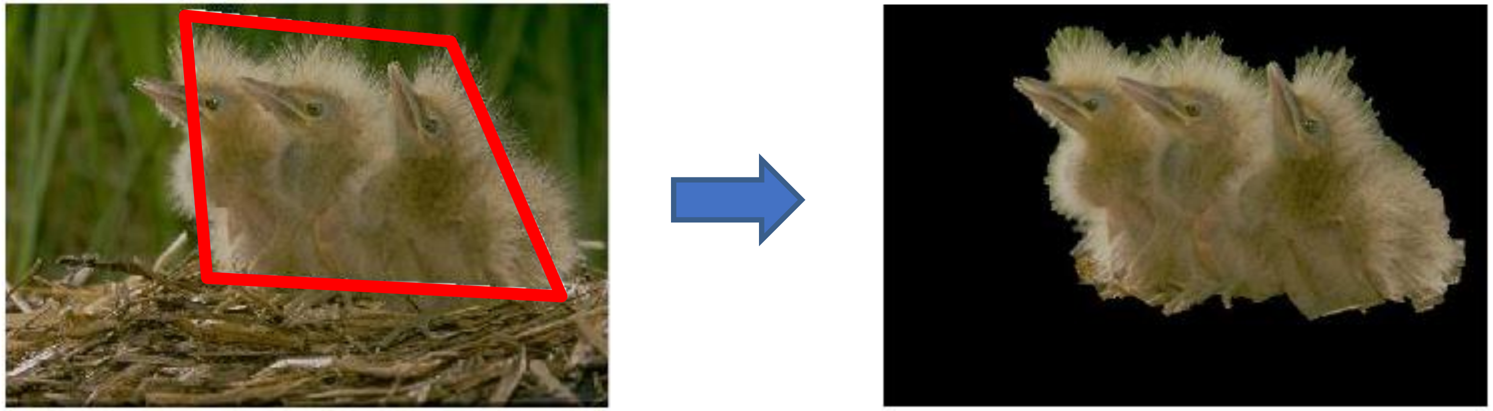
6. Interactive Segmentation & Graph-Cut

Semi-automated Segmentation

- User provides imprecise and incomplete specification of region – your algorithm has to read his/her mind.



Key problems

1. What groups of pixels form cohesive regions?
2. What pixels are likely to be on the boundary of regions?
3. Which region is the user trying to select?

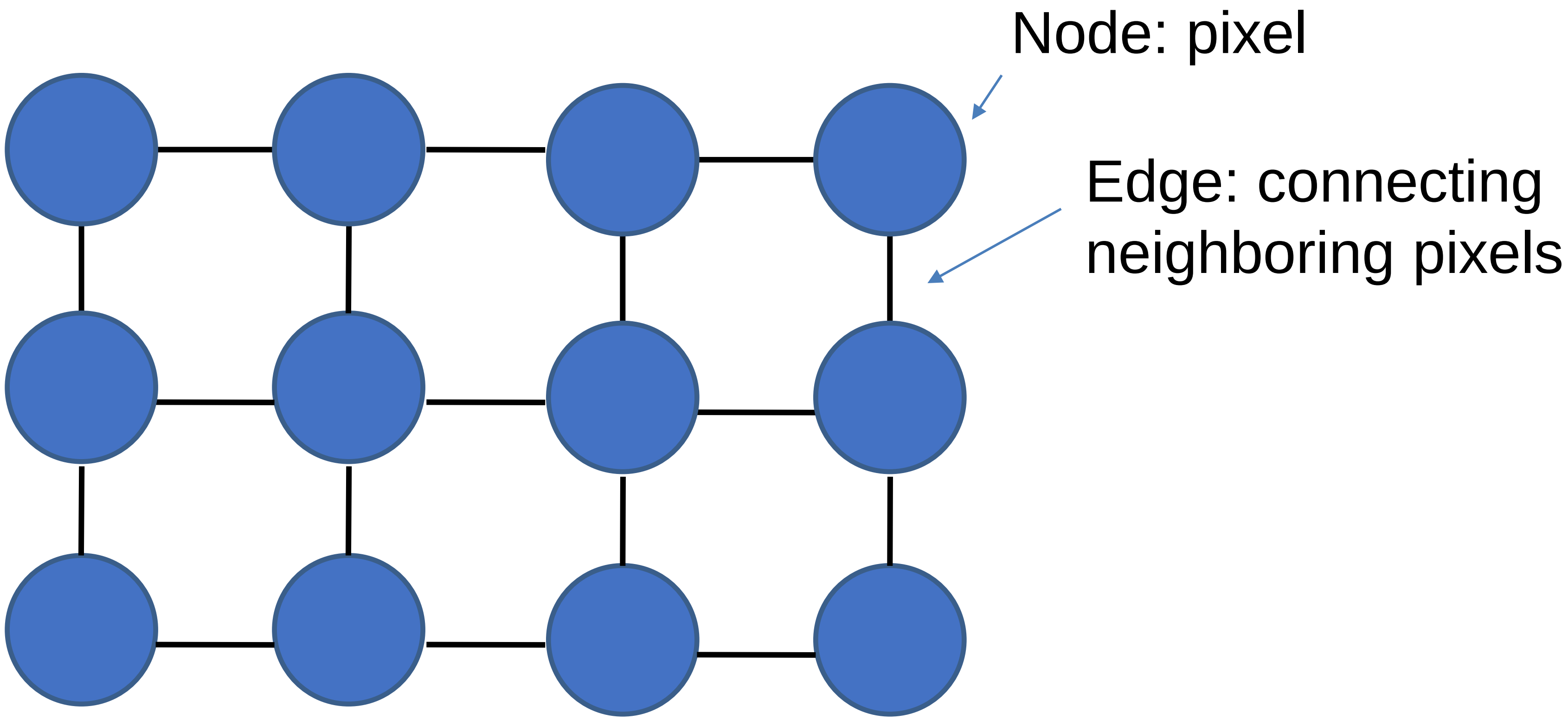
What makes a good region?

- Contains similar color/texture
- Looks different than background
- Compact

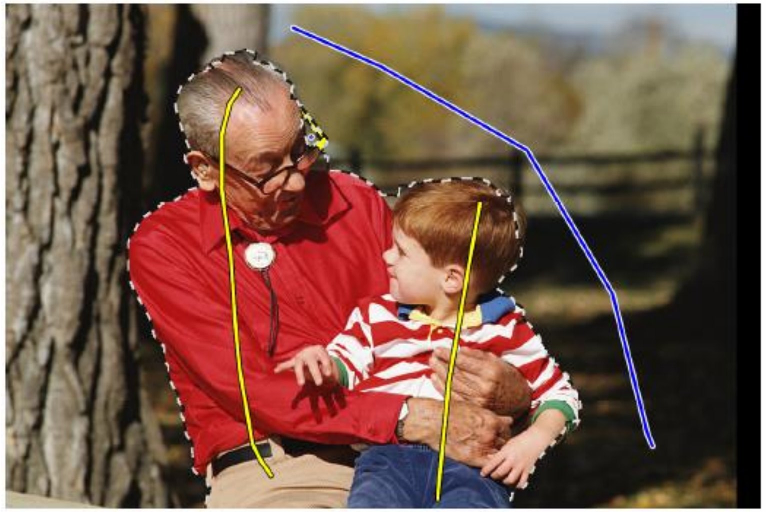
What makes a good boundary?

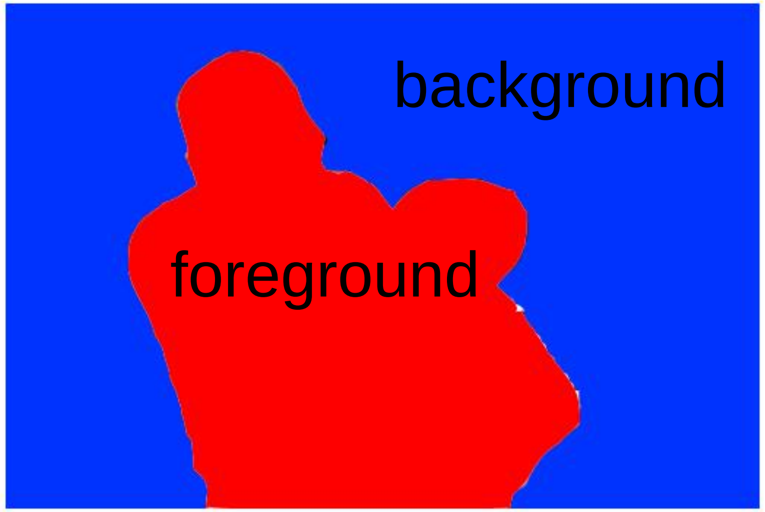
- High gradient along boundary
- Gradient in right direction
- Smooth

The Image as a Graph



Segmentation as a 2-class classification problem

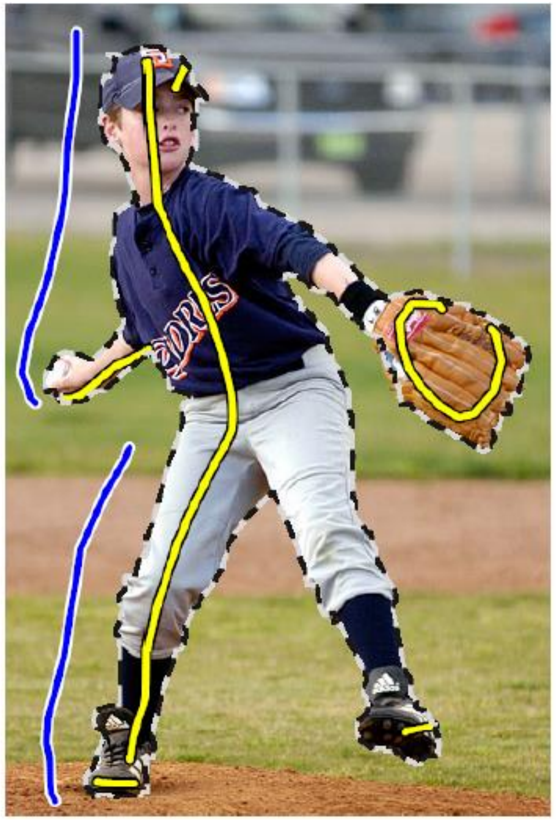


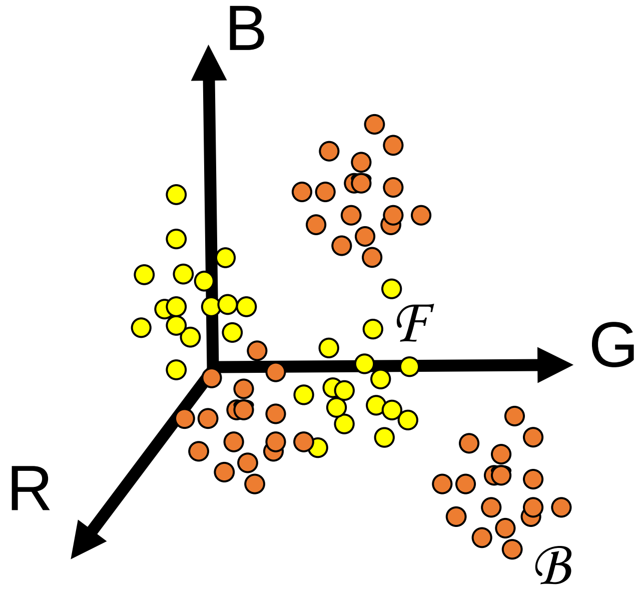


Each pixel in the image should
be labeled as either "background" or "foreground"

Result

Training examples via markup





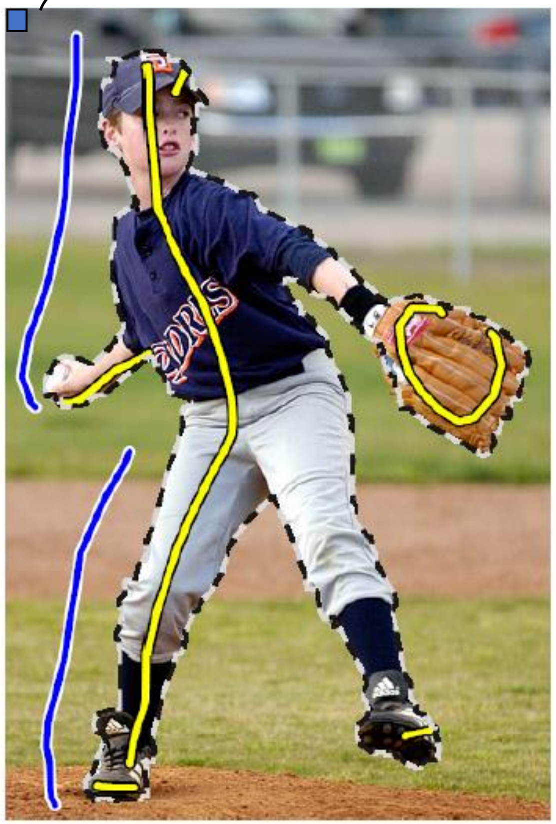
Result: n $\mathcal{F}$ pixels
m $\mathcal{B}$ pixels

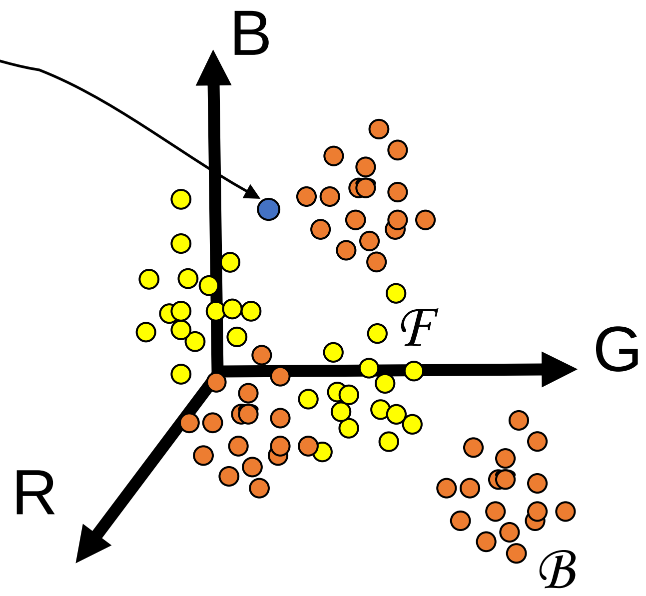
Pixels along user-scribble provide
"supervised" RGB training-data
Blue = background ($\mathcal{B}$), yellow = foreground ($\mathcal{F}$)

Simple 1-NN Classifier

Given an unlabeled pixel, $C(i)$.
Decide whether is background or foreground.

?





Compute new pixels RGB Euclidean distance (L2-norm) to all labeled B pixels, and all labeled F pixels.

