DD2476: Search Engines and Information Retrieval Systems

Johan Boye*
KTH

Lecture 5

* Many slides inspired by Manning, Raghavan and Schütze

Short recap

- **Ranked retrieval** model
- Documents are **vectors of weights**

$$d = (w_1, w_2, \ldots, w_N)$$

- where N is the number of unique words in the corpus.

- Weights are **tf_idf** values
  - $tf_{t,d}$ = number of times term t occurs in document d (**term frequency**)
  - idf_t = **inverse document frequency**, bigger for rare words
  - $w_{t,d} = tf_{t,d} \times idf_t$

Short recap

- **Cosine similarity cos(q,d)** is a measure of how similar a query q is to a document d

$$\cos(q,d) = \frac{q \cdot d}{|q||d|}$$

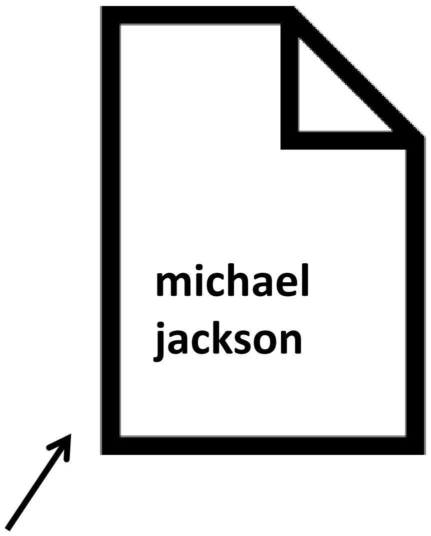
- Given document d, a useful approximation is to:
  - sum the tf-idf-weights for all query terms appearing in d
  - divide the sum by the number of words in d

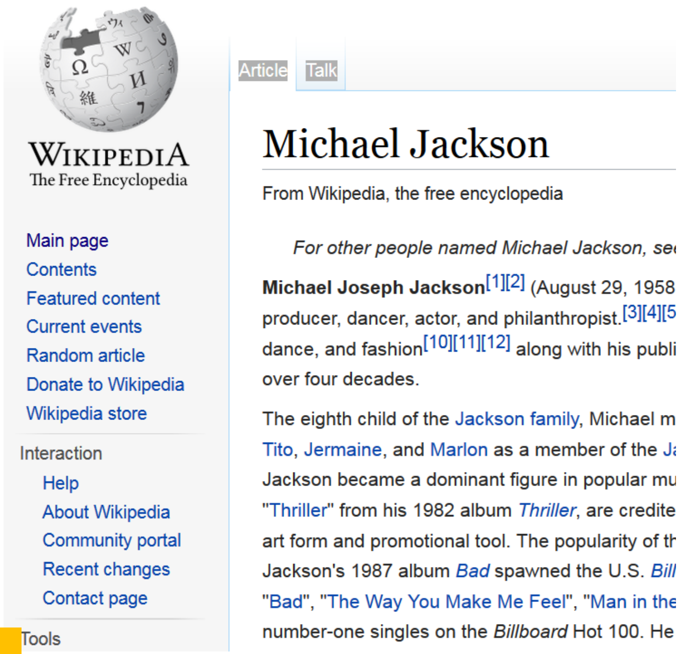
Assignment 2

- 2.1 – 2.2 Implement ranked retrieval with tf_idf weighting
- 2.3 Variants of cosine similarity and tf_idf
- 2.4 Evaluation of ranked results
- 2.6 (C) Implement cosine similarity for Euclidean length

However, a problem…

michael jackson




WIKIPEDIA
The Free Encyclopedia

Main page
Contents
Featured content
Current events
Random article
Donate to Wikipedia
Wikipedia store

Interaction
Help
About Wikipedia
Community portal
Recent changes
Contact page

Tools

Article Talk

Michael Jackson

From Wikipedia, the free encyclopedia

For other people named Michael Jackson, se

Michael Joseph Jackson[1][2] (August 29, 1958 producer, dancer, actor, and philanthropist.[3][4][5 dance, and fashion[10][11][12] along with his publi over four decades.

The eighth child of the Jackson family, Michael m Tito, Jermaine, and Marlon as a member of the J Jackson became a dominant figure in popular mu "Thriller" from his 1982 album *Thriller*, are credite art form and promotional tool. The popularity of th Jackson's 1987 album *Bad* spawned the U.S. *Bill* "Bad", "The Way You Make Me Feel", "Man in the number-one singles on the *Billboard* Hot 100. He

Better cosine similarity with the query!

Solution: Static quality scores

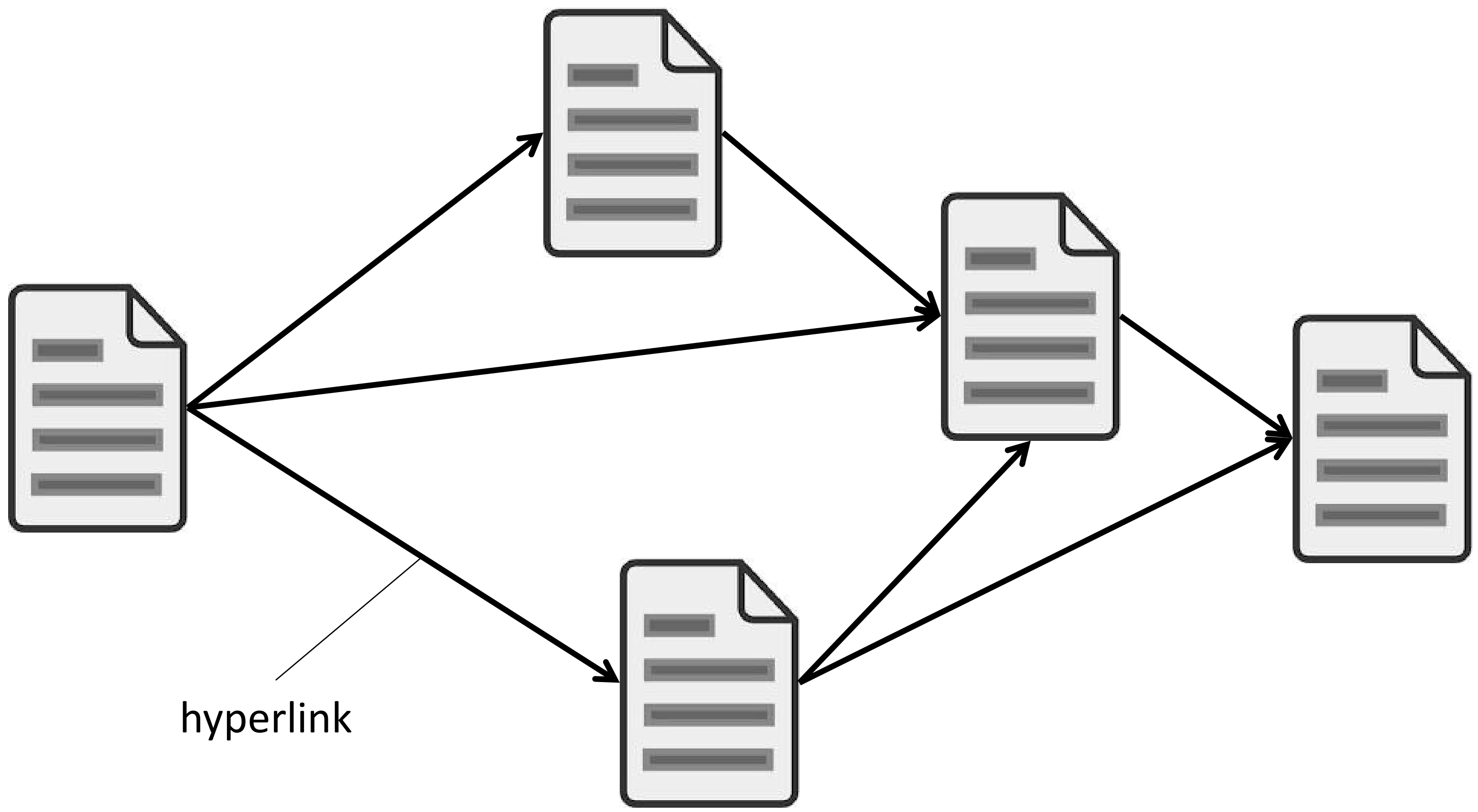
- We want top-ranking documents to be both **relevant** and **authoritative**
  - Relevance – cosine scores
  - Authority – query-independent property
- Assign **query-independent quality score** g(*d*) in [0,1] to each document *d*
- $\text{net-score}(q,d) = w_1 \times g(d) + w_2 \times \cos(q,d)$

