Link Analysis: PageRank and Similar Ideas

CS246: Mining Massive Datasets
Jure Leskovec, Stanford University
http://cs246.stanford.edu
New Topic: Graph Data!

- High dim. data
  - Locality sensitive hashing
  - Clustering
  - Dimensionality reduction

- Graph data
  - PageRank, SimRank
  - Community Detection
  - Spam Detection

- Infinite data
  - Filtering data streams
  - Web advertising
  - Queries on streams

- Machine learning
  - SVM
  - Decision Trees
  - Perceptron, kNN

- Apps
  - Recommender systems
  - Association Rules
  - Duplicate document detection

2/6/2013
Jure Leskovec, Stanford C246: Mining Masssive Datasets
Example: PageRank Scores
Random Teleports ($\beta = 0.8$)

\[ y = \begin{bmatrix} 1/3 & 0.33 & 0.24 & 0.26 \\ 1/3 & 0.20 & 0.20 & 0.18 \\ 1/3 & 0.46 & 0.52 & 0.56 \end{bmatrix} \cdot A \]

\[ 1/n \cdot 1 \cdot 1^T \]

\[ M = \begin{bmatrix} 1/2 & 1/2 & 0 \\ 1/2 & 0 & 0 \\ 0 & 1/2 & 1 \end{bmatrix} + 0.2 \]

\[ A = \begin{bmatrix} 1/3 & 1/3 & 1/3 \\ 1/3 & 1/3 & 1/3 \\ 1/3 & 1/3 & 1/3 \end{bmatrix} \]

\[ y = 7/15 \]

\[ a = 7/15 \]

\[ m = 1/15 \]

\[ r = A r \]
Computing Page Rank

- **Key step is matrix-vector multiplication**
  - \( \mathbf{r}^{\text{new}} = \mathbf{A} \cdot \mathbf{r}^{\text{old}} \)
- **Easy if we have enough main memory to hold \( \mathbf{A}, \mathbf{r}^{\text{old}}, \mathbf{r}^{\text{new}} \)**
- **Say \( N = 1 \) billion pages**
  - We need 4 bytes for each entry (say)
  - 2 billion entries for vectors, approx 8GB
  - **Matrix \( \mathbf{A} \) has \( N^2 \) entries**
    - \( 10^{18} \) is a large number!

\[
\mathbf{A} = \beta \mathbf{M} + (1 - \beta) \frac{1}{N} \mathbf{I}
\]

\[
\begin{bmatrix}
\frac{1}{2} & \frac{1}{2} & 0 \\
\frac{1}{2} & 0 & 0 \\
0 & \frac{1}{2} & 1 \\
\end{bmatrix} + 0.2
\begin{bmatrix}
\frac{7}{15} & \frac{7}{15} & \frac{1}{15} \\
\frac{7}{15} & \frac{1}{15} & \frac{1}{15} \\
\frac{1}{15} & \frac{7}{15} & \frac{13}{15} \\
\end{bmatrix}
\]
Matrix Formulation

- Suppose there are $N$ pages
- Consider page $i$, with $d_i$ out-links
- We have $M_{ji} = \frac{1}{|d_i|}$ when $i \rightarrow j$ and $M_{ji} = 0$ otherwise
- The random teleport is equivalent to:
  - Adding a teleport link from $i$ to every other page and setting transition probability to $(1-\beta)/N$
  - Reducing the probability of following each out-link from $1/|d_i|$ to $\beta/|d_i|$  
  - Equivalent: Tax each page a fraction $(1-\beta)$ of its score and redistribute evenly
Rearranging the Equation

- \( r = A \cdot r \), where \( A_{ji} = \beta M_{ji} + \frac{1-\beta}{N} \)
- \( r_j = \sum_{i=1}^{N} A_{ji} \cdot r_i \)
- \( r_j = \sum_{i=1}^{N} \left[ \beta M_{ji} + \frac{1-\beta}{N} \right] \cdot r_i \)
  \[= \sum_{i=1}^{N} \beta M_{ji} \cdot r_i + \frac{1-\beta}{N} \sum_{i=1}^{N} r_i\]
  \[= \sum_{i=1}^{N} \beta M_{ji} \cdot r_i + \frac{1-\beta}{N} \]
  \[\text{since } \sum r_i = 1\]
- So we get: \( r = \beta M \cdot r + \left[ \frac{1-\beta}{N} \right]_N \)

Note: Here we assumed \( M \) has no dead-ends.

