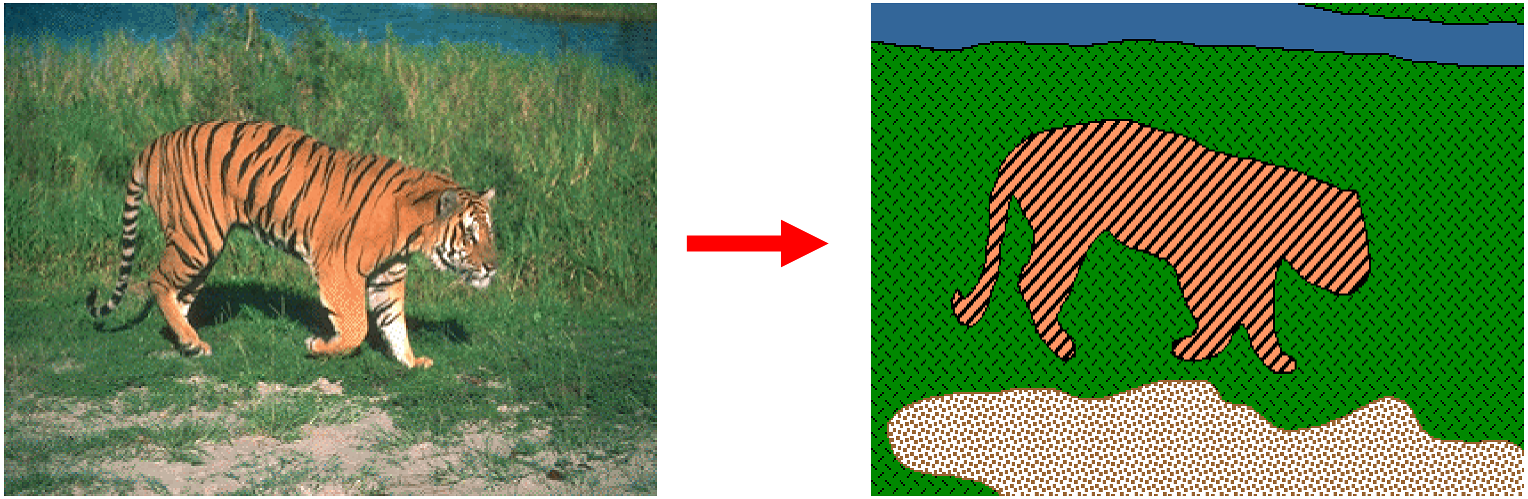
Segmentation and Clustering Methods

From [Sandlot Science]

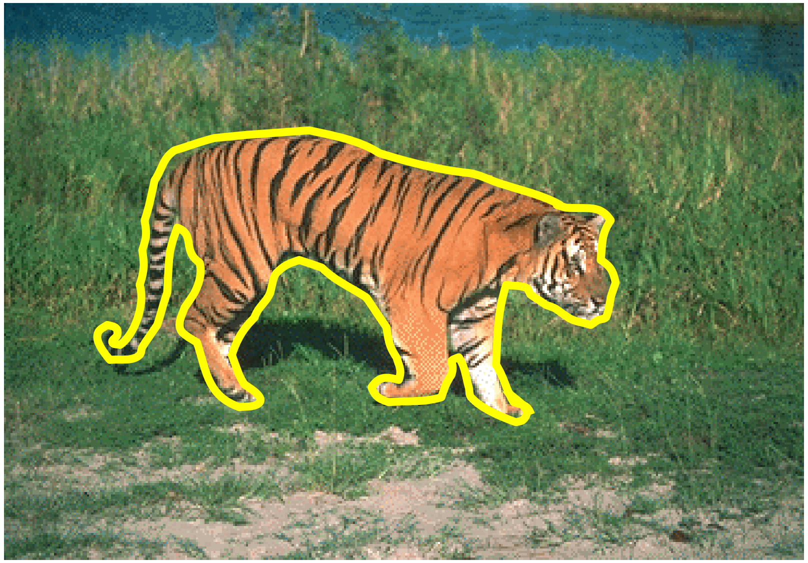
From images to objects



What Defines an Object?

- Subjective problem, but has been well-studied
- Gestalt Laws seek to formalize this
  - proximity, similarity, continuation, closure, common fate
  - see [notes] by Steve Joordens, U. Toronto

Extracting objects



How could this be done?

Image Segmentation

Many approaches proposed

- cues: color, regions, contours
- automatic vs. user-guided
- no clear winner
- we’ll consider several approaches today

Intelligent Scissors (demo)

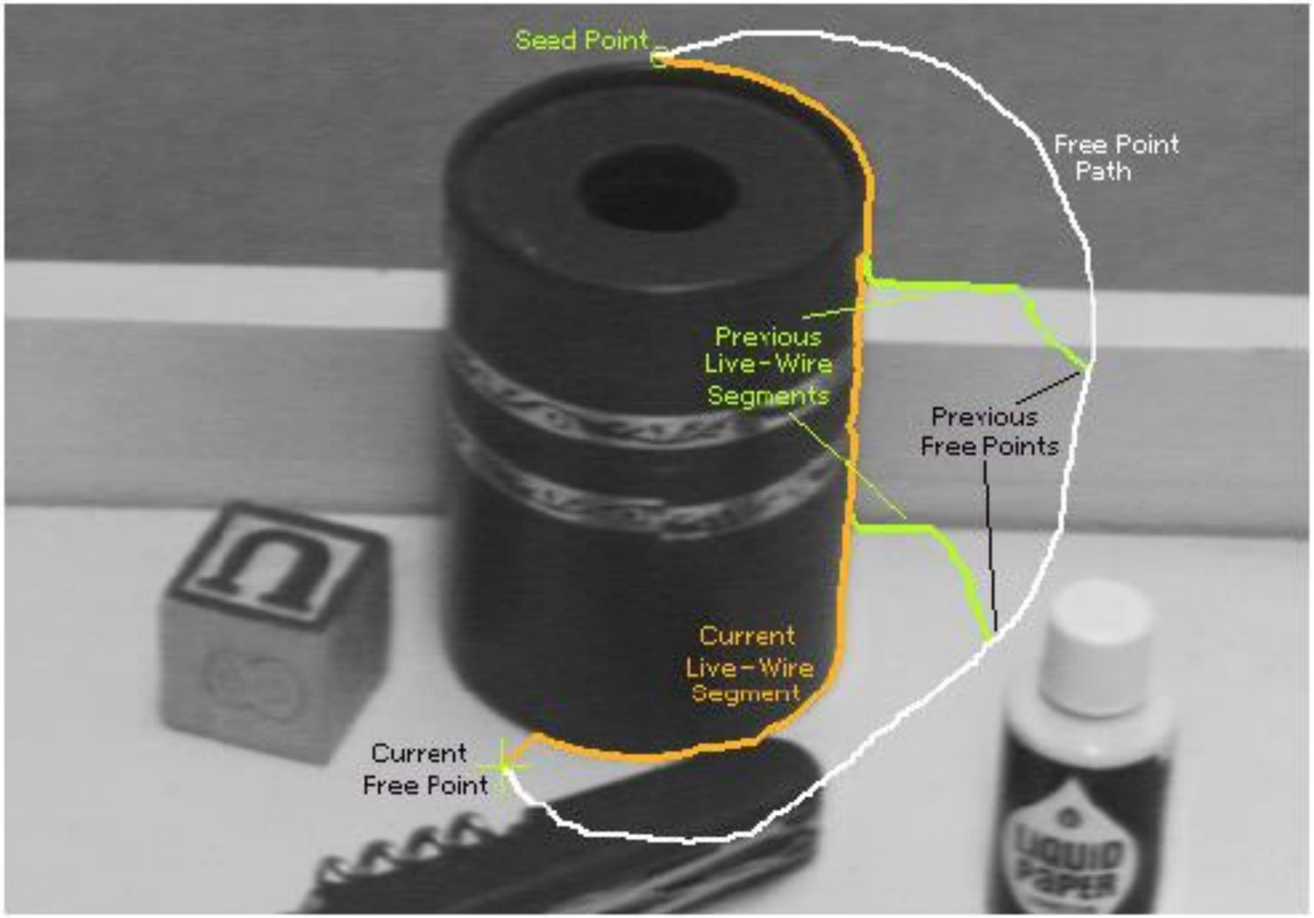


Figure 2: *Image demonstrating how the live-wire segment adapts and snaps to an object boundary as the free point moves (via cursor movement). The path of the free point is shown in white. Live-wire segments from previous free point positions (t_0, t_1, and t_2) are shown in green.*

Intelligent Scissors [Mortensen 95]

Approach answers a basic question

- Q: how to find a path from seed to mouse that follows object boundary as closely as possible?