Select nearest from each.

$$d_i^{\mathcal{F}} = \min_n ||C(i) - K_n^{\mathcal{F}}||$$

$$d_i^{\mathcal{B}} = \min_m ||C(i) - K_m^{\mathcal{B}}||$$

Problem Formulation

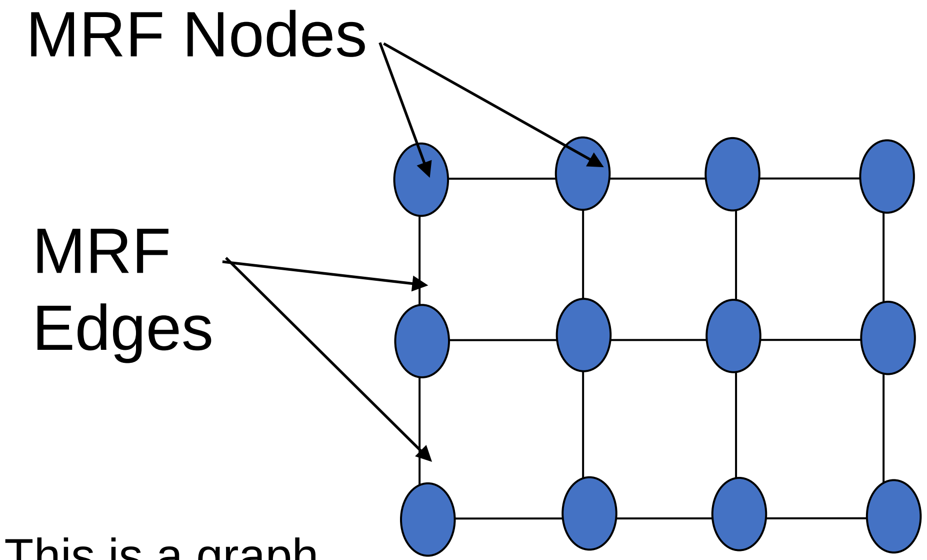
- For each pixel, we can assign a “label” that this pixel is either foreground or background.
- To automate this process, we define a cost for foreground/background at each pixel.
- The lower the cost, the more confident a pixel is to belong to a class.

$$
\begin{array}{lll}
E_1(x_i = 1) = 0 & E_1(x_i = 0) = \infty & \forall i \in \mathcal{F} \\
E_1(x_i = 1) = \infty & E_1(x_i = 0) = 0 & \forall i \in \mathcal{B} \\
E_1(x_i = 1) = \frac{d_i^{\mathcal{F}}}{d_i^{\mathcal{F}} + d_i^{\mathcal{B}}} & E_1(x_i = 0) = \frac{d_i^{\mathcal{B}}}{d_i^{\mathcal{F}} + d_i^{\mathcal{B}}} & \forall i \in \mathcal{U}
\end{array}
$$

x_i is a pixel label (not its color). 1= Foreground, 0 = Background
E_1 is the cost;
$\mathcal{F}$, $\mathcal{B}$ represent the training-data (already labeled). $\mathcal{U}$ are unlabeled/uncertain pixels.

Adding a Markov Random Field

- The per-pixel cost is not enough
- To perform the final labeling, an MRF is used – this enforces spatial constraints



This is a graph,
with $\{\mathcal{V},\mathcal{E}\}$
$\mathcal{V}$ = nodes (vertices)
$\mathcal{E}$ = edges

Cost for labeling a node is $E_1(xi)$
(as defined on the previous slide)
Node cost often called the "data cost" or "likelihood energy"

Edges have two vertices x_i and x_j.
Cost for an edge depend on what labels are assigned to x_i and x_j.

We will call this cost $E_2(x_i, xj)$
(defined on next slide)

Edges cost often called **"smoothness term", or "smoothness prior"**, or **"prior energy"**

Excellent MRF code: http://vision.middlebury.edu/MRF/

Edge Costs

$$E_2(x_i, x_j) = |x_i - x_j| \cdot g(C_{ij})$$

where

$$g(\xi) = \frac{1}{\xi + 1}, \quad C_{ij} = ||C(i) - C(j)||^2$$

Possible Edge Configurations and cost:

Configuration 1

$F \leftrightarrow B$

Cost = 1/[Color Difference]

Small Color Difference = Large Cost
Large Color Difference = Small Cost
(Ask yourself why?)

Configuration 2

$F \leftrightarrow F$

Cost = 0
|1-1| = 0

Configuration 3

$B \leftrightarrow B$

Cost = 0
|0-0| = 0

How Edge Cost work

Two labels (l_1 and l_2)

Three nodes, n_1, n_2, n_3

(A very simple example)

l_1 data cost	0.2	0.8	0.1
l_2 data cost	0.8	0.2	0.9
	?	?	?
nodes	n_1	n_2	n_3

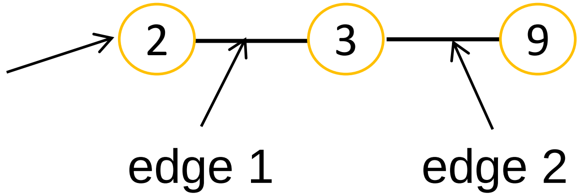
If we only assume "data cost" this is the optimal label solution (i.e. min energy)

l_1 data cost	$\boxed{0.4}$	0.8	$\boxed{0.1}$
l_2 data cost	0.6	$\boxed{0.2}$	0.9
	n_1	n_2	n_3

Minimum cost

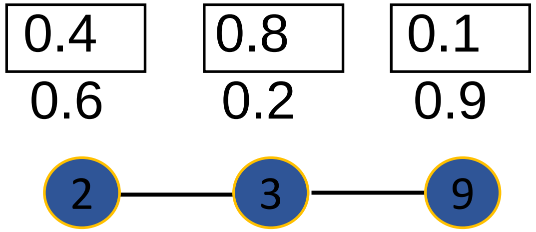
0.4 + 0.2 + 0.1 = 0.7

Assume these are the color values C at each node.

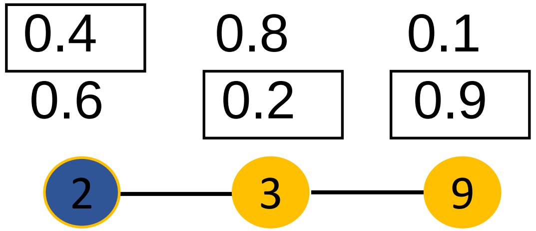


We now consider edge costs as shown in page 11.
New min energy is data_cost + edge_cost.
Where is the optimal label configuration? Compare with previous slide.

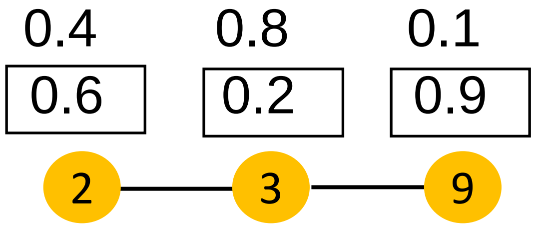
Some label configurations w/ edge cost



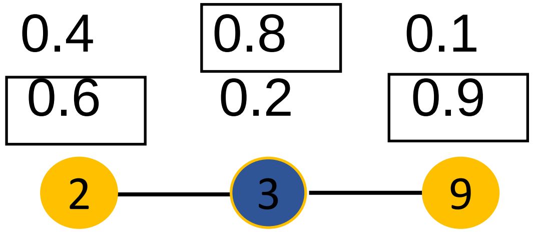
Datacost = 1.3
Edgecost = 0+0
Total = 1.3 + 0
1.3



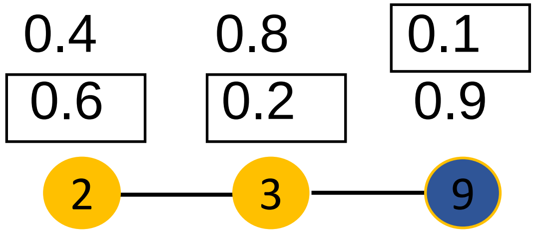
Datacost = 1.5
Edgecost = $1/[(2-3)^2+1]$ + 0
Total = 1.5+0.5
2.00



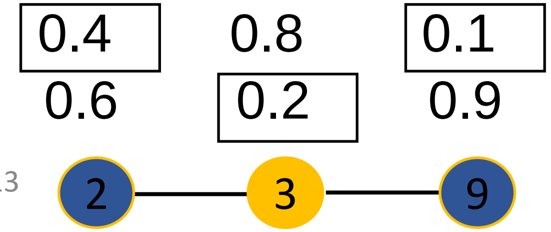
Datacost = 1.7
Edgecost = 0+0
Total = 1.7 + 0
1.7



Datacost = 2.3
Edgecost= $1/[(3-2)+1^2]$ + $1/[(9-3)^2+1]$
Total = 2.3 + 0.53
2.83



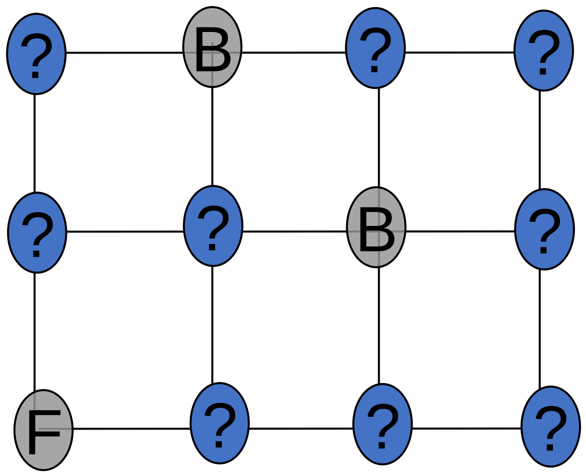
Datacost = 0.9
Edgecost = 0+ $1/[(9-3)^2+1]$
Total = 0.9+ 0.03
0.93



Datacost = 0.7
Edgecost = $1/[(2-1)^2]$ + $1/[(9-3)^2+1]$
Total = 0.7 + 0.53
1.73

Solving MRF

- Put all of these costs together and find the optimal labeling for the whole network



Remember, some points are already labeled (from markup), so they are fixed.

$$E(X) = \sum_{i \in \mathcal{V}} E_1(x_i) + \lambda \sum_{(i,j) \in \mathcal{E}} E_2(x_i, x_j)$$

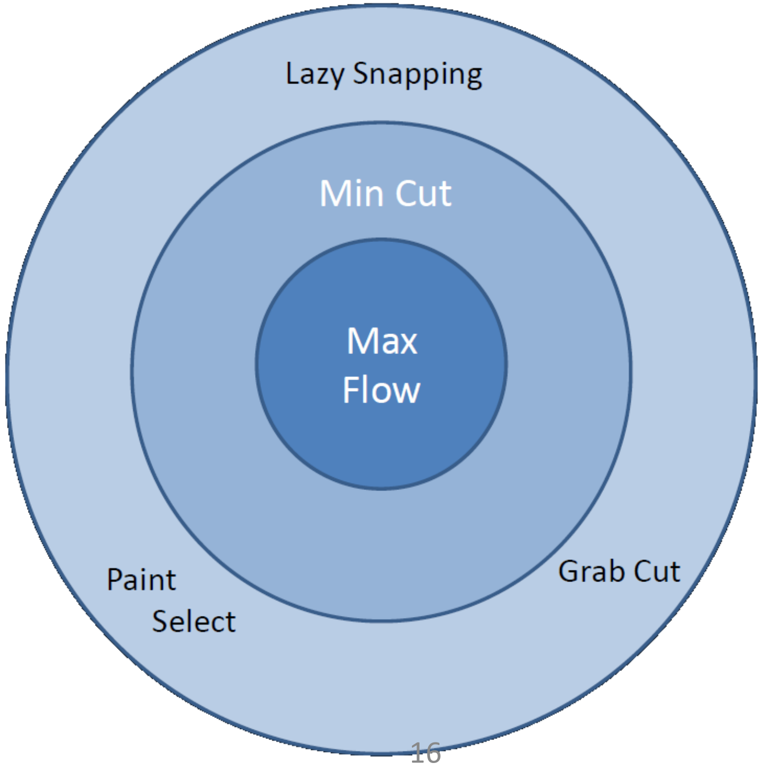
Solution is the label set that minimizes the cost function $E(X)$.

Solution is often an approximation.
Many approaches for minimizing $E(X)$.

Questions?

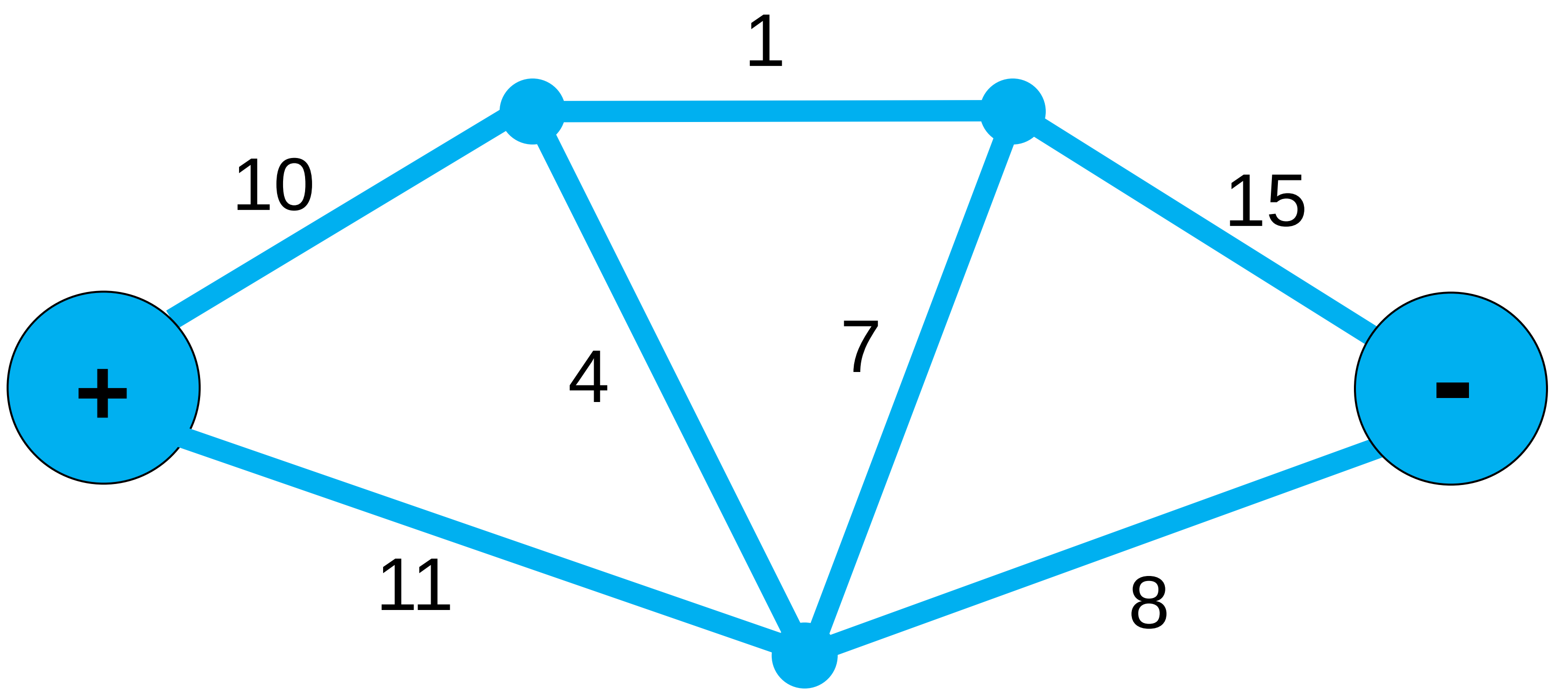
Min-cut & Max-flow

- We need to begin with the Max-flow problem
- Max-flow is mathematically equivalent to Min-cut
- The interactive segmentation can be formulated as a Min-cut problem



Max Flow

- Given a network of links of varying capacity, a source, and a sink, what is the maximum amount of total flow from the source to the sink??
  - Equivalently, how much flow along each edge?