Static quality scores

- Which pages should be highly ranked? Alternatives:
  - Personalised search: **Pages I’ve visited before**
  - Pages **visited by lots of users**
  - **Well-known quality** pages (Wikipedia, NY Times, …)
  - Pages with **many important in-links** (PageRank)
  - Money talks: **Sponsored links**
  - (and recently): Veracity: **Pages with true information** are ranked highly, alternative facts less so.

PageRank

The Web as a directed graph

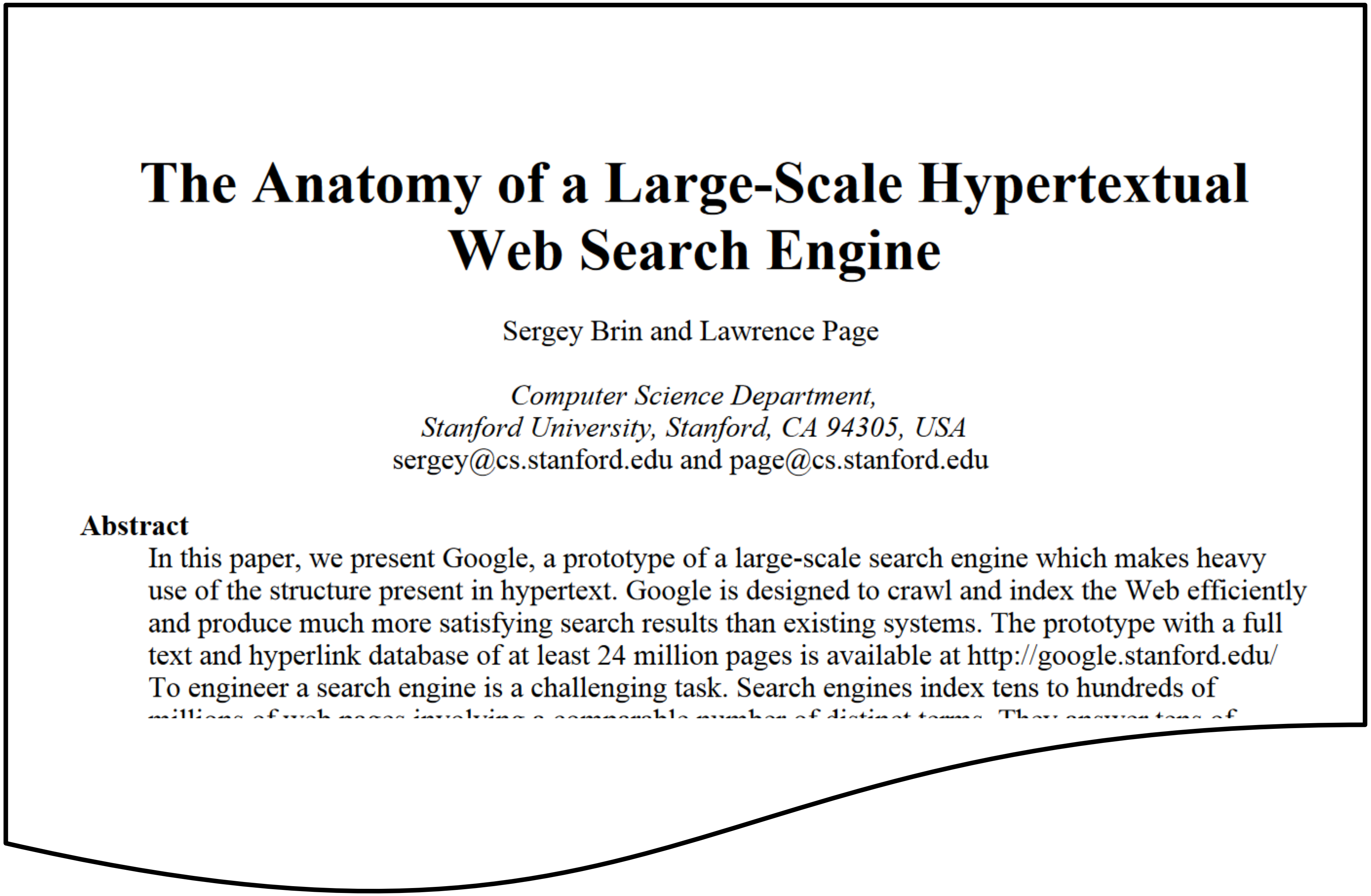


Using link structure for ranking

- **Assumption:** A link from X to Y signals that X's author perceives Y to be an authoritative page.
  - X "casts a vote" on Y.
- **Simple suggestion:** Rank = number of in-links
- However, there are problems with this naive approach.