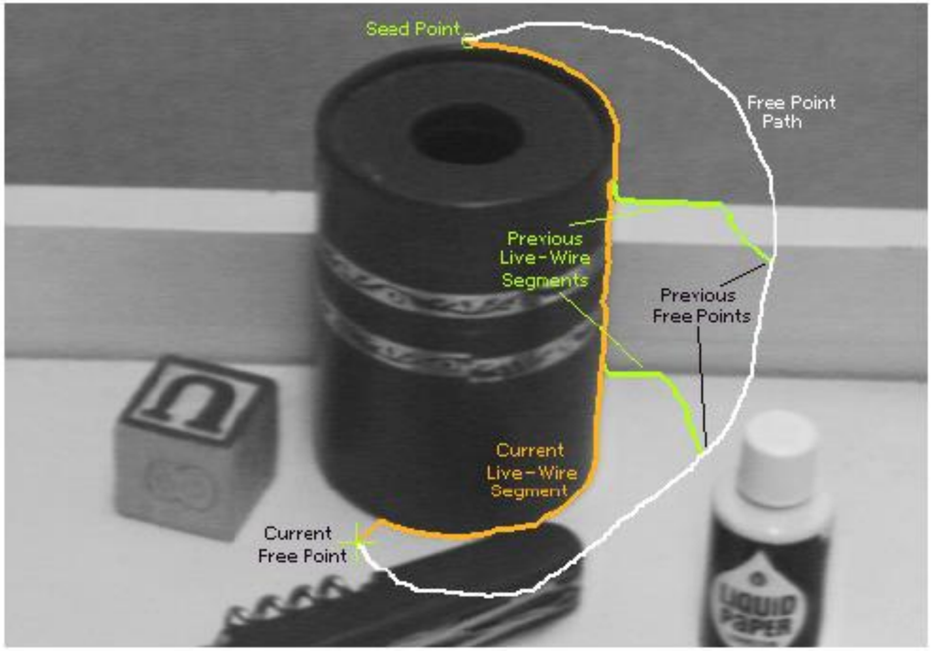
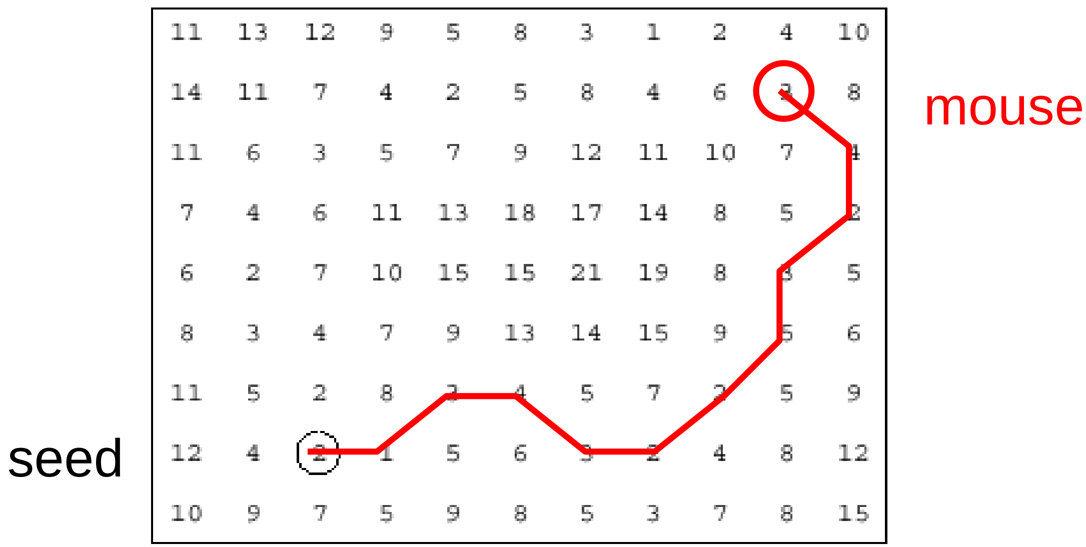


Figure 2: *Image demonstrating how the live-wire segment adapts and snaps to an object boundary as the free point moves (via cursor movement). The path of the free point is shown in white. Live-wire segments from previous free point positions (t_0, t_1, and t_2) are shown in green.*

Intelligent Scissors

Basic Idea

- Define edge score for each pixel
  - edge pixels have low cost
- Find lowest cost path from seed to mouse



Questions

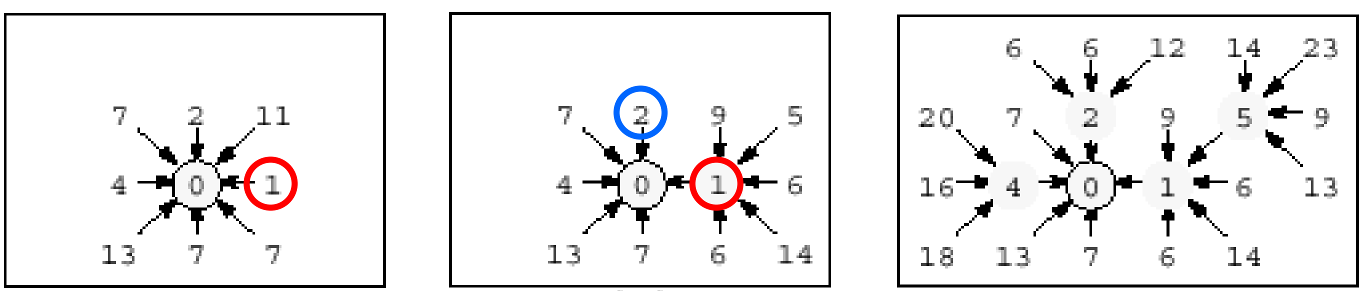
- How to define costs?
- How to find the path?

Path Search (basic idea)

Graph Search Algorithm

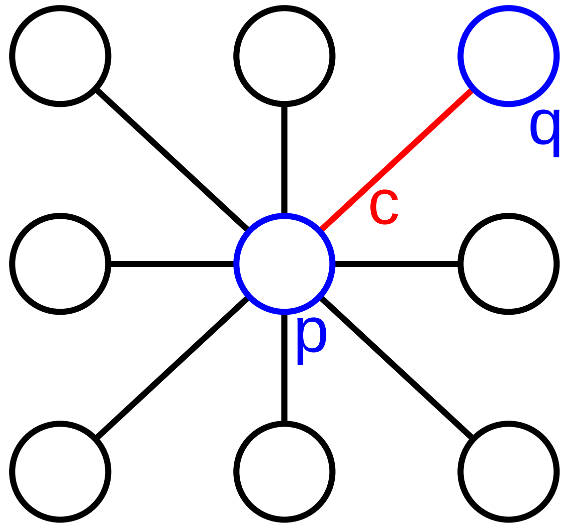
- Computes minimum cost path from seed to *all other pixels*

11	13	12	9	5	8	3	1	2	4	10
14	11	7	4	2	5	8	4	6	3	8
11	6	3	5	7	9	12	11	10	7	4
7	4	6	11	13	18	17	14	8	5	2
6	2	7	10	15	15	21	19	8	3	5
8	3	4	7	9	13	14	15	9	5	6
11	5	2	8	3	4	5	7	2	5	9
12	4	2	1	5	6	3	2	4	8	12
10	9	7	5	9	8	5	3	7	8	15



How does this really work?

Treat the image as a graph



Graph

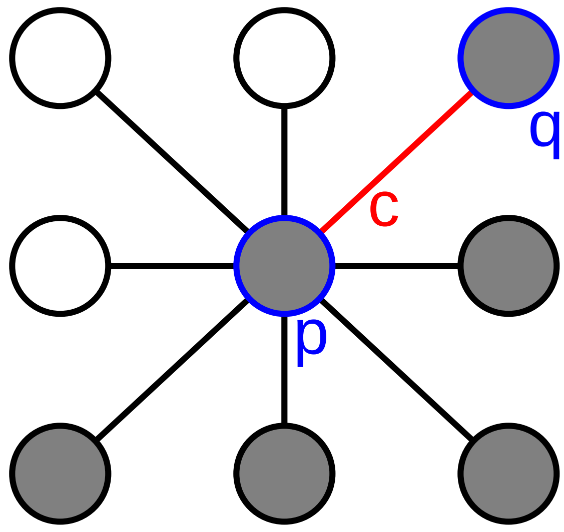
- node for every pixel **p**
- link between every adjacent pair of pixels, **p**,**q**
- cost **c** for each link

Note: each *link* has a cost

- this is a little different than the figure before where each pixel had a cost

Defining the costs

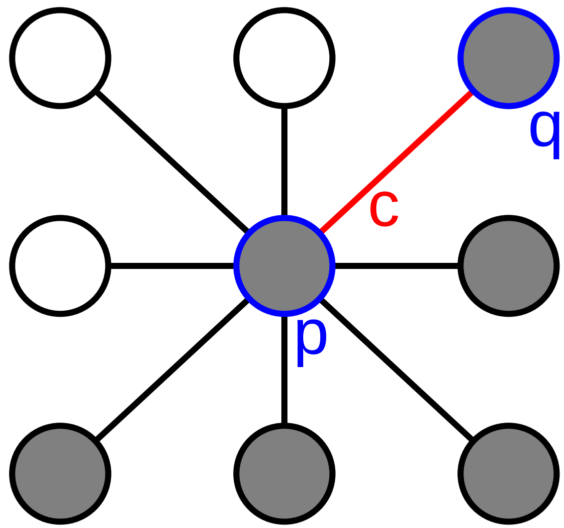
Treat the image as a graph

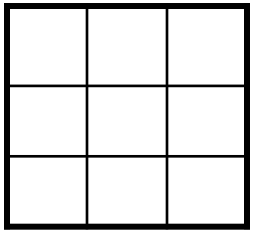


Want to hug image edges: how to define cost of a link?

