CAP5415-Computer Vision
Lecture 13-Image/Video Segmentation
Part II

Dr. Ulas Bagci

bagci@ucf.edu

Labeling & Segmentation

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

Labeling & Segmentation

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

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

Labeling & Segmentation

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

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

- Or continuous (as in optical flow)

$$L \subset \mathbb{R}^n \text{ for } n \geq 1$$

Labeling is a function

- Labels are assigned to *sites* (pixel locations)

Labeling is a function

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

Labeling is a function

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

Labeling is a function

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

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

Labeling is a function

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

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

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

**A is an adjacency relation between pixel locations*

Energy Function

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

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

Correct solution is embedded as the minimum.

Segmentation as an Energy Minimization Problem

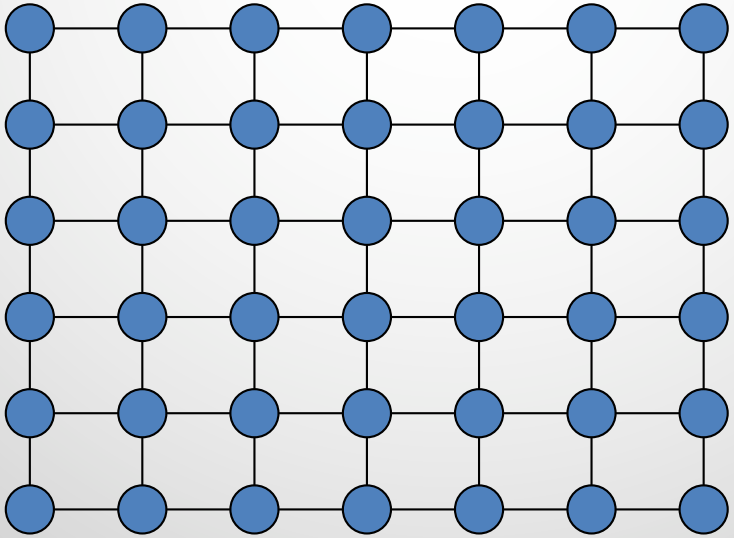
- E_{data} assigns non-negative penalties to a pixel location p when assigning a label to this location.

Segmentation as an Energy Minimization Problem

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

Segmentation as an Energy Minimization Problem

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



Segmentation as an Energy Minimization Problem

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

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

Energy Function-Details

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

Energy Function-Details

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

- Sample Data Term: $E_{data}(p, f_p) = \Psi(p)$

$$\Psi(p = 0) = -logP(p \in BG)$$

$$\Psi(p = 1) = -logP(p \in FG)$$

Energy Function-Details

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

- Sample Data Term: $E_{data}(p, f_p) = \Psi(p)$

$$\Psi(p = 0) = -logP(p \in BG)$$

$$\Psi(p = 1) = -logP(p \in FG)$$

- Sample Smoothness Term:

$$\Psi(p, q) = K_{pq}\delta(p \neq q) \text{ where}$$

$$K_{pq} = \frac{exp(-\beta(I_p - I_q)^2/(2\sigma^2))}{||p, q||}$$

Energy Function-Details

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

$$p^* = \underset{p \in L}{\operatorname{argmin}}\ E(p)$$

Energy Function-Details

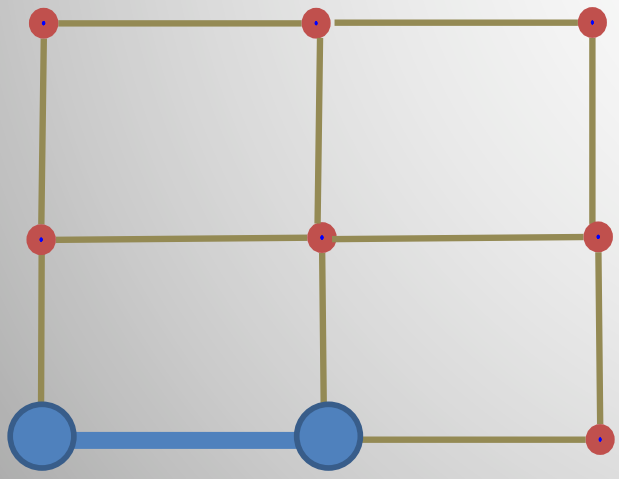
$$E(p) = \sum_{p \in \Omega} \Psi_p(p) + \sum_{p \in \Omega} \sum_{q \in A(p)} \Psi_{pq}(p, q)$$

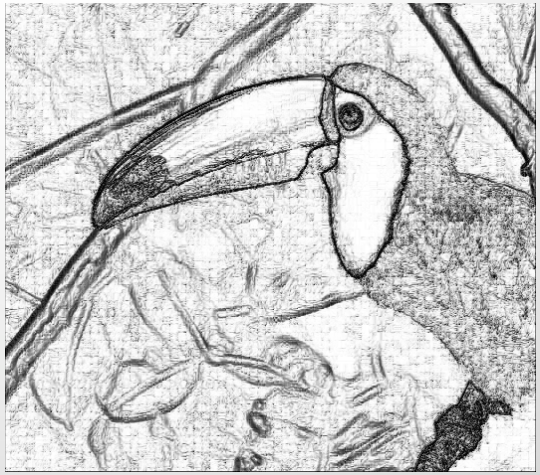
$$p^* = \underset{p \in L}{\operatorname{argmin}}\ E(p)$$

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

Energy Function

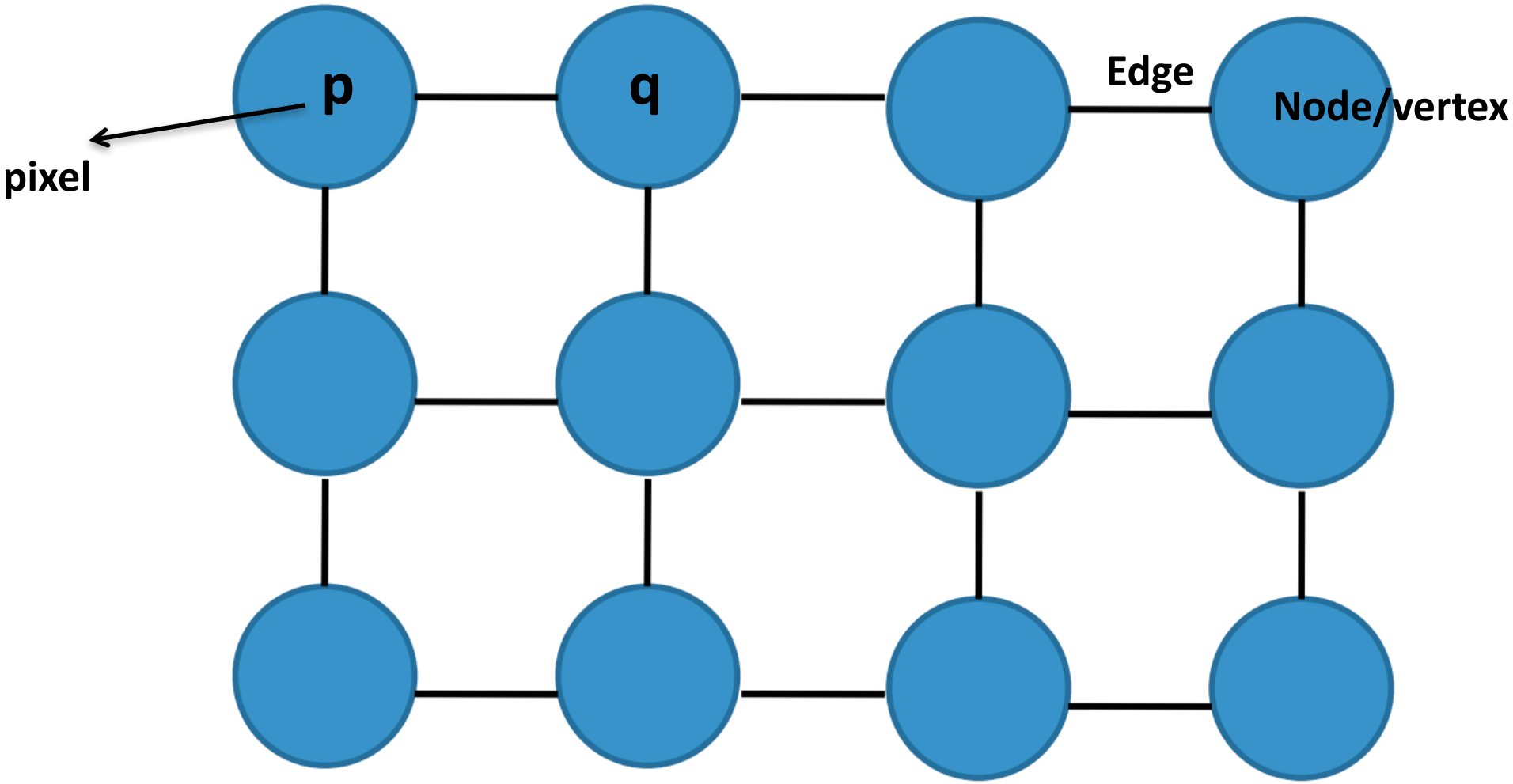
Unary Cost (data term)





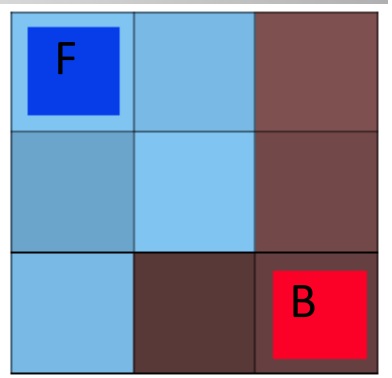
Discontinuity Cost

Recap: Image as a Graph



Graph Cuts for Optimal Boundary Detection (Boykov ICCV 2001)

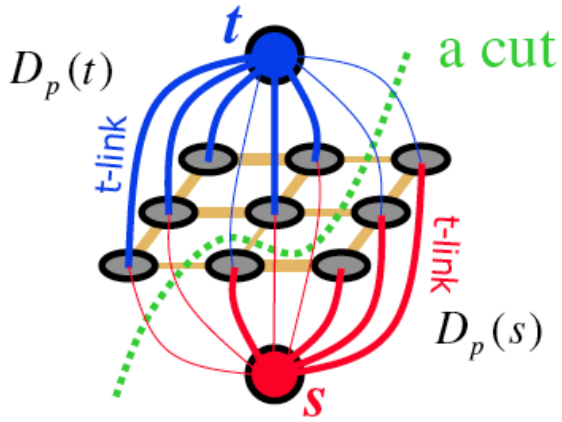
- Binary label: foreground vs. background
- User labels some pixels
- Exploit
  - Statistics of known Fg & Bg
  - Smoothness of label
- Turn into discrete graph optimization
  - Graph cut (min cut / max flow)



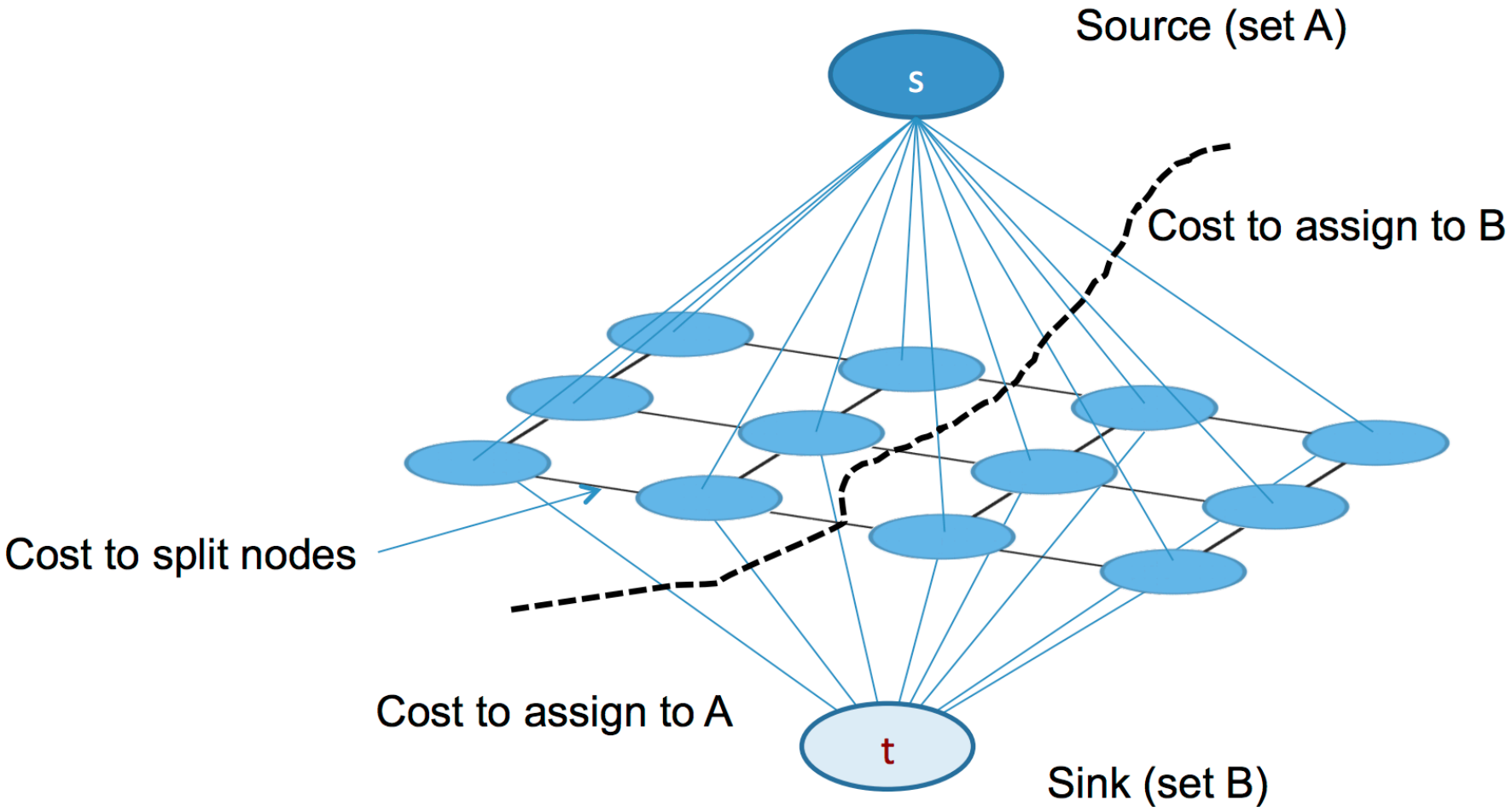
F F B

F F B

F B B



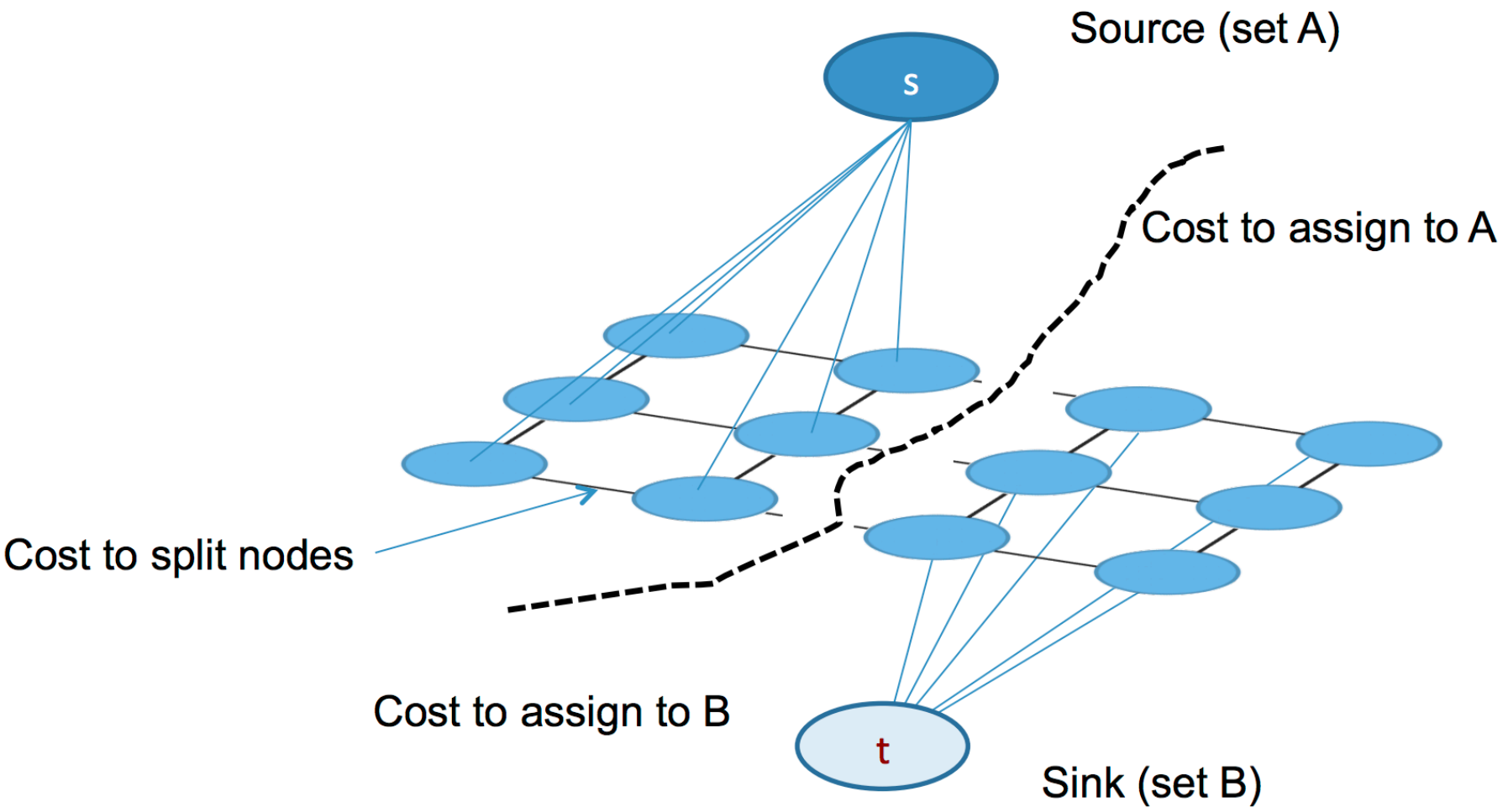
Graph-Cut



Each pixel is connected to its neighbors in an undirected graph

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

Graph-Cut

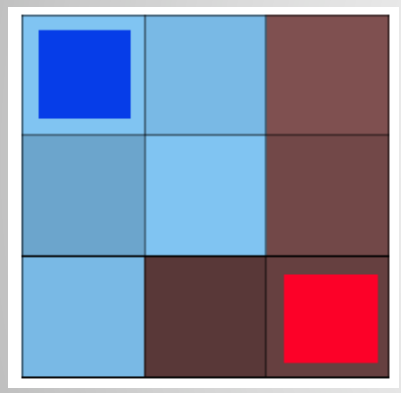


Each pixel is connected to its neighbors in an undirected graph

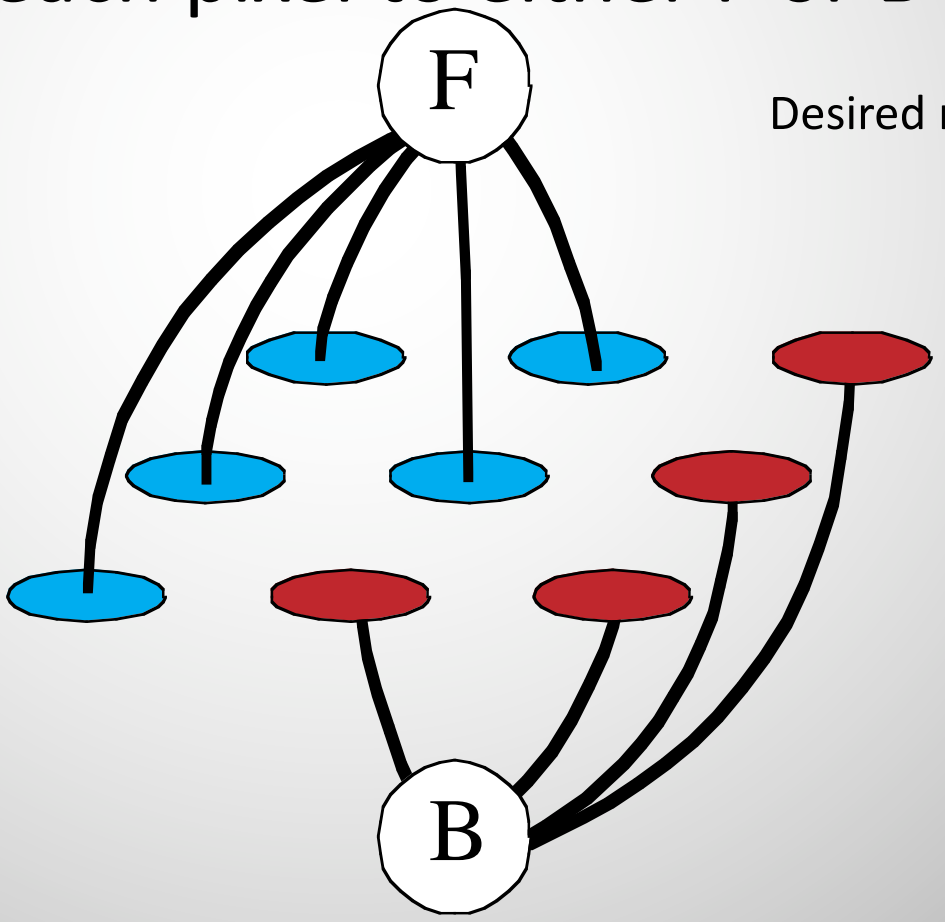
Goal: split nodes into two sets A and B based on pixel values, and try to classify Neighbors in the same way too!

