CS4670 / 5670: Computer Vision

Noah Snavely

Graph-Based Image Segmentation

Stereo as a minimization problem

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

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

Adjacent pixels should (usually) move about the same amount

Related problem: binary segmentation

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

- Can you think of a way to solve this problem?

Related problem: binary segmentation

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

User sketches out a few strokes on foreground and background…

How do we classify the rest of the pixels?

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?")

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

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

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

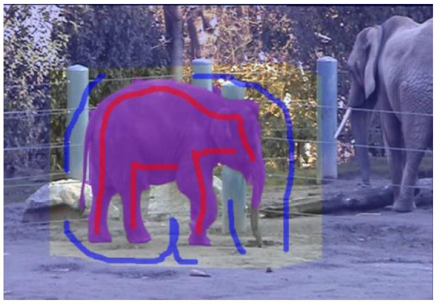
$$C(x, y, L(x, y)) = \begin{cases} \infty & \text{if } L(x, y) \neq \tilde{L}(x, y) \\ C'(x, y, L(x, y)) & \text{otherwise} \end{cases}$$

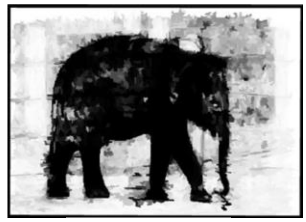
$C'(x, y, 0)$: "distance" from pixel to background pixels

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

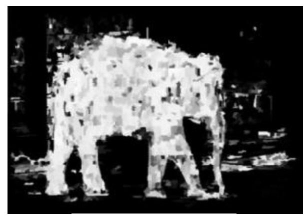
} usually computed by creating a color model from user-labeled pixels

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





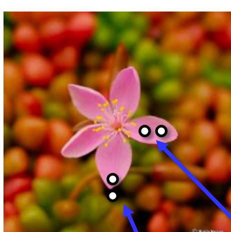
$C'(x, y, 0)$



$C'(x, y, 1)$

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

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



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

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

w_{pq} = 10.0

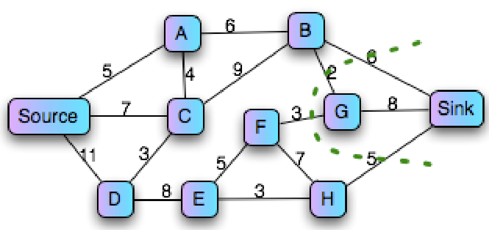
w_{pq} = 0.1

Binary segmentation as energy minimization

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

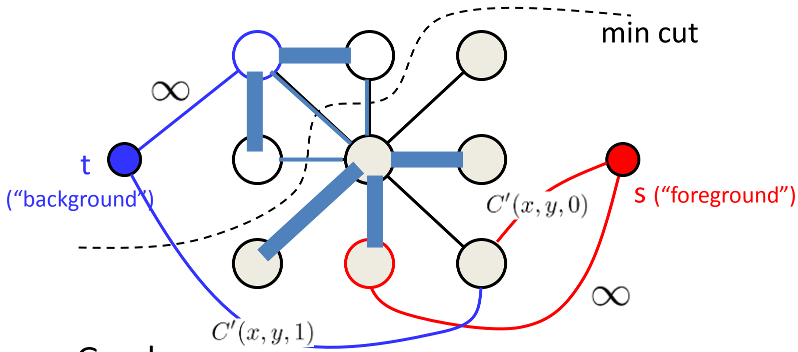
- For this problem, we can easily find the global minimum!
- Use max flow / min cut algorithm

Graph min cut problem



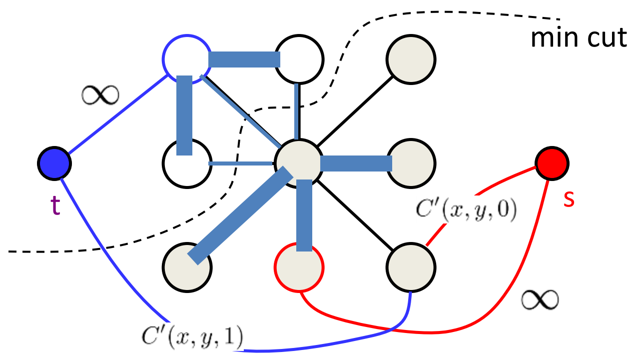
- Given a weighted graph G with source and sink nodes (s and t), partition the nodes into two sets, S and T such that the sum of edge weights spanning the partition is minimized
  - and $s \in S$ and $t \in T$

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

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

GrabCut

Grabcut [Rother et al., SIGGRAPH 2004]

Is user-input required?

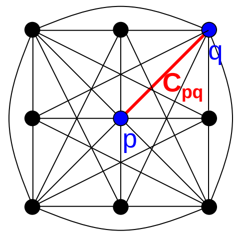
Our visual system is proof that automatic methods are possible

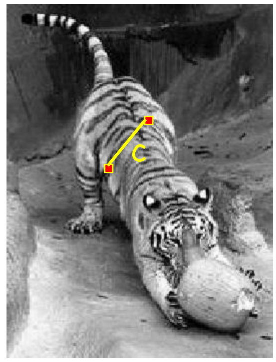
- classical image segmentation methods are automatic

Argument for user-directed methods?

