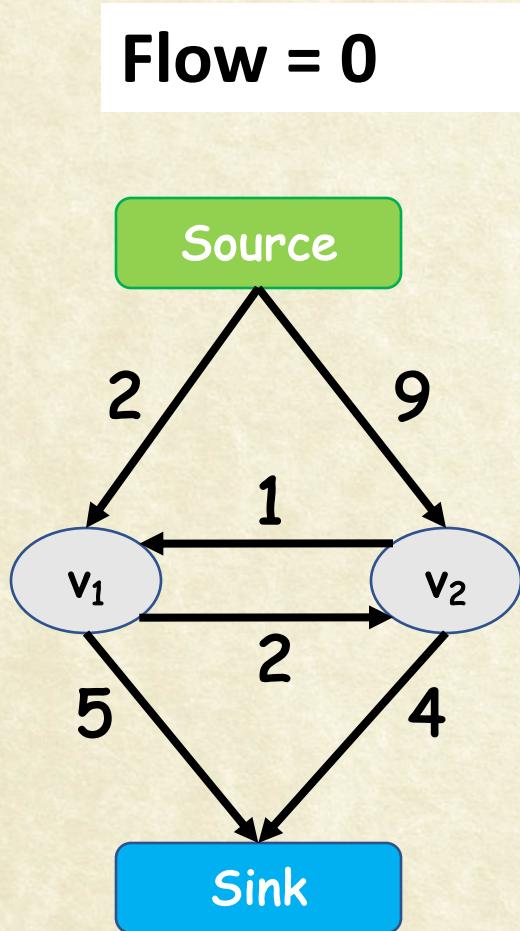
Nodes: Flow in = Flow out

Min-cut/Maxflow Theorem

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



Maxflow Algorithms



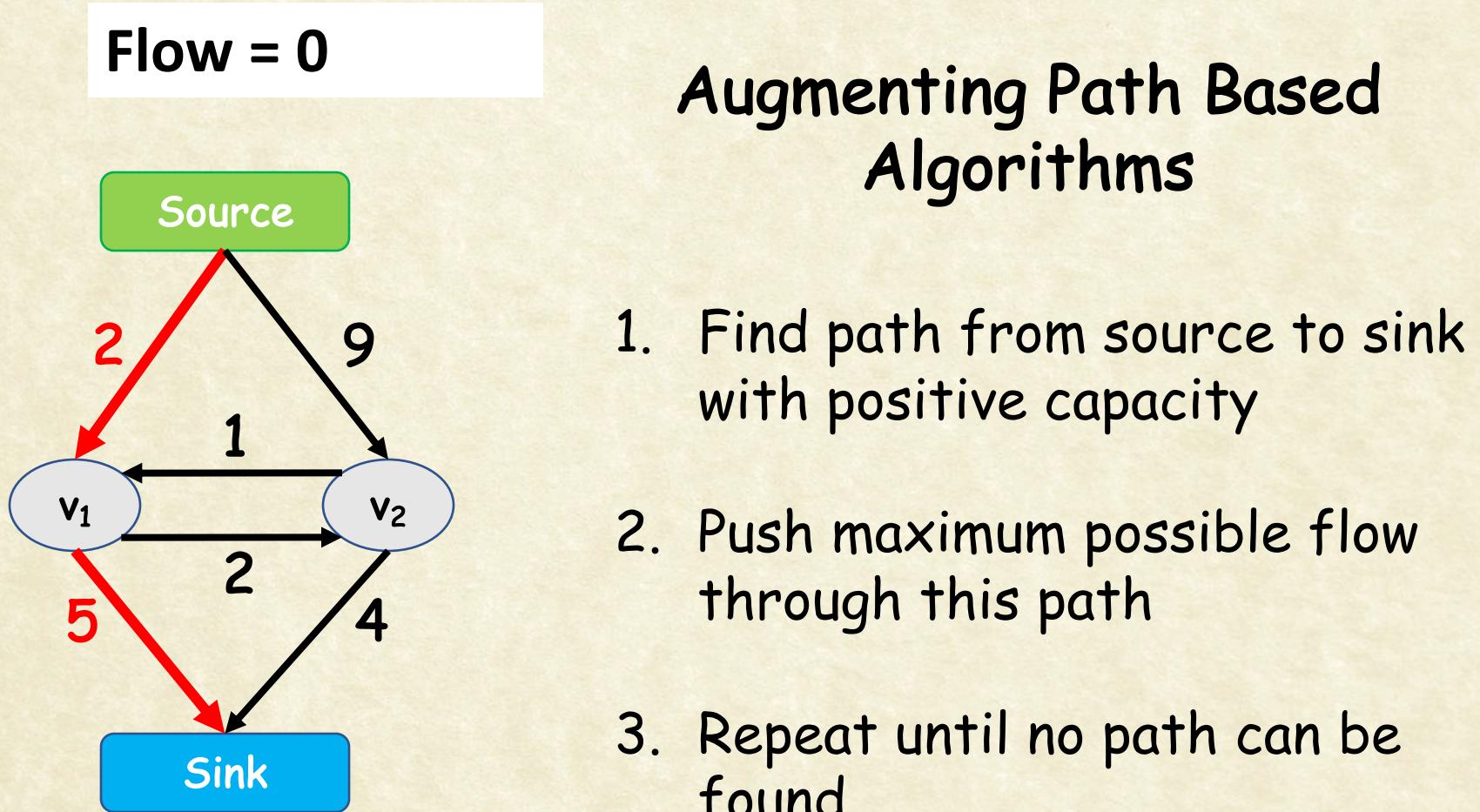
Augmenting Path Based Algorithms

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

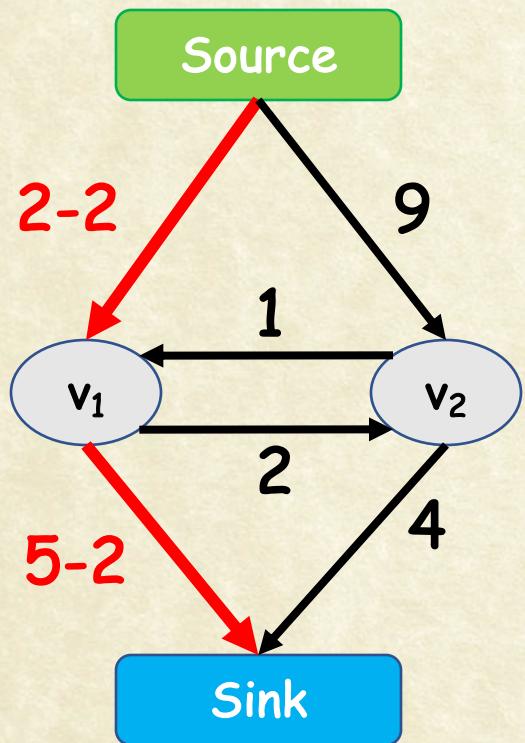


Algorithms assume non-negative capacity



Maxflow Algorithms

Flow = 0 + 2



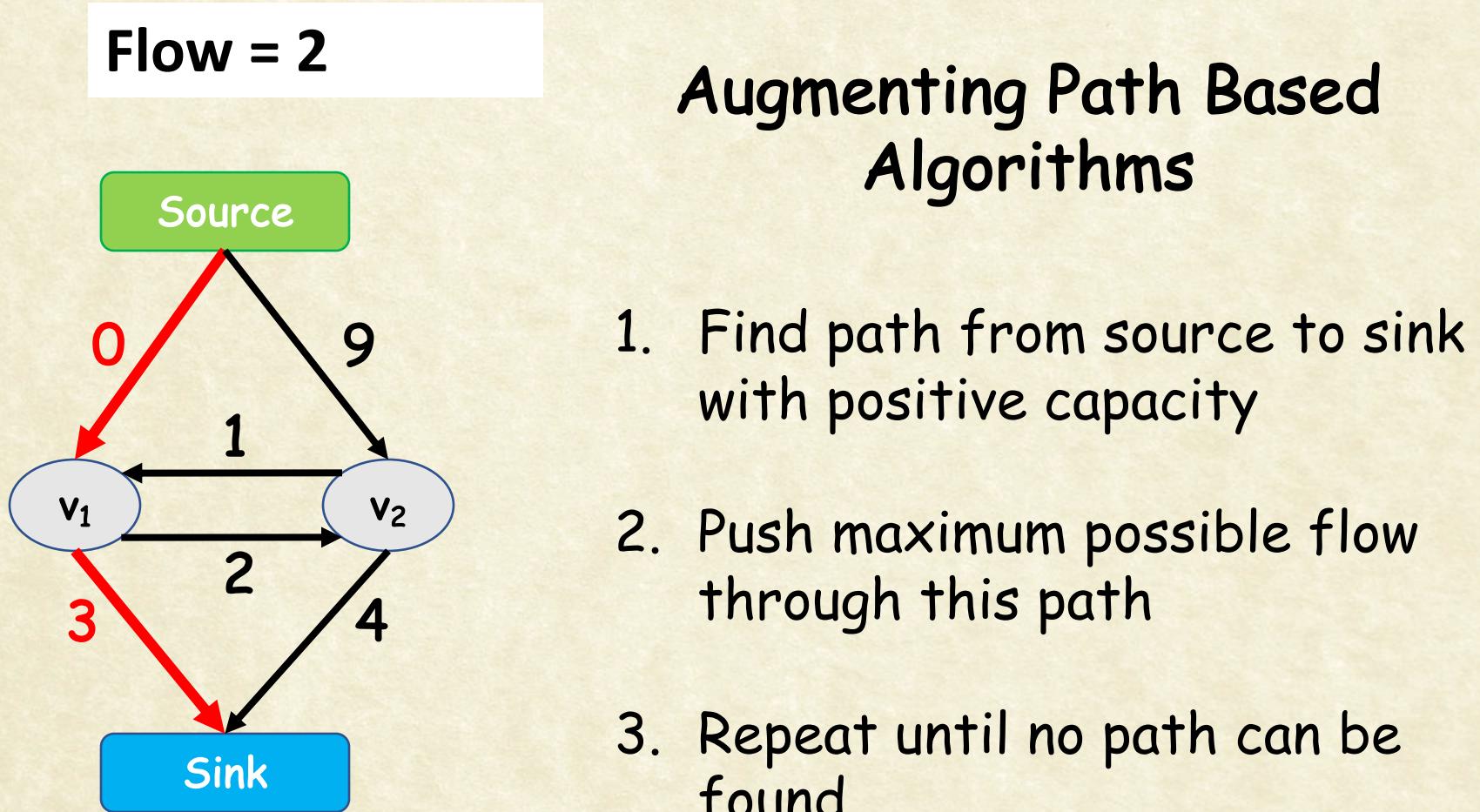
Augmenting Path Based Algorithms

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