Graph-Cut

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



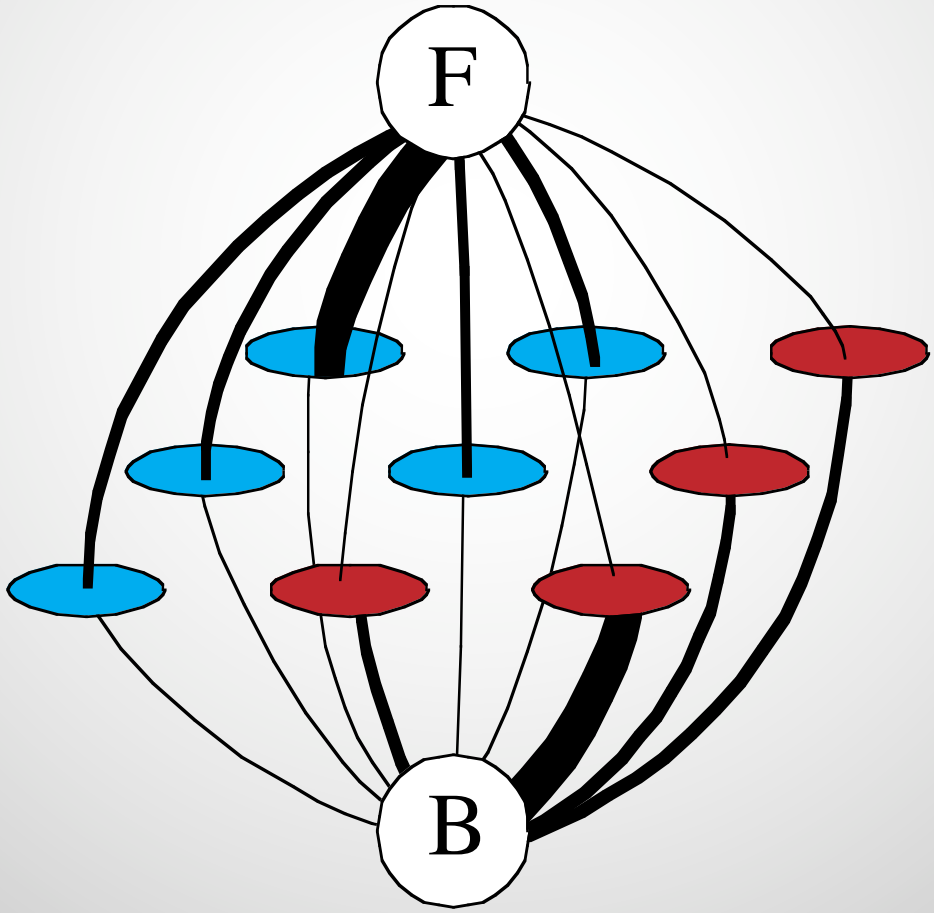
F	F	B
F	F	B
F	B	B



Desired result

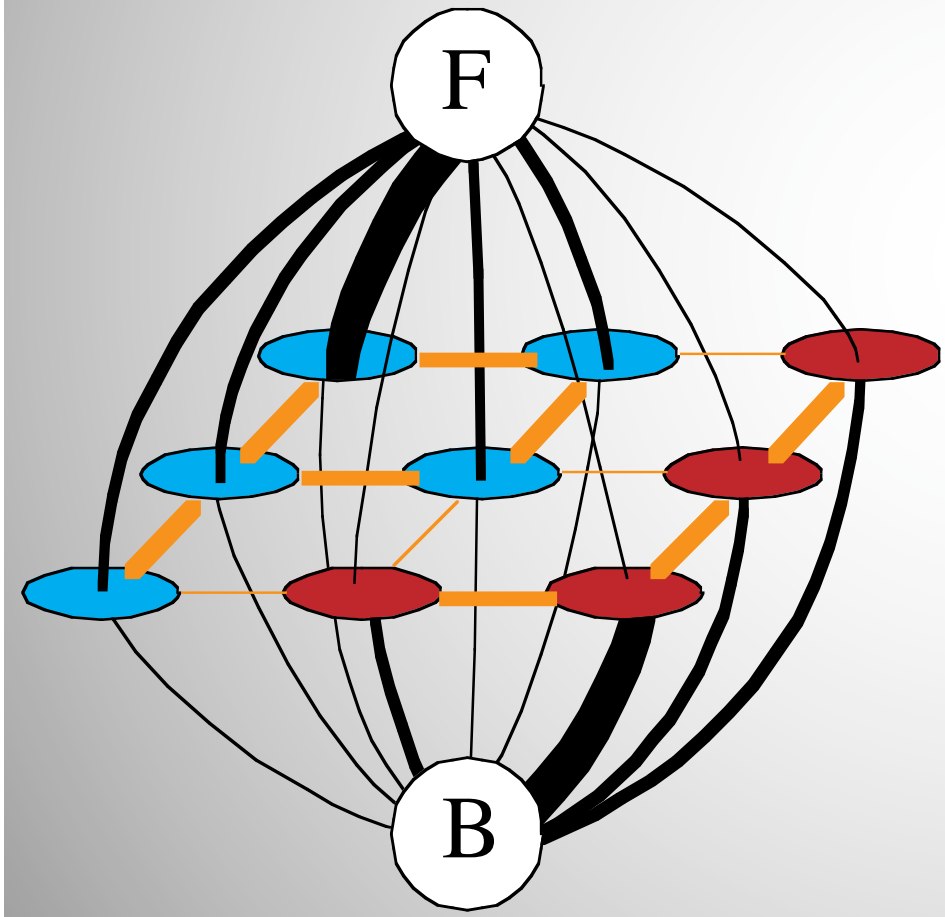
Cost Function: Data term

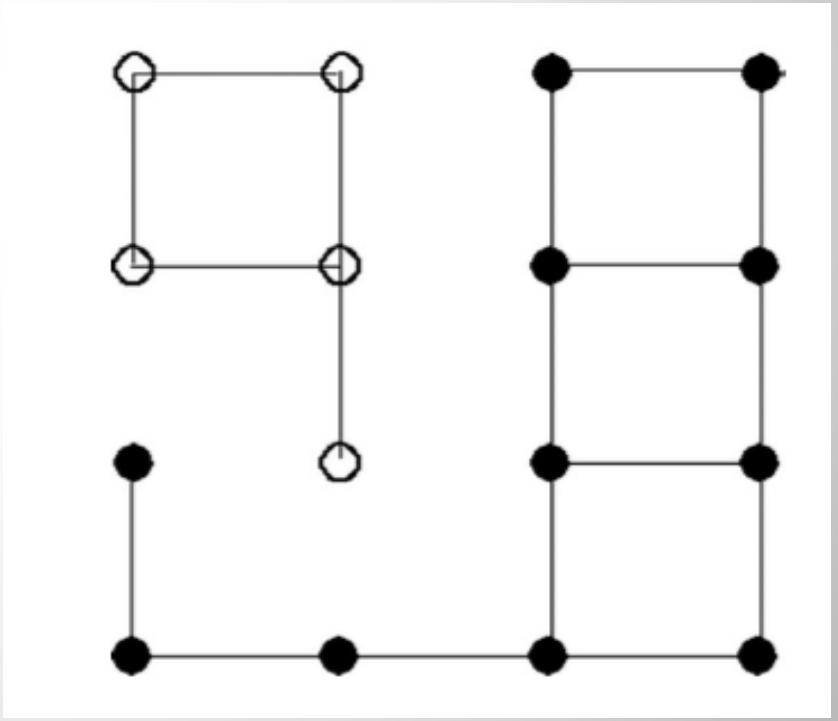
- Put one edge between each pixel and both F & G
- Weight of edge = minus data term



Cost Function: Smoothness term

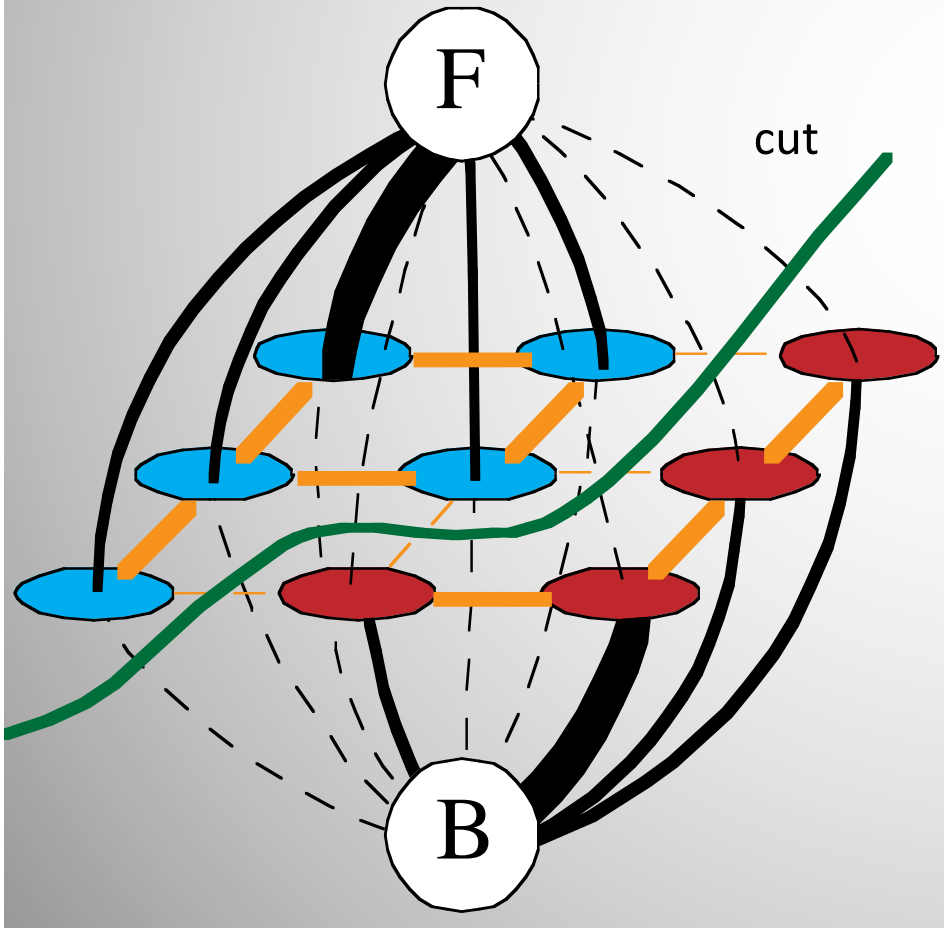
- Add an edge between each neighbor pair
- Weight = smoothness term

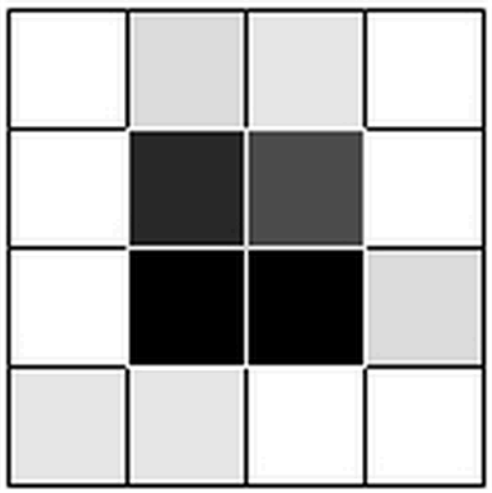




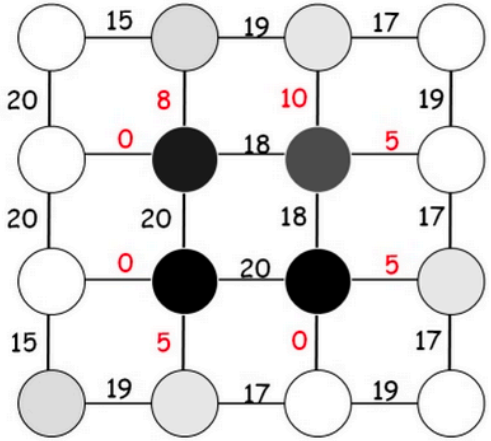
Min-Cut

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



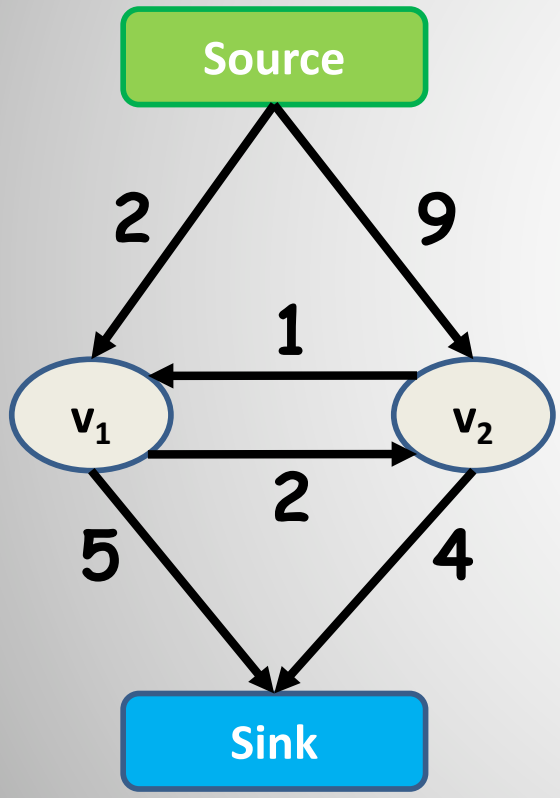


A 4x4 Image



Dissimilarity Graph of Pixels

Min-Cut



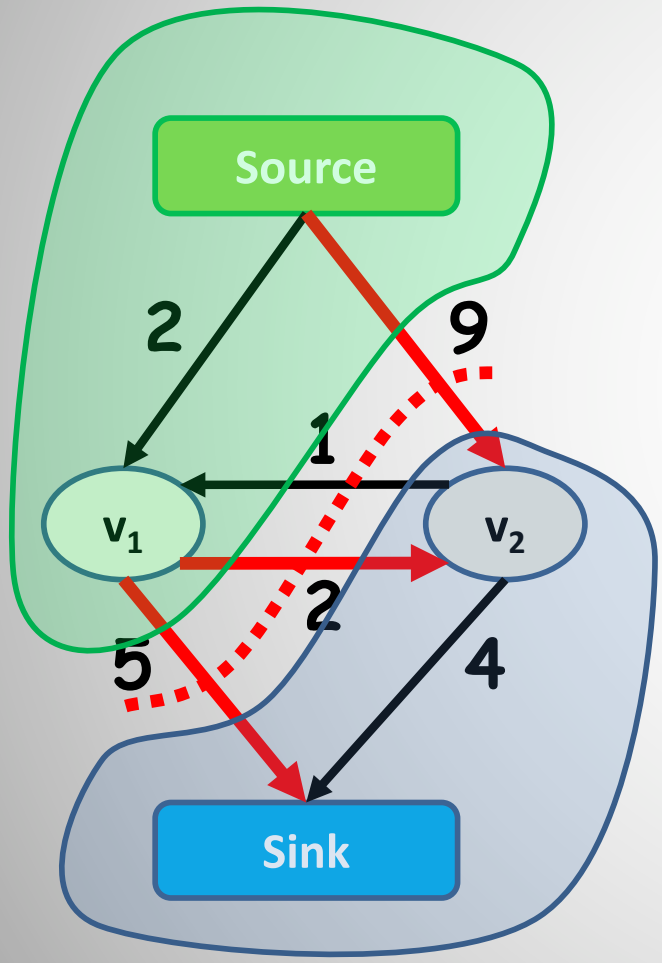
Graph (V, E, C)

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

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

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

Min-Cut



5 + 2 + 9 = 16

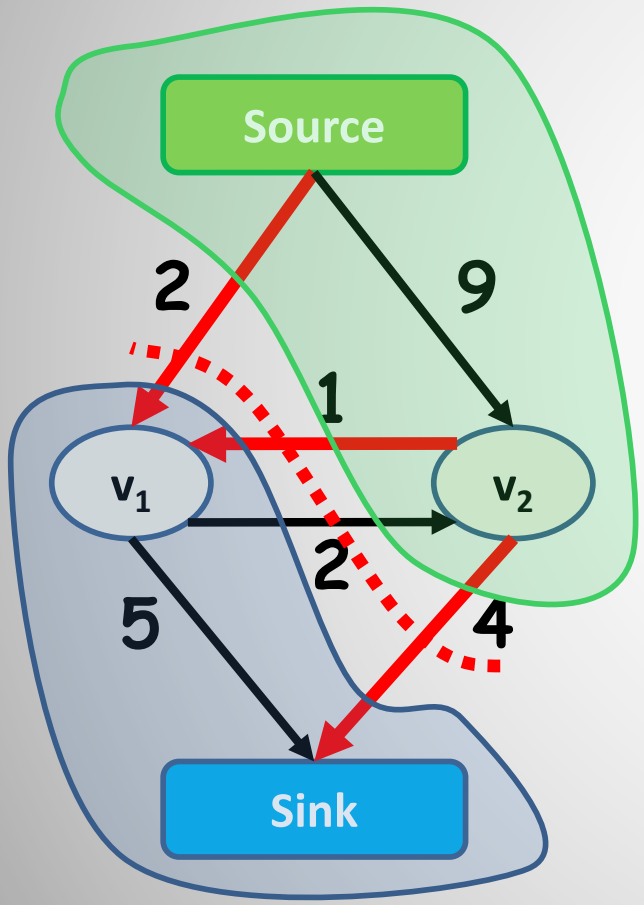
What is a st-cut?

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

What is the cost of a st-cut?

Sum of cost of all edges going from S to T

Min-Cut



2 + 1 + 4 = 7

What is a st-cut?

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

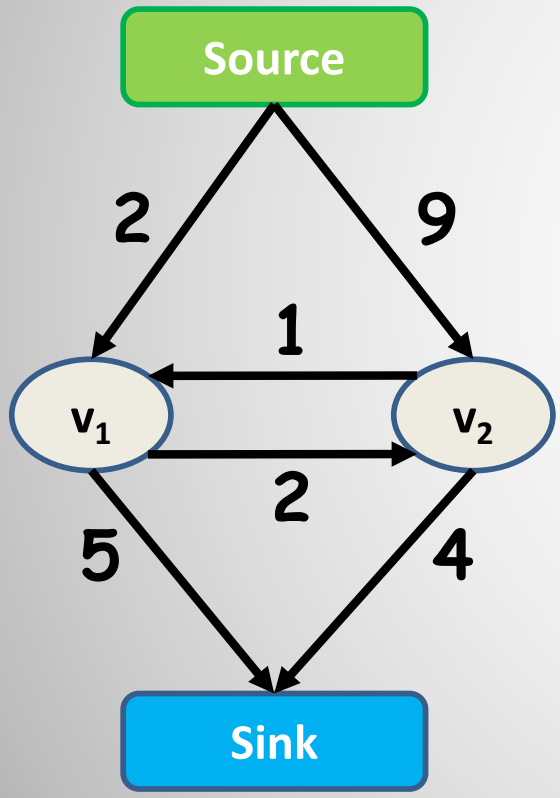
What is the cost of a st-cut?

Sum of cost of all edges going from S to T

What is the st-mincut?

st-cut with the minimum cost

How to compute min-cut?