\([x]_N \) ... a vector of length \( N \) with all entries \( x \)
We just rearranged the PageRank equation

\[ r = \beta M \cdot r + \left[ \frac{1 - \beta}{N} \right] N \]

- where \([(1-\beta)/N]_N\) is a vector with all \(N\) entries \((1-\beta)/N\)

- \(M\) is a sparse matrix! (with no dead-ends)
  - 10 links per node, approx 10\(N\) entries

- So in each iteration, we need to:
  - Compute \(r^{\text{new}} = \beta M \cdot r^{\text{old}}\)
  - Add a constant value \((1-\beta)/N\) to each entry in \(r^{\text{new}}\)
    - Note if \(M\) contains dead-ends then \(\sum_j r_j^{\text{new}} < 1\) and we also have to renormalize \(r^{\text{new}}\) so that it sums to 1
PageRank: The Complete Algorithm

- **Input:** Graph $G$ and parameter $\beta$
  - Directed graph $G$ with spider traps and dead ends
  - Parameter $\beta$

- **Output:** PageRank vector $r$
  - Set: $r_j^{(0)} = \frac{1}{N}, \ t = 1$
  - do:
    - $\forall j$: $r'_j(t) = \sum_{i \to j} \beta \frac{r_{i}^{(t-1)}}{d_i}$
    - $r'_j(t) = 0$ if in-degree of $j$ is 0
    - Now re-insert the leaked PageRank:
      - $\forall j$: $r_j^{(t)} = r'_j(t) + \frac{1-S}{N}$
    - where: $S = \sum_j r'_j(t)$
    - $t = t + 1$
  - while $\sum_j \left| r_j^{(t)} - r_j^{(t-1)} \right| > \varepsilon$

If the graph has no dead-ends then the amount of leaked PageRank is $1-\beta$. But since we have dead-ends the amount of leaked PageRank may be larger. We have to explicitly account for it by computing $S$. 

Sparse Matrix Encoding

- Encode sparse matrix using only nonzero entries
  - Space proportional roughly to number of links
  - Say 10N, or 4*10*1 billion = 40GB
  - Still won’t fit in memory, but will fit on disk

<table>
<thead>
<tr>
<th>source node</th>
<th>degree</th>
<th>destination nodes</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>3</td>
<td>1, 5, 7</td>
</tr>
<tr>
<td>1</td>
<td>5</td>
<td>17, 64, 113, 117, 245</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>13, 23</td>
</tr>
</tbody>
</table>
Basic Algorithm: Update Step

- Assume enough RAM to fit $r^{new}$ into memory
  - Store $r^{old}$ and matrix $M$ on disk
- 1 step of power-iteration is:
  Initialize all entries of $r^{new} = (1-\beta) / N$
  For each page $i$ (of out-degree $d_i$):
    Read into memory: $i$, $d_i$, $dest_1$, ..., $dest_{d_i}$, $r^{old}(i)$
    For $j = 1...d_i$
      $r^{new}(dest_j) += \beta \cdot r^{old}(i) / d_i$
Assume enough RAM to fit \( r^{new} \) into memory
- Store \( r^{old} \) and matrix \( M \) on disk

In each iteration, we have to:
- Read \( r^{old} \) and \( M \)
- Write \( r^{new} \) back to disk
- Cost per iteration of Power method:
  \[
  = 2|r| + |M|
  \]

Question:
- What if we could not even fit \( r^{new} \) in memory?
Block-based Update Algorithm

- Break $r^{\text{new}}$ into $k$ blocks that fit in memory
- Scan $M$ and $r^{\text{old}}$ once for each block
Analysis of Block Update

- Similar to nested-loop join in databases
  - Break $r^{new}$ into $k$ blocks that fit in memory
  - Scan $M$ and $r^{old}$ once for each block

- Total cost:
  - $k$ scans of $M$ and $r^{old}$
  - Cost per iteration of Power method:
    $$k(|M| + |r|) + |r| = k|M| + (k+1)|r|$$

- Can we do better?
  - Hint: $M$ is much bigger than $r$ (approx 10-20x), so we must avoid reading it $k$ times per iteration
**Block-Stripe Update Algorithm**