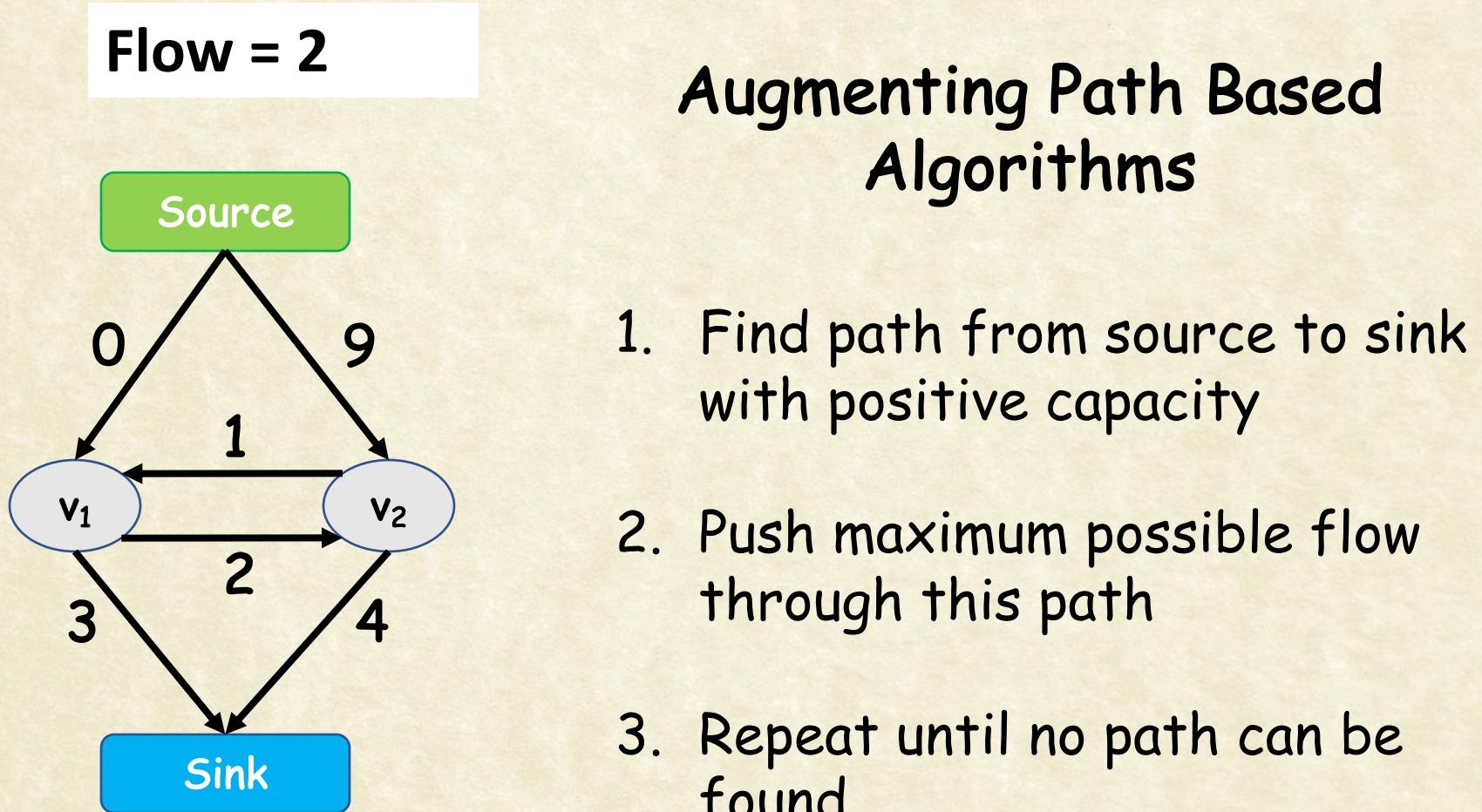
Maxflow Algorithms



Algorithms assume non-negative capacity



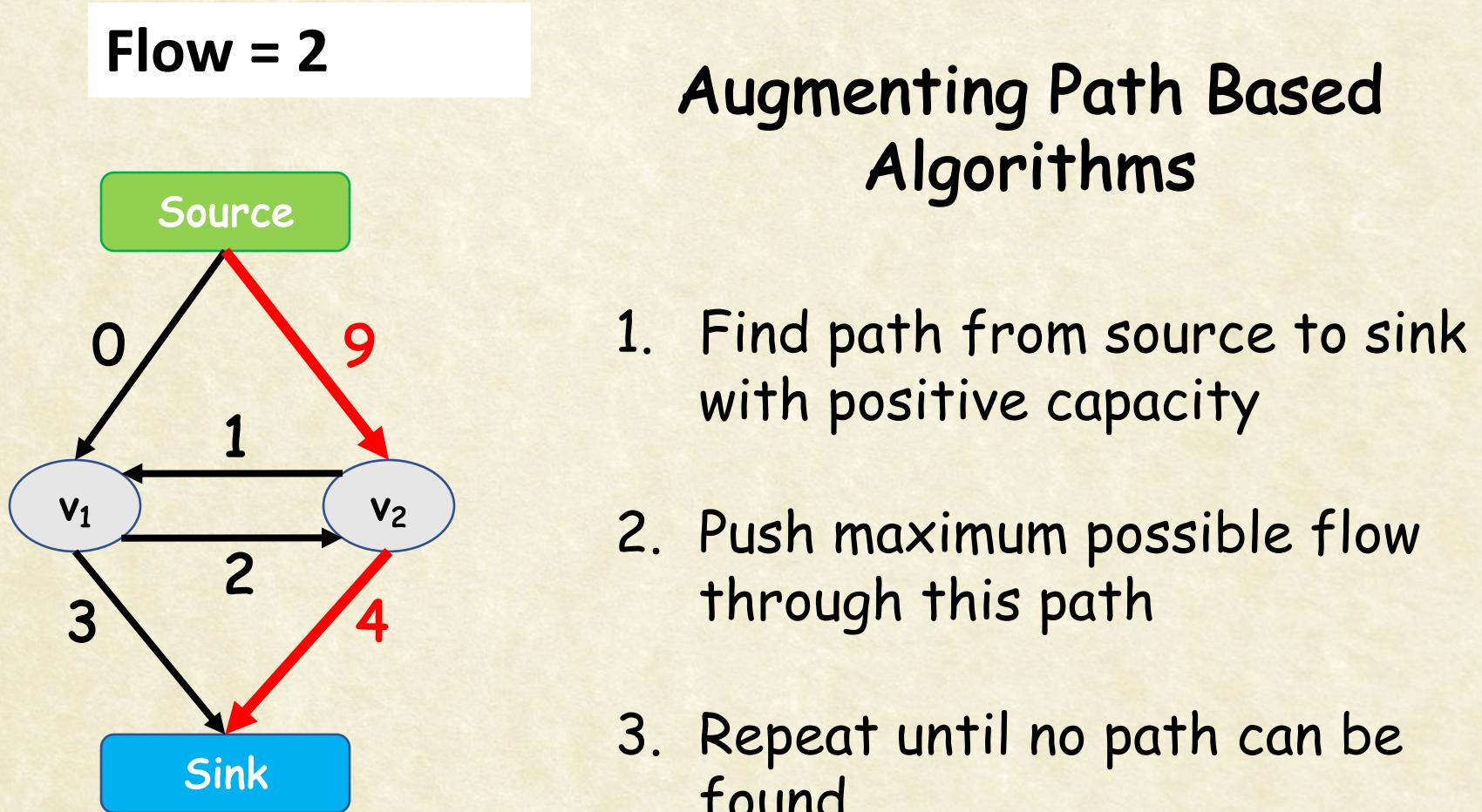
Maxflow Algorithms



Algorithms assume non-negative capacity



Maxflow Algorithms

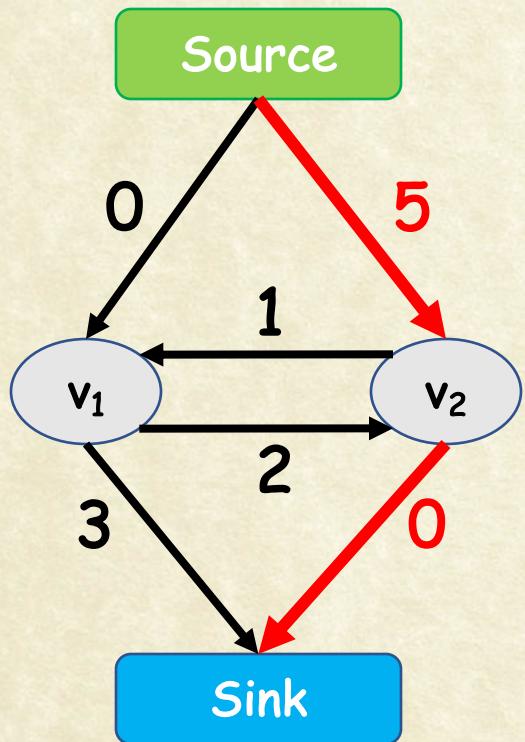


Algorithms assume non-negative capacity



Maxflow Algorithms

Flow = 2 + 4



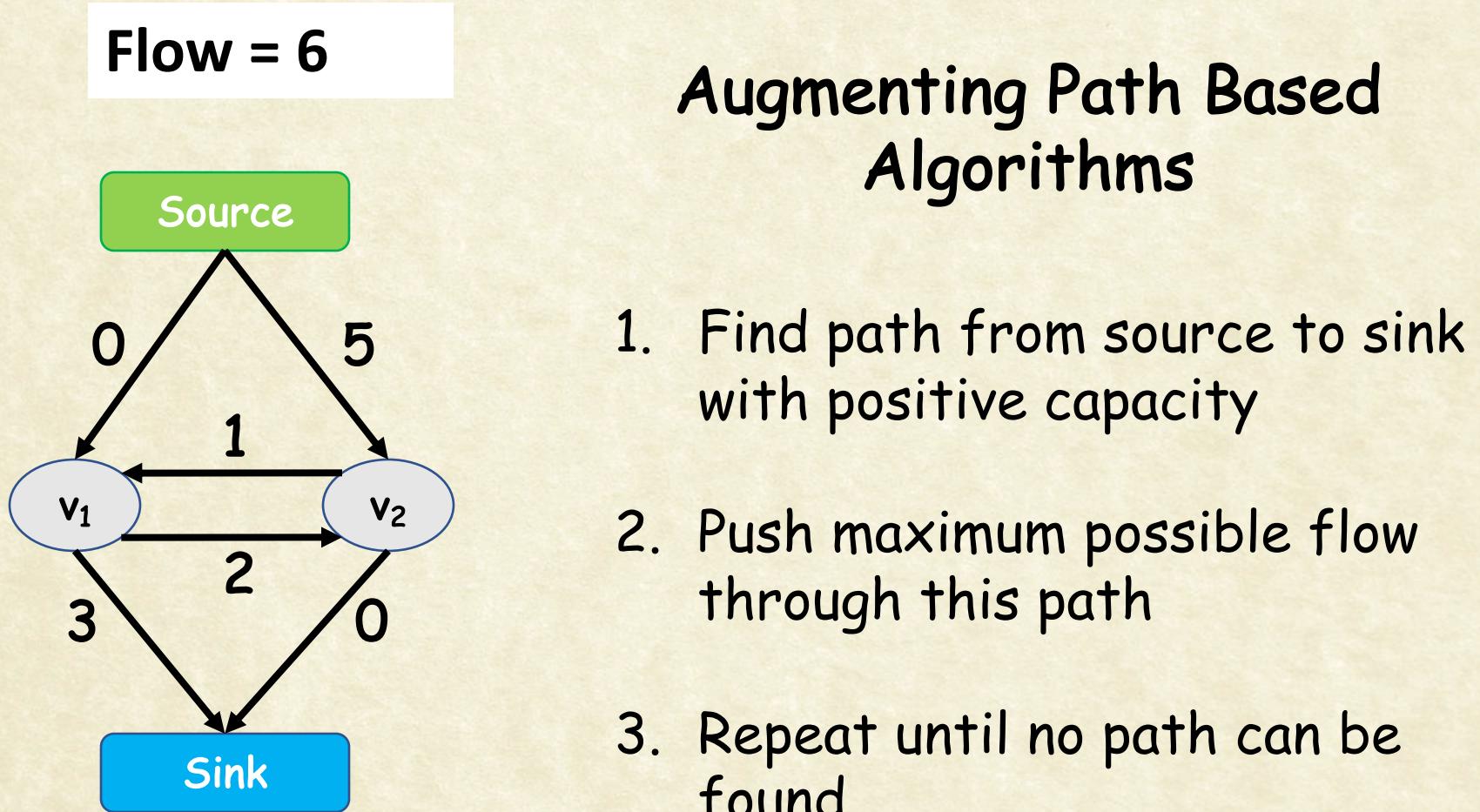
Augmenting Path Based Algorithms

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



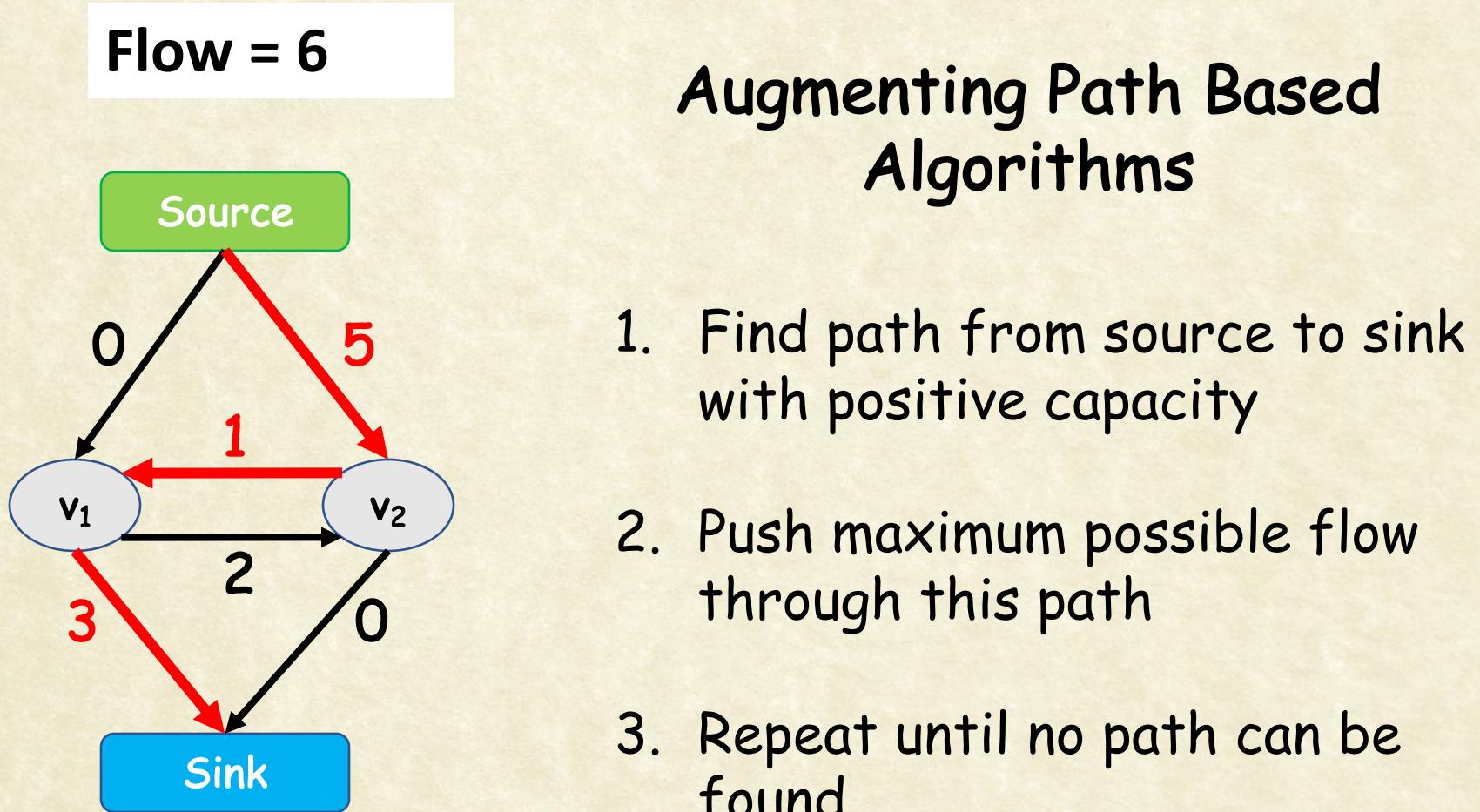
Maxflow Algorithms