Solve the dual **maximum flow** problem

Compute the maximum flow between Source and Sink

Constraints

Edges: Flow < Capacity

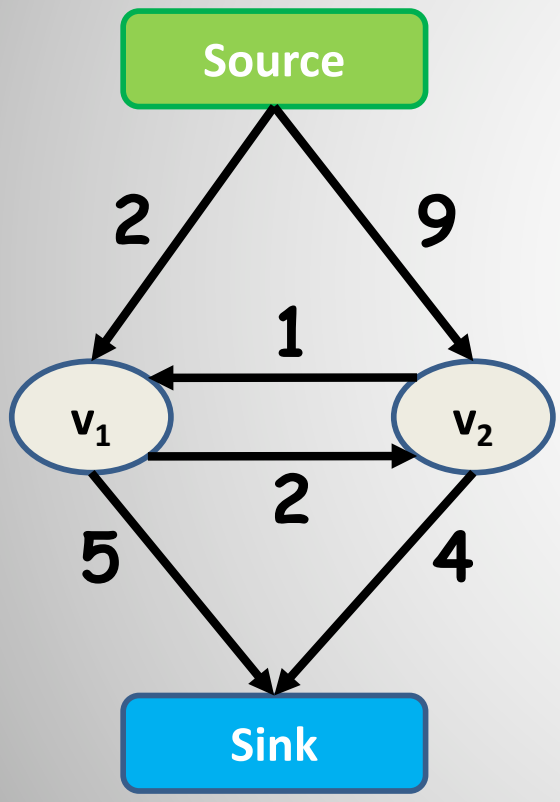
Nodes: Flow in & Flow out

Min-cut\Max-flow Theorem

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

Max-Flow Algorithms

Flow = 0



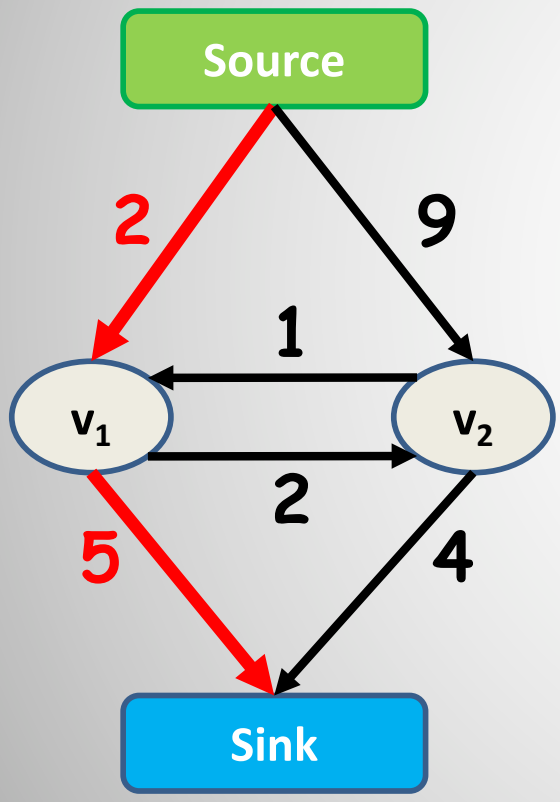
Augmenting Path Based Algorithms

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

Max-Flow Algorithms

Flow = 0



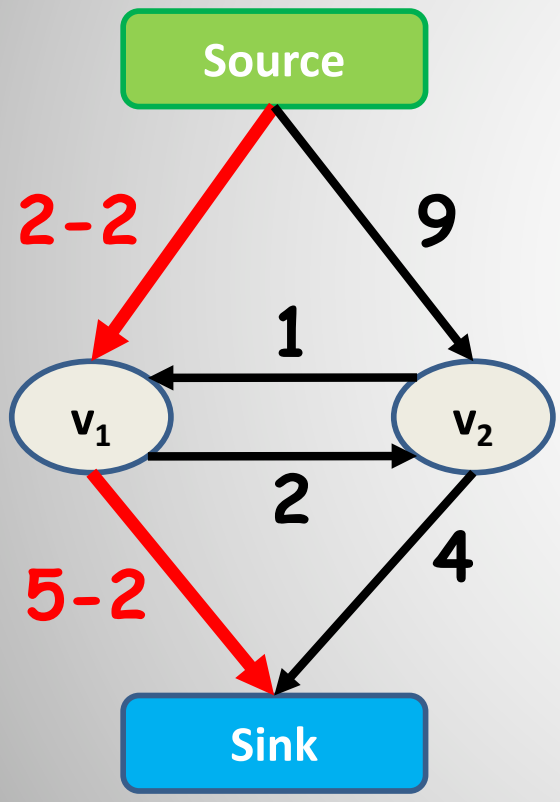
Augmenting Path Based Algorithms

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

Max-Flow Algorithms

Flow = 0 + 2



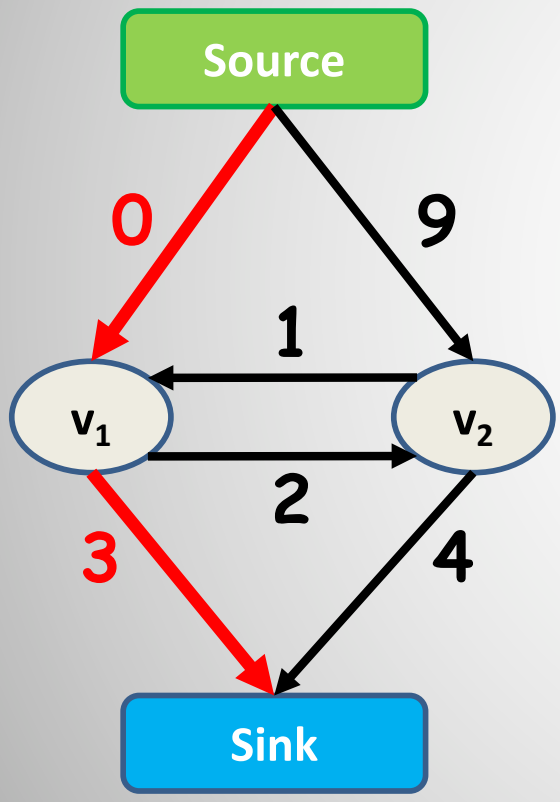
Augmenting Path Based Algorithms

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

Algorithms assume non-negative capacity

Max-Flow Algorithms

Flow = 2



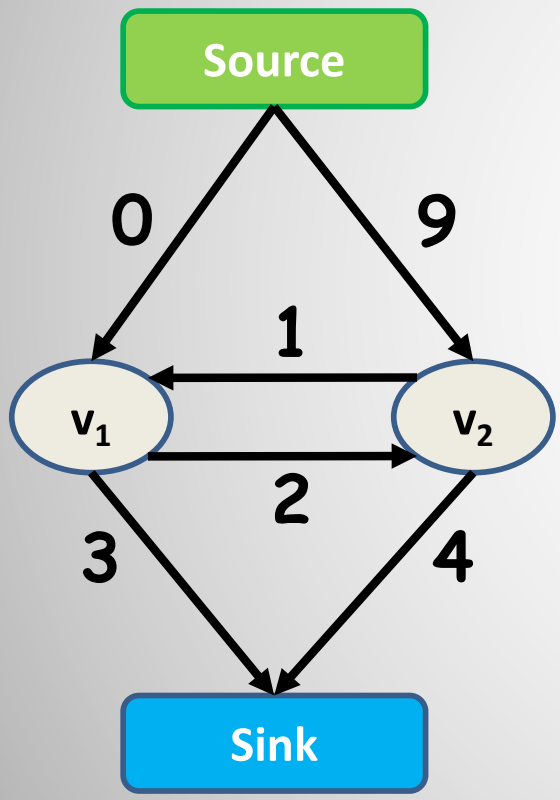
Augmenting Path Based Algorithms

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

Algorithms assume non-negative capacity

Max-Flow Algorithms

Flow = 2



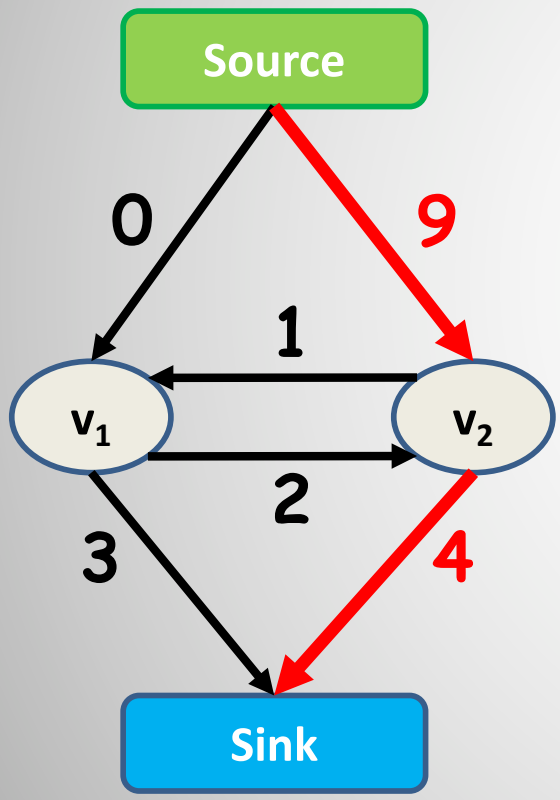
Augmenting Path Based Algorithms

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

Algorithms assume non-negative capacity

Max-Flow Algorithms

Flow = 2



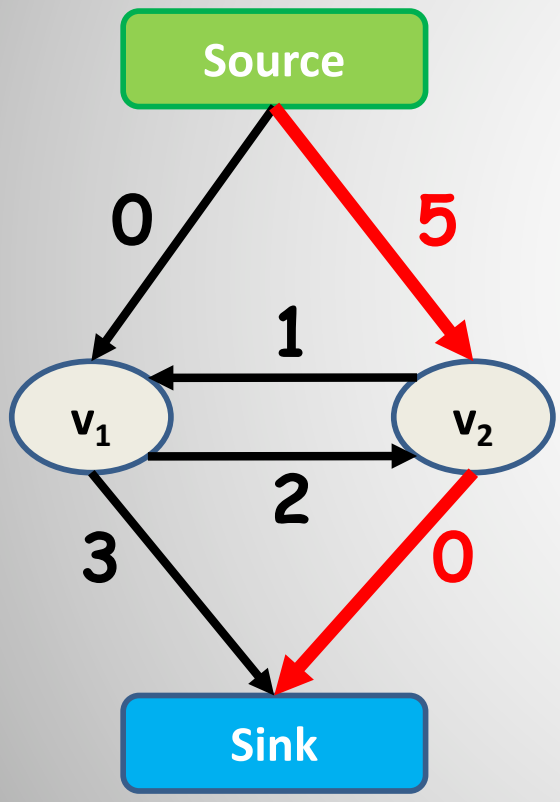
Augmenting Path Based Algorithms

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

Algorithms assume non-negative capacity

Max-Flow Algorithms

Flow = 2 + 4



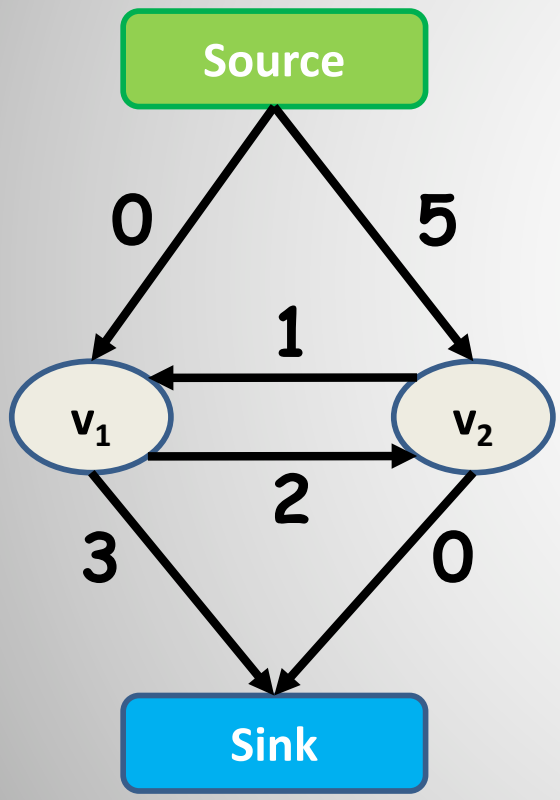
Augmenting Path Based Algorithms

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

Algorithms assume non-negative capacity

Max-Flow Algorithms

Flow = 6



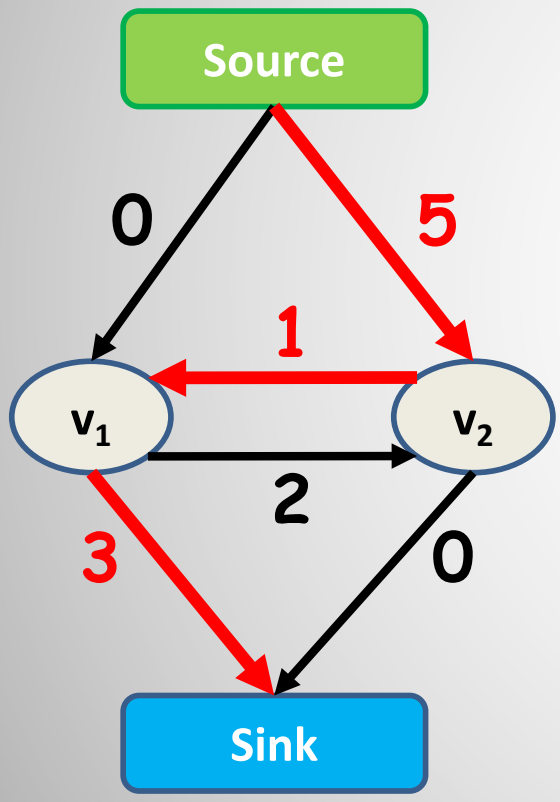
Augmenting Path Based Algorithms

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

Max-Flow Algorithms

Flow = 6



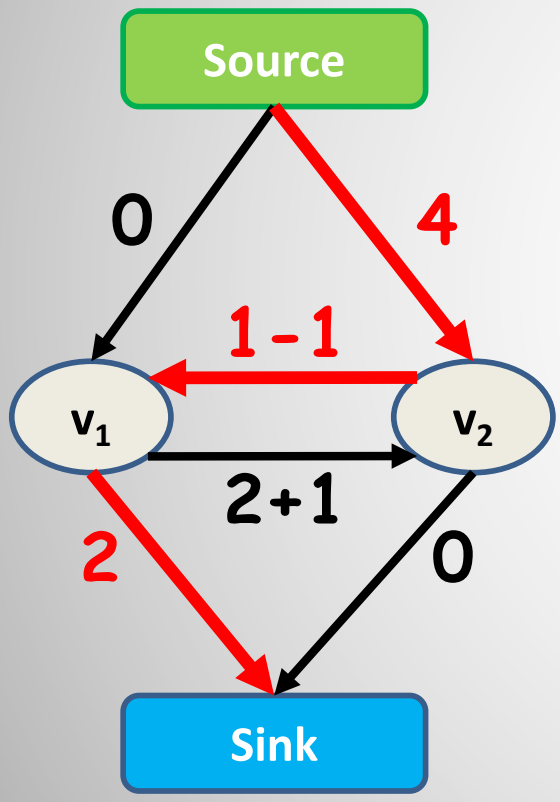
Augmenting Path Based Algorithms

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

Max-Flow Algorithms

Flow = 6 + 1



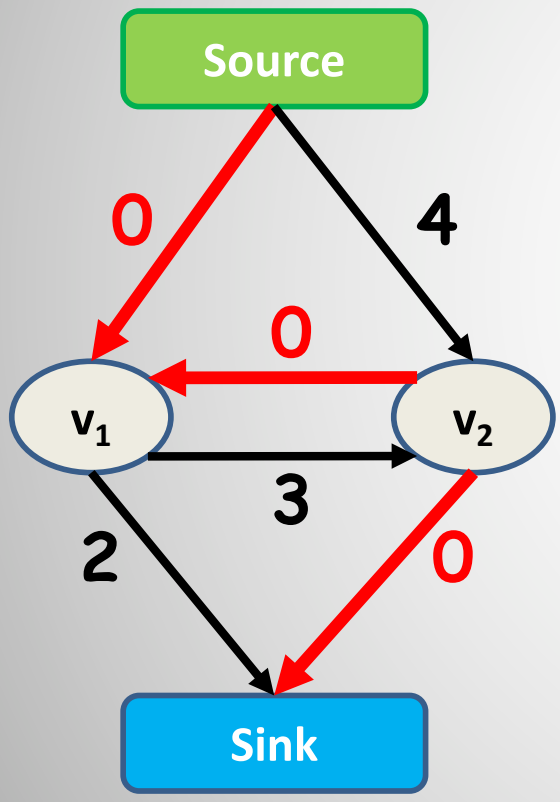
Augmenting Path Based Algorithms

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

Max-Flow Algorithms

Flow = 7

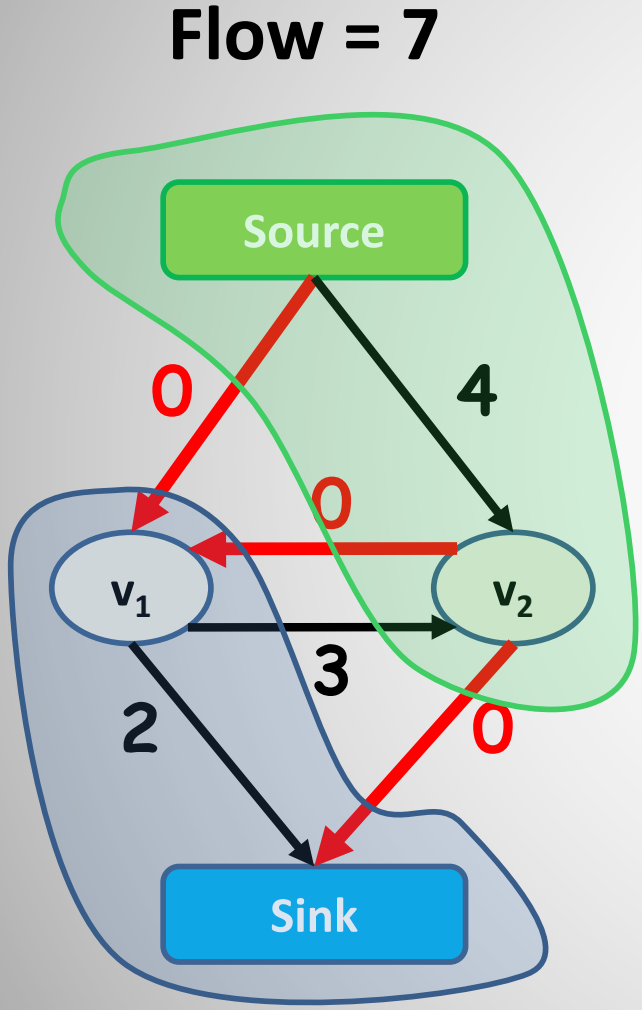


Augmenting Path Based Algorithms

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

Algorithms assume non-negative capacity

Max-Flow Algorithms

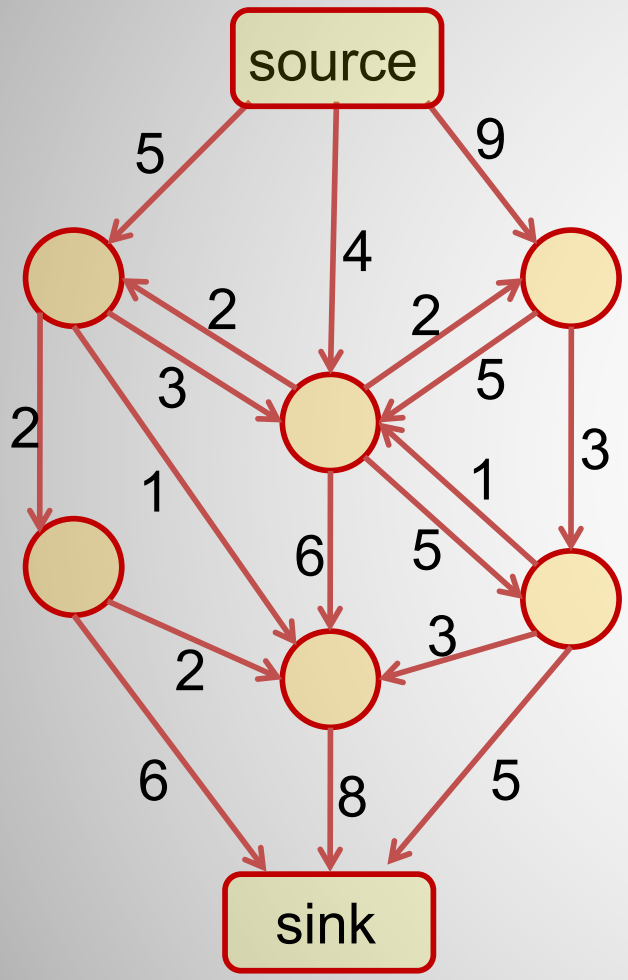


Augmenting Path Based Algorithms

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

Algorithms assume non-negative capacity

Another Example-Max Flow



Ford & Fulkerson algorithm (1956)

Find the path from source to sink

While (path exists)

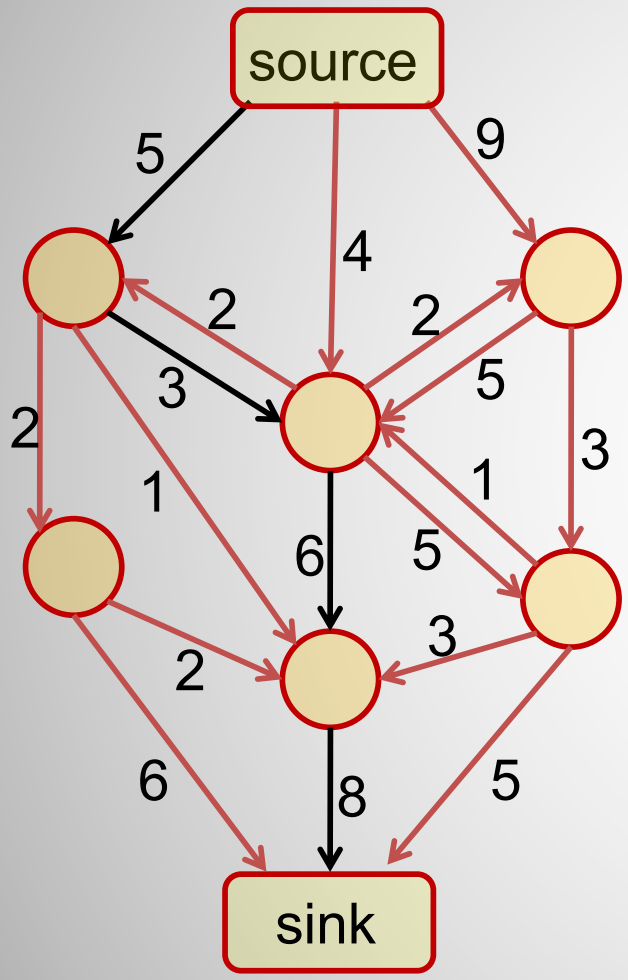
flow += maximum capacity in the path

Build the residual graph (“subtract” the flow)

Find the path in the residual graph

End

Another Example-Max Flow



Ford & Fulkerson algorithm (1956)

Find the path from source to sink

While (path exists)

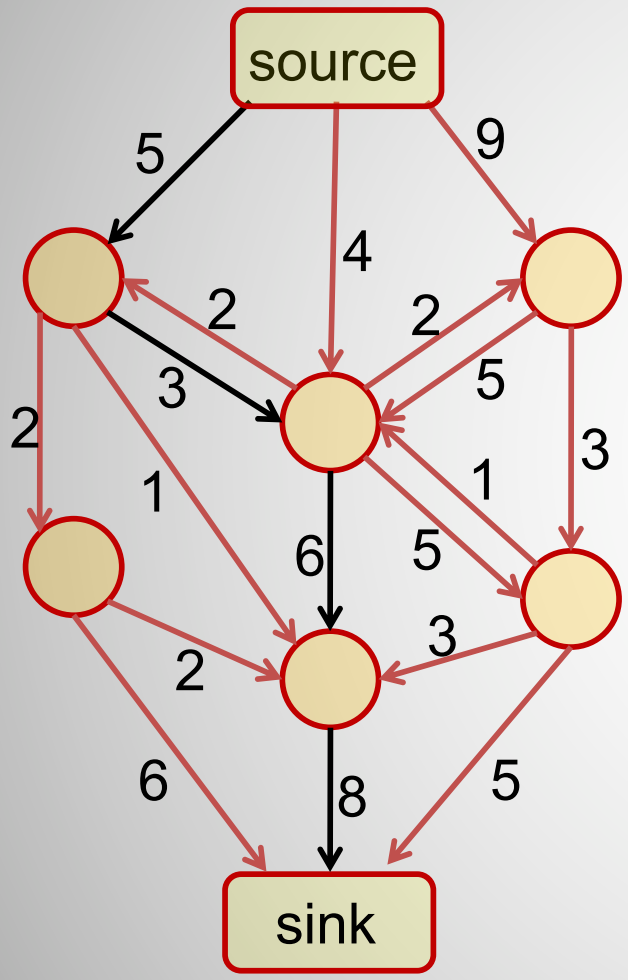
flow += maximum capacity in the path

Build the residual graph (“subtract” the flow)

