CSE 5243 INTRO. TO DATA MINING

Locality Sensitive Hashing (LSH) & Graph Data

Huan Sun, CSE@The Ohio State University

Slides adapted from Prof. Jiawei Han @UIUC, Prof. Srinivasan Parthasarathy @OSU

Min-Hashing Example

Permutation π

2	4	3
3	2	4
7	1	7
6	3	2
1	6	6
5	7	1
4	5	5

Input matrix (Shingles x Documents)

1	0	1	0
1	0	0	1
0	1	0	1
0	1	0	1
0	1	0	1
1	0	1	0
1	0	1	0

Signature matrix *M*

2	1	2	1
2	1	4	1
1	2	1	2

Similarities:

	1-3	2-4	1-2	3-4
Col/Col	0.75	0.75	0	0
Sig/Sig	0.67	1.00	0	0

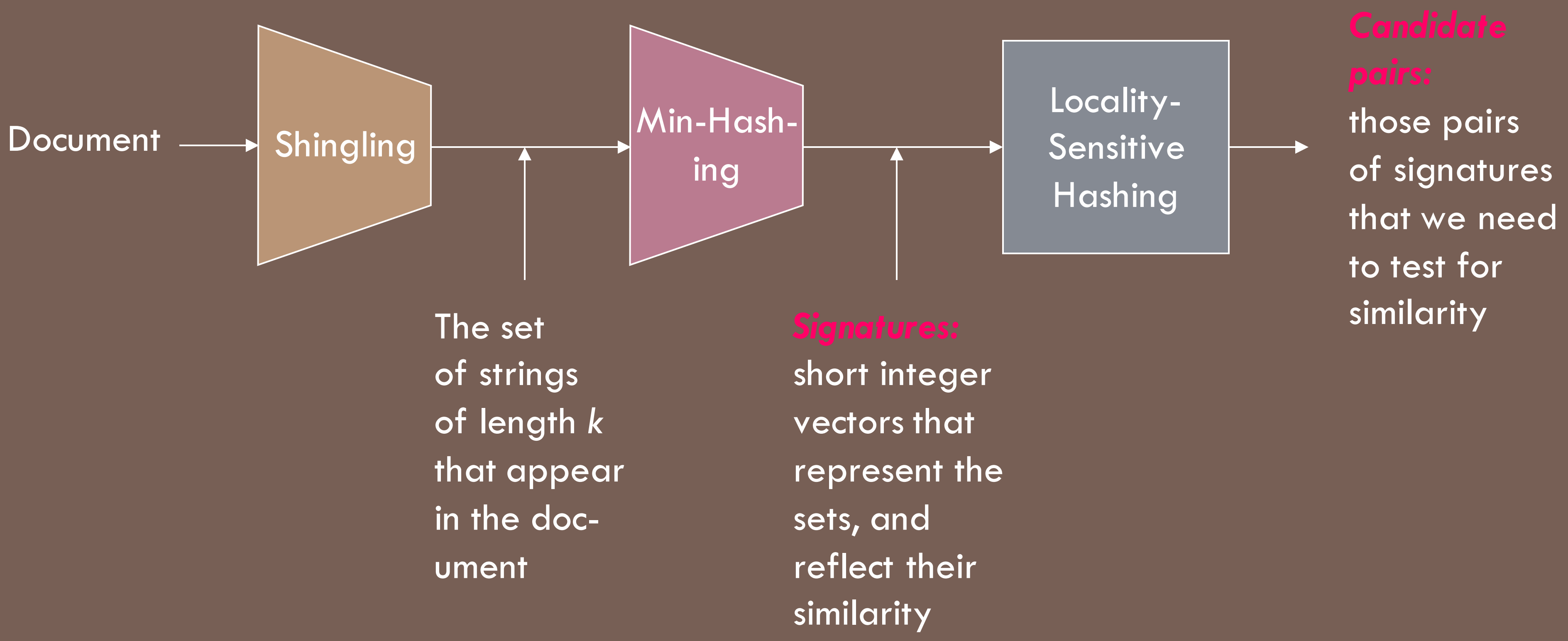
Implementation Trick

- **Permuting rows even once is prohibitive**
- **Row hashing!**
  - Pick **K = 100** hash functions k_i
  - Ordering under k_i gives a random row permutation!
- **One-pass implementation**
  - For each column **C** and hash-func. k_i keep a "slot" for the min-hash value
  - Initialize all $sig(C)[i] = \infty$
  - **Scan rows looking for 1s**
    - Suppose row j has 1 in column C
    - Then for each k_i :
      - If $k_i(j) < sig(C)[i]$, then $sig(C)[i] \leftarrow k_i(j)$

How to pick a random hash function h(x)?