Essentially a Linear Programming Problem

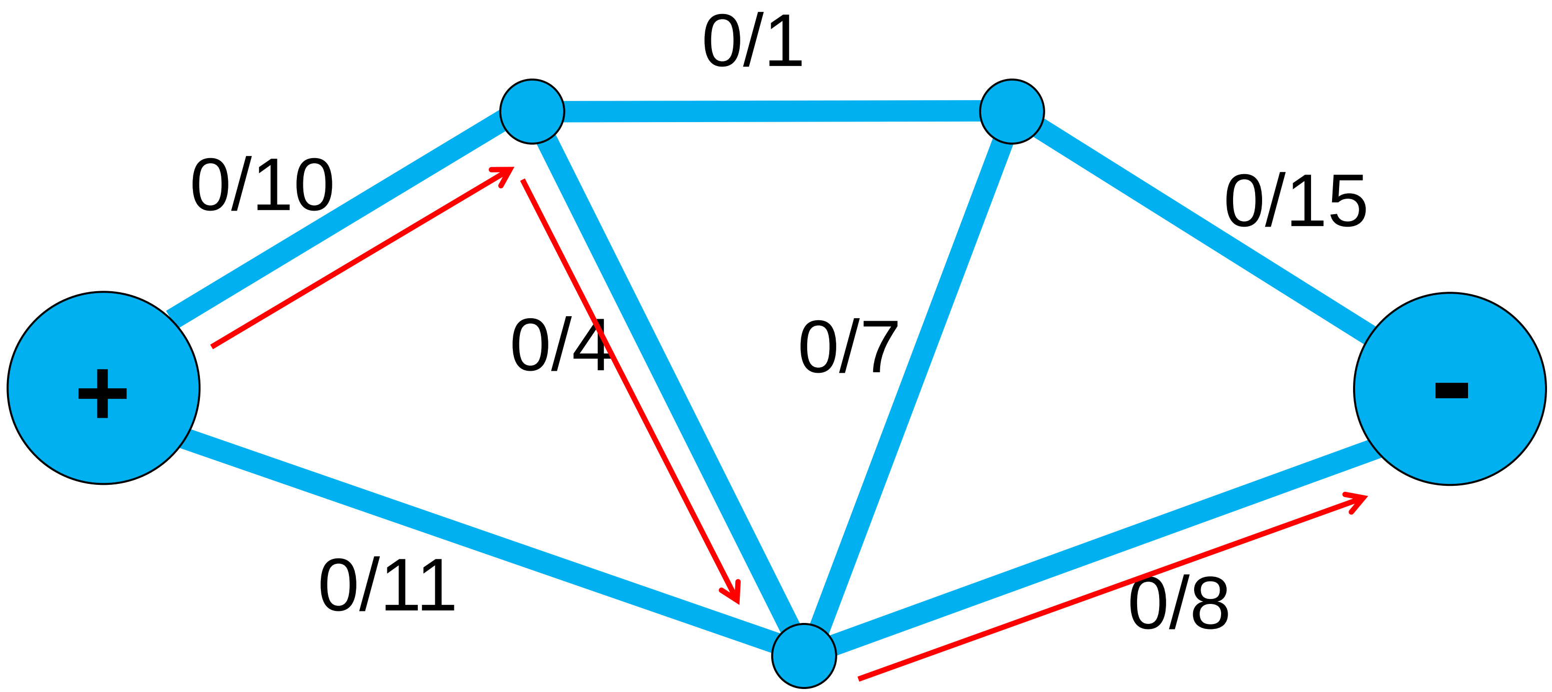
- One variable per edge (how much flow)
- One linear constraint per vertex
  - flow in = flow out
- Two inequalities per edge
  - 0 < flow < capacity
- One linear combination to maximize
  - Total flow leaving source
  - Equivalently, total flow arriving at sink

Essentially a Linear Programming Problem

- The optimal solution occurs at the boundary of some high-D simplex
  - Some variables reach their maximum value
  - The others are then determined by the linear constraints
- The Simplex method:
  - Start from some valid state
  - Find a way to increase one of the variables to its maximum value in an attempt to make the objective function better (here, to maximize the total flow)
  - Repeat until convergence

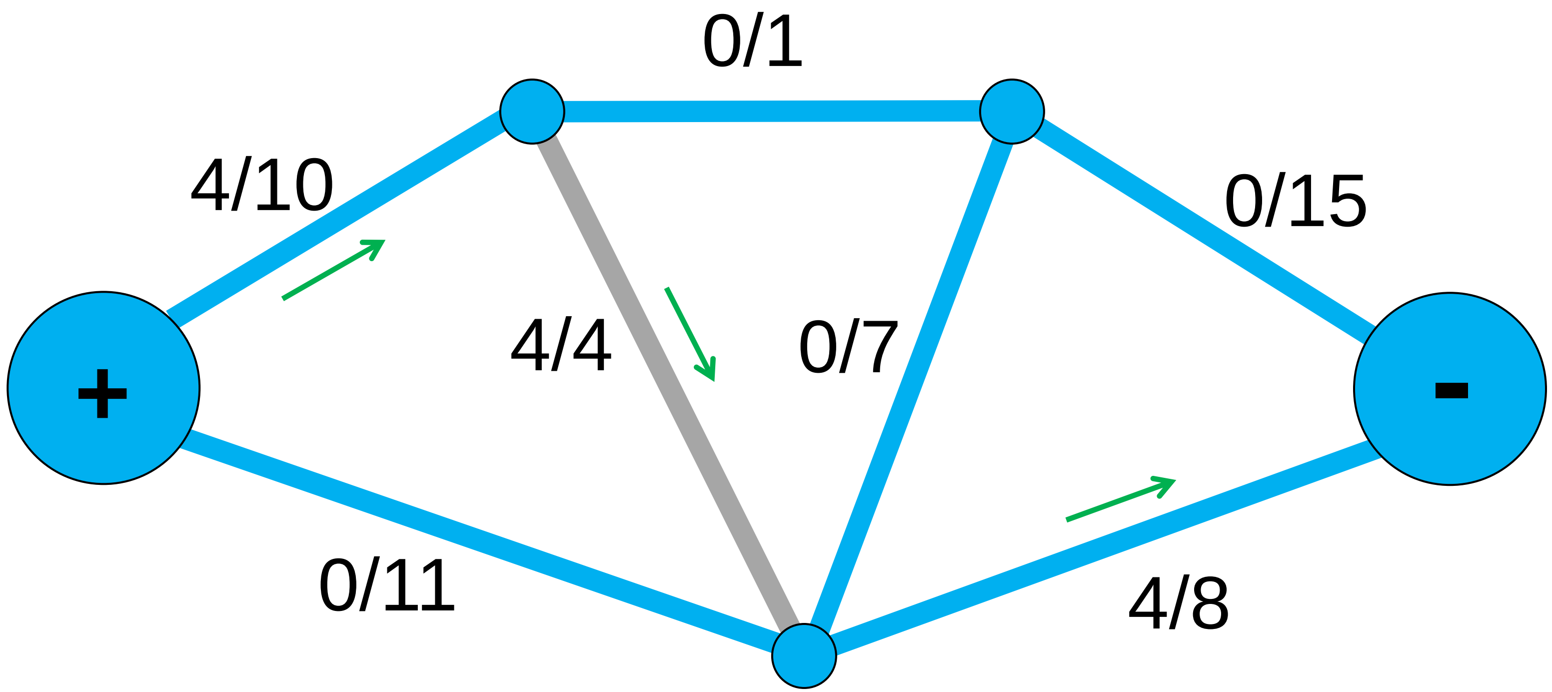
Basic Idea

- Start with no flow
- Find path from source to sink with capacity
  - Typically, by breadth-first search



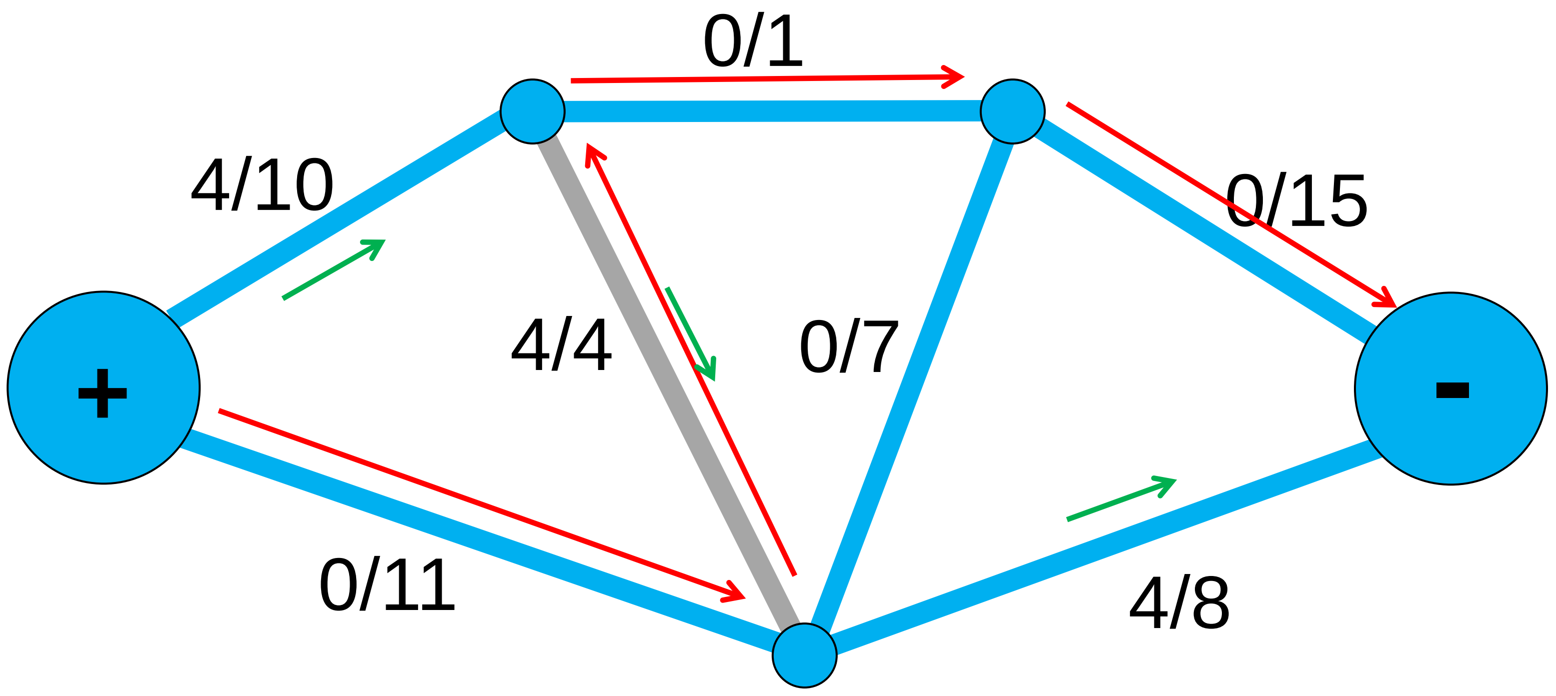
Basic Idea

- Increase the flow in that path to reach its maximum capacity
- Keep track of flow directions