- only user knows desired scale/object of interest

Automatic graph cut [Shi & Malik]

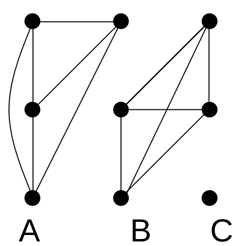




Fully-connected graph

- node for every pixel
- link between *every* pair of pixels, **p**,**q**
- cost $\mathbf{c_{pq}}$ for each link
  - $\mathbf{c_{pq}}$ measures *similarity*
    - » similarity is *inversely proportional* to difference in color and position

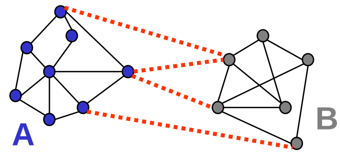
Segmentation by Graph Cuts



Break Graph into Segments

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

Cuts in a graph



Link Cut

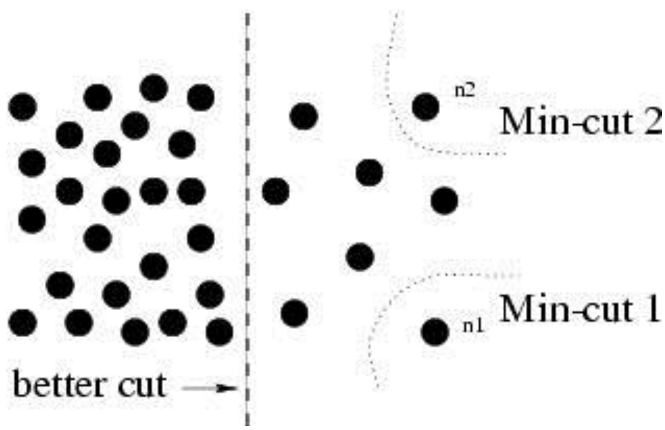
- set of links whose removal makes a graph disconnected
- cost of a cut:

$$cut(A,B) = \sum_{p \in A, q \in B} c_{p,q}$$

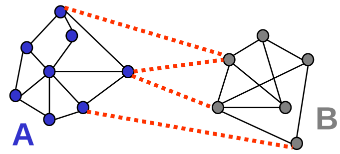
Find minimum cut

- gives you a segmentation

But min cut is not always the best cut...



Cuts in a graph



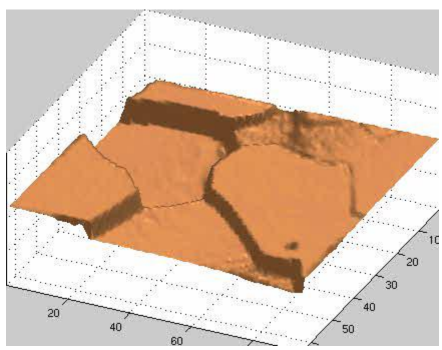
Normalized Cut

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

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

- volume(A) = sum of costs of all edges that touch A

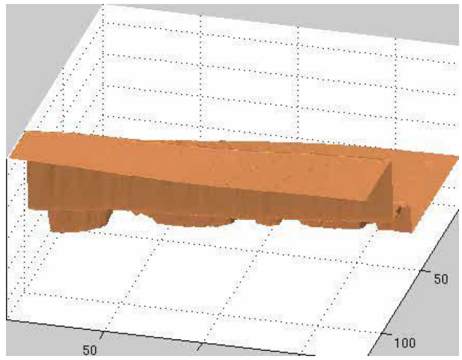
Interpretation as a Dynamical System



Treat the links as springs and shake the system

- elasticity proportional to cost
- vibration “modes” correspond to segments
  - can compute these by solving an eigenvector problem
  - http://www.cis.upenn.edu/~jshi/papers/pami_ncut.pdf

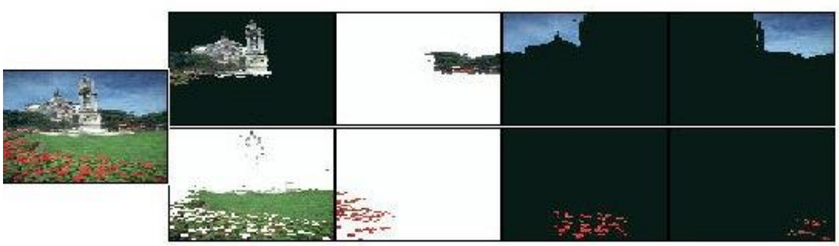
Interpretation as a Dynamical System



Treat the links as springs and shake the system

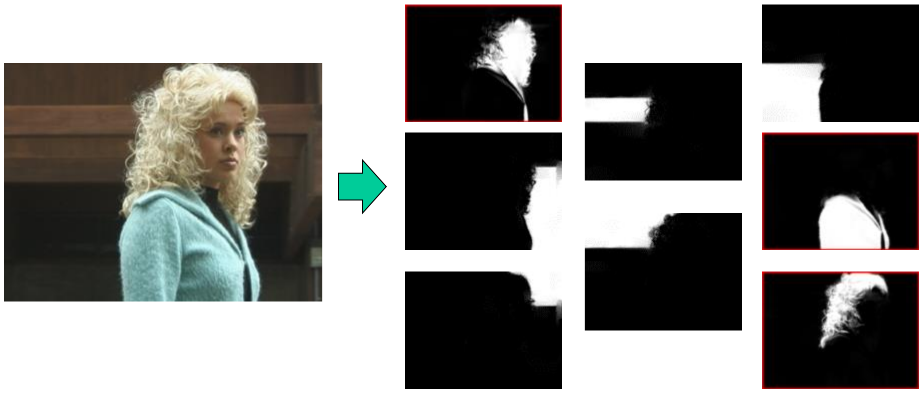
- elasticity proportional to cost
- vibration "modes" correspond to segments
  - can compute these by solving an eigenvector problem
  - http://www.cis.upenn.edu/~jshi/papers/pami_ncut.pdf

Color Image Segmentation



Extension to Soft Segmentation

- Each pixel is convex combination of segments. [Levin et al. 2006]
  - compute mattes by solving eigenvector problem



Questions?