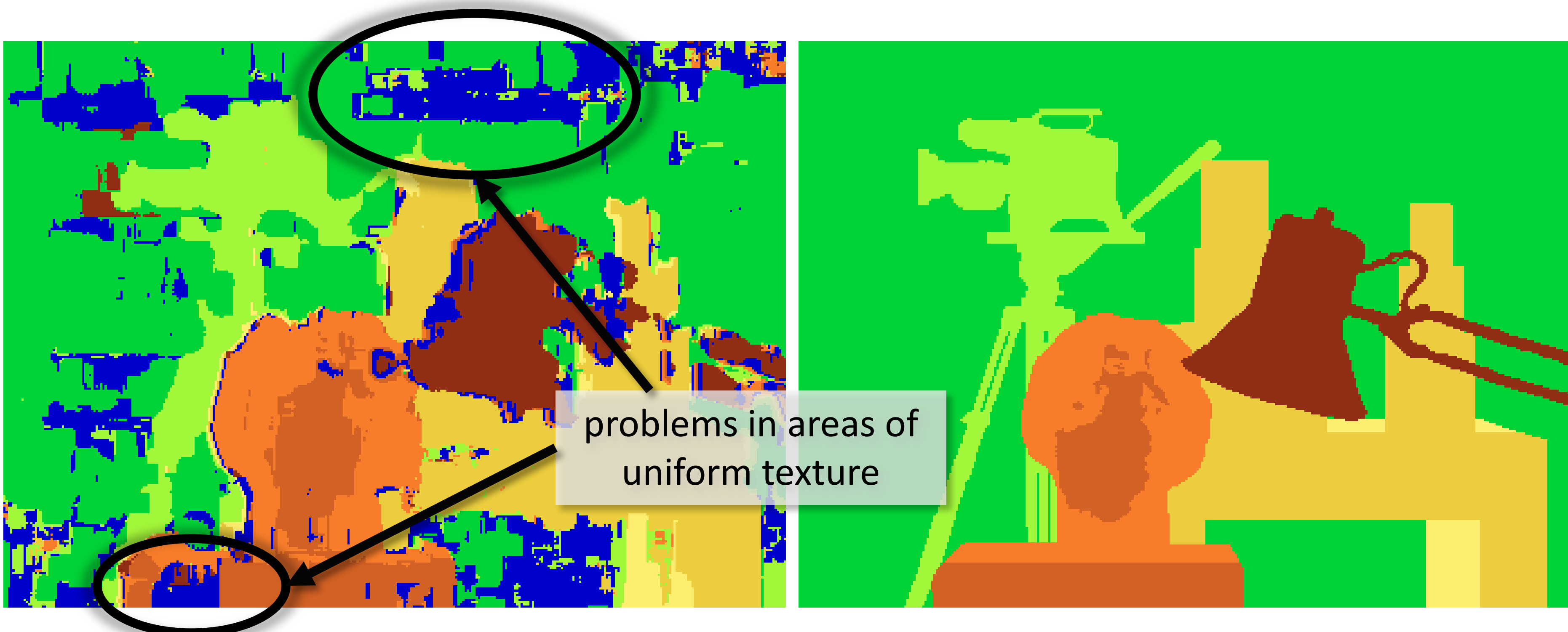
Graph Cuts

Vinay P. Namboodiri

- Slide credit to Noah Snavely, Lubor Ladicky

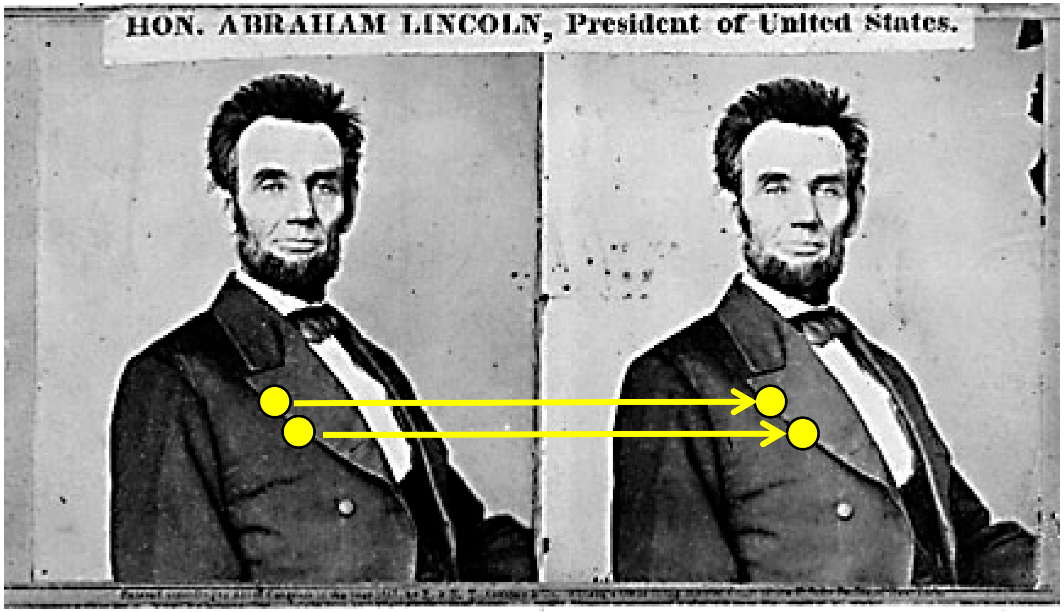
Stereo results with window search



Window-based matching
(best window size)

Ground truth

Can we do better?



- What defines a good stereo correspondence?
  1. Match quality
     - Want each pixel to find a good match in the other image
  2. *Smoothness*
     - two adjacent pixels should (usually) move about the same amount

Stereo as energy minimization

- Better objective function

$$E(d) = \underbrace{E_d(d)}_{\text{match cost}} + \lambda \underbrace{E_s(d)}_{\text{smoothness cost}}$$

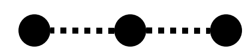
Want each pixel to find a good match in the other image

Adjacent pixels should (usually) move about the same amount

Dynamic programming

$$E(d) = E_d(d) + \lambda E_s(d)$$

- Can minimize this independently per scanline using dynamic programming (DP)

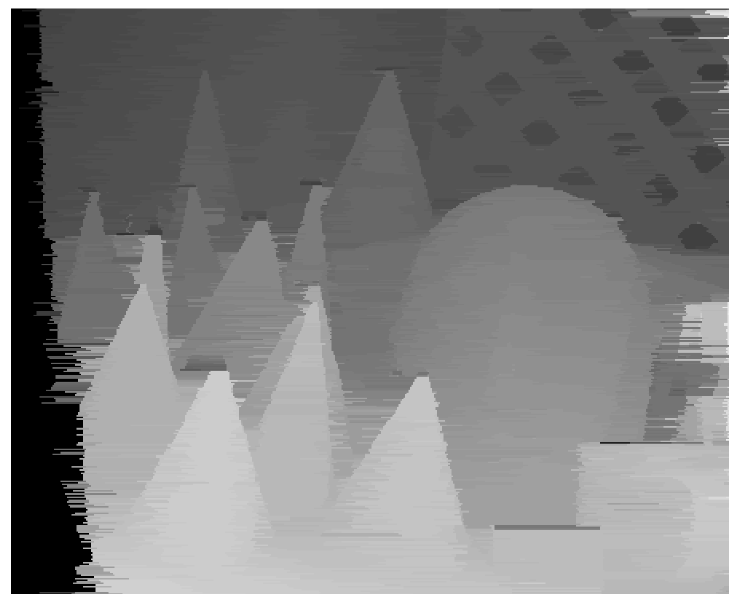


$D(x, y, d)$: minimum cost of solution such that *d*(*x*,*y*) = *d*

$$D(x, y, d) = C(x, y, d) + \min_{d'} \{D(x-1, y, d') + \lambda |d - d'|\}$$

Dynamic Programming





Stereo as a minimization problem

$$E(d) = E_d(d) + \lambda E_s(d)$$

- The 2D problem has many local minima
  - Gradient descent doesn't work well
  - Simulated annealing works a little better
- And a large search space
  - n x m image w/ k disparities has k^{nm} possible solutions
  - Finding the global minimum is NP-hard

- Good approximations exist…

Image Labelling Problems

Assign a label to each image pixel

Geometry Estimation

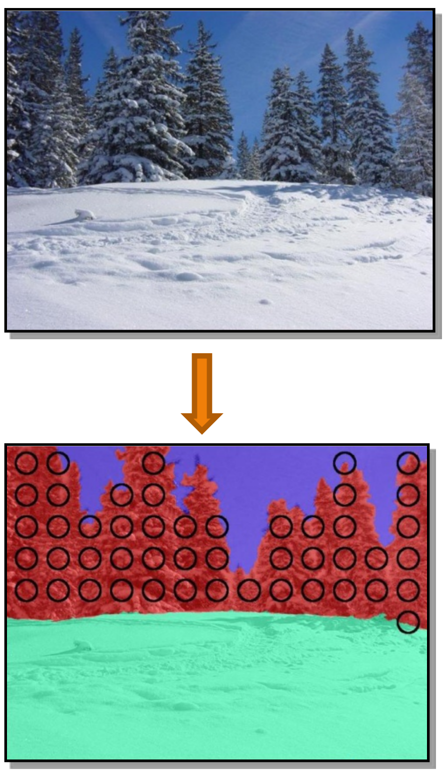
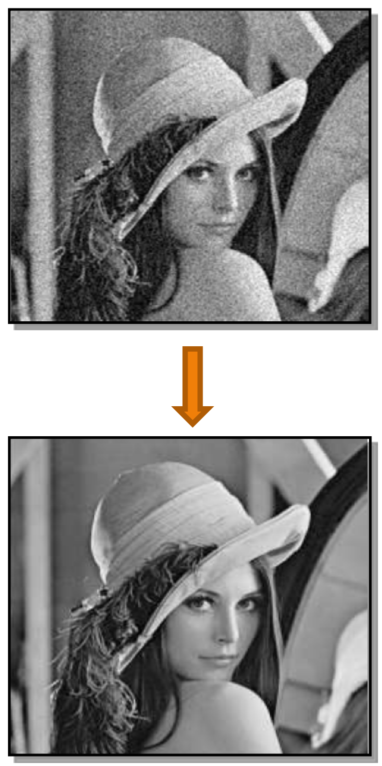
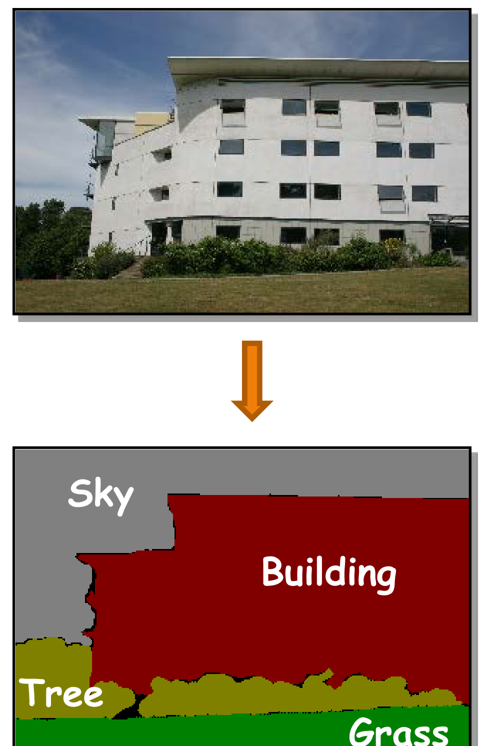


Image Denoising



Object Segmentation



Depth Estimation

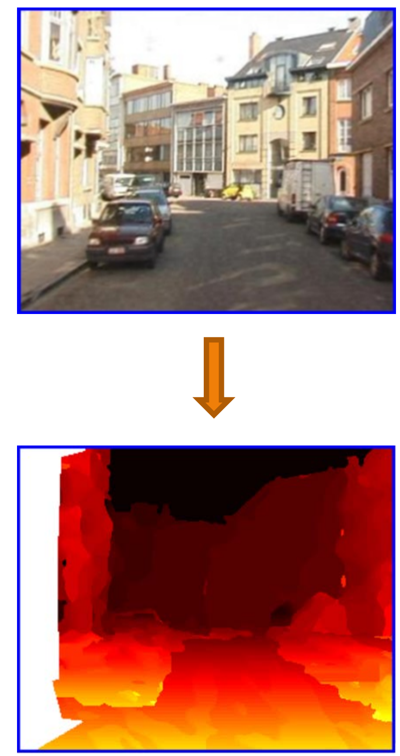
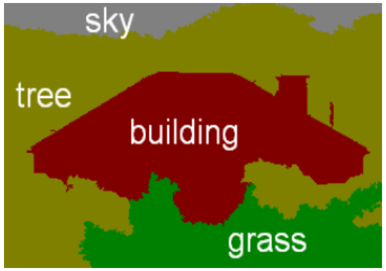
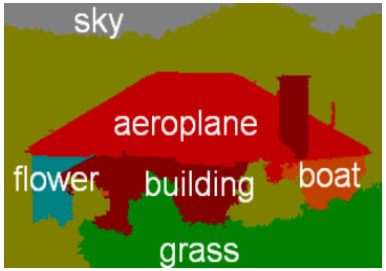


Image Labelling Problems

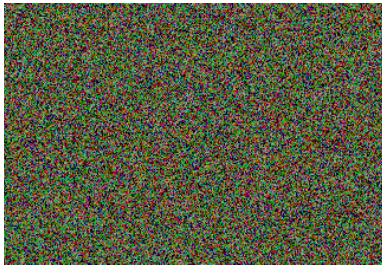
- Labellings highly structured



Possible labelling



Unprobable labelling



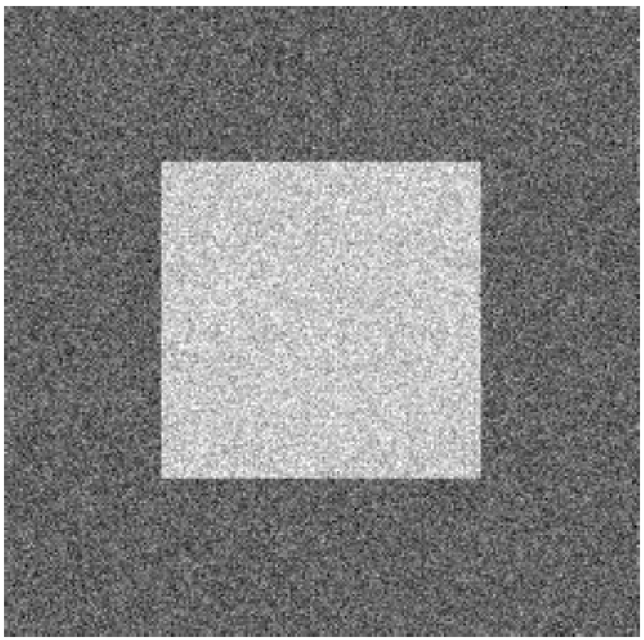
Impossible labelling

Image Labelling Problems

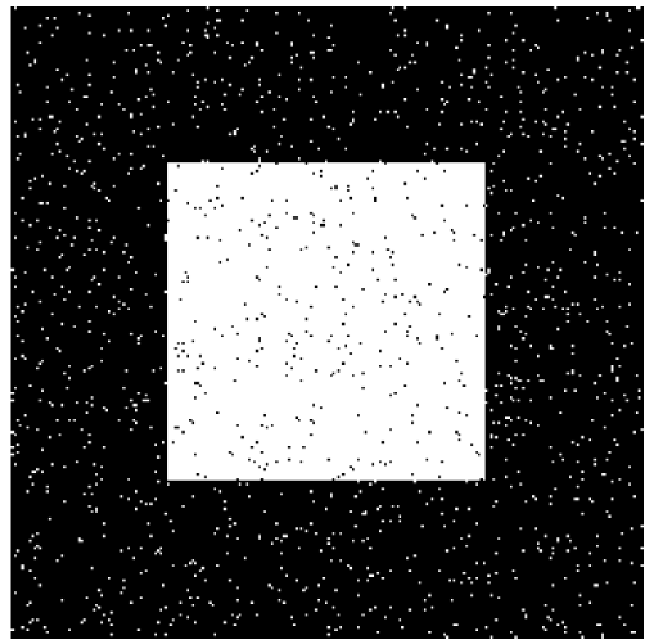
- Labelling highly structured
- Labels highly correlated with very complex dependencies
- Independent label estimation too hard
- Whole labelling should be formulated as one optimisation problem
- Number of pixels up to millions
  - Hard to train complex dependencies
  - Optimisation problem is hard to infer

