

13: Betweenness Centrality

Machine Learning and Real-world Data (MLRD)

Simone Teufel

Last session: some simple network statistics

- You measured the **degree** of each node and the **diameter** of the network.
- Next two sessions:
 - Today: finding **gatekeeper** nodes via **betweenness centrality**.
 - Next session: using betweenness centrality of edges to split graph into **cliques**.
- Reading for social networks (all sessions):
 - Easley and Kleinberg for background: Chapters 1, 2, 3 and first part of Chapter 20.
 - Brandes algorithm: two papers by Brandes (links in practical notes).

Intuition behind clique finding

- Certain nodes/edges are most crucial in linking densely connected regions of the graph: informally **gatekeepers**.
- Cutting those edges isolates the cliques/clusters.

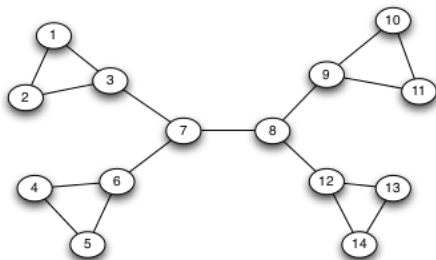


Figure 3-14a from Easley and Kleinberg (2010)

Intuition behind clique finding

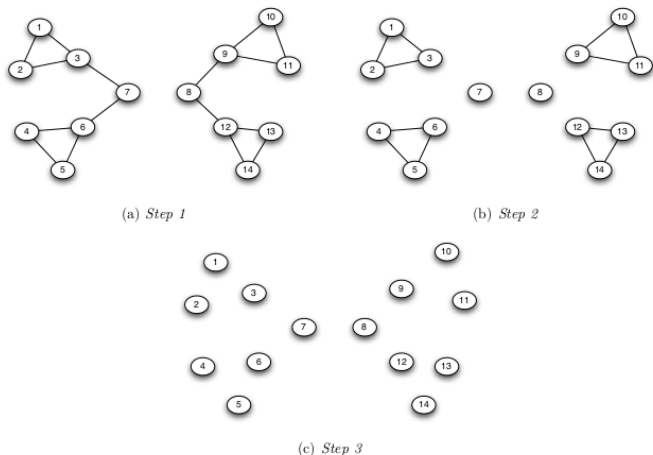
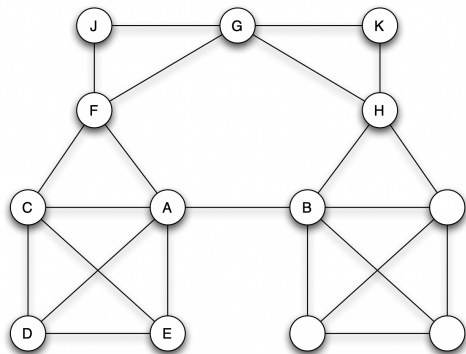


Figure 3-16 from Easley and Kleinberg (2010)

Local bridge

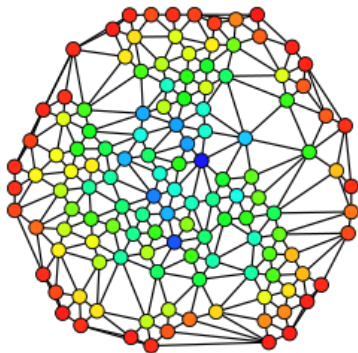
- Last time we saw the concept of **local bridge**: an edge which increased the shortest paths if cut.
- A–B is a local bridge here.



Gatekeepers: generalising the notion of a local bridge

- Up to now we discussed gate-keeping property of *edges*.
- If we want to find *nodes* which are gatekeepers, we can use [betweenness centrality](#).

Betweenness, example



Claudio Rocchini: https://commons.wikimedia.org/wiki/File:Graph_betweenness.svg

- Betweenness: red is minimum; dark blue is maximum.