PageRank: basic ideas

- WWW's particular structure can be exploited
  - pages have links to one another
  - the more in-links, the higher rank
  - in-links from pages having **high rank** are **worth more** than links from pages having low rank
  - this idea is the cornerstone of **PageRank** (Brin & Page 1998)



The Anatomy of a Large-Scale Hypertextual Web Search Engine

Sergey Brin and Lawrence Page

Computer Science Department,
Stanford University, Stanford, CA 94305, USA
sergey@cs.stanford.edu and page@cs.stanford.edu

Abstract

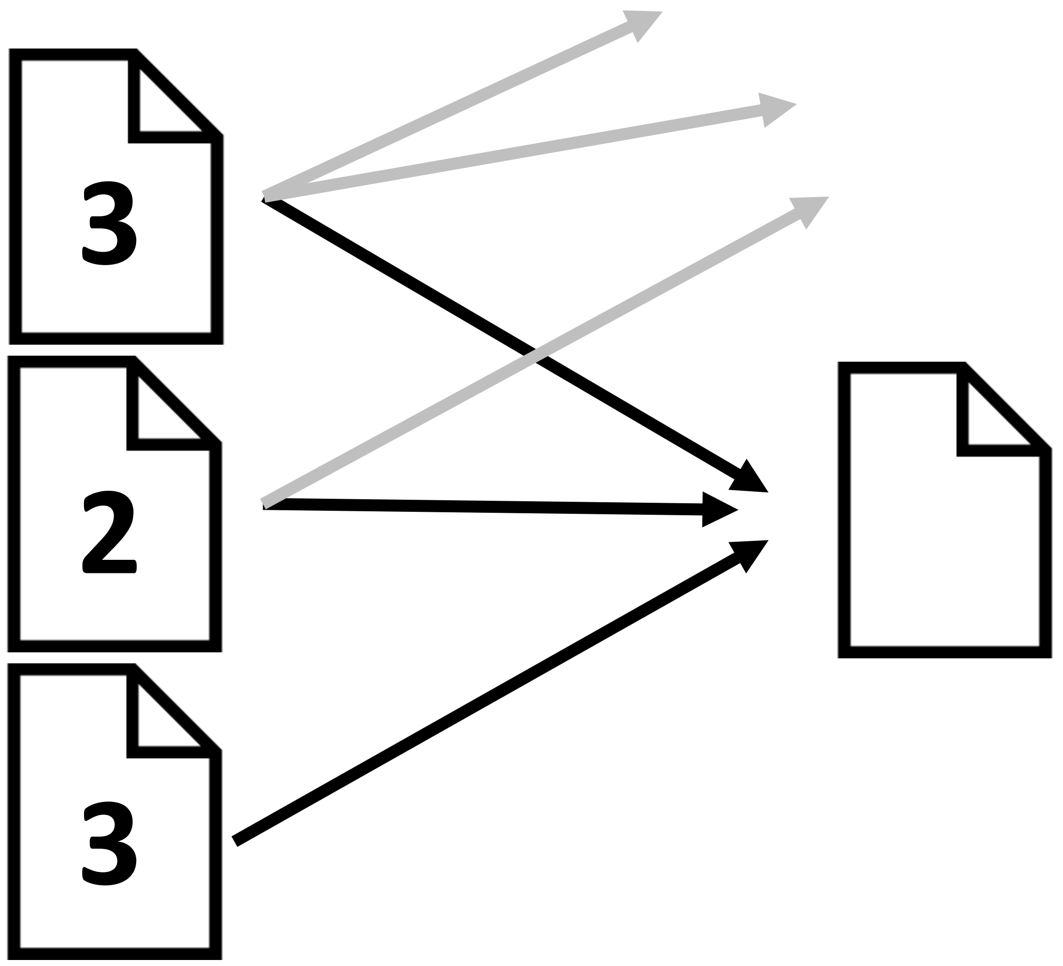
In this paper, we present Google, a prototype of a large-scale search engine which makes heavy use of the structure present in hypertext. Google is designed to crawl and index the Web efficiently and produce much more satisfying search results than existing systems. The prototype with a full text and hyperlink database of at least 24 million pages is available at http://google.stanford.edu/ To engineer a search engine is a challenging task. Search engines index tens to hundreds of millions of web pages involving a comparable number of distinct terms. They answer tens of

PageRank – first attempt

$$PR(p) = \sum_{q \in in(p)} \frac{PR(q)}{L_q}$$

- *p* and *q* are pages
- *in(p)* is the set of pages linking to *p*
- L_q is the number of out-links from *q*

PageRank – first attempt



The random surfer model

- Imagine a **random surfer** that follow links
- The link to follow is selected with uniform probability
- If the surfer reaches a **sink** (a page without links), he **randomly restarts** on a new page
- Every once in a while, the surfer **jumps to a random page** (even if there are links to follow)

PageRank

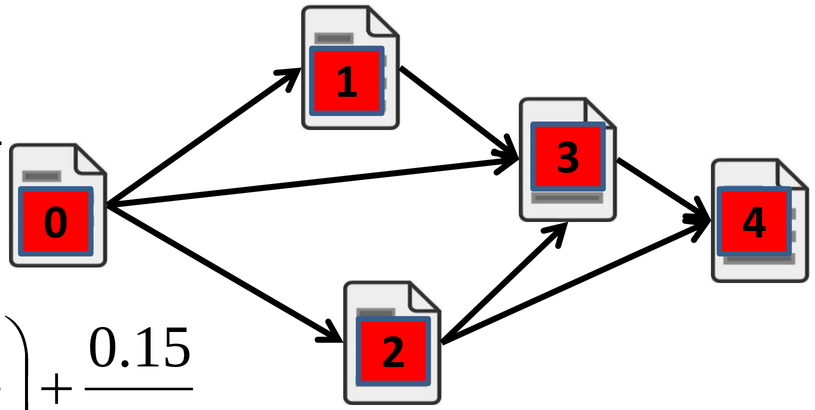
- With **probability *1-c*** the surfer is bored, **stops following links**, and **restarts** on a random page
- Guess: Google uses *c*=0.85

$$PR(p) = c\left(\sum_{q \in in(p)} \frac{PR(q)}{L_q}\right) + \frac{(1-c)}{N}$$

- Without this assumption, the surfer will get stuck in a subset of the web.

