Lecture 4: Segmentation

Segmentation problems ask us to classify pixels/regions of an image

Fg/Bg

Semantic

Instance

Challenges:

- Lots of decisions
- Local labels, global information
- Consistency is hard

Today:

- Foreground seg. example
- Min-cut as a tool to solve Fg/bg segmentation
- Spanning trees as a tool

Foreground - Background Segmentation

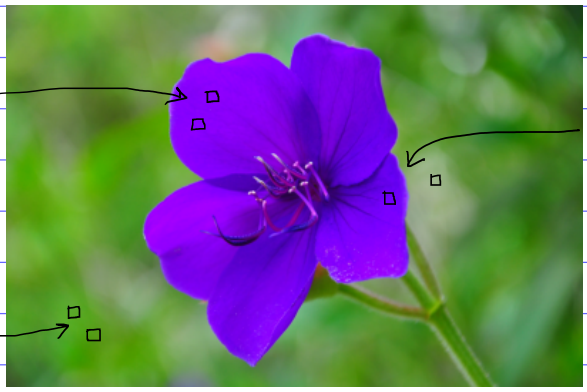
Goal: Find all pixels in the "foreground" of an image

- How do we know what counts as fg?
  → We need external info

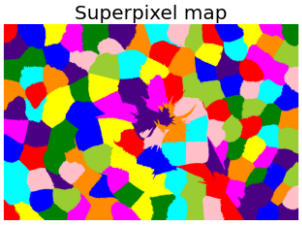
- How do we measure correctness?
  → We need ground truth OR qualitative assessment

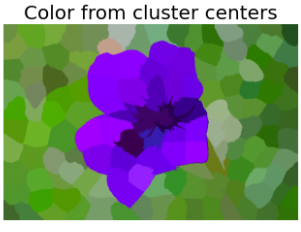
- How do we make the problem easier?
  → Superpixels

Proposition: Pixels that are both visually similar and near each other will almost always belong to the same segment.



We can make segmentation easier by grouping pixels into superpixels





Why does this work?

→ Segmentation = Local + Global

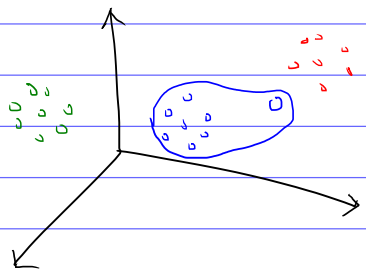
Superpixel

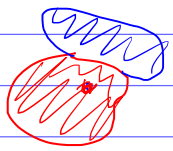
How do we find superpixels?

→ Lots of algorithms! SLIC is very popular for images, Uniform Entropy Slicing is popular for video (and is from UM!)

Simple (and crude) superpixelization alg:

- For each pixel (x, y) in I, make a vector $[x, y, \lambda R, \lambda G, \lambda B]$ for some $\lambda > 0$.
- Treat each vector as a data point in $\mathbb{R}^5$ and cluster them.





- Problem: No way to guarantee contiguous segments

  → Postprocessing

- Now that the problem is easier, how do we solve it?

→ Construct a loss function & minimize

Define:

known

- $S = \{s_1, \ldots, s_N\}$ the set of superpixels
  $i \sim j$ means s_i & s_j are neighbors
- v_i = feature vector (color, location, shape)
- $f(v_i)$ = penalty for v_i belonging to the fg (↑ for non-fg, ↓ for fg)
- $b(v_i)$ = penalty for $v_i \in$ bg
- $p(v_i, v_j)$ = penalty for putting one (i, j) in fg & the other in bg.
- x_i = decision variable
  $x_i = 1$ if $i \in$ fg $x_i = 0$ if $i \in$ bg

Then, for any suggested solution X:

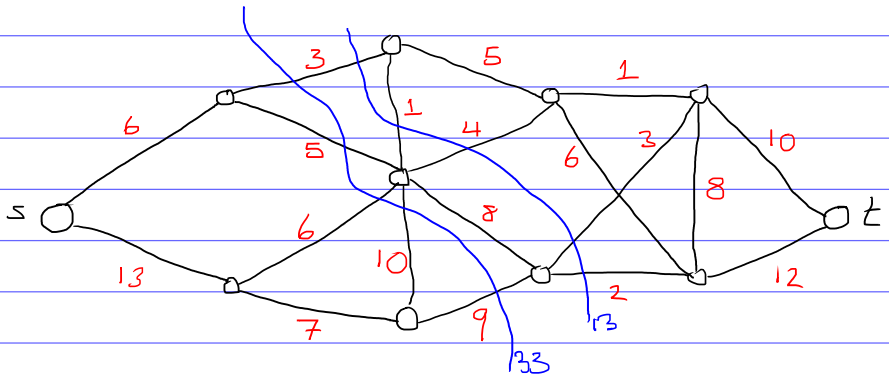
$$L(X) = \sum_{x_i = 1} f(v_i) + \sum_{x_i = 0} b(v_i) + \sum_{\substack{i \sim j \\ x_i \neq x_j}} p(v_i, v_j)$$

→ We want to find the X that minimizes $L(X)$.

Approach: Graph cuts

Min-cut Problem: Overview

- Graph $G(V, E)$ has edge penalties c_e for all $e \in E$. Two special vertices, s = source, t = sink

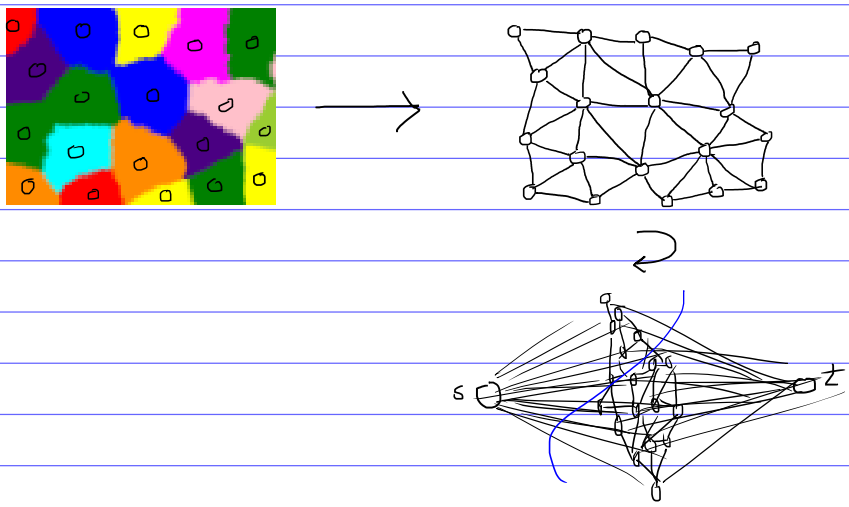


- A cut $C \subseteq E$ is a set of edges that:
  - If remove C from G, there are no paths from $s \leftrightarrow t$
- The cost of a cut is $\sum_{e \in C} c_e$
- Min-cut problem: Find C^*, the cut with minimum possible cost.

(Dual problem: Max-Flow)

Graph cut segmentation

- Each superpixel is a vertex
- Neighboring superpixels are connected
- Source / sink are connected to every other vertex



- What does a cut C look like in this graph?

$$C = \{\underbrace{(s,i)}_{\text{Connect source to sp. vertex}} \dots \underbrace{(i,j)}_{\text{Connect sp} \leftrightarrow \text{sp}} \dots \underbrace{(i,t)}_{\text{Connect sp} \leftrightarrow \text{sink}}\}$$

- Then the cost of C is:

$$\text{cost}(C) = \sum_{(s,i) \in C} C_{si} + \sum_{(i,j) \in C} C_{ij} + \sum_{(i,t) \in C} C_{it}$$

$$L(X) = \sum_{x_i = 1} f(v_i) + \sum_{x_i = 0} b(v_i) + \sum_{\substack{x_i \neq x_j \\ i \sim j}} p(v_i, v_j)$$

$$\text{cost}(C) = \sum_{(i,t) \in C} c_{it} + \sum_{(s,i) \in C} c_{si} + \sum_{(i,j) \in C} c_{ij}$$

→ We can make these equivalent if:

$x_i = 1 \Leftrightarrow (i,t) \in C \qquad f(v_i) = c_{it}$

$x_i = 0 \Leftrightarrow (s,i) \in C \qquad b(v_i) = c_{si}$

$i \sim j, x_i \neq x_j \Leftrightarrow (i,j) \in C \qquad p(v_i, v_j) = c_{ij}$

$$\text{cost}(C) = L(X)$$

By constructing our network with these specific costs, solving the min cut solves the segmentation