Universal hashing:
$h_{a,b}(x)=((a \cdot x+b) \bmod p) \bmod N$
where:
a,b … random integers
p … prime number (p > N)

More details:
Section 3.3.5 in



LOCALITY SENSITIVE HASHING

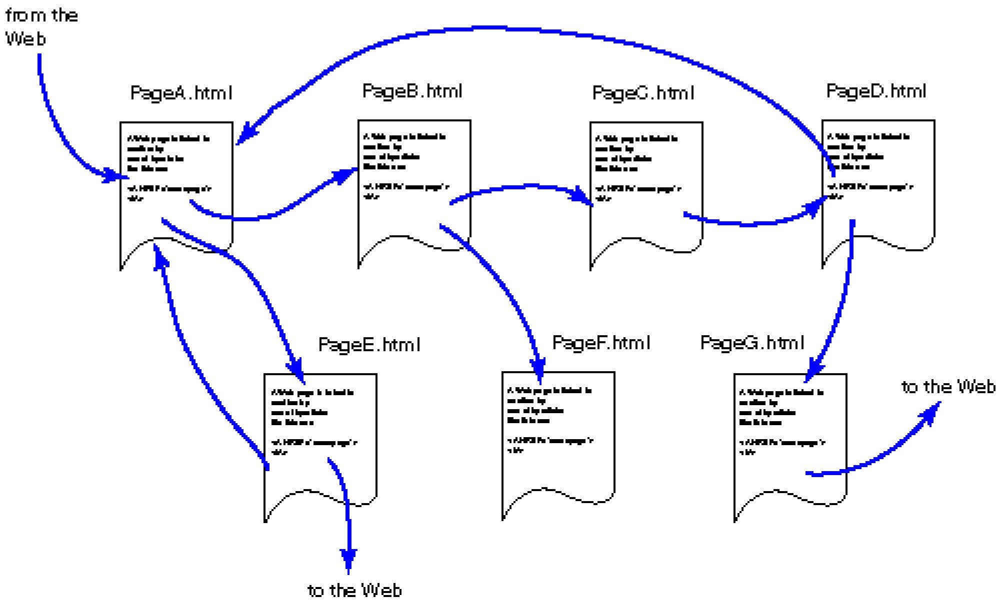
Step 3: ***Locality-Sensitive Hashing:*** Focus on pairs of signatures likely to be from similar documents (Optional, See backup slides)

Chapter 4 Graph Data:
http://www.dataminingbook.info/pmwiki.php/Main/BookPathUploads?action=downloadman&upname=book-20160121.pdf ,
http://www.dataminingbook.info/pmwiki.php

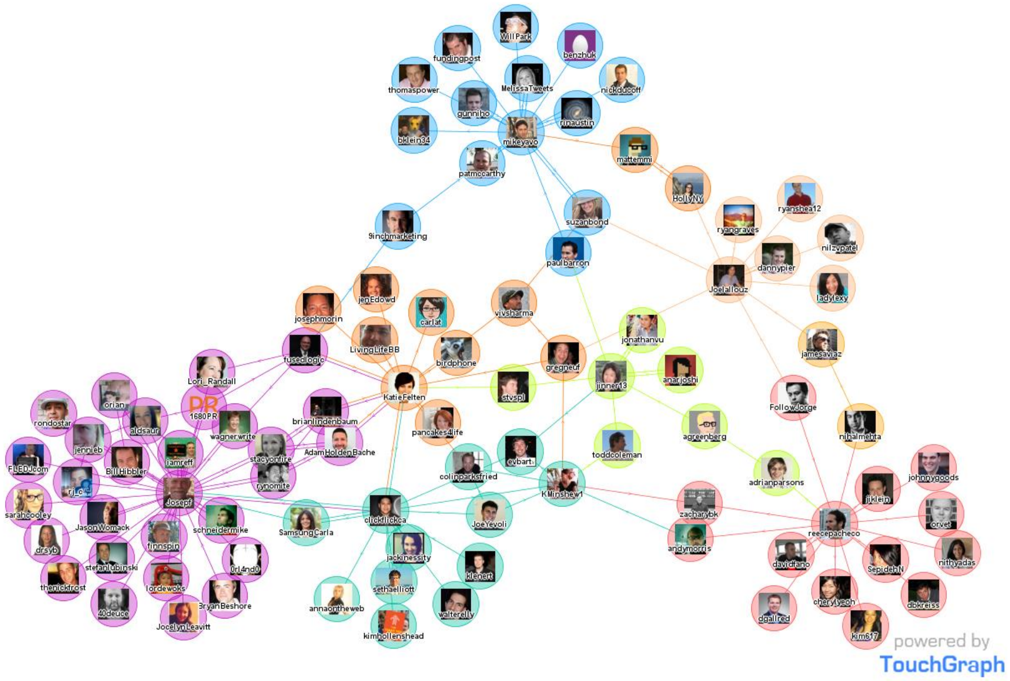
GRAPH BASICS AND A GENTLE INTRODUCTION TO PAGERANK

