More About PageRank

Combatting Web Spam
Dealing with Non-Main-Memory Web Graphs
SimRank

Jeffrey D. Ullman
Stanford University/Infolab

Web Spam

Term Spamming
Link Spamming

What Is Web Spam?

- *Spamming* = any deliberate action intended solely to boost a Web page's position in search-engine results.
- *Web Spam* = Web pages that are the result of spamming.
- SEO industry might disagree!
  - *SEO* = search engine optimization

Web Spam Taxonomy

- *Boosting* techniques.
  - Techniques for making a Web page appear to be a good response to a search query.
- *Hiding* techniques.
  - Techniques to hide the use of boosting from humans and Web crawlers.

Boosting

- *Term spamming*.
  - Manipulating the text of web pages in order to appear relevant to queries.
- *Link spamming*.
  - Creating link structures that boost PageRank.

Term-Spamming Techniques

- *Repetition* of terms, e.g., “Viagra,” in order to subvert TF.IDF-based rankings.
- *Dumping* = adding large numbers of words to your page.
  - Example: run the search query you would like your page to match, and add copies of the top 10 pages.
  - Example: add a dictionary, so you match every search query.
  - Key hiding technique: words are hidden by giving them the same color as the background.

Link Spam

Design of a Spam Farm
TrustRank
Spam Mass

Link Spam

- PageRank prevents spammers from using term spam to fool a search engine.
  - While spammers can still use the techniques, they cannot get a high-enough PageRank to be in the top 10.
- Spammers now attempt to fool PageRank with *link spam* by creating structures on the Web, called *spam farms*, that increase the PageRank of undeserving pages.

Building a Spam Farm

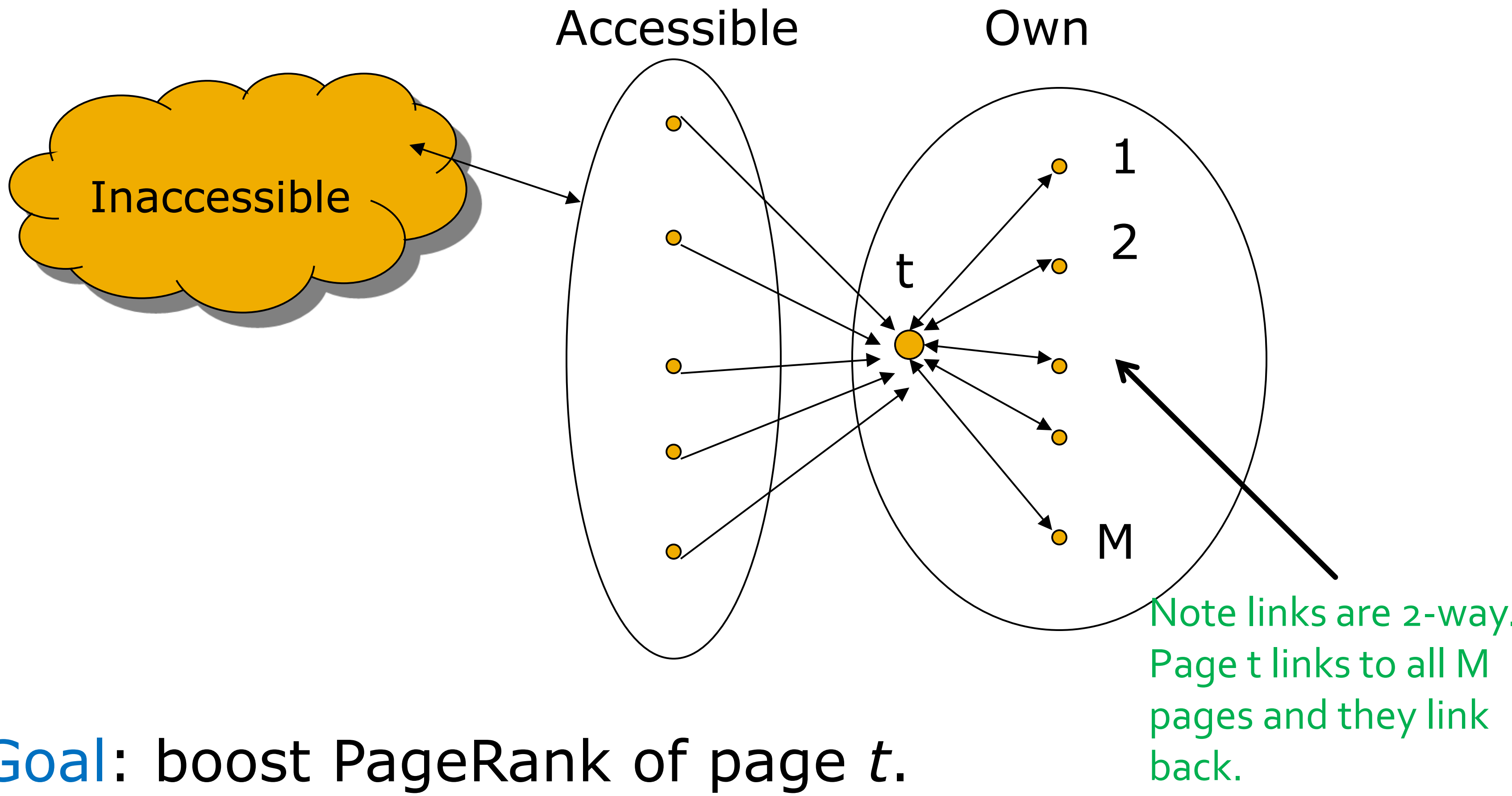
- Three kinds of Web pages from a spammer's point of view:

1. *Own pages*.
   - Completely controlled by spammer.
2. *Accessible pages*.
   - E.g., Web-log comment pages: spammers can post links to their pages.
     - "I totally agree with you. Here's what I wrote about the subject at www.MySpamPage.com."
3. *Inaccessible pages*.
   - Everything else.