First: denoising

- Suppose we want to find a bright object against a bright background, given a noisy image



Input



Best thresholded images

Second: binary segmentation

- Suppose we want to segment an image into foreground and background

Binary segmentation as energy minimization

- Define a labeling *L* as an assignment of each pixel with a 0-1 label (background or foreground)

- Problem statement: find the labeling *L* that minimizes

$$E(L) = \underbrace{E_d(L)}_{\text{match cost}} + \lambda \underbrace{E_s(L)}_{\text{smoothness cost}}$$

("how similar is each labeled pixel to the foreground / background?")

$$E(L) = E_d(L) + \lambda E_s(L)$$

$\tilde{L}(x,y)$

$$E_d(L) = \sum_{(x,y)} C(x, y, L(x,y))$$

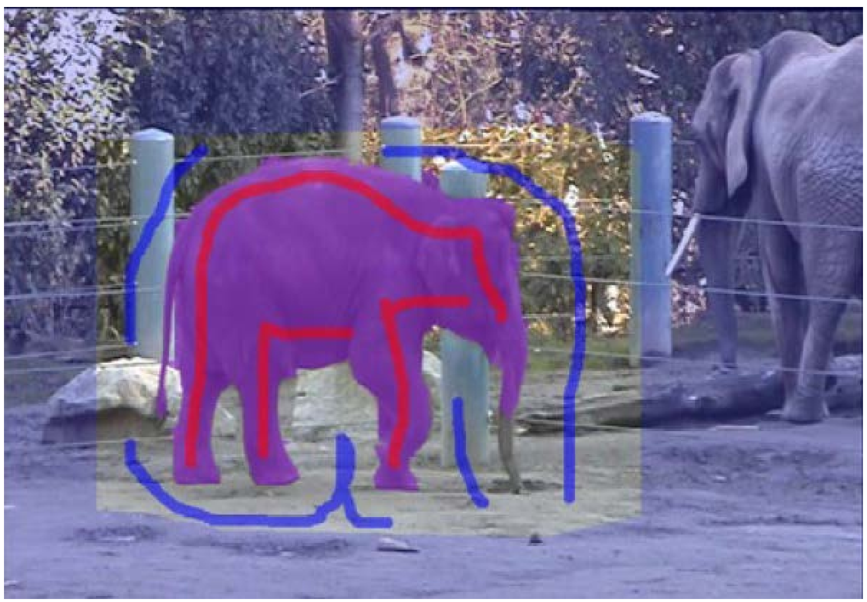
$$C(x,y,L(x,y)) = \begin{cases} \infty & \text{if } L(x,y) \neq \tilde{L}(x,y) \\ C'(x,y,L(x,y)) & \text{otherwise} \end{cases}$$

$C'(x,y,0)$: "distance" from pixel to background pixels

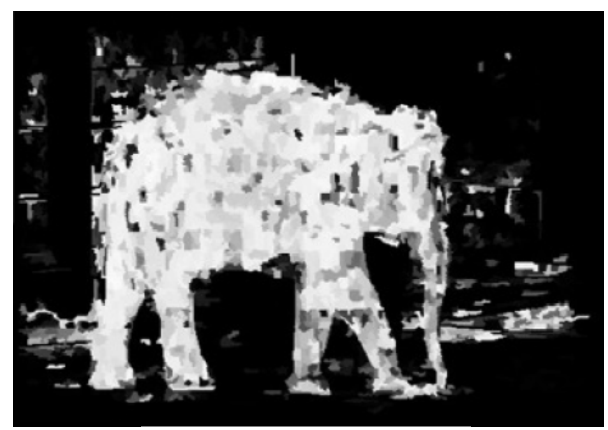
$C'(x,y,1)$: "distance" from pixel to foreground pixels

usually computed by creating a color model from user-labeled pixels

$$E(L) = E_d(L) + \lambda E_s(L)$$



$C'(x, y, 0)$



$C'(x, y, 1)$

$$E(L) = E_d(L) + \lambda E_s(L)$$

- Neighboring pixels should generally have the same labels
  - Unless the pixels have very different intensities

w_{pq} = 10.0

w_{pq} = 0.1

$$E_s(L) = \sum_{\text{neighbors } (p,q)} w_{pq} |L(p) - L(q)|$$

w_{pq} : similarity in intensity of *p* and *q*

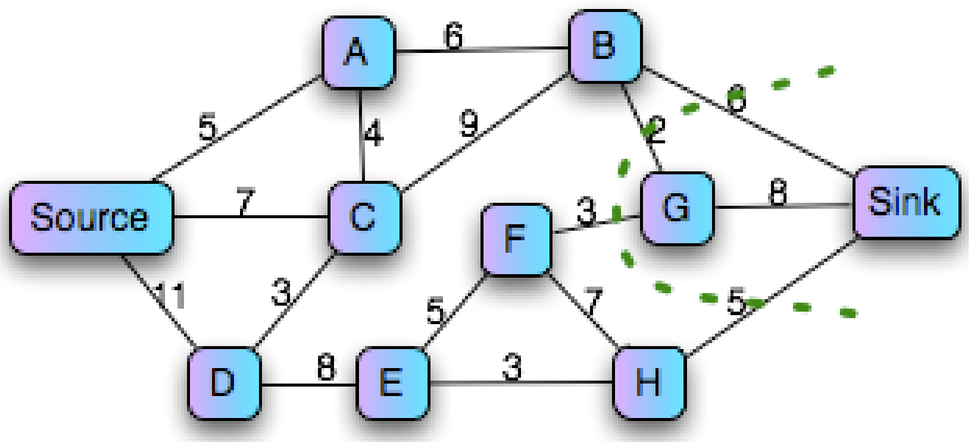
(can use the same trick for stereo)

Binary segmentation as energy minimization

$$E(L) = E_d(L) + \lambda E_s(L)$$

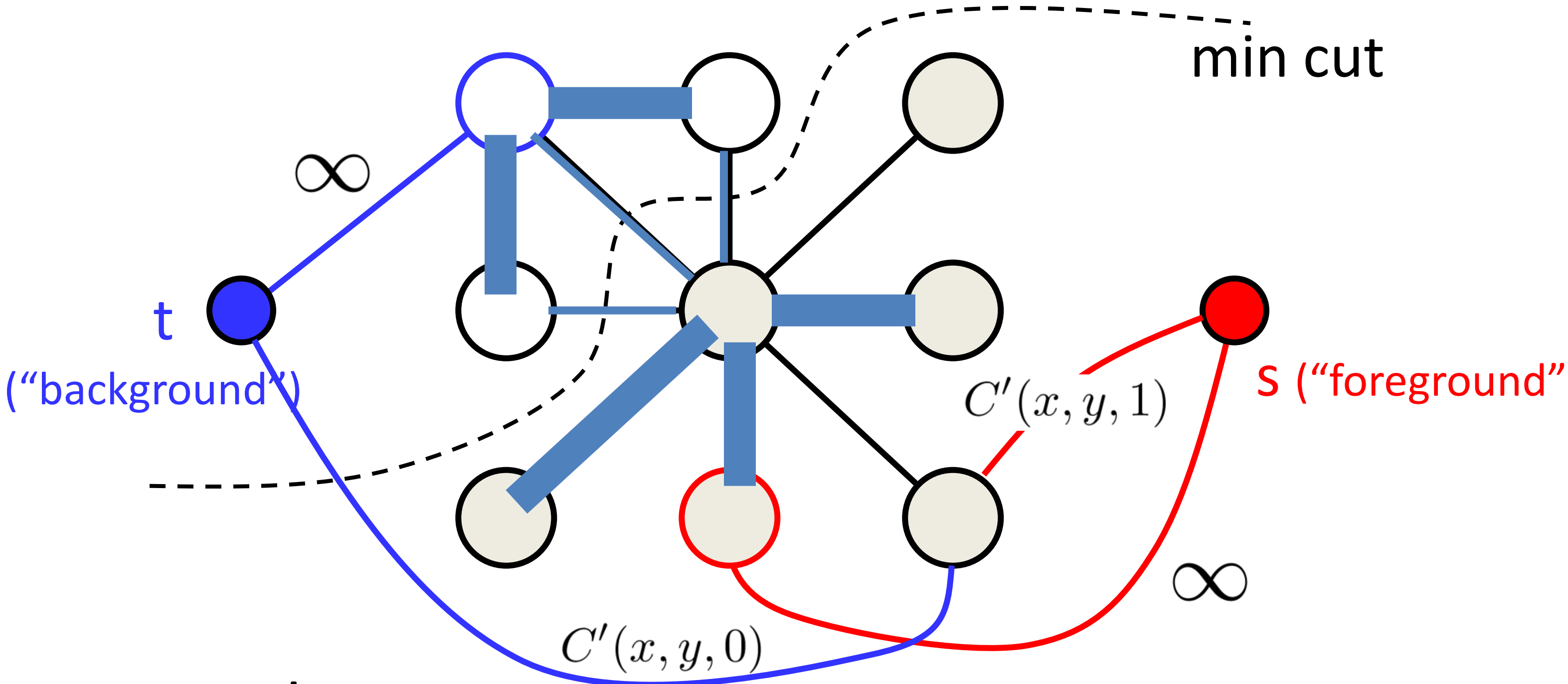
- For this problem, we can quickly find the globally optimal labelling!
- Use max flow / min cut algorithm

Graph min cut problem



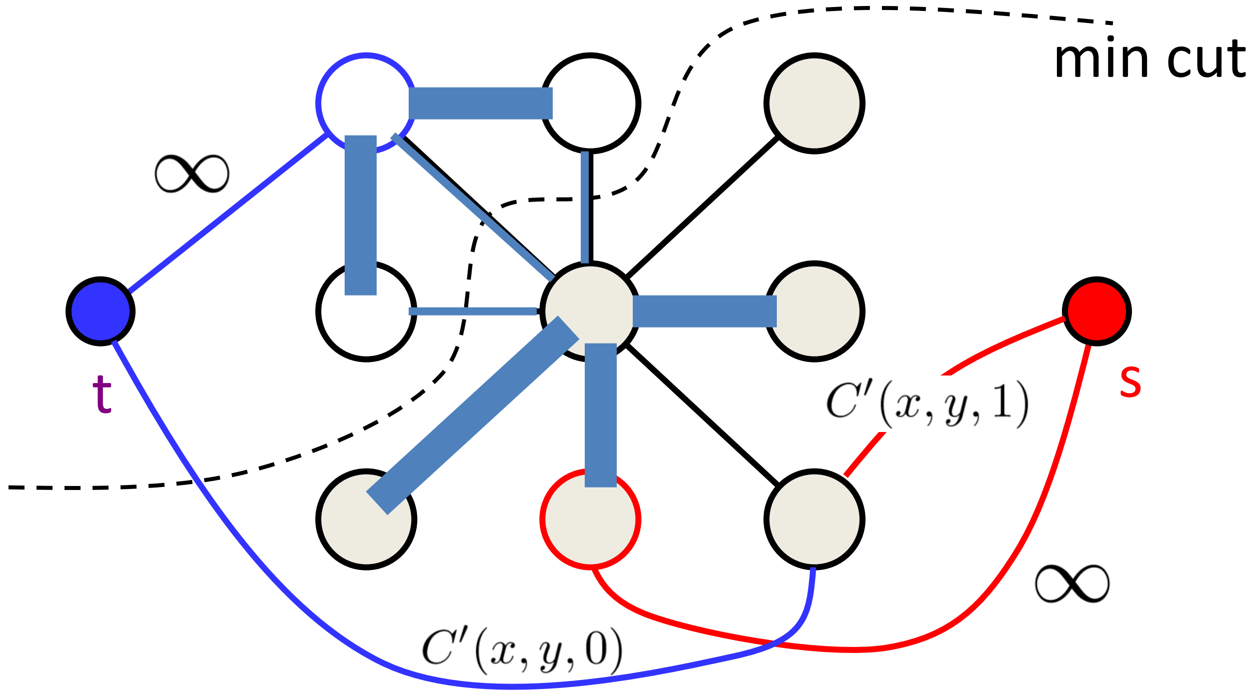
- Given a weighted graph *G* with source and sink nodes (*s* and *t*), partition the nodes into two sets, *S* and *T* such that the sum of edge weights spanning the partition is minimized
  - and $s \in S$ and $t \in T$

Segmentation by min cut



- Graph
  - node for each pixel, link between adjacent pixels
  - specify a few pixels as foreground and background
    - create an infinite cost link from each bg pixel to the *t* node
    - create an infinite cost link from each fg pixel to the *s* node
    - create finite cost links from *s* and *t* to each other node
  - compute min cut that separates s from t
    - The min-cut max-flow theorem [Ford and Fulkerson 1956]