- the link should follow the intensity edge
  - want intensity to change rapidly $\perp$ to the link
- c $\approx$ - |difference of intensity $\perp$ to link|

Defining the costs





H_c

c can be computed using a cross-correlation filter

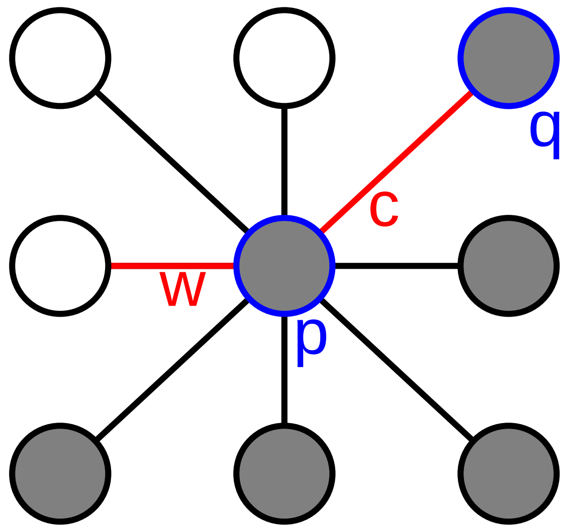
- assume it is centered at p

Also typically scale c by its length

- set c = (max-|filter response|)
  - where max = maximum |filter response| over all pixels in the image

Defining the costs

$$H_w = \frac{1}{4}\begin{bmatrix} 1 & 1 & \\ & & \\ -1 & -1 & \end{bmatrix}$$



$$H_c = \frac{1}{\sqrt{2}}\begin{bmatrix} & 1 & \\ & & -1 \\ & & \end{bmatrix}$$

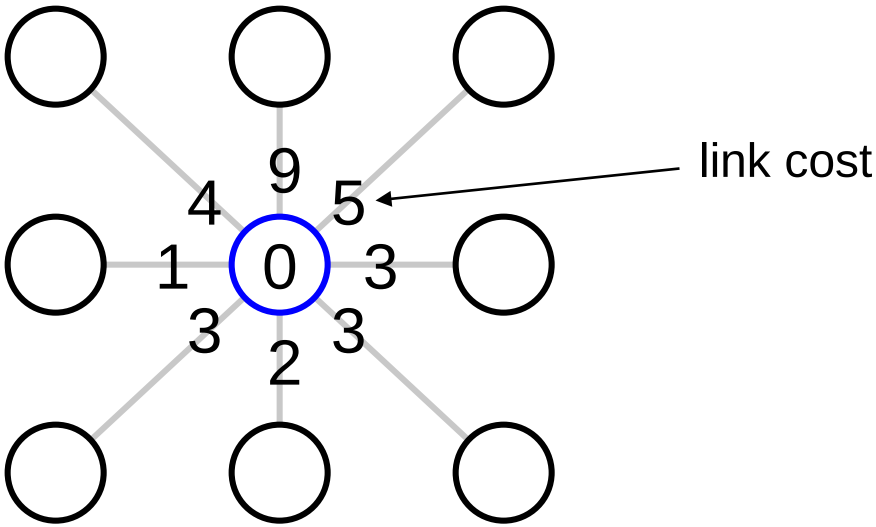
c can be computed using a cross-correlation filter

- assume it is centered at p

Also typically scale c by its length

- set c = (max-|filter response|)
  - where max = maximum |filter response| over all pixels in the image

Dijkstra's shortest path algorithm



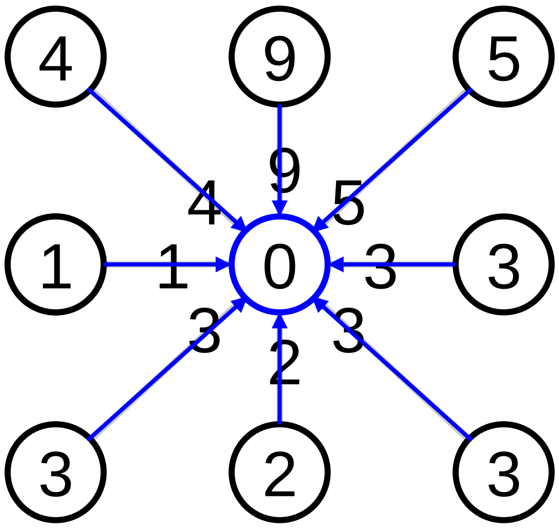
Algorithm

1. init node costs to ∞, set p = seed point, cost(p) = 0
2. expand p as follows:

   for each of p's neighbors q that are not expanded

   » set cost(q) = min(cost(p) + c_{pq}, cost(q))

Dijkstra's shortest path algorithm



Algorithm

1. init node costs to ∞, set p = seed point, cost(p) = 0
2. expand p as follows:

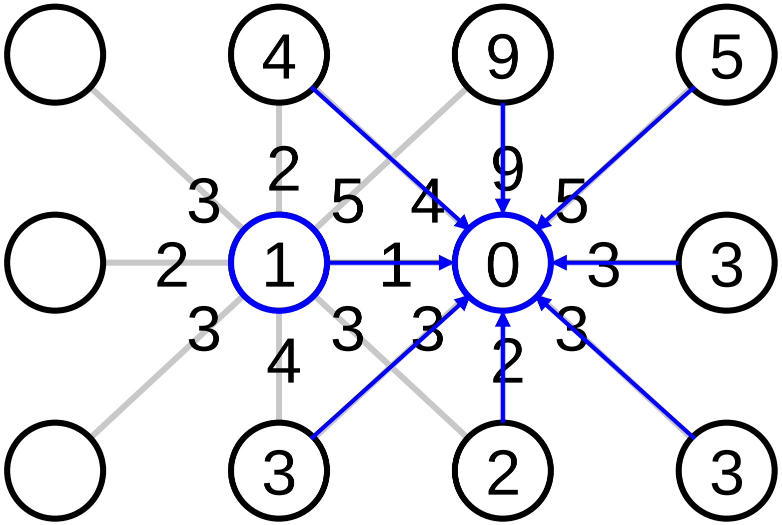
   for each of p's neighbors q that are not expanded

   » set cost(q) = min(cost(p) + c_{pq}, cost(q))

   » if q's cost changed, make q point back to p

   » put q on the ACTIVE list (if not already there)

Dijkstra's shortest path algorithm



Algorithm

1. init node costs to ∞, set p = seed point, cost(p) = 0
2. expand p as follows:

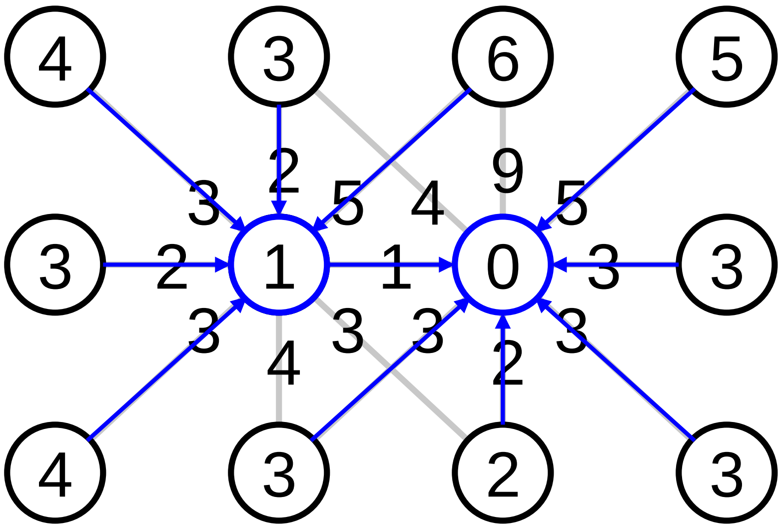
   for each of p's neighbors q that are not expanded

   » set cost(q) = min(cost(p) + c_{pq}, cost(q))

   » if q's cost changed, make q point back to p

   » put q on the ACTIVE list (if not already there)
3. set r = node with minimum cost on the ACTIVE list
4. repeat Step 2 for p = r

Dijkstra’s shortest path algorithm



Algorithm

1. init node costs to ∞, set p = seed point, cost(p) = 0
2. expand p as follows:

   for each of p’s neighbors q that are not expanded

   » set cost(q) = min(cost(p) + c_{pq}, cost(q))