PageRank example

$$PR_4 = 0.85 \cdot \left(\frac{PR_2}{2} + PR_3 + \frac{PR_4}{5}\right) + \frac{0.15}{5}$$



$$PR_3 = 0.85 \cdot \left(\frac{PR_0}{3} + PR_1 + \frac{PR_2}{2} + \frac{PR_4}{5}\right) + \frac{0.15}{5}$$

$$PR_2 = PR_1 = 0.85 \cdot \left(\frac{PR_0}{3} + \frac{PR_4}{5}\right) + \frac{0.15}{5}$$

$$PR_0 = 0.85 \cdot \left(\frac{PR_4}{5}\right) + \frac{0.15}{5}$$

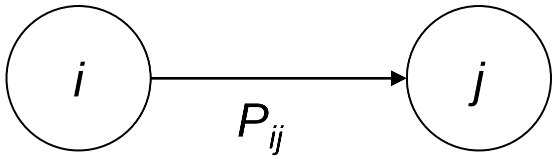
PageRank- interpretations

- Authority / popularity / relative information value
- PR_p = the probability that the random surfer will be at page *p* at any given point in time
- This is called the **stationary probability**
- How do we compute it?

Computing PageRank by power iteration

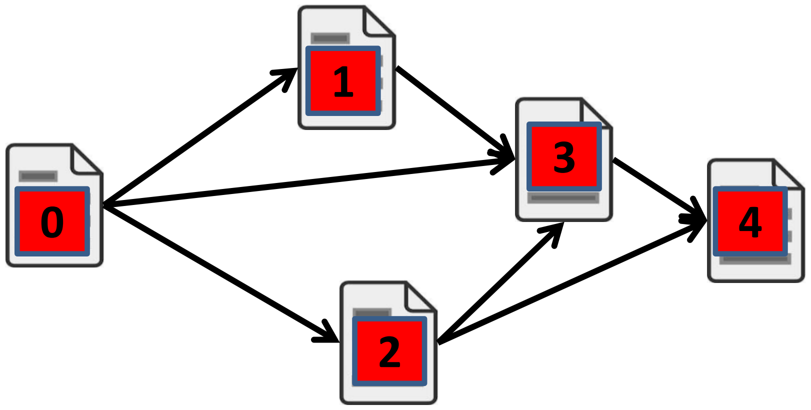
Random surfer as a Markov chain

- The random surfer model suggests a a **Markov chain** formulation
  - A Markov chain consists of *n* states, plus an $n \times n$ transition probability matrix **P**.
  - At each step, we are in exactly one of the states.
  - For $1 \leq i,j \leq n$, the matrix entry P_{ij} tells us the probability of *j* being the next state, given we are currently in state *i*.


$$\begin{tikzcd} \boxed{i} \arrow[r, "P_{ij}"'] & \boxed{j} \end{tikzcd}$$

Transition matrices

$$P = \begin{bmatrix} 0 & 0.33 & 0.33 & 0.33 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0.5 & 0.5 \\ 0 & 0 & 0 & 0 & 1 \\ 0.2 & 0.2 & 0.2 & 0.2 & 0.2 \end{bmatrix}$$



$$J = \begin{bmatrix} 0.2 & 0.2 & 0.2 & 0.2 & 0.2 \\ 0.2 & 0.2 & 0.2 & 0.2 & 0.2 \\ 0.2 & 0.2 & 0.2 & 0.2 & 0.2 \\ 0.2 & 0.2 & 0.2 & 0.2 & 0.2 \\ 0.2 & 0.2 & 0.2 & 0.2 & 0.2 \end{bmatrix}$$

G = cP+(1-c)J

$$\begin{bmatrix} 0.0300 & 0.3105 & 0.3105 & 0.3105 & 0.0300 \\ 0.0300 & 0.0300 & 0.0300 & 0.8800 & 0.0300 \\ 0.0300 & 0.0300 & 0.0300 & 0.4550 & 0.4550 \\ 0.0300 & 0.0300 & 0.0300 & 0.0300 & 0.8800 \\ 0.2000 & 0.2000 & 0.2000 & 0.2000 & 0.2000 \end{bmatrix}$$

Ergodic Markov chains

- A Markov chain is **ergodic** if
  - you have a path from any state to any other
  - For any start state, after a finite transient time T_0, **the probability of being in any state at a fixed time $T>T_0$ is nonzero.**
- Our transition matrix is non-zero positive everywhere $\leftrightarrow$
  the graph is strongly connected $\leftrightarrow$
  the Markov chain is ergodic $\leftrightarrow$
  unique stationary probabilities exist

Probability vectors

- A **probability** (row) **vector x** = (x_1, … x_n) tells us where the walk is at any point.
- E.g., (000…1…000) means we're in state *i*.

  1 *i* *n*

- More generally, the vector **x** = (x_1, … x_n) means the walk is in state *i* with probability x_i.
- **xG** gives the next time step.
- **x** is stationary if **x=xG**

$$\sum_{i=1}^{n} x_i = 1.$$

Updating probabilities

$$\overset{\mathbf{x}}{\begin{bmatrix} p_0 & p_1 & p_2 & p_3 & p_4 \end{bmatrix}} \overset{\mathbf{G}}{\begin{bmatrix} p_{00} & p_{01} & \cdots & & p_{04} \\ p_{10} & & & & \\ \cdots & & \cdots & & \cdots \\ & & & & \\ p_{40} & & & & p_{44} \end{bmatrix}} = \overset{\mathbf{x'}}{\begin{bmatrix} q_0 & q_1 & q_2 & q_3 & q_4 \end{bmatrix}}$$

$$q_0 = p_0 \times p_{00} + p_1 \times p_{10} + p_2 \times p_{20} + p_3 \times p_{30} + p_4 \times p_{40}$$

Stationary probabilites

- Let **a** = (a_1, … a_n) denote the row vector of stationary probabilities.
  - If our current position is described by **a**, then the next step is distributed as **aG**.
  - But **a** is the steady state, so **a**=**aG**.
- Solving the matrix equation **x=xG** gives us **a**.
  - So **a** is the (left) eigenvector for **G**.
- **x** can be computed using **power iteration.**

Power iteration

- Start with any distribution (say **x**=(*10…0*)).
  - After one step, we're at **xG**;
  - after two steps at (**xG)G** , then **((xG)G)G** and so on.
- "Eventually", for "large" *k*, $\mathbf{xG}^k = \mathbf{a}$.

