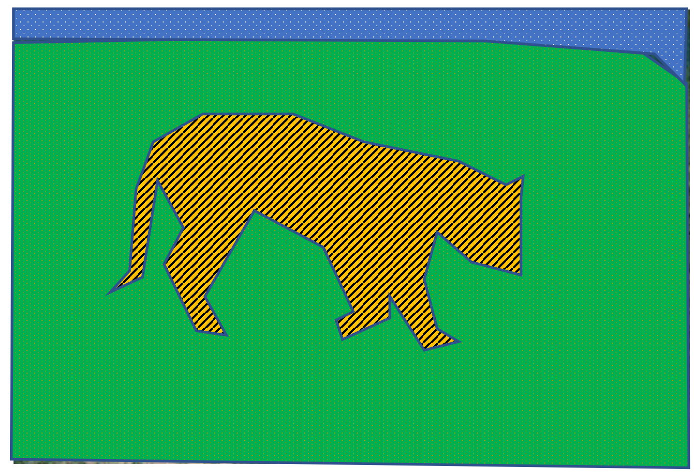
Grouping

What is grouping?



K-means

Input: set of data points, k

1. Randomly pick k points as means
2. For i in [0, maxiters]:
    1. Assign each point to nearest center
    2. Re-estimate each center as mean of points assigned to it

K-means - the math

Input: set of data points X, k

1. Randomly pick k points as means $\mu_i, i = 1, \dots, k$
2. For iteration in [0, maxiters]:
   1. Assign each point to nearest center
   $$y_i = \arg\min_j \|x_i - \mu_j\|^2$$
   2. Re-estimate each center as mean of points assigned to it
   $$\mu_j = \frac{\sum_{i:y_i=j} x_i}{\sum_{i:y_i=j} 1}$$

K-means - the math

- An objective function that must be minimized:

$$\min_{\mu, y} \sum_i \|x_i - \mu_{y_i}\|^2$$

- Every iteration of k-means takes a downward step:
    - Fixes μ and sets y to minimize objective
    - Fixes y and sets μ to minimize objective

K-means on image pixels

K-means on image pixels

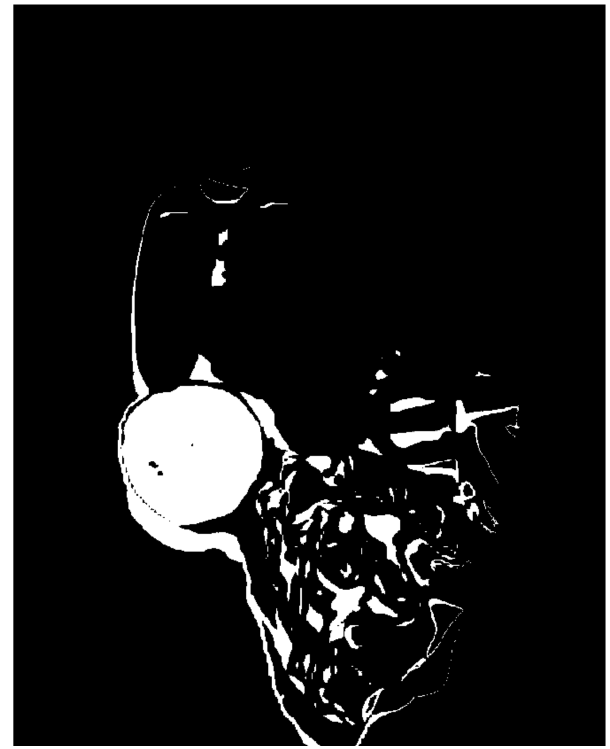
Picture courtesy David Forsyth

One of the clusters from k-means

K-means on image pixels

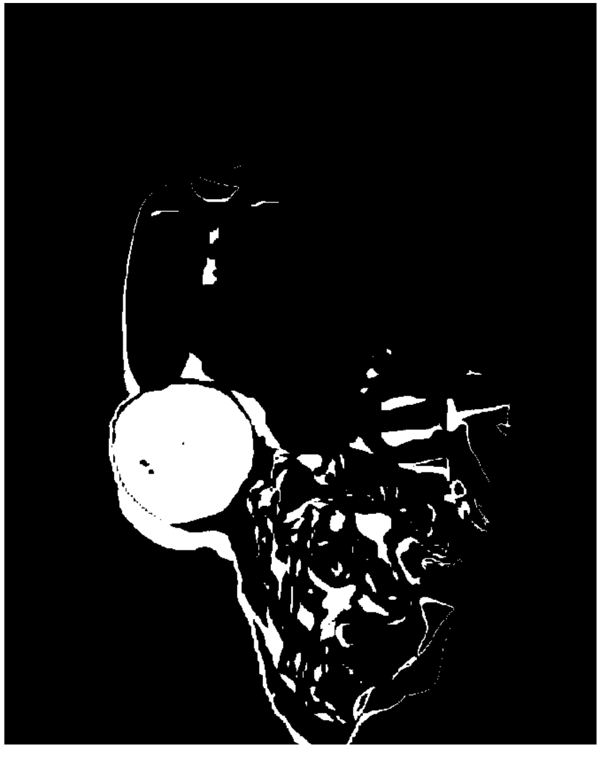
- What is wrong?
- Pixel position
  - Nearby pixels are likely to belong to the same object
  - Far-away pixels are likely to belong to different objects
- How do we incorporate pixel position?
  - Instead of representing each pixel as (r,g,b)
  - Represent each pixel as (r,g,b,x,y)

K-means on image pixels



The issues with k-means

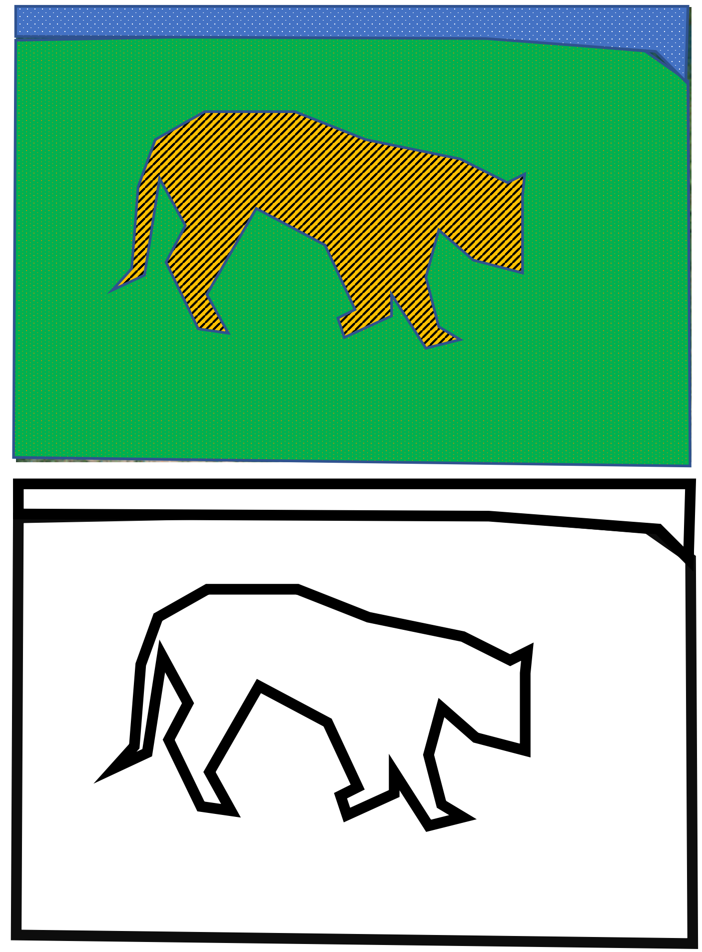
- Captures pixel similarity but
  - Doesn't capture continuity
  - Captures proximity only weakly
  - Can merge far away objects together
- Requires knowledge of k!



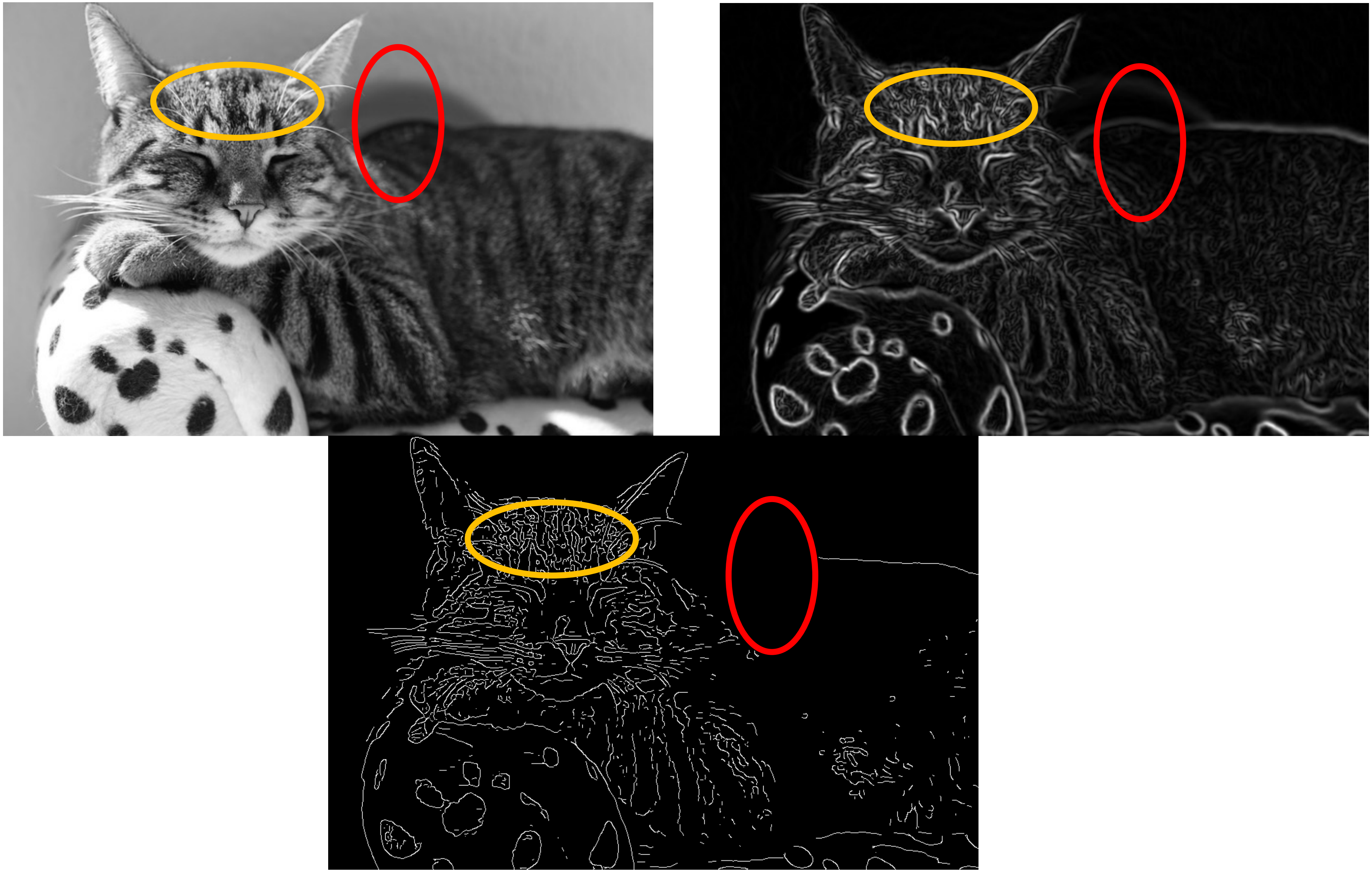
Oversegmentation and superpixels

- We don't know k. What is a safe choice?
- Idea: Use large k
  - Can potentially break big objects, but will hopefully not merge unrelated objects
  - Later processing can decide which groups to merge
  - Called *superpixels*

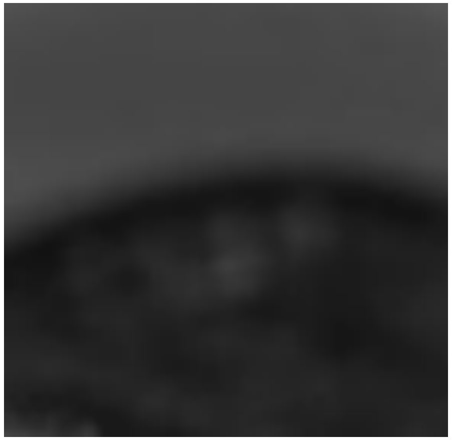
Regions ⟷ Boundaries

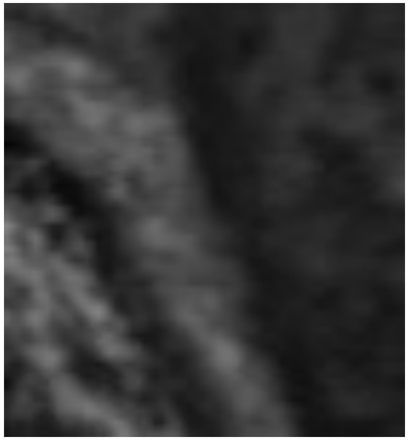


Does Canny always work?