Algorithms assume non-negative capacity



Maxflow Algorithms

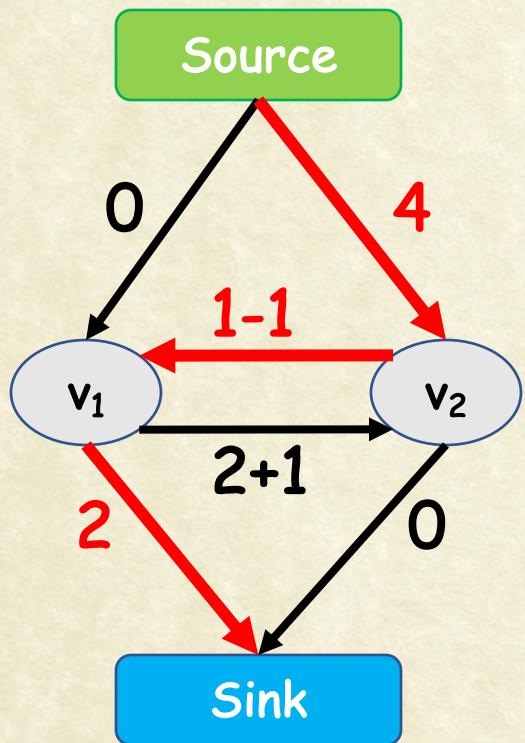


Algorithms assume non-negative capacity



Maxflow Algorithms

Flow = 6 + 1



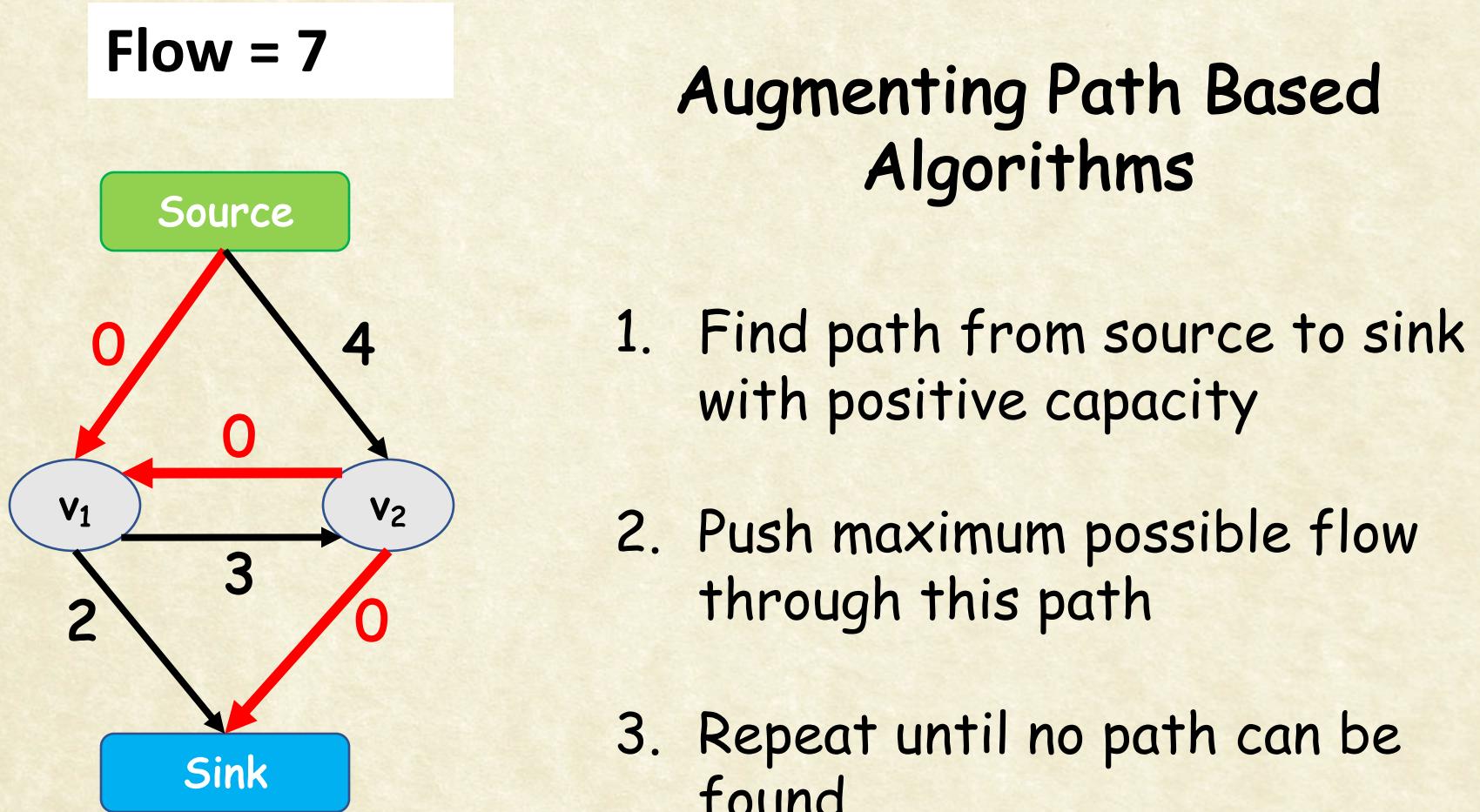
Augmenting Path Based Algorithms

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



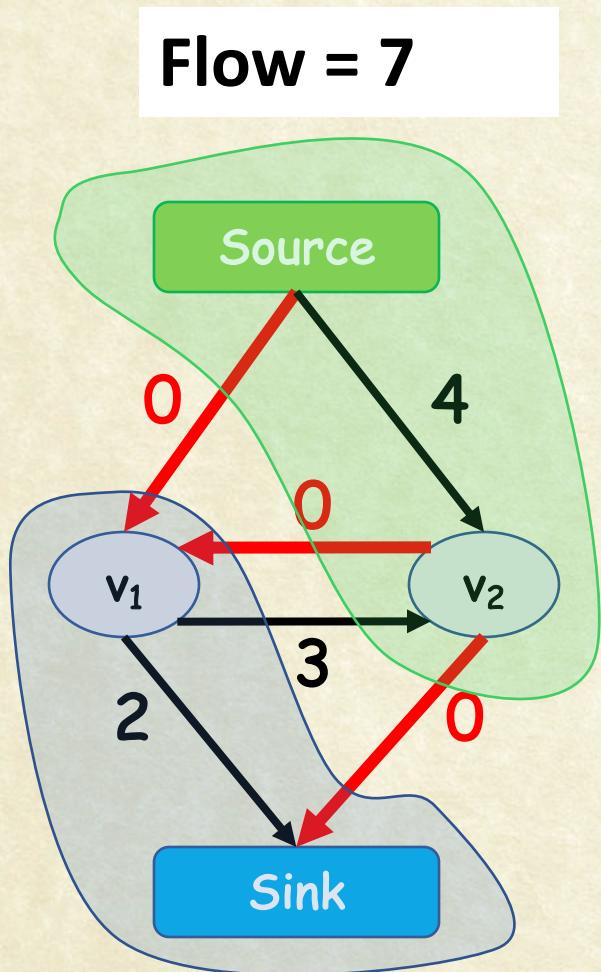
Maxflow Algorithms



Algorithms assume non-negative capacity



Maxflow Algorithms



Augmenting Path Based Algorithms

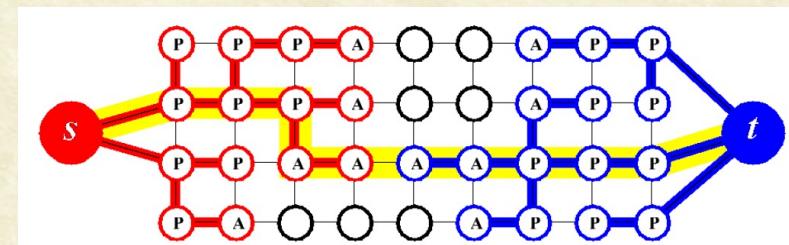
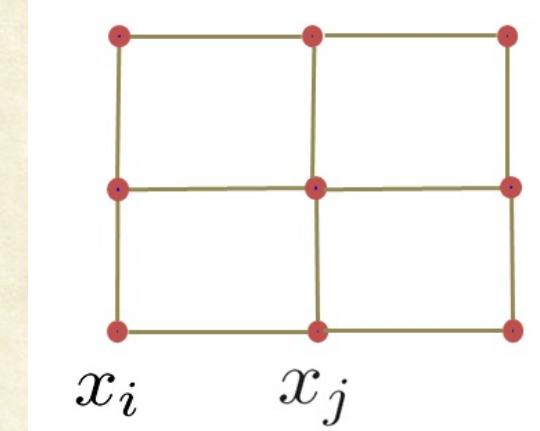