Betweenness centrality, formally (from Brandes 2008)

- Directed graph $G = \langle V, E \rangle$
- $\sigma(s, t)$: number of shortest paths between nodes s and t
- $\sigma(s, t|v)$: number of shortest paths between nodes s and t that pass through v .
- $C_B(v)$, the betweenness centrality of v :

$$C_B(v) = \sum_{s, t \in V} \frac{\sigma(s, t|v)}{\sigma(s, t)}$$

- If $s = t$, then $\sigma(s, t) = 1$
- If $v \in \{s, t\}$, then $\sigma(s, t|v) = 0$

Number of shortest paths

- $\sigma(s, t)$ can be calculated recursively:

$$\sigma(s, t) = \sum_{u \in \text{Pred}(t)} \sigma(s, u)$$

- $\text{Pred}(t) = \{u : (u, t) \in E, d(s, t) = d(s, u) + 1\}$ predecessors of t on shortest path from s
- $d(s, u)$: Distance between nodes s and u
- This can be done by running Breadth First search with each node as source s once, for total complexity of $O(V(V + E))$.

Pairwise dependencies

- There are a cubic number of pairwise dependencies $\delta(s, t|v)$ where:

$$\delta(s, t|v) = \frac{\sigma(s, t|v)}{\sigma(s, t)}$$

- Naive algorithm uses lots of space.
- Brandes (2001) algorithm intuition: the dependencies can be aggregated without calculating them all explicitly.
- Recursive: can calculate dependency of s on v based on dependencies one step further away.

One-sided dependencies

Define **one-sided dependencies**:

$$\delta(s|v) = \sum_{t \in V} \delta(s, t|v)$$

Then Brandes (2001) shows:

$$\delta(s|v) = \sum_{\substack{(v,w) \in E \\ w: d(s,w)=d(s,v)+1}} \frac{\sigma(s, v)}{\sigma(s, w)} \cdot (1 + \delta(s|w))$$

And:

$$C_B(v) = \sum_{s \in V} \delta(s|v)$$

Brandes algorithm

- Iterate over all vertices s in V
- Calculate $\delta(s|v)$ for all $v \in V$ in two phases:
 - 1 Breadth-first search, calculating distances and shortest path counts from s , push all vertices onto stack as they're visited.
 - 2 Visit all vertices in reverse order (pop off stack), aggregating dependencies according to equation.

Brandes (2008) pseudocode

Shortest-path vertex betweenness (Brandes, 2001).

```
input: directed graph  $G = (V, E)$ 
data: queue  $Q$ , stack  $S$  (both initially empty)
and for all  $v \in V$ :
   $dist[v]$ : distance from source
   $Pred[v]$ : list of predecessors on shortest paths from source
   $\sigma[v]$ : number of shortest paths from source to  $v \in V$ 
   $\delta[v]$ : dependency of source on  $v \in V$ 
output: betweenness  $c_B[v]$  for all  $v \in V$  (initialized to 0)

for  $s \in V$  do
  ▼ single-source shortest-paths problem
  ▼ initialization
  for  $w \in V$  do  $Pred[w] \leftarrow$  empty list
  for  $t \in V$  do  $dist[t] \leftarrow \infty$ ;  $\sigma[t] \leftarrow 0$ 
   $dist[s] \leftarrow 0$ ;  $\sigma[s] \leftarrow 1$ 
  enqueue  $s \rightarrow Q$ 

  while  $Q$  not empty do
    dequeue  $v \leftarrow Q$ ; push  $v \rightarrow S$ 
    foreach vertex  $w$  such that  $(v, w) \in E$  do
      ▼ path discovery //  $w$  found for the first time?
      if  $dist[w] = \infty$  then
         $dist[w] \leftarrow dist[v] + 1$ 
        enqueue  $w \rightarrow Q$ 
      ▼ path counting // edge  $(v, w)$  on a shortest path?
      if  $dist[w] = dist[v] + 1$  then
         $\sigma[w] \leftarrow \sigma[w] + \sigma[v]$ 
        append  $v \rightarrow Pred[w]$ 

  ▼ accumulation // back-propagation of dependencies
  for  $v \in V$  do  $\delta[v] \leftarrow 0$ 
  while  $S$  not empty do
    pop  $w \leftarrow S$ 
    for  $v \in Pred[w]$  do  $\delta[v] \leftarrow \delta[v] + \frac{\sigma[v]}{\sigma[w]} \cdot (1 + \delta[w])$ 
    if  $w \neq s$  then  $c_B[w] \leftarrow c_B[w] + \delta[w]$ 
```