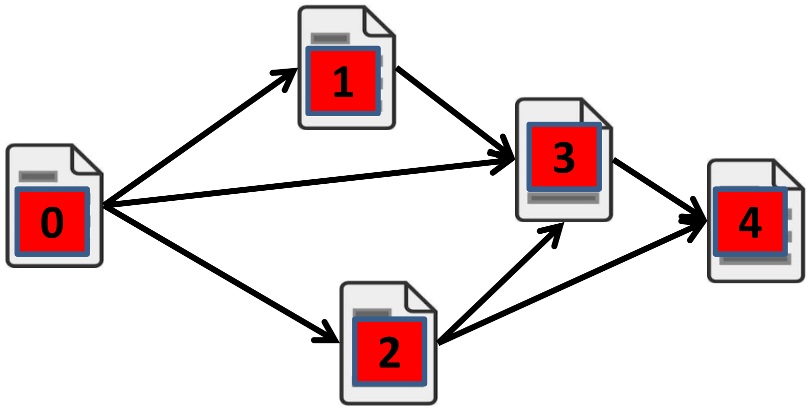
Power iteration algorithm

```
Let x=(0,…,0) and x′ an initial
  state, say (1,0,…,0)

while ( |x-x′| > ε ):
     x = x′
     x′= xG
```

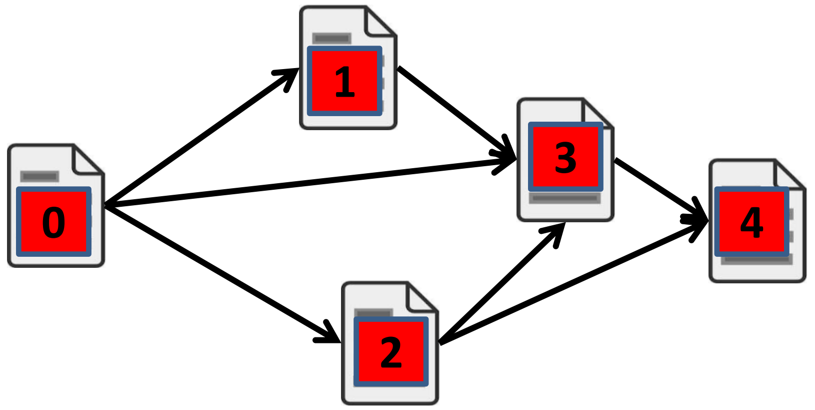
This algorithm converges **very** slowly.

Example



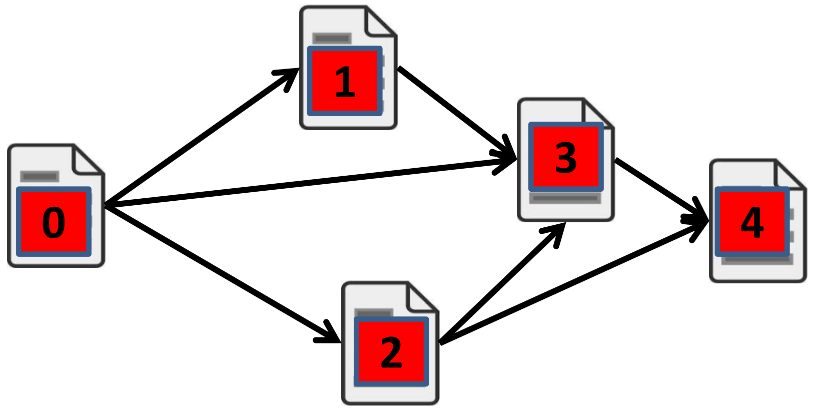
	t=0	t=1
0	1	.030
1	0	.313
2	0	.313
3	0	.313
4	0	.030

Example



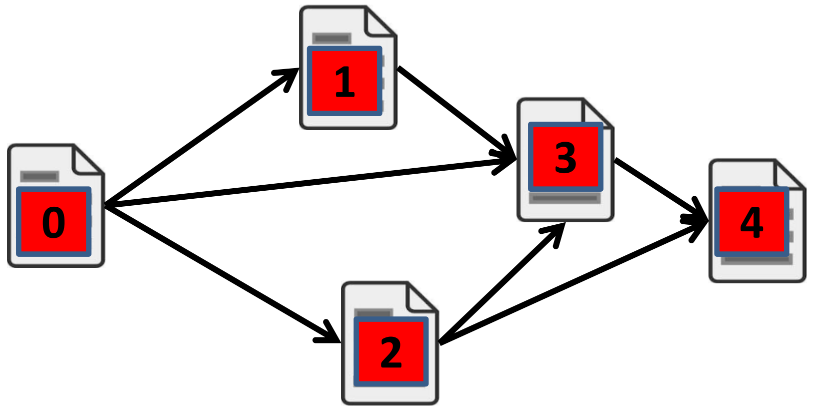
	t=0	t=1	t=2
0	1	.030	.035
1	0	.313	.044
2	0	.313	.044
3	0	.313	.443
4	0	.030	.435

Example



	t=0	t=1	t=2	t=3	t=4	t=5		t=16
0	1	.030	.035	.104	.115	.082	…	.095
1	0	.313	.044	.114	.144	.115	…	.122
2	0	.313	.044	.114	.144	.115	…	.122
3	0	.313	.443	.169	.289	.299	…	.278
4	0	.030	.435	.499	.307	.390	…	.383

Example



- Stationary probabilities:
  (0.095, 0.122, 0.122, 0.278, 0.383)

  found after 16 iterations starting from
  (1, 0, 0, 0, 0)

Example

- Stationary probabilities:
  x = (0.095, 0.122, 0.122, 0.278, 0.383)
- Check: xG =

$$
(0.095 \quad 0.122 \quad 0.122 \quad 0.278 \quad 0.383)
\begin{bmatrix}
0.0300 & 0.3105 & 0.3105 & 0.3105 & 0.0300 \\
0.0300 & 0.0300 & 0.0300 & 0.8800 & 0.0300 \\
0.0300 & 0.0300 & 0.0300 & 0.4550 & 0.4550 \\
0.0300 & 0.0300 & 0.0300 & 0.0300 & 0.8800 \\
0.2000 & 0.2000 & 0.2000 & 0.2000 & 0.2000
\end{bmatrix}
$$

$$= (0.0951 \quad 0.1218 \quad 0.1218 \quad 0.2773 \quad 0.3833)$$

- So x is (approximately) the left eigenvector of G.

Normalization

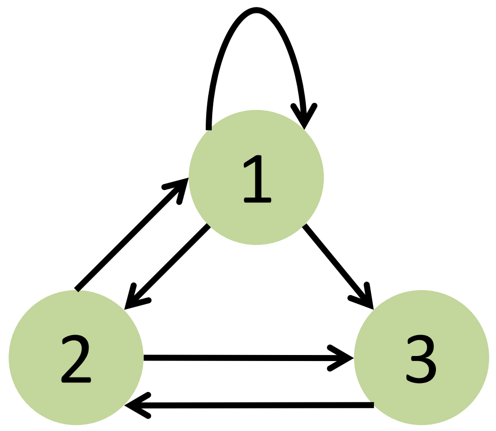
x = (0.095, 0.122, 0.122, 0.278, 0.383)

- In every iteration, $\sum_{i=1}^{n} x_i = 1$ ideally
- Due to round-off errors, this is not always the case. Therefore, it might be a good idea to compute

  sum = $\sum_{i=1}^{n} x_i$; $\forall i \; x_i := \frac{x_i}{sum}$

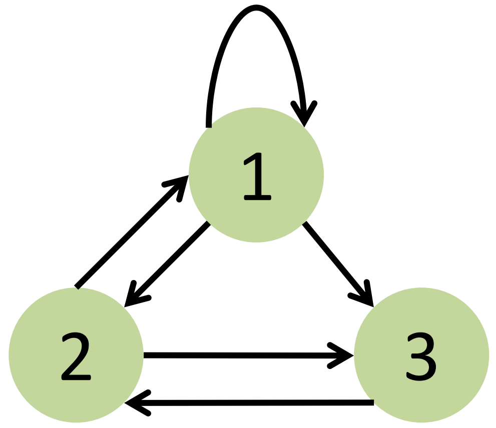
  in every iteration.