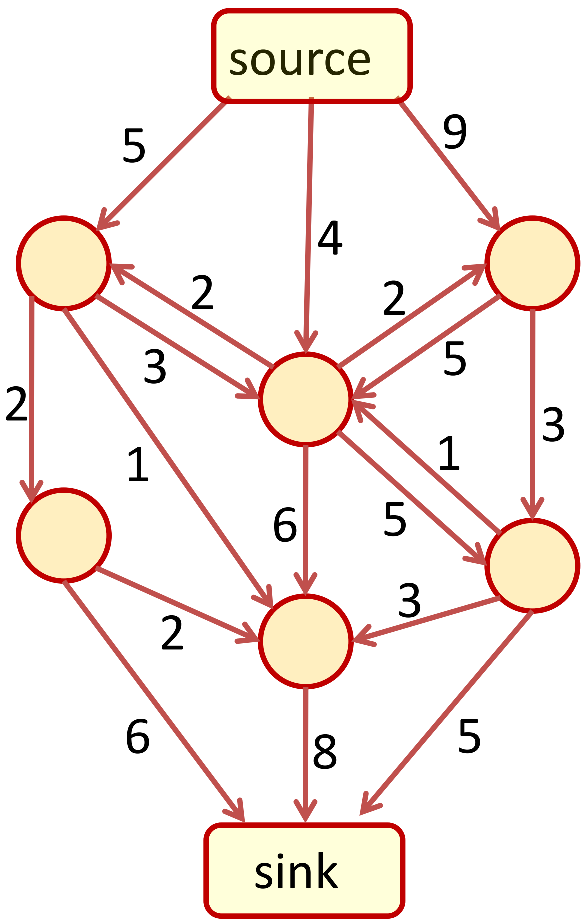
Segmentation by min cut



- The partitions *S* and *T* formed by the min cut give the optimal foreground and background segmentation
- I.e., the resulting labels minimize

$$E(d) = E_d(d) + \lambda E_s(d)$$

Max-Flow Problem



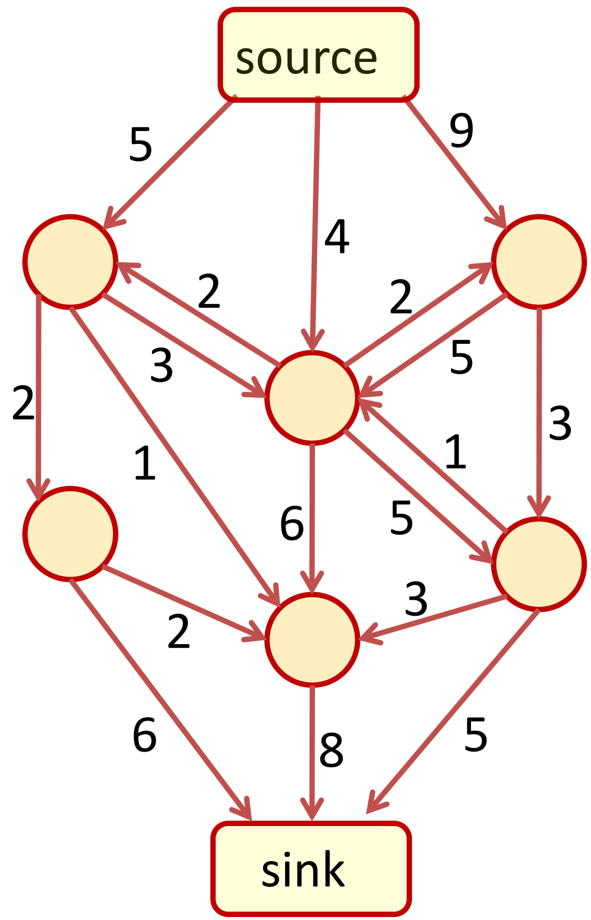
Task :

Maximize the flow from the sink to the source such that

1) The flow it conserved for each node

2) The flow for each pipe does not exceed the capacity

Max-Flow Problem

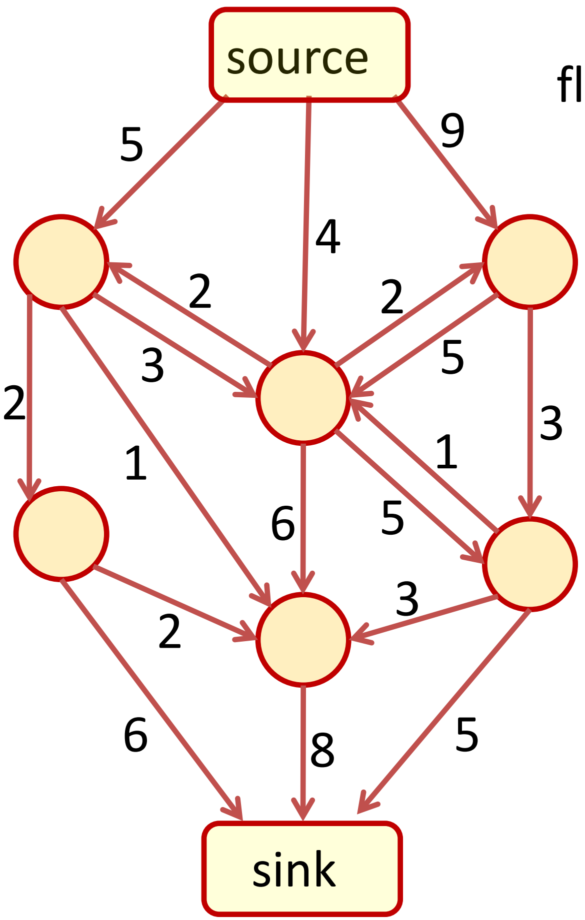


$$\max \textstyle\sum_{i \in V} f_{si}$$

$$\text{s.t.} \qquad 0 \leq f_{ij} \leq c_{ij}, \qquad \forall (i,j) \in E$$

$$\textstyle\sum_{j \in N(i)} f_{ji} - f_{ij} = 0, \qquad \forall i \in V \setminus \{s,t\}$$

Max-Flow Problem



flow from node i to node j

flow from the source

capacity

set of edges

$$\max \boxed{\textstyle\sum_{i \in V} f_{si}}$$

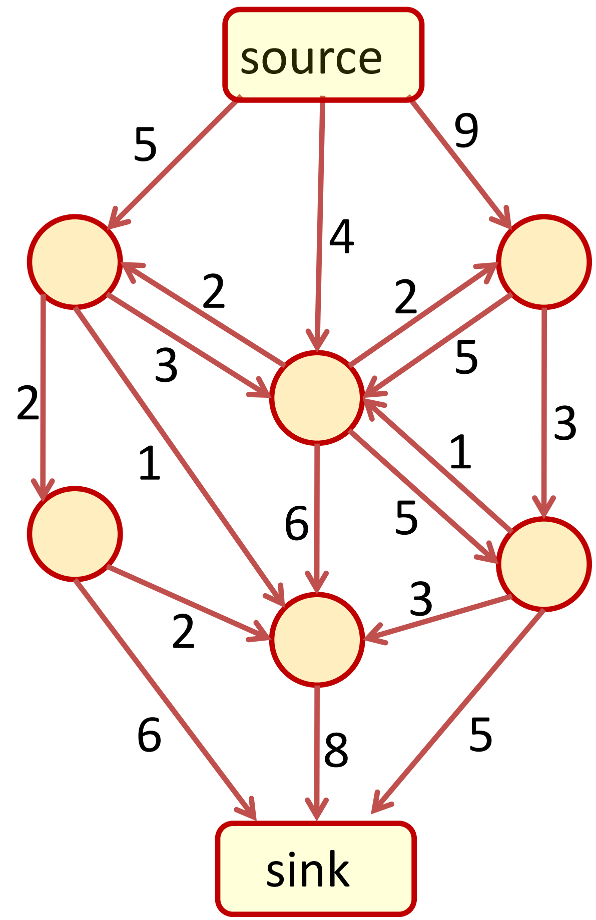
$$\text{s.t.} \qquad 0 \leq \boxed{f_{ij}} \leq \boxed{c_{ij},} \qquad \boxed{\forall (i,j) \in E}$$

$$\boxed{\textstyle\sum_{j \in N(i)} f_{ji} - f_{ij} = 0,} \qquad \boxed{\forall i \in V \setminus \{s,t\}}$$

conservation of flow

set of nodes

Max-Flow Problem



Ford & Fulkerson algorithm (1956)

Find the path from source to sink

While (path exists)

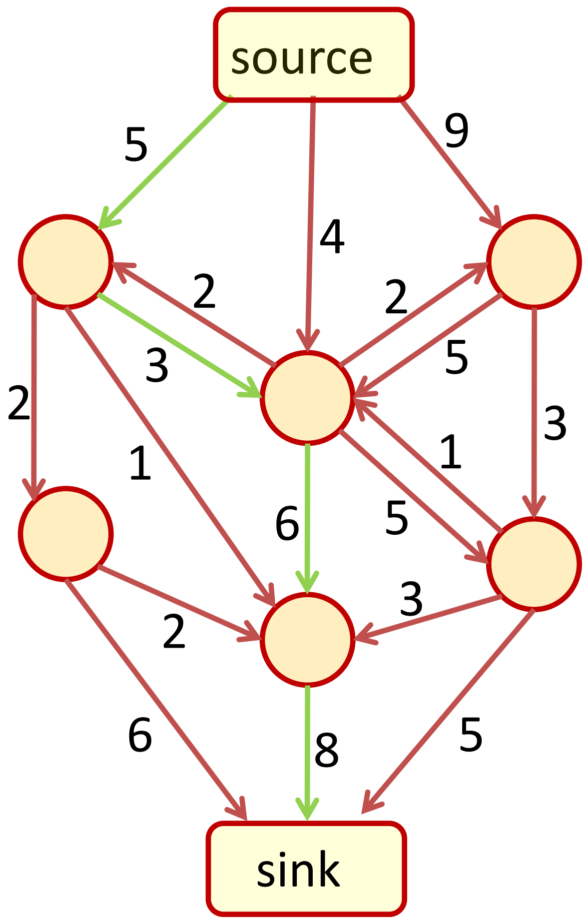
    flow += maximum capacity in the path

    Build the residual graph ("subtract" the flow)

    Find the path in the residual graph

End

Max-Flow Problem



Ford & Fulkerson algorithm (1956)

Find the path from source to sink

While (path exists)

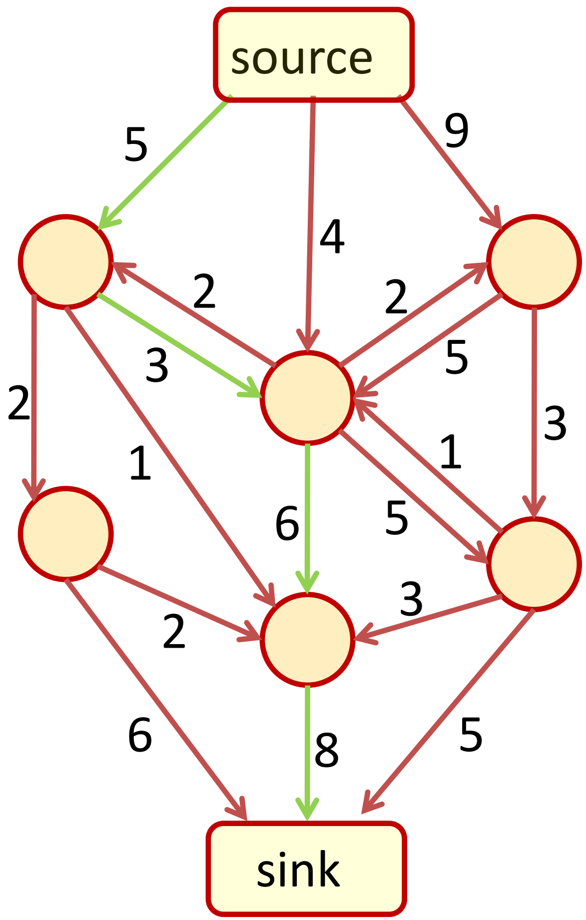
    flow += maximum capacity in the path

    Build the residual graph ("subtract" the flow)

    Find the path in the residual graph

End

Max-Flow Problem



flow = 3

Ford & Fulkerson algorithm (1956)

Find the path from source to sink

While (path exists)

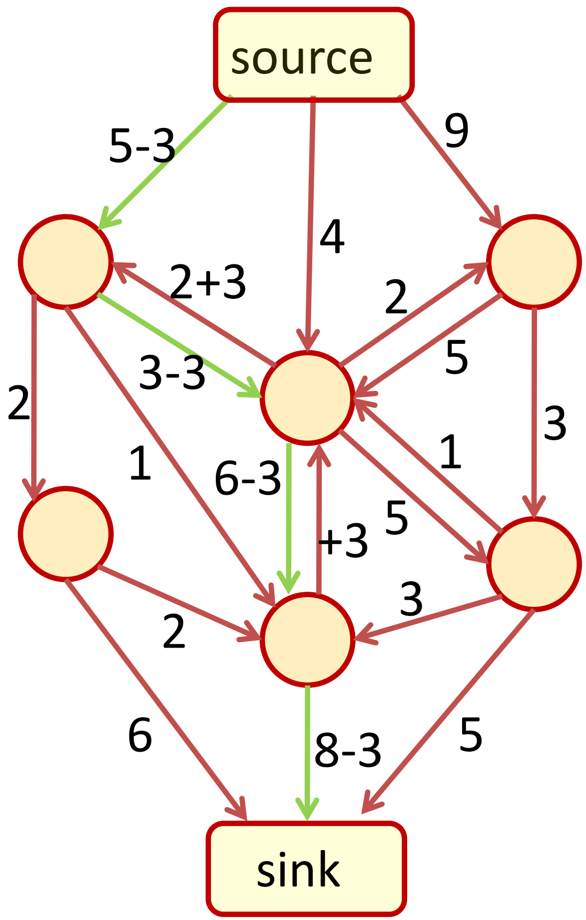
flow += maximum capacity in the path

Build the residual graph ("subtract" the flow)

Find the path in the residual graph

End

Max-Flow Problem



flow = 3

Ford & Fulkerson algorithm (1956)

Find the path from source to sink

While (path exists)

flow += maximum capacity in the path

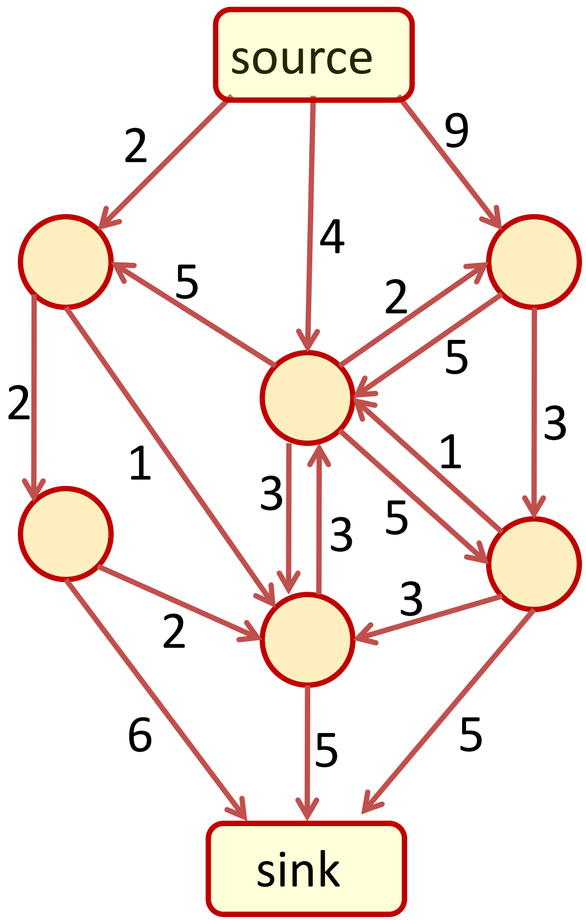
Build the residual graph ("subtract" the flow)

Find the path in the residual graph

End

$$r_{ij} = c_{ij} - f_{ij} + f_{ji}$$

Max-Flow Problem



flow = 3

Ford & Fulkerson algorithm (1956)

Find the path from source to sink

While (path exists)