     » if q’s cost changed, make q point back to p

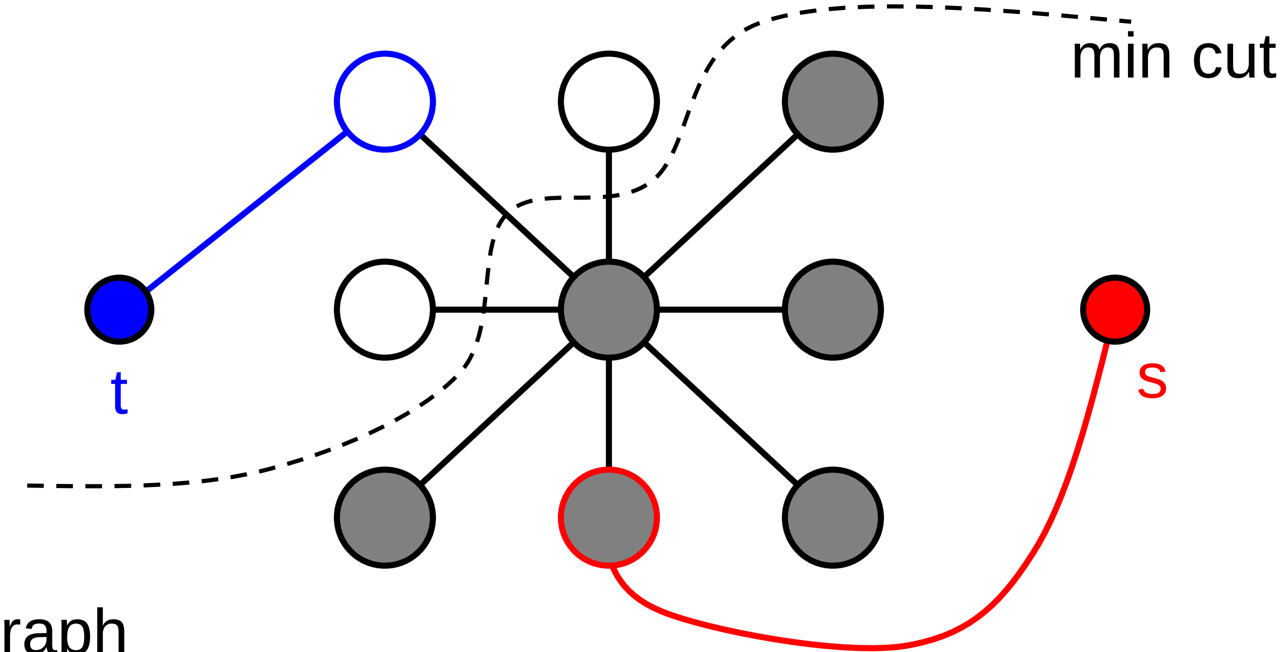
   » put q on the ACTIVE list (if not already there)
3. set r = node with minimum cost on the ACTIVE list
4. repeat Step 2 for p = r

Dijkstra’s shortest path algorithm

Properties

- It computes the minimum cost path from the seed to every node in the graph. This set of minimum paths is represented as a *tree*
- Running time, with N pixels:
  - $O(N^2)$ time if you use an active list
  - O(N log N) if you use an active priority queue (heap)
  - takes fraction of a second for a typical (640x480) image
- Once this tree is computed once, we can extract the optimal path from any point to the seed in O(N) time.
  - it runs in real time as the mouse moves
- What happens when the user specifies a new seed?

Segmentation by min (s-t) cut [Boykov 2001]



Graph

- node for each pixel, link between 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
- compute min cut that separates s from t
- how to define link cost between neighboring pixels?

Grabcut [Rother et al., SIGGRAPH 2004]

Is user-input required?

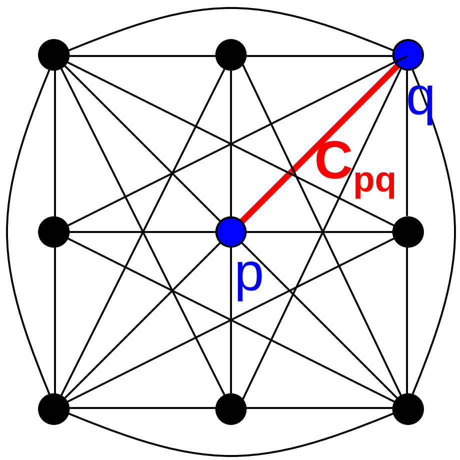
Our visual system is proof that automatic methods are possible

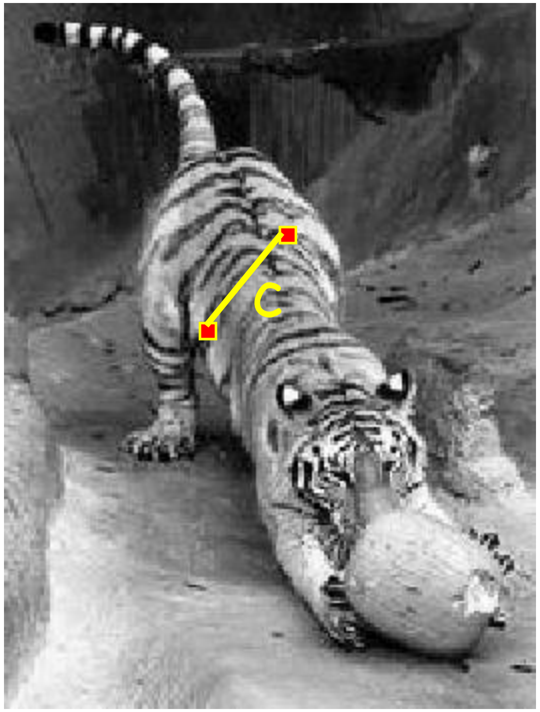
- classical image segmentation methods are automatic

Argument for user-directed methods?

- only user knows desired scale/object of interest

Automatic graph cut [Shi & Malik]

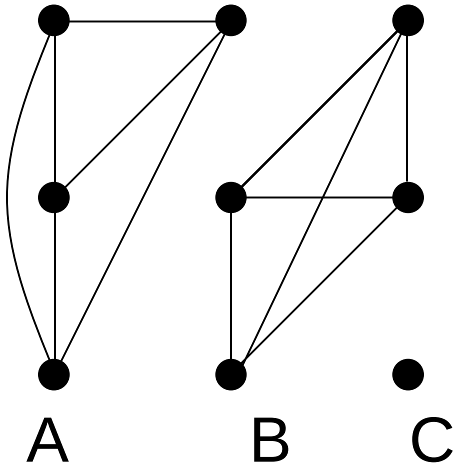




Fully-connected graph

- 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

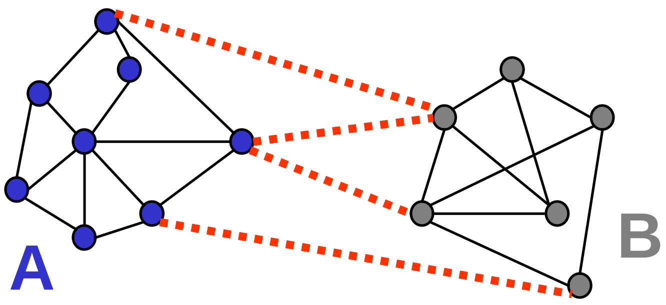
Segmentation by Graph Cuts



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

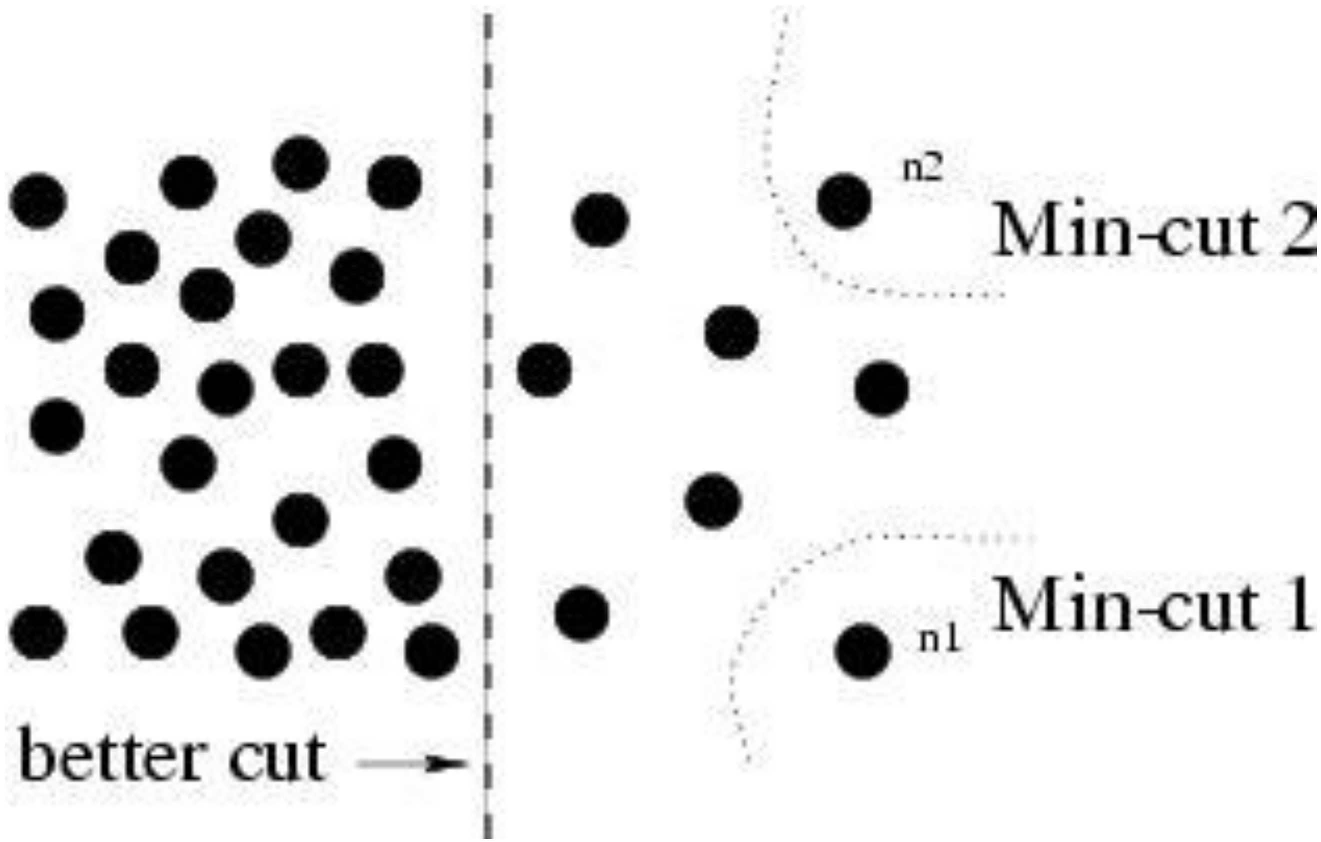
- set of links whose removal makes a graph disconnected
- cost of a cut:

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

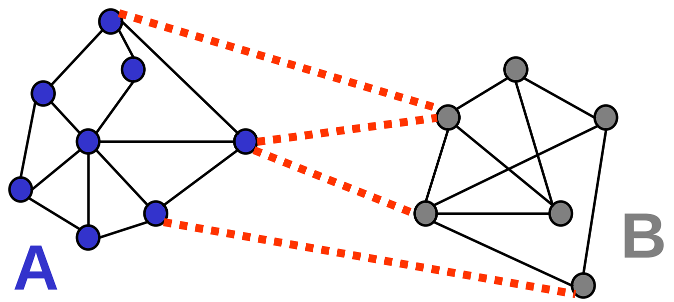
Find minimum cut

- gives you a segmentation

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



Cuts in a graph



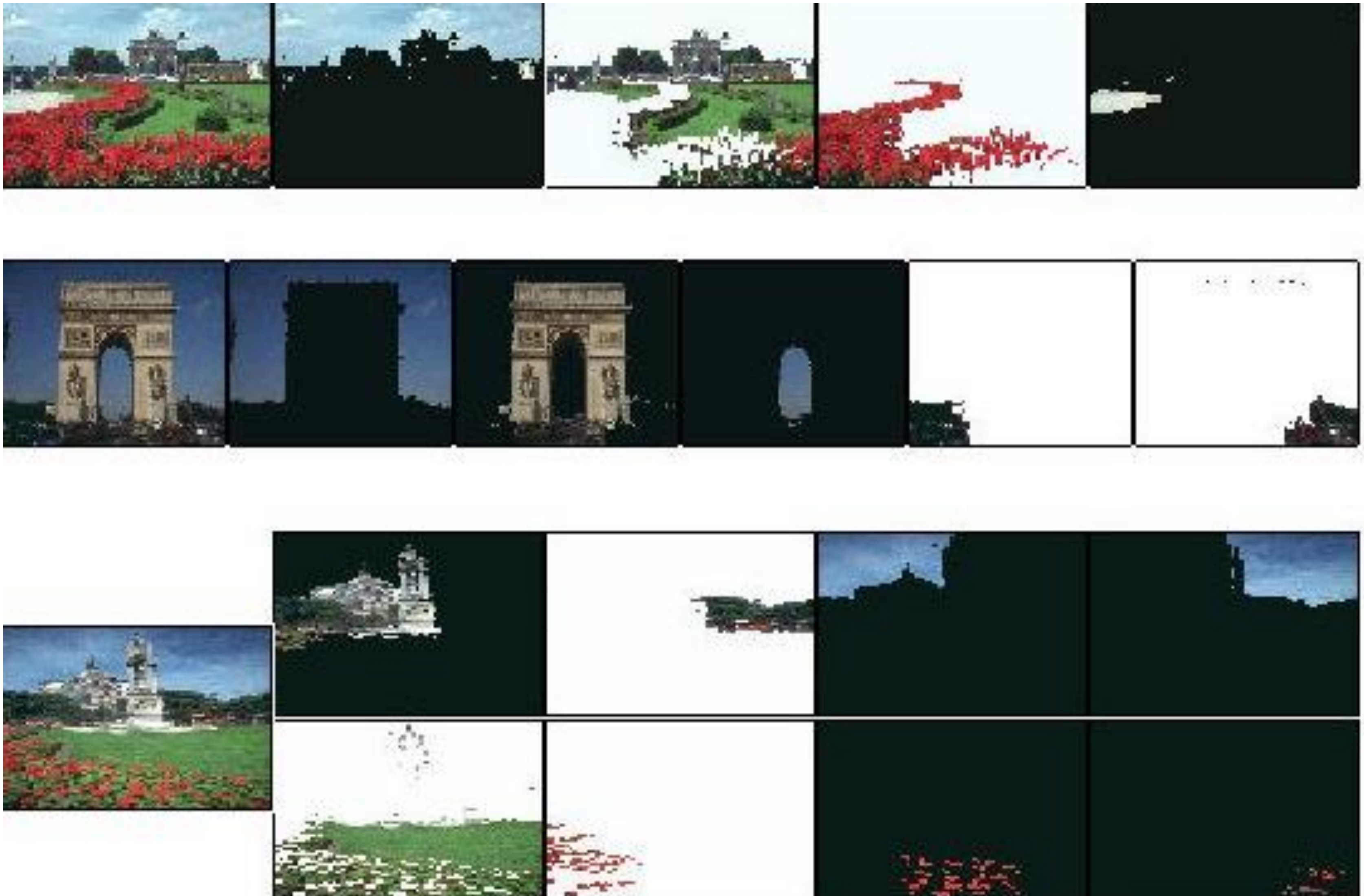
Normalized Cut

- 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

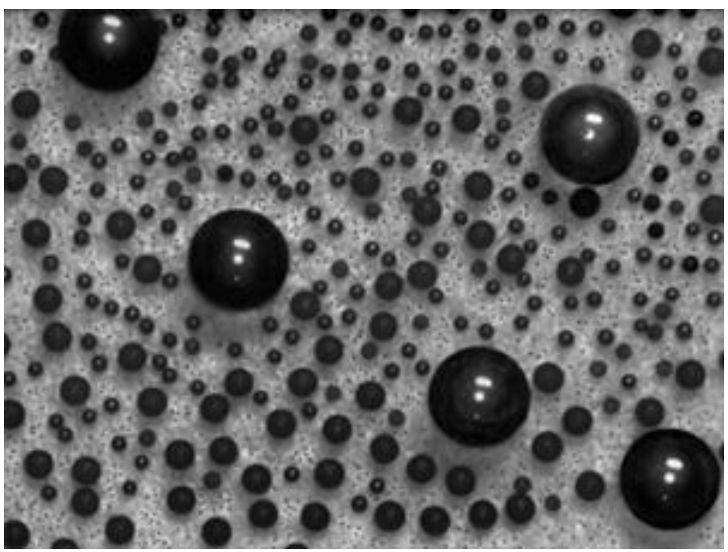
Color Image Segmentation



Histogram-based segmentation

Goal

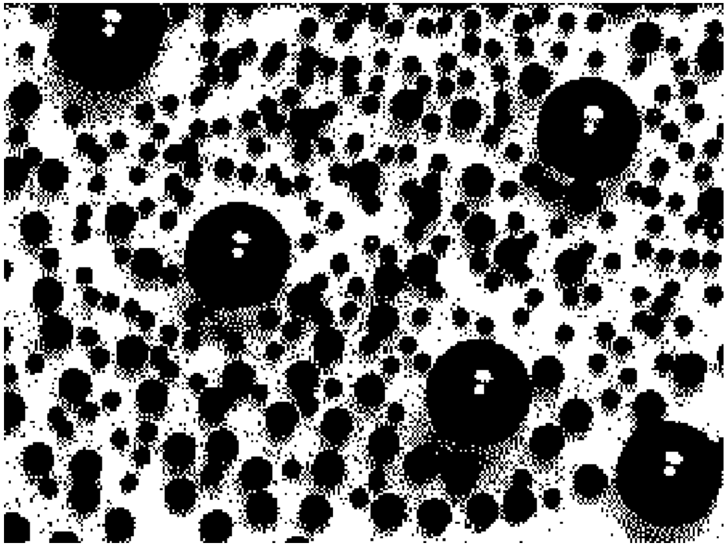
- Break the image into K regions (segments)
- Solve this by reducing the number of colors to K and mapping each pixel to the closest color