1. Find path from source to sink with positive capacity
2. Push maximum possible flow through this path
3. Repeat until no path can be found

Algorithms assume non-negative capacity



Maxflow in Computer Vision

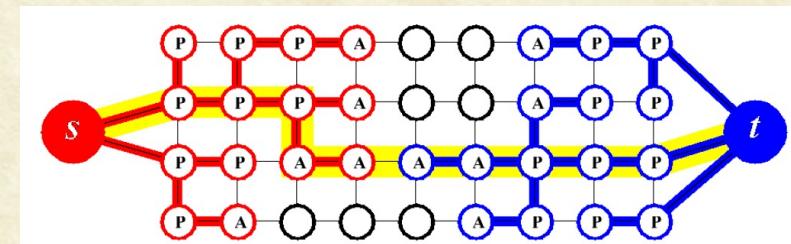
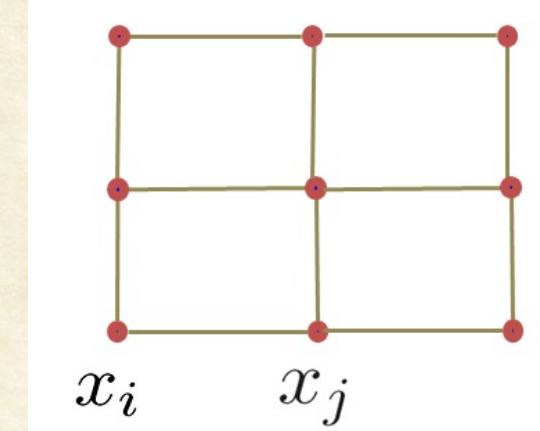
- Specialized algorithms for vision problems
 - Grid graphs
 - Low connectivity ($m \sim O(n)$)
- Dual search tree augmenting path algorithm
[Boykov and Kolmogorov PAMI 2004]
 - Finds approximate shortest augmenting paths efficiently
 - High worst-case time complexity