Spam Farms – (2)

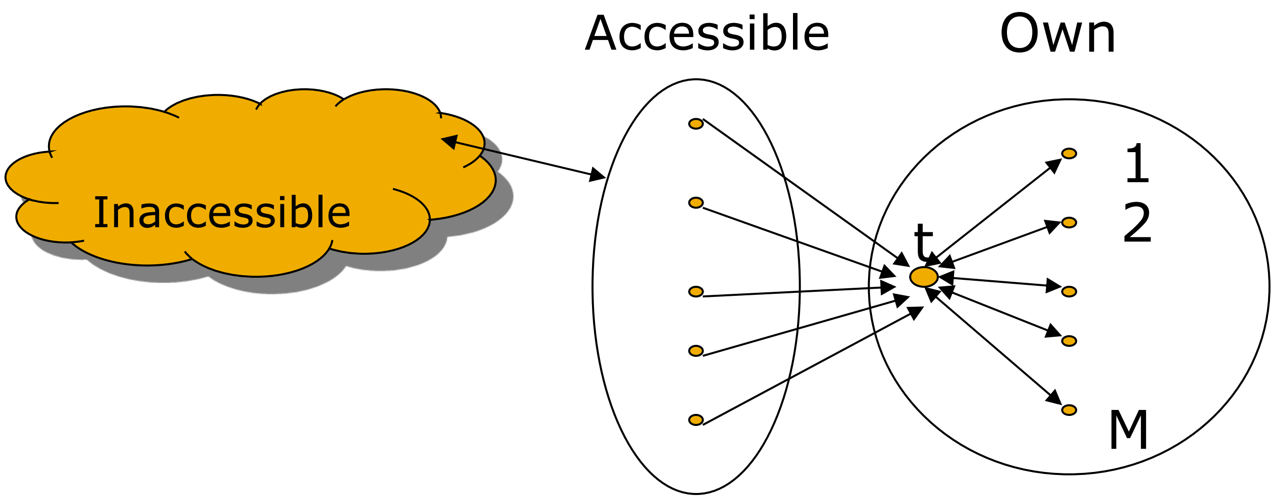
- Spammer's goal:
  - Maximize the PageRank of target page *t*.
- Technique:
  1. Get as many links as possible from accessible pages to target page *t*.
     - Note: if there are none at all, then search engines will not even be aware of the existence of page t.
  2. Construct a spam farm to get a PageRank-multiplier effect.

Spam Farms – (3)



Goal: boost PageRank of page *t*.
Here is one of the most common and effective organizations for a spam farm.

Analysis



Suppose rank from accessible pages = *x* (known).

PageRank of target page = *y* (unknown).

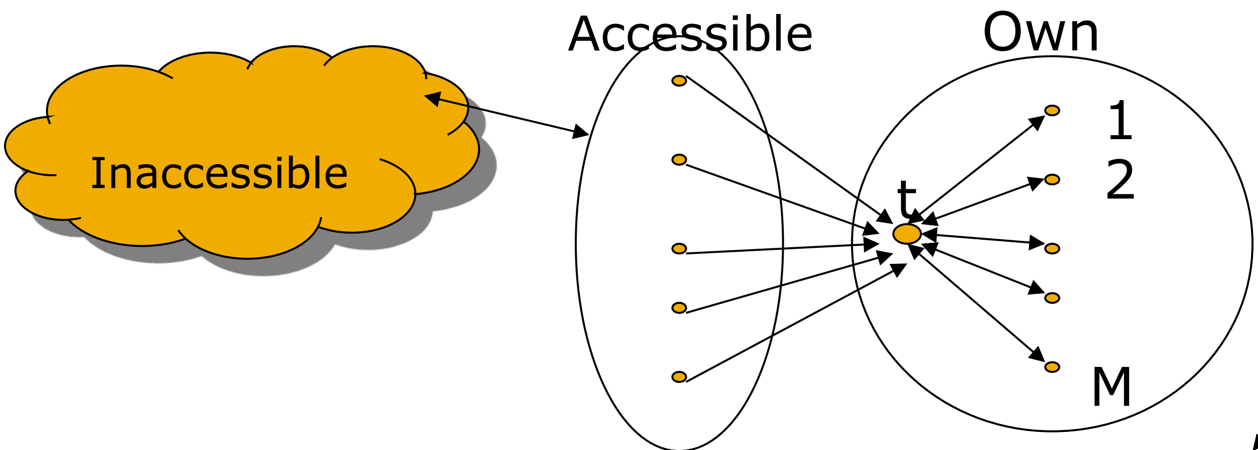
Taxation rate = 1-β.

Rank of each "farm" page = $\boxed{\beta y/M} + \boxed{(1-\beta)/N}$.

From *t*; M = number of farm pages

Share of "tax"; N = size of the Web. Total PageRank = 1.