Quiz



What is the G matrix for this link graph?

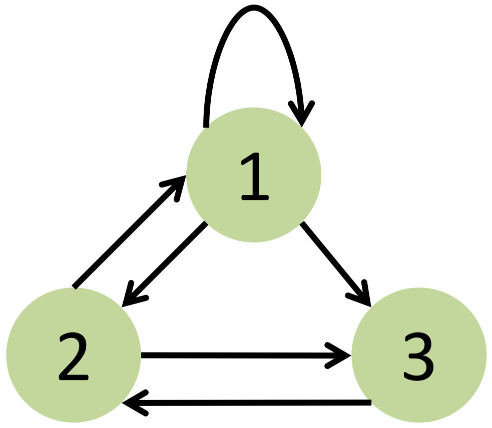
Quiz



What is the G matrix for this link graph?

$$\begin{bmatrix} 0.33 & 0.33 & 0.33 \\ & & \\ & & \end{bmatrix}$$

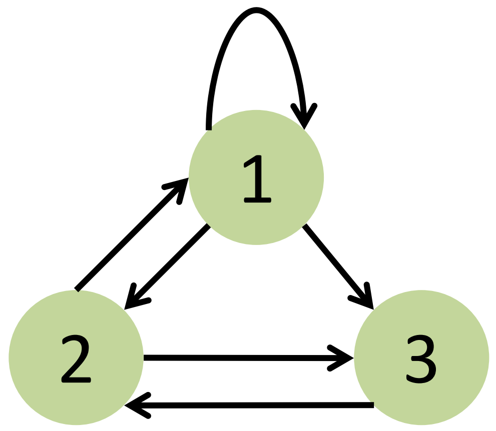
Quiz



What is the G matrix for this link graph?

$$\begin{bmatrix} 0.33 & 0.33 & 0.33 \\ 0.475 & 0.05 & 0.475 \\ & & \end{bmatrix}$$

Quiz



What is the G matrix for this link graph?

$$\begin{bmatrix} 0.33 & 0.33 & 0.33 \\ 0.475 & 0.05 & 0.475 \\ 0.05 & 0.9 & 0.05 \end{bmatrix}$$

Approximating PageRank

Approximating PageRank

- For large graphs, power iteration takes a looooong time.
- We can also **approximate** PageRank by **simulating the random surfer**.

Monte Carlo simulation to approximate PageRank

- Avrachenkov et al. describe a number of simulation algorithm based on the random surfer model
- D = document id
- Consider a random walk $\{D_t\}_{t \geq 0}$ that starts from some page.
- At each step t:
  - Prob c: D_t = one of the documents with edges from $D_{t\text{-}1}$
  - Prob $(1 - c)$: The random walk terminates

1. MC end-point with random start

- Simulate N runs of the random walk $\{D_t\}_{t\geq 0}$ initiated at a **randomly chosen page**
- PageRank of page j = 1,...,n:

  π_j = (#walks which end at j)/N
- **Worst case: N = $O(n^2)$**
- **Mean case: N = $O(n)$**

2. MC end-point with cyclic start

- Simulate N = mn runs of the random walk $\{D_t\}_{t \geq 0}$ initiated at **each page exactly m times**
- PageRank of page j = 1,...,n:

$$\boldsymbol{\pi_j = (\#\text{walks which end at } j)/N}$$

4. MC complete path stopping at dangling nodes

- Simulate N = mn runs of the random walk $\{D_t\}_{t \geq 0}$ initiated at **each page exactly m times** and **stopping when it reaches a dangling node**
- PageRank of page j = 1,...,n:

π_j = (#visits to node j during walks)/ (total #visits during walks)

5. MC complete path with random start

- Simulate N runs of the random walk $\{D_t\}_{t \geq 0}$ initiated at a **randomly chosen page** and **stopping when it reaches a dangling node**
- PageRank of page j = 1,...,n:

  π_j = (#visits to node j during walks)/(total #visits during walks)

Monte Carlo advantages

- **Power interation:**
  - precision improves linearly for all docs
  - computationally expensive
  - must be redone when new pages are added

- **Monte Carlo method:**
  - precision improves faster for high-rank docs
  - parallel implementation possible
  - continuous update