<table>
<thead>
<tr>
<th>src</th>
<th>degree</th>
<th>destination</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>4</td>
<td>0, 1</td>
</tr>
<tr>
<td>1</td>
<td>3</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>4</td>
<td>3</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>0</td>
<td>4</td>
<td>5</td>
</tr>
<tr>
<td>1</td>
<td>3</td>
<td>5</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>4</td>
</tr>
</tbody>
</table>

**Break $M$ into stripes!** Each stripe contains only destination nodes in the corresponding block of $r_{new}$.
Block-Stripe Analysis

- Break $M$ into stripes
  - Each stripe contains only destination nodes in the corresponding block of $r^{\text{new}}$
- Some additional overhead per stripe
  - But it is usually worth it
- Cost per iteration of Power method:
  $$= |M|(1+\varepsilon) + (k+1)|r|$$
Some Problems with Page Rank

- Measures generic popularity of a page
  - Biased against topic-specific authorities
  - **Solution:** Topic-Specific PageRank (next)

- Uses a single measure of importance
  - Other models of importance
  - **Solution:** Hubs-and-Authorities

- Susceptible to Link spam
  - Artificial link topographies created in order to boost page rank
  - **Solution:** TrustRank
Topic-Specific PageRank
Instead of generic popularity, can we measure popularity within a topic?

**Goal:** Evaluate Web pages not just according to their popularity, but by how close they are to a particular topic, e.g. “sports” or “history”

**Allows search queries to be answered based on interests of the user**

**Example:** Query “Trojan” wants different pages depending on whether you are interested in sports, history and computer security
Random walker has a small probability of teleporting at any step

Teleport can go to:
- **Standard PageRank**: Any page with equal probability
  - To avoid dead-end and spider-trap problems
- **Topic Specific PageRank**: A topic-specific set of “relevant” pages (teleport set)

Idea: Bias the random walk
- When walker teleports, she pick a page from a set \( S \)
- \( S \) contains only pages that are relevant to the topic
  - E.g., Open Directory (DMOZ) pages for a given topic/query
- For each teleport set \( S \), we get a different vector \( r_s \)
To make this work all we need is to update the teleportation part of the PageRank formulation:

\[ A_{ij} = \begin{cases} 
\beta M_{ij} + (1-\beta)/|S| & \text{if } i \in S \\
\beta M_{ij} & \text{otherwise}
\end{cases} \]

- \( A \) is stochastic!

- We have weighted all pages in the teleport set \( S \) equally
  - Could also assign different weights to pages!

**Compute as for regular PageRank:**

- Multiply by \( M \), then add a vector
- Maintains sparseness
Example

Suppose $S = \{1\}, \beta = 0.8$

<table>
<thead>
<tr>
<th>Node</th>
<th>Iteration</th>
<th>Stable</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0.25</td>
<td>0.4</td>
</tr>
<tr>
<td>2</td>
<td>0.25</td>
<td>0.1</td>
</tr>
<tr>
<td>3</td>
<td>0.25</td>
<td>0.3</td>
</tr>
<tr>
<td>4</td>
<td>0.25</td>
<td>0.2</td>
</tr>
</tbody>
</table>

$S = \{1,2,3,4\}, \beta = 0.8$: $r = [0.13, 0.10, 0.39, 0.36]$

$S = \{1,2,3\}, \beta = 0.8$: $r = [0.17, 0.13, 0.38, 0.30]$

$S = \{1,2\}, \beta = 0.8$: $r = [0.26, 0.20, 0.29, 0.23]$

$S = \{1\}, \beta = 0.8$: $r = [0.29, 0.11, 0.32, 0.26]$

$S = \{1\}, \beta = 0.90$: $r = [0.17, 0.07, 0.40, 0.36]$

$S = \{1\}, \beta = 0.8$: $r = [0.29, 0.11, 0.32, 0.26]$

$S = \{1\}, \beta = 0.70$: $r = [0.39, 0.14, 0.27, 0.19]$
Discovering the Topic

- **Create different PageRanks for different topics**
  - The 16 DMOZ top-level categories:
    - arts, business, sports,...
- **Which topic ranking to use?**
  - User can pick from a menu
  - Classify query into a topic
  - Can use the **context** of the query
    - E.g., query is launched from a web page talking about a known topic
    - History of queries e.g., “basketball” followed by “Jordan”
  - User context, e.g., user’s bookmarks, ...
SimRank: An Application of Personalized PageRank
SimRank: Idea