Analysis – (2)



Tax share for *t*.
Very small; ignore.

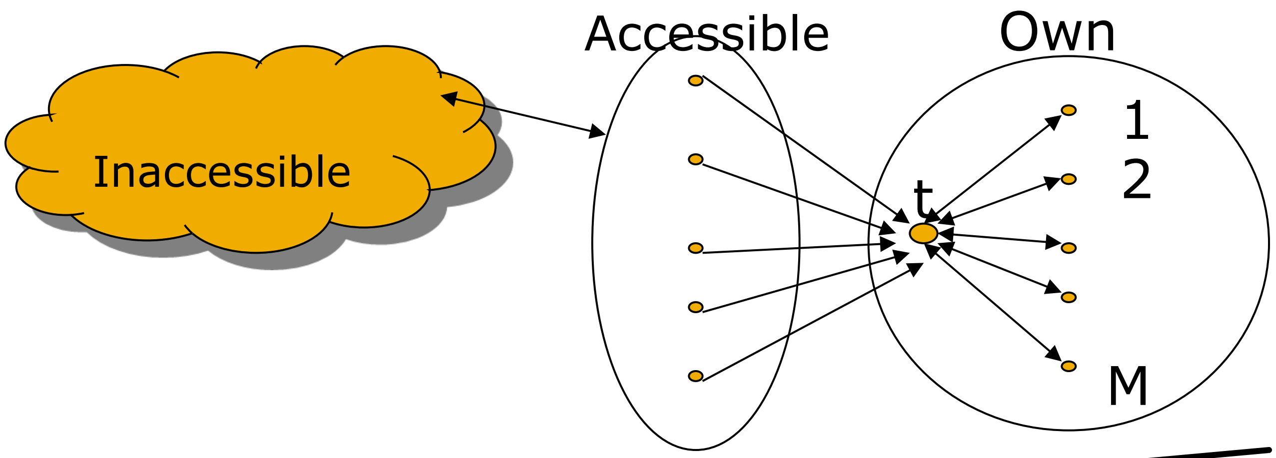
$$y = x + \beta M[\boxed{\beta y/M + (1-\beta)/N}] + \boxed{(1-\beta)/N}$$

$$y = x + \beta^2 y + \beta(1-\beta)M/N$$

$$y = x/(1-\beta^2) + cM/N \text{ where } c = \beta/(1+\beta)$$

PageRank of
each "farm" page

Analysis – (3)



Average page has PageRank 1/N. c is about ½, so this term gives you M/2 times as much PageRank as average.

- $y = x/(1-\beta^2) + cM/N$ where $c = \beta/(1+\beta)$.
- For $\beta = 0.85$, $1/(1-\beta^2) = 3.6$.
  - Multiplier effect for "acquired" page rank.
- By making M large, we can make *y* almost as large as we want.

Question for Thought:
What if $\beta = 1$ (i.e., no tax)?

War Between Spammers and Search Engines

- If you design your spam farm just as was described, Google will notice it and drop it from the Web.
- More complex designs might be undetected, although SEO innovations are tracked by Google et al.
- Fortunately, there are other techniques for combatting spam that do not rely on direct detection of spam farms.

Detecting Link Spam

- Topic-specific PageRank, with a set of "trusted" pages as the teleport set is called *TrustRank*.
- *Spam Mass* = (PageRank – TrustRank)/PageRank.
  - High spam mass means most of your PageRank comes from untrusted sources – you may be link-spam.

Picking the Trusted Set

- Two conflicting considerations:
  - Human may have to inspect each trusted page, so this set should be as small as possible.
  - Must ensure every “good page” gets adequate TrustRank, so all good pages should be reachable from the trusted set by short paths.
    - Implies that the trusted set must be geographically diverse, hence large.

Approaches to Picking the Trusted Set

1. Pick the top *k* pages by PageRank.
   - It is almost impossible to get a spam page to the very top of the PageRank order.
2. Pick the home pages of universities.
   - Domains like .edu are controlled.

- Notice that both these approaches avoid the requirement for human intervention and (probably) provide adequate distribution.

Efficiency Considerations for PageRank

Multiplication of Huge Vector and Matrix

Representing Blocks of a Stochastic Matrix

The Problem

- Google computes the PageRank of a trillion pages (at least!).
- The PageRank vector of double-precision reals requires 8 terabytes.
  - And another 8 terabytes for the next estimate of PageRank.

The Problem – (2)

- The matrix of the Web has two special properties:
  1. It is very sparse: the average Web page has about 10 out-links.
  2. Each column has a single value – 1 divided by the number of out-links – that appears wherever that column is not 0.

The Problem – (3)

- Trick: for each column, store n = the number of out-links and a list of the rows with nonzero values (which must be 1/n).
- Thus, the matrix of the Web requires at least $(\boxed{4*1}+\boxed{8*10})*10^{12}$ = 84 terabytes.

Integer n

Average 10 links/column,
8 bytes per row number.

The Solution: Blocking

- Divide the current and next PageRank vectors into k *blocks* of equal size.
  - Each block is the components in some consecutive rows.
- Divide the matrix into squares whose sides are the same length as one of the blocks.
- Pick k large enough to fit a block of each vector in main memory at the same time.
  - Note: We also need a square of the matrix, but that can be piped through main memory and won't use much memory at any time.