Assignment 2

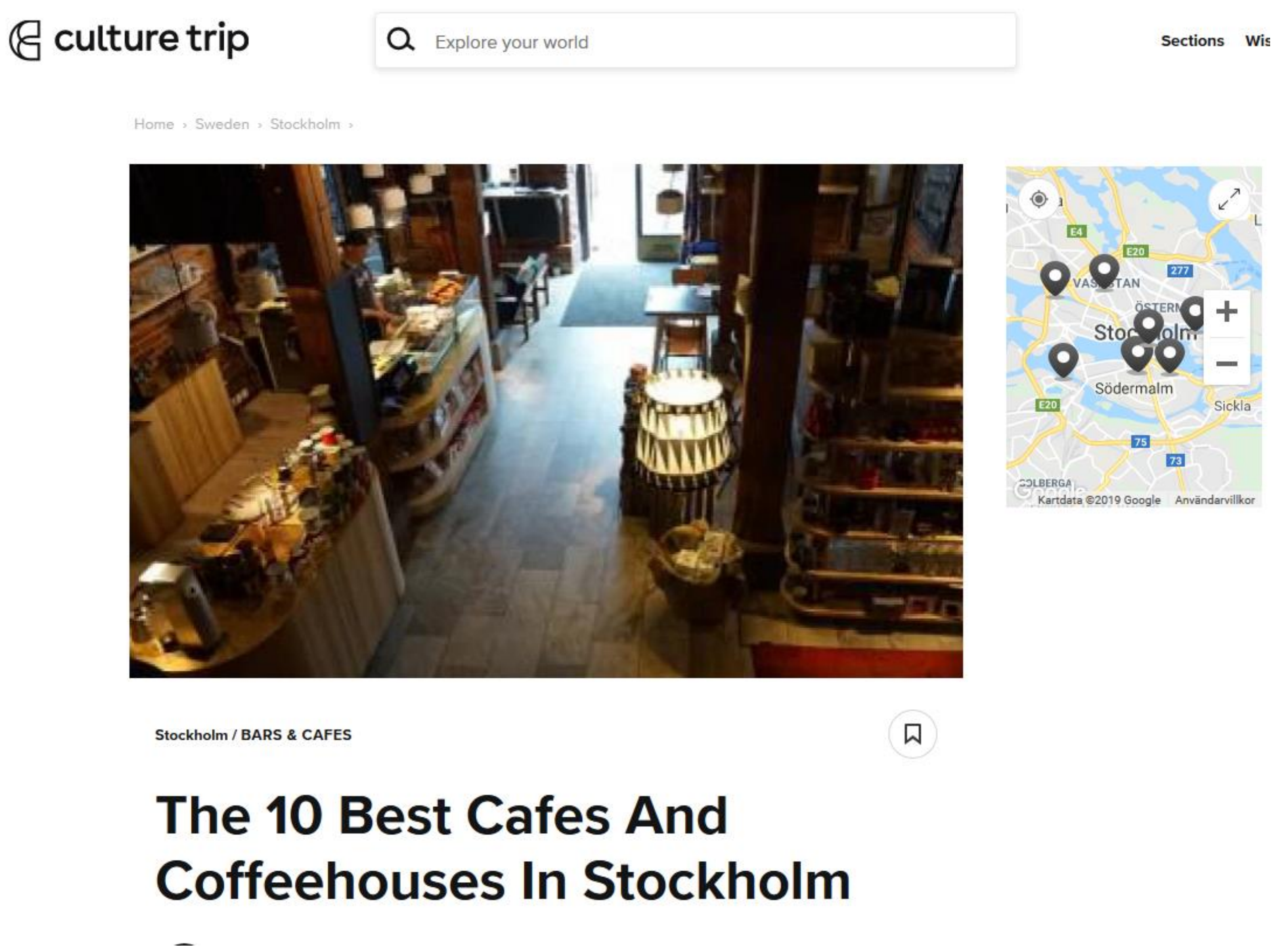
- 2.5 Implement power iteration
  Combine tf-idf and pagerank
- 2.7 Implement Monte-Carlo pagerank computations
  Estimate pageranks of linksDavis.txt
  Estimate pageranks of linksWiki.txt
- 2.8 Hubs and authorities

Hubs and authorities

HITS (Hypertext-Induced Topic Selection)

- HITS is an method similar to PageRank
  - proposed the same year (1998)
- Goal is to find high-quality pages of two kinds:
  - **Hubs**: linking to pages with great authority
  - **Authorities**: linked from great hubs

A typical hub



culture trip

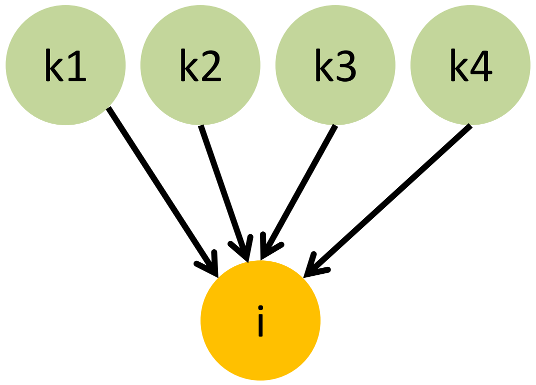
Home › Sweden › Stockholm ›

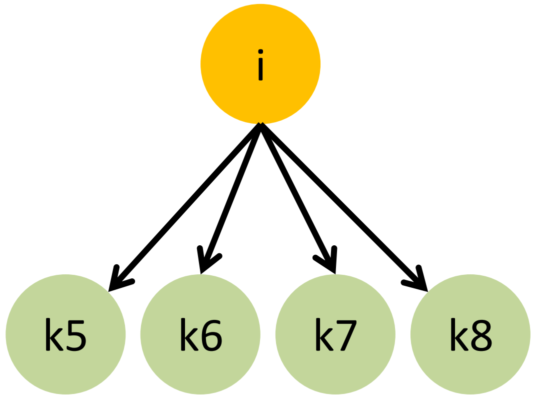
Stockholm / BARS & CAFES

The 10 Best Cafes And Coffeehouses In Stockholm

Hubs and authorities

- Each page *i* has two scores:
  - authority score a_i
  - hub score h_i
- HITS algorithm: Initialize $a_i=1$ $h_i=1$
- Then iterate until convergence:
  - Authority: $\forall \mathrm{i} : a_i := \sum_{\mathrm{k}\to\mathrm{i}} h_k$
  - Hub: $\forall \mathrm{i} : h_i := \sum_{\mathrm{i}\to\mathrm{k}} a_k$
  - Normalize $\sqrt{\sum_i a_i^2} := 1 \quad \sqrt{\sum_i h_i^2} := 1$





Matrix formulation

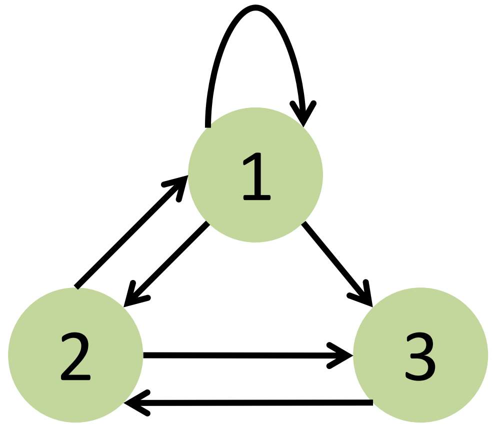
- $a = (a_1, \ldots, a_n)$ $h=(h_1,\ldots, h_n)$ are authority/hub scores for n documents
- A is the **adjacency matrix**: $A_{ik} = 1$ iff $i \rightarrow k$
- Then do:

$$h_i := \sum_{i \rightarrow k} a_k \Leftrightarrow h_i := \sum_k A_{ik} a_k$$

- i.e. $h := Aa$
- and likewise $a := A^T h$

HITS algorithm (example)

$$A = \begin{bmatrix} 1 & 1 & 1 \\ 1 & 0 & 1 \\ 0 & 1 & 0 \end{bmatrix} \qquad A^T = \begin{bmatrix} 1 & 1 & 0 \\ 1 & 0 & 1 \\ 1 & 1 & 0 \end{bmatrix}$$