Step 1 - Prepare for BFS tree walk (Node A as s)

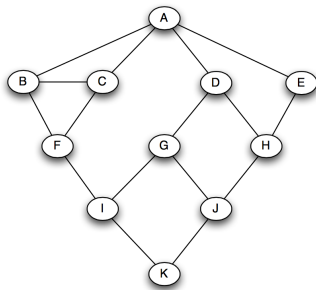
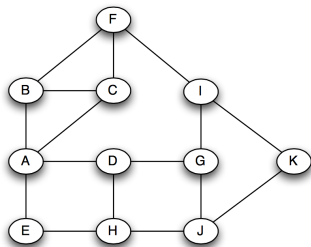
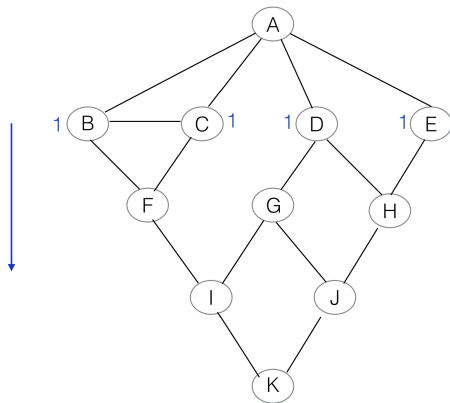


Figure 3-18 from Easley and Kleinberg (2010)

Brandes (2008) pseudocode: phase 1

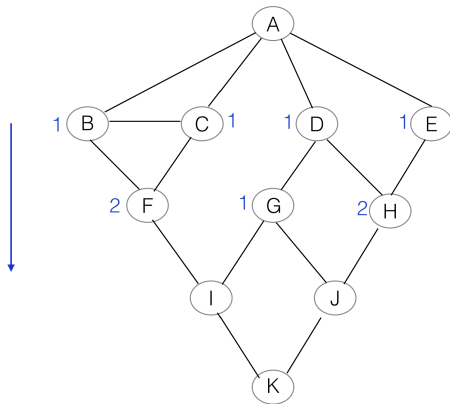
```
while  $Q$  not empty do
  dequeue  $v \leftarrow Q$ ; push  $v \rightarrow S$ 
  foreach vertex  $w$  such that  $(v, w) \in E$  do
    ▼ path discovery // —  $w$  found for the first time?
    if  $dist[w] = \infty$  then
       $dist[w] \leftarrow dist[v] + 1$ 
      enqueue  $w \rightarrow Q$ 
    ▼ path counting // — edge  $(v, w)$  on a shortest path?
    if  $dist[w] = dist[v] + 1$  then
       $\sigma[w] \leftarrow \sigma[w] + \sigma[v]$ 
      append  $v \rightarrow Pred[w]$ 
```


Step 2 - Calculate $\sigma(s, v)$, the number of shortest paths between s and v



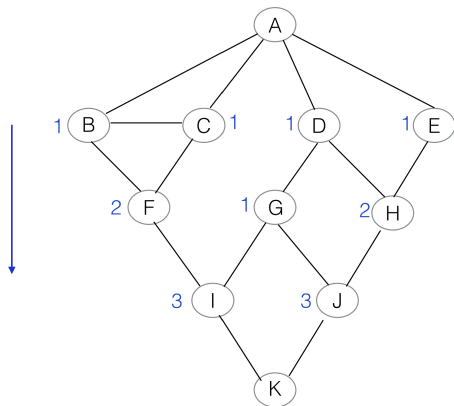
$$\sigma(s, t) = \sum_{u \in \text{Pred}(t)} \sigma(s, u)$$