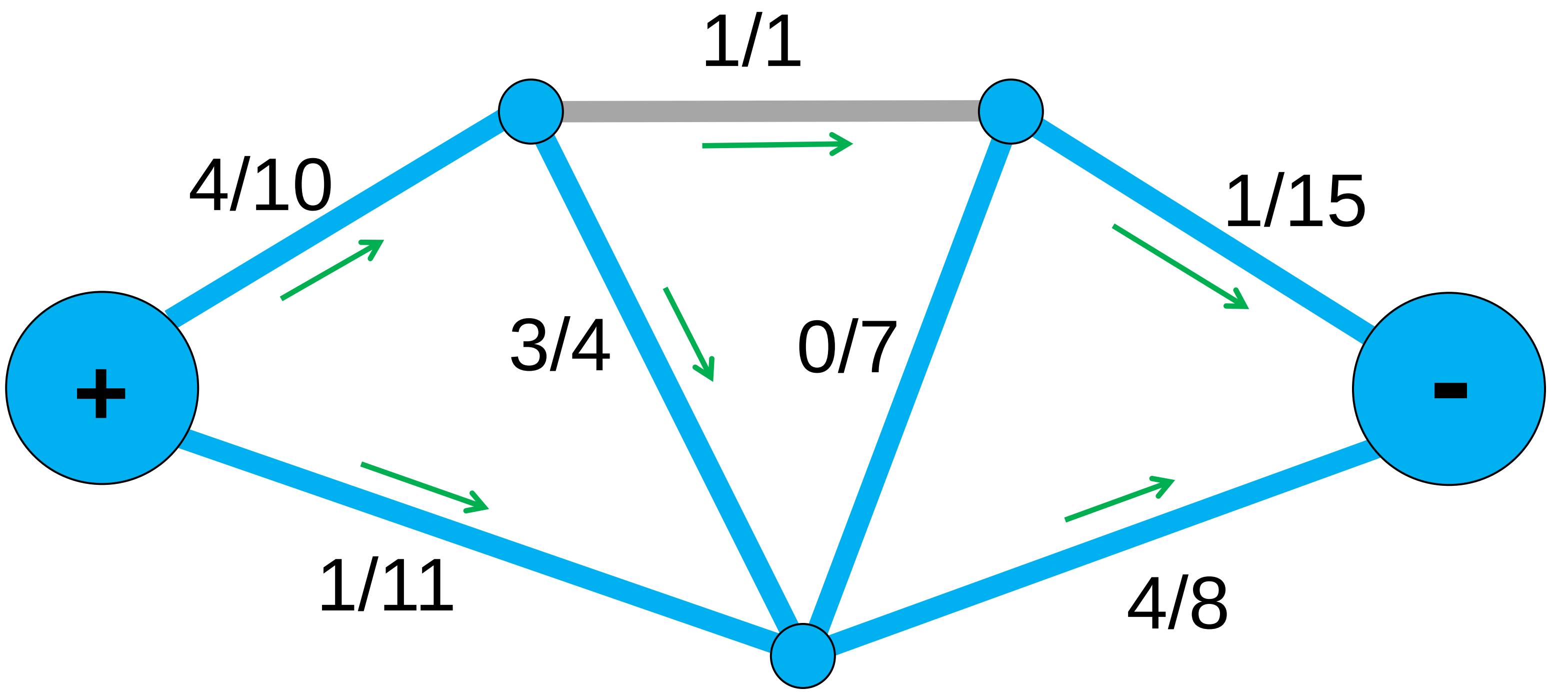
Repeat

- Find path from source to sink with capacity
  - E.g. by breadth first search

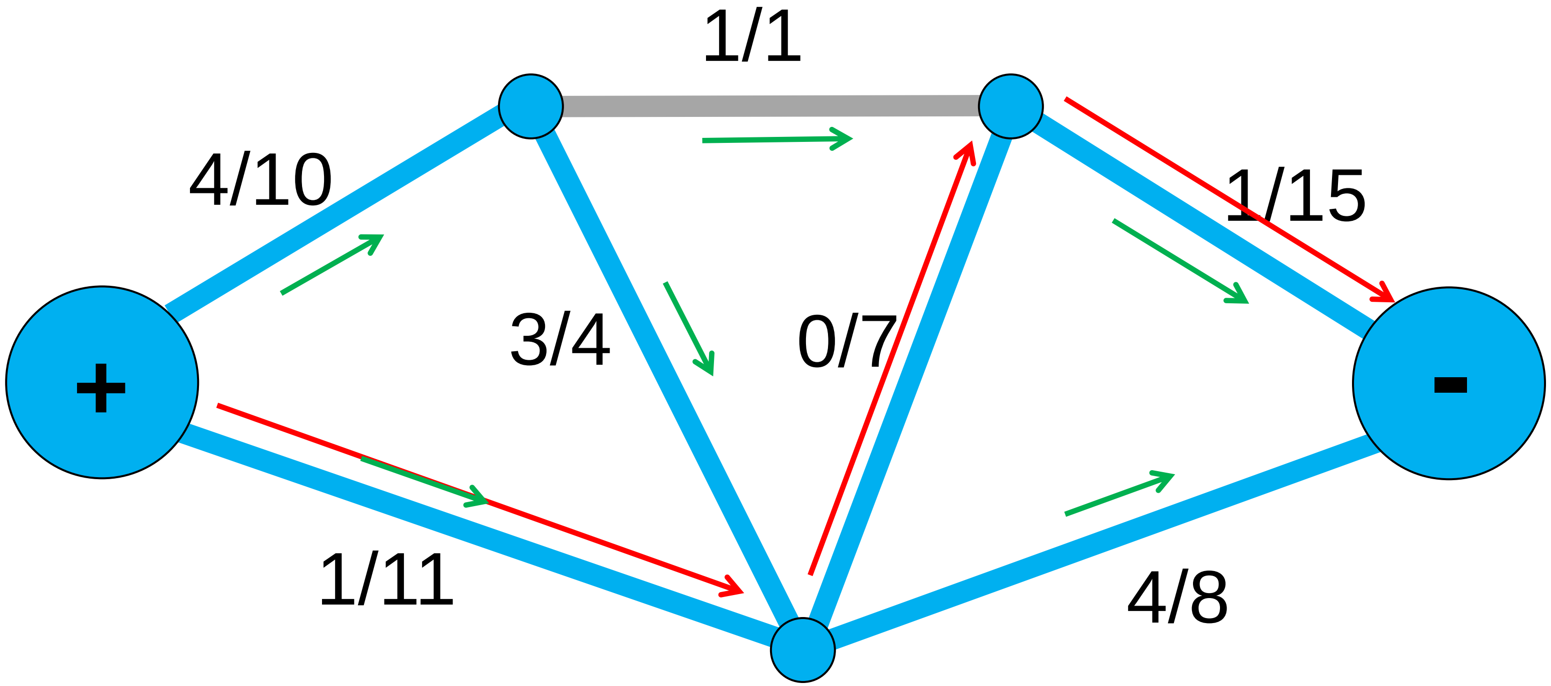


Repeat

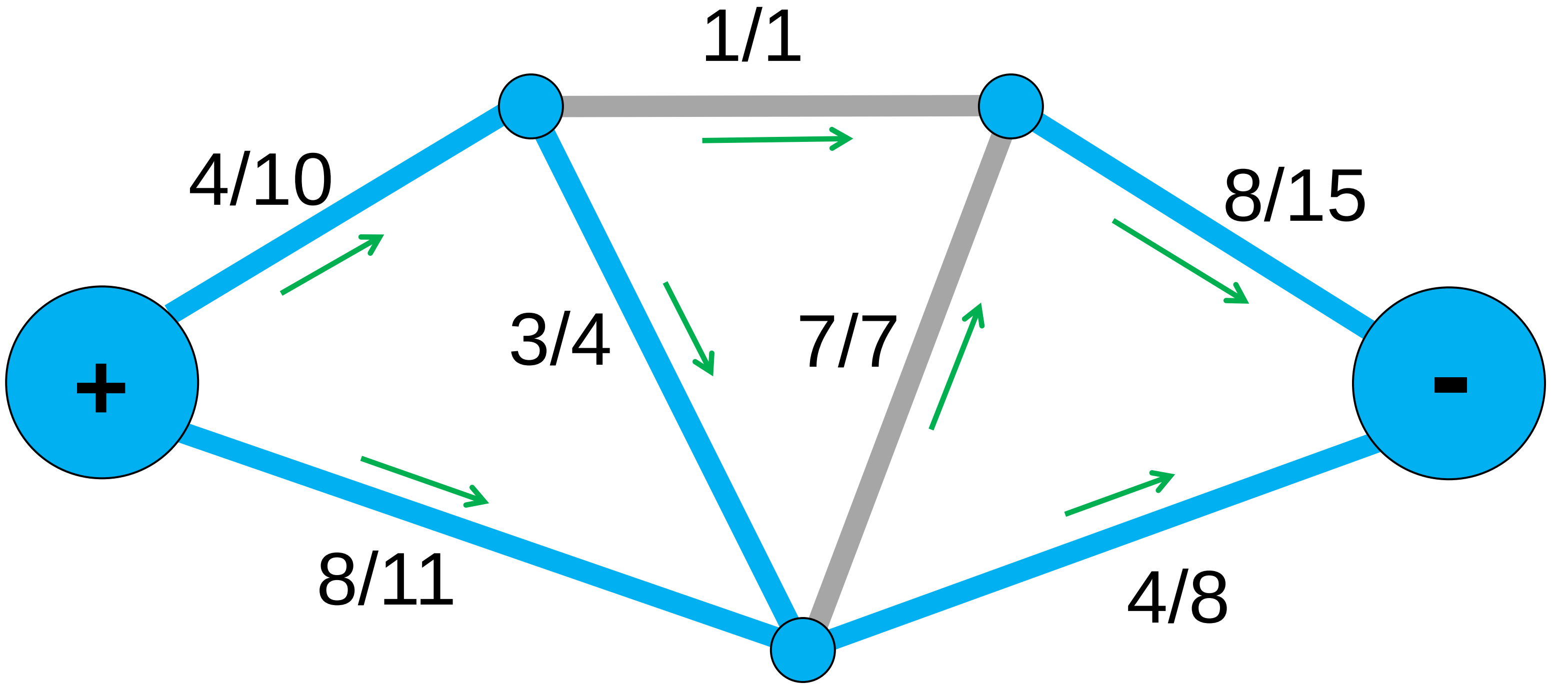
- Find path from source to sink with capacity
- Increase the flow in that path to reach its maximum capacity
- Keep track of flow directions



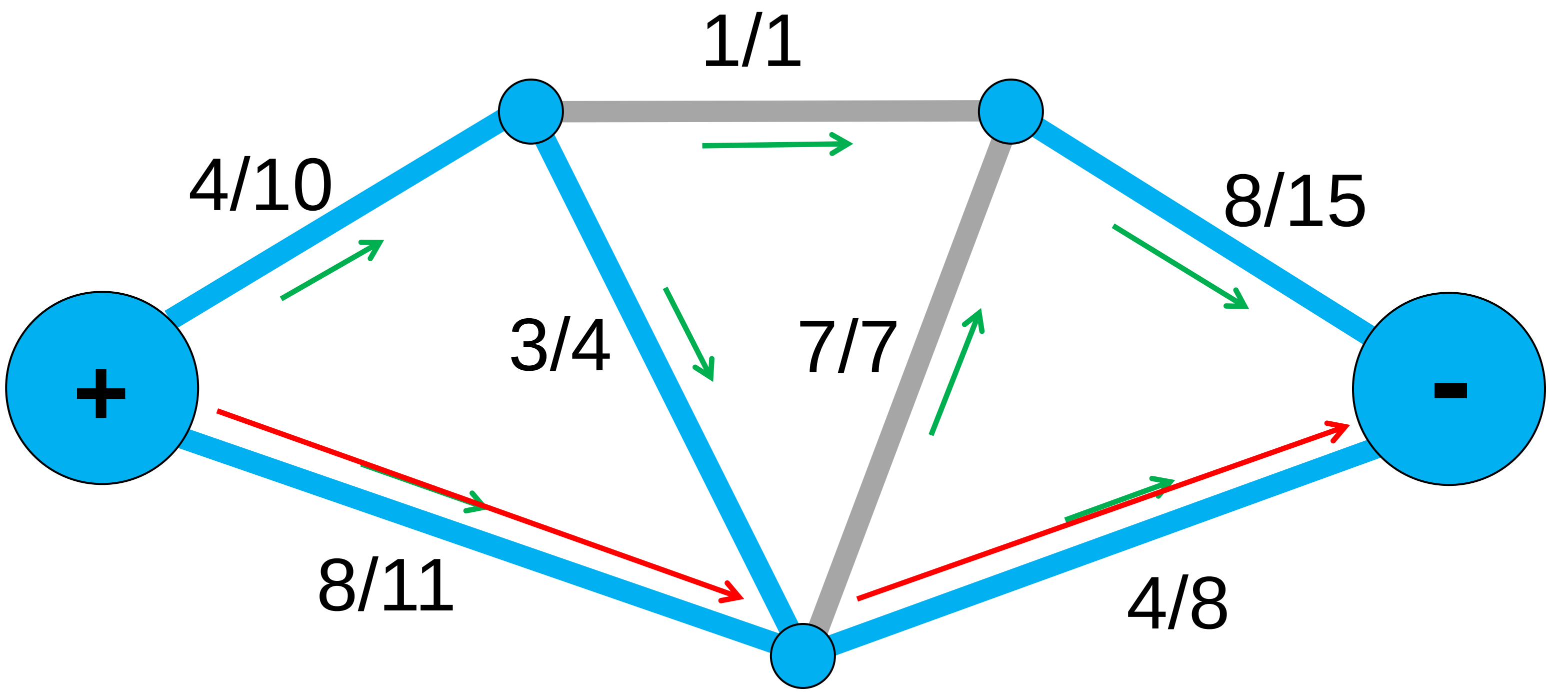
Continue...



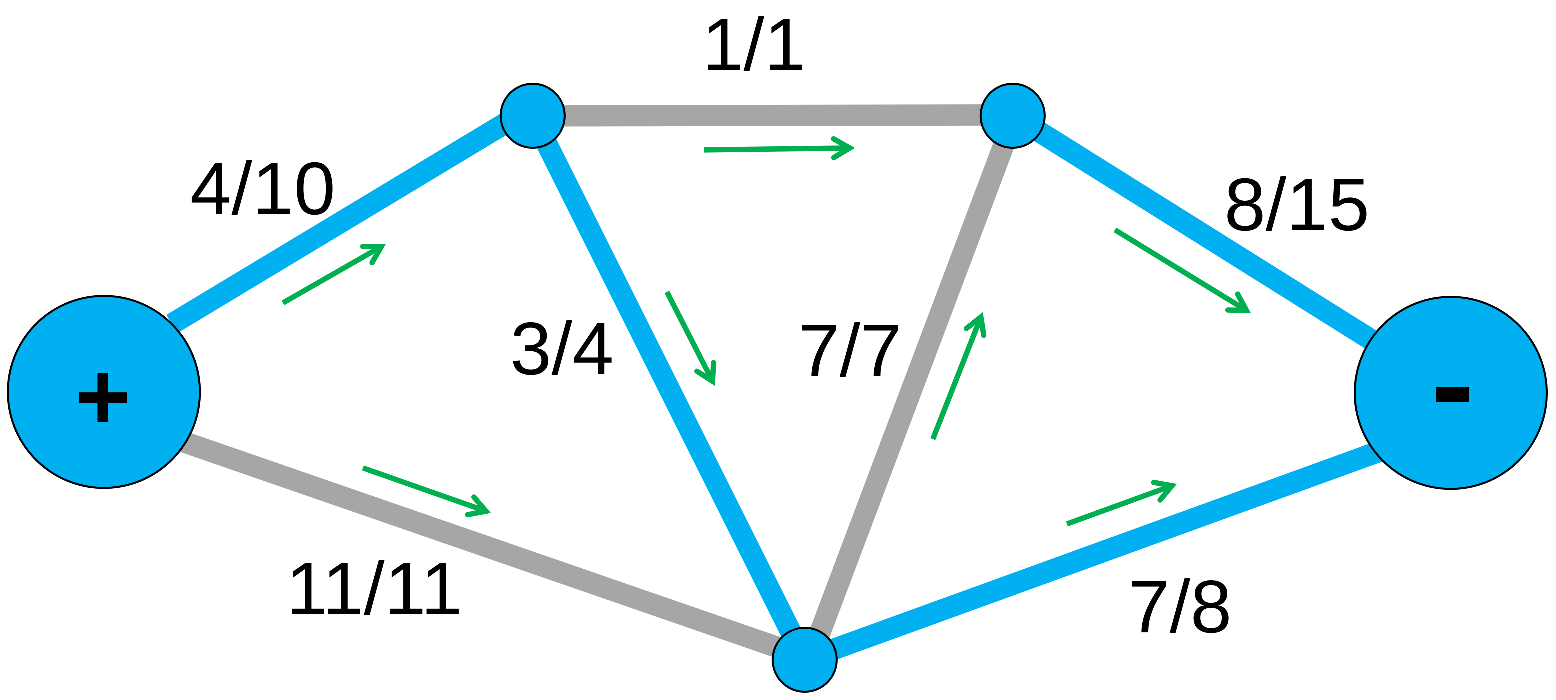
Continue...