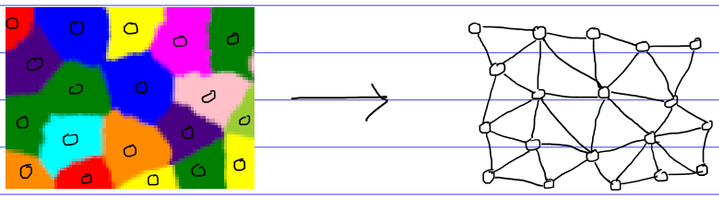
What are f(), b(), and p(,)?

→ Whatever we want!

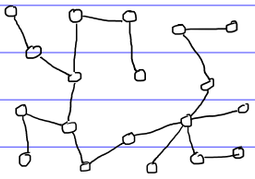
(HW 1.4 has an example problem where we choose f(), b(), & p(,) concretely)

Segmentation by Minimum Spanning Forest:

Recall graph (no source/sink):



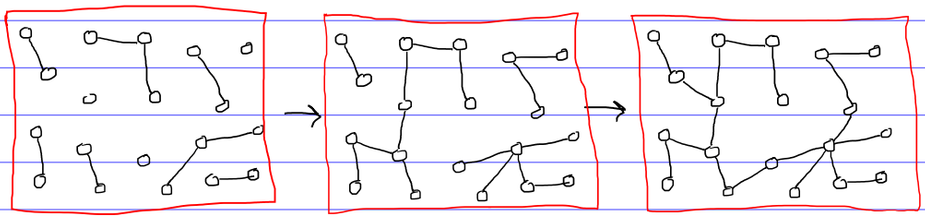
A spanning tree is an acyclic, connected subgraph containing each vertex.



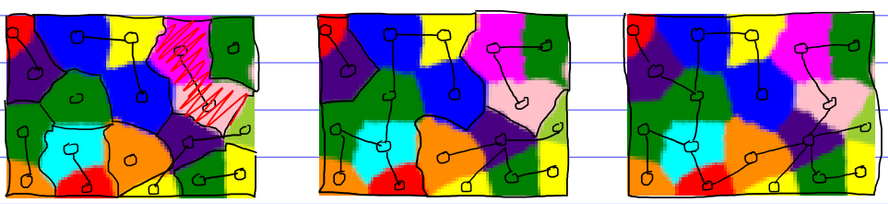
When edges are weighted, the spanning tree of minimum total weight, the Minimum Spanning Tree (MST) can be found in polynomial time:

Kruskal's Algorithm:
- Sort edges by weight $c_{e_1} < c_{e_2} < \ldots < c_{e_N}$
- Initialize tree T with every vertex, no edges
- For $i = 1, 2, \ldots, N$:
  - If e_i connects two vertices that are not already reachable in T:
    - Add e_i to T

Key insight: At every step of the algorithm, we have a minimum spanning forest (MSF):



Each of these can represent a segmentation, by merging connected superpixels:



10 segments | 4 segments | 1 segment

Useful segmentations | Not

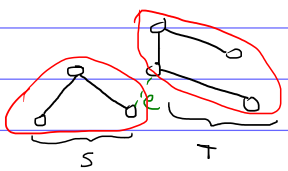
If we stop Kruskal's Alg. early, we can return a segmentation of the image

Stopping earlier ≡ Smaller segments

Modified Kruskal's Algorithm:

- Sort edges by weight $c_{e_1} < c_{e_2} < \ldots < c_{e_N}$
- Initialize tree T with every vertex, no edges
- For $i = 1, 2, \ldots, N$:
  - If e_i connects two vertices that are not already reachable in T:
    - If $g(e_i) <$ threshold:
      - Add e_i to T

- Stopping function $g(\cdot)$ can be whatever we want. A published example:



$$g(e) = \min\left(\text{maxweight}(S) + \frac{k}{|S|}, \text{maxweight}(T) + \frac{k}{|T|}\right) - c_e$$

(k: constant)

(Felzenszwalb-Huttenlocher Algorithm)