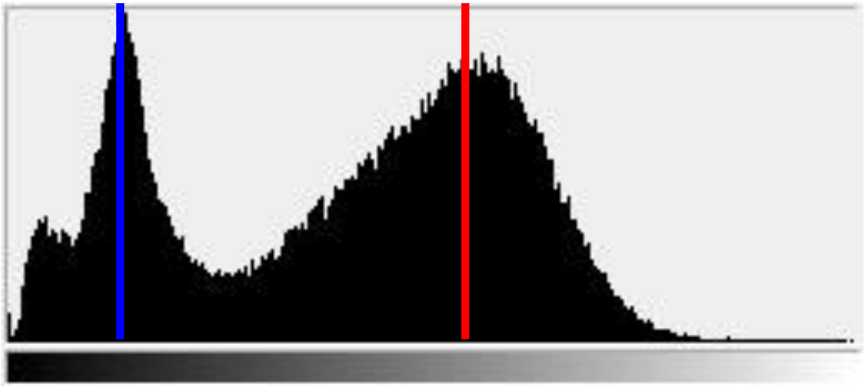


Histogram-based segmentation

Goal

- Break the image into K regions (segments)
- Solve this by reducing the number of colors to K and mapping each pixel to the closest color



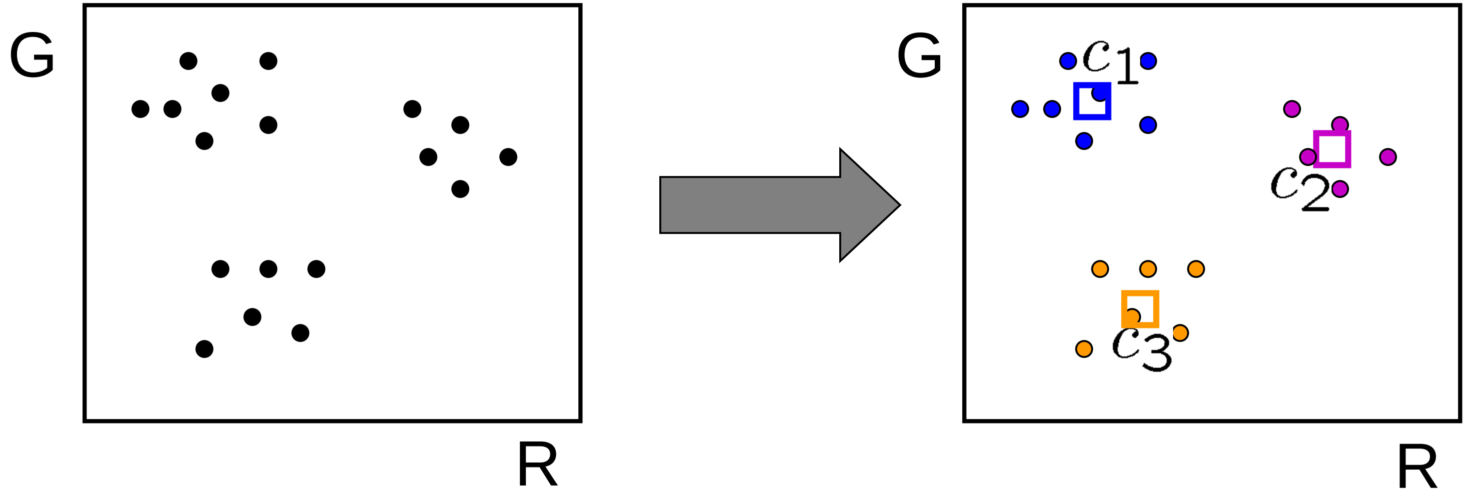


Here's what it looks like if we use two colors

Clustering

How to choose the representative colors?

- This is a clustering problem!



Objective

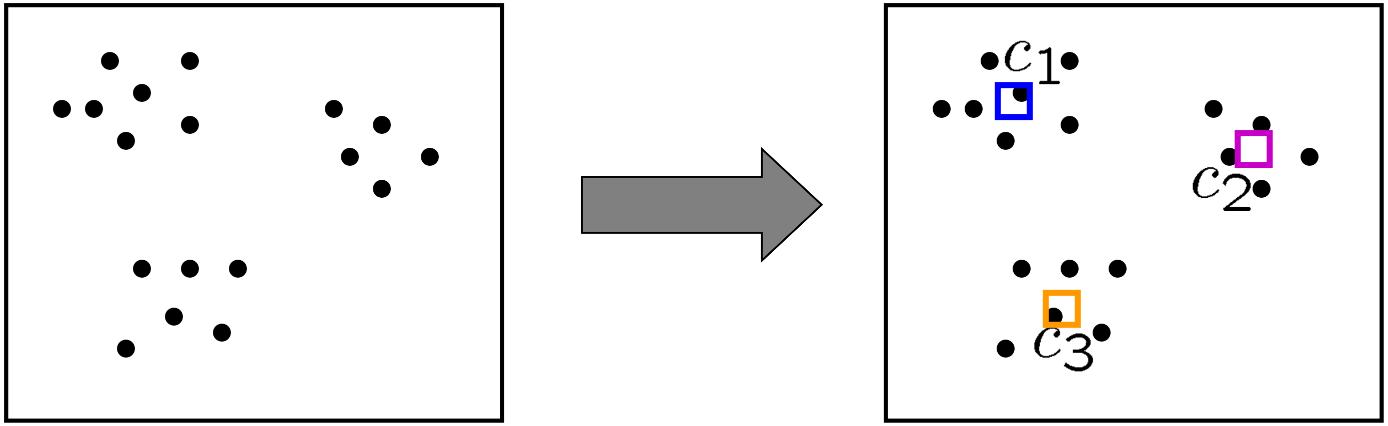
- Each point should be as close as possible to a cluster center
  - Minimize sum squared distance of each point to closest center

$$\sum_{\text{clusters } i} \quad \sum_{\text{points p in cluster } i} \quad ||p - c_i||^2$$

Break it down into subproblems

Suppose I tell you the cluster centers c_i

- Q: how to determine which points to associate with each c_i?
- A: for each point p, choose closest c_i



Suppose I tell you the points in each cluster

- Q: how to determine the cluster centers?
- A: choose c_i to be the mean of all points in the cluster

K-means clustering

K-means clustering algorithm

1. Randomly initialize the cluster centers, $c_1, ..., c_K$
2. Given cluster centers, determine points in each cluster
   - For each point p, find the closest c_i. Put p into cluster i
3. Given points in each cluster, solve for c_i
   - Set c_i to be the mean of points in cluster i
4. If c_i have changed, repeat Step 2

Java demo: http://home.dei.polimi.it/matteucc/Clustering/tutorial_html/AppletKM.html

Properties

- Will always converge to *some* solution
- Can be a "local minimum"
  - does not always find the global minimum of objective function:

$$\sum_{\text{clusters } i} \quad \sum_{\text{points p in cluster } i} ||p - c_i||^2$$

K-Means++

Can we prevent arbitrarily bad local minima?

1. Randomly choose first center.
2. Pick new center with prob. proportional to: $||p - c_i||^2$

   (contribution of *p* to total error)
3. Repeat until *k* centers.

expected error = $O(\log k)$ * optimal

Arthur & Vassilvitskii 2007

Probabilistic clustering

Basic questions

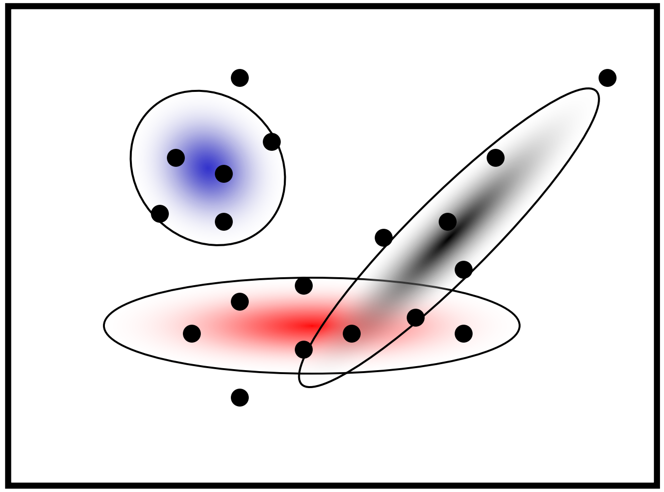
- what’s the probability that a point $\mathbf{x}$ is in cluster m?
- what’s the shape of each cluster?