 - Empirically outperforms other algorithms on vision problems





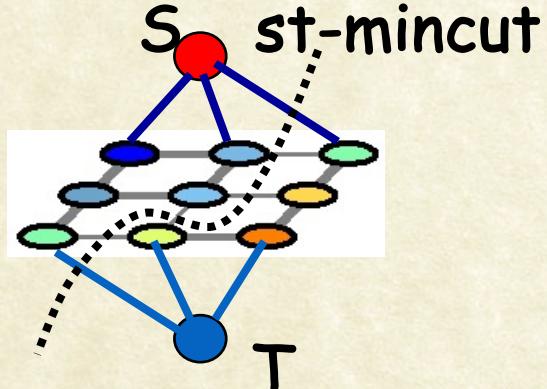
Maxflow in Computer Vision

- 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





St-mincut and Energy Minimization



Minimizing a Quadratic
Pseudoboolean
function $E(x)$

Functions of boolean variables

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

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

$$c_{ij} \geq 0$$

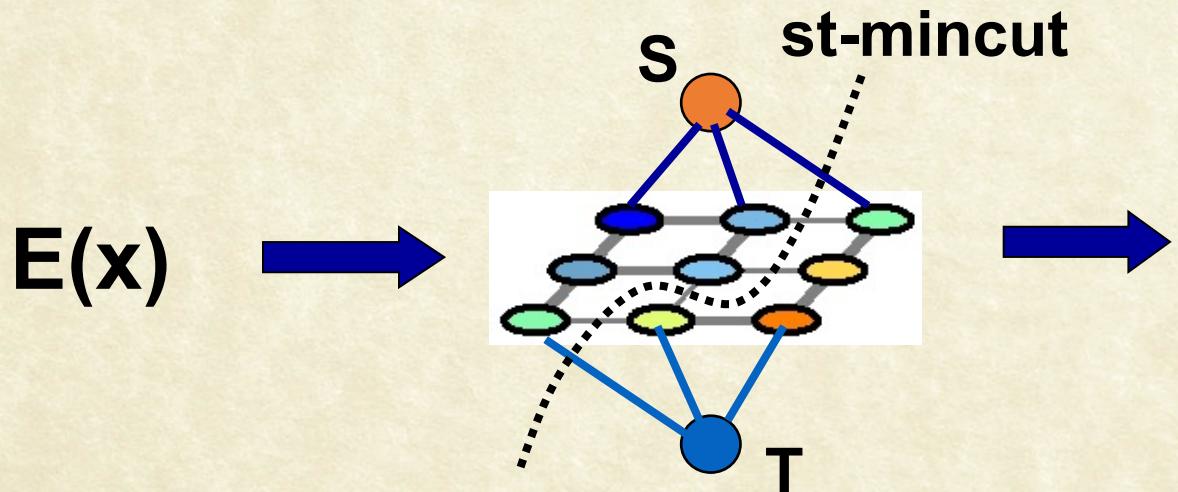
Polynomial time *st*-mincut algorithms
require non-negative edge weights