Find the path in the residual graph

End

Another Example-Max Flow



flow = 3

Ford & Fulkerson algorithm (1956)

Find the path from source to sink

While (path exists)

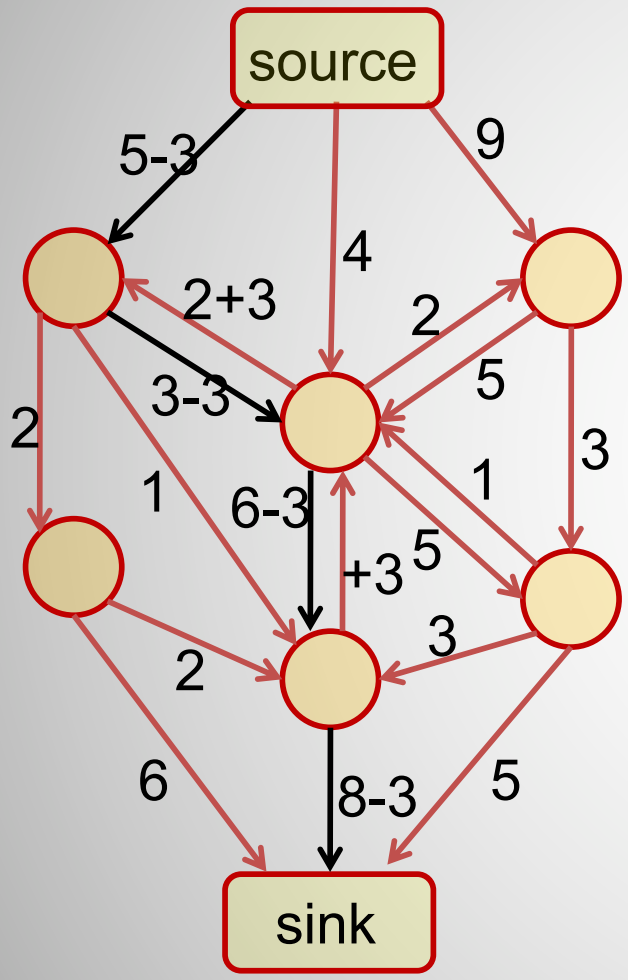
flow += maximum capacity in the path

Build the residual graph (“subtract” the flow)

Find the path in the residual graph

End

Another Example-Max Flow



flow = 3

Ford & Fulkerson algorithm (1956)

Find the path from source to sink

While (path exists)

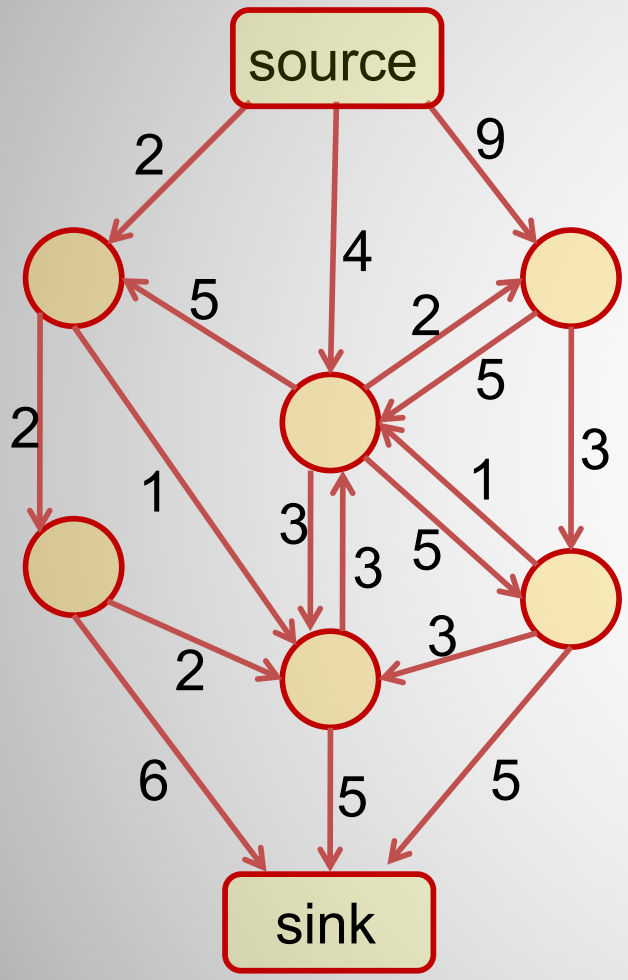
flow += maximum capacity in the path

Build the residual graph (“subtract” the flow)

Find the path in the residual graph

End

Another Example-Max Flow



flow = 3

Ford & Fulkerson algorithm (1956)

Find the path from source to sink

While (path exists)

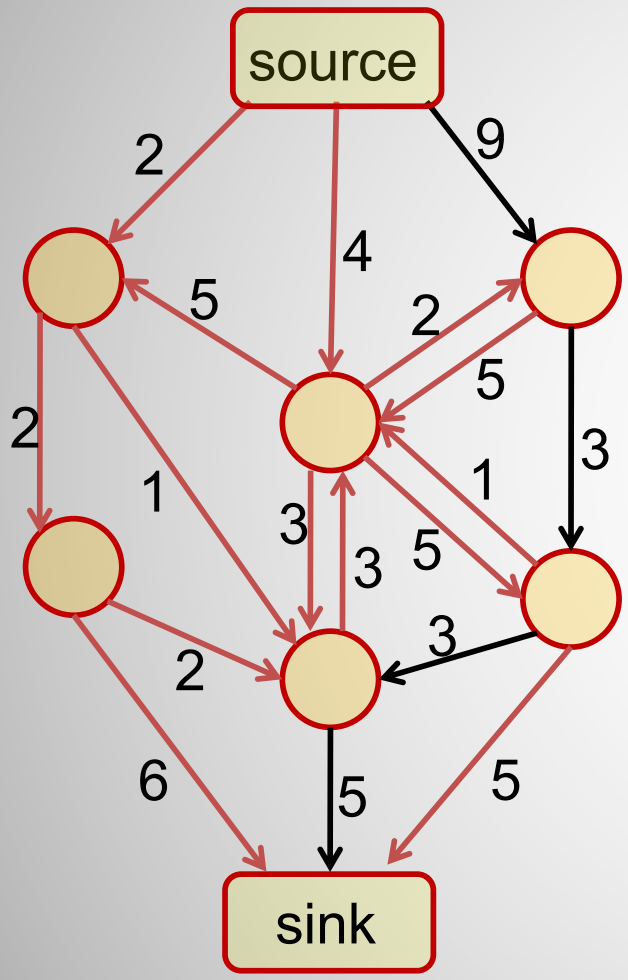
flow += maximum capacity in the path

Build the residual graph (“subtract” the flow)

Find the path in the residual graph

End

Another Example-Max Flow



flow = 3

Ford & Fulkerson algorithm (1956)

Find the path from source to sink

While (path exists)

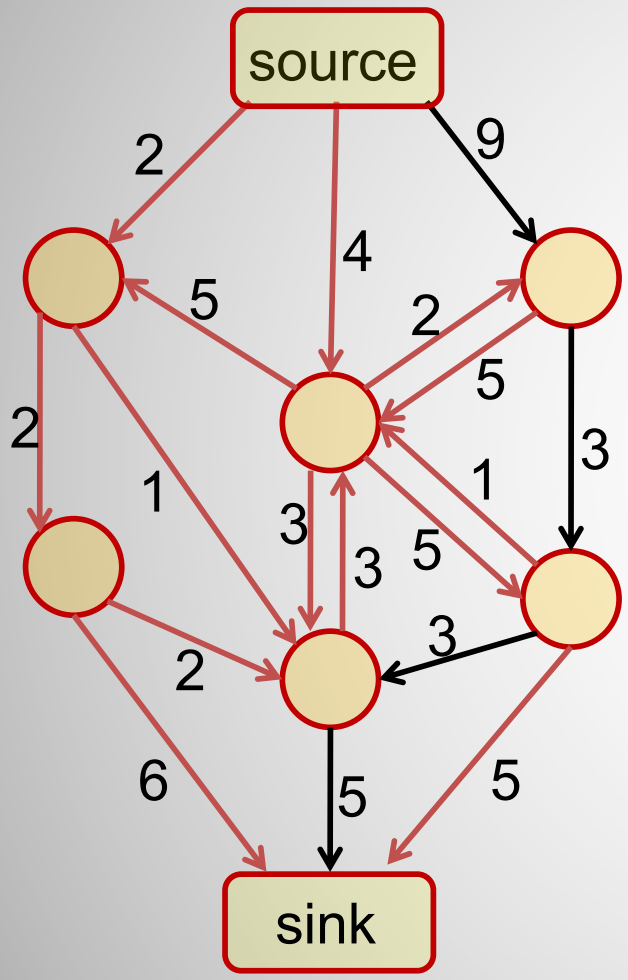
flow += maximum capacity in the path

Build the residual graph (“subtract” the flow)

Find the path in the residual graph

End

Another Example-Max Flow



flow = 6

Ford & Fulkerson algorithm (1956)

Find the path from source to sink

While (path exists)

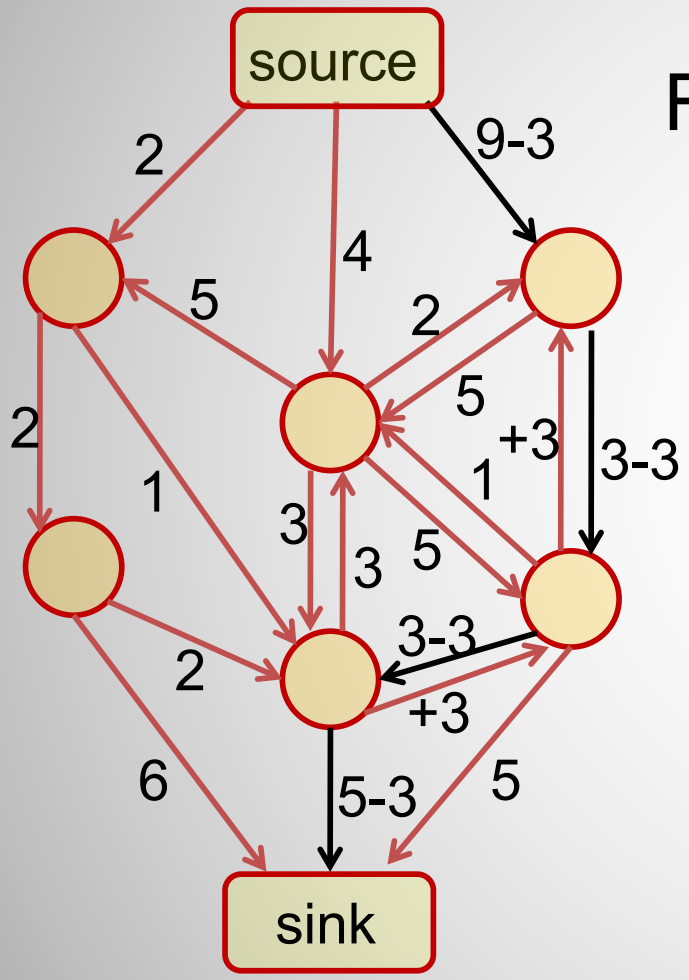
flow += maximum capacity in the path

Build the residual graph (“subtract” the flow)

Find the path in the residual graph

End

Another Example-Max Flow



flow = 6

Ford & Fulkerson algorithm (1956)

Find the path from source to sink

While (path exists)

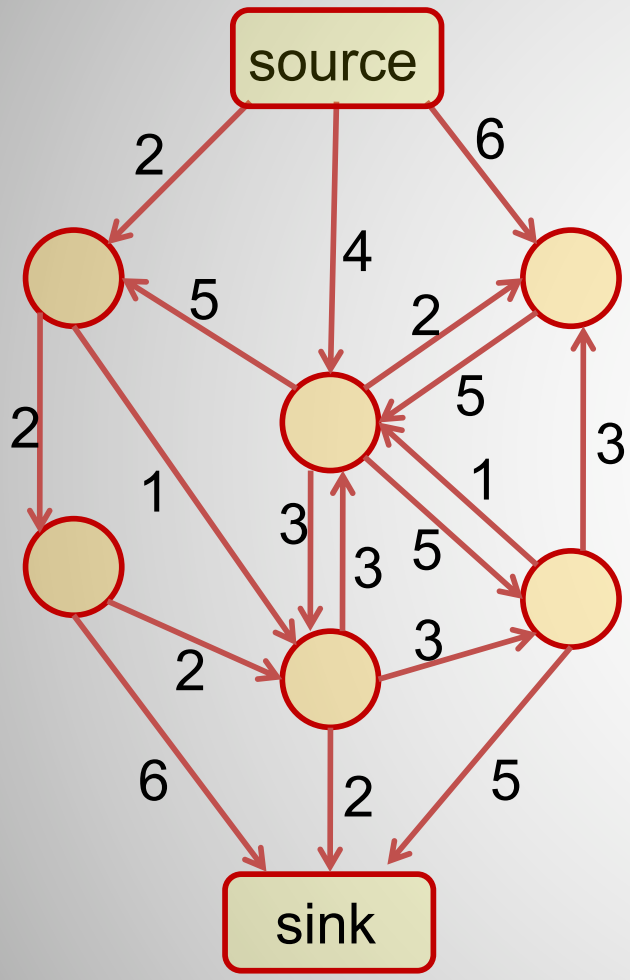
flow += maximum capacity in the path

Build the residual graph (“subtract” the flow)

Find the path in the residual graph

End

Another Example-Max Flow



flow = 6

Ford & Fulkerson algorithm (1956)

Find the path from source to sink

While (path exists)

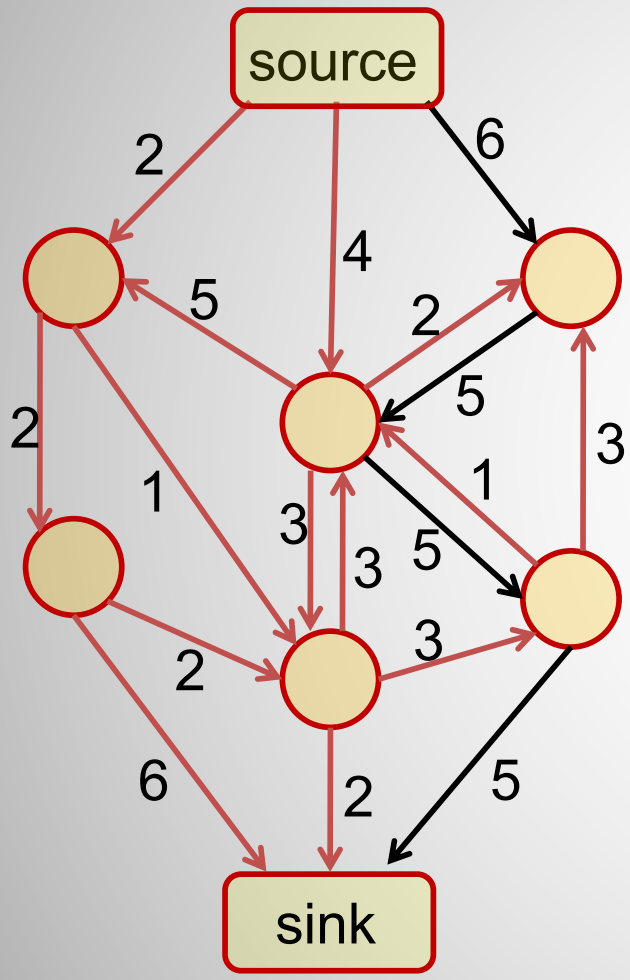
flow += maximum capacity in the path

Build the residual graph (“subtract” the flow)

Find the path in the residual graph

End

Another Example-Max Flow



flow = 6

Ford & Fulkerson algorithm (1956)

Find the path from source to sink

While (path exists)

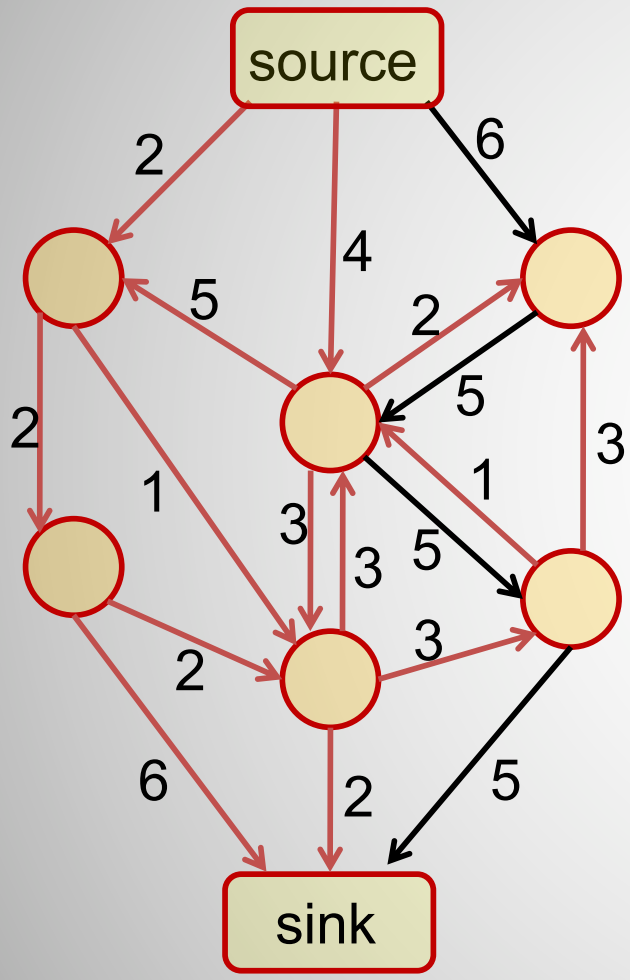
flow += maximum capacity in the path

Build the residual graph (“subtract” the flow)

Find the path in the residual graph

End

Another Example-Max Flow



flow = 11

Ford & Fulkerson algorithm (1956)

Find the path from source to sink

While (path exists)

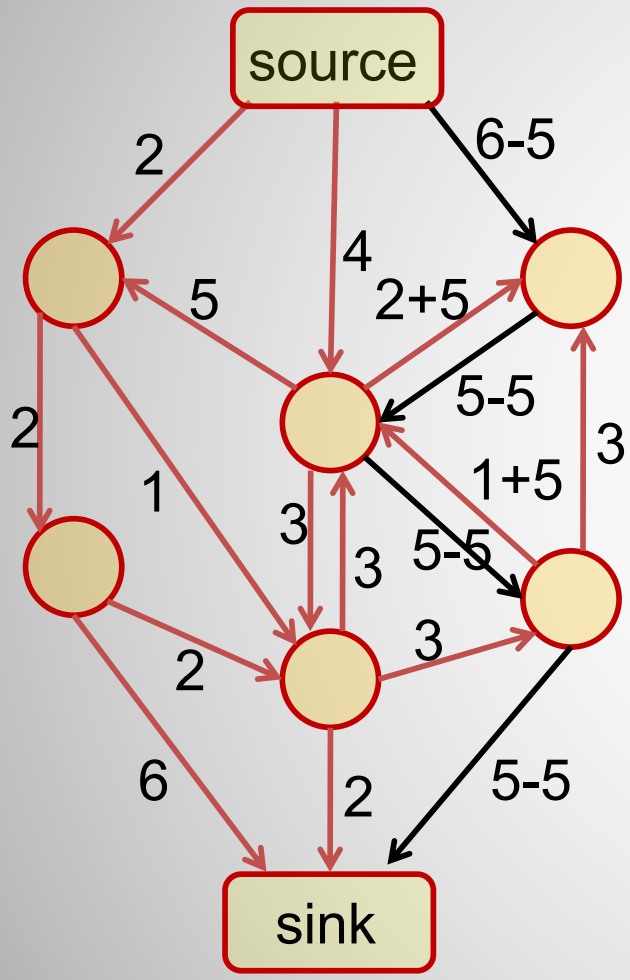
flow += maximum capacity in the path

Build the residual graph (“subtract” the flow)

Find the path in the residual graph

End

Another Example-Max Flow



flow = 11

Ford & Fulkerson algorithm (1956)

Find the path from source to sink

While (path exists)

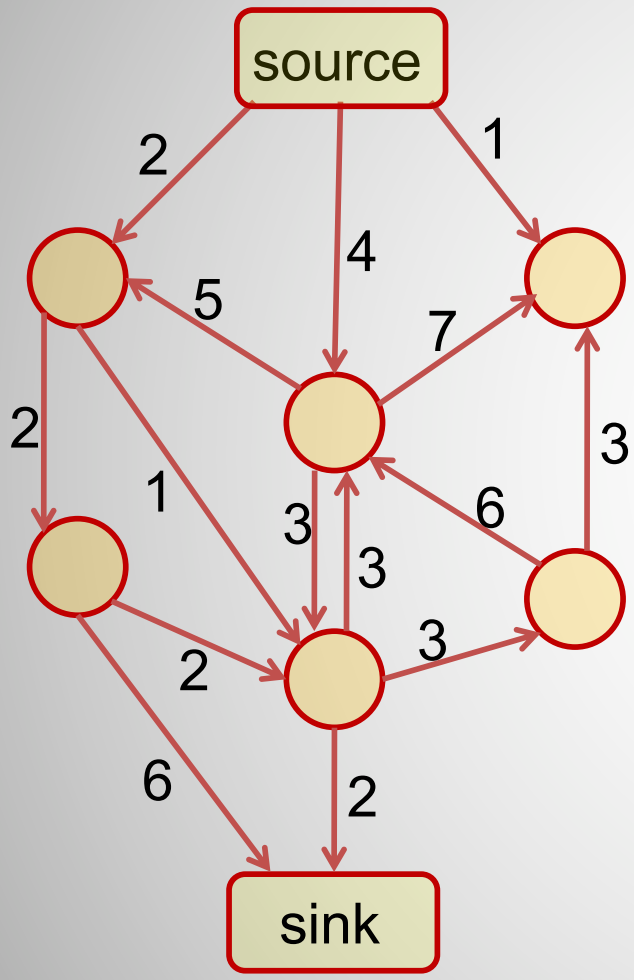
flow += maximum capacity in the path

Build the residual graph (“subtract” the flow)