Step 2 - Calculate $\sigma(s, v)$, the number of shortest paths between s and v



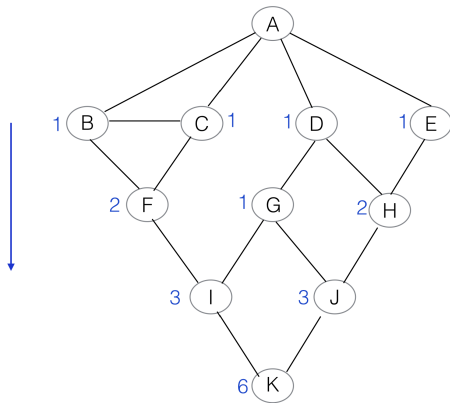
$$\sigma(s, t) = \sum_{u \in \text{Pred}(t)} \sigma(s, u)$$

Step 2 - Calculate $\sigma(s, v)$, the number of shortest paths between s and v



$$\sigma(s, t) = \sum_{u \in \text{Pred}(t)} \sigma(s, u)$$

Step 2 - Calculate $\sigma(s, v)$, the number of shortest paths between s and v



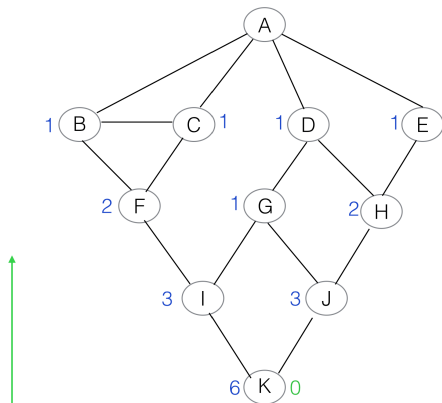
$$\sigma(s, t) = \sum_{u \in \text{Pred}(t)} \sigma(s, u)$$

Brandes (2008) pseudocode: phase 2

└

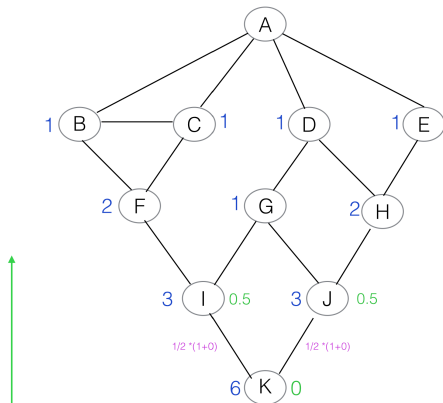
```
▼ accumulation // — back-propagation of dependencies  
  for  $v \in V$  do  $\delta[v] \leftarrow 0$   
  while  $S$  not empty do  
    pop  $w \leftarrow S$   
    for  $v \in \text{Pred}[w]$  do  $\delta[v] \leftarrow \delta[v] + \frac{\sigma[v]}{\sigma[w]} \cdot (1 + \delta[w])$   
    if  $w \neq s$  then  $c_B[w] \leftarrow c_B[w] + \delta[w]$ 
```

Step 3 - Calculate $\delta(s|v)$, the dependency of s on v



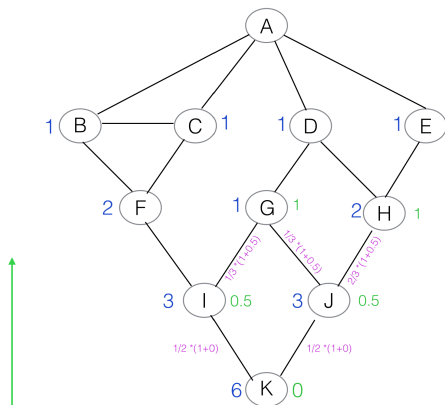
$$\delta(s|v) = \sum_{\substack{(v,w) \in E \\ w: d(s,w)=d(s,v)+1}} \sigma(s,v)/\sigma(s,w) \cdot (1 + \delta(s|w))$$