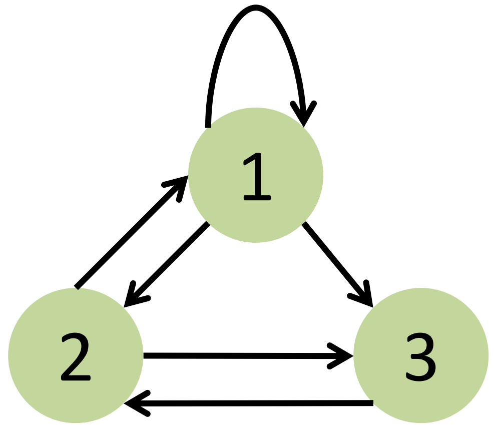


a(1)	1					
a(2)	1					
a(3)	1					

h(1)	1					
h(2)	1					
h(3)	1					

HITS algorithm (example)

$$A = \begin{bmatrix} 1 & 1 & 1 \\ 1 & 0 & 1 \\ 0 & 1 & 0 \end{bmatrix} \qquad A^T = \begin{bmatrix} 1 & 1 & 0 \\ 1 & 0 & 1 \\ 1 & 1 & 0 \end{bmatrix}$$



a= A^{T}h

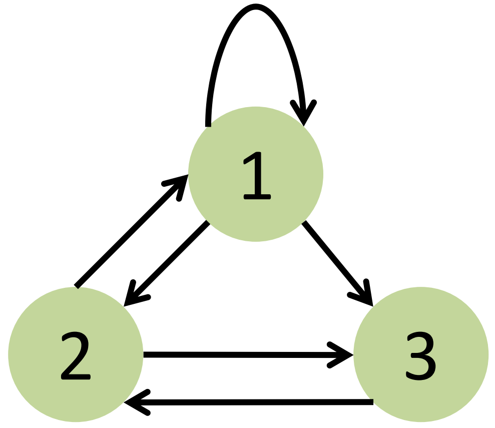
a(1)	1	2				
a(2)	1	2				
a(3)	1	2				

h= Aa

h(1)	1	3				
h(2)	1	2				
h(3)	1	1				

HITS algorithm (example)

$$A = \begin{bmatrix} 1 & 1 & 1 \\ 1 & 0 & 1 \\ 0 & 1 & 0 \end{bmatrix} \qquad A^T = \begin{bmatrix} 1 & 1 & 0 \\ 1 & 0 & 1 \\ 1 & 1 & 0 \end{bmatrix}$$



Normalize

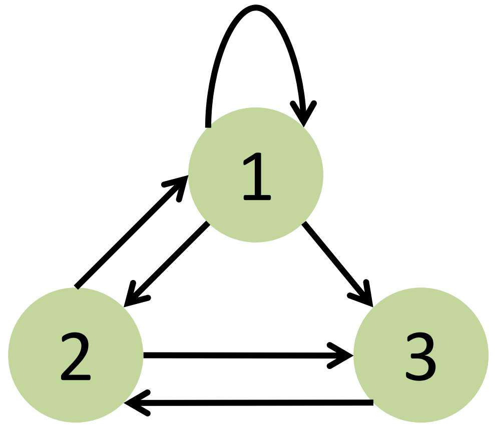
a(1)	1	2	2/√12=0.577		
a(2)	1	2	2/√12=0.577		
a(3)	1	2	2/√12=0.577		

Normalize

h(1)	1	3	3/√14 = 0.802		
h(2)	1	2	2/√14=0.535		
h(3)	1	1	1/√14=0.267		

HITS algorithm (example)

$$A = \begin{bmatrix} 1 & 1 & 1 \\ 1 & 0 & 1 \\ 0 & 1 & 0 \end{bmatrix} \qquad A^T = \begin{bmatrix} 1 & 1 & 0 \\ 1 & 0 & 1 \\ 1 & 1 & 0 \end{bmatrix}$$



a= A^Th, then normalize

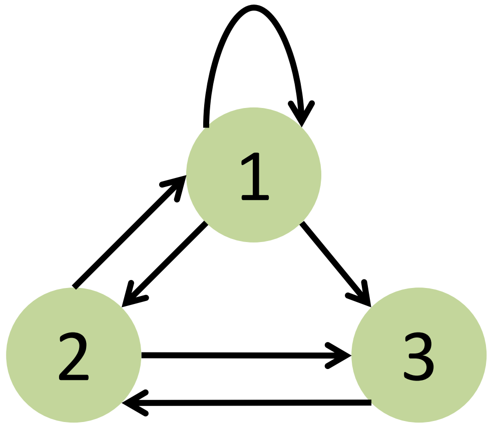
a(1)	1	0.577	0.615			
a(2)	1	0.577	0.492			
a(3)	1	0.577	0.615			

h= Aa, then normalize

h(1)	1	0.802	0.793			
h(2)	1	0.535	0.566			
h(3)	1	0.267	0.226			

HITS algorithm (example)

$$A = \begin{bmatrix} 1 & 1 & 1 \\ 1 & 0 & 1 \\ 0 & 1 & 0 \end{bmatrix} \qquad A^T = \begin{bmatrix} 1 & 1 & 0 \\ 1 & 0 & 1 \\ 1 & 1 & 0 \end{bmatrix}$$



a(1)	1	0.577	0.615	…	…	0.627
a(2)	1	0.577	0.492	…	…	0.460
a(3)	1	0.577	0.615	…	…	0.627

h(1)	1	0.802	0.793	…	…	0.789
h(2)	1	0.535	0.566	…	…	0.577
h(3)	1	0.267	0.226	…	…	0.211

Convergence

Results of the HITS algorithm

- When we have reached convergence, we have:

  $h = c_h A a$ and $a = c_a A^T h$

  where c_h and c_a are scalars.

  i.e. $h = \lambda_h A A^T h$ and $a = \lambda_a A^T A a$

  where h is the eigenvector of AA^T, and λ_h is the eigenvalue.
  Likewise, a is the eigenvector of A^TA, and λ_a eigenvalue.

Results of the HITS algorithm

- Check:

$$AA^T h = \begin{bmatrix} 1 & 1 & 1 \\ 1 & 0 & 1 \\ 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} 1 & 1 & 0 \\ 1 & 0 & 1 \\ 1 & 1 & 0 \end{bmatrix} \begin{bmatrix} 0.789 \\ 0.577 \\ 0.211 \end{bmatrix} = \begin{bmatrix} 3.732 \\ 2.732 \\ 0.999 \end{bmatrix} = 4.73 \begin{bmatrix} 0.789 \\ 0.577 \\ 0.211 \end{bmatrix}$$

i.e. *h* is an eigenvector of AA^T, with corresponding eigenvalue 4.73.

Using the HITS algorithm

- Unlike PageRank, the hubs and authorities scores are **computed at query time**
- Given a query q:
  - the **root set** are all documents that include at least one query word
  - the **base set** are all documents linking to, or linked from, a root doucument
- We then use the base set for computing the hubs and authorities scores