Graphs / Networks
Centrality measures, algorithms, interactive applications

Duen Horng (Polo) Chau
Georgia Tech

Partly based on materials by Professors Guy Lebanon, Jeffrey Heer, John Stasko, Christos Faloutsos, Le Song
Recap…

- **Last time:** Basics, how to build graph, store graph, laws, etc.
- **Today:** Centrality measures, algorithms, interactive applications for visualization and recommendation
Centrality
= “Importance”
Why Node Centrality?

What can we do if we can rank all the nodes in a graph (e.g., Facebook, LinkedIn, Twitter)?

• Find **celebrities** or influential people in a social network (Twitter)

• Find “**gatekeepers**” who connect communities (headhunters love to find them on LinkedIn)

• What else?
More generally

Helps **graph analysis, visualization, understanding**, e.g.,

- Let us *rank* nodes, group or study them by centrality
- Only show subgraph formed by the **top 100 nodes**, out of the millions in the full graph
- Similar to *google search results* (ranked, and they only show you 10 per page)
- Most graph analysis packages already have centrality algorithms implemented. **Use them!**

Can also compute edge centrality. Here we focus on node centrality.
Degree Centrality (easiest)

Degree = number of neighbors

For directed graphs

- in degree = No. of incoming edges
- out degree = No. of outgoing edges

Algorithms?

- Sequential scan through edge list
- What about for a graph stored in SQLite?
Computing degrees using SQL

Recall simplest way to store a graph in SQLite:

```
edges(source_id, target_id)
```

1. Create index for each column

2. Use `group by` statement to find node degrees

```
select count(*) from edges group by source_id;
```
Betweenness Centrality

High betweenness

= important “gatekeeper” or liaison

Betweenness of a node v

= \sum_{s \neq v \neq t \in V} \frac{\sigma_{st}(v)}{\sigma_{st}}

= how often a node serves as the “bridge” that connects two other nodes.
(Local) Clustering Coefficient

A node’s clustering coefficient is a measure of how close the node’s neighbors are from forming a clique.

- $1 = \text{neighbors form a clique}$
- $0 = \text{No edges among neighbors}$

(Assuming undirected graph)

“Local” means it’s for a node; can also compute a graph’s “global” coefficient.
Computing Clustering Coefficients...

Requires triangle counting

Real social networks have a lot of triangles

• Friends of friends are friends

But: triangles are expensive to compute

(3-way join; several approx. algos)

Can we do that quickly?
Super Fast Triangle Counting
[Tsourakakis ICDM 2008]

But: triangles are expensive to compute
(3-way join; several approx. algos)
Q: Can we do that quickly?
A: Yes!

\[
\text{#triangles} = \frac{1}{6} \sum (\lambda_i)^3
\]
(and, because of skewness, we only need the top few eigenvalues!)
Wikipedia graph 2006-Nov-04
\(\approx 3.1\text{M nodes} \approx 37\text{M edges}\)

1000x+ speed-up, >90% accuracy

1000 1050 1100 1150 1200 1250 1300 1350

90 92 94 96 98

(1021x, 97.4%)

(1277x, 94.7%)

(1329x, 92.8%)

1000x+ speed-up, >90% accuracy
More Centrality Measures…

- Degree
- Betweenness
- Closeness, by computing
  - Shortest paths
- “Proximity” (usually via random walks) — used successfully in a lot of applications
- Eigenvector
- …
PageRank (Google)

PageRank: Problem

Given a directed graph, find its most interesting/central node

A node is important, if it is connected with important nodes (recursive, but OK!)
PageRank: Solution

Given a directed graph, find its most interesting/central node

Proposed solution:
use random walk; spot most ‘popular’ node
(-> steady state probability (ssp))

A node has high ssp, if it is connected with high ssp nodes
(recursive, but OK!)

“state” = webpage
(Simplified) PageRank

Let $B$ be the transition matrix: transposed, column-normalized.
(Simplified) PageRank

\[ B \, p = p \]
(Simplified) PageRank

- $B p = 1 \times p$
- thus, $p$ is the eigenvector that corresponds to the highest eigenvalue ($=1$, since the matrix is column-normalized)
- Why does such a $p$ exist?
  - $p$ exists if $B$ is nxn, nonnegative, irreducible [Perron–Frobenius theorem]
(Simplified) PageRank

• In short: imagine a particle randomly moving along the edges
• compute its steady-state probability (ssp)

Full version of algorithm:
  with occasional random jumps

Why? To make the matrix irreducible
Full Algorithm

- With probability $1-c$, fly-out to a random node
- Then, we have

$$p = c \mathbf{B} p + (1-c)/n \quad \Rightarrow$$
With probability $1-c$, fly-out to a random node.

Then, we have

\[ p = c \mathbf{B} p + \frac{1-c}{n} \mathbf{1} \Rightarrow \]

\[ p = \frac{1-c}{n} [\mathbf{I} - c \mathbf{B}]^{-1} \mathbf{1} \]
PageRank Explained with Javascript

http://williamcotton.com/pagerank-explained-with-javascript
PageRank for graphs (generally)

You can compute PageRank for any graphs!