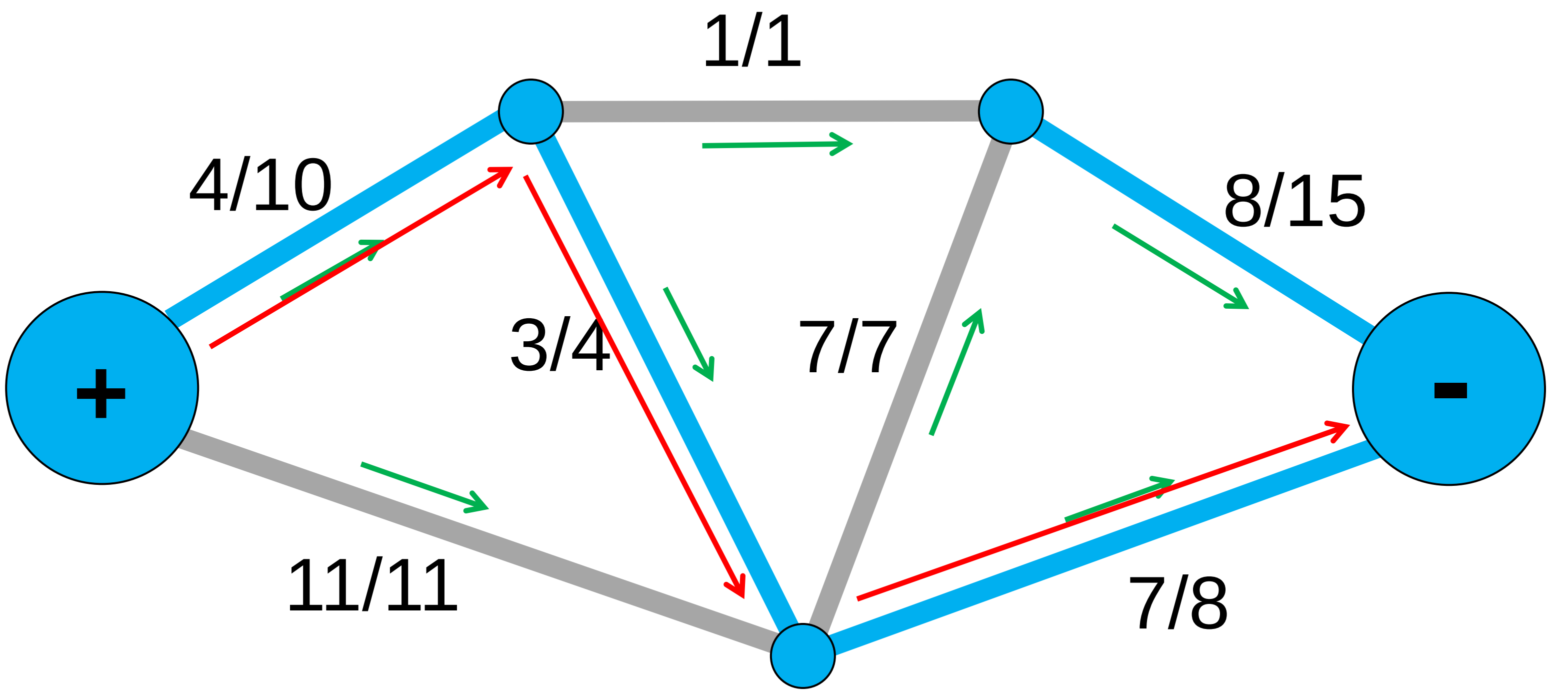
Continue…



Continue…

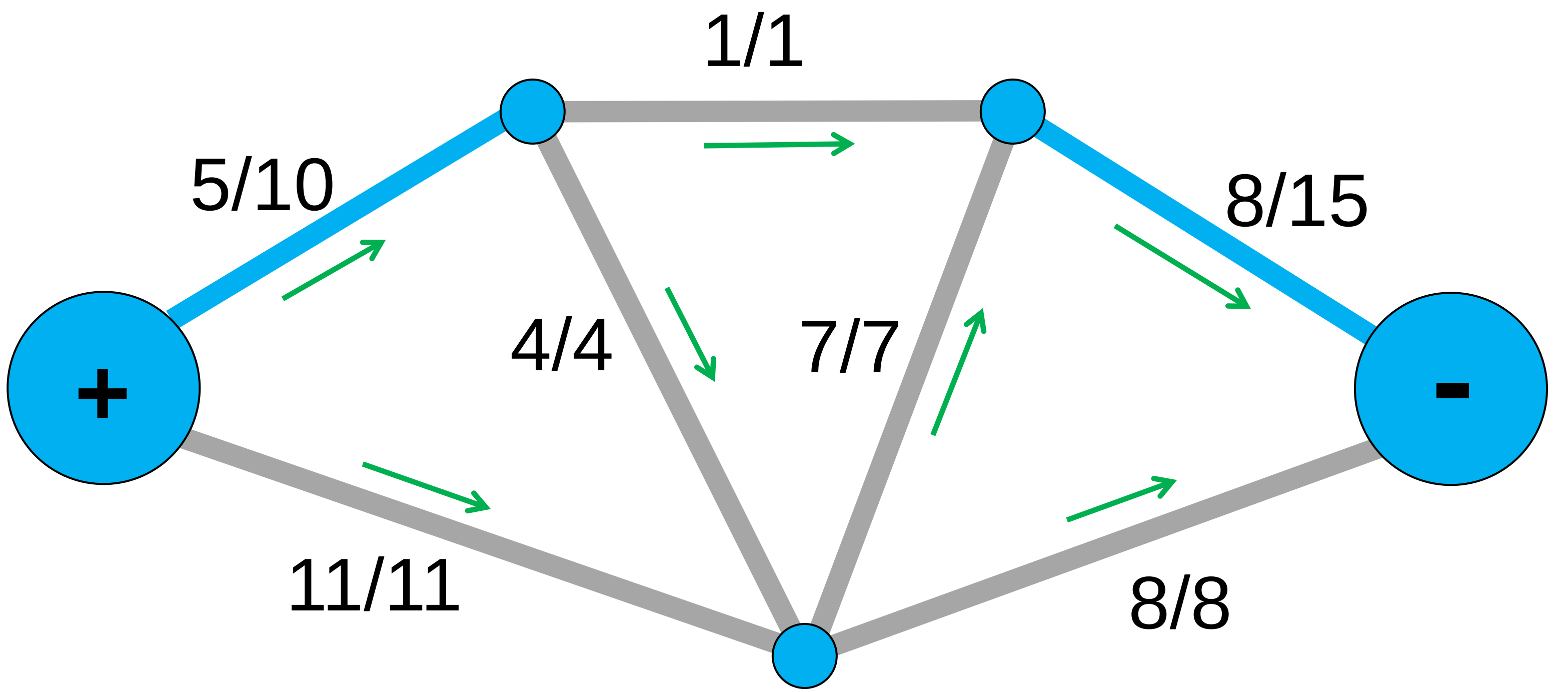


Continue…



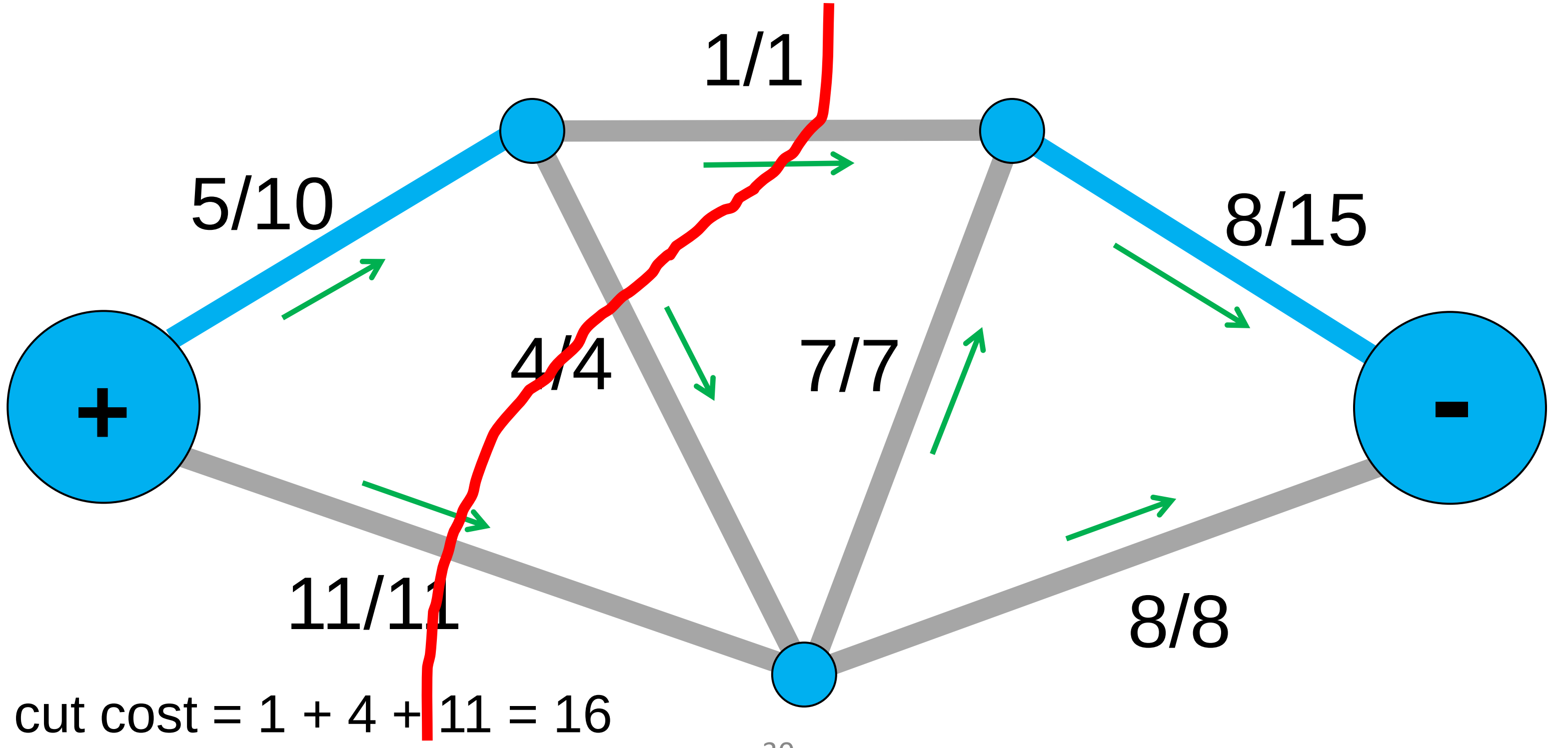
No path left. Done.

- The maximum amount of flow is 16



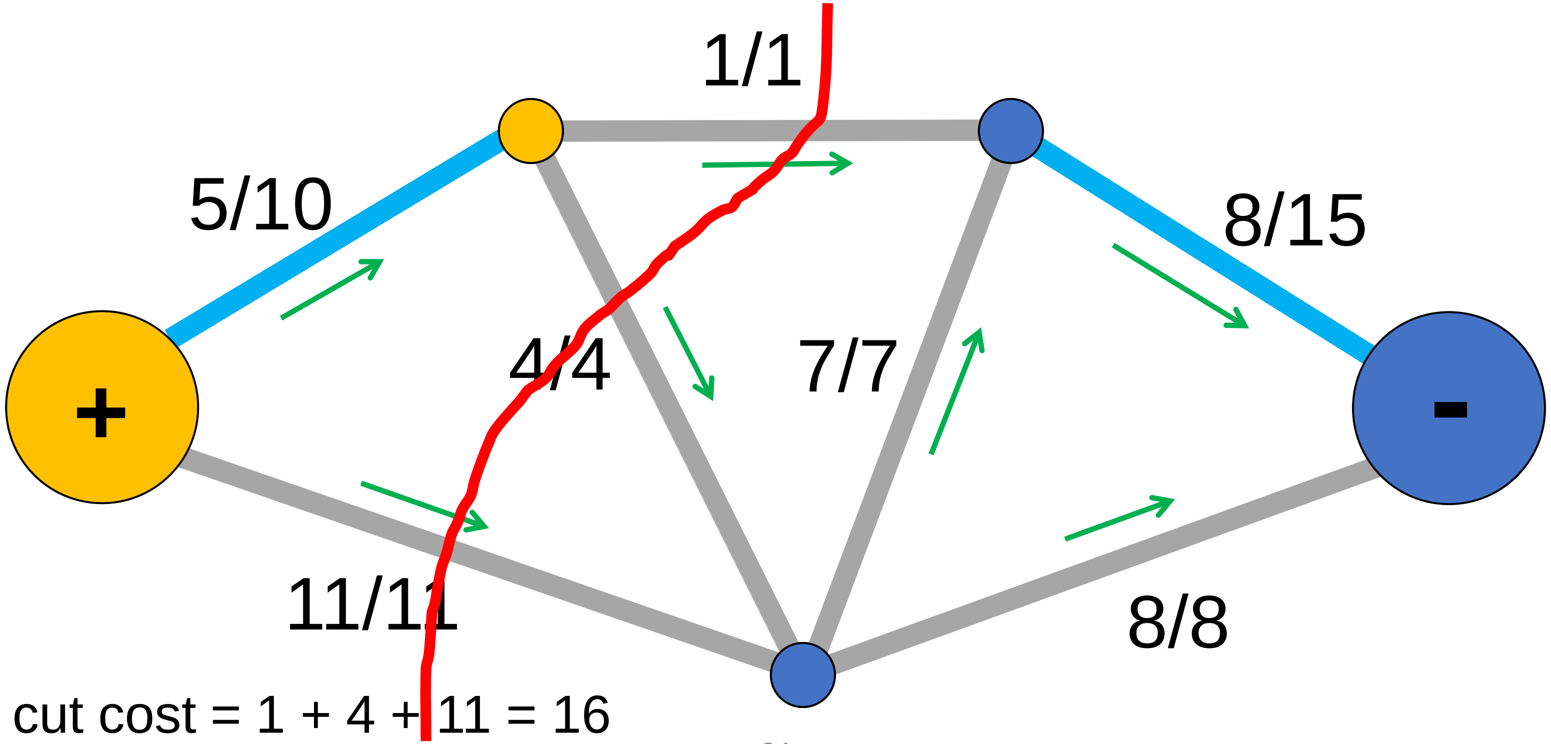
No path left. Done.

- Saturated edges represent the bottleneck
- Cutting across them breaks the graph into two pieces while removing the minimum amount of capacity



No path left. Done.