- **SimRank**: Random walks from a fixed node on *k*-partite graphs
- **Setting**: *k*-partite graph with *k* types of nodes
  - Example: picture nodes and tag nodes
- Do a **Random-Walk with Restarts** from node *u*
  - i.e., teleport set *S* = \{*u*\}
- Resulting scores measures similarity/proximity to node *u*
- **Problem**:
  - Must be done once for each node *u*
  - Suitable for sub-Web-scale applications
SimRank: Example

Q: What is most related conference to ICDM?
SimRank: Example

ICDM
KDD
SDM
ECML
PKDD
PAKDD
CIKM
DMKD
SIGMOD
ICML
ICDE

0.009
0.011
0.008
0.007
0.005
0.005
0.005
0.004
0.004
0.004
0.004
0.005
0.005
0.005
0.005

2/6/2013
Jure Leskovec, Stanford C246: Mining Massive Datasets
3 announcements:
- We will hold office hours via Google hangout:
  - Sat, Sun 10am Pacific time
- HW2 is due
- HW3 is out

TrustRank: Combating the Web Spam
What is Web Spam?

- **Spamming:**
  - Any deliberate action to boost a web page’s position in search engine results, incommensurate with page’s real value

- **Spam:**
  - Web pages that are the result of spamming

- This is a very broad definition
  - **SEO** industry might disagree!
  - SEO = search engine optimization

- Approximately **10-15%** of web pages are spam
Early search engines:
- Crawl the Web
- Index pages by the words they contained
- Respond to search queries (lists of words) with the pages containing those words

Early page ranking:
- Attempt to order pages matching a search query by “importance”

First search engines considered:
- (1) Number of times query words appeared
- (2) Prominence of word position, e.g. title, header
First Spammers

- As people began to use search engines to find things on the Web, those with commercial interests tried to exploit search engines to bring people to their own site – whether they wanted to be there or not

- Example:
  - Shirt-seller might pretend to be about “movies”

- Techniques for achieving high relevance/importance for a web page
How do you make your page appear to be about movies?

1. Add the word movie 1,000 times to your page.
2. Set text color to the background color, so only search engines would see it.
3. Or, run the query “movie” on your target search engine.
4. See what page came first in the listings.
5. Copy it into your page, make it “invisible”.

These and similar techniques are term spam.
Believe what people say about you, rather than what you say about yourself

- Use words in the anchor text (words that appear underlined to represent the link) and its surrounding text

- PageRank as a tool to measure the “importance” of Web pages
Why It Works?

- Our hypothetical shirt-seller looses
  - Saying he is about movies doesn’t help, because others don’t say he is about movies
  - His page isn’t very important, so it won’t be ranked high for shirts or movies
- Example:
  - Shirt-seller creates 1,000 pages, each links to his with “movie” in the anchor text
  - These pages have no links in, so they get little PageRank
  - So the shirt-seller can’t beat truly important movie pages, like IMDB
SPAM FARMING
Once Google became the dominant search engine, spammers began to work out ways to fool Google

**Spam farms** were developed to concentrate PageRank on a single page

**Link spam:**
- Creating link structures that boost PageRank of a particular page
Link Spamming

- Three kinds of web pages from a spammer’s point of view
  - Inaccessible pages
  - Accessible pages
    - e.g., blog comments pages
    - spammer can post links to his pages
  - Own pages
    - Completely controlled by spammer
    - May span multiple domain names
Link Farms

- **Spammer’s goal:**
  - Maximize the PageRank of target page $t$

- **Technique:**
  - Get as many links from accessible pages as possible to target page $t$
  - Construct “link farm” to get PageRank multiplier effect
Link Farms

One of the most common and effective organizations for a link farm
**Analysis**

- **x**: PageRank contributed by accessible pages
- **y**: PageRank of target page \( t \)
- Rank of each “farm” page \( = \frac{\beta y}{M} + \frac{1-\beta}{N} \)

\[
y = x + \beta M \left[ \frac{\beta y}{M} + \frac{1-\beta}{N} \right] + \frac{1-\beta}{N} = x + \beta^2 y + \frac{\beta (1-\beta) M}{N} + \frac{1-\beta}{N}
\]