K-means doesn’t answer these questions

Basic idea

- instead of treating the data as a bunch of points, assume that they are all generated by sampling a continuous function
- This function is called a **generative model**
  - defined by a vector of parameters $\boldsymbol{\theta}$

Mixture of Gaussians



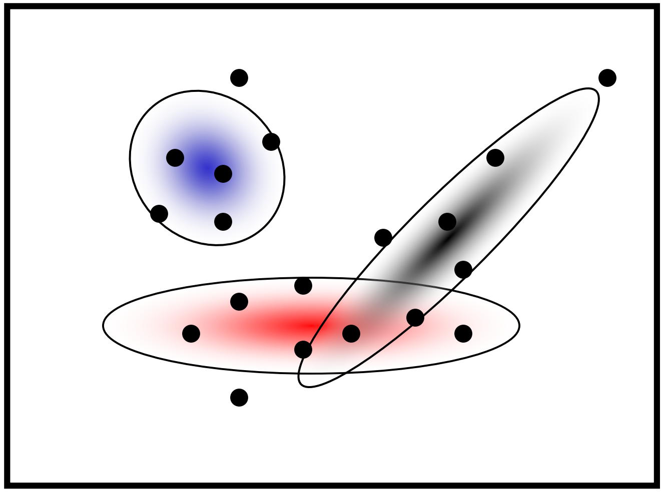
One generative model is a mixture of Gaussians (MOG)

- K Gaussian blobs with means $\boldsymbol{\mu}_b$ covariance matrices $\mathbf{V}_b$, dimension d
  - blob *b* defined by: $P(x|\mu_b, V_b) = \frac{1}{\sqrt{(2\pi)^d |V_b|}} e^{-\frac{1}{2}(x-\mu_b)^T V_b^{-1}(x-\mu_b)}$
- blob *b* is selected with probability α_b
- the likelihood of observing **x** is a weighted mixture of Gaussians

$$P(x|\theta) = \sum_{b=1}^{K} \alpha_b P(x|\theta_b)$$

- where $\theta = [\mu_1, \ldots, \mu_n, V_1, \ldots, V_n]$

Expectation maximization (EM)



Goal

- find blob parameters θ that maximize the likelihood function:

$$P(data|\theta) = \prod_x P(x|\theta)$$

Approach:

1. E step: given current guess of blobs, compute ownership of each point
2. M step: given ownership probabilities, update blobs to maximize likelihood function
3. repeat until convergence

EM details

E-step

- compute probability that point **x** is in blob i, given current guess of **θ**

$$P(b|x, \mu_b, V_b) = \frac{\alpha_b P(x|\mu_b, V_b)}{\sum_{i=1}^{K} \alpha_i P(x|\mu_i, V_i)}$$

M-step

- compute probability that blob b is selected

$$\alpha_b^{new} = \frac{1}{N} \sum_{i=1}^{N} P(b|x_i, \mu_b, V_b)$$

N data points

- mean of blob b

$$\mu_b^{new} = \frac{\sum_{i=1}^{N} x_i P(b|x_i, \mu_b, V_b)}{\sum_{i=1}^{N} P(b|x_i, \mu_b, V_b)}$$

- covariance of blob b

$$V_b^{new} = \frac{\sum_{i=1}^{N} (x_i - \mu_b^{new})(x_i - \mu_b^{new})^T P(b|x_i, \mu_b, V_b)}{\sum_{i=1}^{N} P(b|x_i, \mu_b, V_b)}$$

Applications of EM

Turns out this is useful for all sorts of problems

- any clustering problem
- any model estimation problem
- missing data problems
- finding outliers
- segmentation problems
  - segmentation based on color
  - segmentation based on motion
  - foreground/background separation
- ...

Problems with EM

1. Local minima

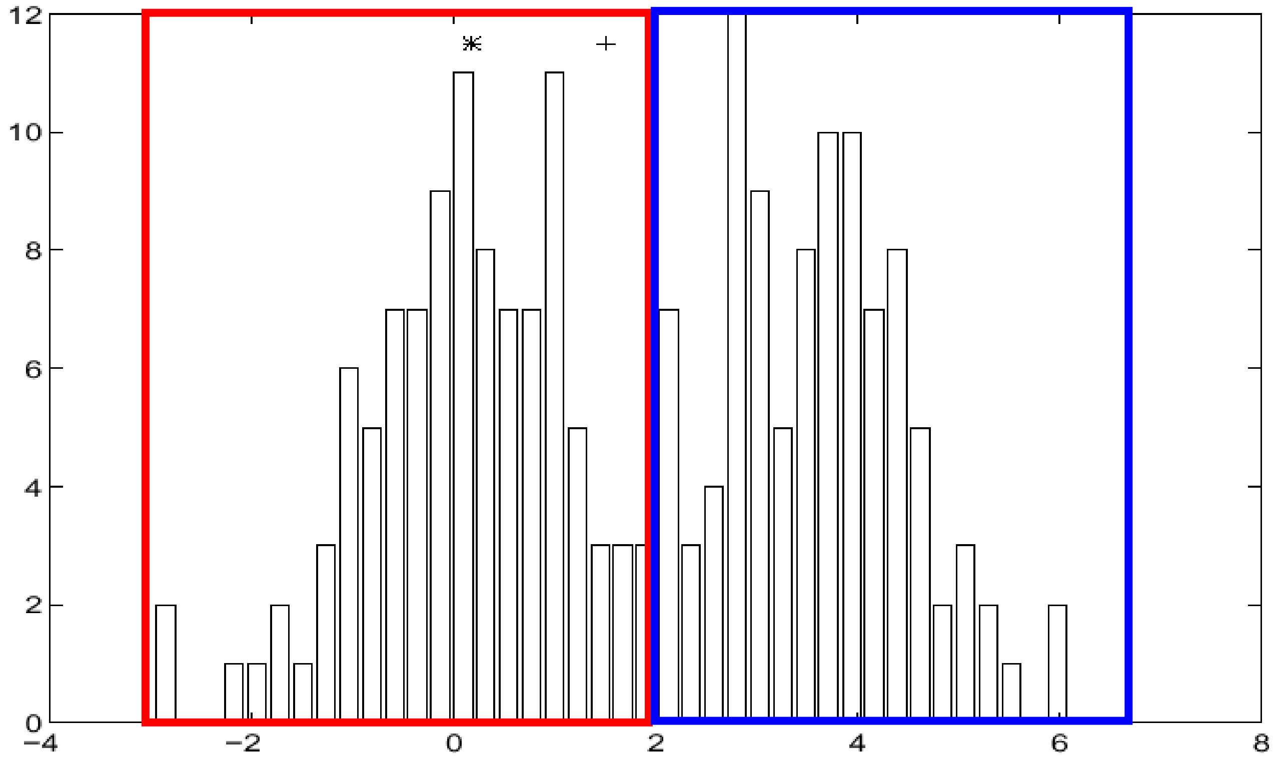
   k-means is NP-hard even with k=2

2. Need to know number of segments

   solutions: AIC, BIC, Dirichlet process mixture

3. Need to choose generative model

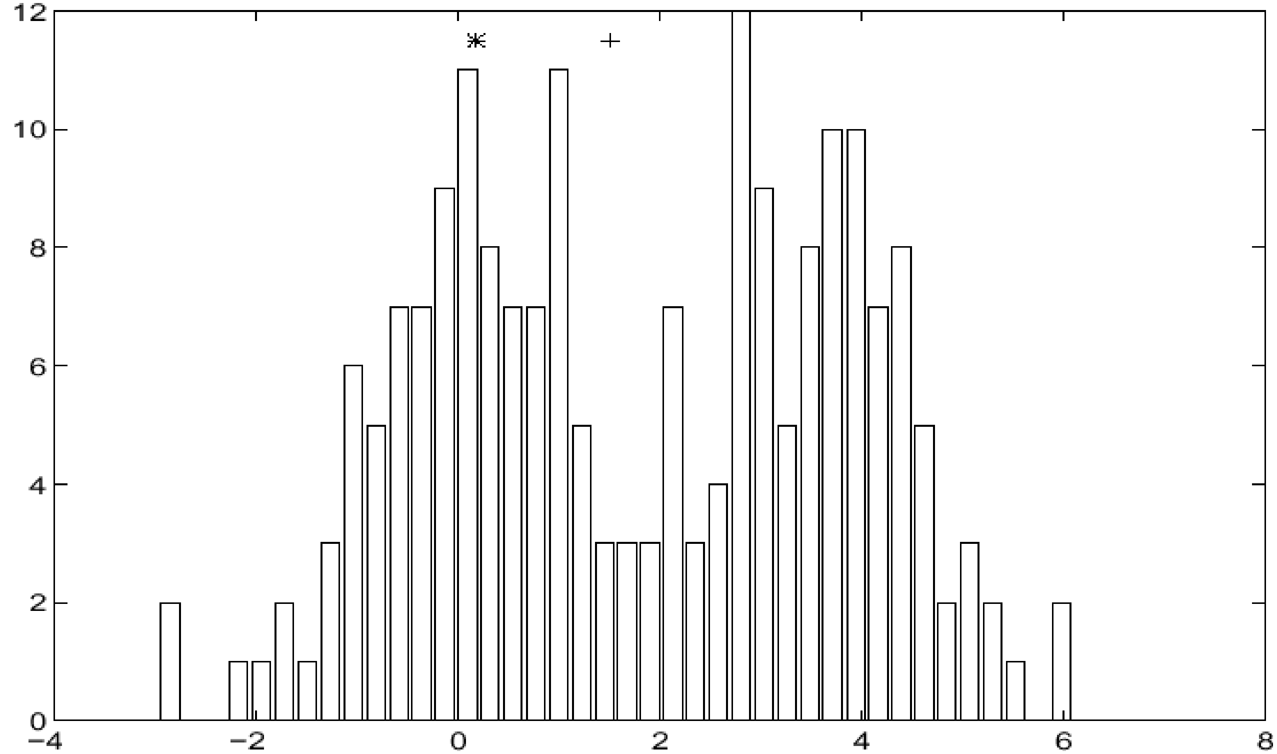
Finding Modes in a Histogram



How Many Modes Are There?