Step 3 - Calculate $\delta(s|v)$, the dependency of s on v



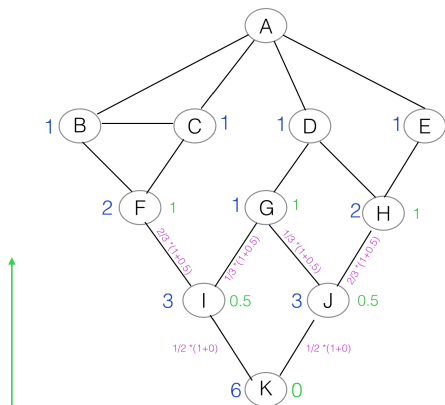
$$\delta(s|v) = \sum_{\substack{(v,w) \in E \\ w: d(s,w)=d(s,v)+1}} \sigma(s,v) / \sigma(s,w) \cdot (1 + \delta(s|w))$$

Step 3 - Calculate $\delta(s|v)$, the dependency of s on v



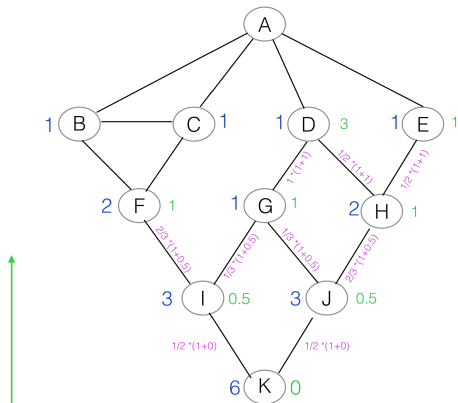
$$\delta(s|v) = \sum_{\substack{(v,w) \in E \\ w: d(s,w)=d(s,v)+1}} \sigma(s,v) / \sigma(s,w) \cdot (1 + \delta(s|w))$$

Step 3 - Calculate $\delta(s|v)$, the dependency of s on v



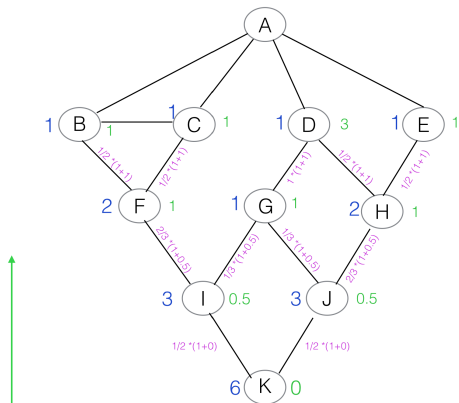
$$\delta(s|v) = \sum_{\substack{(v,w) \in E \\ w: d(s,w)=d(s,v)+1}} \sigma(s,v) / \sigma(s,w) \cdot (1 + \delta(s|w))$$

Step 3 - Calculate $\delta(s|v)$, the dependency of s on v



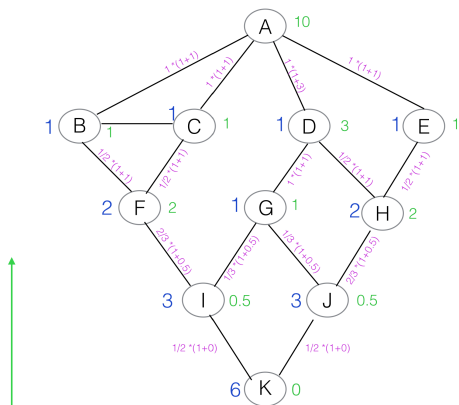
$$\delta(s|v) = \sum_{\substack{(v,w) \in E \\ w: d(s,w)=d(s,v)+1}} \sigma(s,v)/\sigma(s,w) \cdot (1 + \delta(s|w))$$