Find the path in the residual graph

End

Another Example-Max Flow



flow = 11

Ford & Fulkerson algorithm (1956)

Find the path from source to sink

While (path exists)

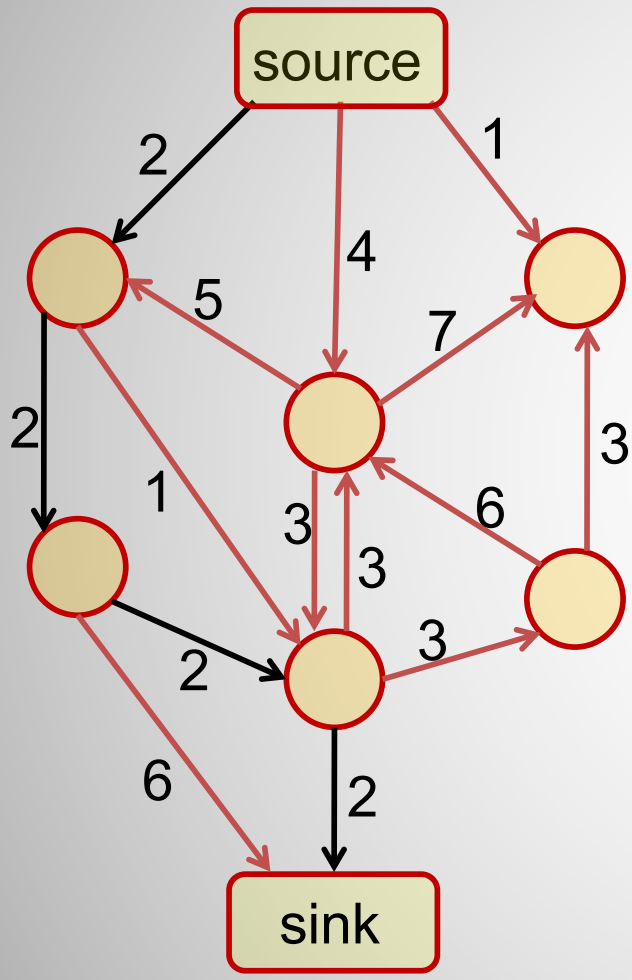
flow += maximum capacity in the path

Build the residual graph (“subtract” the flow)

Find the path in the residual graph

End

Another Example-Max Flow



flow = 11

Ford & Fulkerson algorithm (1956)

Find the path from source to sink

While (path exists)

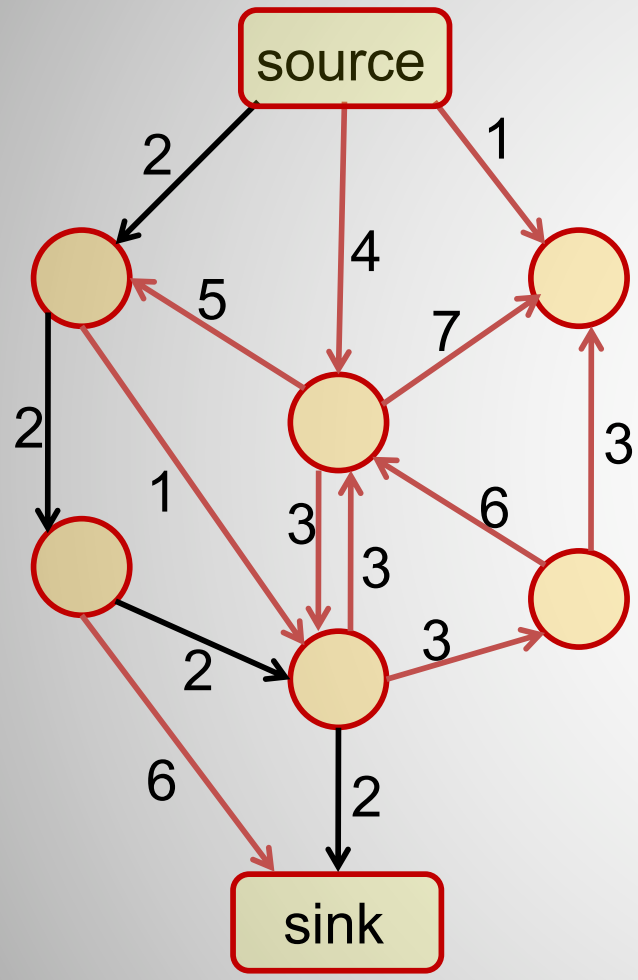
flow += maximum capacity in the path

Build the residual graph (“subtract” the flow)

Find the path in the residual graph

End

Another Example-Max Flow



flow = 13

Ford & Fulkerson algorithm (1956)

Find the path from source to sink

While (path exists)

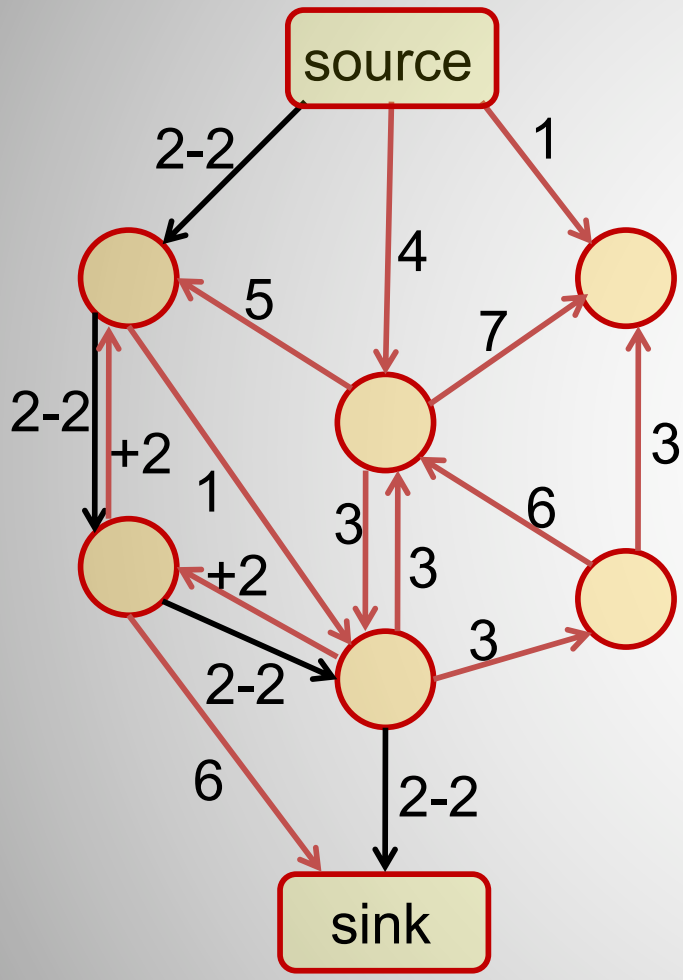
flow += maximum capacity in the path

Build the residual graph (“subtract” the flow)

Find the path in the residual graph

End

Another Example-Max Flow



flow = 13

Ford & Fulkerson algorithm (1956)

Find the path from source to sink

While (path exists)

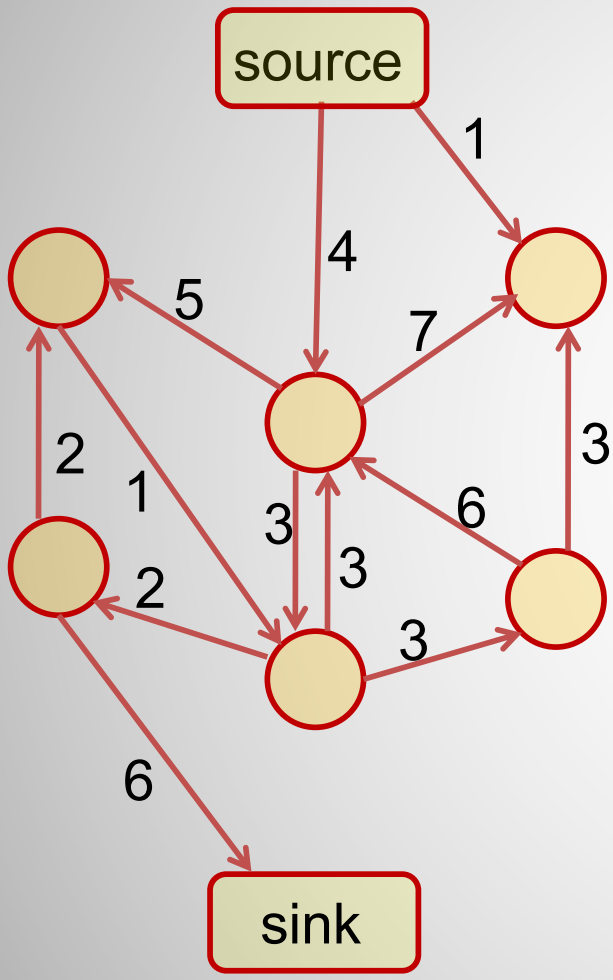
flow += maximum capacity in the path

Build the residual graph (“subtract” the flow)

Find the path in the residual graph

End

Another Example-Max Flow



flow = 13

Ford & Fulkerson algorithm (1956)

Find the path from source to sink

While (path exists)

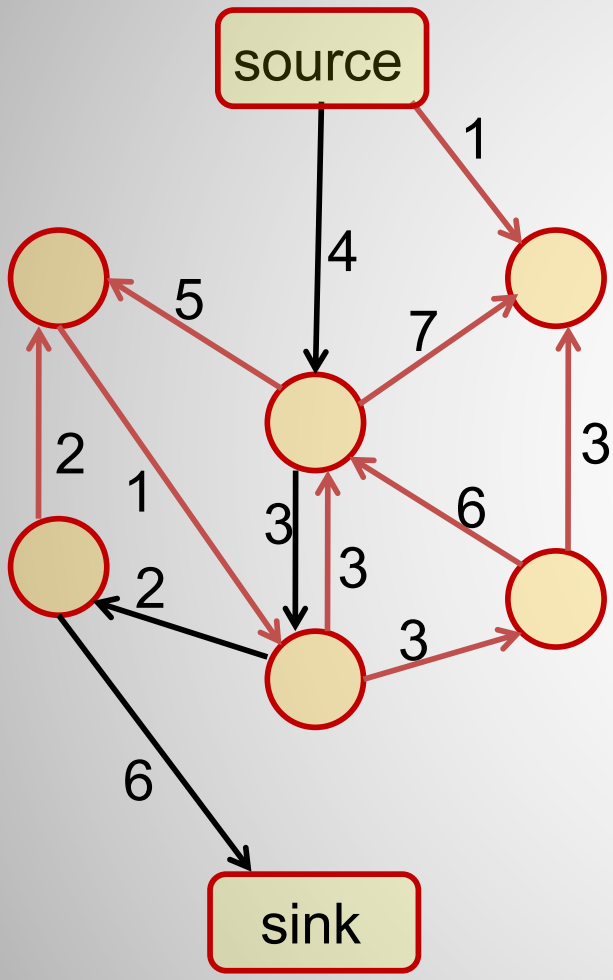
flow += maximum capacity in the path

Build the residual graph (“subtract” the flow)

Find the path in the residual graph

End

Another Example-Max Flow



flow = 13

Ford & Fulkerson algorithm (1956)

Find the path from source to sink

While (path exists)

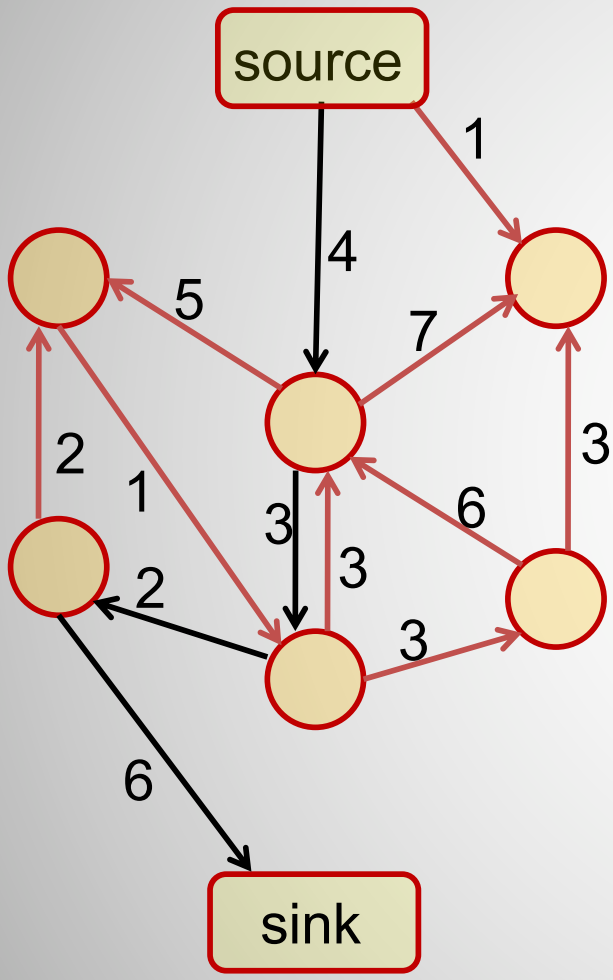
flow += maximum capacity in the path

Build the residual graph (“subtract” the flow)

Find the path in the residual graph

End

Another Example-Max Flow



flow = 15

Ford & Fulkerson algorithm (1956)

Find the path from source to sink

While (path exists)

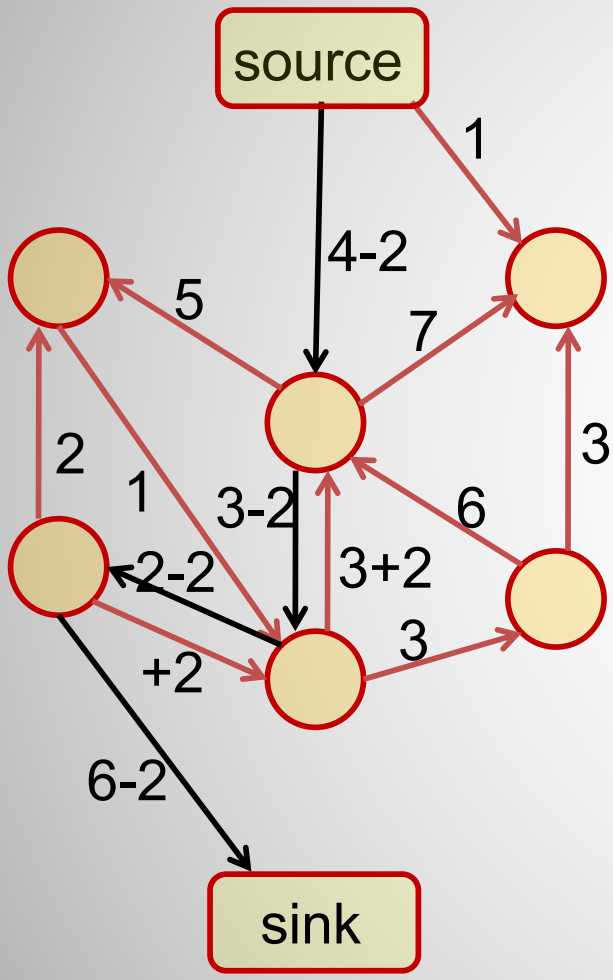
flow += maximum capacity in the path

Build the residual graph (“subtract” the flow)

Find the path in the residual graph

End

Another Example-Max Flow



flow = 15

Ford & Fulkerson algorithm (1956)

Find the path from source to sink

While (path exists)

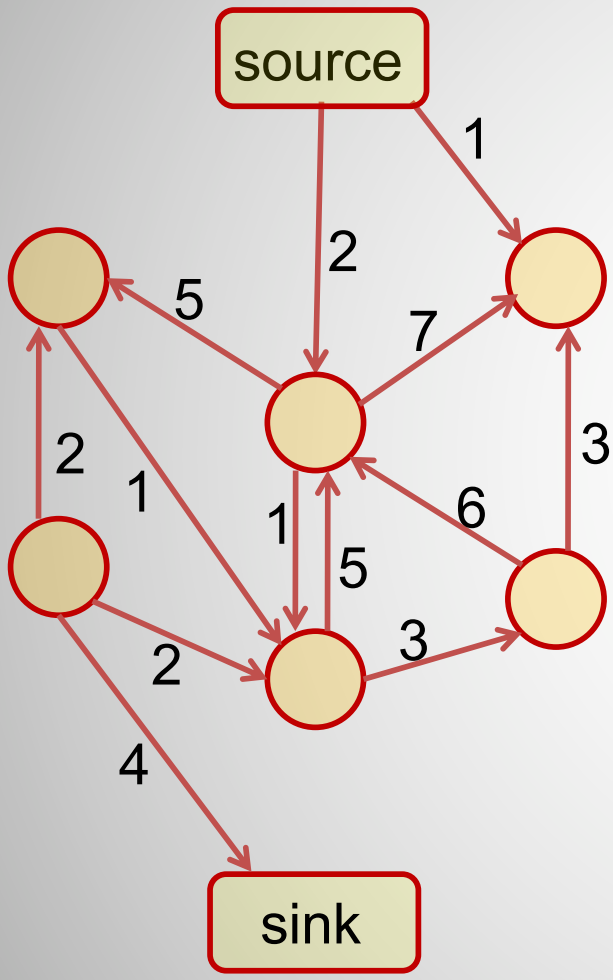
flow += maximum capacity in the path

Build the residual graph (“subtract” the flow)

Find the path in the residual graph

End

Another Example-Max Flow



flow = 15

Ford & Fulkerson algorithm (1956)

Find the path from source to sink

While (path exists)

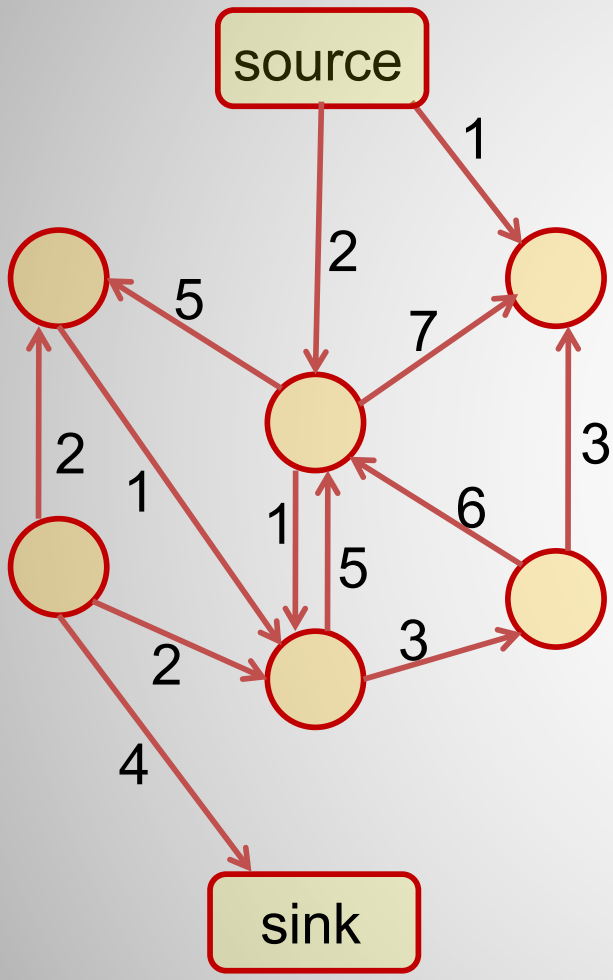
flow += maximum capacity in the path

Build the residual graph (“subtract” the flow)

Find the path in the residual graph

End

Another Example-Max Flow



flow = 15

Ford & Fulkerson algorithm (1956)

Find the path from source to sink

While (path exists)

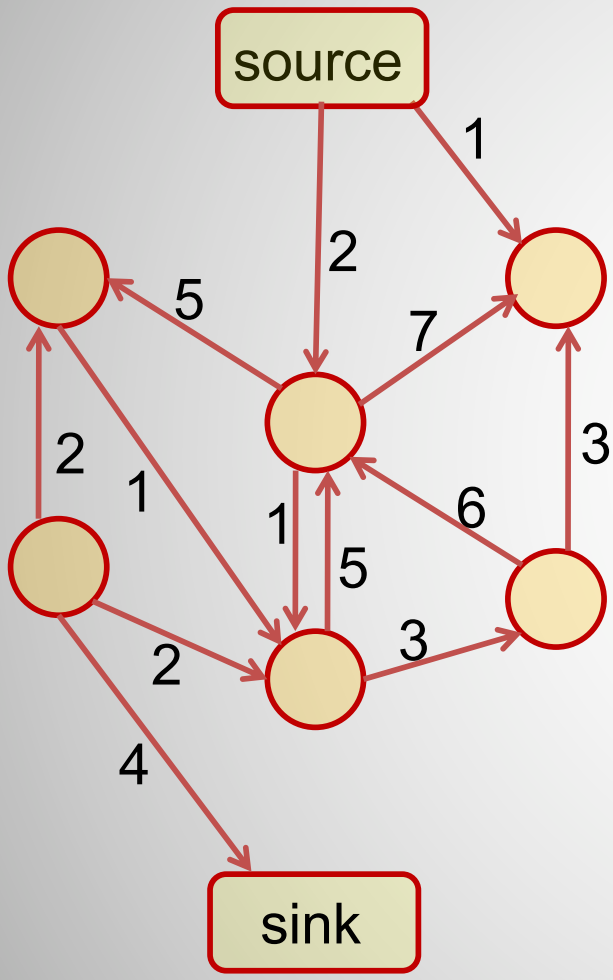
flow += maximum capacity in the path

Build the residual graph (“subtract” the flow)