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



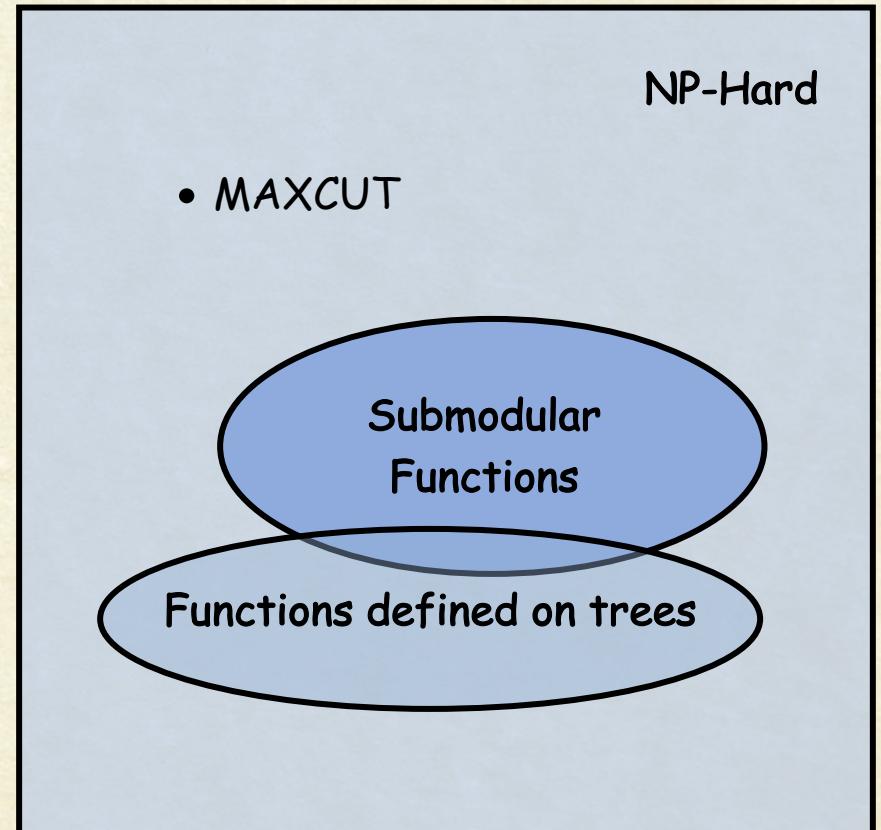
Minimizing Energy Functions

- **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

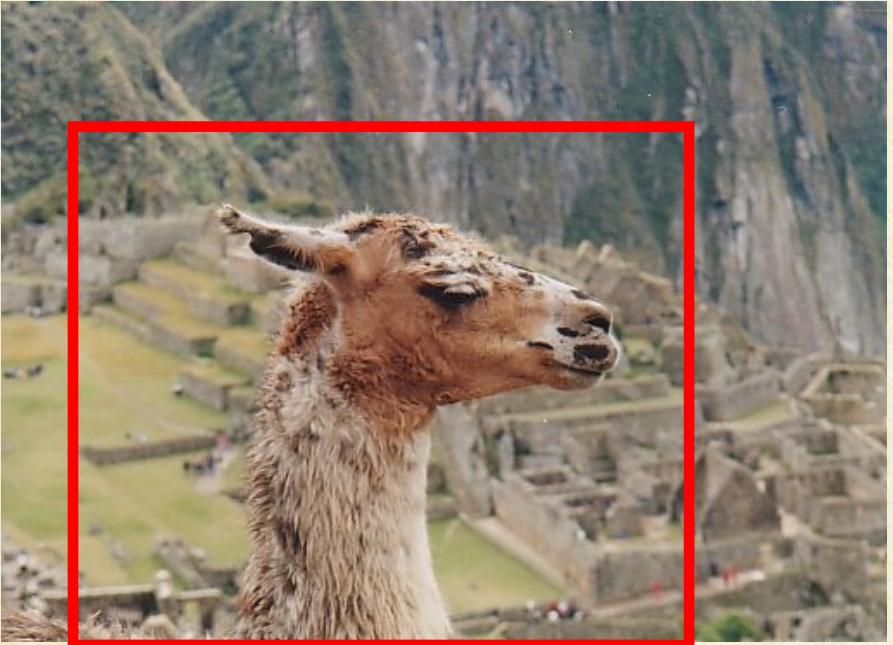
Interactive Foreground Extraction
using Iterated Graph Cuts

Carsten Rother
Vladimir Kolmogorov
Andrew Blake

Microsoft Research Cambridge-UK



The Problem



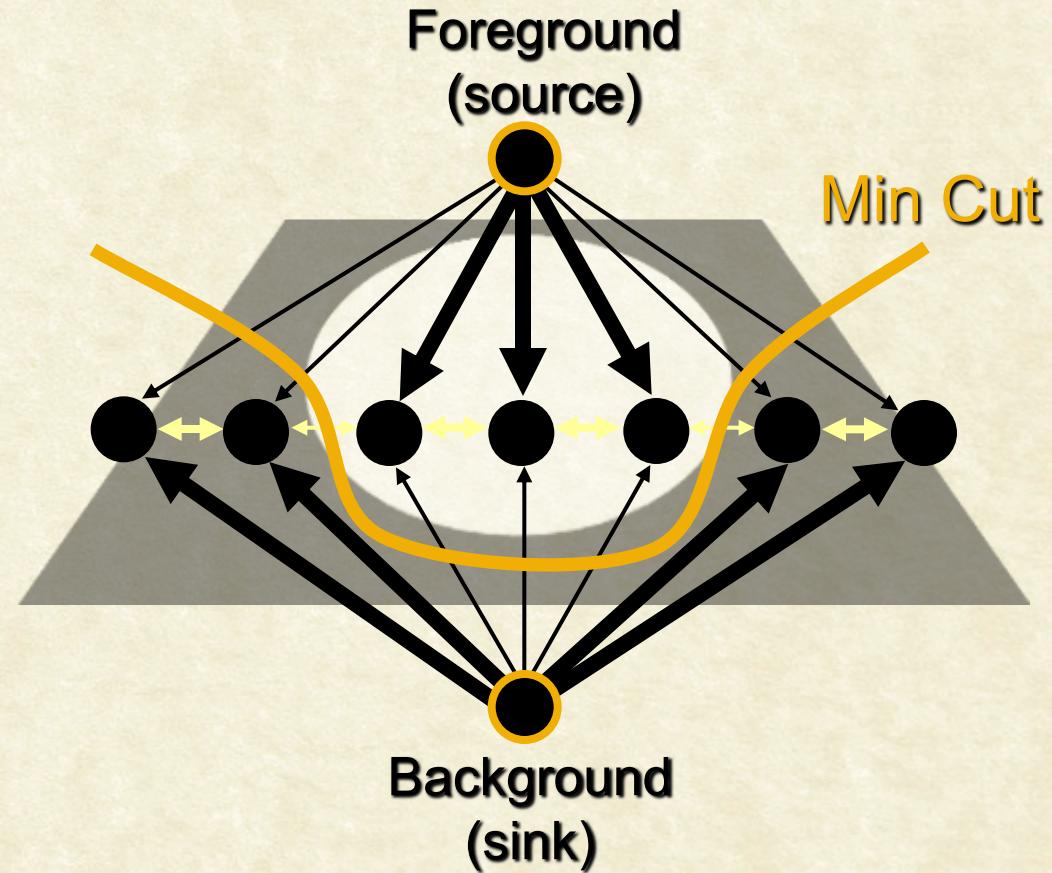
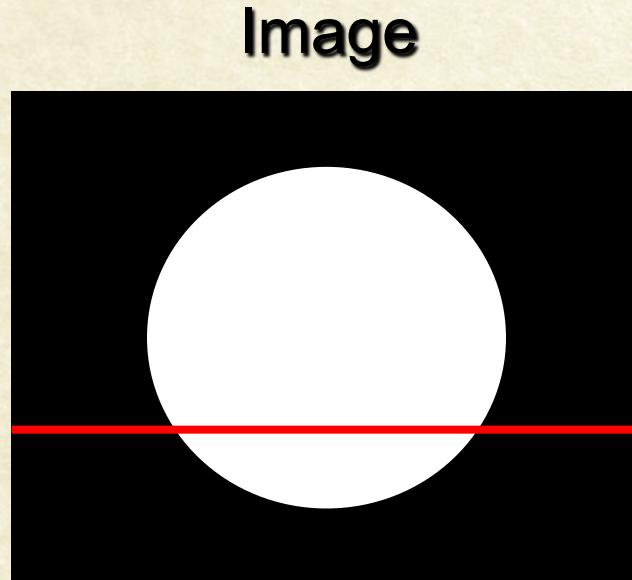
Fast &
Accurate ?