Example: k = 3

$$
\begin{bmatrix} w1 \\ w2 \\ w3 \end{bmatrix} = \begin{bmatrix} M11 & M12 & M13 \\ M21 & M22 & M23 \\ M31 & M32 & M33 \end{bmatrix} \begin{bmatrix} v1 \\ v2 \\ v3 \end{bmatrix}
$$

At one time, we need wi, vj, and (a tiny part of) Mij in memory.

Vary v slowest: w1 = M11 v1; w2 = M21 v1; w3 = M31 v1; w1 += M12 v2; w2 += M22 v2; w3 += M32 v2; w1 += M13 v3; w2 += M23 v3; w3 += M33 v3

Representing a Matrix Square

- Each column of a square is represented by:
  1. The number n of nonzero elements in the entire column of the matrix (i.e., the total number of out-links for the corresponding Web page).
  2. The list of rows of that square only that have nonzero values (which must be 1/n).
- I.e., for each column, we store n with each of the k squares in one column of the matrix and each out-link with whatever square has the row to which the link goes.

Representing a Square – (2)

- Total space to represent the matrix = $(\boxed{4*k+}\boxed{8*10})*10^{12}$ = 4k+80 terabytes.

Integer n for a column is represented in each of k squares. Possible savings: if a square has all 0's in a column, then n is not Needed for that column.

Average 10 links/column, 8 bytes per row number, spread over k squares.

Note: if 10 is the average number of out-links, then there will be integers in an average of 10 squares for each column, so k can be thought of as the maximum of 10 and the number of blocks.

Needed Modifications

- We are not just multiplying a matrix and a vector.
- We need to multiply the result by a constant to reflect the "taxation."
- We need to add a constant to each component of the result **w**.
- Neither of these changes are hard to do.
  - After computing each block w_i of **w**, multiply by β and then add $(1-\beta)/N$ to each component.

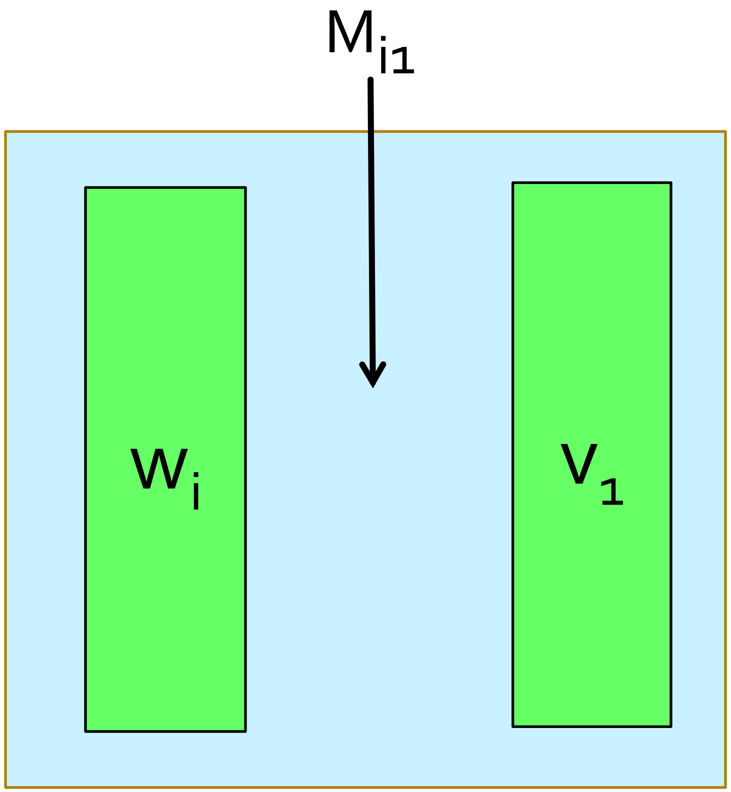
Parallelization

- The strategy described can be executed on a single machine.
- But who would want to?
- There is a simple MapReduce algorithm to perform matrix-vector multiplication.
  - But since the matrix is sparse, better to treat it as a relational join.

Parallelization – (2)

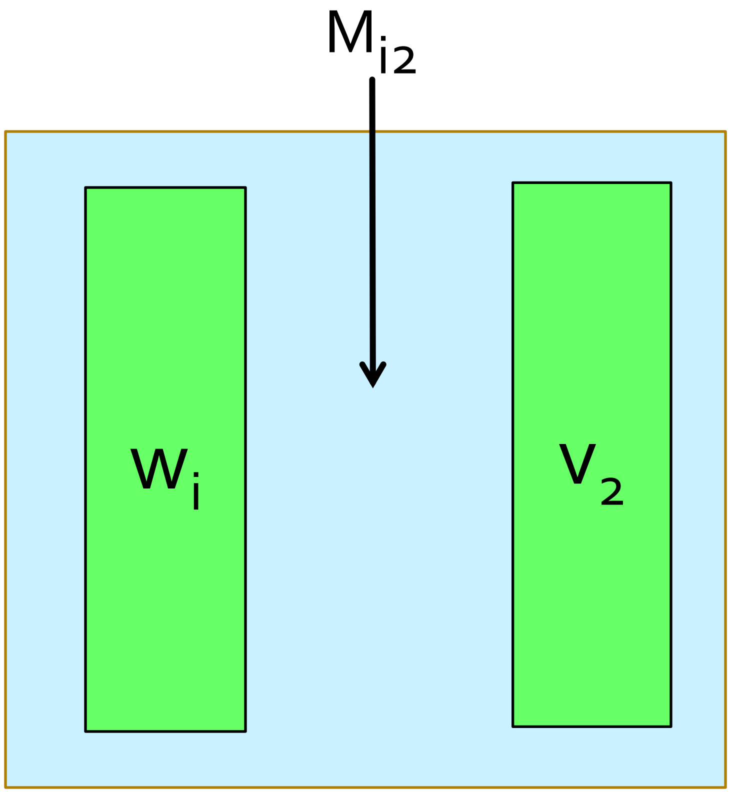
- Another approach is to use many jobs, each to multiply a row of matrix squares by the entire **v**.
- Use main memory to hold the one block of **w** that will be produced.
- Read one block of **v** into main memory at a time.
- Read the square of M that needs to multiply the current block of **v**, a tiny bit at a time.
- Works as long as k is large enough that two blocks fit in memory.
- M read once; **v** read k times, among all the jobs.
  - OK, because M is much larger than **v**.