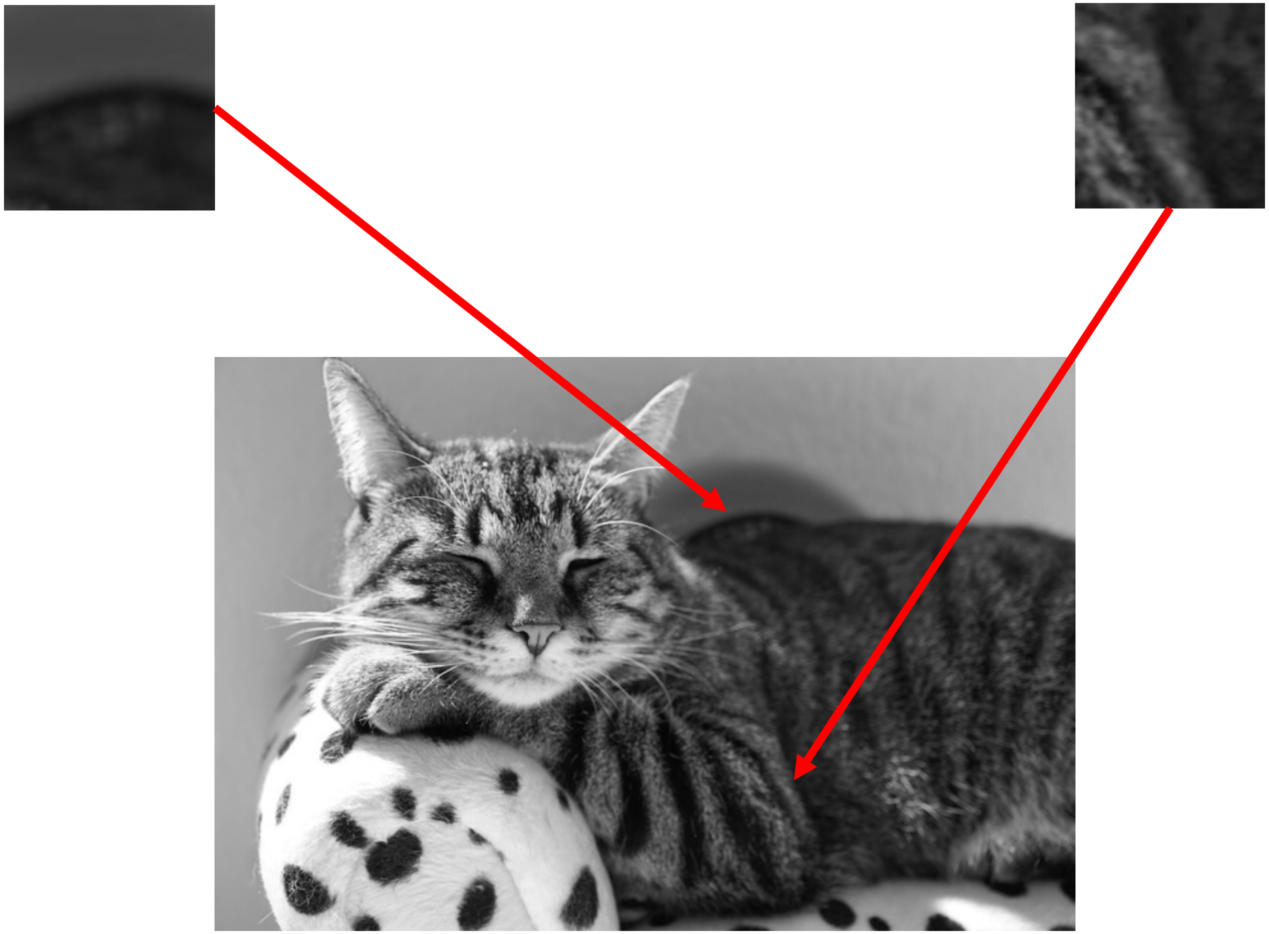


The aperture problem

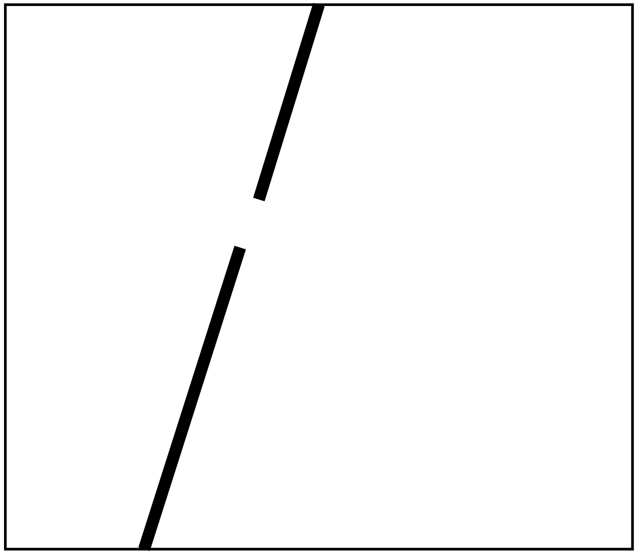




The aperture problem

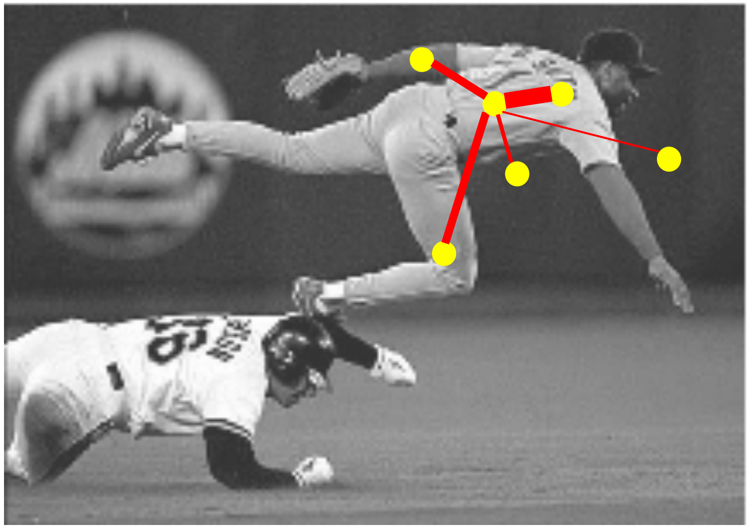


“Globalisation”