Find the path in the residual graph

End

Another Example-Max Flow



flow = 15

Ford & Fulkerson algorithm (1956)

Find the path from source to sink

While (path exists)

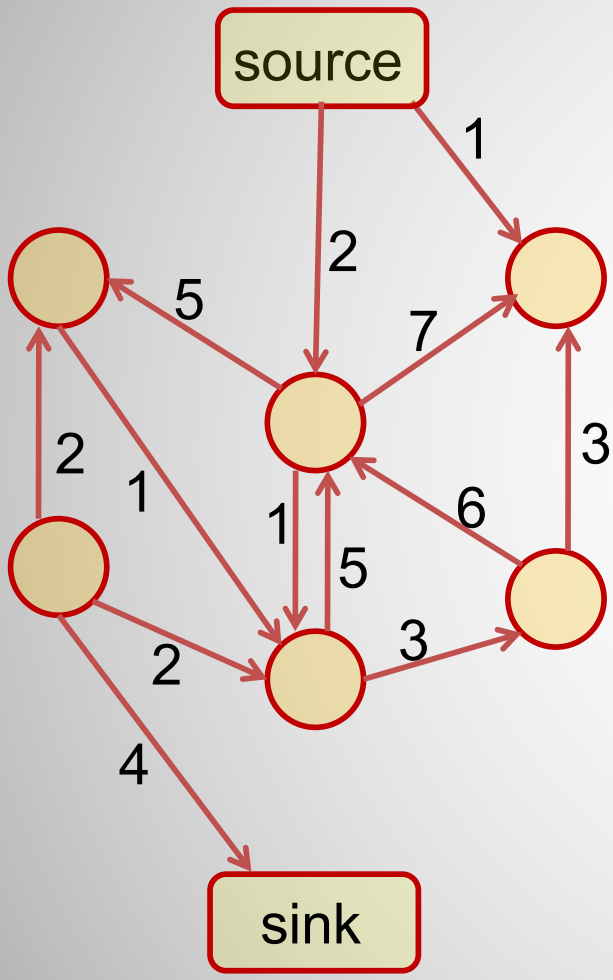
flow += maximum capacity in the path

Build the residual graph (“subtract” the flow)

Find the path in the residual graph

End

Another Example-Max Flow

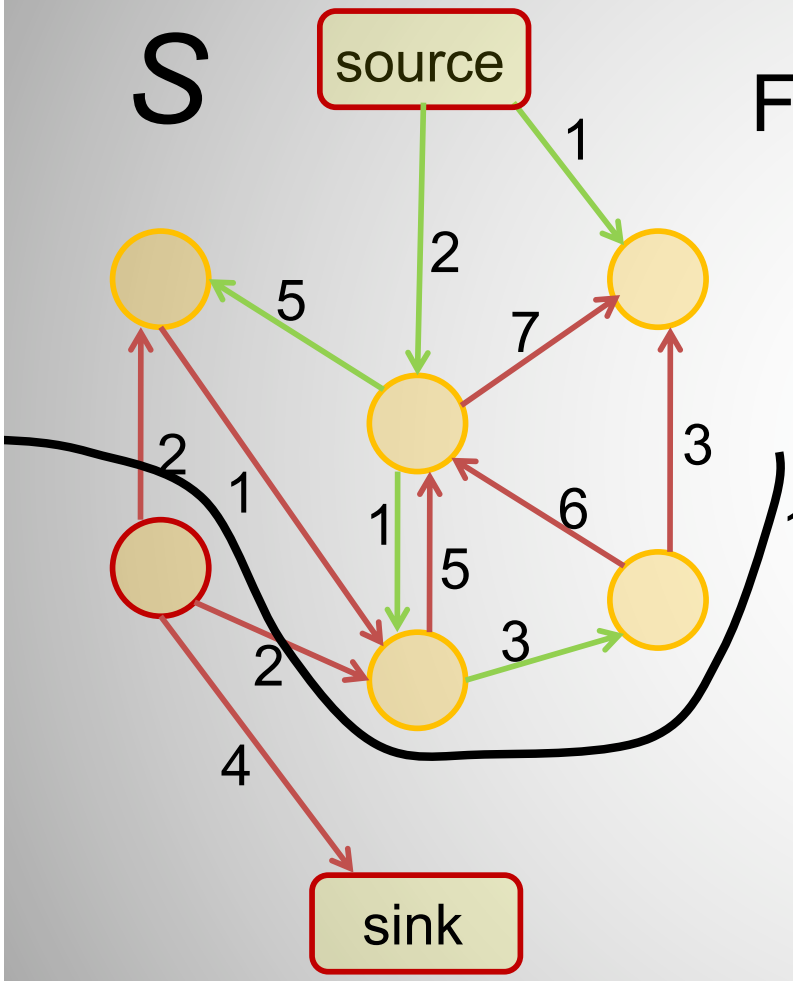


Ford & Fulkerson algorithm (1956)

Why is the solution globally optimal ?

flow = 15

Another Example-Max Flow



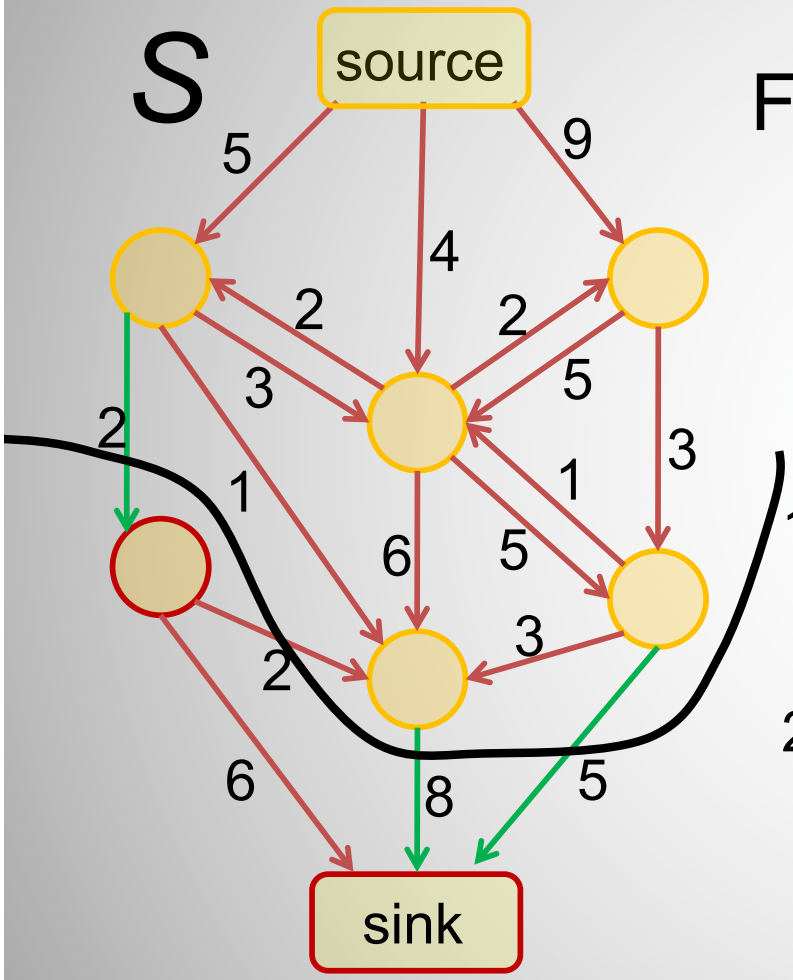
flow = 15

Ford & Fulkerson algorithm (1956)

Why is the solution globally optimal ?

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

Another Example-Max Flow

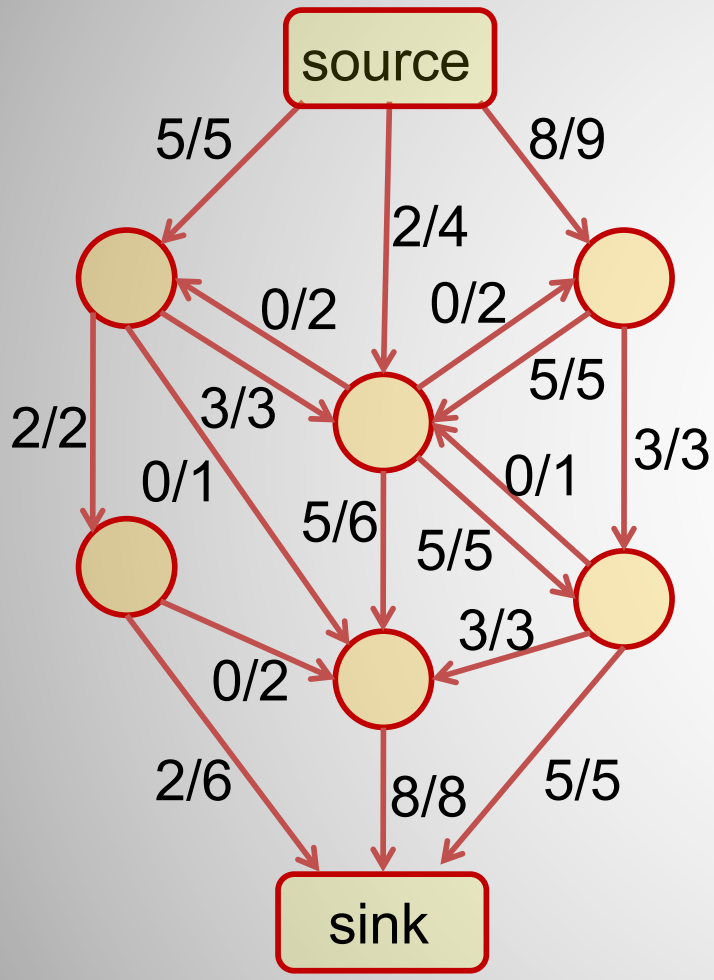


Ford & Fulkerson algorithm (1956)

Why is the solution globally optimal ?

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

Another Example-Max Flow



flow = 15

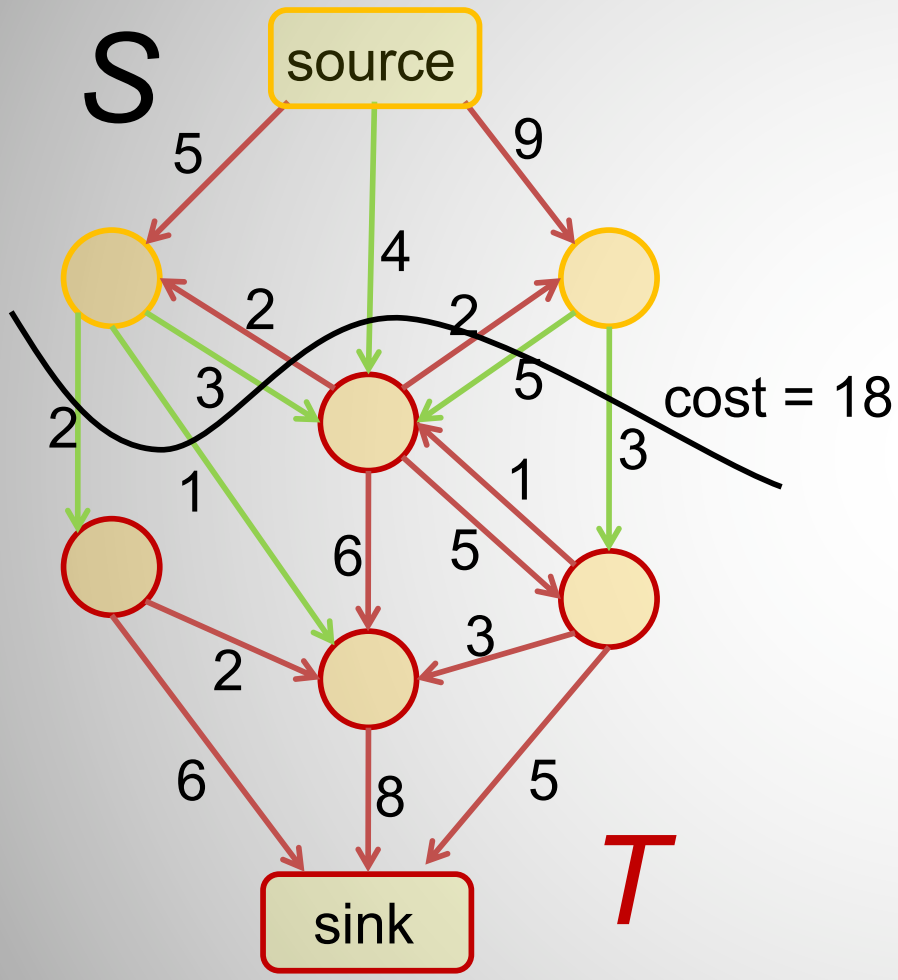
Individual flows obtained by summing up all paths

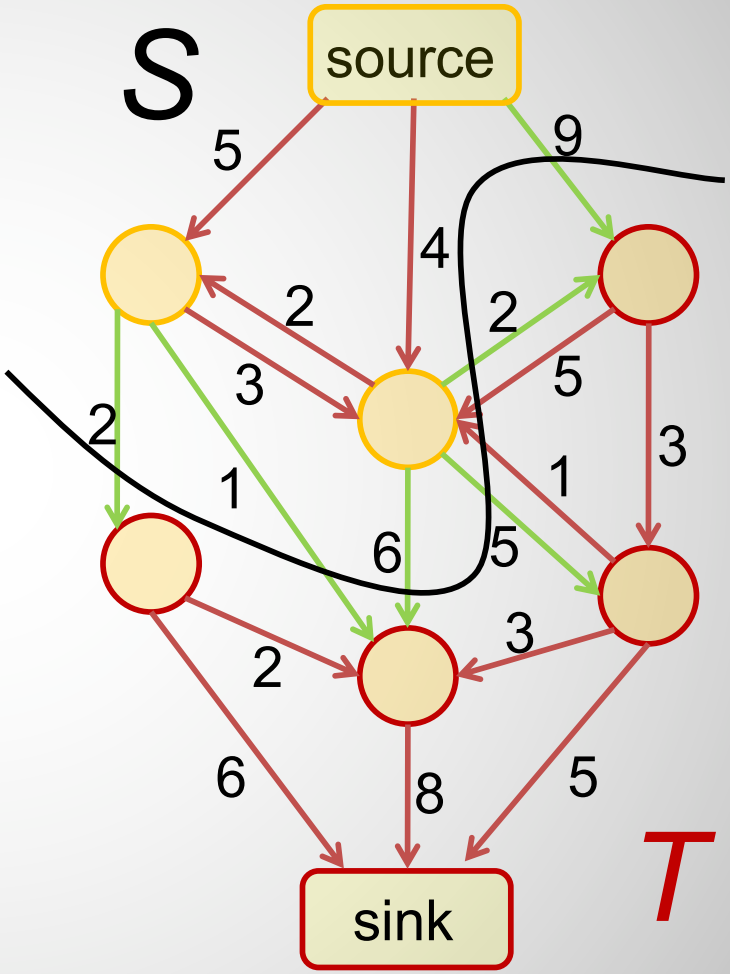
Ford & Fulkerson algorithm (1956)

Why is the solution globally optimal ?

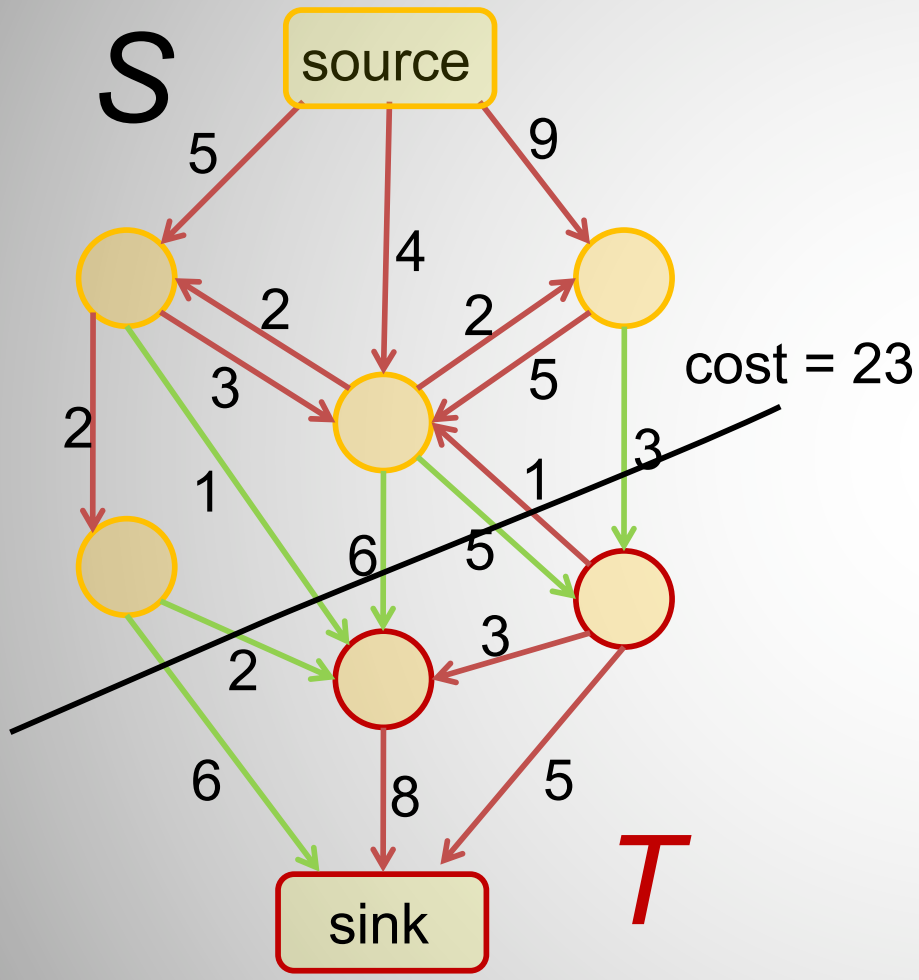
1. Let S be the set of reachable nodes in the residual graph
2. The flow from S to V - S equals to the sum of capacities from S to V – S
3. The flow from any A to V - A is upper bounded by the sum of capacities from A to V – A
4. The solution is globally optimal

Another Example-Max Flow

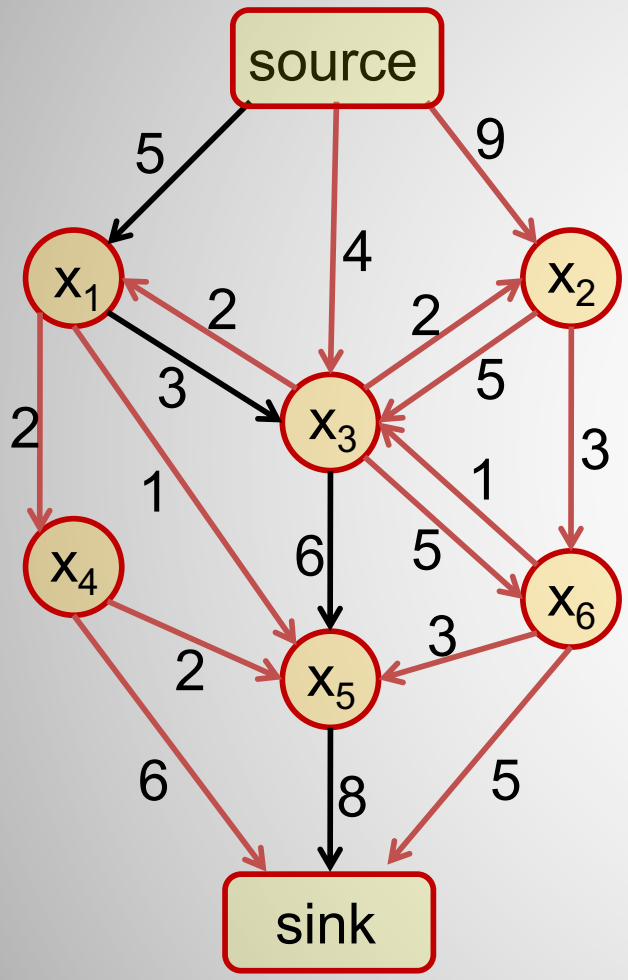




Another Example-Max Flow

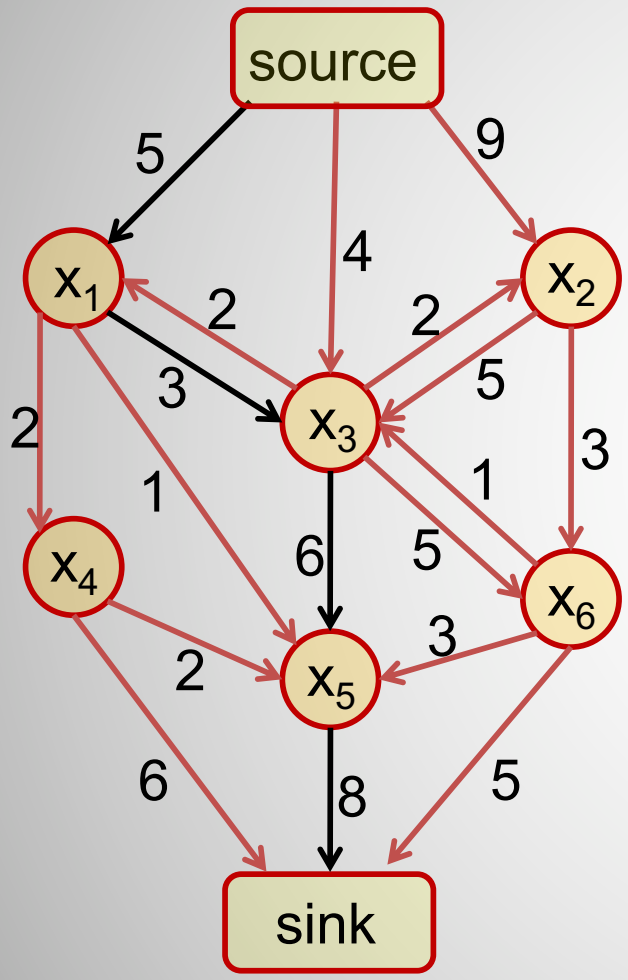


Another Example-Max Flow



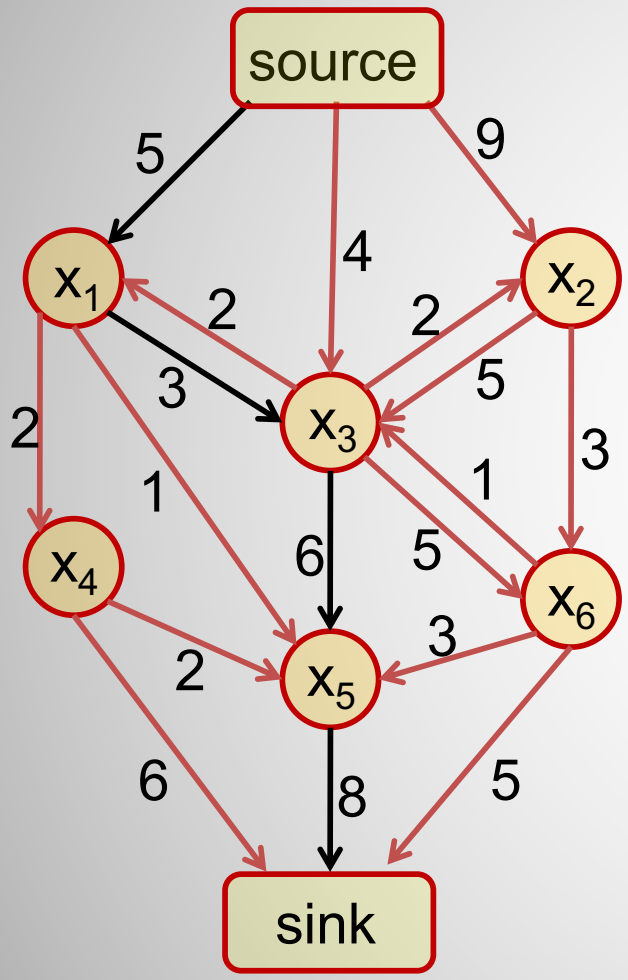
$$C(\boldsymbol{x}) = 5x_1 + 9x_2 + 4x_3 + 3x_3(1-x_1) + 2x_1(1-x_3)$$
$$+ 3x_3(1-x_1) + 2x_2(1-x_3) + 5x_3(1-x_2) + 2x_4(1-x_1)$$
$$+ 1x_5(1-x_1) + 6x_5(1-x_3) + 5x_6(1-x_3) + 1x_3(1-x_6)$$
$$+ 3x_6(1-x_2) + 2x_4(1-x_5) + 3x_6(1-x_5) + 6(1-x_4)$$
$$+ 8(1-x_5) + 5(1-x_6)$$