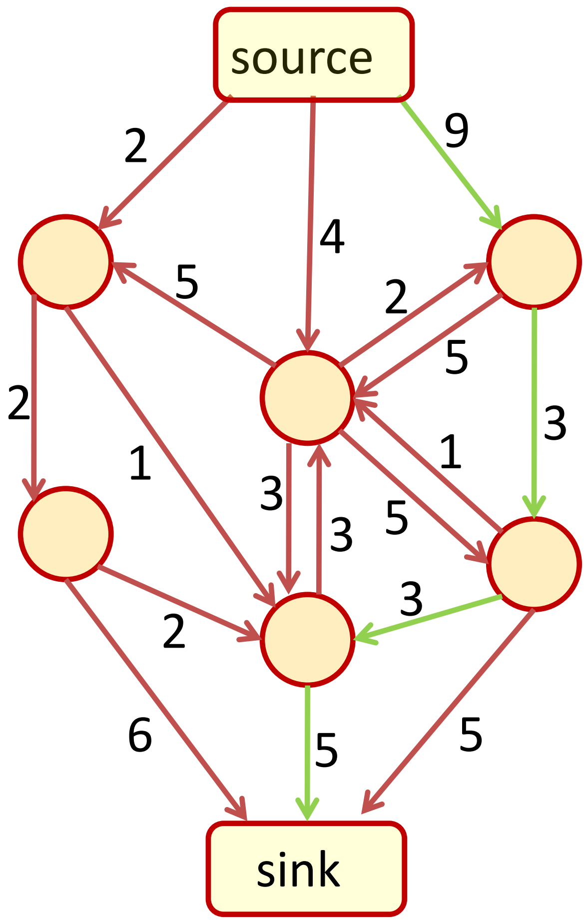
flow += maximum capacity in the path

Build the residual graph ("subtract" the flow)

Find the path in the residual graph

End

Max-Flow Problem



flow = 3

Ford & Fulkerson algorithm (1956)

Find the path from source to sink

While (path exists)

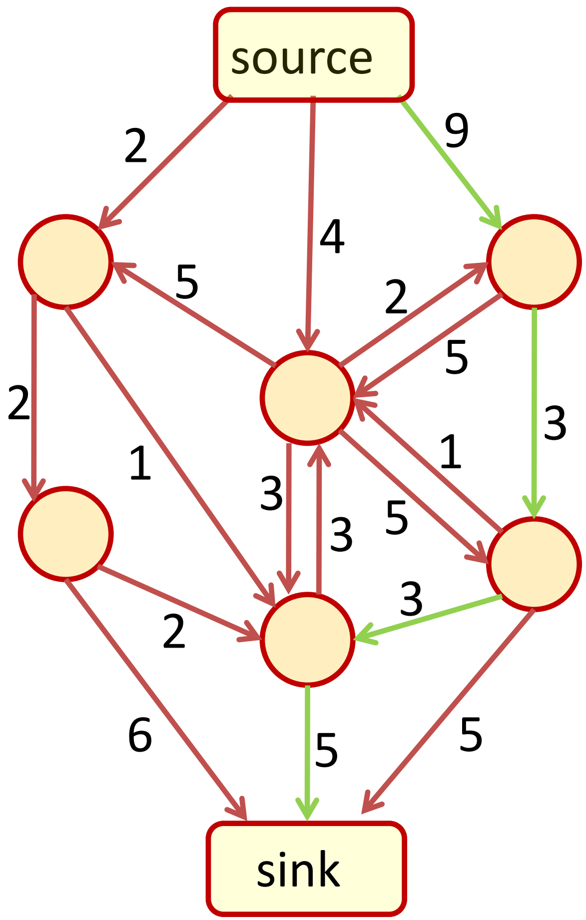
flow += maximum capacity in the path

Build the residual graph (“subtract” the flow)

Find the path in the residual graph

End

Max-Flow Problem



flow = 6

Ford & Fulkerson algorithm (1956)

Find the path from source to sink

While (path exists)

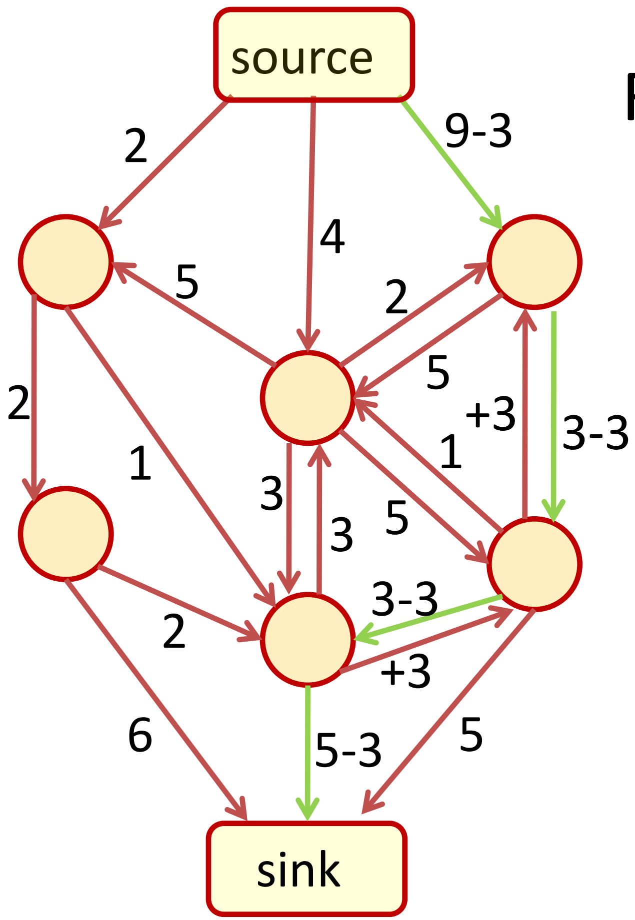
flow += maximum capacity in the path

Build the residual graph ("subtract" the flow)

Find the path in the residual graph

End

Max-Flow Problem



flow = 6

Ford & Fulkerson algorithm (1956)

Find the path from source to sink

While (path exists)

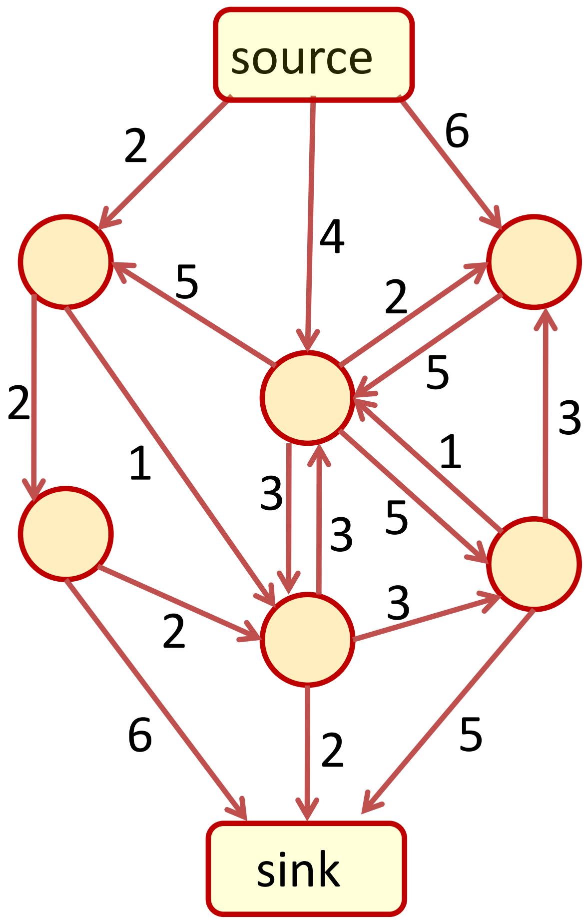
flow += maximum capacity in the path

Build the residual graph ("subtract" the flow)

Find the path in the residual graph

End

Max-Flow Problem



flow = 6

Ford & Fulkerson algorithm (1956)

Find the path from source to sink

While (path exists)

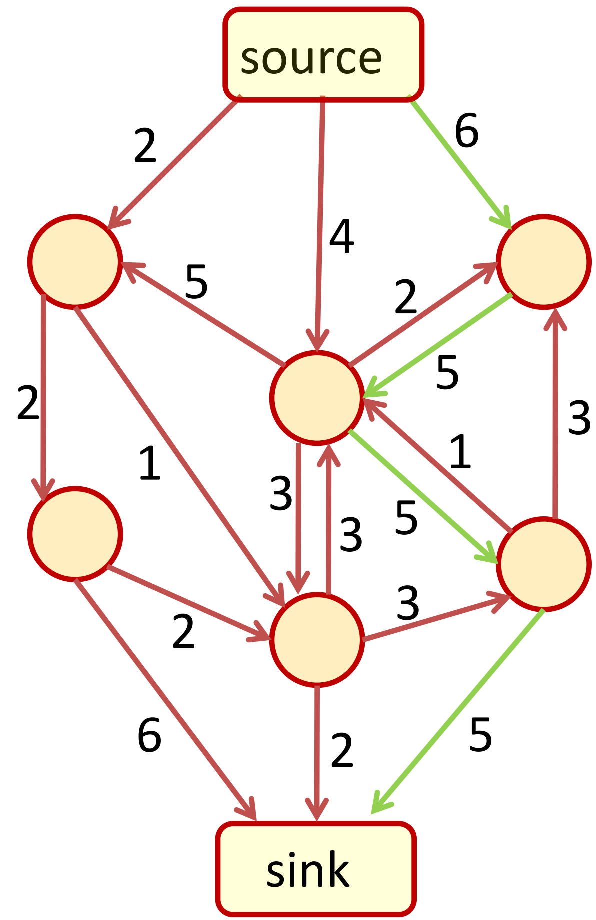
flow += maximum capacity in the path

Build the residual graph ("subtract" the flow)

Find the path in the residual graph

End

Max-Flow Problem



flow = 6

Ford & Fulkerson algorithm (1956)

Find the path from source to sink

While (path exists)

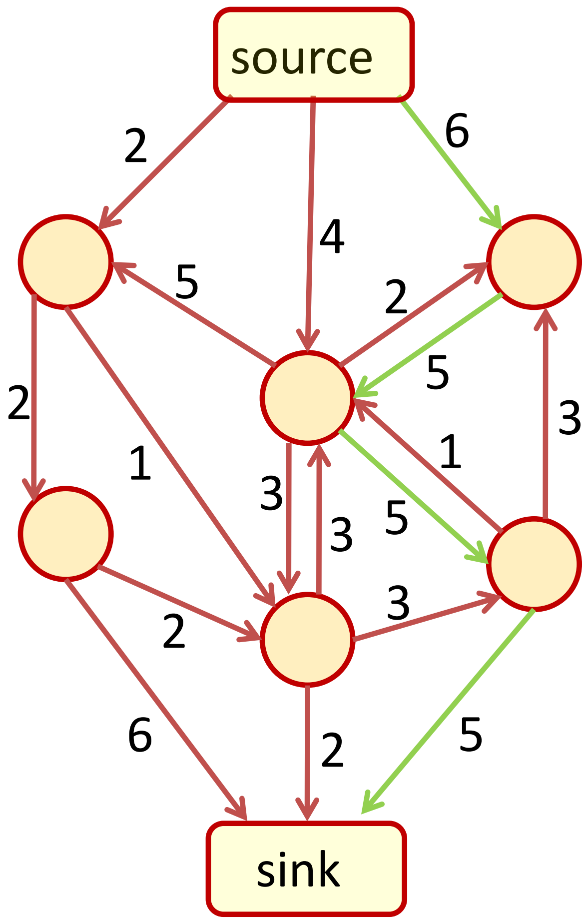
flow += maximum capacity in the path

Build the residual graph (“subtract” the flow)

Find the path in the residual graph

End

Max-Flow Problem



flow = 11

Ford & Fulkerson algorithm (1956)

Find the path from source to sink

While (path exists)

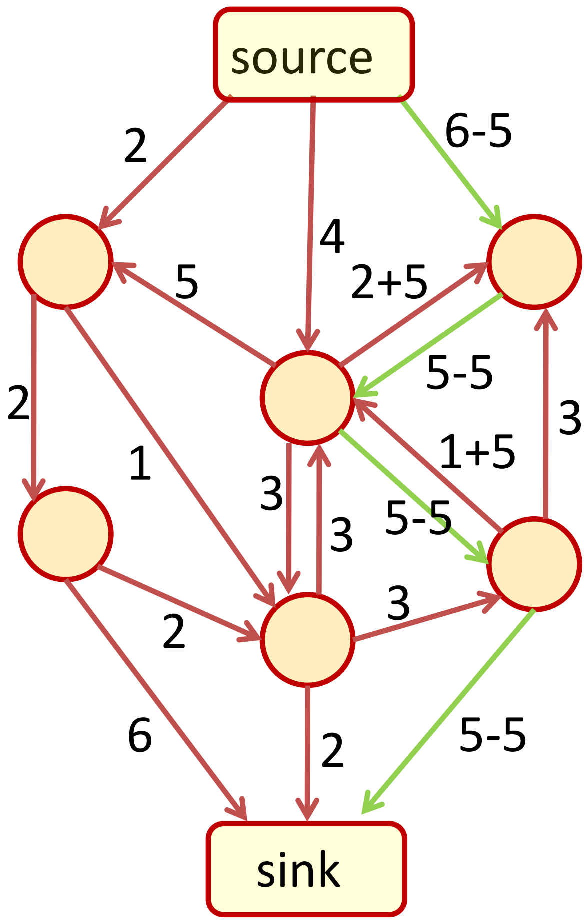
flow += maximum capacity in the path

Build the residual graph ("subtract" the flow)

Find the path in the residual graph

End

Max-Flow Problem



flow = 11

Ford & Fulkerson algorithm (1956)

Find the path from source to sink

While (path exists)

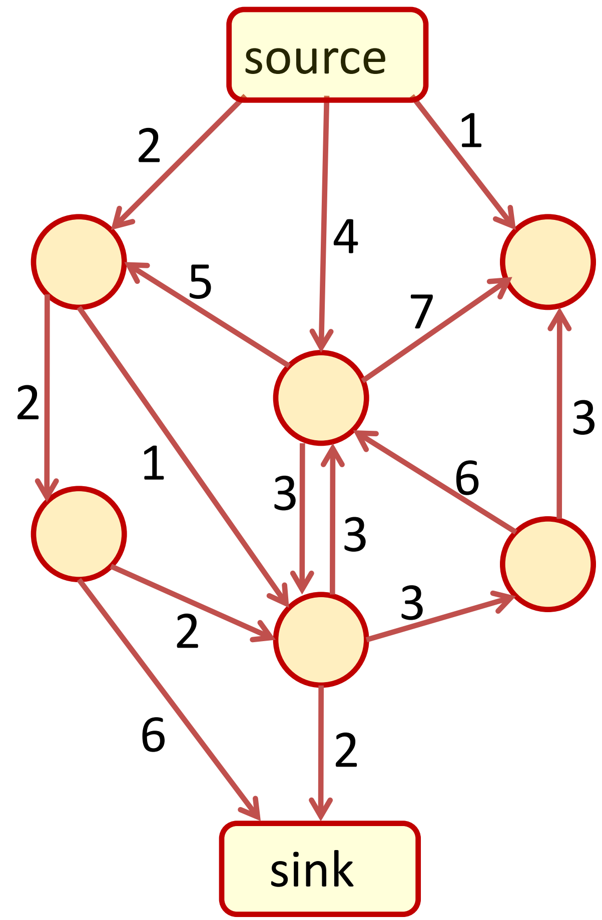
flow += maximum capacity in the path

Build the residual graph ("subtract" the flow)