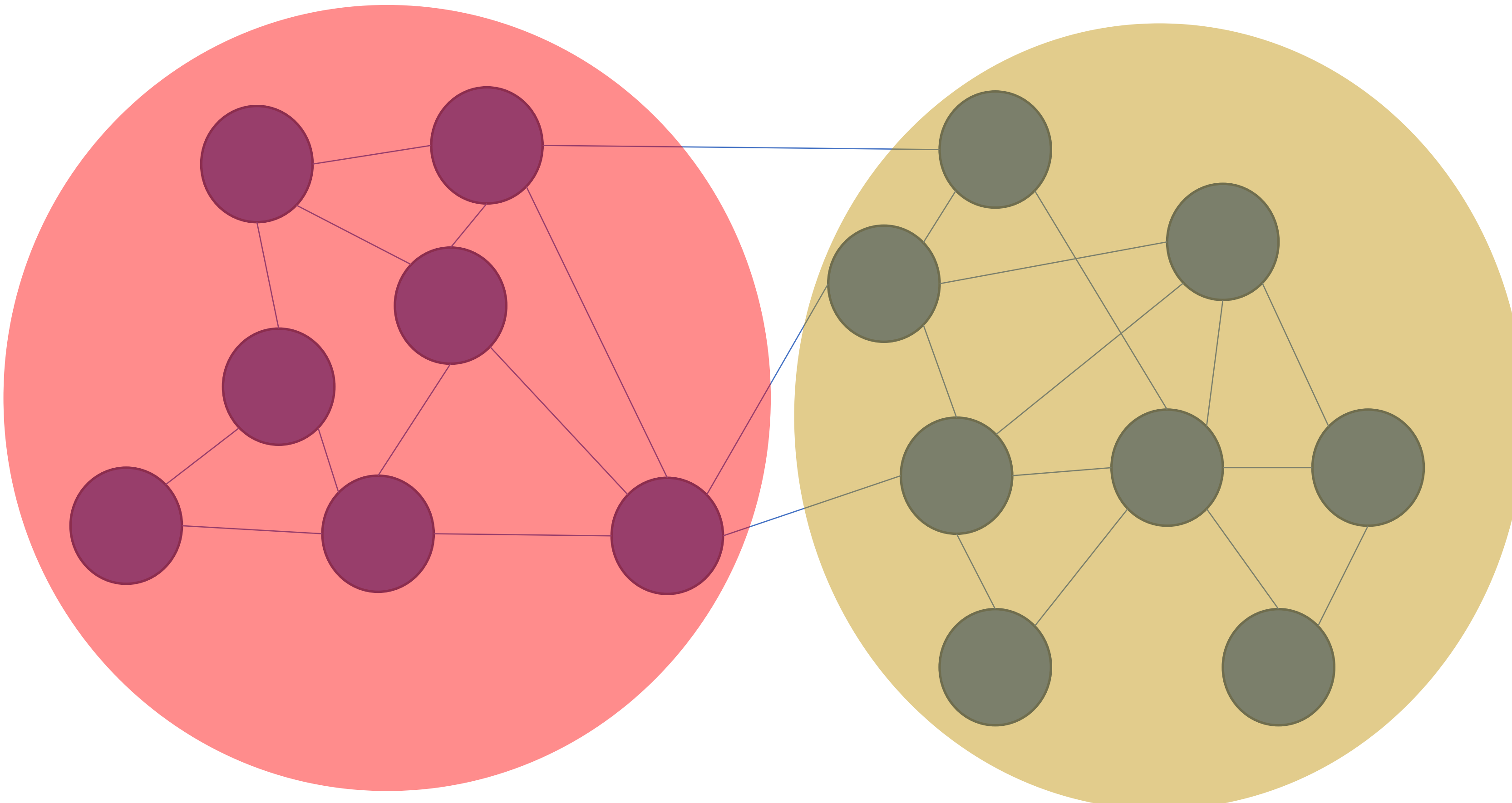


Images as graphs

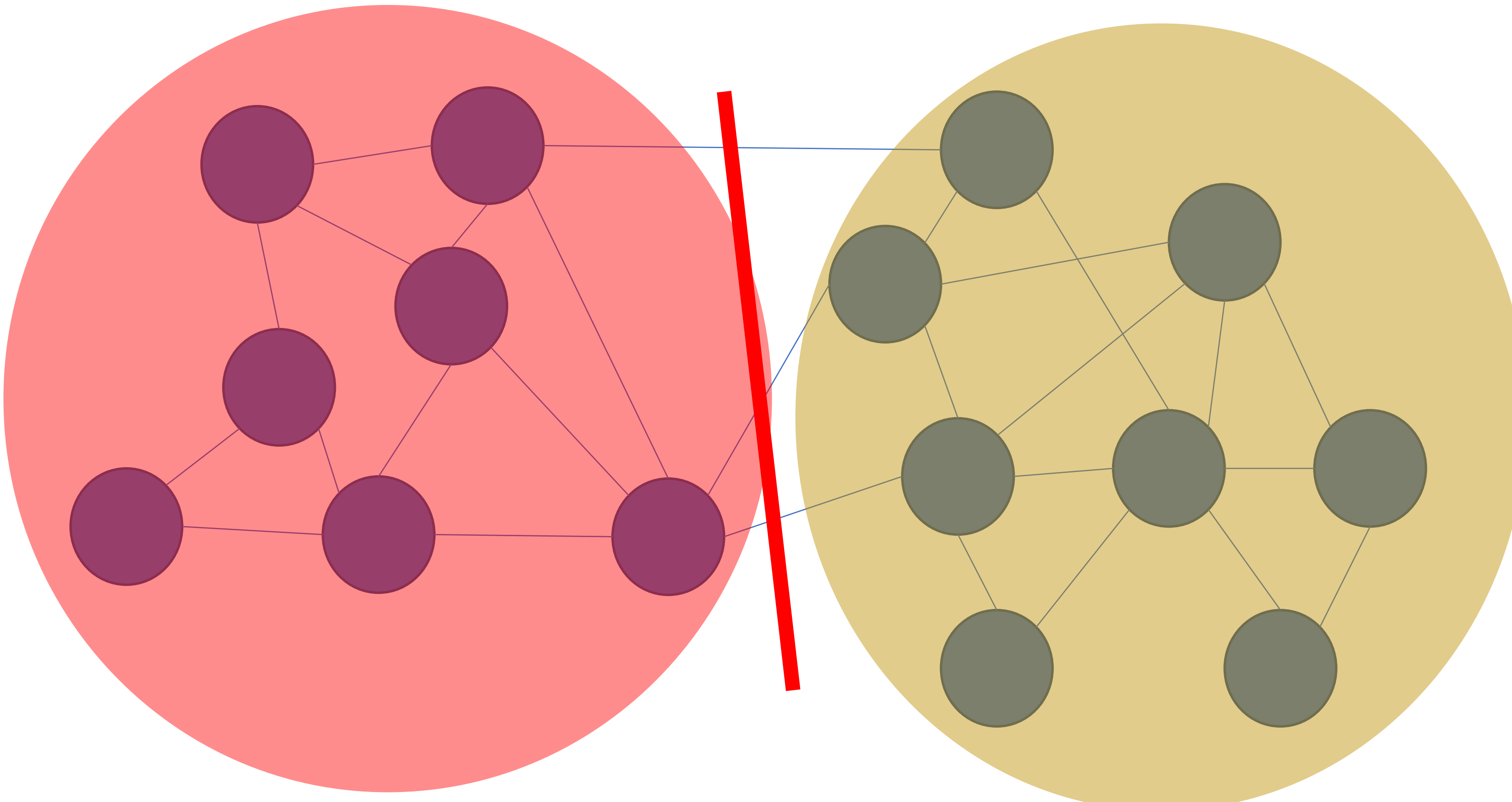
- Each pixel is node
- Edge between "similar pixels"
  - *Proximity:* nearby pixels are more similar
  - *Similarity:* pixels with similar color are more similar
- Weight of edge = similarity



Segmentation is graph partitioning

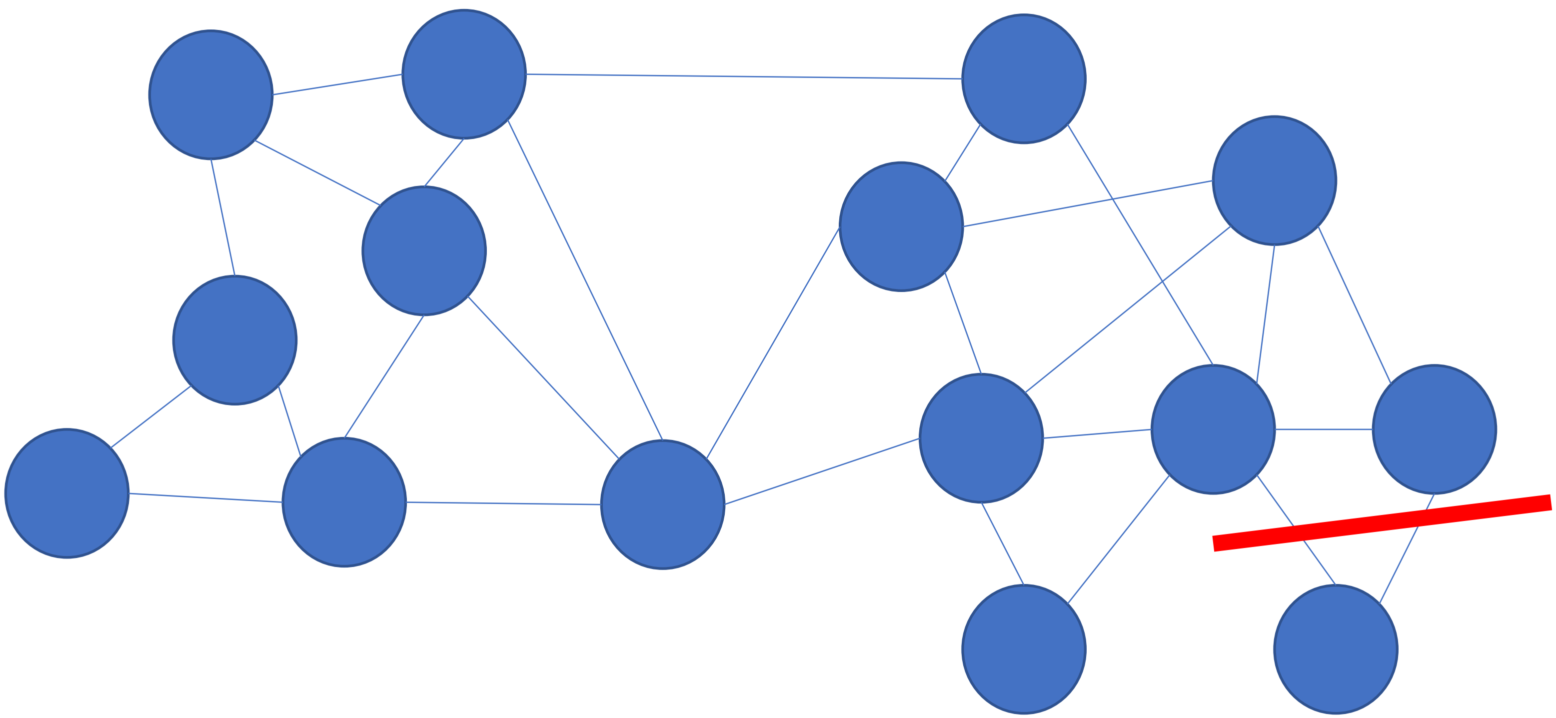


Segmentation is graph partitioning



- Every partition "cuts" some edges
- Idea: minimize total weight of edges cut!

Criterion: Min-cut?

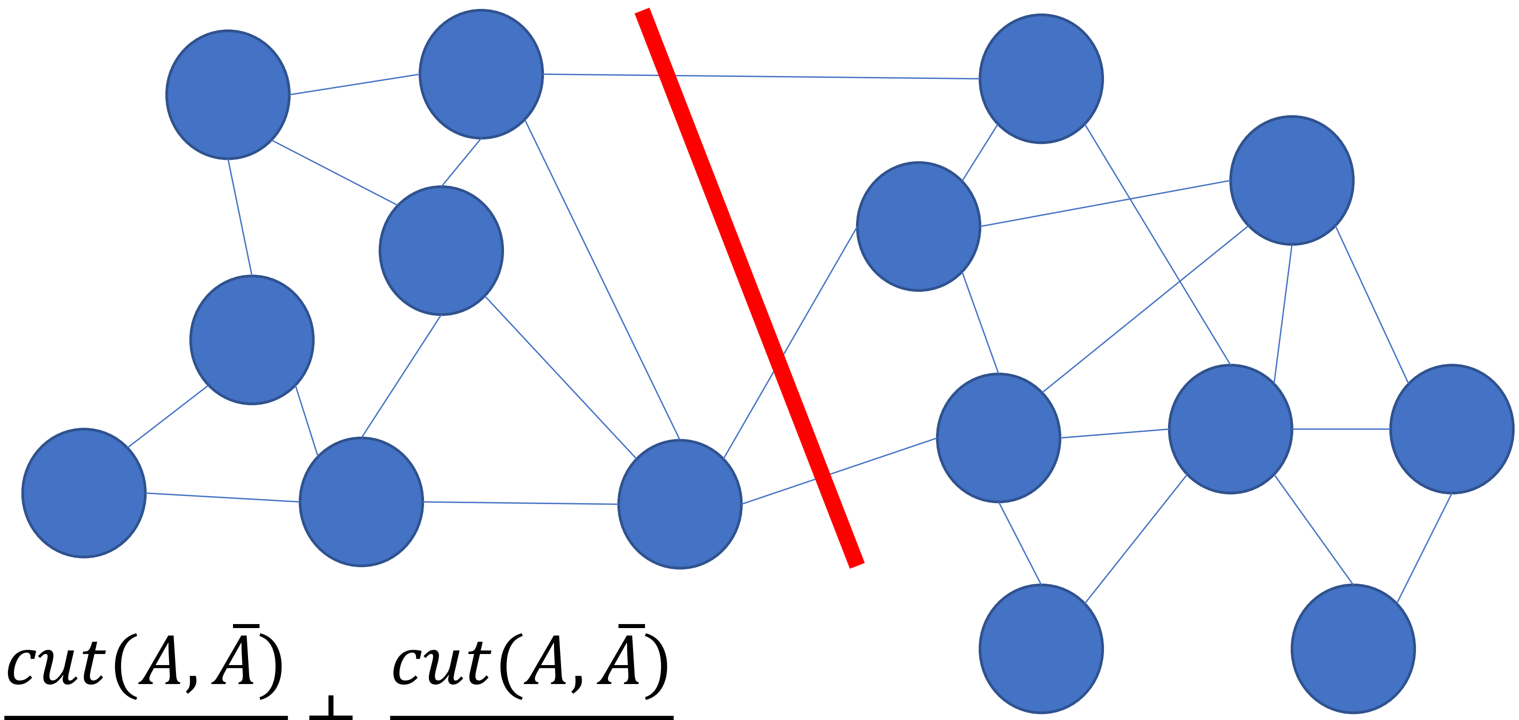


- Min-cut carves out small isolated parts of the graph
- In image segmentation: individual pixels

Normalized cuts

- “Cut” = total weight of cut edges
- Small cut means the groups don’t “like” each other
- But need to normalize w.r.t how much they like *themselves*
- *“Volume”* of a subgraph = total weight of edges within the subgraph