Another Example-Max Flow



$$C(\boldsymbol{x}) = 2x_1 + 9x_2 + 4x_3 + 2x_1(1-x_3)$$
$$+ 3x_3(1-x_1) + 2x_2(1-x_3) + 5x_3(1-x_2) + 2x_4(1-x_1)$$
$$+ 1x_5(1-x_1) + 3x_5(1-x_3) + 5x_6(1-x_3) + 1x_3(1-x_6)$$
$$+ 3x_6(1-x_2) + 2x_4(1-x_5) + 3x_6(1-x_5) + 6(1-x_4)$$
$$+ 5(1-x_5) + 5(1-x_6)$$
$$+ 3x_1 + 3x_3(1-x_1) + 3x_5(1-x_3) + 3(1-x_5)$$

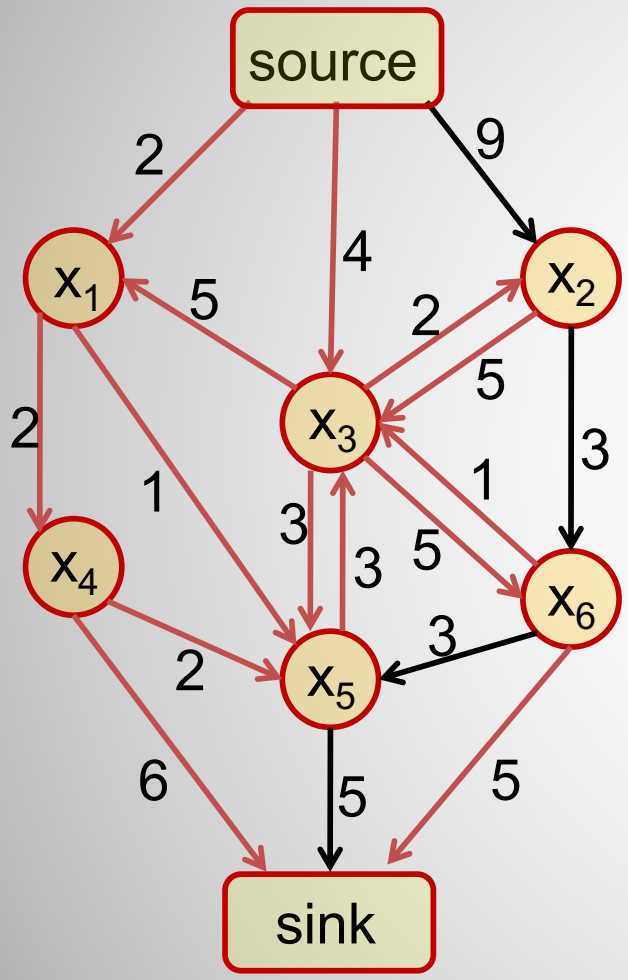
Another Example-Max Flow



$$C(\boldsymbol{x}) = 2x_1 + 9x_2 + 4x_3 + 2x_1(1-x_3)$$
$$+ 3x_3(1-x_1) + 2x_2(1-x_3) + 5x_3(1-x_2) + 2x_4(1-x_1)$$
$$+ 1x_5(1-x_1) + 3x_5(1-x_3) + 5x_6(1-x_3) + 1x_3(1-x_6)$$
$$+ 3x_6(1-x_2) + 2x_4(1-x_5) + 3x_6(1-x_5) + 6(1-x_4)$$
$$+ 5(1-x_5) + 5(1-x_6)$$
$$+ 3x_1 + 3x_3(1-x_1) + 3x_5(1-x_3) + 3(1-x_5)$$

$$3x_1 + 3x_3(1-x_1) + 3x_5(1-x_3) + 3(1-x_5)$$
$$=$$
$$3 + 3x_1(1-x_3) + 3x_3(1-x_5)$$

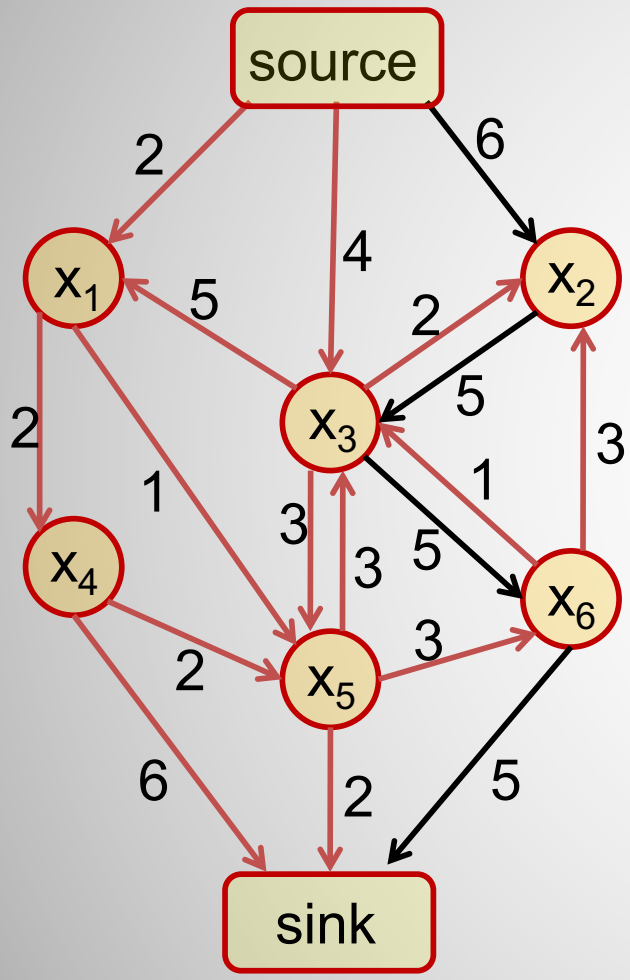
Another Example-Max Flow



$$C(\boldsymbol{x}) = 3 + 2x_1 + 6x_2 + 4x_3 + 5x_1(1-x_3)$$
$$+ 3x_3(1-x_1) + 2x_2(1-x_3) + 5x_3(1-x_2) + 2x_4(1-x_1)$$
$$+ 1x_5(1-x_1) + 3x_5(1-x_3) + 5x_6(1-x_3) + 1x_3(1-x_6)$$
$$+ 2x_5(1-x_4) + 6(1-x_4)$$
$$+ 2(1-x_5) + 5(1-x_6) + 3x_3(1-x_5)$$
$$+ 3x_2 + 3x_6(1-x_2) + 3x_5(1-x_6) + 3(1-x_5)$$

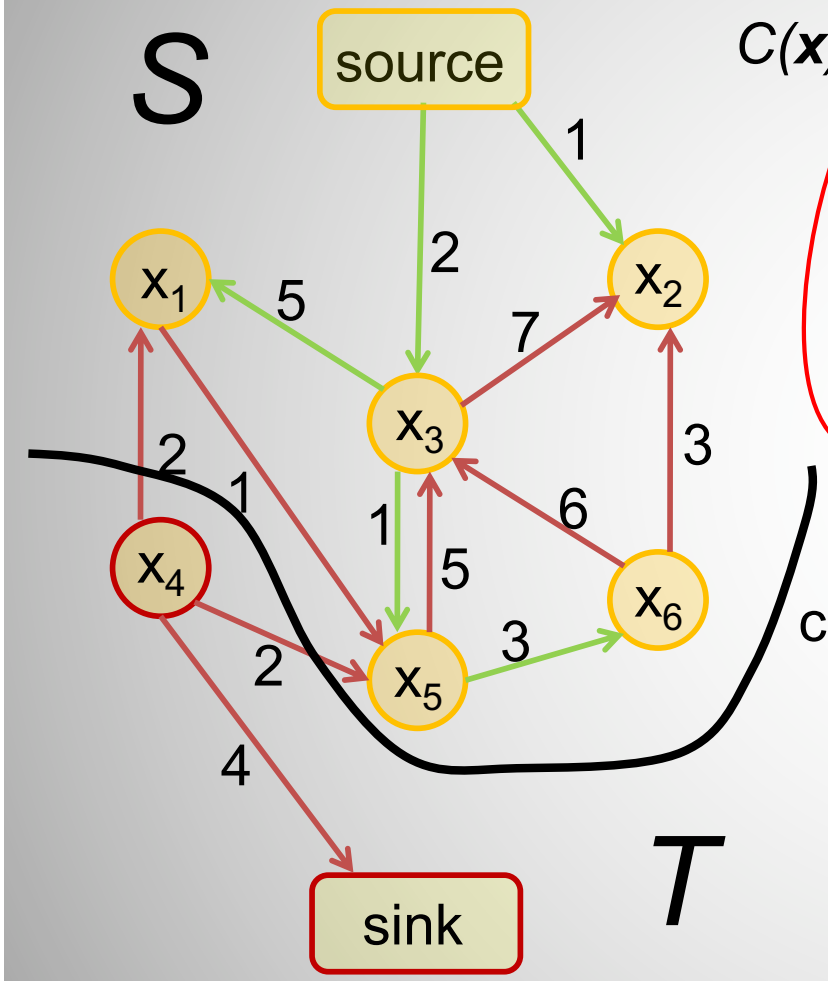
$$3x_2 + 3x_6(1-x_2) + 3x_5(1-x_6) + 3(1-x_5)$$
$$=$$
$$3 + 3x_2(1-x_6) + 3x_6(1-x_5)$$

Another Example-Max Flow



$$C(\boldsymbol{x}) = 6 + 2x_1 + 6x_2 + 4x_3 + 5x_1(1-x_3)$$
$$+ 3x_3(1-x_1) + 2x_2(1-x_3) + 5x_3(1-x_2) + 2x_4(1-x_1)$$
$$+ 1x_5(1-x_1) + 3x_5(1-x_3) + 5x_6(1-x_3) + 1x_3(1-x_6)$$
$$+ 2x_5(1-x_4) + 6(1-x_4) + 3x_2(1-x_6) + 3x_6(1-x_5)$$
$$+ 2(1-x_5) + 5(1-x_6) + 3x_3(1-x_5)$$

Another Example-Max Flow



min cut

$$C(\boldsymbol{x}) = 15 + 1x_2 + 4x_3 + 5x_1(1-x_3)$$
$$+ 3x_3(1-x_1) + 7x_2(1-x_3) + 2x_1(1-x_4)$$
$$+ 1x_5(1-x_1) + 6x_3(1-x_6) + 6x_3(1-x_5)$$
$$+ 2x_5(1-x_4) + 4(1-x_4) + 3x_2(1-x_6) + 3x_6(1-x_5)$$

cost = 0

cost = 0

- All coefficients positive
- Must be global minimum

S – set of reachable nodes from s

History of Max-Flow Algorithms

Augmenting Path and Push-Relabel

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

n: #nodes

m: #edges

U: maximum edge weight

Algorithms assume non-negative edge weights

Ford-Fulkerson Alg.

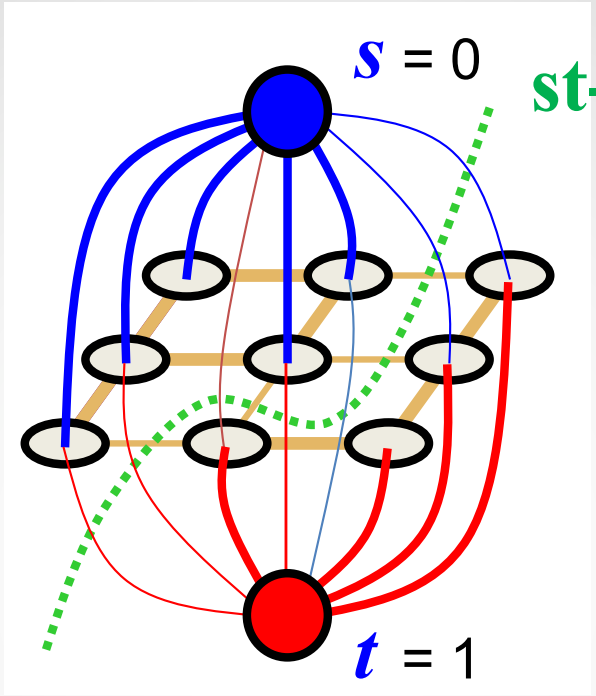
```
FORD-FULKERSON(G, s, t)
1  for each edge (u, v) □ E[G]
2       do f[u, v] ← 0
3          f[v, u] ← 0
4  while there exists a path p from s to t in the residual network G_f
5      do c_f(p) ← min {c_f(u, v) : (u, v) is in p}
6         for each edge (u, v) in p
7             do f[u, v] ← f[u, v] + c_f(p)
8                f[v, u] ← -f[u, v]
```

In each iteration of the Ford-Fulkerson method, we find some augmenting path **p** and increase the flow **f** on each edge of **p** by the residual capacity $\mathbf{c_f(p)}$.

If u and v are not connected by an edge in either direction, we assume implicitly that f[u, v] = 0.

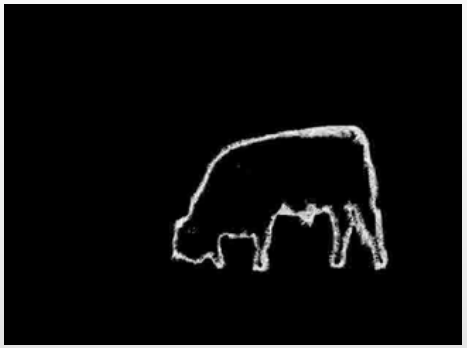
The capacities c(u, v) are assumed to be given along with the graph, and c(u, v) = 0 if $(u, v) \notin E$.