Very small; ignore

Now we solve for \( y \)

\[
y = \frac{x}{1-\beta^2} + c \frac{M}{N} \quad \text{where} \quad c = \frac{\beta}{1+\beta}
\]
\[ y = \frac{x}{1-\beta^2} + c \frac{M}{N} \]

where \( c = \frac{\beta}{1+\beta} \)

- For \( \beta = 0.85 \), \( 1/(1-\beta^2)= 3.6 \)

- Multiplier effect for “acquired” PageRank
- By making \( M \) large, we can make \( y \) as large as we want
TrustRank: Combating the Web Spam
Combating Spam

- **Combating term spam**
  - Analyze text using statistical methods
  - Similar to email spam filtering
  - Also useful: Detecting approximate duplicate pages

- **Combating link spam**
  - Detection and blacklisting of structures that look like spam farms
    - Leads to another war – hiding and detecting spam farms
  - **TrustRank** = topic-specific PageRank with a teleport set of “trusted” pages
    - Example: .edu domains, similar domains for non-US schools
Basic principle: Approximate isolation

- It is rare for a “good” page to point to a “bad” (spam) page

Sample a set of seed pages from the web

Have an oracle (human) to identify the good pages and the spam pages in the seed set

- Expensive task, so we must make seed set as small as possible
Call the subset of seed pages that are identified as **good** the **trusted pages**

Perform a topic-sensitive PageRank with teleport set = trusted pages

- Propagate trust through links:
  - Each page gets a trust value between 0 and 1

Use a threshold value and mark all pages below the trust threshold as spam
Why is it a good idea?

- **Trust attenuation:**
  - The degree of trust conferred by a trusted page decreases with the distance in the graph.

- **Trust splitting:**
  - The larger the number of out-links from a page, the less scrutiny the page author gives each out-link.
  - Trust is *split* across out-links.
Two conflicting considerations:

- Human has to inspect each seed page, so seed set must be as small as possible
- Must ensure every good page gets adequate trust rank, so need make all good pages reachable from seed set by short paths
Suppose we want to pick a seed set of $k$ pages

How to do that?

(1) PageRank:
   - Pick the top $k$ pages by PageRank
   - Theory is that you can’t get a bad page’s rank really high

(2) Use trusted domains whose membership is controlled, like .edu, .mil, .gov
In the TrustRank model, we start with good pages and propagate trust.

**Complementary view:** What fraction of a page’s PageRank comes from spam pages?

In practice, we don’t know all the spam pages, so we need to estimate.
Spam Mass Estimation

- $r_p = \text{PageRank of page } p$
- $r_p^+ = \text{page rank of } p \text{ with teleport into trusted pages only}$

Then: What fraction of a page’s PageRank comes from spam pages?

- $r_p^- = r_p - r_p^+$

- Spam mass of $p = \frac{r_p^-}{r_p}$
HITS: Hubs and Authorities
Hubs and Authorities

- **HITS (Hypertext-Induced Topic Selection)**
  - Is a measure of importance of pages or documents, similar to PageRank
  - Proposed at around same time as PageRank (‘98)

- **Goal**: Imagine we want to find good newspapers
  - Don’t just find newspapers. Find “experts” – people who link in a coordinated way to good newspapers

- **Idea: Links as votes**
  - Page is more important if it has more links
    - In-coming links? Out-going links?
Finding newspapers

- Hubs and Authorities
  Each page has 2 scores:
  - Quality as an expert (hub):
    - Total sum of votes of pages pointed to
  - Quality as an content (authority):
    - Total sum of votes of experts

- Principle of repeated improvement
Interesting pages fall into two classes:

1. **Authorities** are pages containing useful information
   - Newspaper home pages
   - Course home pages
   - Home pages of auto manufacturers

2. **Hubs** are pages that link to authorities
   - List of newspapers
   - Course bulletin
   - List of US auto manufacturers
Counting in-links: Authority

Each page starts with hub score 1
Authorities collect their votes

(Note this is idealized example. In reality graph is not bipartite and each page has both the hub and authority score)
Hubs collect authority scores

(Note this is idealized example. In reality graph is not bipartite and each page has both the hub and authority score)
Reweighting

Authorities collect hub scores

(Note this is idealized example. In reality graph is not bipartite and each page has both the hub and authority score)
A good hub links to many good authorities