Normalized cut

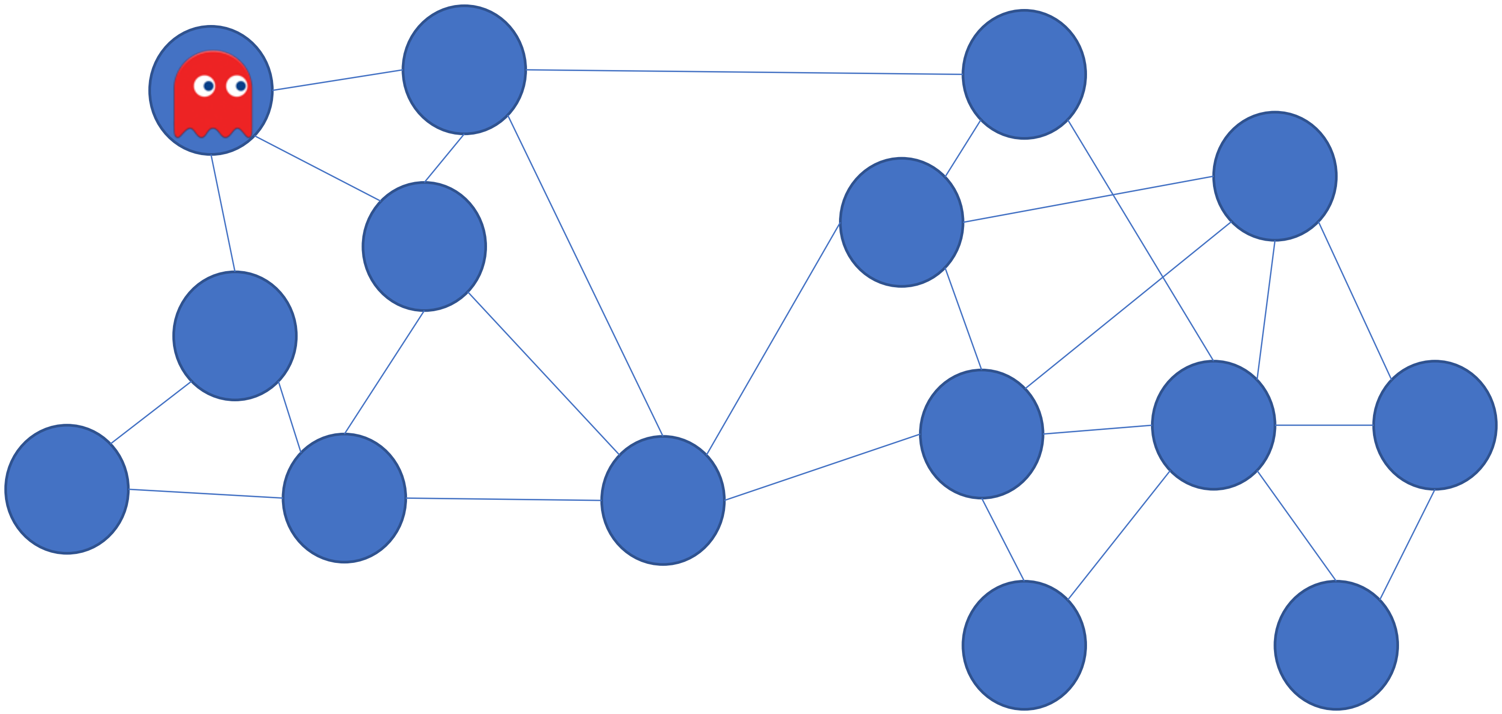


$$\frac{cut(A,\bar{A})}{vol(A)} + \frac{cut(A,\bar{A})}{vol(\bar{A})}$$

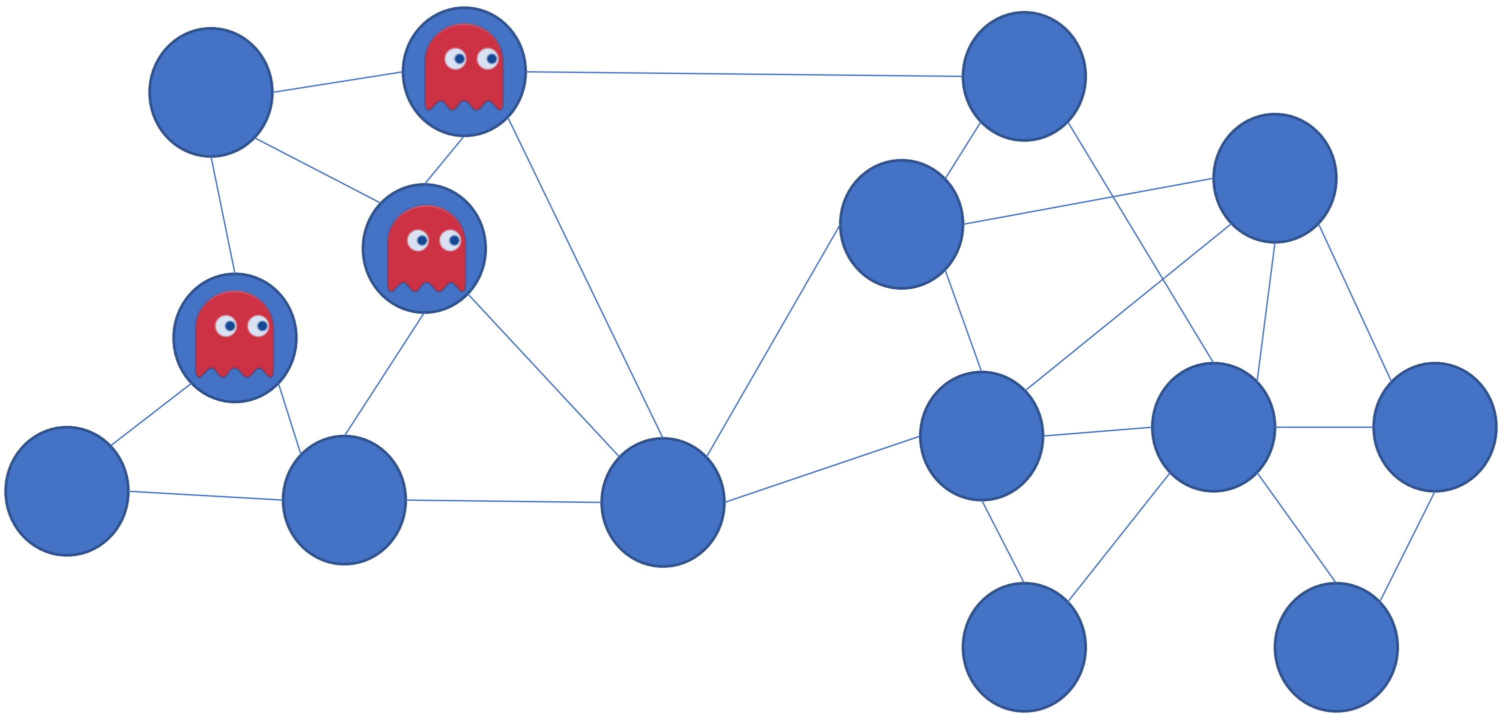
Min-cut vs normalized cut

- Both rely on interpreting images as graphs
- By itself, min-cut gives small isolated pixels
    - But can work if we add other constraints
- min-cut can be solved in polynomial time
    - Dual of max-flow
- N-cut is NP-hard
    - But approximations exist!