Example Segmentation in a Video

E(x) → 

→ x^*

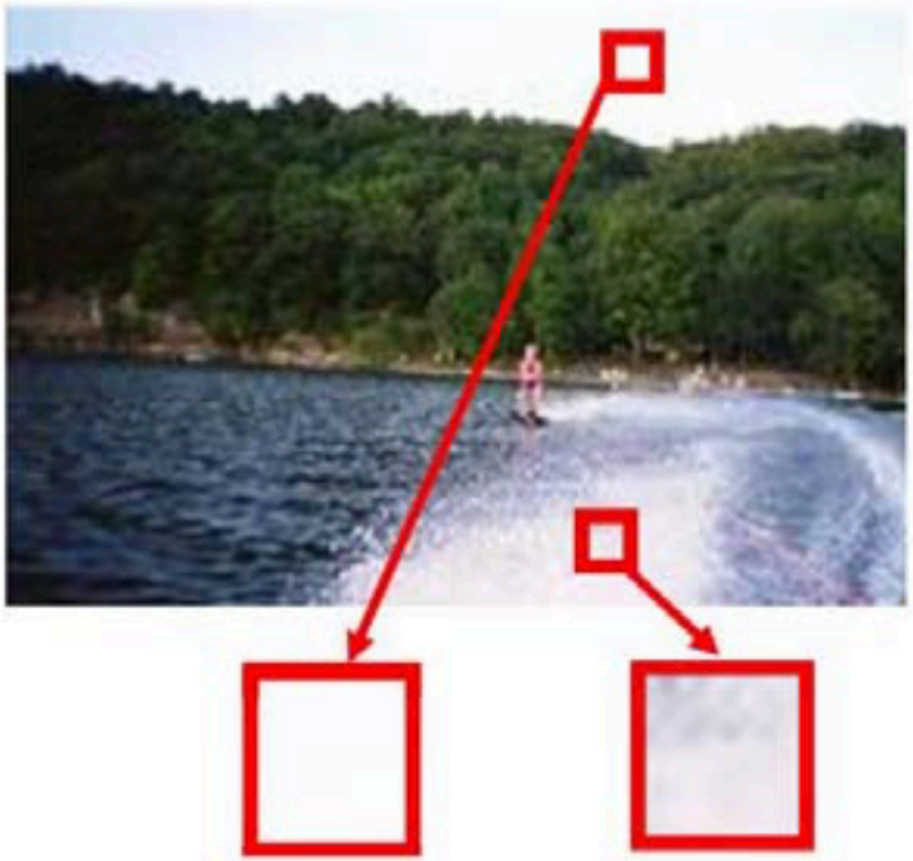
Image

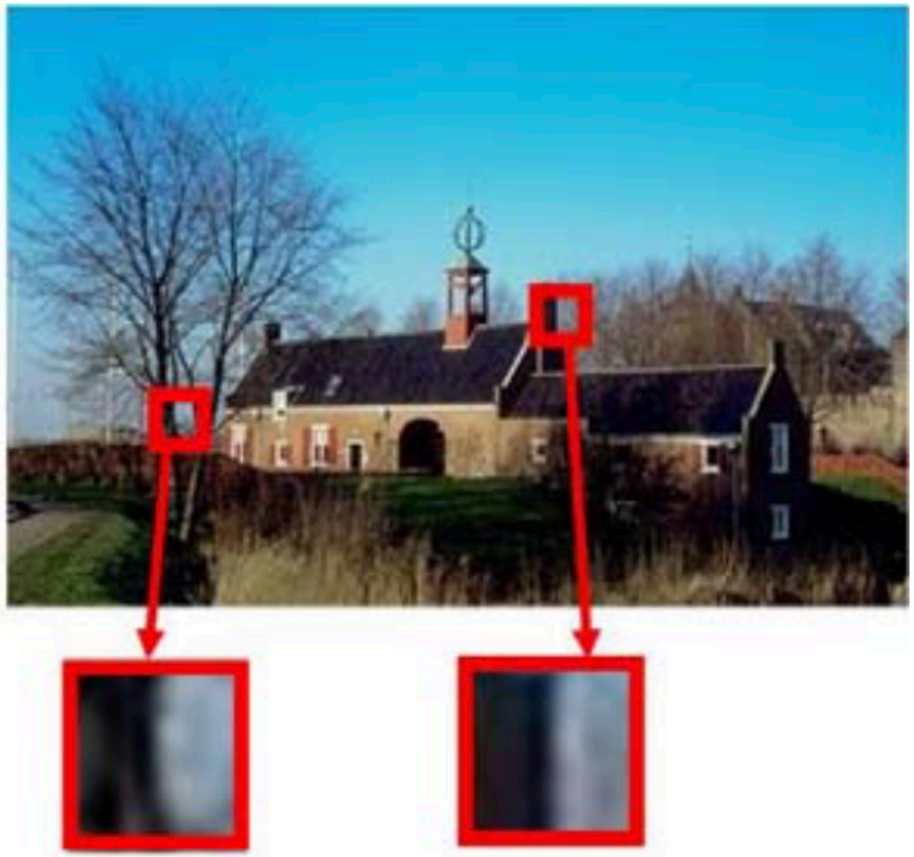


Flow

Global Optimum

Recap: Data Term

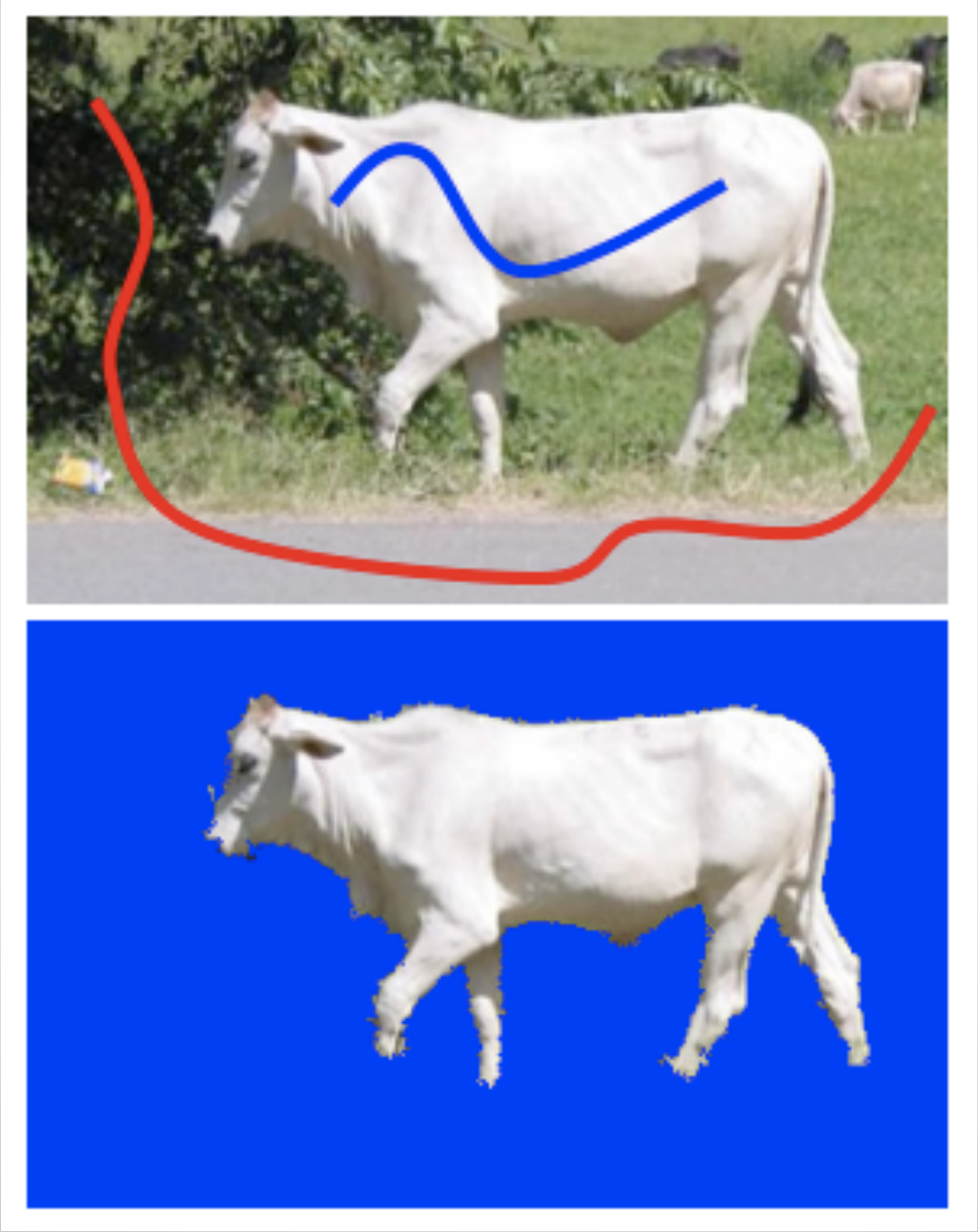




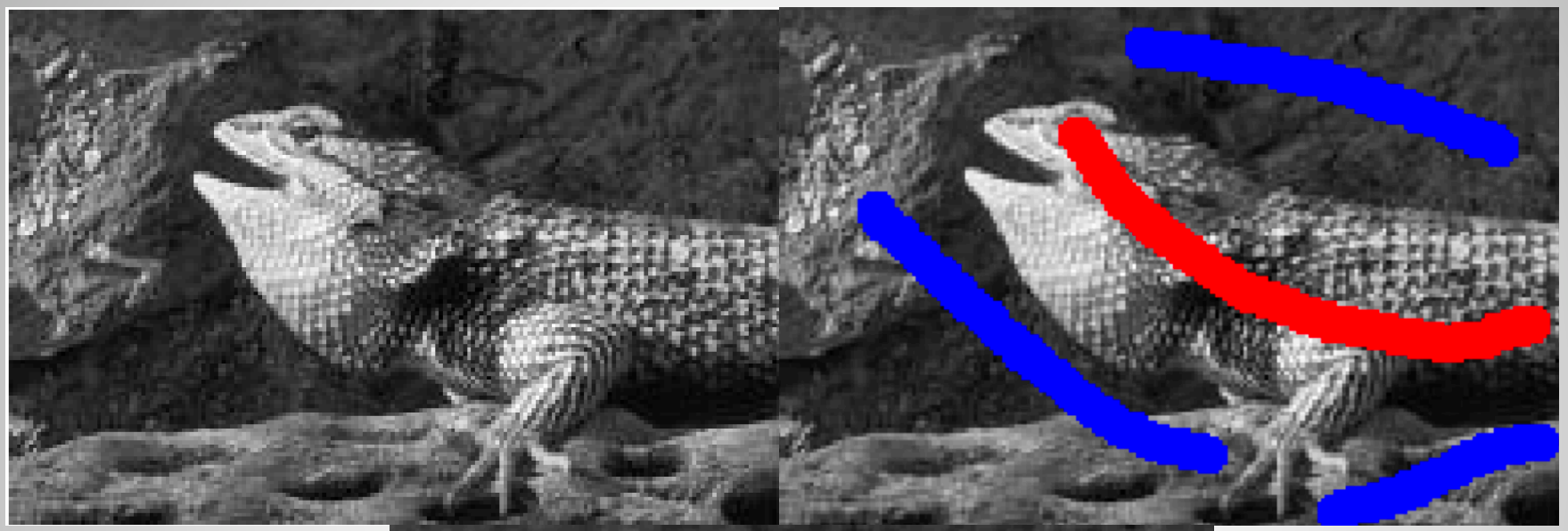
Without spatial relationship, data term can be the same for regions that Belong to different objects.

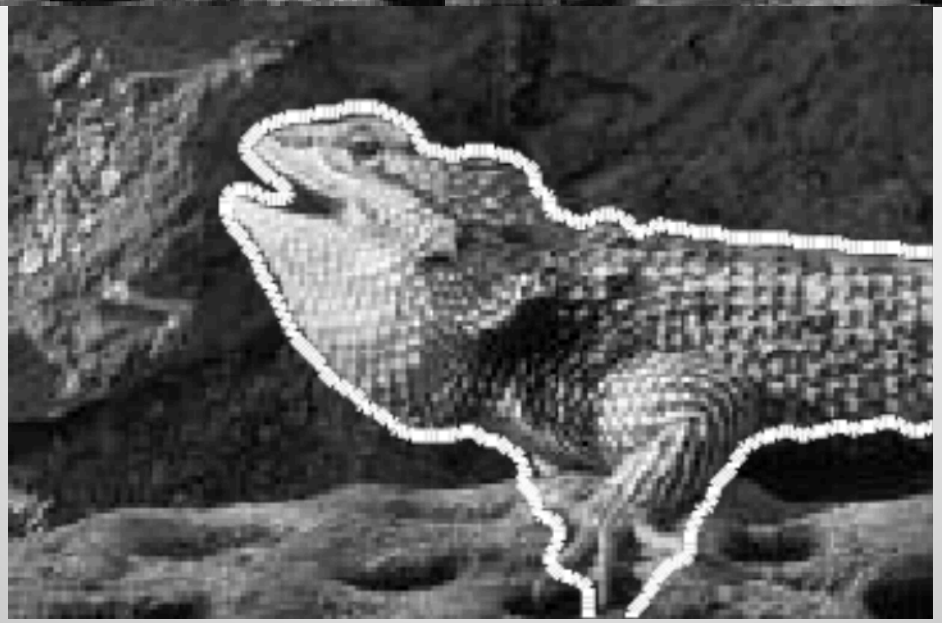
Image patches show their similarities although they pertain to different objects

Segmentation Example

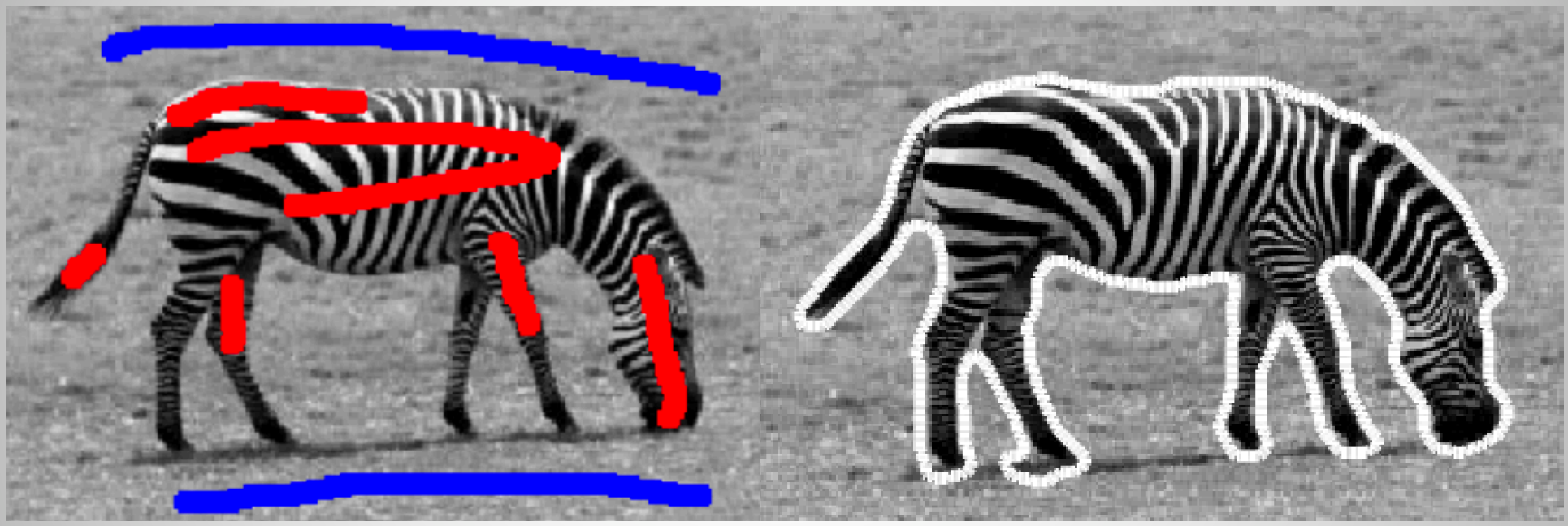


Segmentation Example

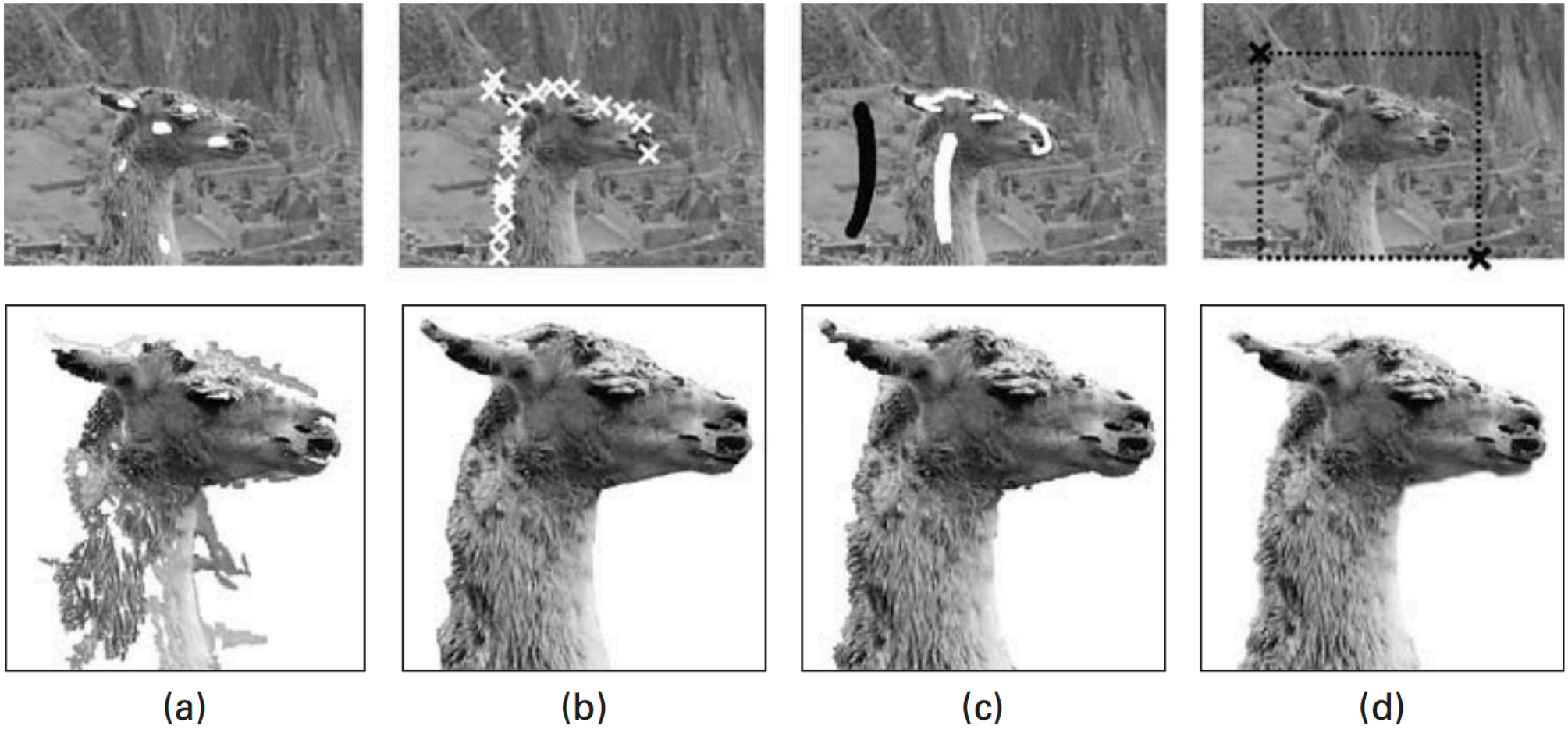




Segmentation Example



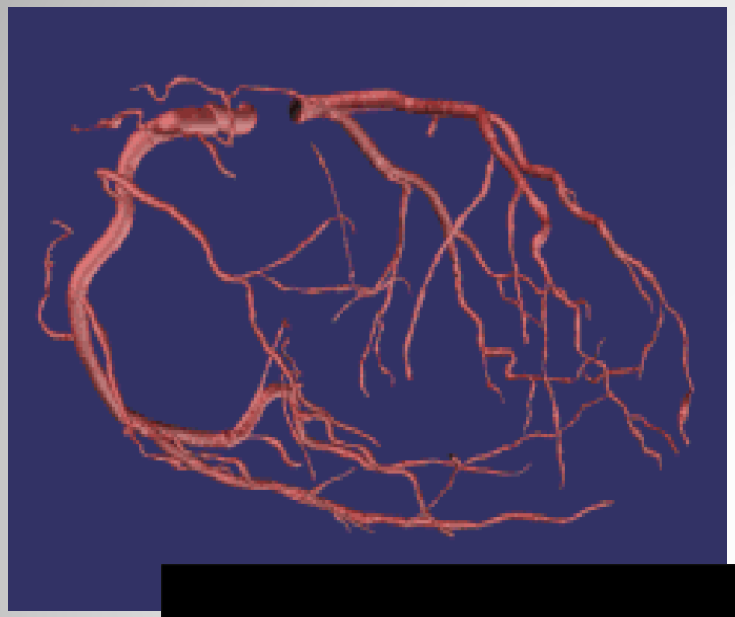
Optimality?

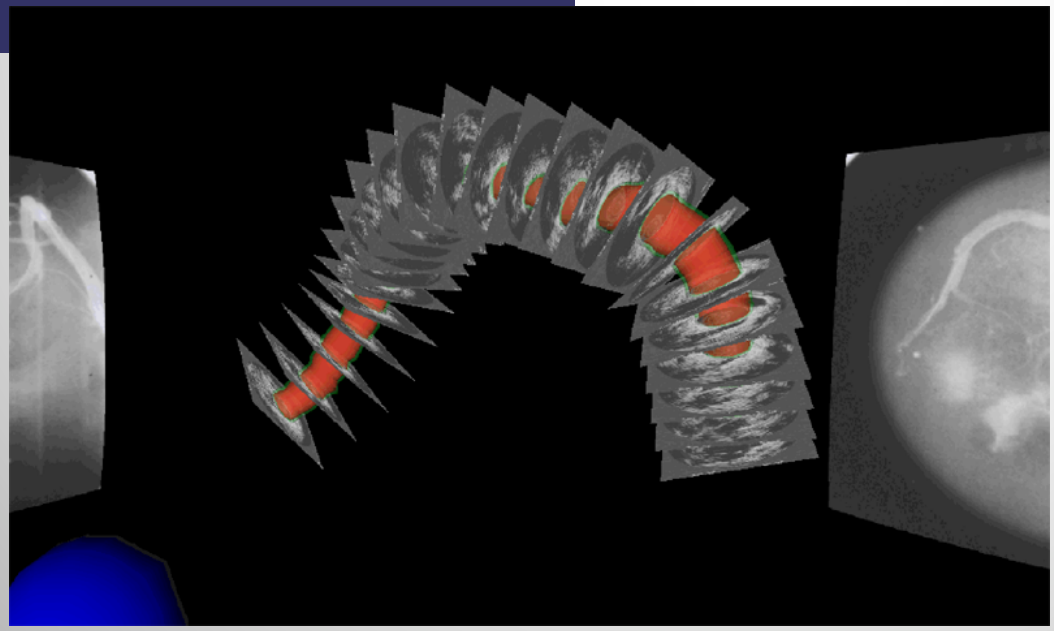


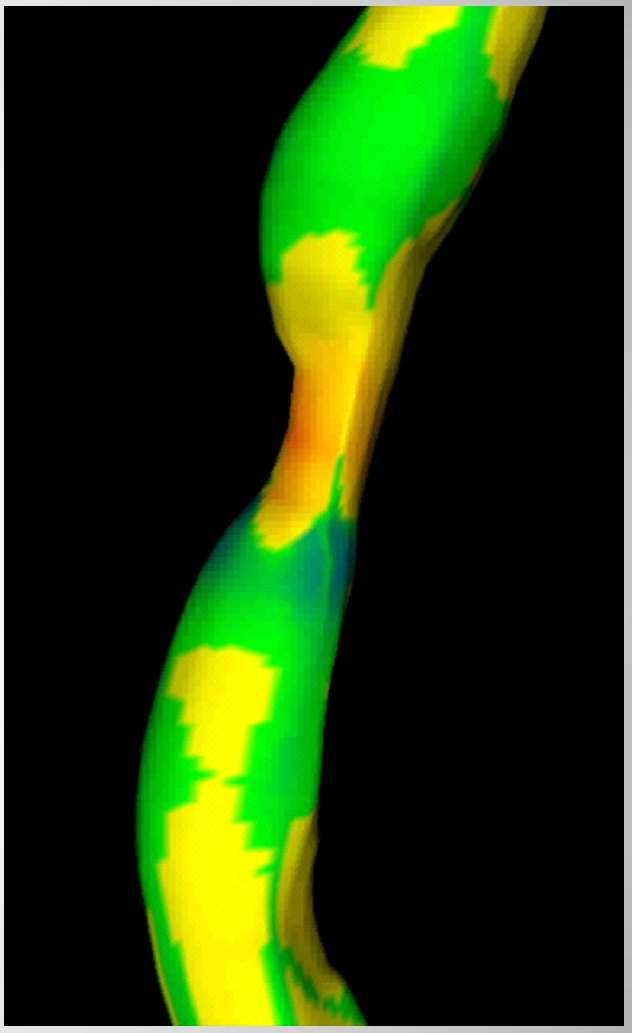
Seeds

GrabCut (iterative GC With box prior)

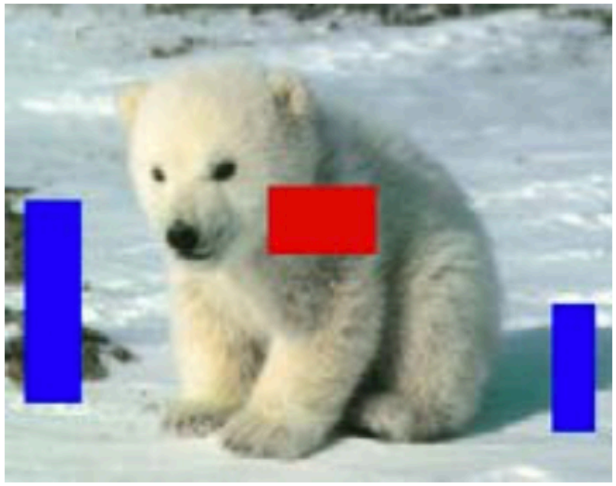
Vascular Surface Segmentation via GC







GC Refinement



(a)

(b)

(c)

Kohli and Torr

GC Segmentation in Videos

Segmentation of human lame walk in a video sequences
Kohli and Torr

Slide Credits and References

- Fredo Durand of MIT
- M. Tappen of Amazon
- R. Szelisky, Univ. of Washington/Seatle
- http://www.csd.uwo.ca/faculty/yuri/Abstracts/eccv06-tutorial.html
- J.Malcolm, Graph Cut in Tensor Scale
- Interactive Graph Cuts for Optimal Boundary & Region Segmentation of Objects in N-D images.
  Yuri Boykov and Marie-Pierre Jolly.
  In International Conference on Computer Vision, (ICCV), vol. I, 2001.
  http://www.csd.uwo.ca/~yuri/Abstracts/iccv01-abs.html
- http://www.cse.yorku.ca/~aaw/Wang/MaxFlowStart.htm
- http://research.microsoft.com/vision/cambridge/i3l/segmentation/GrabCut.htm
- http://www.cc.gatech.edu/cpl/projects/graphcuttextures/
- A Comparative Study of Energy Minimization Methods for Markov Random Fields.
  Rick Szeliski, Ramin Zabih, Daniel Scharstein, Olga Veksler, Vladimir Kolmogorov, Aseem Agarwala, Marshall Tappen, Carsten Rother. ECCV 2006
  www.cs.cornell.edu/~rdz/Papers/SZSVKATR.pdf
- P. Kumar, Oxford University.