Social-Network Graphs
Last Time ...

- Social Network Graphs
- Betweenness
  - Girvan-Newman Algorithm
- Graph Laplacian
  - Spectral Bisection
  - $\lambda_2, \psi_2$
- Eigenvalue problems
Projects

- Yelp data challenge

- Global Disease Monitoring and Forecasting with Wikipedia
  - Recent paper, PLoS Nov '14
Direct discovery of communities

- Although partitioning the graph using betweenness is effective, it has some drawbacks
  - Not possible to place an individual in two different communities
  - Everyone is assigned a community

- Alternatively, discover communities by looking for subsets of the nodes that have a relatively large number of edges among them
  - Finding cliques $\rightarrow$ NP Complete
  - Easier to find complete bipartite subgraphs
  - Counting triangles
Why count triangles

- Clustering coefficient:

given an undirected graph $G = (V, E)$

$cc(v) =$ fraction of $v$’s neighbors who are neighbors themselves

$$cc(v) = \frac{|\{(u, w) \in E \mid u \in N(v) \land w \in N(v)\}|}{\binom{d(v)}{2}}$$

number of $\Delta$s incident on $v$

- $cc(\text{purple}) = N/A$
- $cc(\text{red}) = 1/3$
- $cc(\text{orange}) = 1$
- $cc(\text{gray}) = 1$
Why clustering coefficients?

Captures how tight-knit the network is around a node

Network Cohesion
- Tightly knit communities foster more trust, social norms
How to count triangles

Sequential

\[
\sum_{v \in V} d_v^2
\]

Even for sparse graphs can be quadratic if one vertex has high degree
Parallel Version

Parallelize the edge checking phase

- **Map 1**: foreach $v$ generate $(v, N(v))$
- **Reduce 1**: Input $(v, N(v))$
  
  Output: all 2 paths $((v_1, v_2), u)$ where $v_1, v_2 \in N(u)$

- **Map 2**: generate $((v_1, v_2), u)$ and $((v_1, v_2), \phi)$ for $(v_1, v_2) \in E$
- **Reduce 2**: input $((v_1, v_2), u_1, u_2, ..., u_k, \phi)$
  
  If $\phi$ is part of the input, then increment triangle count by $1/3$
Data skew

- How much parallelism can we achieve?
  - Generate all paths to check in parallel
  - The runtime becomes $\max_{v \in V} d_v^2$

- Naïve parallelization does not help with data skew
  - Some nodes will have very high degree
  - Remember power-log distribution
  - Most reducers will be done quickly
  - A few will take forever
    - Curse of the last reducer
Adapting the algorithm

- Dealing with skew directly
  - Currently each triangle is counted 3 times
  - Running time is quadratic in the degree of the vertex
  - Idea: count each triangle once, by the lowest degree vertex
How to count $\Delta$s better

Sequential version [Shank ‘07]

```plaintext
foreach $v \in V$
  foreach $u, w \in N(v)$
    if $d(u) > d(v) \& d(w) > d(v)$
      if $(u, w) \in E$
        triangles[v]++
```

CS 5965/6965 - Big Data Systems - Fall 2014
Dealing with skew

Why does it help?

- Partition nodes into two groups:
  - Low: $L = \{v : d_v \leq \sqrt{m}\}$
  - High: $H = \{v : d_v > \sqrt{m}\}$
- There are at most $n$ low nodes; each produces at most $m$ paths
- There are at most $2\sqrt{m}$ high nodes
- These two are identical
- no mapper can produce substantially more work than others
- Total work is $O(m^{3/2})$, which is optimal
Triangles with all $\mathcal{H}$ nodes

- There are only $\mathcal{O}(\sqrt{m})$ $\mathcal{H}$ nodes
- Therefore, there are at most $\mathcal{O}(m^{3/2})$ triangles with all $\mathcal{H}$ nodes
- Using $E$, check if these triangles exist in $\mathcal{O}(1)$ time
- Total time – $\mathcal{O}(m^{3/2})$
Triangles with all at least 1 \( \mathcal{L} \) node

- Consider all \( m \) edges \(-\mathcal{O}(m)\)
  - Given an edge \((v_1, v_2)\)
    - Ignore if \( v_1, v_2 \in \mathcal{H} \)
    - Consider the smaller node, say \( v_1 < v_2 \)
      - This node has \( k \) nodes in its adjacency list, with \( k < \sqrt{m} \) \(-\mathcal{O}(\sqrt{m})\)
      - Count triangle with node \( u_i \) iff edge \((v_2, u_i)\) exists and \( v_1 < u_i \)

\( \mathcal{O}(m^{3/2}) \)