Find the path in the residual graph

End

Max-Flow Problem



flow = 11

Ford & Fulkerson algorithm (1956)

Find the path from source to sink

While (path exists)

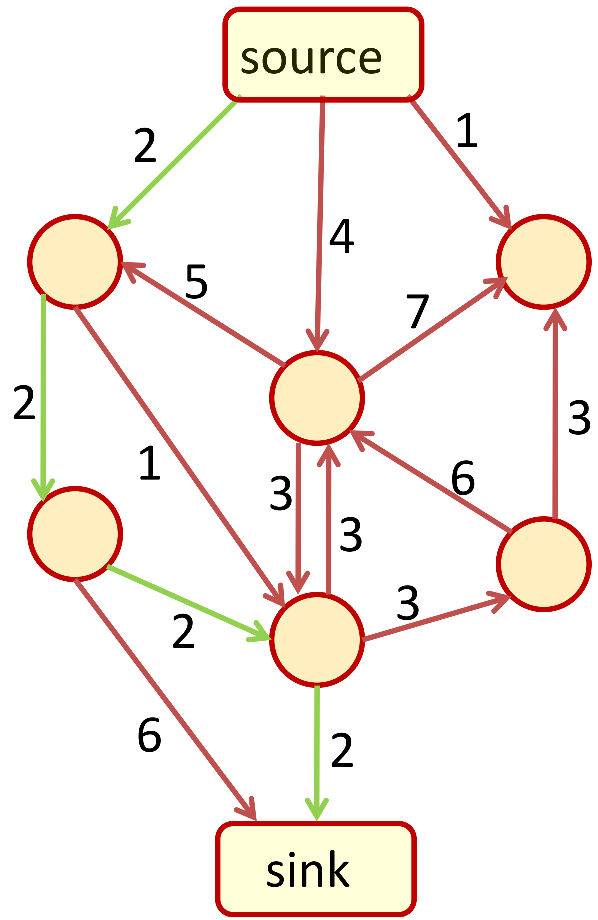
flow += maximum capacity in the path

Build the residual graph ("subtract" the flow)

Find the path in the residual graph

End

Max-Flow Problem



flow = 11

Ford & Fulkerson algorithm (1956)

Find the path from source to sink

While (path exists)

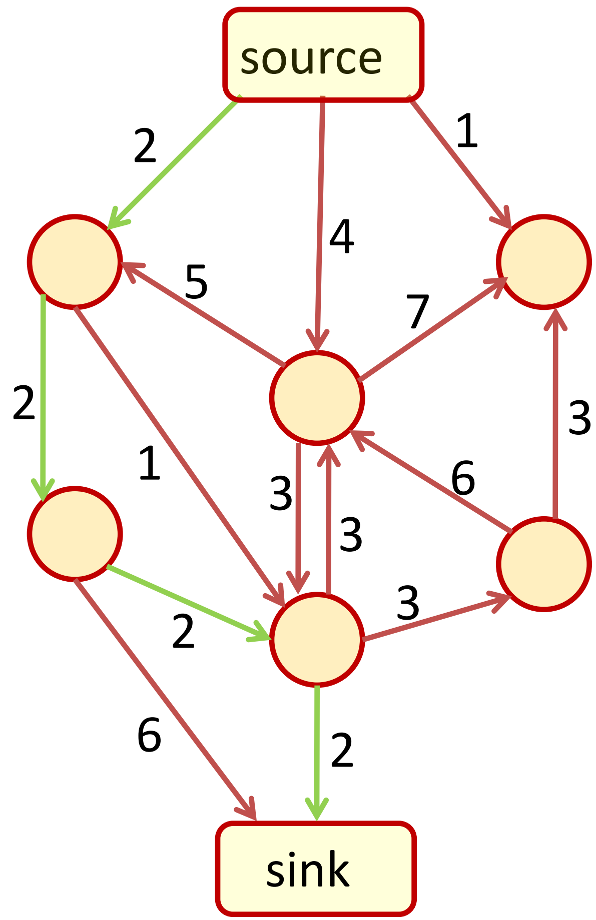
flow += maximum capacity in the path

Build the residual graph ("subtract" the flow)

Find the path in the residual graph

End

Max-Flow Problem



flow = 13

Ford & Fulkerson algorithm (1956)

Find the path from source to sink

While (path exists)

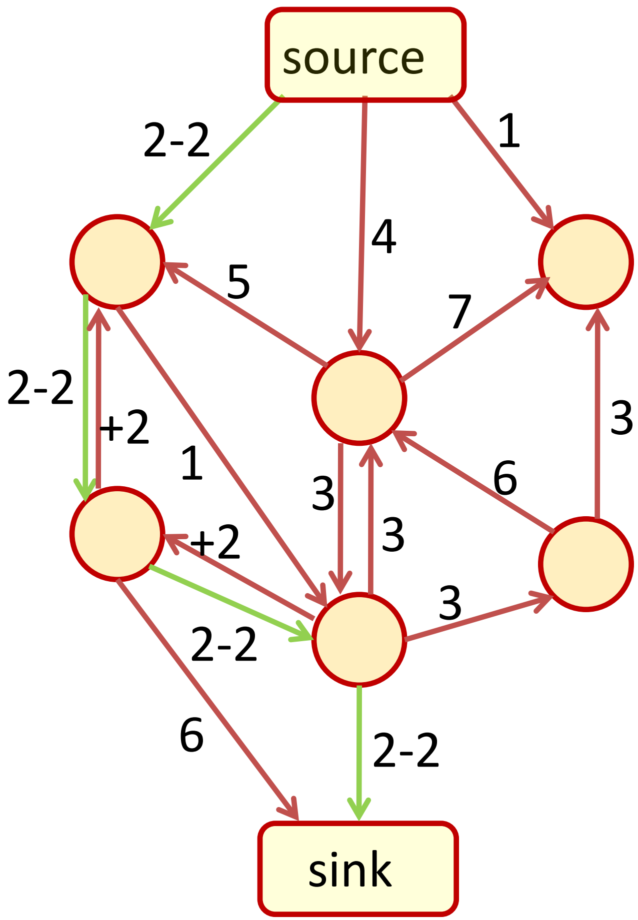
flow += maximum capacity in the path

Build the residual graph (“subtract” the flow)

Find the path in the residual graph

End

Max-Flow Problem



flow = 13

Ford & Fulkerson algorithm (1956)

Find the path from source to sink

While (path exists)

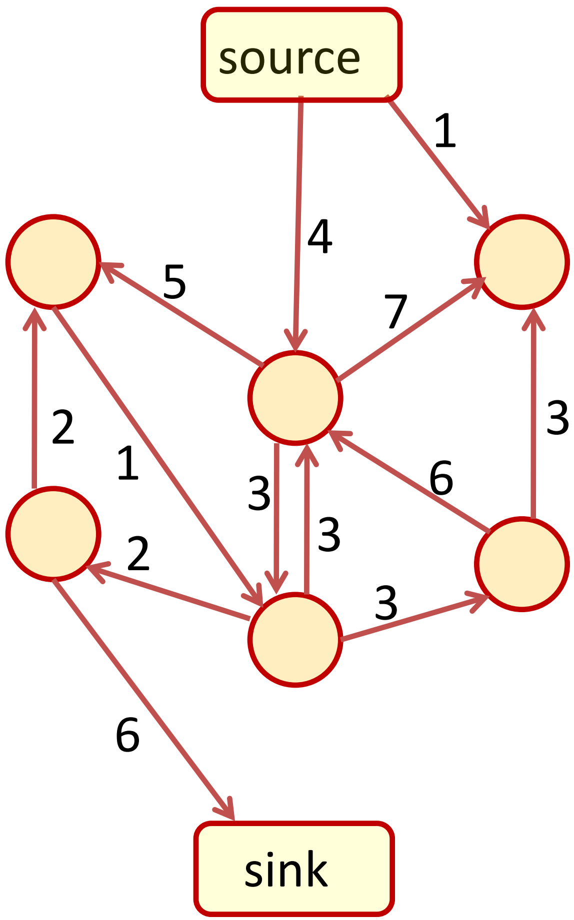
flow += maximum capacity in the path

Build the residual graph ("subtract" the flow)

Find the path in the residual graph

End

Max-Flow Problem



flow = 13

Ford & Fulkerson algorithm (1956)

Find the path from source to sink

While (path exists)

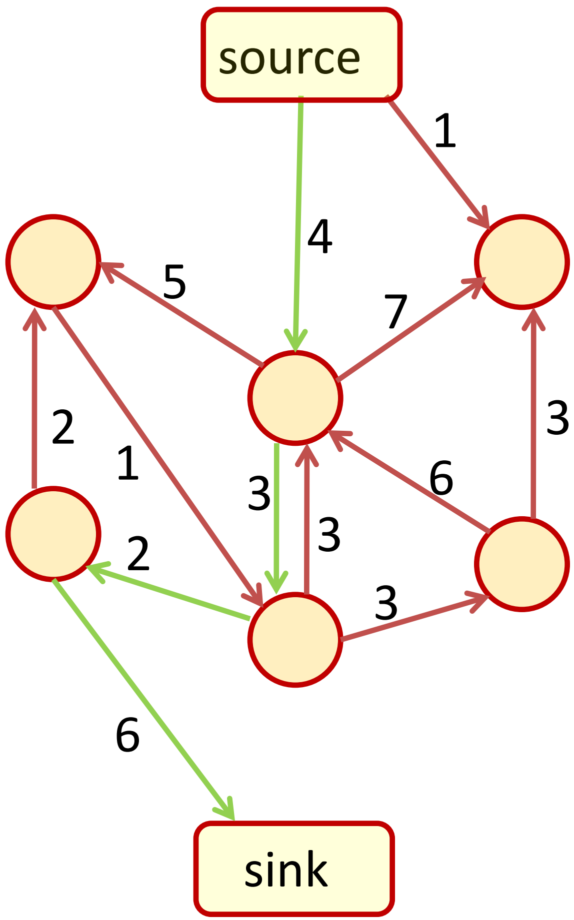
flow += maximum capacity in the path

Build the residual graph ("subtract" the flow)

Find the path in the residual graph

End

Max-Flow Problem



Ford & Fulkerson algorithm (1956)

Find the path from source to sink

While (path exists)

flow += maximum capacity in the path

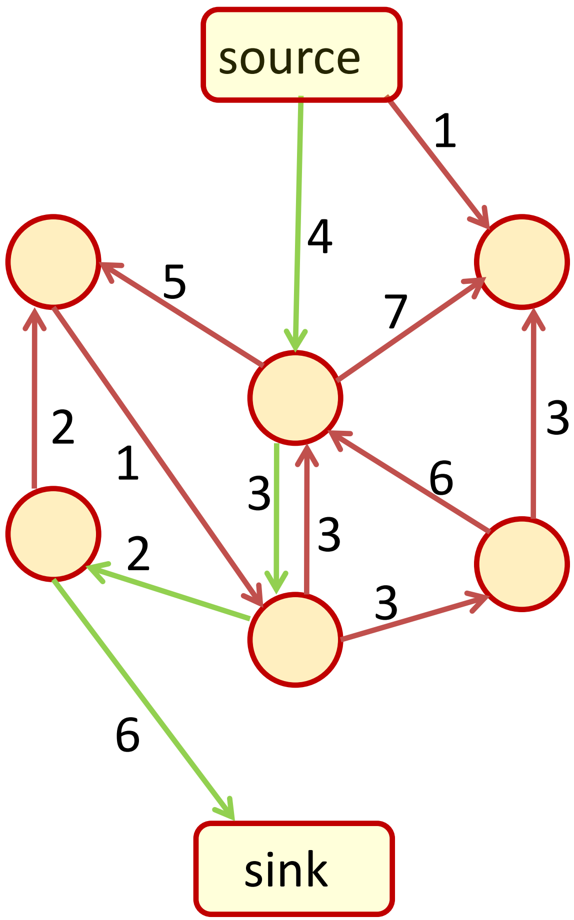
Build the residual graph (“subtract” the flow)

Find the path in the residual graph

End

flow = 13

Max-Flow Problem



flow = 15

Ford & Fulkerson algorithm (1956)

Find the path from source to sink

While (path exists)

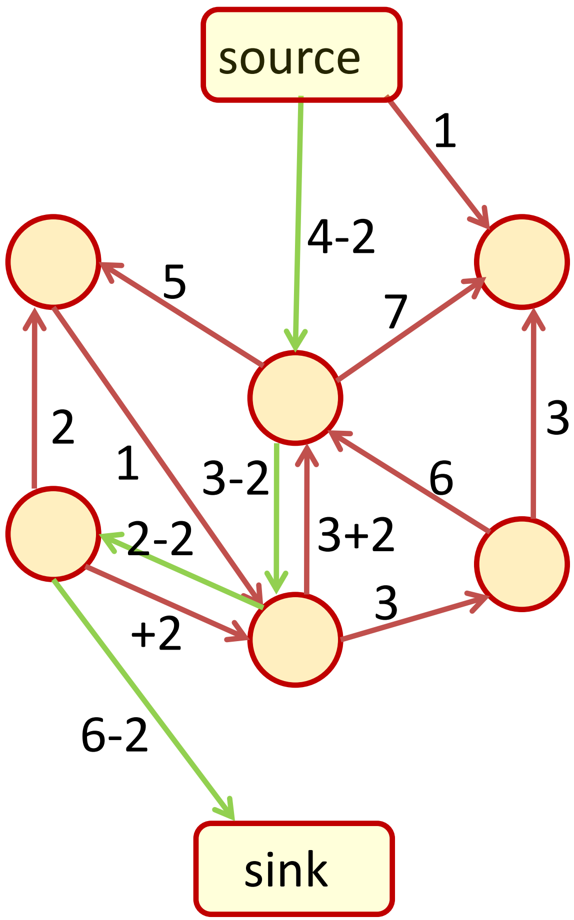
flow += maximum capacity in the path

Build the residual graph ("subtract" the flow)

Find the path in the residual graph

End

Max-Flow Problem



flow = 15

Ford & Fulkerson algorithm (1956)

Find the path from source to sink

While (path exists)

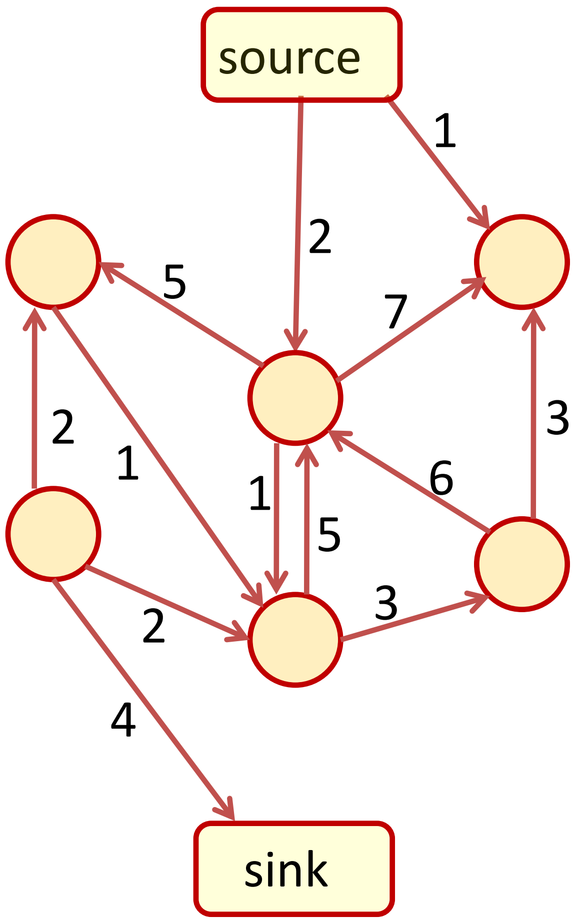
flow += maximum capacity in the path

Build the residual graph ("subtract" the flow)