Animation: First Block of v



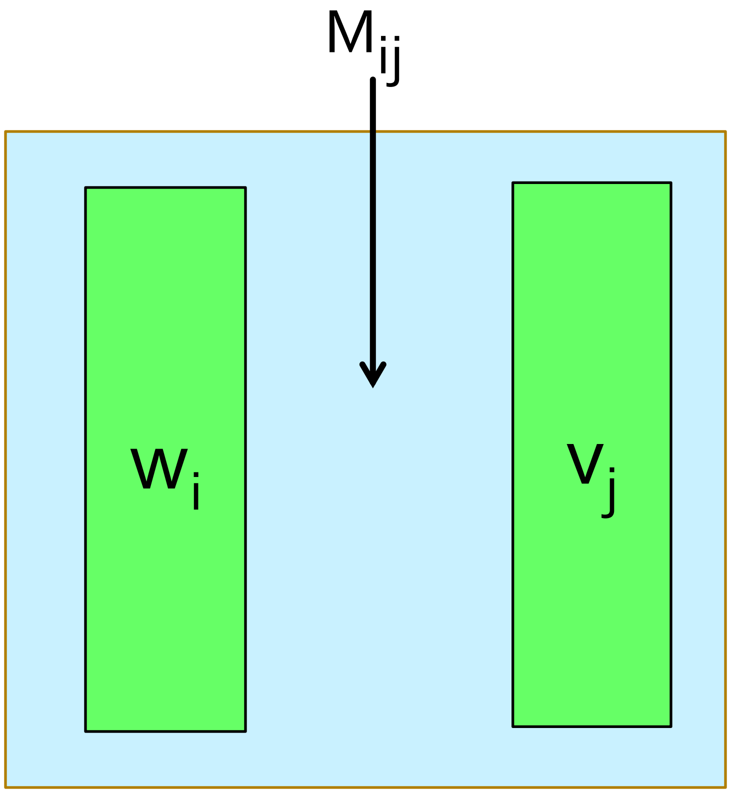
Main Memory for job i

Animation: Second Block of v



Main Memory for job i

Animation: j-th Block of v



Main Memory for job i

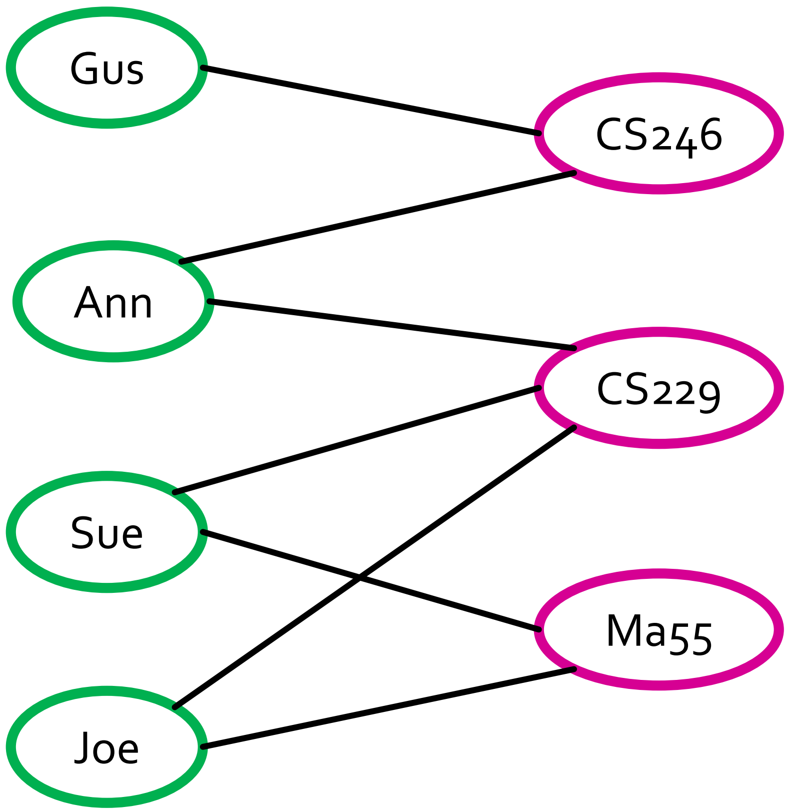
SimRank

Graphs of Entities and Connections
Finding Similar Entities by Random Walks

Similiarity in Networks

- Unlike similarity based on a distance measure, which we discussed with regard to LSH, we may instead wish to look for entities that play similar roles in a complex network.
- Example: Nodes represent students and classes; find students with similar interests, classes on similar subjects.