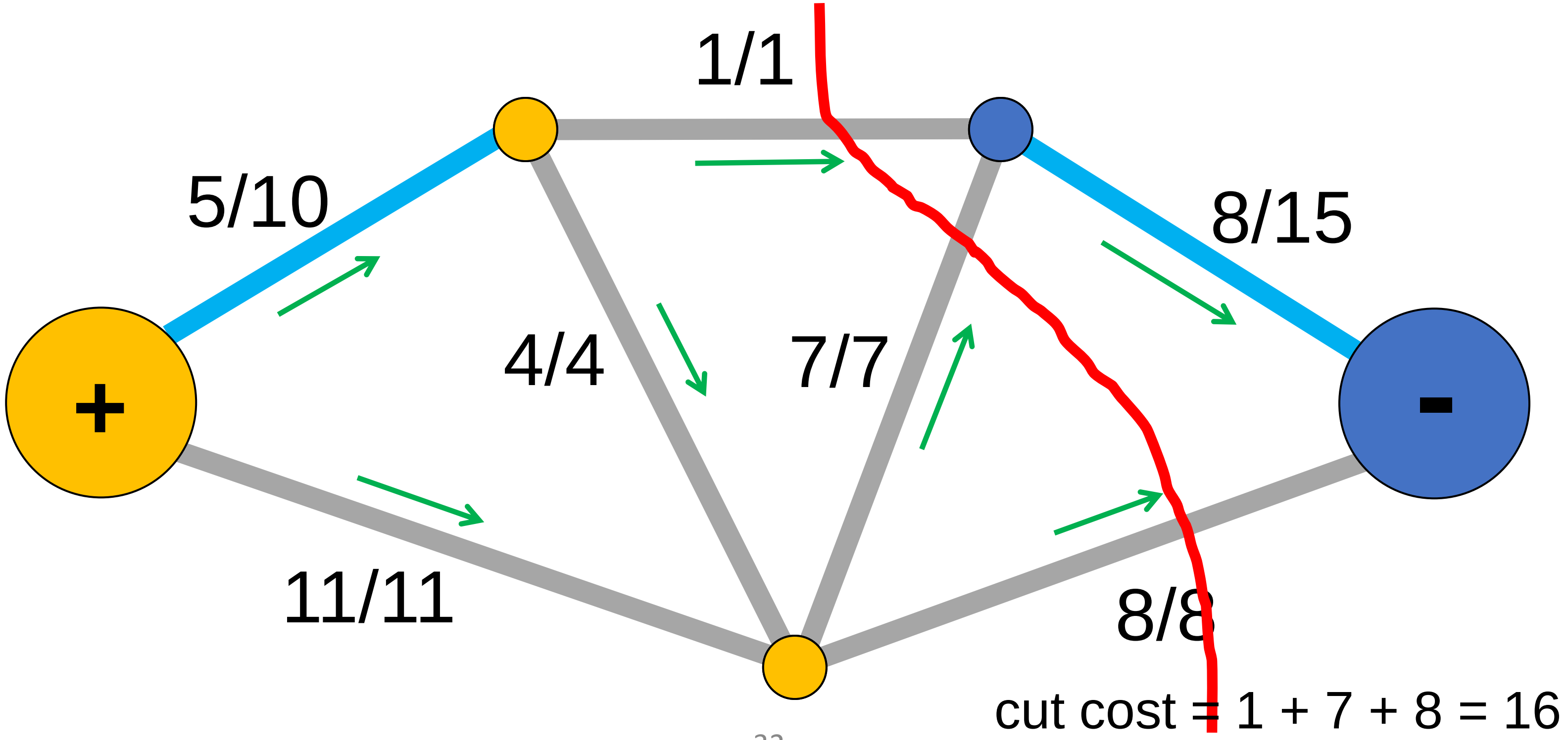
- All nodes connected to source form a group
- The others form another group



cut cost = 1 + 4 + 11 = 16

No path left. Done.

- All nodes connected to source form a group
- The others form another group



Questions?

Essentially a Linear Programming Problem

- Min-cut is the dual problem to Max-flow
- So optimizing max flow also optimizes Min-cut
- The breadth-first search for paths can be made more efficient for typical graphs in computer vision

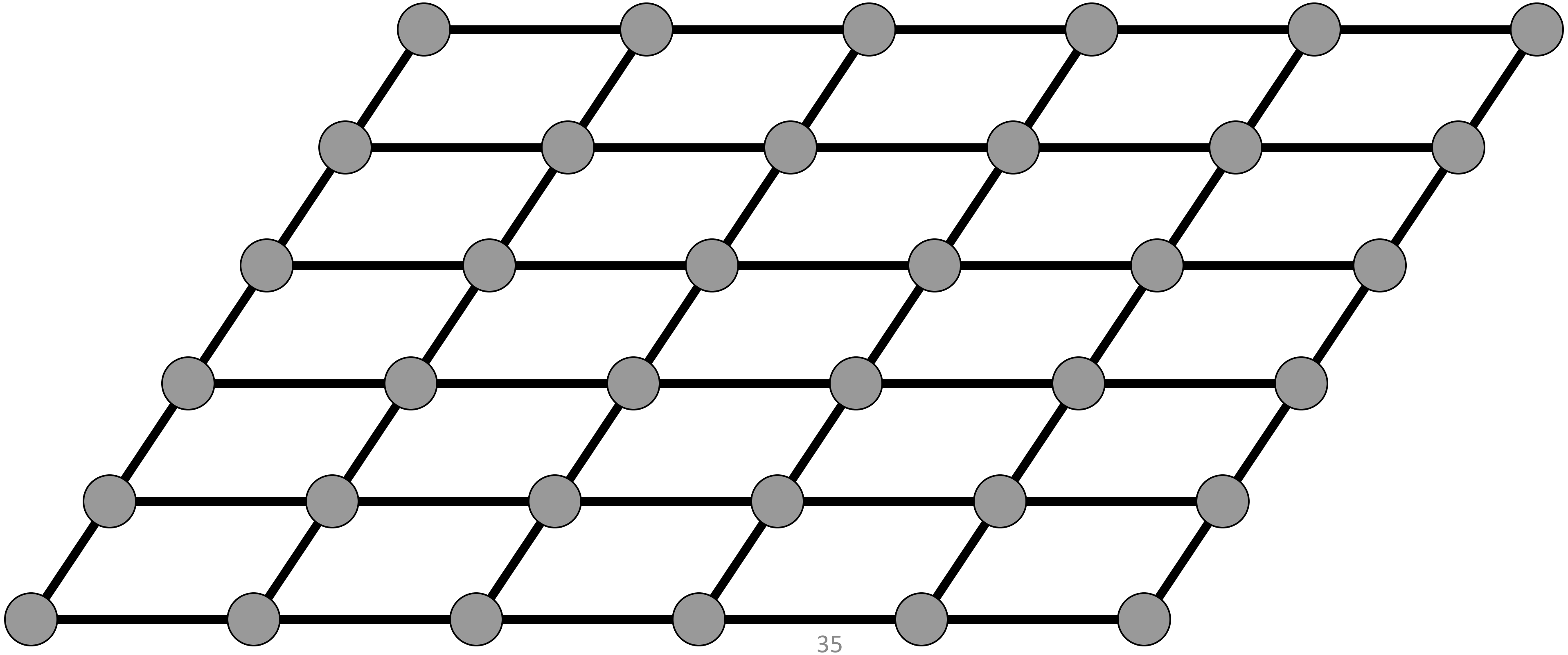
An Experimental Comparison of Min-Cut/Max-Flow Algorithms for Energy Minimization in Vision

Yuri Boykov and Vladimir Kolmogorov*

[PAMI 2004]

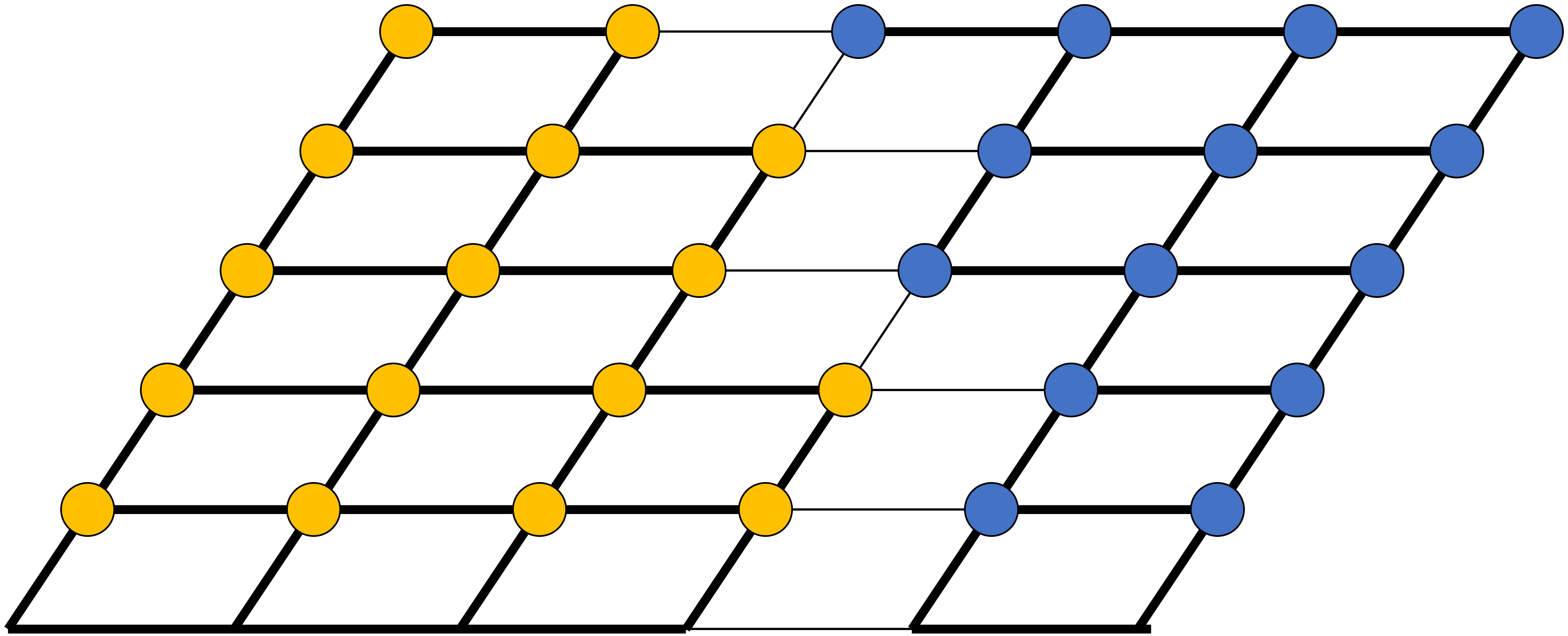
How does this relate to segmentation?

- Build a graph from pixels. 4 or 8-way connected.
  - Need to assign a 0 or 1 value to each vertex



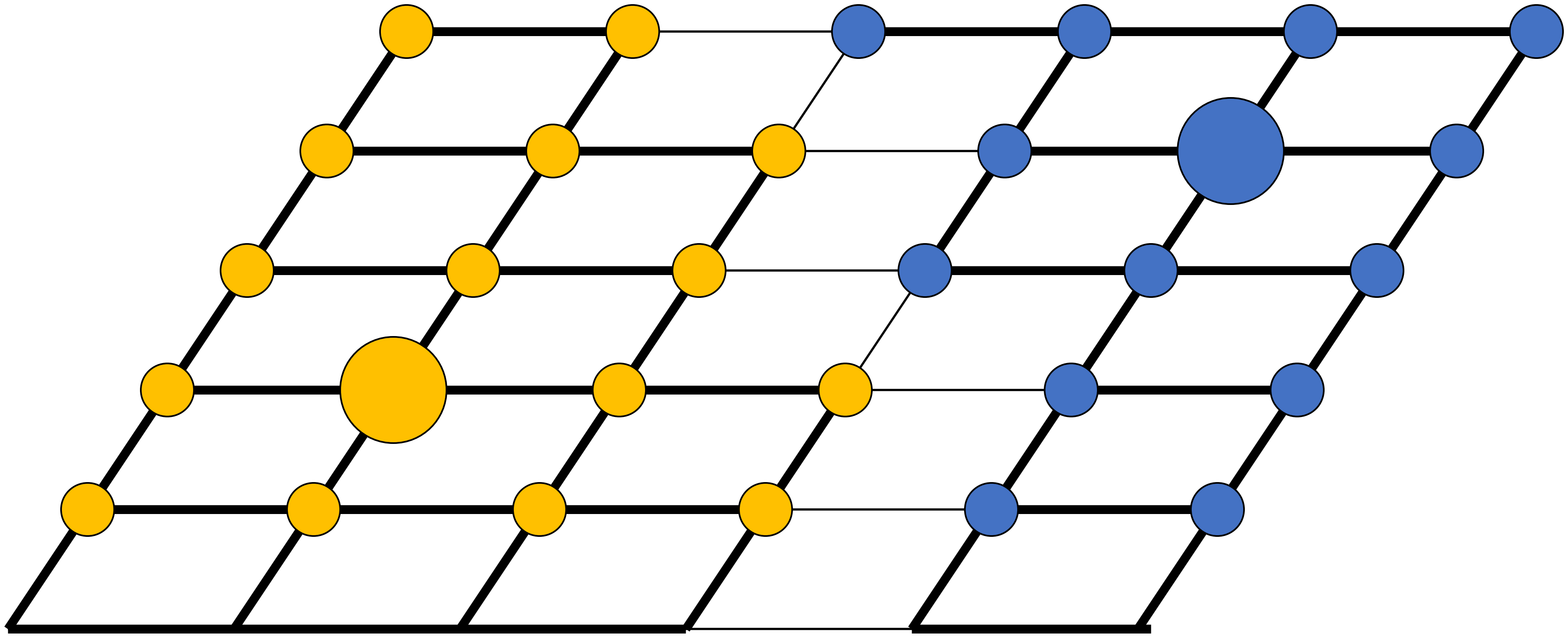
Foreground vs Background

- Edge capacity = Similarity of neighboring pixels
  - So we want to cut between dissimmilar pixels
  - Edge thickness indicates smoothness cost S_{pq}



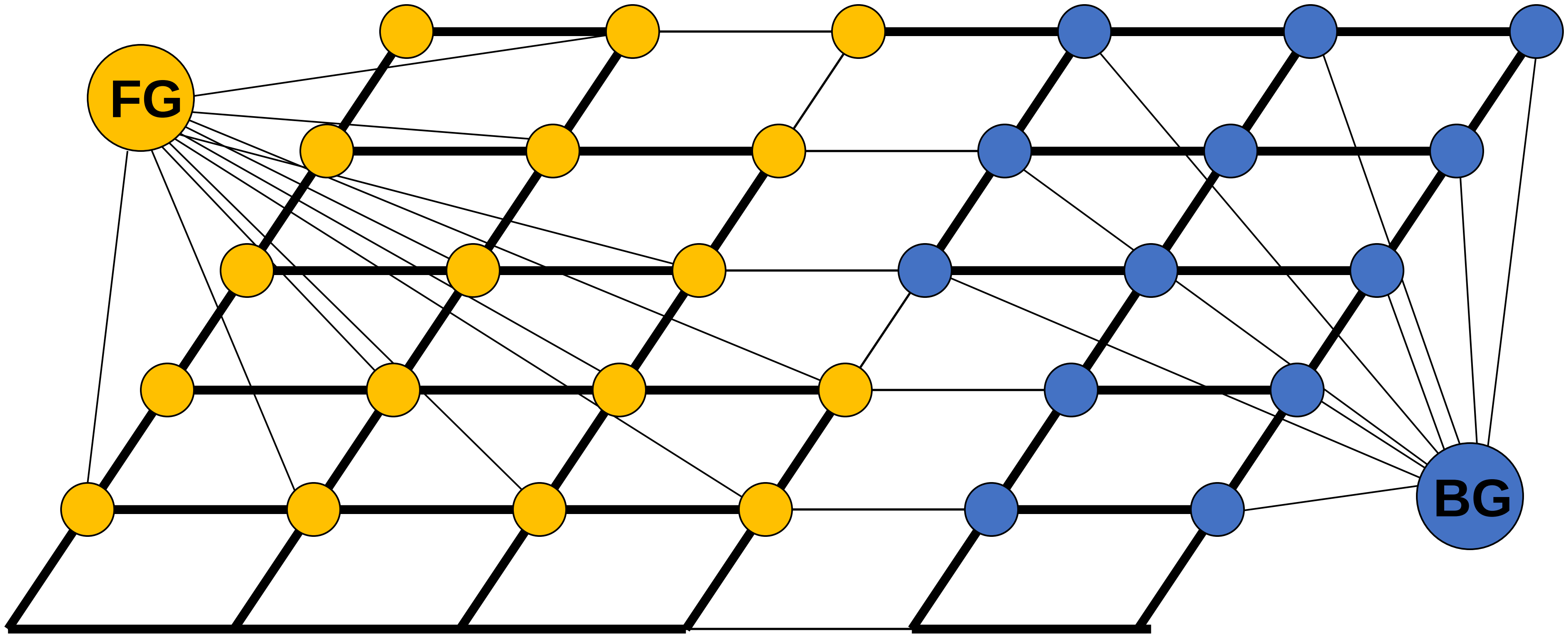
What are the source and sink?