- 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

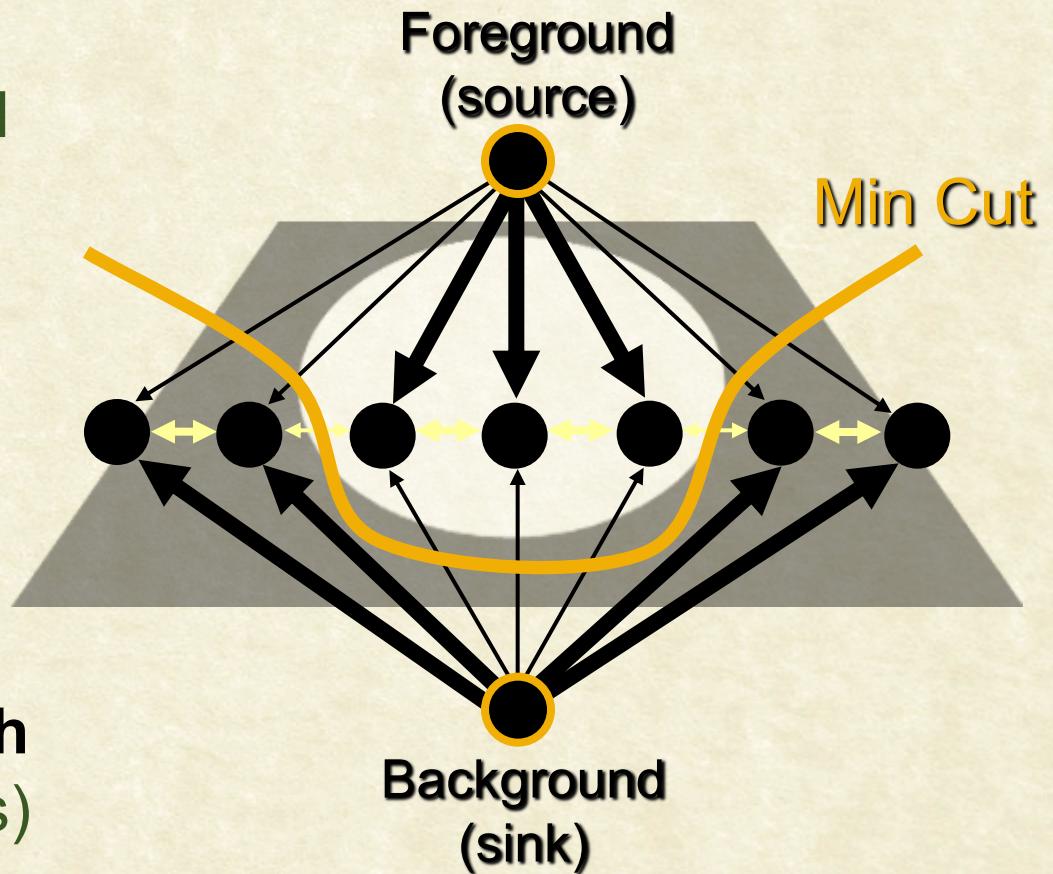


Graph Cuts for Foreground Extraction

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

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

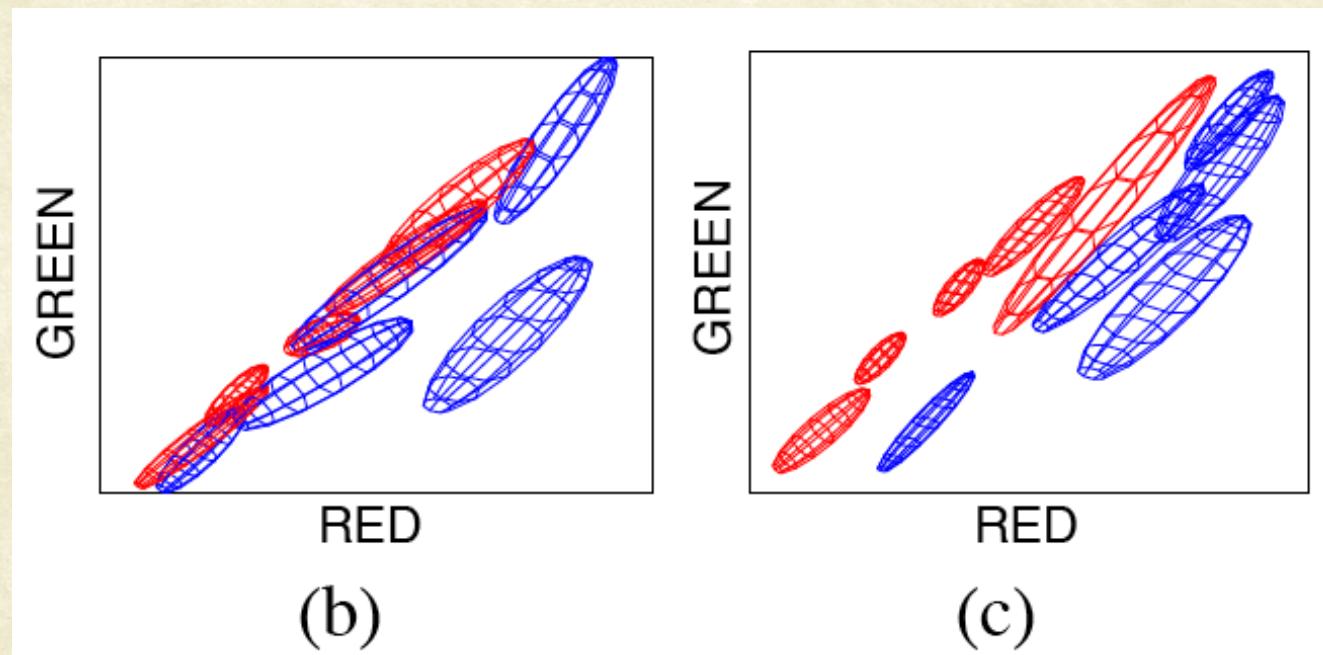
Regularization = **color match**
(cost of separating neighbors)





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





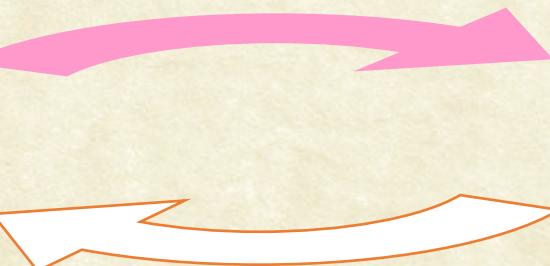
Iterated Graph Cuts



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



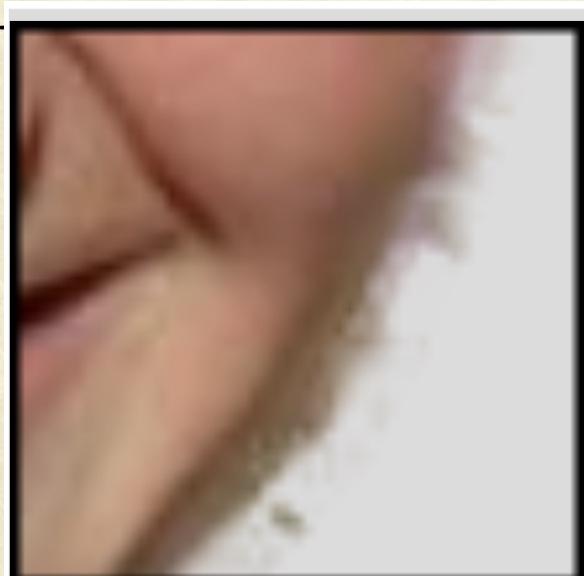
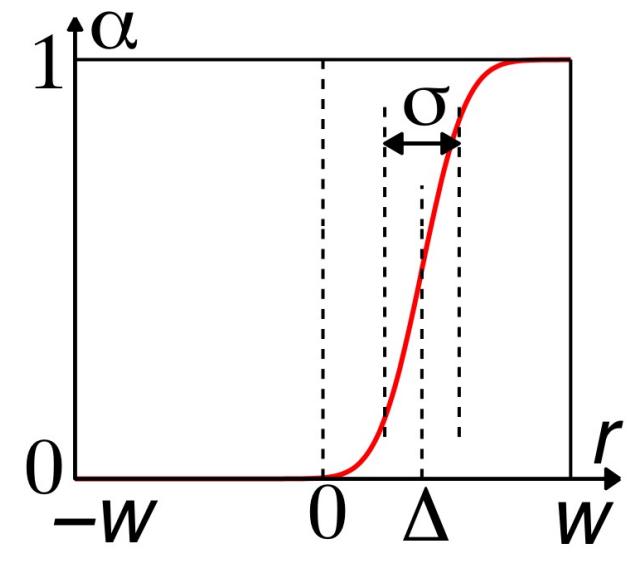
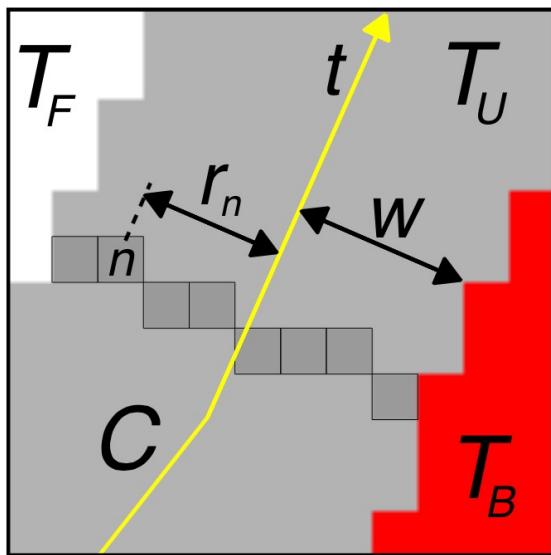
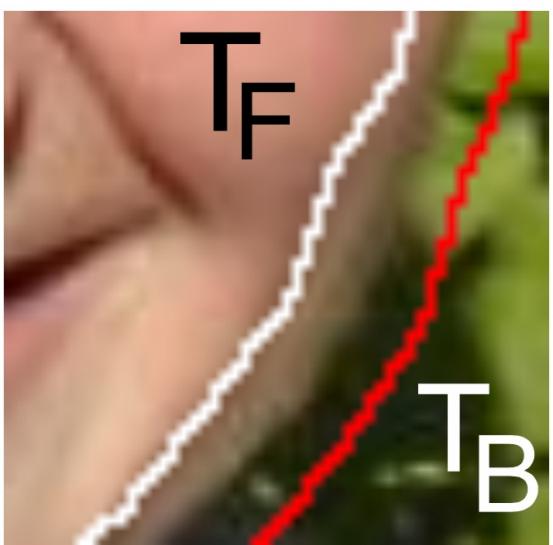
Result