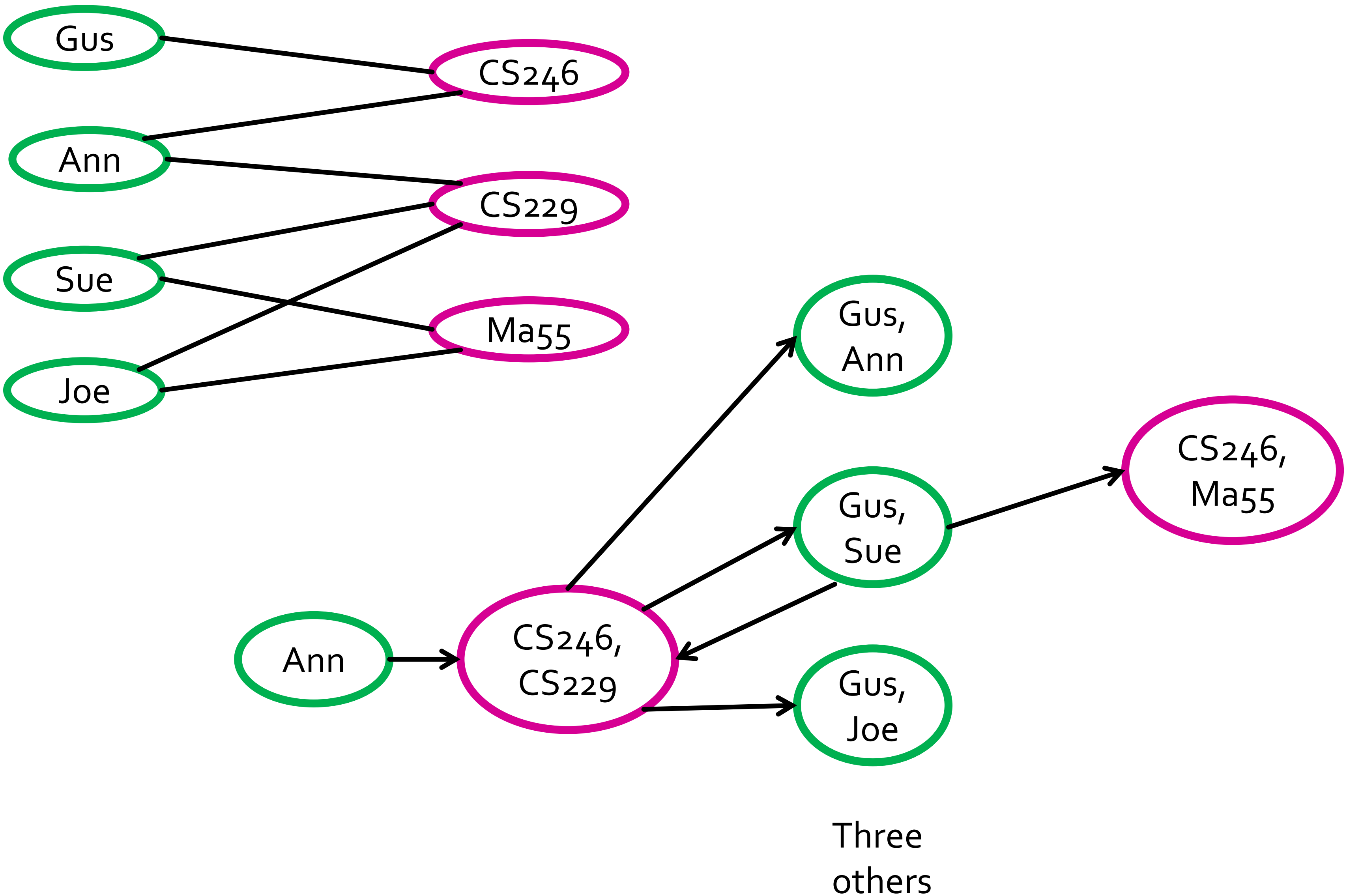
Example: Network



Approach: Pair Graphs

- Intuition:
  1. An entity is similar to itself.
  2. If two entities A and B are similar, then that is some evidence that entities C and D connected to A and B, respectively, are similar.