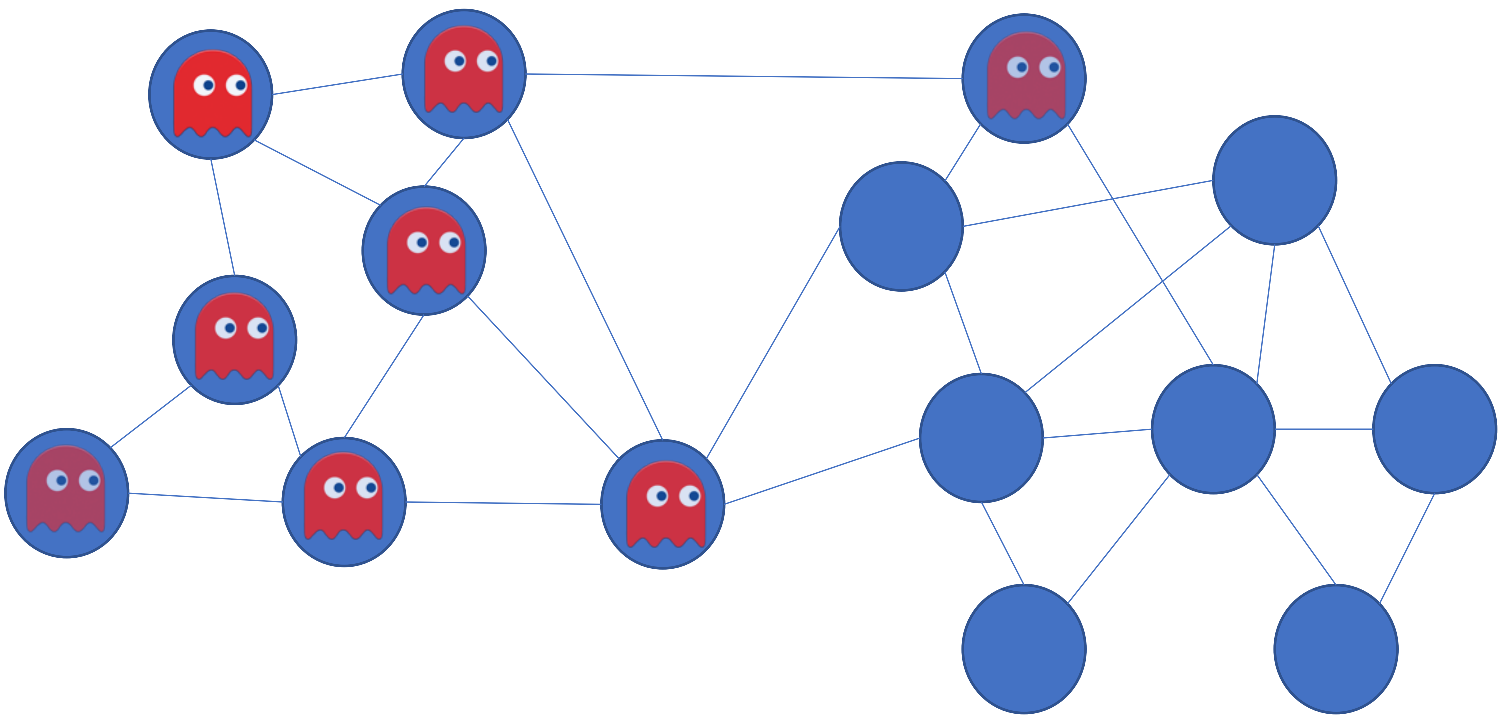
Random walk



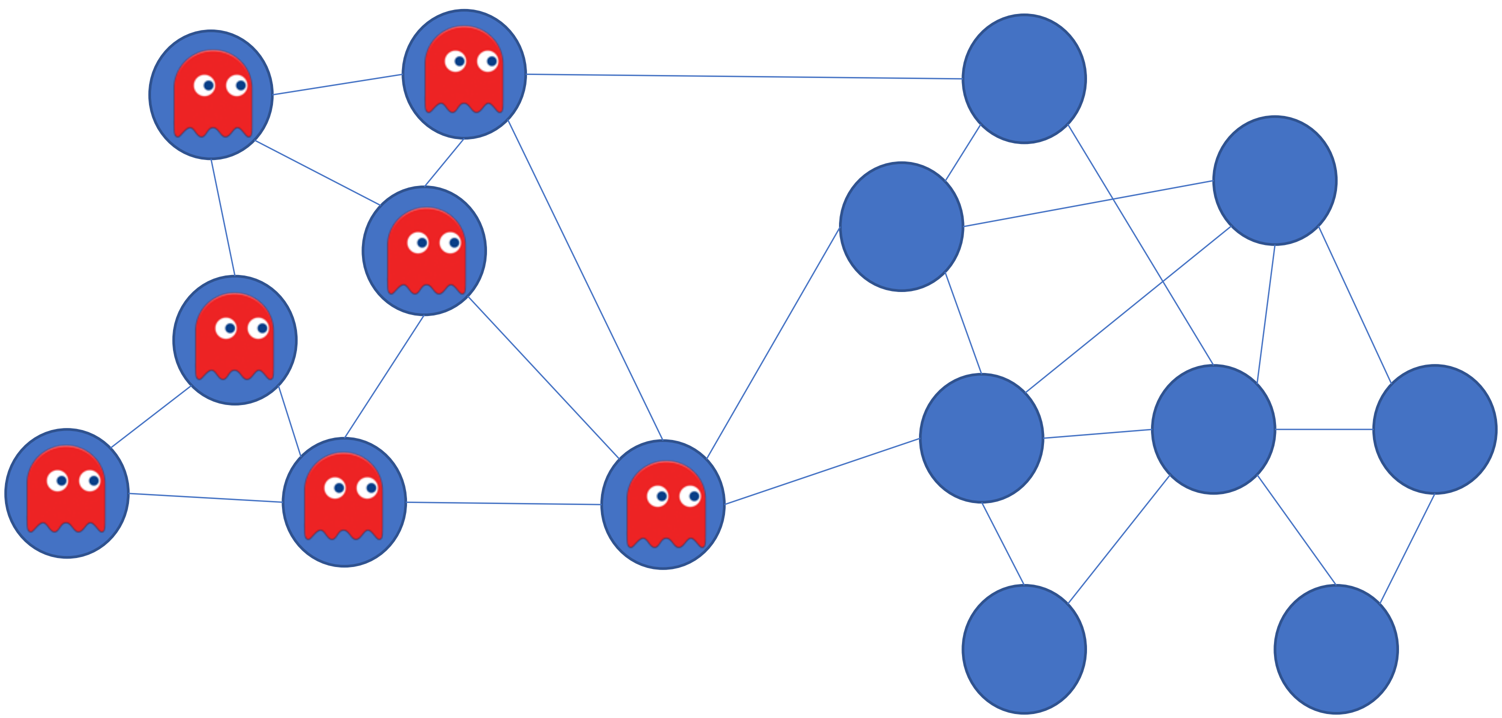
Random walk



Random walk

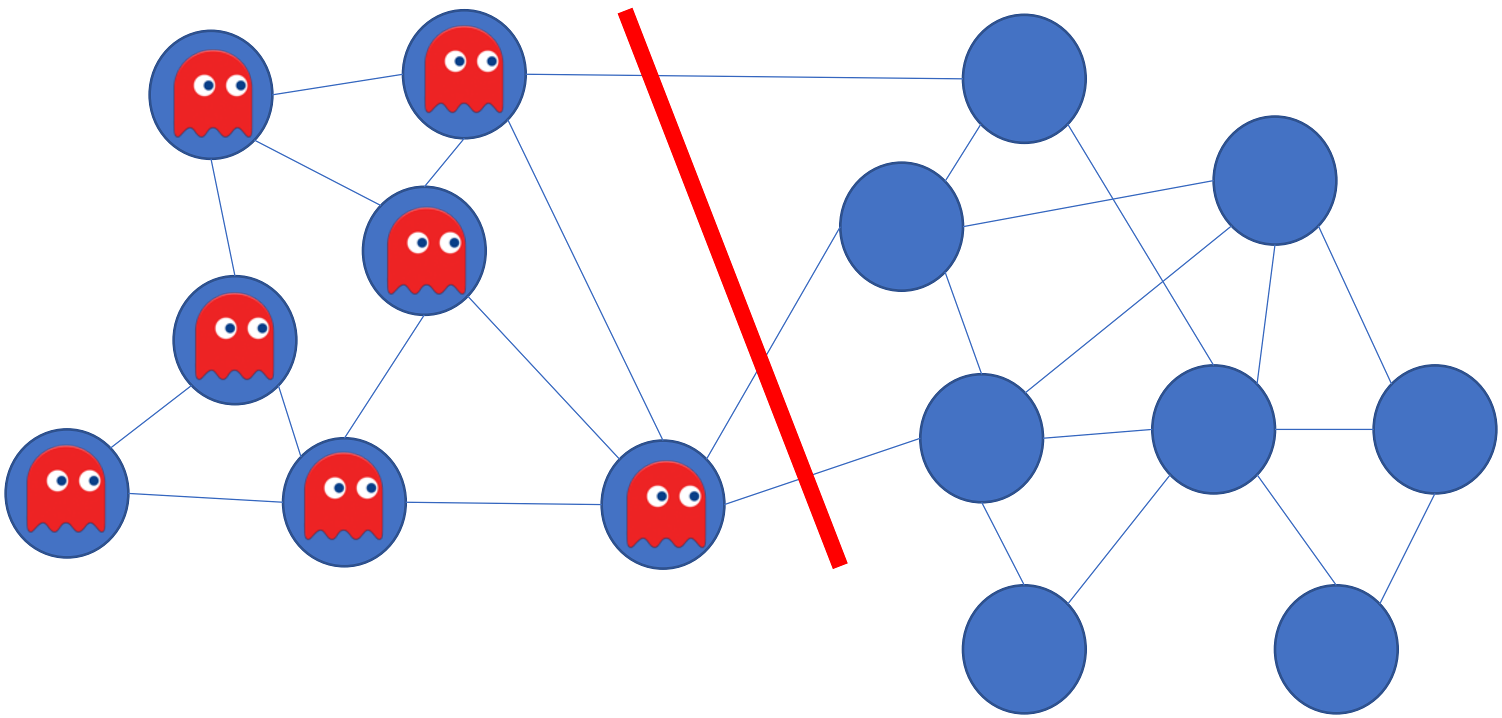


Random walk



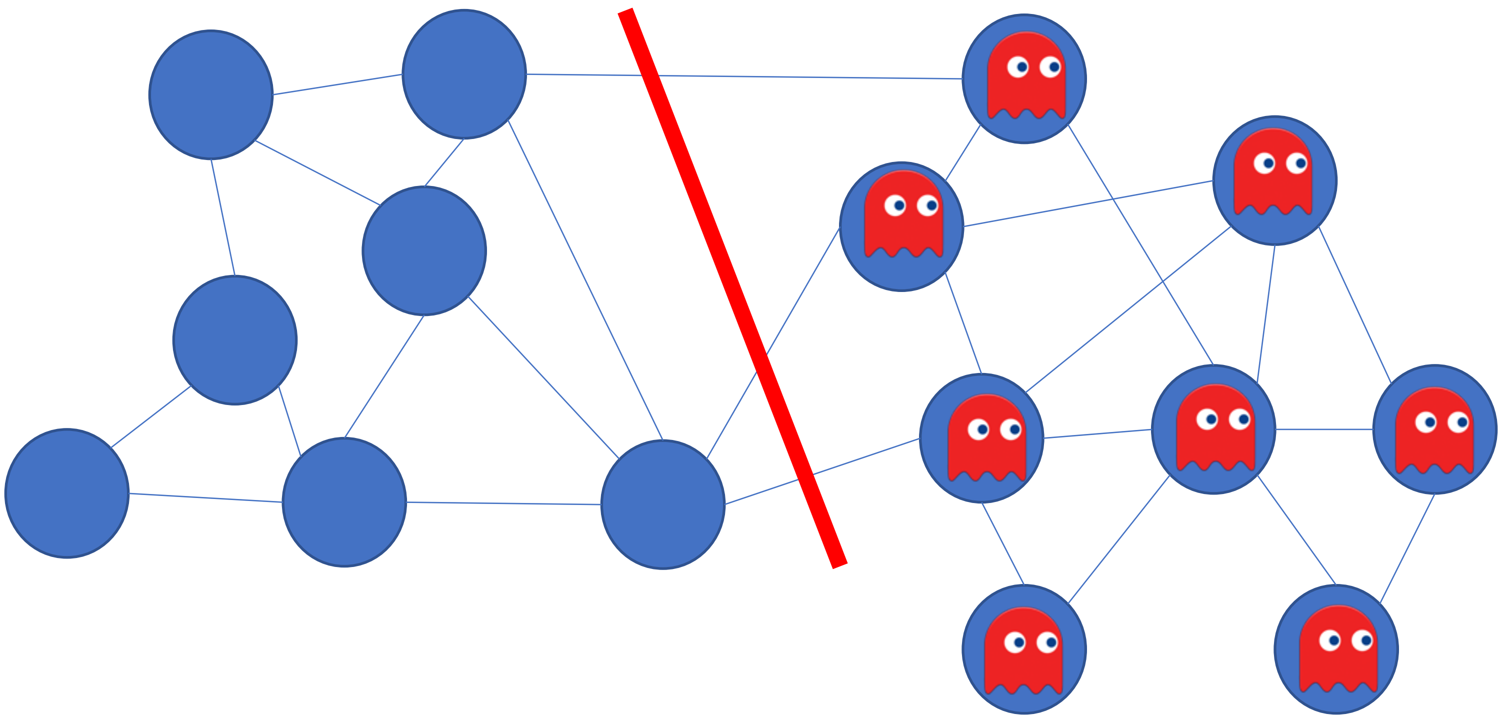
- Given that ghosts inhabit set A, how likely are they to stay in A?

Random walk



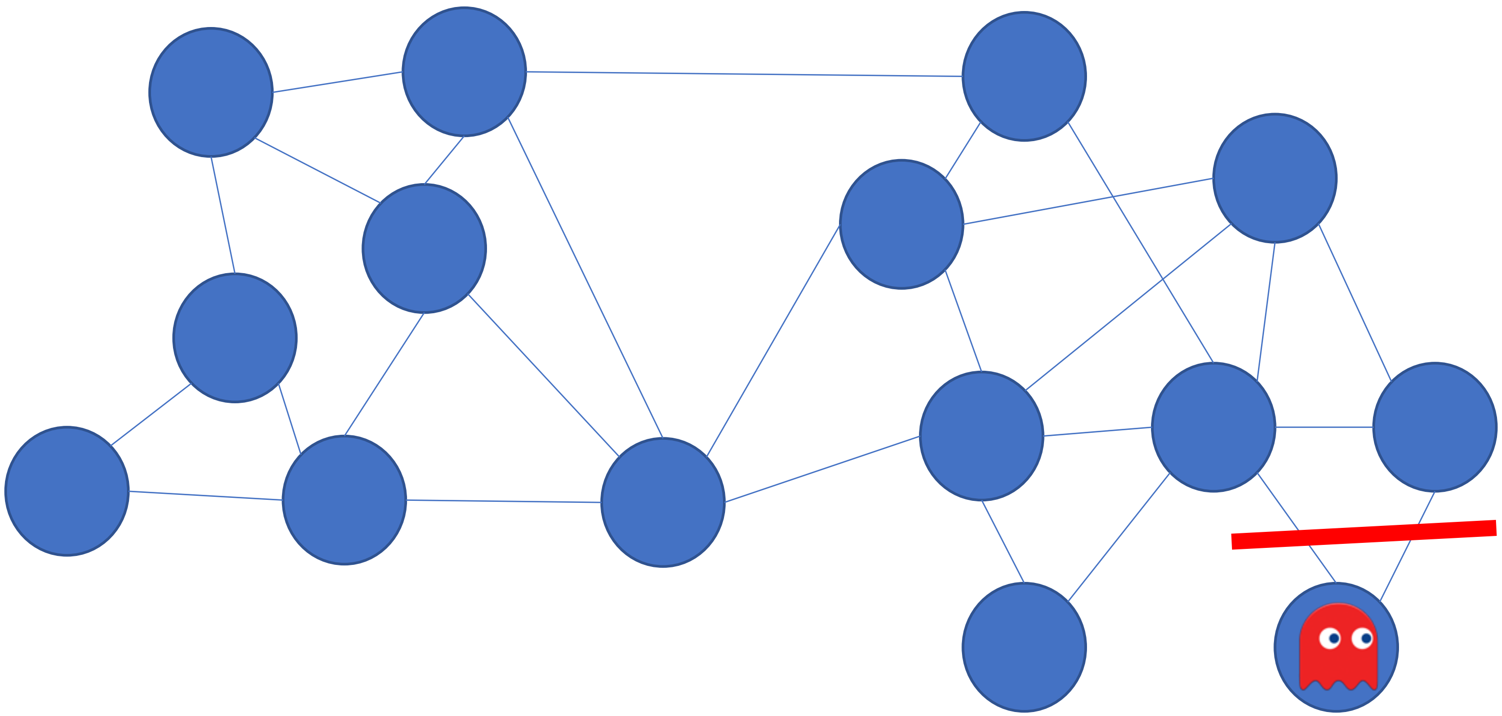
- Given that ghosts inhabit set A, how likely are they to stay in A?

Random walk



- Given that ghosts inhabit set A, how likely are they to stay in A?

Random walk



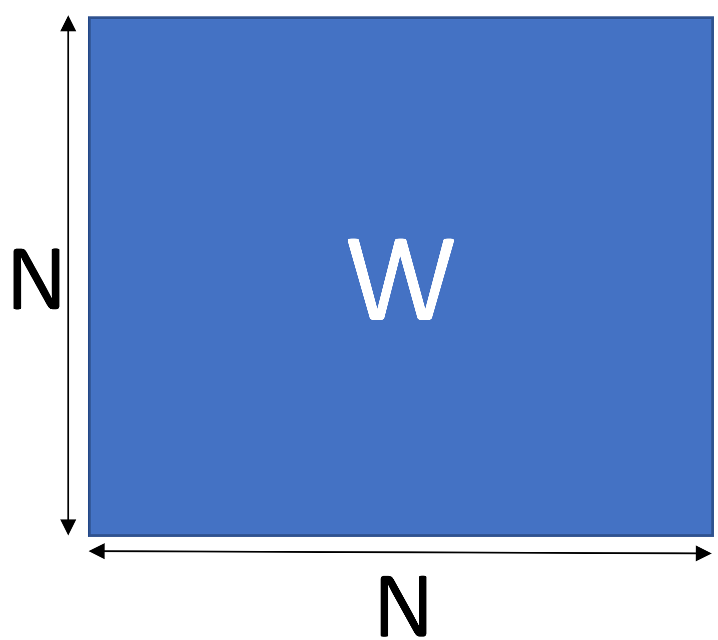
- Given that ghosts inhabit set A, how likely are they to stay in A?

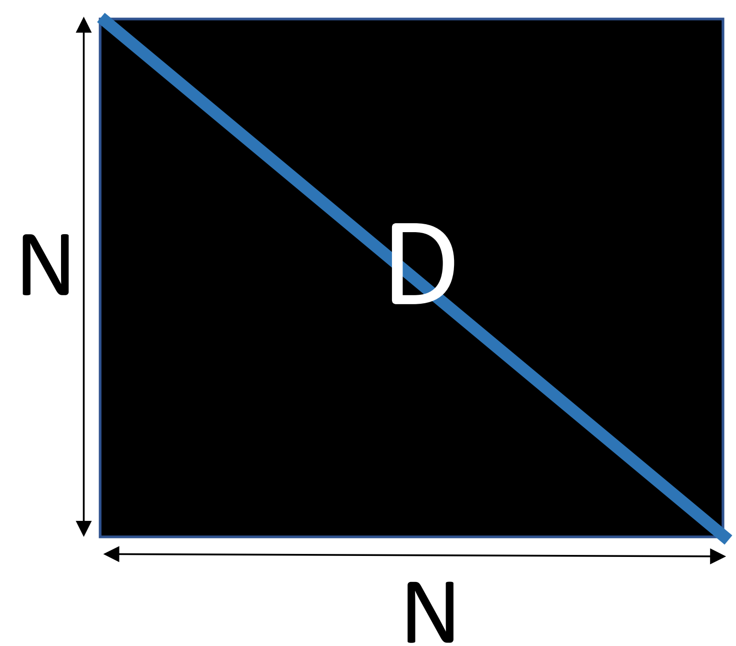
Random walk

- Key idea: Partition should be such that ghost should be likely to stay in one partition
- Normalized cut criterion is the same as this
- But how do we find this partition?