Should be in your algorithm “toolbox”

• Better than simple centrality measure (e.g., degree)

• Fast to compute for large graphs (O(E))

But can be “misled” (Google Bomb)

• How?
Personalized PageRank

Make one small variation of PageRank

• Intuition: not all pages are equal, some more relevant to a person’s specific needs

• How?
“Personalizing” PageRank

- With probability $1 - c$, fly-out to a random node some preferred nodes
- Then, we have

$$p = c \mathbf{B} p + (1 - c)/n \mathbf{1} \Rightarrow$$

$$p = (1 - c)/n \left( \mathbf{I} - c \mathbf{B} \right)^{-1} \mathbf{1}$$
Why learn Personalized PageRank?

Can be used for **recommendation**, e.g.,

- If I like this webpage, what would I also be interested?
- If I like this product, what other products I also like? (in a user-product bipartite graph)
- Also helps with **visualizing large graphs**
  - Instead of visualizing every single nodes, visualize the **most important ones**

Again, very flexible. Can be run on **any graph**.
How to compute (Simplified) PageRank for huge matrix?

Use the power iteration method

http://en.wikipedia.org/wiki/Power_iteration

Can initialize this vector to any non-zero vector, e.g., all “1”s
Building an interactive application

Will show you an example application (Apolo) that uses a “diffusion-based” algorithm to perform recommendation on a large graph

- **Personalized PageRank**
  (= Random Walk with Restart)

- Belief Propagation
  (powerful inference algorithm, for fraud detection, image segmentation, error-correcting codes, etc.)

- “Spreading activation” or “degree of interest” in Human-Computer Interaction (HCI)

- Guilt-by-association techniques
Building an interactive application

Why diffusion-based algorithms are widely used?

• **Intuitive to interpret**
  uses “network effect”, homophily, etc.

• **Easy to implement**
  Math is relatively simple

• **Fast**
  run time linear to #edges, or better

• **Probabilistic** meaning
Human-In-The-Loop Graph Mining

Apolo: Machine Learning + Visualization

CHI 2011
Finding More Relevant Nodes

Apolo uses guilt-by-association (Belief Propagation, similar to personalized PageRank)
**Demo: Mapping the Sensemaking Literature**

**Nodes:** 80k papers from Google Scholar (node size: #citation)

**Edges:** 150k citations

---

*The cost structure of sensemaking*


245 citations 8 versions
The cost structure of sensemaking


245 citations  8 versions
Key Ideas (Recap)

Specify exemplars

Find other relevant nodes (BP)
Apolo’s Contributions

1 Human + Machine

It was like having a partnership with the machine.

2 Personalized Landscape
Apolo 2009
Apolo 2010
Apolo 2011

22,000 lines of code. Java 1.6. Swing. Uses SQLite3 to store graph on disk.

The cost structure of sensemaking
245 citations 8 versions
User Study

Used citation network

**Task:** Find related papers for 2 sections in a survey paper on *user interface*

- Model-based generation of UI
- Rapid prototyping tools
Between subjects design
Participants: grad student or research staff
Higher is better.

Apolo wins.

* Statistically significant, by two-tailed t test, p <0.05
Apolo: Recap

A mixed-initiative approach for exploring and creating personalized landscape for large network data

Apolo = ML + Visualization + Interaction
Practitioners’ guide to building (interactive) applications

Think about scalability early

• e.g., picking a scalable algorithm early on

When building interactive applications, use iterative design approach (as in Apolo)

• Why? It’s hard to get it right the first time

• Create prototype, evaluate, modify prototype, evaluate, ...

• Quick evaluation helps you identify important fixes early (can save you a lot of time)
Practitioners’ guide to building (interactive) applications

How to do iterative design?

What kinds of prototypes?
  • Paper prototype, lo-fi prototype, high-fi prototype

What kinds of evaluation? Important to involve REAL users as early as possible
  • Recruit your friends to try your tools
  • Lab study (controlled, as in Apolo)
  • Longitudinal study (usage over months)
  • Deploy it and see the world’s reaction!

• To learn more:
  • CS 6750 Human-Computer Interaction
  • CS 6455 User Interface Design and Evaluation
If you want to know more about people…

http://amzn.com/0321767535
Polonium: Tera-Scale Graph Mining and Inference for Malware Detection

SDM 2011
Typical Malware Detection Method

**Signature-based detection**
1. Collect malware
2. Generate **signatures**
3. Distribute to users
4. Scan computers for matches

What about "zero-day" malware?