Example: Pair Graph



Using Pair Graphs

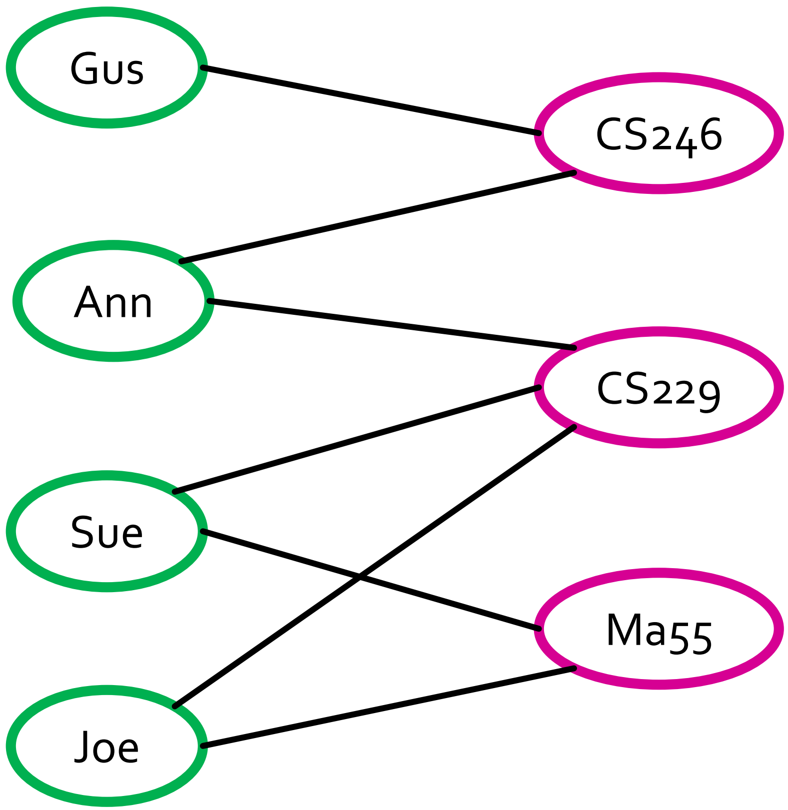
- You can run Topic-Sensitive PageRank on such a graph, with the nodes representing single entities as the teleport set.
- Resulting PageRank of a node measures how similar the two entities are.
- A high tax rate may be appropriate, or else you conclude things like CS246 is similar to Hist101.
- Problem: Using node pairs squares the number of nodes.
  - Can be too large, even for university-sized data.

Alternative: SimRank

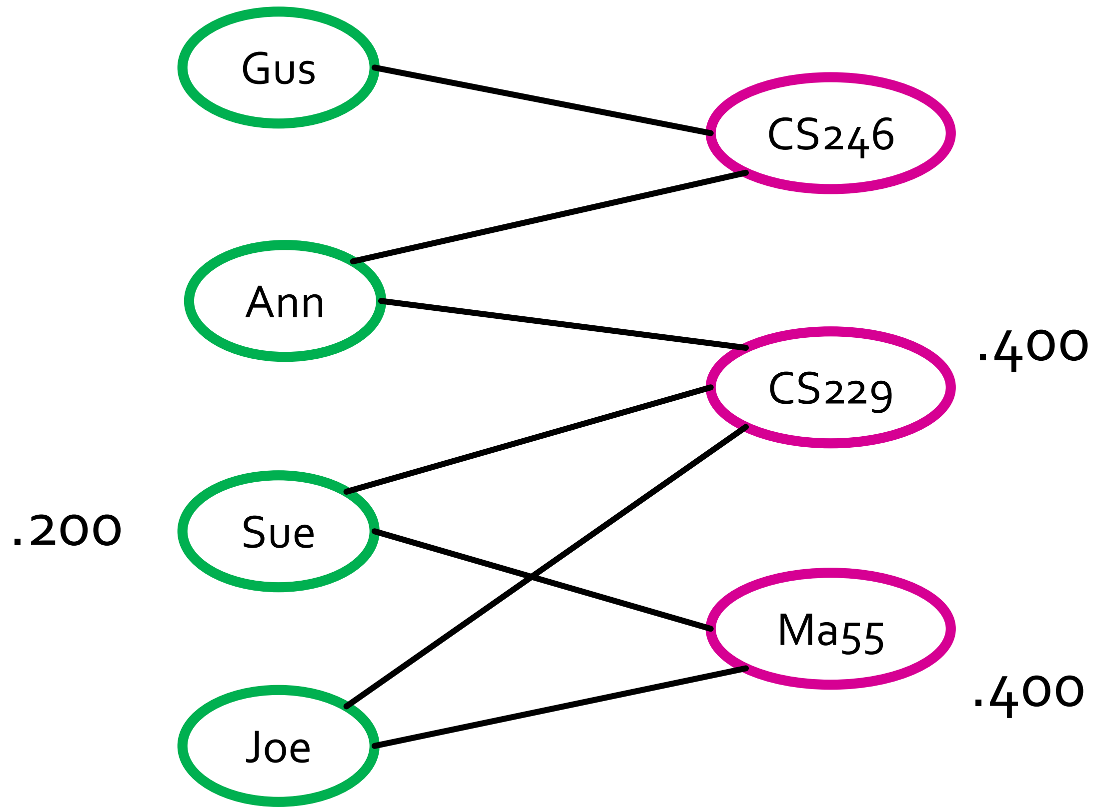
- Another approach is to work from the original network.
- Treat undirected edges as arcs or links in both directions.
- Find the entities similar to a single entity, which becomes the sole member of the teleport set.
- Example: "Who is similar to Sue?" on next slides.
- Allows us to work on the original graph rather than on pairs.
  - But we need to run many searches (in parallel?).

