L26 -- Graph Sparsification
[Jeff Phillips - Utah - Data Mining]

Large graph
  $G = (V,E)$

Might be slow to handle if $|V|$ large and $|E| = |V|^{1+c}$
want:
$H = (V,E')$ close to G
and
$|E'| \approx |V| \log |V|$

Technique 1:

degree of vertex $v_i = d_i$
Sample each edge (i,j) w.p.
  $p_{ij} = \min\{1, t/\min\{d_i, d_j\}\}$
  re-weight sampled edged, inverse to probability chosen
    or with same weight if chosen w.p. 1

Keep all edges of nodes with degree at most t
All other edges keep proportional to t/d_i for min degree endpoint

$E[|E|] < t*|V|$

Set $t = (1/\mathrm{eps}^2) \log n$

--> Preserves "cut" within eps
     Useful in Spectral Clustering
               Finding Communities

Laplacian
$L_G = D_G - A_G$
 $D_G = \mathrm{diag}(d_1, d_2, \ldots, d_{|V|})$
 A_G = adjacency matrix

Want sparse graph H s.t.

$||L_G - L_H||_2 \leq \mathrm{eps}$

$(1-\mathrm{eps})\, x^T L_G x \leq x^T L_H x \leq (1+\mathrm{eps})\, x^T L_G x$ forall x in R^n

(Technique 1 only works for x in $[0,1]^{|V|}$)

```
Technique 2:
*** Effective Resistance ***

R_eff(e) is effective resistance between end points e = (u,v)

(u,v) (u,a) (a,v)  all strength 1
R_eff(u,v) = 1 / (1/2 + 1/1) = 2/3

Sample edges w.p. p_e ~~ "proportional to" R_eff(e)
Weight edge as 1/p_e
-->  Take O((1/eps^2) n log n) edges (with replacement, add weights)

Analysis very similar to column sampling (L14).

Recent papers (2011) improve runtime to about O(|V| log |V| log(1/eps))
idea:  construct rough approx H_1
    remove degree 1,2 nodes -> G_2  (contract edges)
    construct rough approx H_2
    remove degree 1,2 nodes -> G_3
    ... log n rounds
can be done faster with series of subtle but simple tricks

***********
Currently, these are not quite practical.  But expect to be practical in next
5 years?  May lead to many very useful techniques...
...but worry about the (1/eps^2) factor
***********


Approach 2:
---------------------------------------------------------
Spanners

Start with metric d_G(a,b)  for all a,b in V
  often: d_G(a,b) = shortest path in Euclidean graph
         a,b in R^d   (for small d   e.g. d=2,3)
               (can be low doubling-dimension)
  sometimes G is complete graph (all edges)

G = (V,E)
 if (a,b) in E, then d_G(a,b) = ||a-b||
 else (shortest path) = best combination

t-spanner H if
  for all 1 <= d_H(a,b) / d_G(a,b) <= 1+t
```

```
measure(H):
 + # edges
 + total weight
 + maximum degree
(we want each of these things to be small)

Algorithms:
  + Greedy:  start no edges.  Sort pairs be distance (small -> large)
             If error > 1+t -->  add edge
             (works ok, hard to say much about measure)
  + Cone Based:  around each point, divide space into k > 6 cones.
                 Each cone defines set of directions.  Find closest point +
connect
                 angle = 2pi/k  ->  t <= 1/(1-sin(angle/2))
  + WSPD:  Set of pairs {(A,B)} s.t. A, B subset V
                                     each (a,b) in exactly one pair
           min_{a in A, b in B} d(a,b) >
               s*max{max_{a1,a2 in A} d(a1,a2), max_{b1,b2 in B} d(b1,b2)

        Compute with (compressed) Quad Tree:
          split node -> 4  (TL,TR,BL,BR)
            for all A,B in (TL,TR,BL,BR)
            if A,B s-WS -> into pairs
            else  check all pairs in  split(A)  vs.  split(B)

        -->  size O(s^d |V|) and computed in O(|V| log |V| + s^d |V|)
        -->  each pair forms the edge of a spanner
```