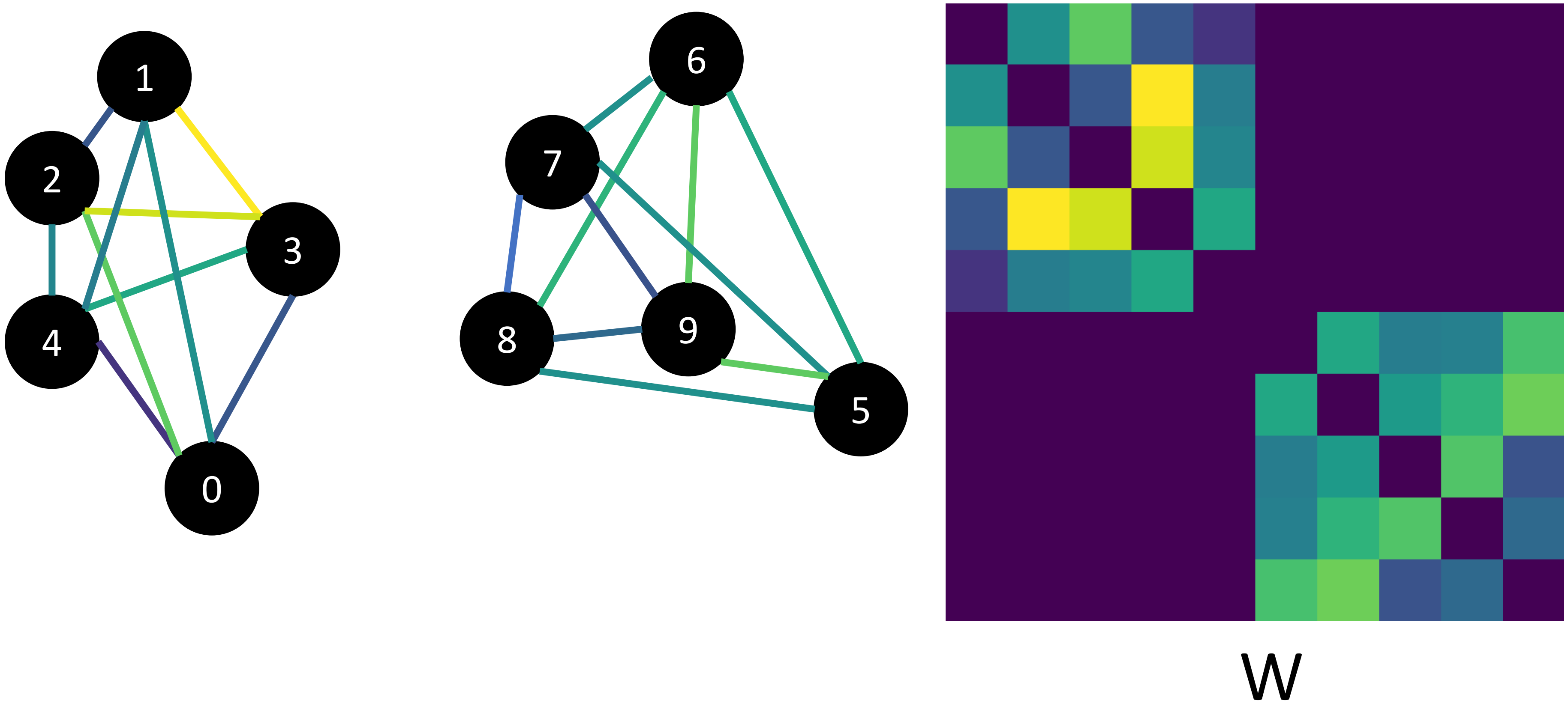
Graphs and matrices

- w(i,j) = weight between i and j (*Affinity matrix)*
- d(i) = degree of i = $\sum_j w(i,j)$
- D = diagonal matrix with d(i) on diagonal

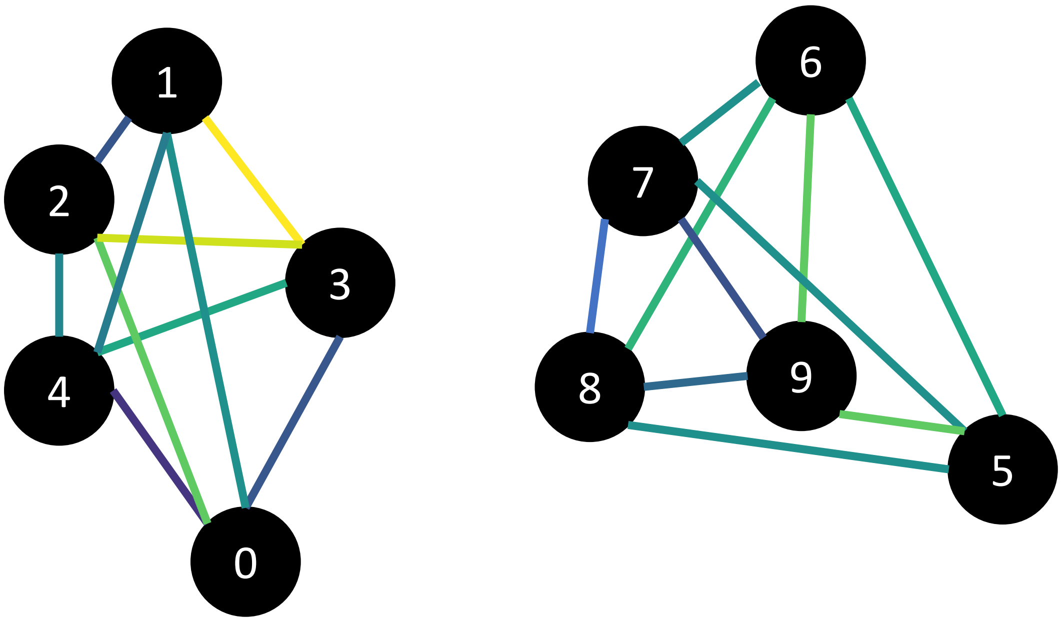




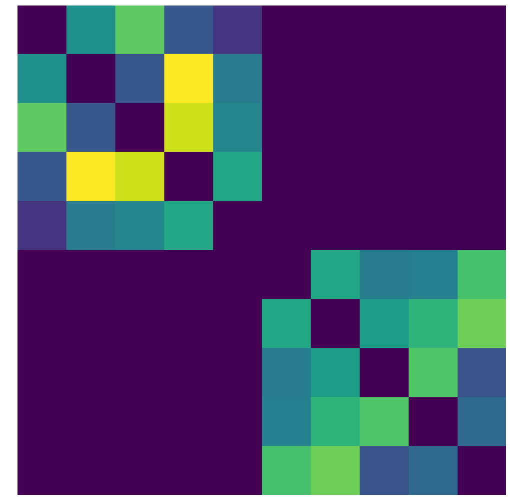
Graphs and matrices



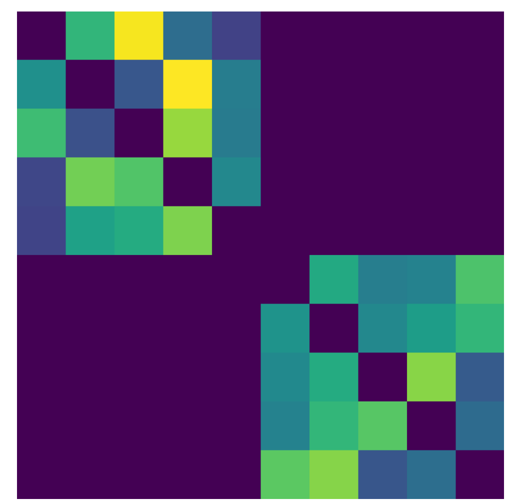
Graphs and matrices



$$E_{ij} = \frac{w_{ij}}{\sum_k w_{ik}}$$



W

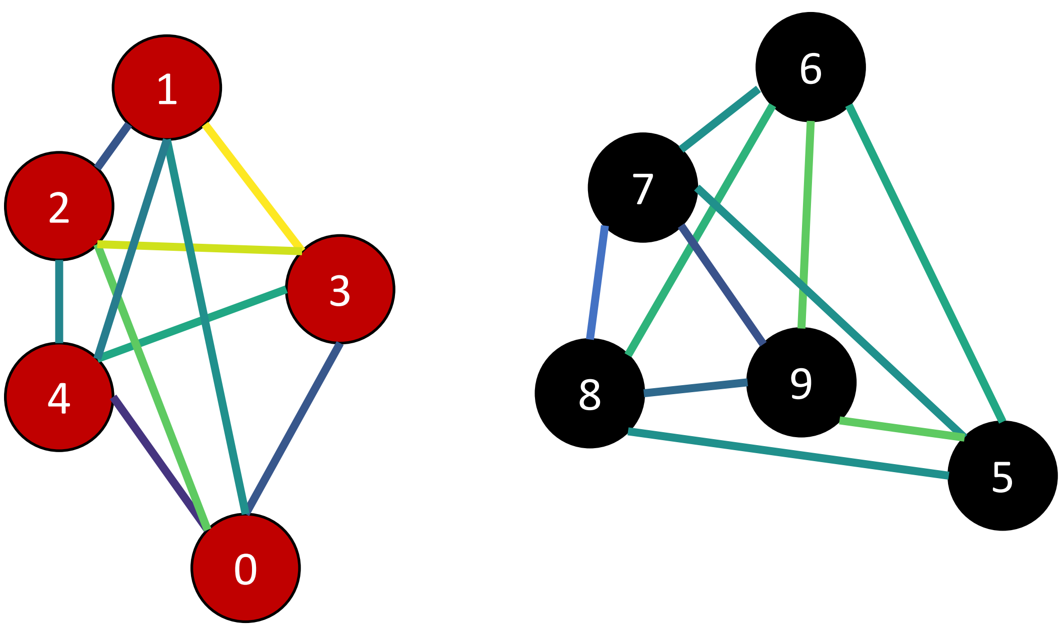


$E = D^{-1}W$

Graphs and matrices

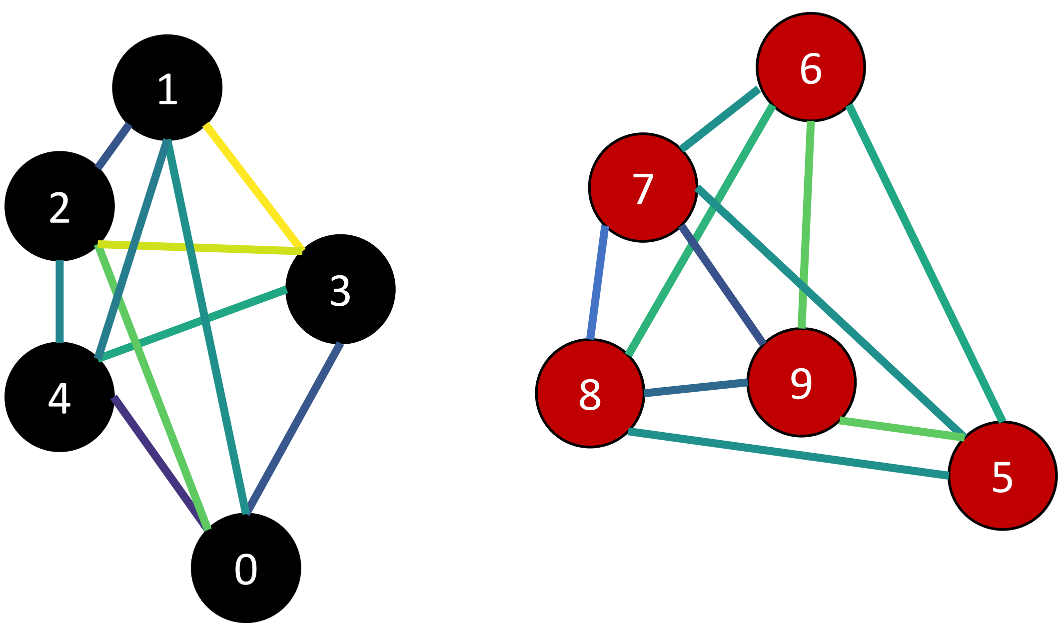
- How do we represent a clustering?
- A label for N nodes
    - 1 if part of cluster A, 0 otherwise
- An N-dimensional vector!

	v_1
0:	1
1:	1
2:	1
3:	1
4:	1
5:	0
6:	0
7:	0
8:	0
9:	0



Graphs and matrices

- How do we represent a clustering?
- A label for N nodes
  - 1 if part of cluster A, 0 otherwise
- An N-dimensional vector!