- Option A: Pick two pixels, one as source the other as sink



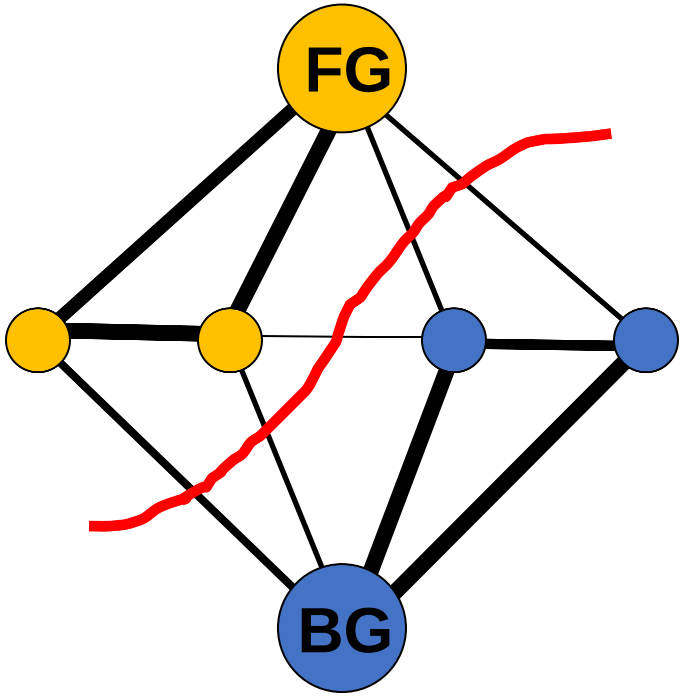
What are the source and sink?

- Option B (better): Add two additional nodes representing the foreground and background
- Connect them with different capacities to pixels belong to FG or BG



1D Case (to simplify the figure)

- Edges between pixels
  - Capacity = likelihood (or -cost) that they belong to the same group
- Edges from FG to pixels
  - Capacity = likelihood (or -cost) that they belong to FG
- Edges from BG to pixels
  - Capacity = likelihood (or -cost) that they belong to BG
- The Min-cut leaves each pixel either connected to the FG node or the BG node

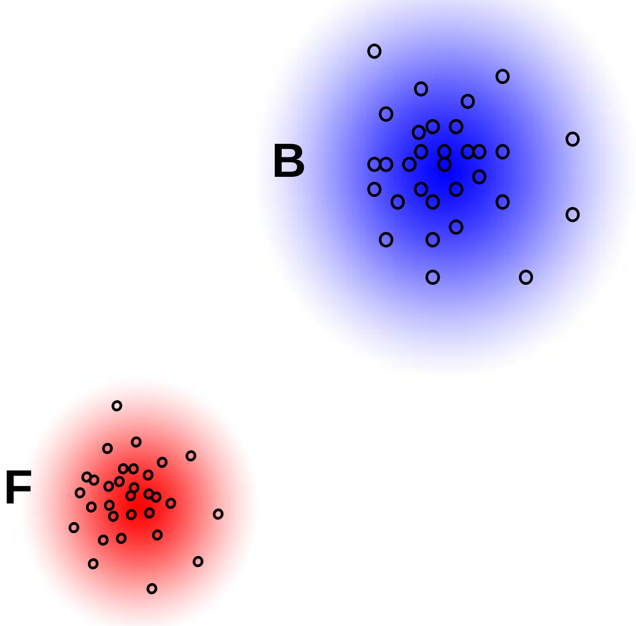


Likelihood/Cost of FG and BG

- How likely is the foreground to have color F? the background to have color B?
- Fit a Gaussian (or Gaussian Mixture) models in RGB space based on pixel color from user strokes
- The likelihood is computed according to the distance between the color and the Gaussian centers

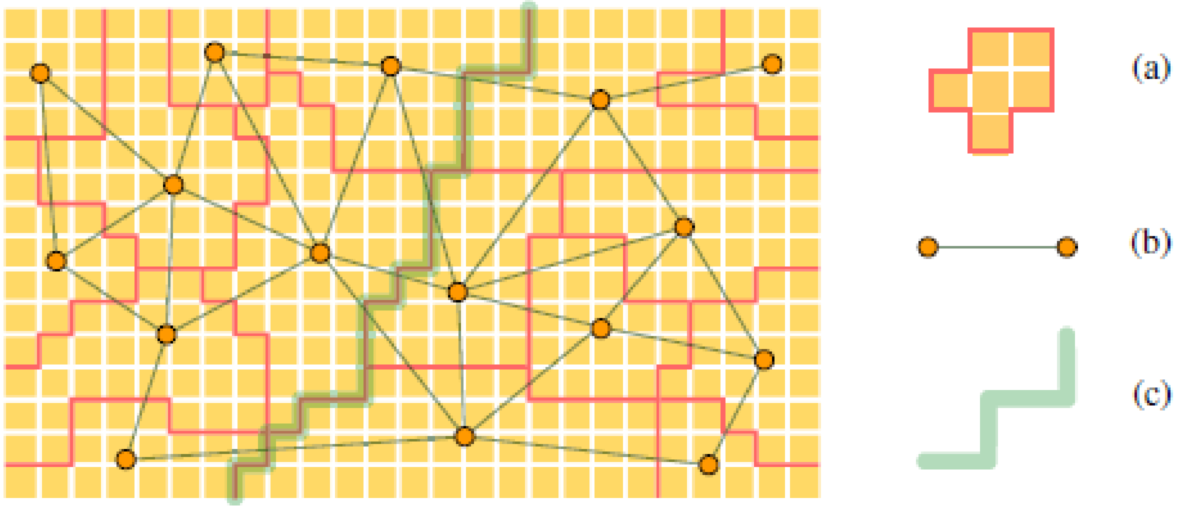
$$P_x = \sum_n \omega_n \exp(-|C_x - K_n|2)$$

K_n is the n-th Gaussian center, ω_n is the proportion of marked pixels that belong to the n-th center.

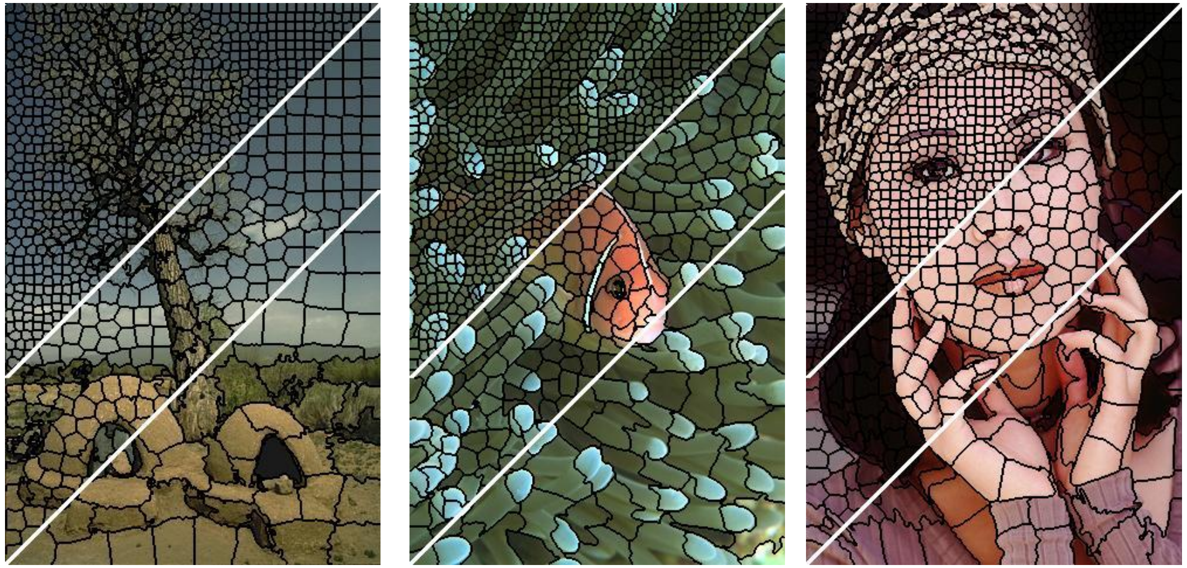


Speedup Strategies

- Apply a pre-segmentation (over-segmentation) to the image
- Break image into super-pixels
- Then group those super-pixels into different segments



SLIC Superpixels



Results

Interactive Segmentation Summary

- Scribble based segmentation
  - Very fast and intuitive
- Developed by Microsoft
  - Photoshop developed 'Quick Selection' later
- Very popular in research papers
  - Easy to implement

Siggraph 2004

Lazy Snapping

[†]Yin Li* [‡]Jian Sun [†]Chi-Keung Tang [‡]Heung-Yeung Shum

[†]Hong Kong University of Science and Technology [‡]Microsoft Research Asia

Questions?

Binary Graph-Cut Optimization

- Minimize an objective function defined on a graph

$$\boldsymbol{E}(\boldsymbol{X}) = \sum_{\boldsymbol{p}\in\boldsymbol{V}} \boldsymbol{D}_{\boldsymbol{p}}(\boldsymbol{x}_{\boldsymbol{p}}) + \sum_{(\boldsymbol{p},\boldsymbol{q})\in\boldsymbol{E}} \boldsymbol{S}_{\boldsymbol{pq}}(\boldsymbol{x}_{\boldsymbol{p}}, \boldsymbol{x}_{\boldsymbol{q}})$$

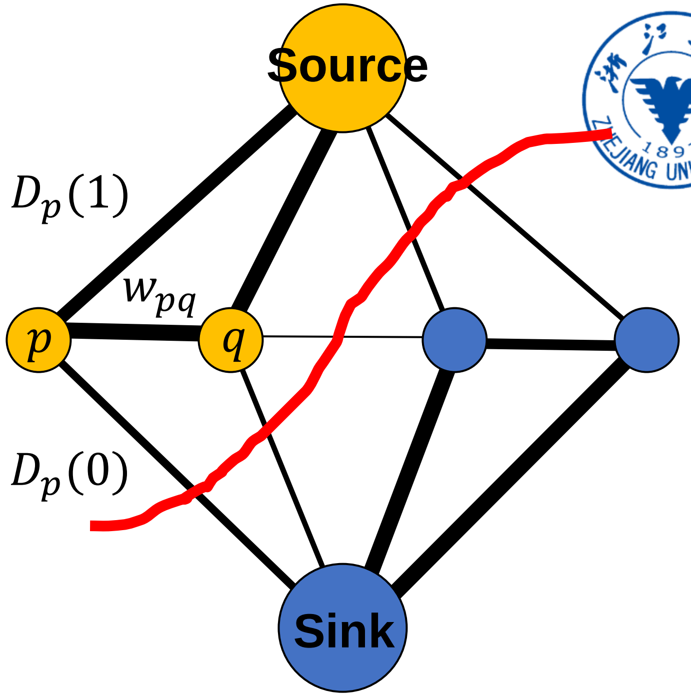
$X = \{x_1, x_2, \cdots, x_N\}, x_p = \{0,1\}$,
V, E are the set of vertices and edges of a graph
$D_p(\cdot)$ and $S_{pq}(\cdot)$ are functions defined on vertices and edges

- We begin with the binary problem with Potts model:

$$S_{pq}(x_p, x_q) = w_{pq}\delta(|x_p - x_q|) = \begin{cases} w_{pq} & if \quad x_p \neq x_q \\ 0 & otherwise \end{cases}$$

- It is possible to define edge weights to solve the minimization by min-cut

Binary Graph-Cut Optimization



- Set edge weights
  - Set weight between p and q as w_{pq}
  - Set weight between p and source as $D_p(1)$
  - Set weight between p and sink as $D_p(0)$
- Effectively:
  - $x_p = 1$ means p is assigned to source (the edge between p and sink is cut)
  - $x_p = 0$ means p is assigned to sink (the edge between p and source is cut)
  - When $x_p \neq x_q$, the edge between x_p and x_q is cut
- Any configuration of X corresponds to a cut
  - So min-cut minimizes $E(X)$

Binary Graph-Cut Optimization

- More general result is provided in the following paper

What Energy Functions Can Be Minimized via Graph Cuts?

[PAMI 2004]

Vladimir Kolmogorov, *Member, IEEE*, and Ramin Zabih, *Member, IEEE*

- Given a binary function, $x_p \in \{0, 1\}$

$$E(X) = \sum_{p \in V} D_p(x_p) + \sum_{(p,q) \in E} S_{pq}(x_p, x_q)$$

Graph-cut can find the GLOBAL minimum of E iff

$$S_{pq}(0,0) + S_{pq}(1,1) < S_{pq}(0,1) + S_{pq}(1,0)$$

What if x_p is not binary?

- What if x_p is not binary, e.g. $x_p \in \{1,2,\ldots,n\}$?

- The basic idea: convert this problem to a binary one
- Start from an initial configuration, and iteratively improve the result
  - Two possible solutions:
  - Alpha-expansion and Alpha-beta swap
  - Both methods improve the result by solving a binary graph-cut problem at each iteration
  - Converge to a LOCAL minimum

Multi-lable Graph-Cut

- Start from an initial configuration
  - Alpha-expansion: pick any statue “alpha” and decide if the statues at a vertex should change to “alpha” or keep unchanged
  - This is a binary problem. We can define a binary parameter y_p at each vertex, where y_p=1 (or 0) means change to “alpha” (or not).
  - Then we can obtain an optimal y_p at each vertex by the binary graph-cut algorithm.