A good authority is linked from many good hubs

Model using two scores for each node:
- **Hub** score and **Authority** score
- Represented as vectors \( h \) and \( a \)
Each page $i$ has 2 scores:
- Authority score: $a_i$
- Hub score: $h_i$

**HITS algorithm:**
- Initialize: $a_i = 1, h_i = 1$
- Then keep iterating:
  - $\forall i$: Authority: $a_i = \sum_{j \rightarrow i} h_j$
  - $\forall i$: Hub: $h_i = \sum_{i \rightarrow j} a_j$
  - $\forall i$: normalize: $\sum_j a_j = 1, \sum_j h_j = 1$
HITS converges to a single stable point

Slightly change the notation:

- Vector $a = (a_1, \ldots, a_n)$, $h = (h_1, \ldots, h_n)$
- Adjacency matrix $(n \times n)$: $A_{ij} = 1$ if $i \rightarrow j$

Then:

$$h_i = \sum_{i \rightarrow j} a_j \Leftrightarrow h_i = \sum_j A_{ij} a_j$$

So: $h = Aa$

And likewise: $a = A^T h$
The hub score of page \( i \) is proportional to the sum of the authority scores of the pages it links to: 

\[
h = \lambda A a
\]

- Constant \( \lambda \) is a scale factor, \( \lambda = 1/\sum h_i \)

The authority score of page \( i \) is proportional to the sum of the hub scores of the pages it is linked from: 

\[
a = \mu A^T h
\]

- Constant \( \mu \) is scale factor, \( \mu = 1/\sum a_i \)
The HITS algorithm:

- Initialize $h$, $a$ to all 1’s
- Repeat:
  - $h = A a$
  - Scale $h$ so that its sums to 1.0
  - $a = A^T h$
  - Scale $a$ so that its sums to 1.0
- Until $h$, $a$ converge (i.e., change very little)
Example

\[ A = \begin{bmatrix} 1 & 1 & 1 \\ 1 & 0 & 1 \\ 0 & 1 & 0 \end{bmatrix} \quad A^T = \begin{bmatrix} 1 & 1 & 0 \\ 1 & 0 & 1 \\ 1 & 1 & 0 \end{bmatrix} \]

\[
\begin{align*}
a(\text{yahoo}) & = 1 & 1 & 1 & 1 & \ldots & 1 \\
a(\text{amazon}) & = 1 & 1 & 4/5 & 0.75 & \ldots & 0.732 \\
a(\text{m’soft}) & = 1 & 1 & 1 & 1 & \ldots & 1
\end{align*}
\]

\[
\begin{align*}
h(\text{yahoo}) & = 1 & 1 & 1 & 1 & \ldots & 1.000 \\
h(\text{amazon}) & = 1 & 2/3 & 0.71 & 0.73 & \ldots & 0.732 \\
h(\text{m’soft}) & = 1 & 1/3 & 0.29 & 0.27 & \ldots & 0.268
\end{align*}
\]

2/6/2013
Hubs and Authorities

- **HITS algorithm in new notation:**
  - Set: \( a = h = 1^n \)
  - **Repeat:**
    - \( h = A a, \quad a = A^T h \)
    - Normalize
  - **Then:** \( a = A^T (A a) \)
  - Thus, in \( 2k \) steps:
    - \( a = (A^T A)^k a \)
    - \( h = (A A^T)^k h \)

- \( a \) is being updated (in 2 steps):
  - \( A^T (A a) = (A^T A) a \)
- \( h \) is updated (in 2 steps):
  - \( A (A^T h) = (A A^T) h \)

Repeated matrix powering
Existence and Uniqueness

- $h = \lambda A a$
- $a = \mu A^T h$
- $h = \lambda \mu A A^T h$
- $a = \lambda \mu A^T A a$

- Under reasonable assumptions about $A$, the HITS iterative algorithm converges to vectors $h^*$ and $a^*$:
  - $h^*$ is the principal eigenvector of matrix $A A^T$
  - $a^*$ is the principal eigenvector of matrix $A^T A$

$\lambda = 1/\sum h_i$
$\mu = 1/\sum a_i$
PageRank and HITS are two solutions to the same problem:

- What is the value of an in-link from \( u \) to \( v \)?
- In the PageRank model, the value of the link depends on the links into \( u \)
- In the HITS model, it depends on the value of the other links out of \( u \)

The destinies of PageRank and HITS post-1998 were very different