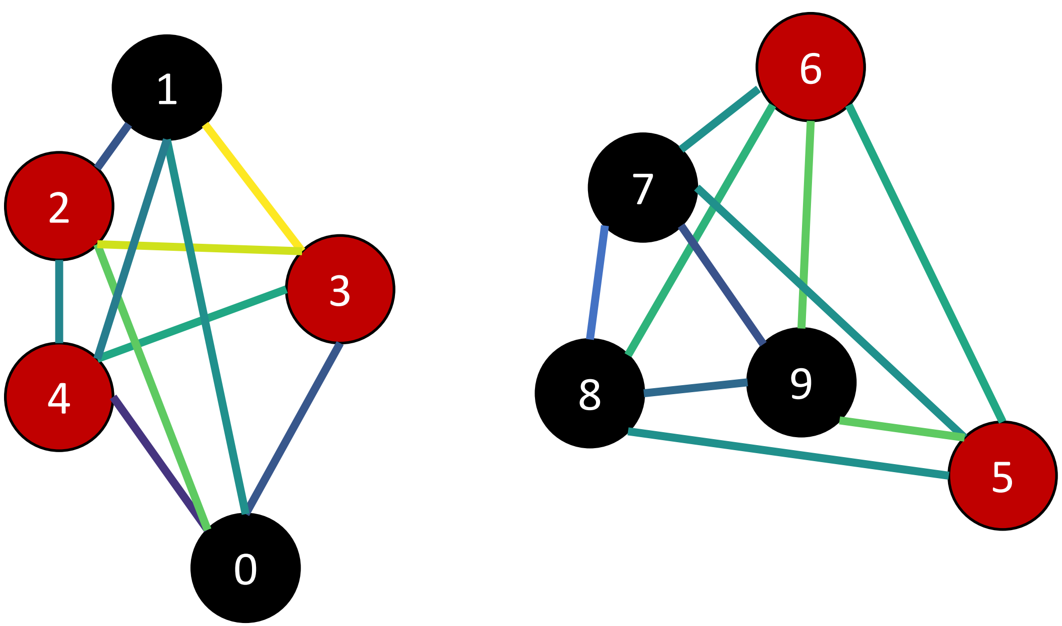


	v_1	v_2
0:	1	0
1:	1	0
2:	1	0
3:	1	0
4:	1	0
5:	0	1
6:	0	1
7:	0	1
8:	0	1
9:	0	1

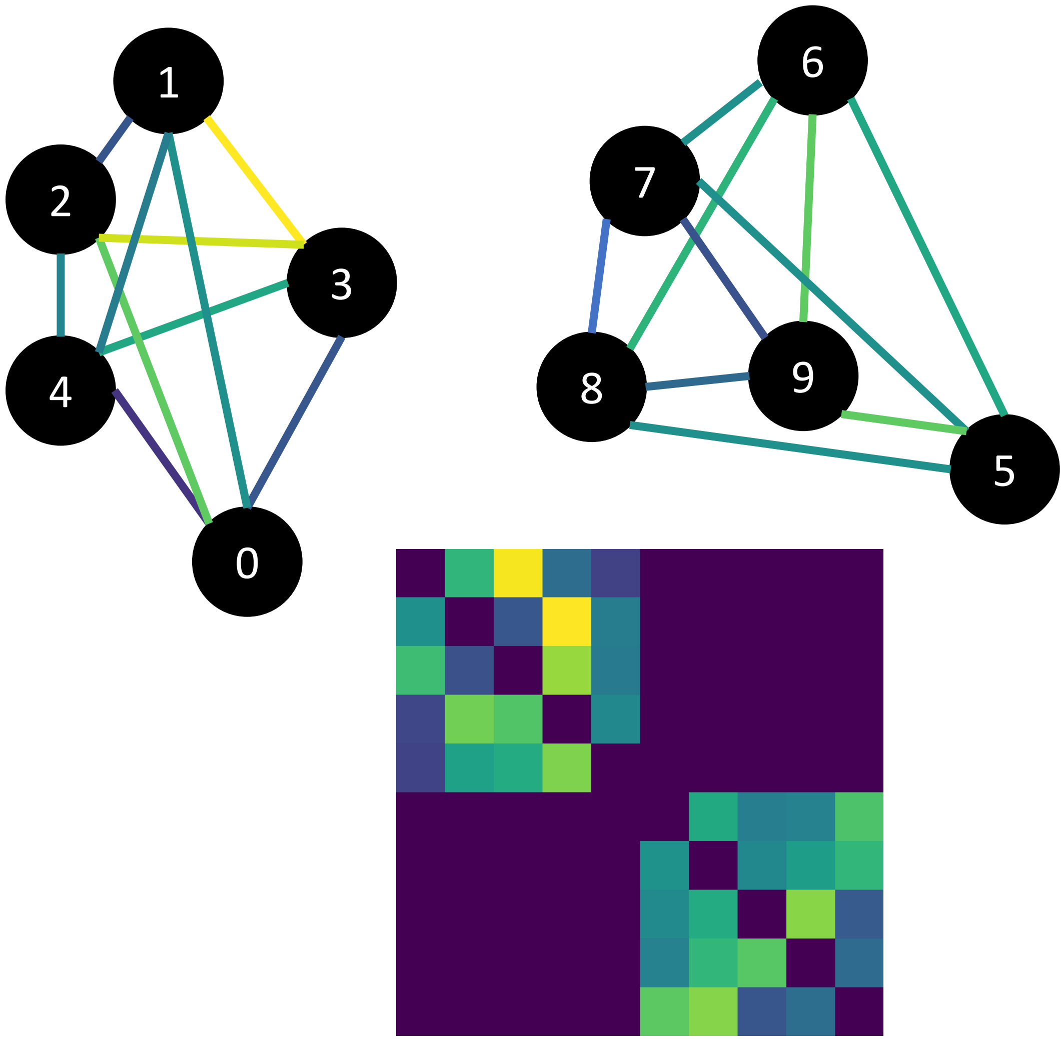
Graphs and matrices

- How do we represent a clustering?
- A label for N nodes
  - 1 if part of cluster A, 0 otherwise
- An N-dimensional vector!

	v_1	v_2	v_3
0:	1	0	0
1:	1	0	0
2:	1	1	1
3:	1	1	1
4:	1	1	1
5:	0	1	1
6:	0	1	1
7:	0	0	0
8:	0	0	0
9:	0	0	0



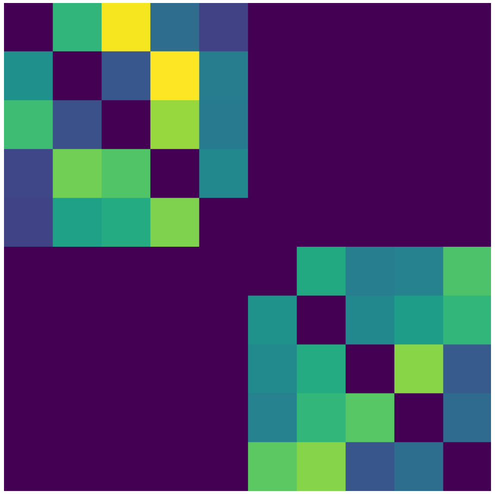
Graphs and matrices



$E = D^{-1}W$

	v_1
0:	1
1:	1
2:	1
3:	1
4:	1
5:	0
6:	0
7:	0
8:	0
9:	0

Graphs and matrices

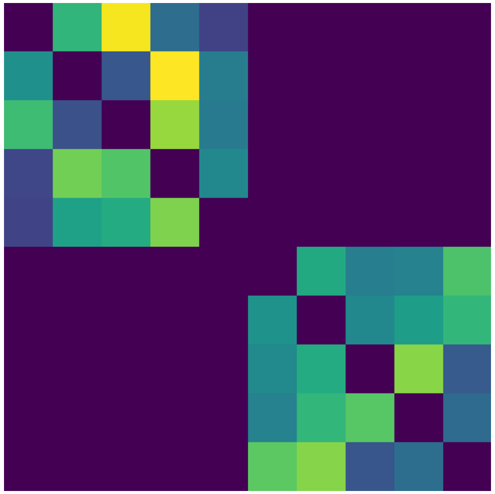


$E = D^{-1}W$

$$E_{ij} = \frac{w_{ij}}{\sum_k w_{ik}}$$

	v_1	Ev_1
0:	1	1
1:	1	1
2:	1	1
3:	1	1
4:	1	1
5:	0	0
6:	0	0
7:	0	0
8:	0	0
9:	0	0

Graphs and matrices

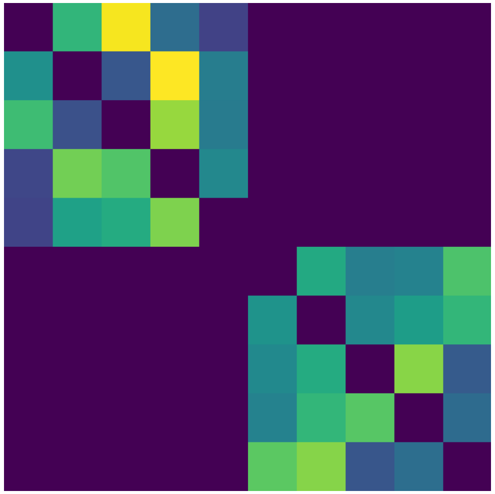


$E = D^{-1}W$

$$E_{ij} = \frac{w_{ij}}{\sum_k w_{ik}}$$

	v_2	Ev_2
0:	0	0
1:	0	0
2:	0	0
3:	0	0
4:	0	0
5:	1	1
6:	1	1
7:	1	1
8:	1	1
9:	1	1

Graphs and matrices

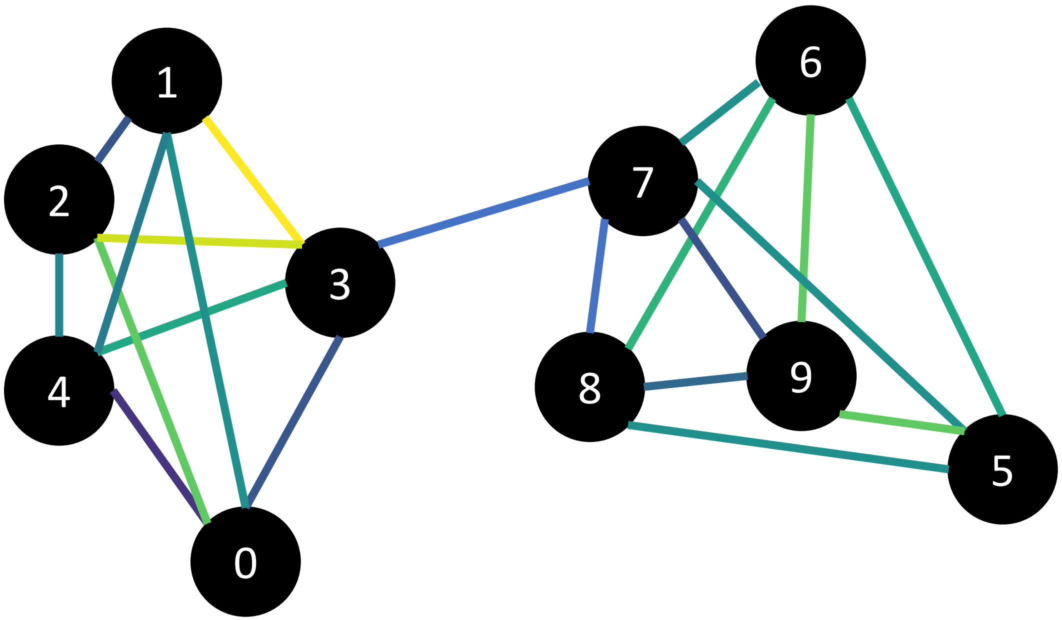


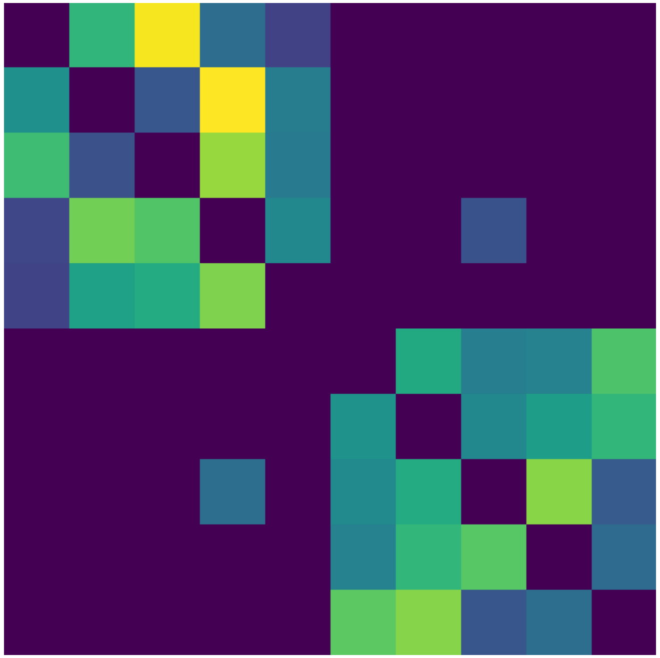
$E = D^{-1}W$

$$E_{ij} = \frac{w_{ij}}{\sum_k w_{ik}}$$

	v_3	Ev_3
0:	0	0.7
1:	0	0.8
2:	1	0.6
3:	1	0.5
4:	1	0.6
5:	1	0.3
6:	1	0.2
7:	0	0.5
8:	0	0.5
9:	0	0.7