Example: SimRank

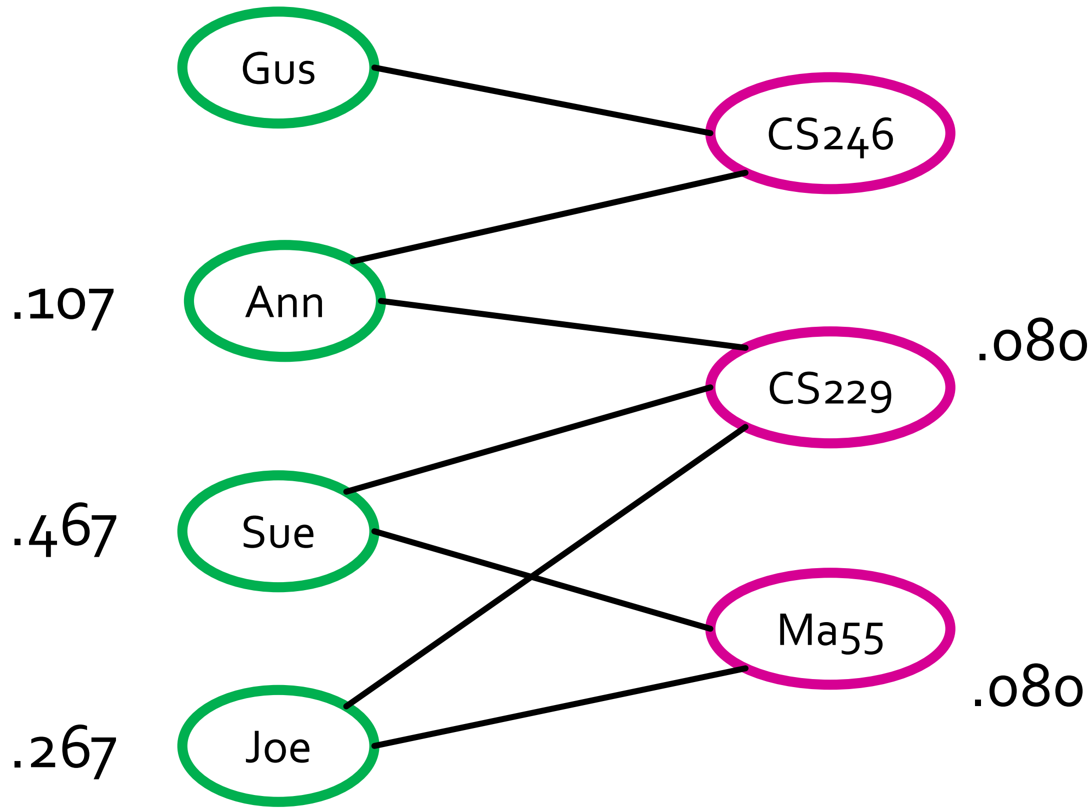
1.000



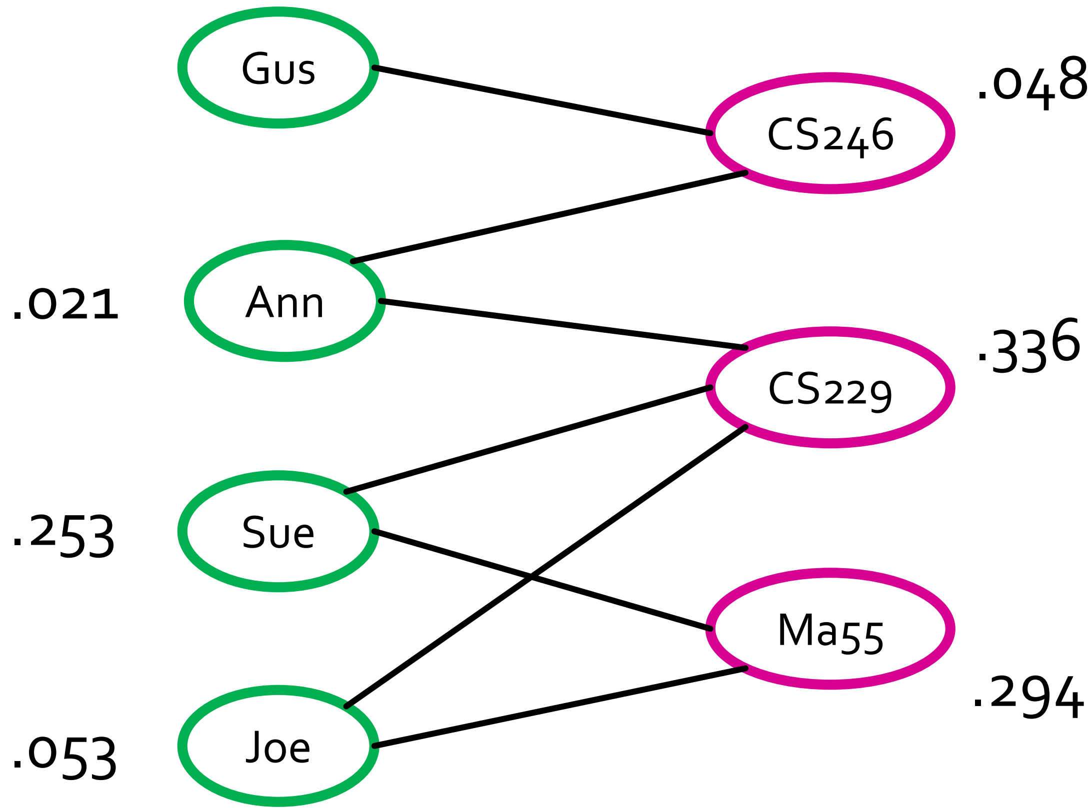
Example: SimRank (20% Tax)



Example: SimRank



Example: SimRank



Example: SimRank