- Easy to see, hard to compute

Mean Shift [Comaniciu & Meer]



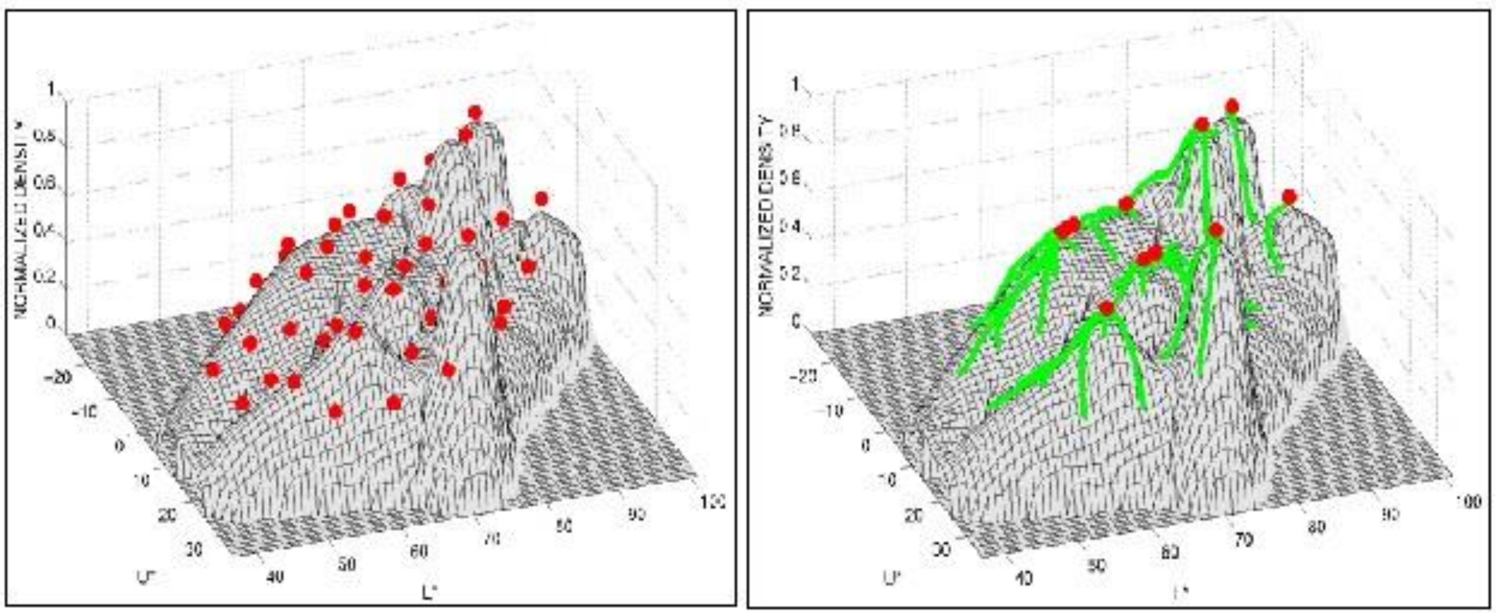
Iterative Mode Search

1. Initialize random seed, and window W
2. Calculate center of gravity (the "mean") of W: $\sum_{x \in W} x H(x)$
3. Translate the search window to the mean
4. Repeat Step 2 until convergence

Mean-Shift

Approach

- Initialize a window around each point
- See where it shifts—this determines which segment it's in
- Multiple points will shift to the same segment

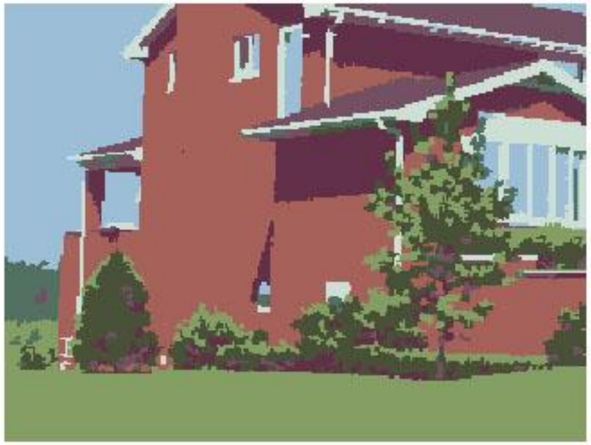


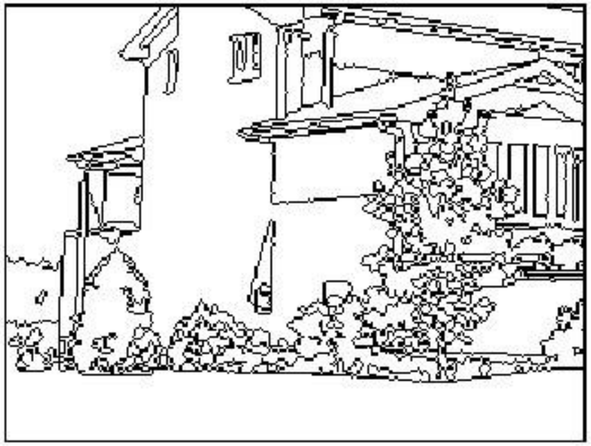
Mean shift trajectories

Mean-shift for image segmentation

Useful to take into account spatial information

- instead of (R, G, B), run in (R, G, B, x, y) space
- D. Comaniciu, P. Meer, Mean shift analysis and applications, *7th International Conference on Computer Vision*, Kerkyra, Greece, September 1999, 1197-1203.
  - http://www.caip.rutgers.edu/riul/research/papers/pdf/spatmsft.pdf





More Examples: http://www.caip.rutgers.edu/~comanici/segm_images.html

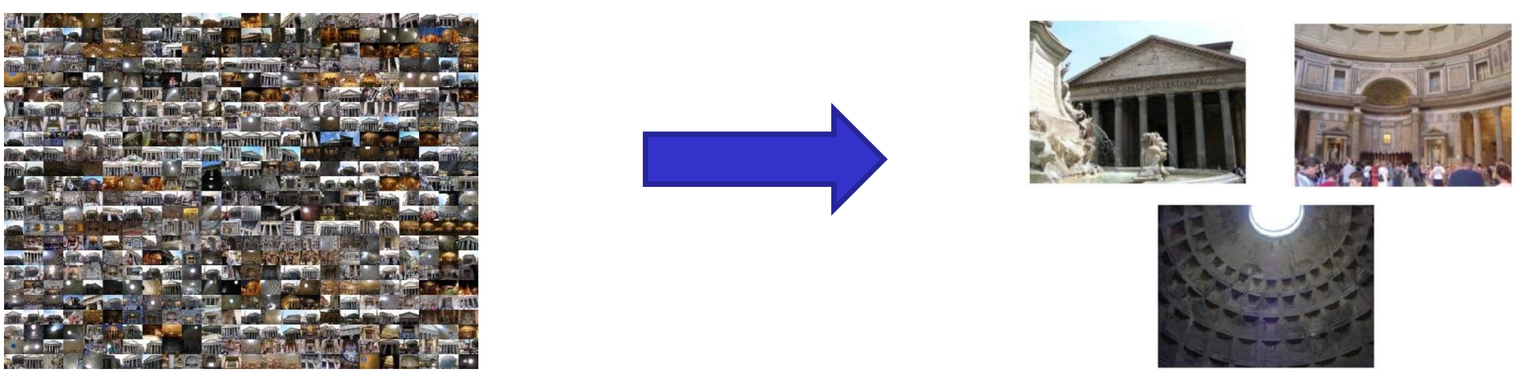
Choosing Exemplars (Medoids)

like k-means, but means must be data points

Algorithms:

- greedy k-means
- affinity propagation (Frey & Dueck 2007)
- medoid shift (Sheikh et al. 2007)

[Scene Summarization]



Taxonomy of Segmentation Methods

- Graph Based vs. Point-Based (bag of pixels)
- User-Directed vs. Automatic
- Partitional vs. Hierarchical

K-Means:

point-based, automatic, partitional

Graph Cut:

graph-based, user-directed, partitional

References

- Mortensen and Barrett, "Intelligent Scissors for Image Composition," Proc. *SIGGRAPH* 1995.
- Boykov and Jolly, "Interactive Graph Cuts for Optimal Boundary & Region Segmentation of Objects in N-D images," Proc. ICCV, 2001.
- Shi and Malik, "Normalized Cuts and Image Segmentation," Proc. CVPR 1997.
- Comaniciu and Meer, "Mean shift analysis and applications," Proc. *ICCV* 1999.