  - Alpha-beta swap: pick any two statues “alpha” and “beta” and decide if we should swap “alpha (or beta)” for “beta (or alpha)” at each vertex
  - This is a binary problem. We can define a binary parameter y_p at each vertex, where y_p = 1 (or 0) means swap (or not).
  - Then we can obtain an optimal y_p at each vertex by the binary graph-cut algorithm

Generally, alpha-expansion outperforms alpha-beta swap

Multi-lable Graph-Cut

Alpha-expansion

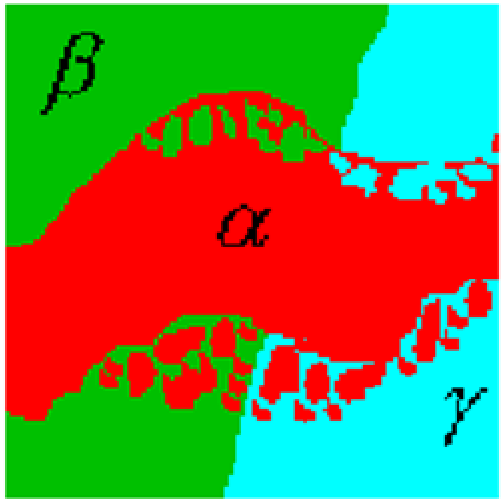
- 1. Start with an arbitrary labeling f
- 2. Set success := 0
- 3. For each label $\alpha \in \mathcal{L}$
- 3.1. Find $\hat{f} \in \operatorname{argmin} E(f')$ among f' within one α-expansion of f
- 3.2. If $E(\hat{f}) < E(f)$, set $f \ := \hat{f}$ and success := 1
- 4. If success = 1 goto 2
- 5. Return f

Multi-lable Graph-Cut

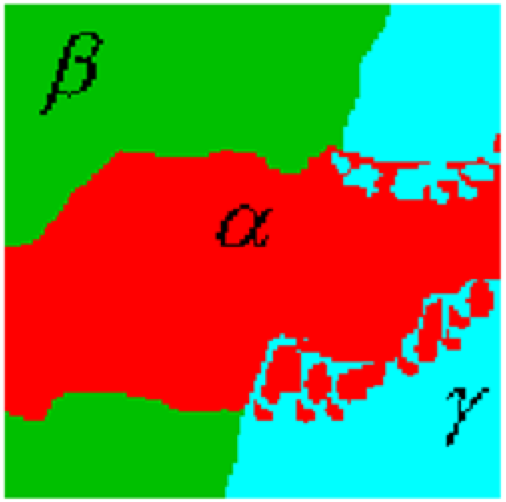
Alpha-beta swap

- 1. Start with an arbitrary labeling f
- 2. Set success := 0
- 3. For each label $\{\alpha, \beta\} \in \mathcal{L}$
- 3.1. Find $\hat{f} \in \operatorname{argmin} E(f')$ among f' within one $\alpha - \beta$ swap of f
- 3.2. If $E(\hat{f}) < E(f)$, set $f \;\coloneqq \hat{f}$ and success := 1
- 4. If success = 1 goto 2
- 5. Return f

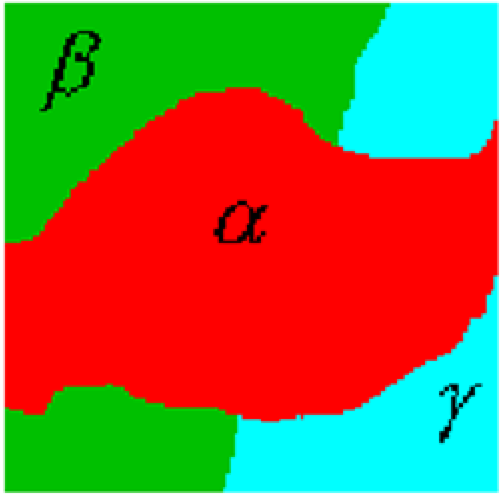
Multi-lable Graph-Cut



initial labeling



α-β-swap



α-expansion

Summary of big ideas

- Treat image as a graph
  - Pixels are nodes
  - Between-pixel edge weights based on color difference
  - Per-pixel weights for affinity to foreground/background
- Good regions are produced by a low-cost cut (GrabCuts, Graph Cut Belief Propagation, etc)

Questions?

Grab Cuts

"GrabCut" — Interactive Foreground Extraction using Iterated Graph Cuts

Carsten Rother* Vladimir Kolmogorov† Andrew Blake‡
Microsoft Research Cambridge, UK

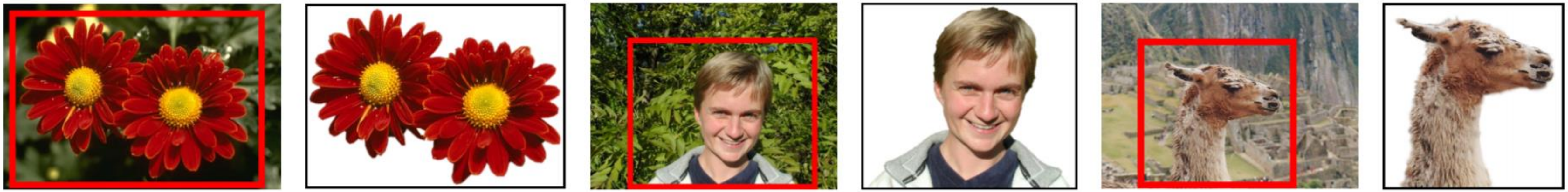
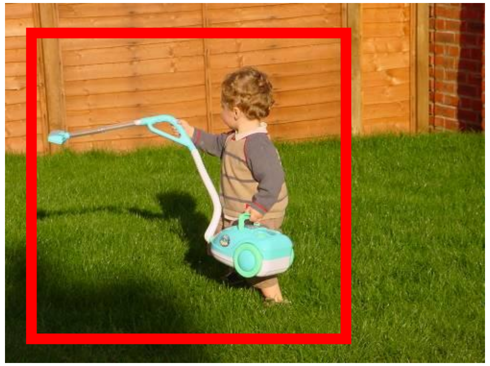


Figure 1: **Three examples of GrabCut .** The user drags a rectangle loosely around an object. The object is then extracted automatically.

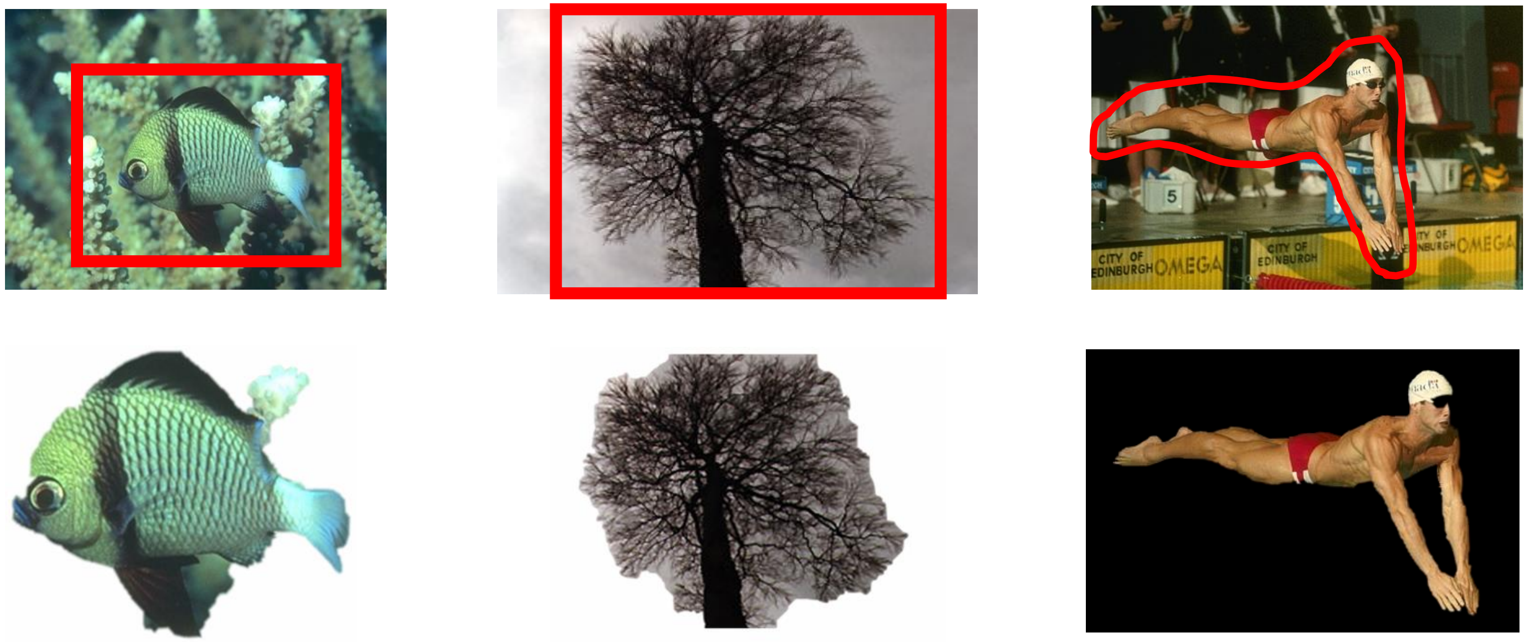
SIGGRAPH 2004

what is easy or hard for graphcut-based segmentation?



easier examples

more difficult examples



What about More General Energies?

- Graph-cut generally produces strong results, but on limited energy functions
- Switch to BP (belief-propagation), TRW (tree-reweighted message passing) for more general energies
  - But also slightly worse results
  - No regularization condition

Convergent Tree-reweighted Message Passing for Energy Minimization

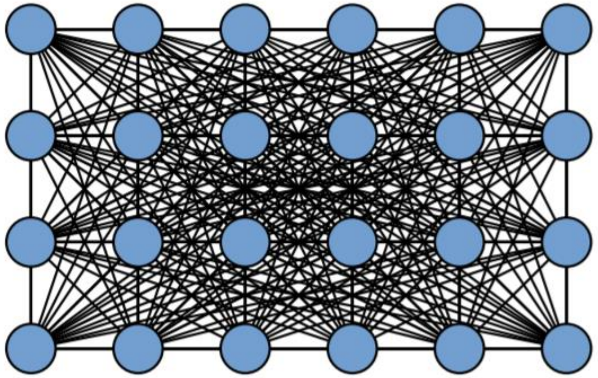
PAMI 2006

Vladimir Kolmogorov
University College London, UK

$$E(X) = \sum_{p \in V} D_p(x_p) + \sum_{(p,q) \in E} S_{pq}(x_p, x_q)$$

What about Dense Pixel Connections?

- Connecting to 4/8 neighbors generates excessive smooth of object boundaries
- This problem can be solved by fully connected a graph
  - Every node is connected to every other node
  - Graph-cut running time is $O(mn^2)$
    - m is the number of edges, n is the number of vertices
  - This paper finds an efficient solution by Gaussian filtering



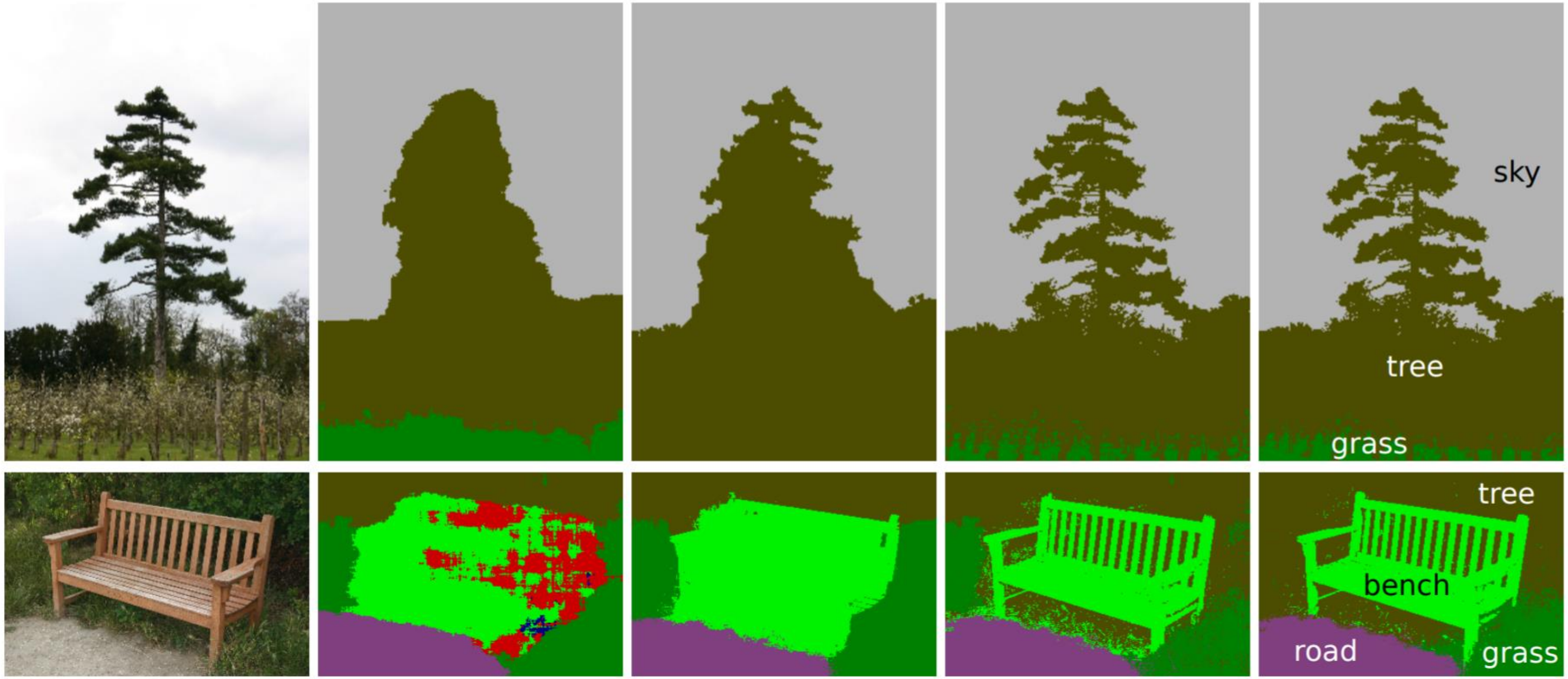
Efficient Inference in Fully Connected CRFs with Gaussian Edge Potentials

Philipp Krähenbühl
Computer Science Department
Stanford University
philkr@cs.stanford.edu

Vladlen Koltun
Computer Science Department
Stanford University
vladlen@cs.stanford.edu

NIPS 2012 best student paper

Fully Connected CRFs



(a) Image (b) Unary classifiers (c) Robust P^n CRF (d) Fully connected CRF, MCMC inference, 36 hrs (e) Fully connected CRF, our approach, 0.2 seconds