Find the path in the residual graph

End

Max-Flow Problem



flow = 15

Ford & Fulkerson algorithm (1956)

Find the path from source to sink

While (path exists)

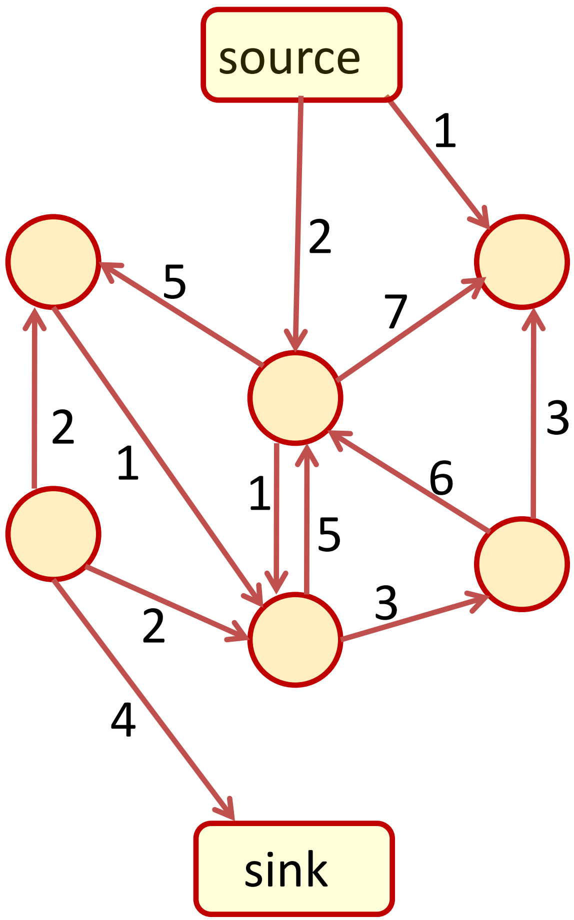
flow += maximum capacity in the path

Build the residual graph ("subtract" the flow)

Find the path in the residual graph

End

Max-Flow Problem



flow = 15

Ford & Fulkerson algorithm (1956)

Find the path from source to sink

While (path exists)

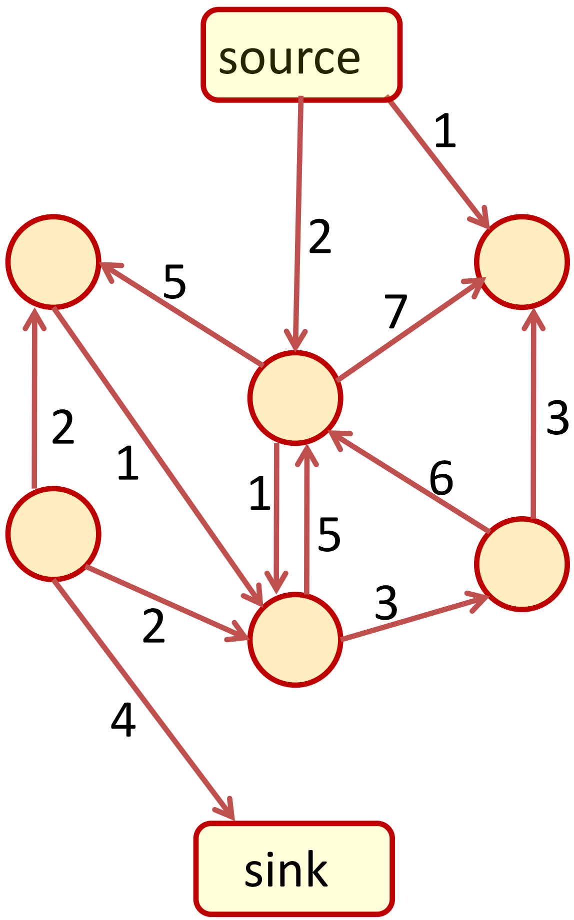
flow += maximum capacity in the path

Build the residual graph (“subtract” the flow)

Find the path in the residual graph

End

Max-Flow Problem



flow = 15

Ford & Fulkerson algorithm (1956)

Find the path from source to sink

While (path exists)

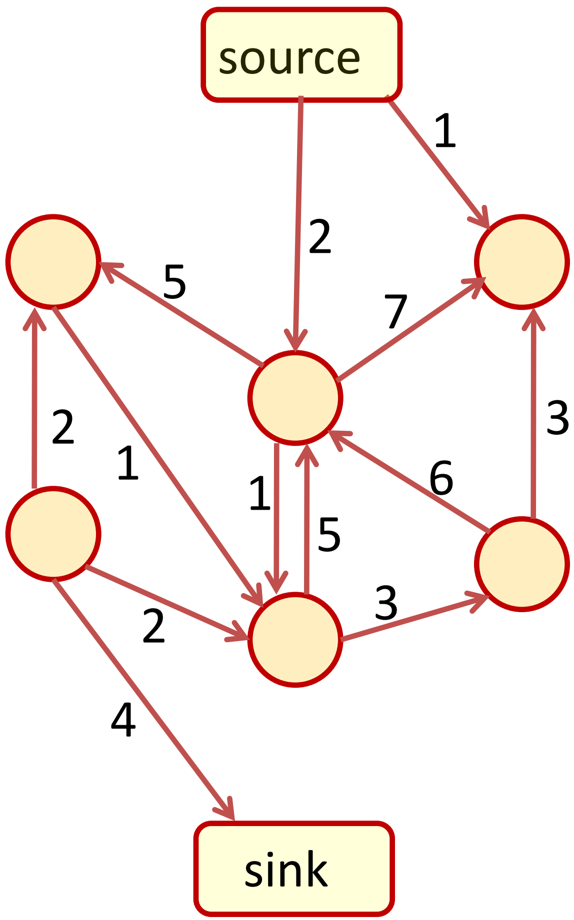
flow += maximum capacity in the path

Build the residual graph (“subtract” the flow)

Find the path in the residual graph

End

Max-Flow Problem

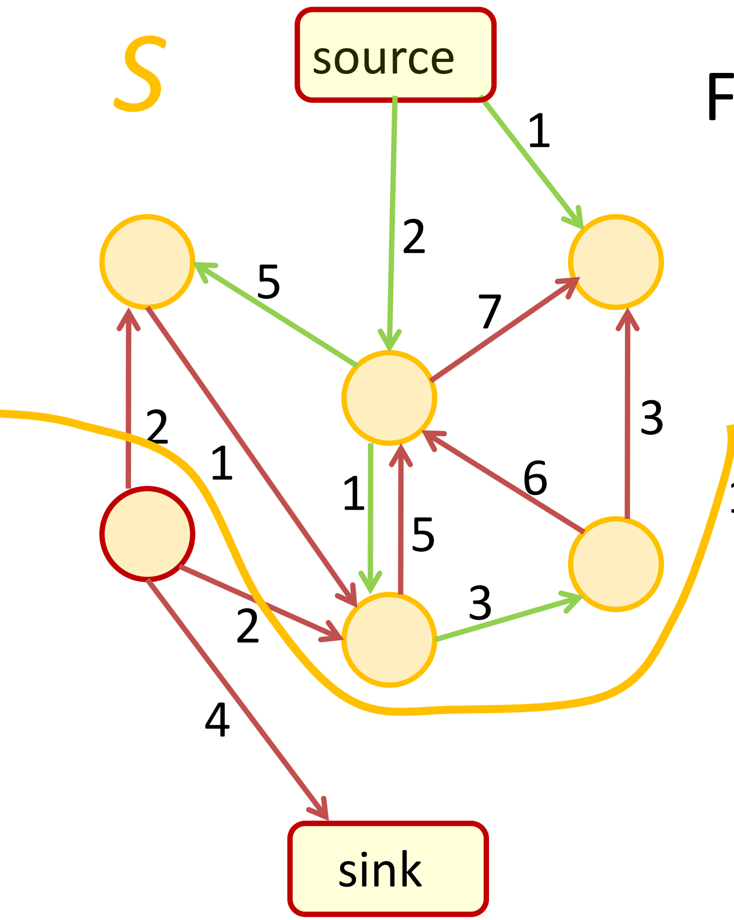


flow = 15

Ford & Fulkerson algorithm (1956)

Why is the solution globally optimal ?

Max-Flow Problem



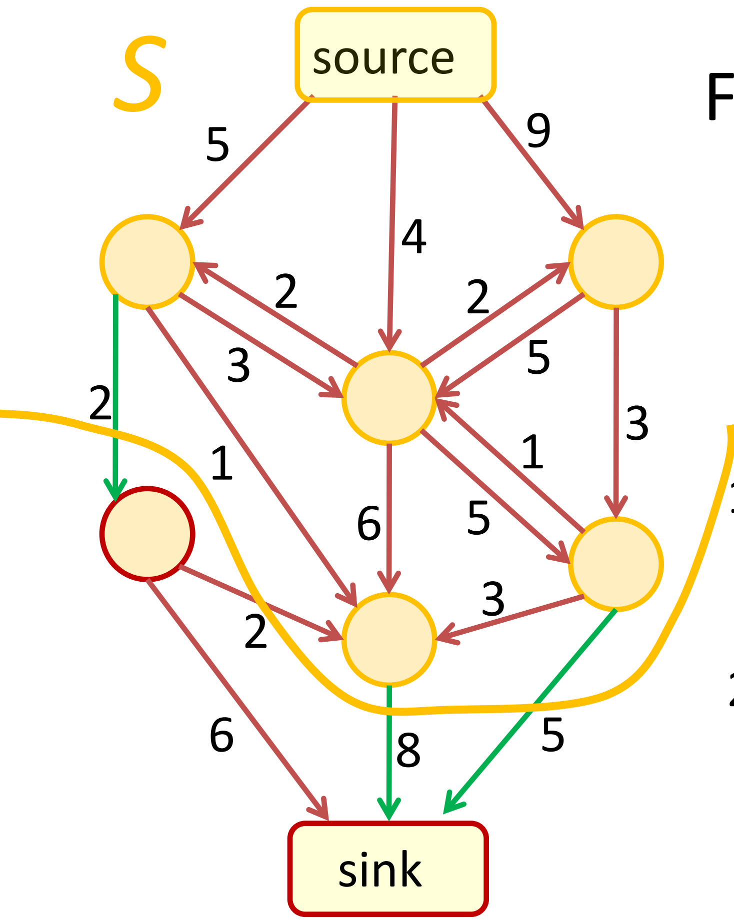
flow = 15

Ford & Fulkerson algorithm (1956)

Why is the solution globally optimal ?

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

Max-Flow Problem

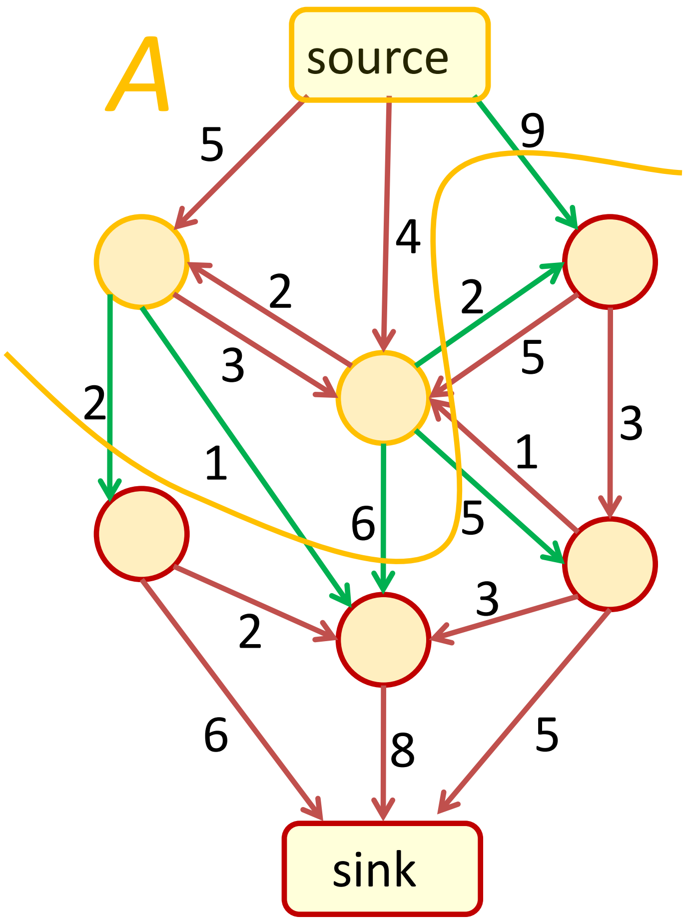


Ford & Fulkerson algorithm (1956)

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

Max-Flow Problem



Ford & Fulkerson algorithm (1956)

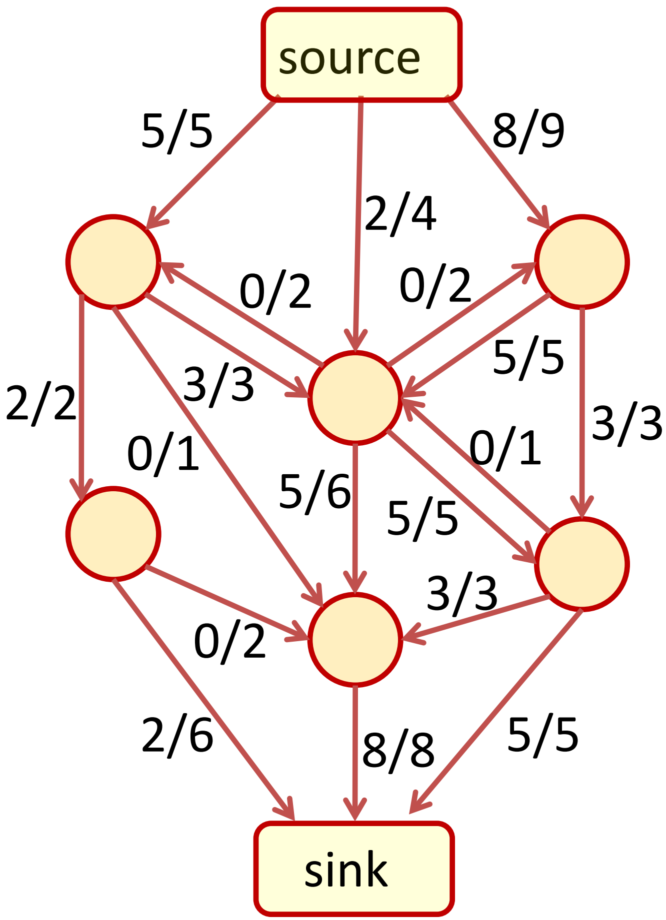
Why is the solution globally optimal ?