**No samples → No signatures → No detection**

How to detect them early?
Reputation-Based Detection

Computes **reputation score** for each application
  e.g., MSWord.exe

Poor reputation = Malware
I led initial design and development

Serving 120 million users

Answered trillions of queries

Polonium

Propagation of leverage of network influence unearths malware

Patented
Polonium works with 60 Terabyte Data

50 million **machines**
anonymously reported their **executable files**

900 million **unique files**
(Identified by their cryptographic hash values)

**Goal:** label **malware** and **good files**
## Why A Hard Problem?

<table>
<thead>
<tr>
<th>Existing Research</th>
<th>Polonium</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Small</strong> dataset</td>
<td><strong>Huge</strong> dataset (60 terabytes)</td>
</tr>
<tr>
<td>Detects <em>specific</em> malware (e.g., worm, trojans)</td>
<td>Detects <em>all</em> types (needs a general method)</td>
</tr>
<tr>
<td><strong>Many</strong> false alarms (&gt;10%)</td>
<td><strong>Strict</strong> (&lt;1%)</td>
</tr>
</tbody>
</table>
Polonium: Problem Definition

Given

Undirected machine-file bipartite graph

37 billion edges, 1 billion nodes (machines, files)

Some file labels from Symantec (good or bad)

Find

Labels for all unknown files
Symantec has a ground truth database of known-good and known-bad files

e.g., set known-good file’s prior to 0.9
How to Gauge Machine Reputation?

Computed using Symantec’s proprietary formula; a value between 0 and 1

Derived from anonymous aspects of machine’s usage and behavior
How to propagate *known* information to the *unknown*?
Key Idea: Guilt-by-Association

**GOOD files** likely appear on **GOOD** machines

**BAD files** likely appear on **BAD** machines

Also known as **Homophily**

<table>
<thead>
<tr>
<th>File</th>
<th>Machine</th>
</tr>
</thead>
<tbody>
<tr>
<td>Good</td>
<td>0.9</td>
</tr>
<tr>
<td>Bad</td>
<td>0.1</td>
</tr>
</tbody>
</table>
How to propagate known information to the unknown?

Adapts **Belief Propagation (BP)**

A powerful **inference** algorithm

Used in image processing, computer vision, error-correcting codes, etc.
Propagating Reputation

<table>
<thead>
<tr>
<th>Machine</th>
<th>Good</th>
<th>Bad</th>
</tr>
</thead>
<tbody>
<tr>
<td>Good</td>
<td>0.9</td>
<td>0.1</td>
</tr>
<tr>
<td>Bad</td>
<td>0.1</td>
<td>0.9</td>
</tr>
</tbody>
</table>

Example

<table>
<thead>
<tr>
<th>File</th>
<th>Good</th>
<th>Bad</th>
</tr>
</thead>
<tbody>
<tr>
<td>Good</td>
<td>0.9</td>
<td>0.1</td>
</tr>
<tr>
<td>Bad</td>
<td>0.1</td>
<td>0.9</td>
</tr>
</tbody>
</table>

Machines

Files

| 1   | 0.92 |
| 2   | 0.58 |
| 3   | 0.38 |
| 4   | 0.06 |
Two Equations in Belief Propagation

\[ b_i (x_i) = k \phi (x_i) \prod_{x_j \in N(i)} m_{ji} (x_i) \]

\[ m_{ij} (x_j) \leftarrow \sum_{x_i \in X} \psi_{ij} (x_i, x_j) \phi (x_i) \prod_{k \in N(i) \setminus j} m_{ki} (x_i) \]
Computing Node Belief (Reputation)

Belief

Prior belief

Neighbors’ opinions

\[ b_i(x_i) = k\phi(x_i) \prod_{x_j \in N(i)} m_{ji}(x_i) \]
Creating Message for Neighbor

\[ m_{ij}(x_j) \leftarrow \sum_{x_i \in X} \psi_{ij}(x_i, x_j) \phi(x_i) \prod_{k \in N(i) \setminus j} m_{ki}(x_i) \]

<table>
<thead>
<tr>
<th></th>
<th>Good</th>
<th>Bad</th>
</tr>
</thead>
<tbody>
<tr>
<td>Good</td>
<td>0.9</td>
<td>0.1</td>
</tr>
<tr>
<td>Bad</td>
<td>0.1</td>
<td>0.9</td>
</tr>
</tbody>
</table>
Evaluation

Using millions of ground truth files, 10-fold cross validation.

85% True Positive Rate
1% False Alarms

Ideal

True Positive Rate
% of bad correctly labeled

Boosted existing methods by
10 absolute % point

False Positive Rate (False Alarms)
% of good labeled as bad
Multi-Iteration Results

True Positive Rate
% of bad correctly labeled

False Positive Rate (False Alarm)
% of good labeled as bad
Scalability

Running Time Per Iteration

Linux
16-core Opteron
256GB RAM

3 hours,
37 billion edges
Scalability
How Did I Scale Up BP?

1. Early termination (after 6 iterations) → Faster

2. Keep edges on disk → Saves 200GB of RAM

3. Computes half of the messages → Twice as fast
Further Scale Up Belief Propagation

Use **Hadoop** if graph doesn’t fit in memory [ICDE’11]

Speed scales up **linearly** with number of machines

---

**Yahoo! M45 cluster**
- 480 machines
- 1.5 PB storage
- 3.5TB machine