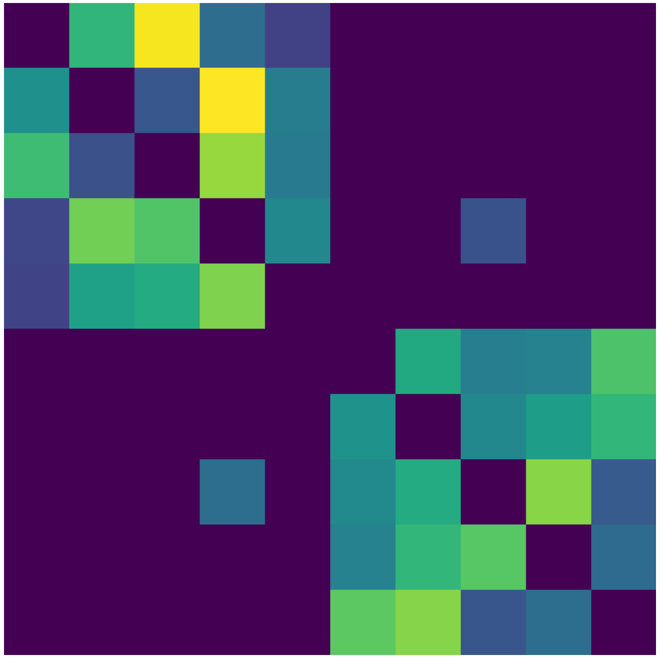
Graphs and matrices





$E = D^{-1}W$

Graphs and matrices



$E = D^{-1}W$

$$E_{ij} = \frac{w_{ij}}{\sum_k w_{ik}}$$

	v_1	**Ev_1**
0:	1	1
1:	1	1
2:	1	1
3:	1	1
4:	1	1
5:	0	0
6:	0	0
7:	0	0.2
8:	0	0
9:	0	0

Graphs and matrices

$$D^{-1}Wy \approx y$$

Define z so that $y = D^{-\frac{1}{2}}z$

$$D^{-1}WD^{-\frac{1}{2}}z \approx D^{-\frac{1}{2}}z$$

$$\Rightarrow D^{-\frac{1}{2}}WD^{-\frac{1}{2}}z \approx z$$

$$\Rightarrow (I - D^{-\frac{1}{2}}WD^{-\frac{1}{2}})z \approx 0$$

Graphs and matrices

$$\Rightarrow (I - D^{-\frac{1}{2}} W D^{-\frac{1}{2}}) z \approx 0$$

$$\Rightarrow \mathcal{L} z \approx 0$$

$$\mathcal{L} = I - D^{-\frac{1}{2}} W D^{-\frac{1}{2}}$$

is called the
Normalized Graph
Laplacian

Graphs and matrices

$$\mathcal{L} = I - D^{-\frac{1}{2}} W D^{-\frac{1}{2}}$$

- We want $\mathcal{L} z \approx 0$
- Trivial solution: all nodes of graph in one cluster, nothing in the other
- To avoid trivial solution, look for the *eigenvector with the second smallest eigenvalue*

$$\mathcal{L} z = \lambda z$$

$$\lambda_1 < \lambda_2 < \ldots < \lambda_N$$

- Find z s.t. $\mathcal{L} z = \lambda_2 z$

Normalized cuts

- Approximate solution to normalized cuts
- Construct matrix W and D
- Construct normalized graph laplacian

$$\mathcal{L} = I - D^{-\frac{1}{2}} W D^{-\frac{1}{2}}$$

- Look for the second smallest eigenvector

$$\mathcal{L}z = \lambda_2 z$$

- Compute $y = D^{-\frac{1}{2}} z$
- *Threshold y to get clusters*
  - Ideally, sweep threshold to get lowest N-cut value

More than 2 clusters

- Given graph, use N-cuts to get 2 clusters
- Each cluster is a graph
  - Re-run N-cuts on each graph

Normalized cuts

- NP Hard
- But approximation using *eigenvector of normalized graph laplacian*
  - Smallest eigenvector : trivial solution
  - *Second smallest eigenvector: good partition*
  - *Other eigenvectors: other partitions*

- An instance of "Spectral clustering"
  - Spectrum = set of eigenvalues
  - Spectral clustering = clustering using eigenvectors of (various versions of) graph laplacian

Images as graphs

- Each pixel is a node
- What is the edge weight between two nodes / pixels?
  - F(i): intensity / color of pixel i
  - X(i): position of pixel i

$$w_{ij} = e^{\frac{-\|\boldsymbol{F}(i)-\boldsymbol{F}(j)\|_2^2}{\sigma_I}} * \begin{cases} e^{\frac{-\|\boldsymbol{X}(i)-\boldsymbol{X}(j)\|_2^2}{\sigma_X}} & \text{if } \|\boldsymbol{X}(i)-\boldsymbol{X}(j)\|_2 < r \\ 0 & \text{otherwise,} \end{cases}$$

Computational complexity

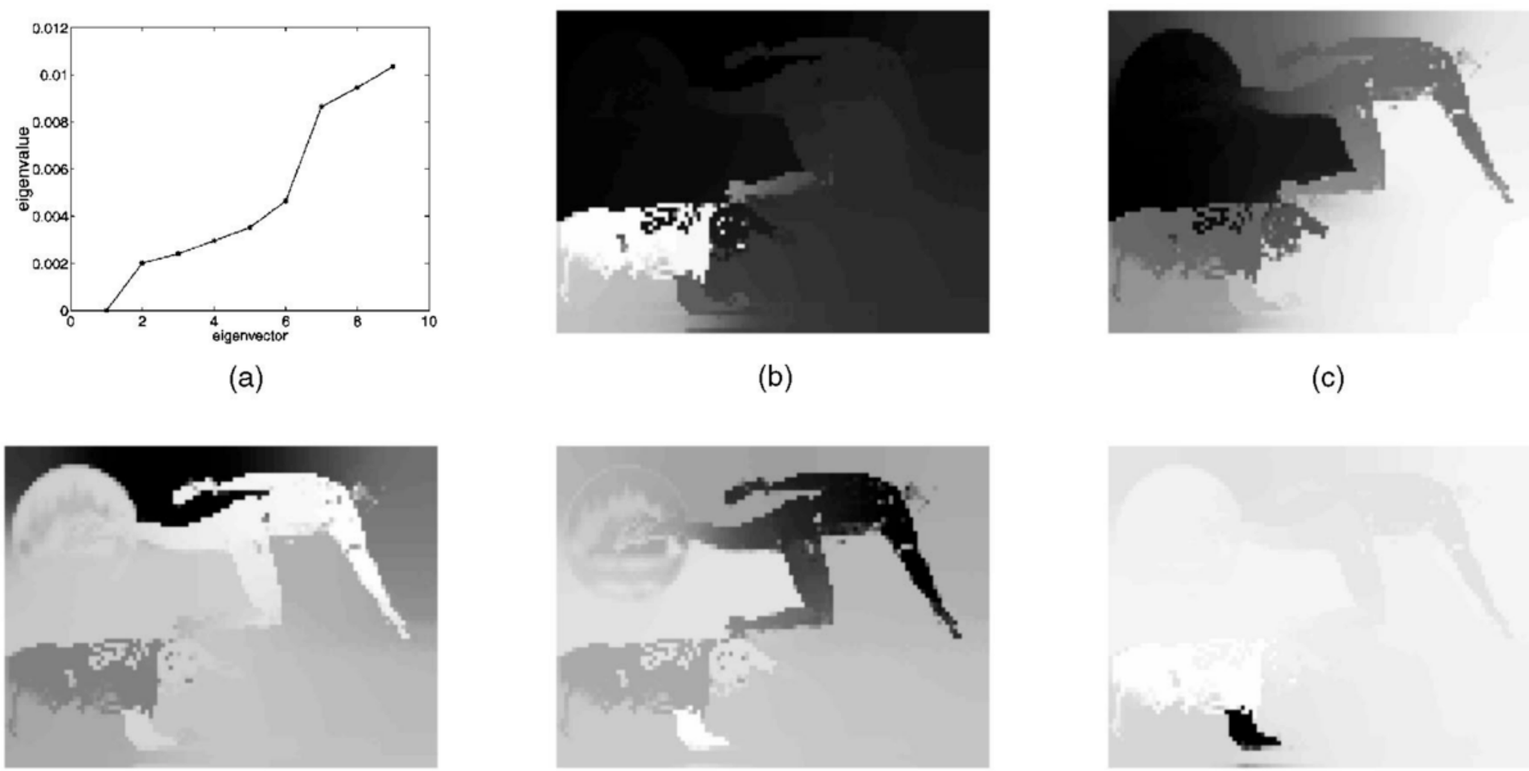
- A 100 x 100 image has 10K pixels
- A graph with 10K pixels has a 10K x 10K affinity matrix
- Eigenvalue computation of an N x N matrix is $O(N^3)$
- Very very expensive!

Eigenvectors of images

- The eigenvector has as many components as pixels in the image

Eigenvectors of images

- The eigenvector has as many components as pixels in the image



(a) (b) (c)

(d) (e) (f)

Another example

2nd eigenvector

3rd eigenvector

4th eigenvector

Recursive N-cuts

2^{nd} eigenvector

First partition

2^{nd} eigenvector of 1^{st} subgraph

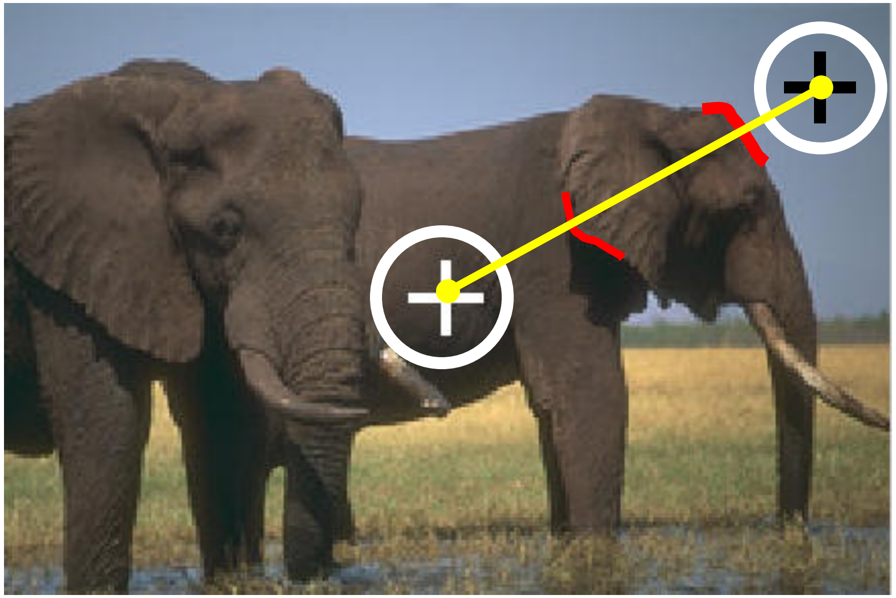
recursive partition

N-Cuts resources

- http://scikit-learn.org/stable/modules/clustering.html#spectral-clustering
- https://people.eecs.berkeley.edu/~malik/papers/SM-ncut.pdf

Images as graphs

- Enhancement: edge between far away pixel, weight = 1 – magnitude of *intervening contour*



Eigenvectors of images