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

Max-Flow Problem



flow = 15

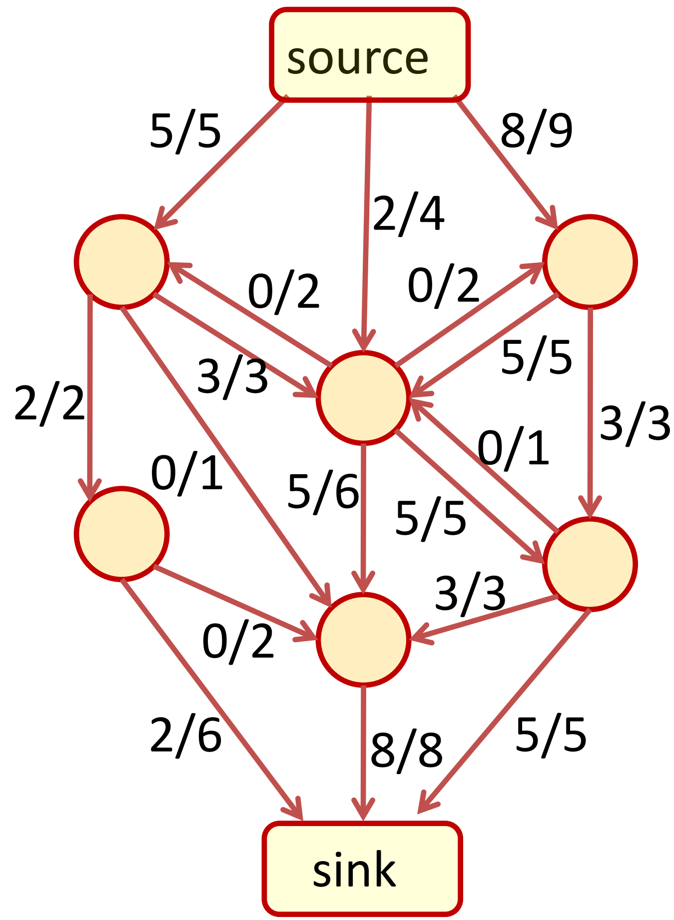
Individual flows obtained by summing up all paths

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

Max-Flow Problem



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
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

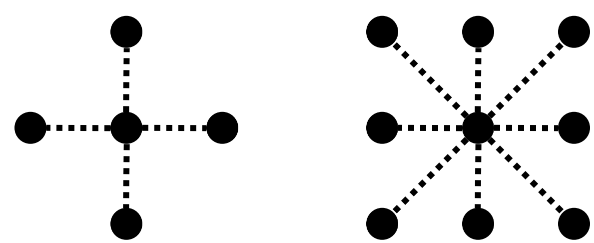
Stereo as energy minimization

$$E(d) = E_d(d) + \lambda E_s(d)$$

match cost: $$E_d(d) = \sum_{(x,y) \in I} C(x, y, d(x, y))$$

smoothness cost: $$E_s(d) = \sum_{(p,q) \in \mathcal{E}} V(d_p, d_q)$$

$\mathcal{E}$: set of neighboring pixels



4-connected neighborhood

8-connected neighborhood

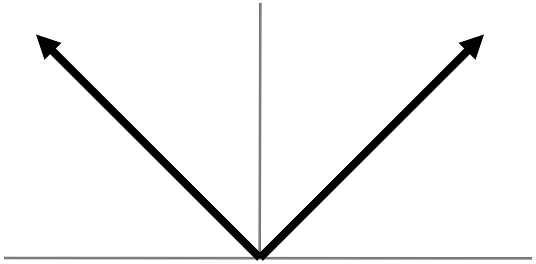
Smoothness cost

$$E_s(d) = \sum_{(p,q)\in\mathcal{E}} V(d_p, d_q)$$

last time: one possibility: quadratic and truncated quadratic models for *V*

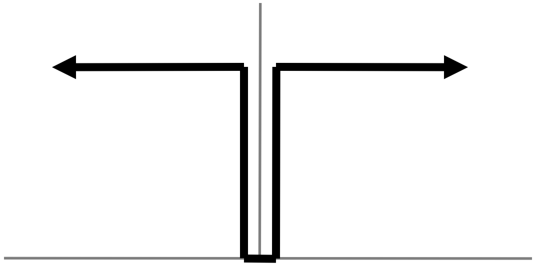
$$V(d_p, d_q) = |d_p - d_q|$$

L_1 distance



$$V(d_p, d_q) = \begin{cases} 0 & \text{if } d_p = d_q \\ 1 & \text{if } d_p \neq d_q \end{cases}$$

"Potts model"



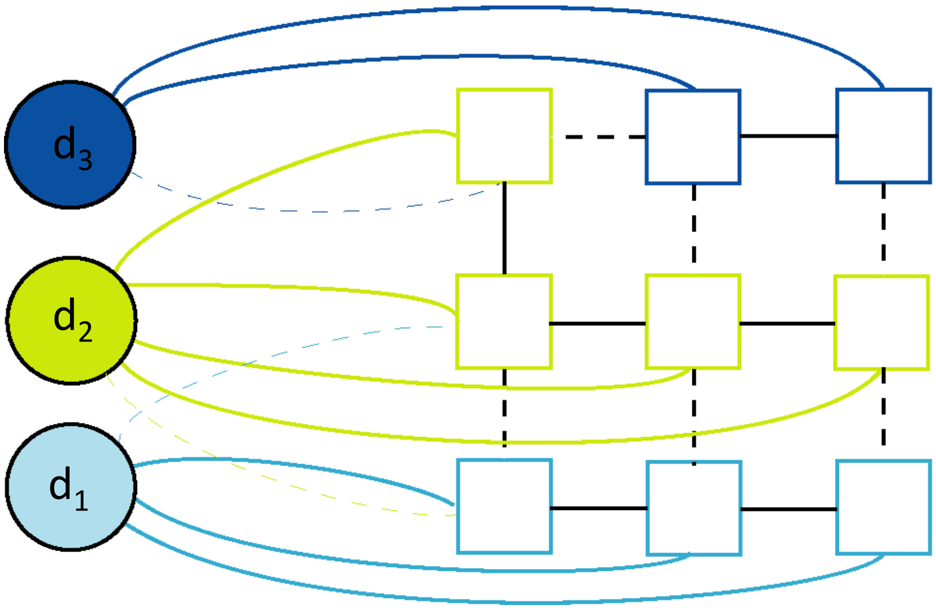
Stereo as a labeling problem

- Can formulate as a (no longer binary) labeling problem
  – with one label per disparity

- Can create similar setup as with segmentation, but with *k* source/sink nodes
  – *k* = number of disparities
  – Multi-source network flow problem
  – Using the Potts model, the setup is straightforward

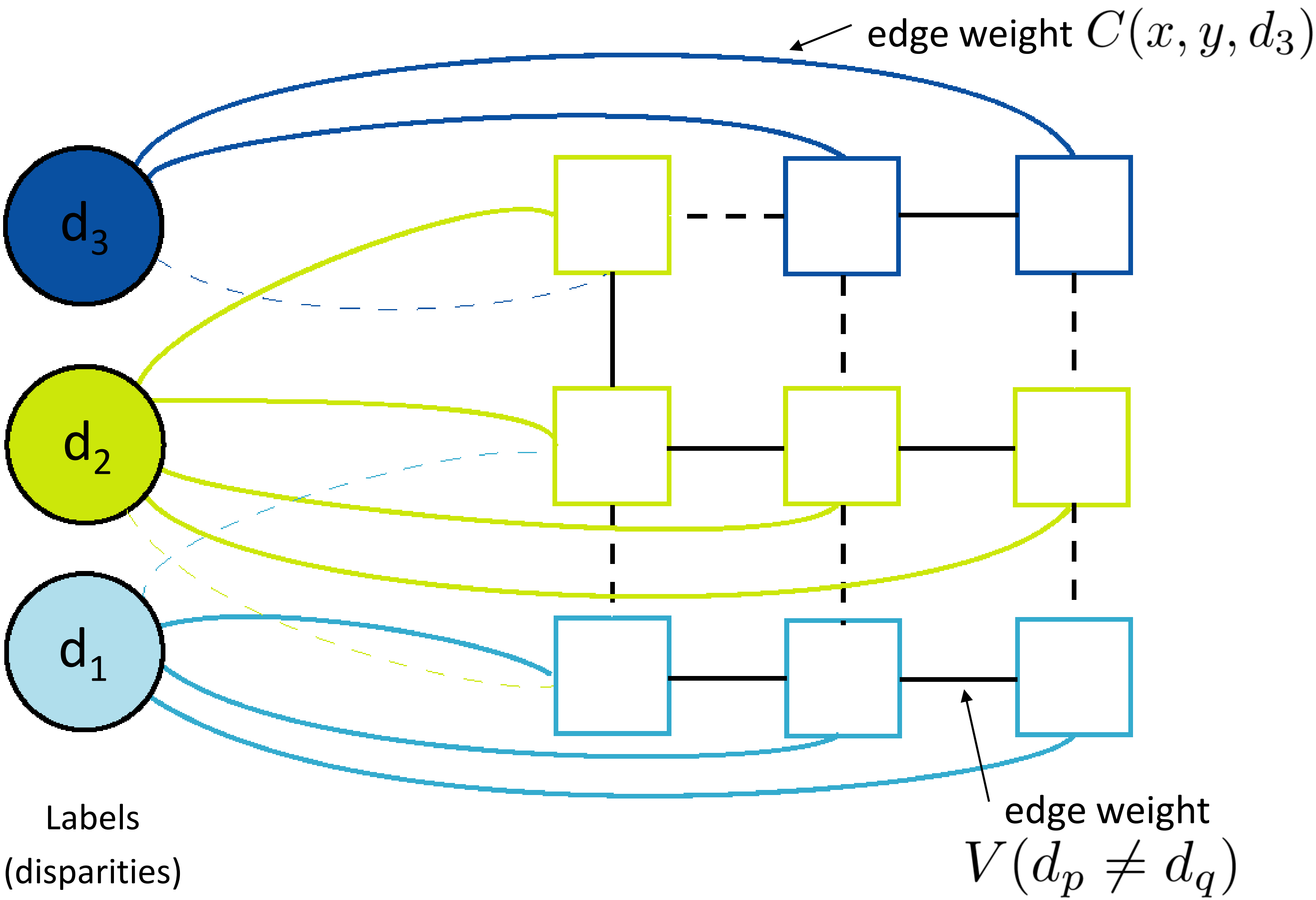
$$V(d_p, d_q) = \begin{cases} 0 & \text{if } d_p = d_q \\ 1 & \text{if } d_p \neq d_q \end{cases}$$

Energy minimization via graph cuts



- Graph Cut
  - Delete enough edges so that
    - each pixel is connected to exactly one label node
  - Cost of a cut: sum of deleted edge weights
  - Finding min cost cut equivalent to finding global minimum of energy function

Energy minimization via graph cuts



Computing a multiway cut

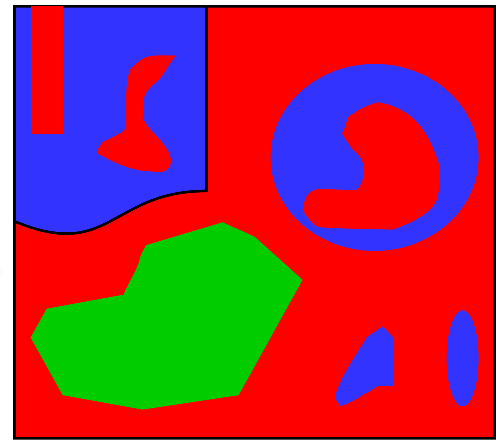
- With 2 labels: classical min-cut problem
  - Solvable by standard flow algorithms
    - polynomial time in theory, nearly linear in practice
  - More than 2 terminals: NP-hard
    [Dahlhaus *et al.*, STOC '92]

- Efficient approximation algorithms exist
  - Boykov, Veksler and Zabih, Fast Approximate Energy Minimization via Graph Cuts, ICCV 1999.
  - Within a factor of 2 of optimal
  - Computes local minimum in a strong sense
    - even very large moves will not improve the energy

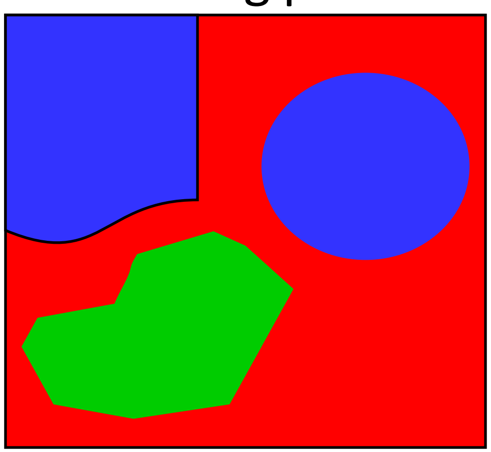
Move examples

Idea: convert multi-way cut into a sequence of binary cut problems

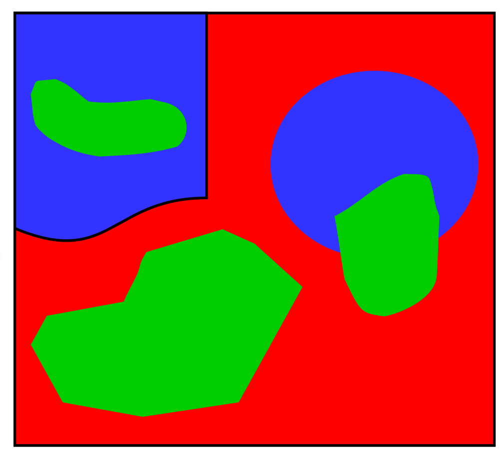
Red-blue swap move



Starting point

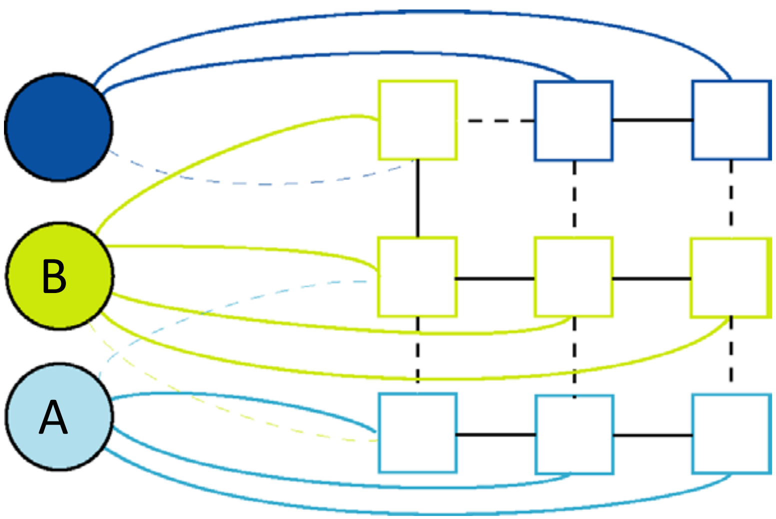


Green expansion move

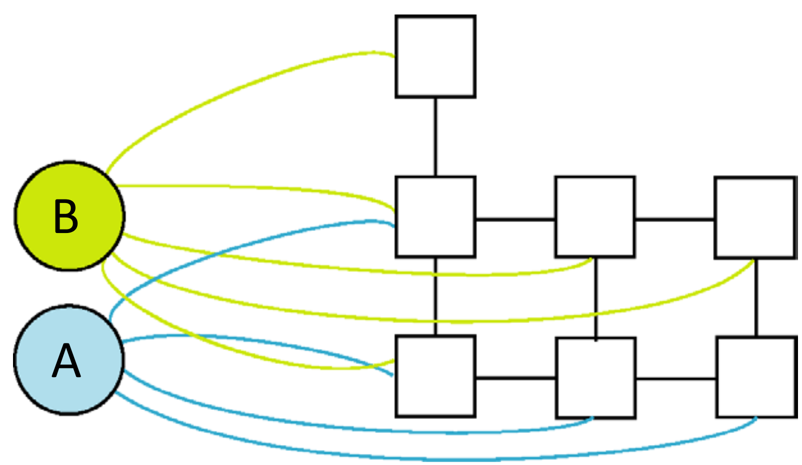


The swap move algorithm

1. Start with an arbitrary labeling
2. Cycle through every label pair *(A,B)* in some order
    2.1 Find the lowest *E* labeling within a single *AB*-swap
    2.2 Go there if it's lower *E* than the current labeling
3. If *E* did not decrease in the cycle, we're done
   Otherwise, go to step 2