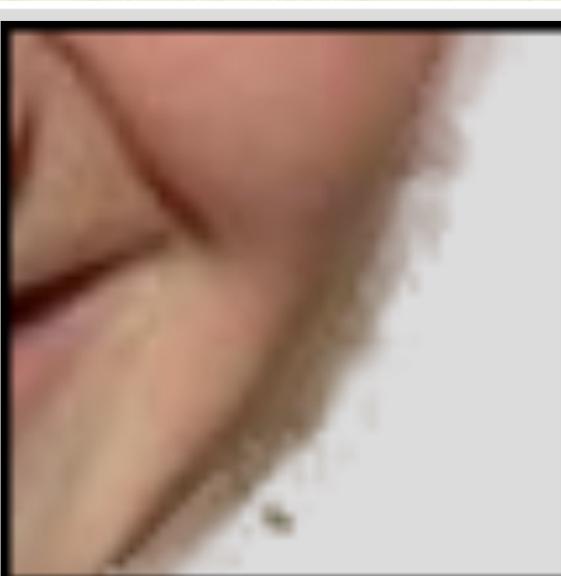
Energy after each Iteration



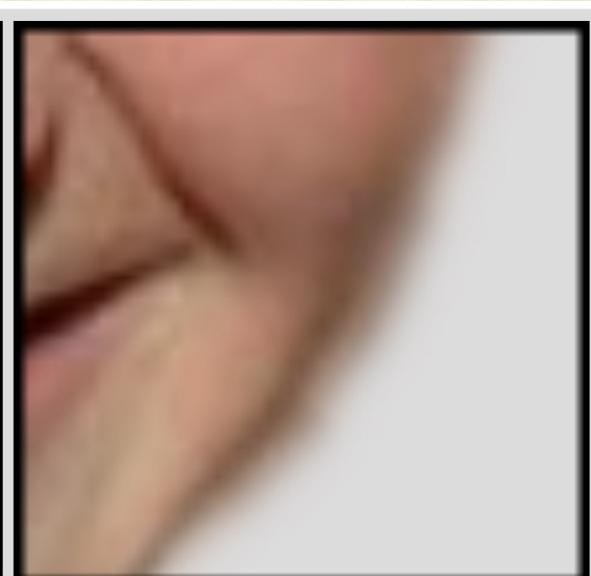
Border Matting



Knockout 2



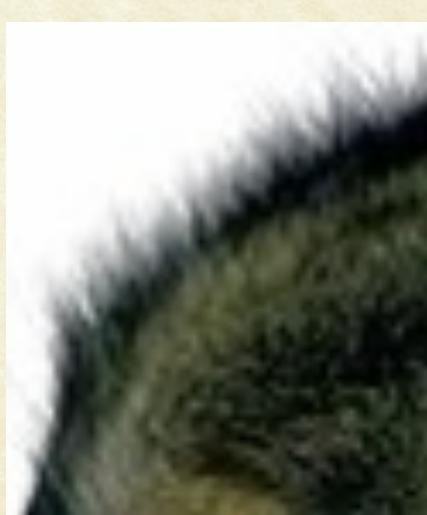
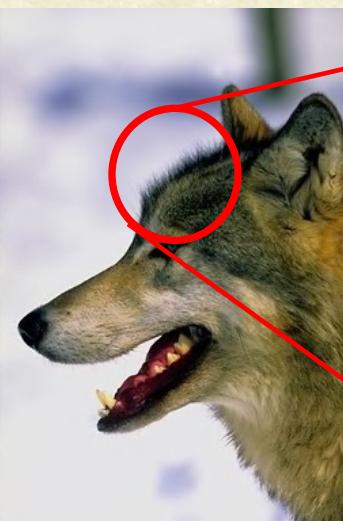
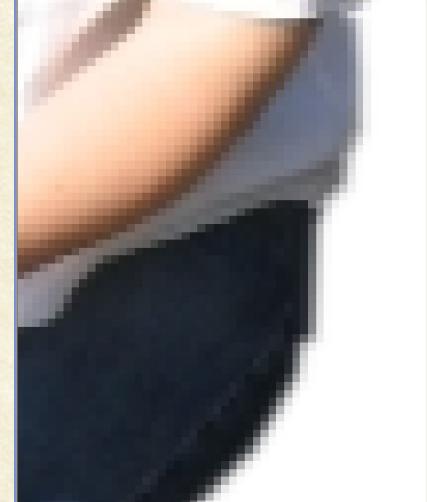
Bayes Matte



GrabCut



Border Matting Results





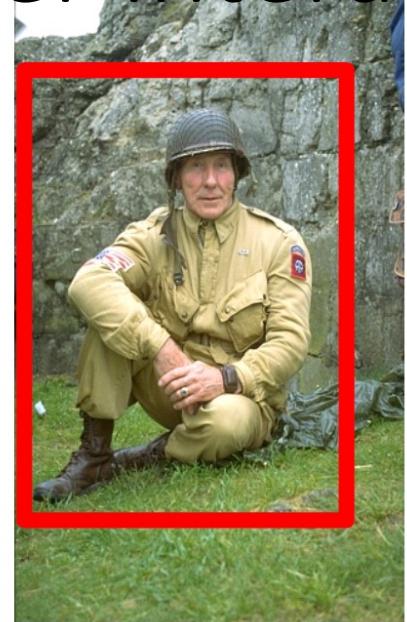
Examples: Moderate Complexity



... GrabCut completes automatically



User Interaction



Automatic



Segmentation



User



Interaction



Automatic



Segmentation





Examples: Difficult Ones

Camouflage &
Low Contrast

Initial
Rectangle



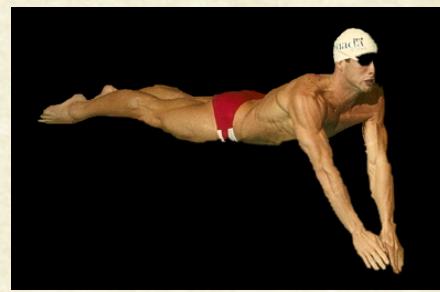
Initial
Result



Fine structure



No telepathy





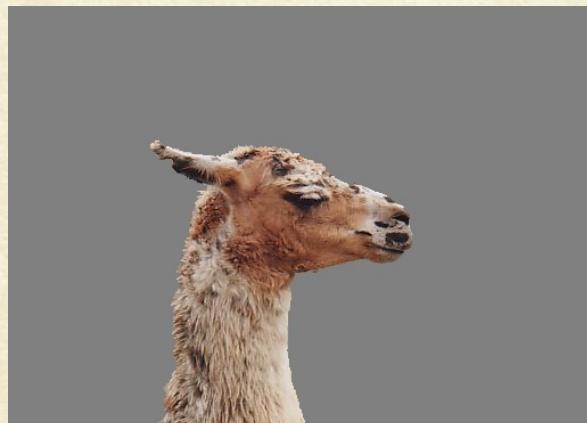
GraphCut vs. GrabCut

Boykov and Jolly (2001)

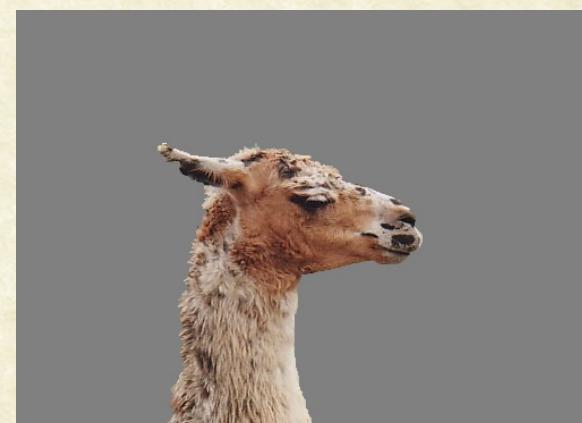
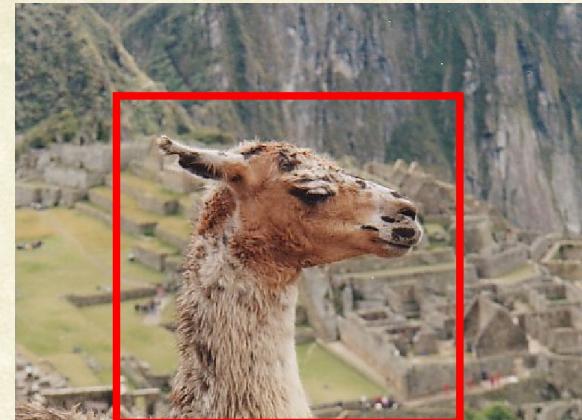
User
Input



Result



GrabCut



Error Rate: 0.72%

Error Rate: 0.72%



Reading Material

- Required
 - “Normalized Cuts and Image Segmentation”, by Jianbo Shi and Jitendra Malik, TPAMI, Aug 2000
 - “Grabcut: Interactive Foreground Extraction using Iterated Graph Cuts”, by Rother, Kolmogorov and Blake, Siggraph 2004.
- Additional
 - “OBJ CUT”, by M. Pawan Kumar P.H.S. Torr and A. Zisserman, CVPR 2005.



Questions?