Step 3 - Calculate $\delta(s|v)$, the dependency of s on v



$$\delta(s|v) = \sum_{\substack{(v,w) \in E \\ w: d(s,w) = d(s,v) + 1}} \sigma(s,v) / \sigma(s,w) \cdot (1 + \delta(s|w))$$

Step 3 - Calculate $\delta(s|v)$, the dependency of s on v



$$\delta(s|v) = \sum_{\substack{(v,w) \in E \\ w: d(s,w)=d(s,v)+1}} \sigma(s,v)/\sigma(s,w) \cdot (1 + \delta(s|w))$$

Step 4 - Calculate betweenness centrality

- You saw one iteration with $s = A$.
- Now perform V iterations, once with each node as source.
- Sum up the $\delta(s|v)$ for each node: this gives the node's betweenness centrality.

Brandes (2008) pseudocode

Shortest-path vertex betweenness (Brandes, 2001).

```
input: directed graph  $G = (V, E)$ 
data: queue  $Q$ , stack  $S$  (both initially empty)
and for all  $v \in V$ :
   $dist[v]$ : distance from source
   $Pred[v]$ : list of predecessors on shortest paths from source
   $\sigma[v]$ : number of shortest paths from source to  $v \in V$ 
   $\delta[v]$ : dependency of source on  $v \in V$ 
output: betweenness  $c_B[v]$  for all  $v \in V$  (initialized to 0)

for  $s \in V$  do
  ▼ single-source shortest-paths problem
  ▼ initialization
  for  $w \in V$  do  $Pred[w] \leftarrow$  empty list
  for  $t \in V$  do  $dist[t] \leftarrow \infty$ ;  $\sigma[t] \leftarrow 0$ 
   $dist[s] \leftarrow 0$ ;  $\sigma[s] \leftarrow 1$ 
  enqueue  $s \rightarrow Q$ 

  while  $Q$  not empty do
    dequeue  $v \leftarrow Q$ ; push  $v \rightarrow S$ 
    foreach vertex  $w$  such that  $(v, w) \in E$  do
      ▼ path discovery //  $w$  found for the first time?
      if  $dist[w] = \infty$  then
         $dist[w] \leftarrow dist[v] + 1$ 
        enqueue  $w \rightarrow Q$ 
      ▼ path counting // edge  $(v, w)$  on a shortest path?
      if  $dist[w] = dist[v] + 1$  then
         $\sigma[w] \leftarrow \sigma[w] + \sigma[v]$ 
        append  $v \rightarrow Pred[w]$ 

  ▼ accumulation // back-propagation of dependencies
  for  $v \in V$  do  $\delta[v] \leftarrow 0$ 
  while  $S$  not empty do
    pop  $w \leftarrow S$ 
    for  $v \in Pred[w]$  do  $\delta[v] \leftarrow \delta[v] + \frac{\sigma[v]}{\sigma[w]} \cdot (1 + \delta[w])$ 
    if  $w \neq s$  then  $c_B[w] \leftarrow c_B[w] + \delta[w]$ 
```

Brandes (2008): undirected graphs

- As specified, this is for directed graphs.
- But undirected graphs are easy: the algorithm works in exactly the same way, except that each pair is considered twice, once in each direction.
- Therefore: halve the scores at the end for undirected graphs.
- Brandes (2008) has lots of other variants, including edge betweenness centrality, which we'll use in the next session.

Today

- **Task 11:** Implement the Brandes algorithm for efficiently determining the betweenness of each node.

Literature

- Detailed notes on the Brandes algorithm on course page / Moodle.
- Easley and Kleinberg (2010, page 79-82). But this is an informal description.
- Ulrich Brandes (2001). A faster algorithm for betweenness centrality. *Journal of Mathematical Sociology*. 25:163–177.
- Ulrich Brandes (2008) On variants of shortest-path betweenness centrality and their generic computation. *Social Networks*. 30 (2008), pp. 136–145