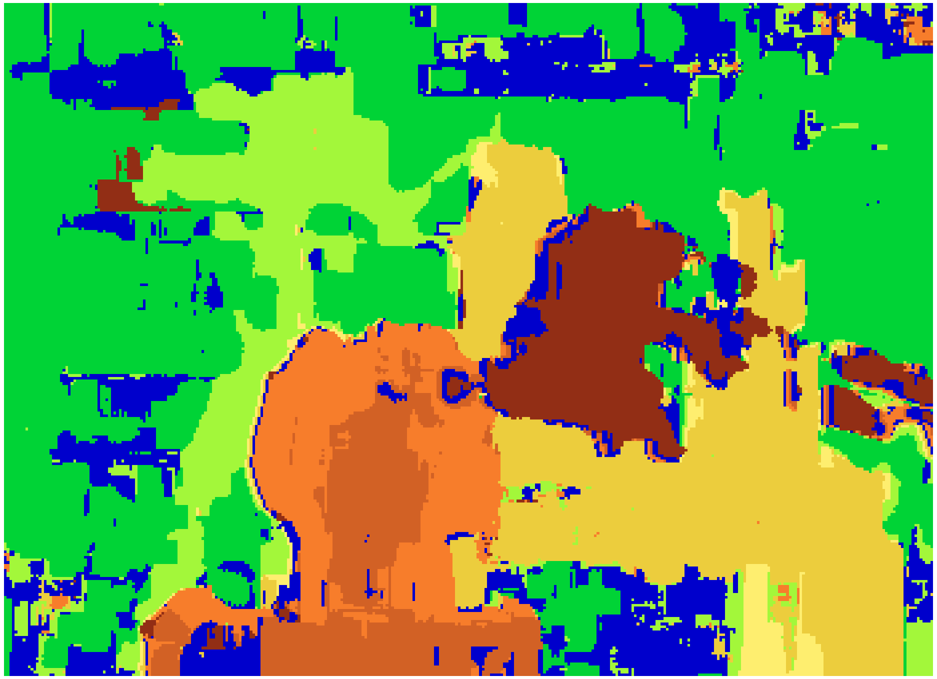


Original graph



AB subgraph
(run min-cut on this graph)

Results with window correlation



normalized correlation
(best window size)

ground truth

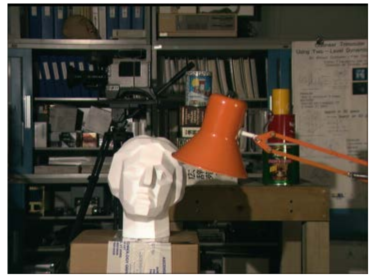
Results with graph cuts

graph cuts
(Potts model,
expansion move algorithm)

ground truth

Dense Stereo Reconstruction

Left Camera Image



Right Camera Image

Dense Stereo Result

Data term

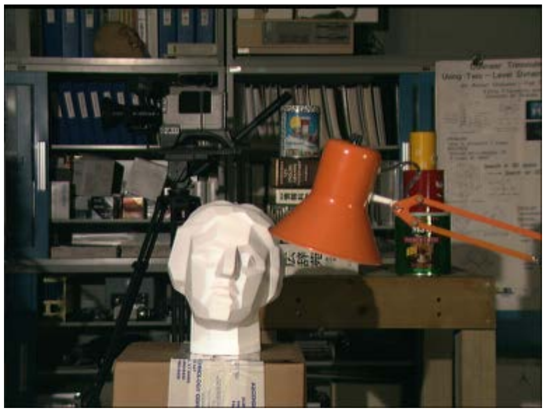
Same as before

Smoothness term

$$\psi_{ij}(z_i, z_j) = \min(\boxed{K|z_i - z_j|}, \boxed{T})$$

Convex range **Truncation**

Graph Cut based Inference



Original Image

Initial Solution

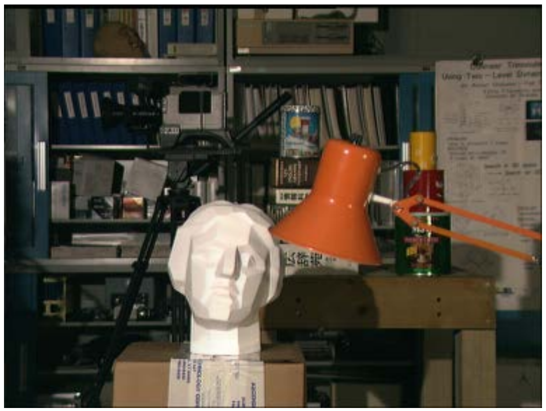
Graph Cut based Inference

Original Image

Initial Solution

After 1st expansion

Graph Cut based Inference



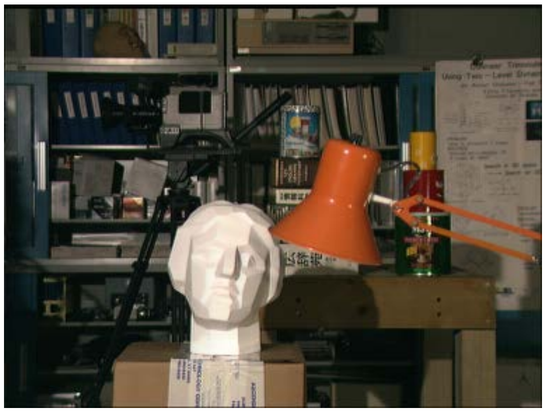
Original Image

Initial Solution

After 1st expansion

After 2nd expansion

Graph Cut based Inference



Original Image

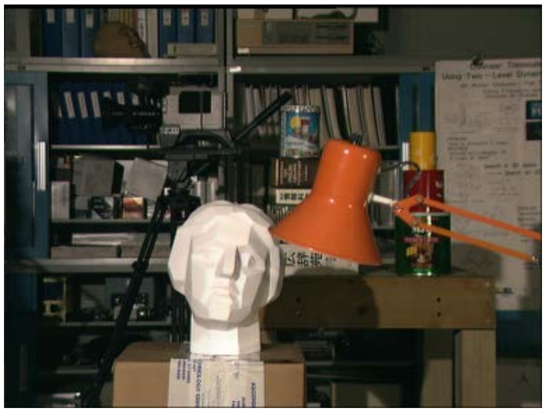
Initial Solution

After 1st expansion

After 2nd expansion

After 3rd expansion

Graph Cut based Inference



Original Image

Initial Solution

After 1st expansion

After 2nd expansion

After 3rd expansion

Final solution

Other energy functions

- Can optimize other functions (exactly or approximately) with graph cuts

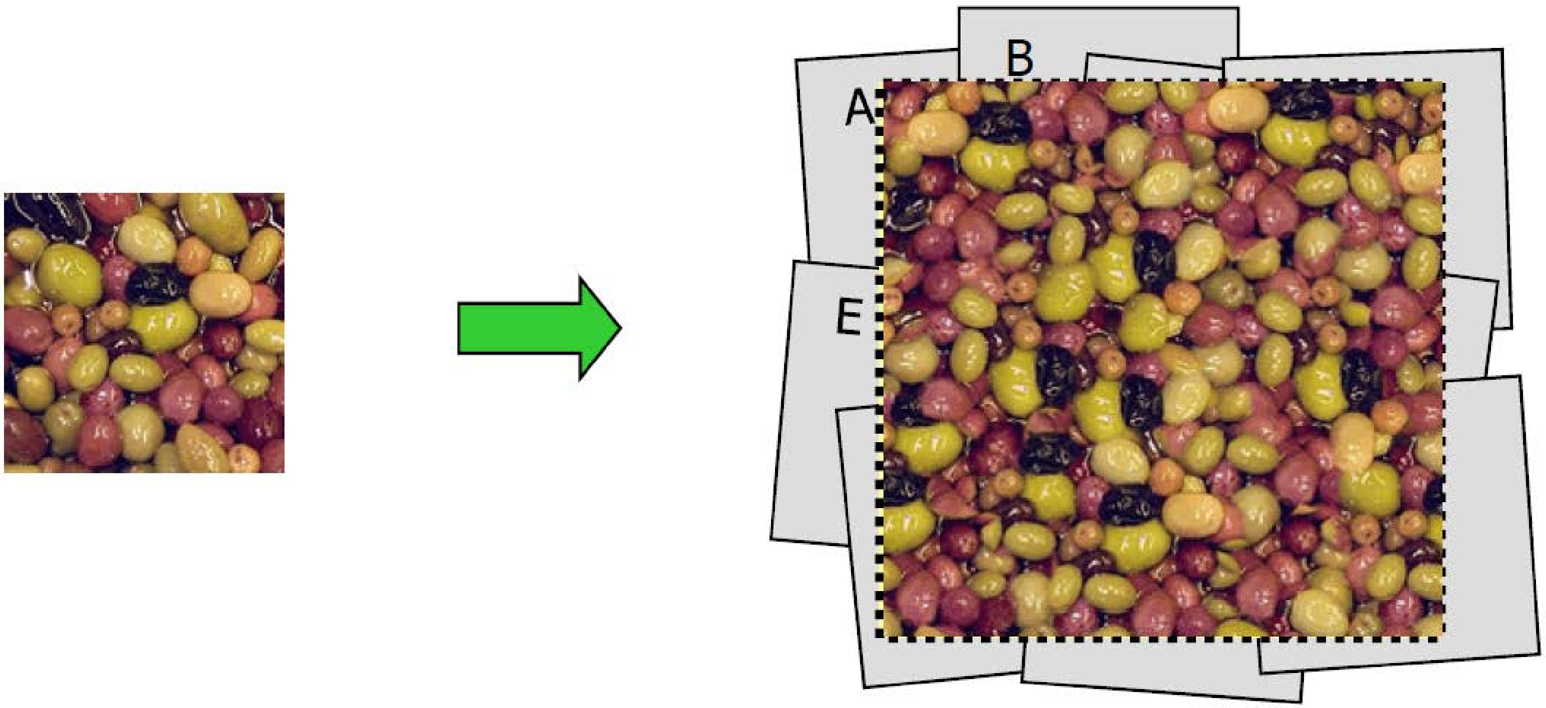
$$V(d_p, d_q) = (d_p - d_q)^2$$

$$V(d_p, d_q) = |d_p - d_q|$$

Questions?

Graph cuts in vision and graphics

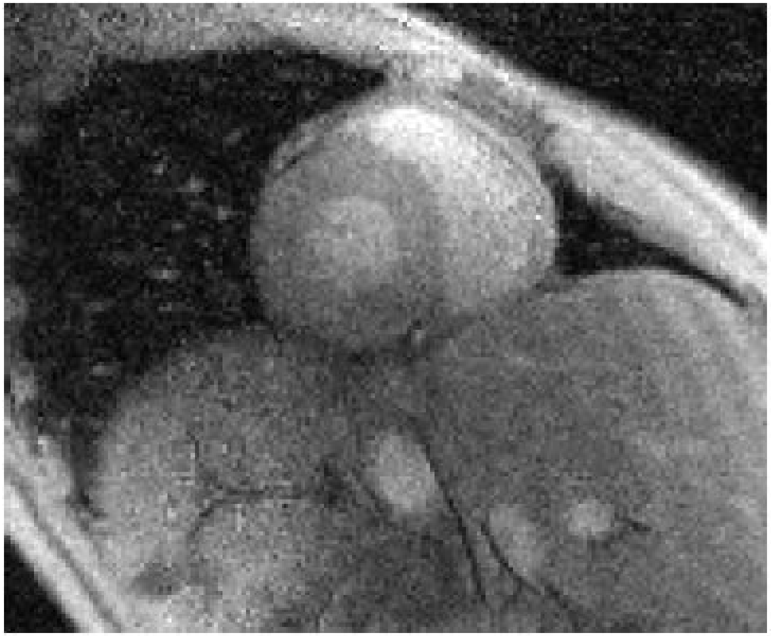
- Texture synthesis



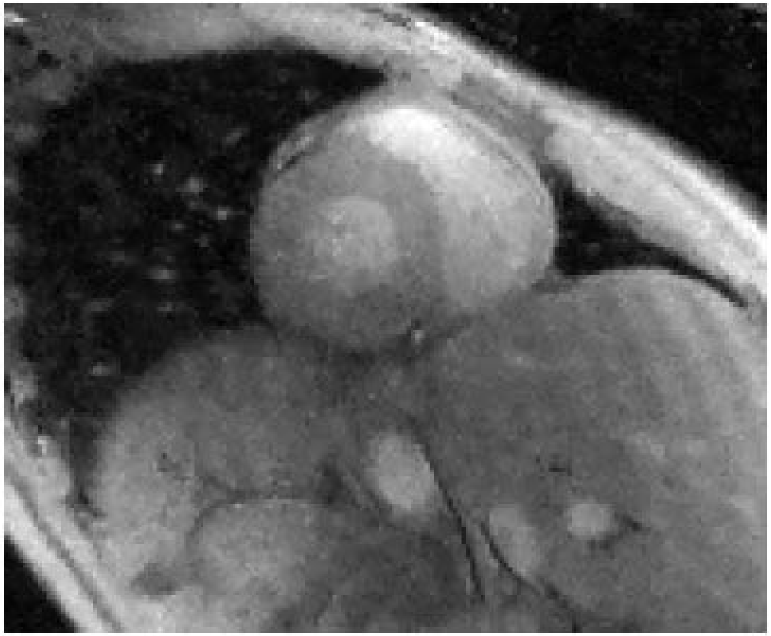
- “Graphcut textures”, [Kwatra, *et al.*, 2003]

Interactive Digital Photo Montage

MRI results [Raj et al. MRM 07]



SENSE
(= LS)



Graph cuts

Other uses of graph cuts

M. Uyttendaele, A. Eden, and R. Szeliski.
***Eliminating ghosting and exposure artifacts in image mosaics*.**
In Proceedings of the Interational Conference on Computer Vision and Pattern Recognition, volume 2, pages 509--516, Kauai, Hawaii, December 2001.

Other uses of graph cuts

M. Uyttendaele, A. Eden, and R. Szeliski.
***Eliminating ghosting and exposure artifacts in image mosaics*.**
In Proceedings of the Interational Conference on Computer Vision and Pattern Recognition, volume 2, pages 509--516, Kauai, Hawaii, December 2001.