Slides adapted from Prof. Srinivasan Parthasarathy @OSU

Graphs from the Real World



The Web: hyperlinked docs

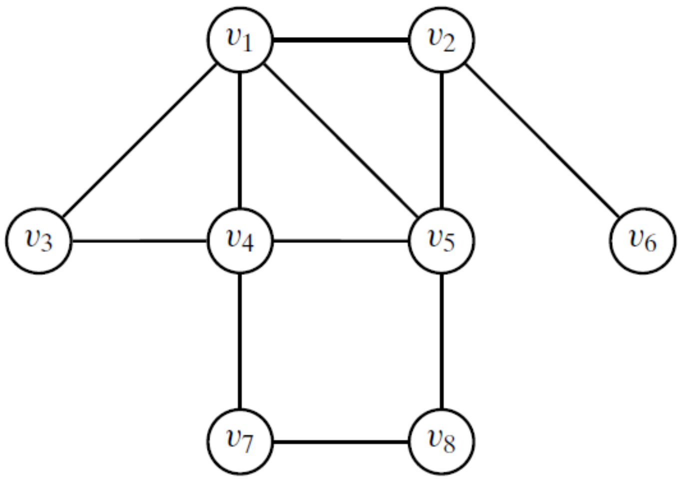


Social networks

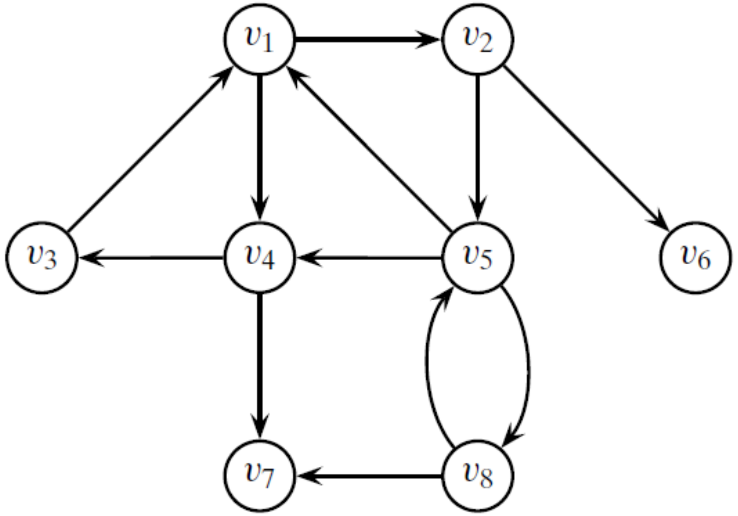
https://chortle.ccsu.edu/Java5/Notes/appendixA/htmlPart2_6.html
http://www.touchgraph.com/news

Primitives and Notations

- G = (V, E)
  - $E \subseteq V \times V$, and can also be represented as an adjacency matrix.
- Undirected vs. directed graph



(a)



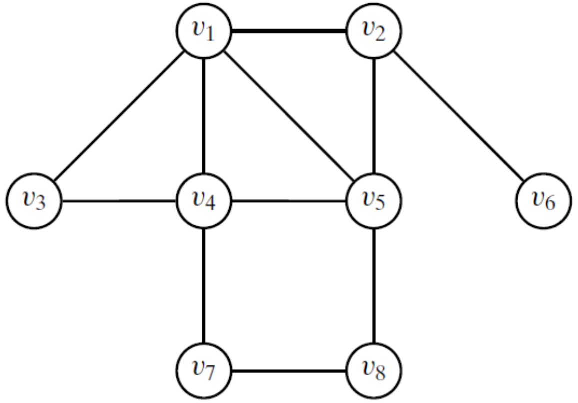
(b)

A directed edge (v_i, v_j) is also called an *arc*, and is said to be *from* v_i *to* v_j. We also say that v_i is the *tail* and v_j the *head* of the arc.

Primitives and Notations

- G = (V, E)
  - E can also be represented as an adjacency matrix
- Undirected vs. directed graph
- Degree

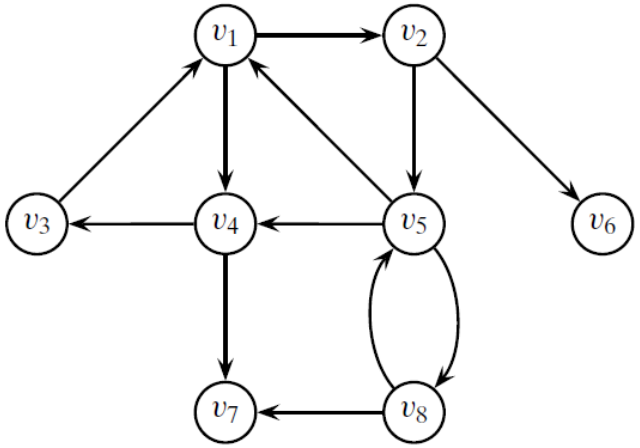
The *degree* of a node $v_i \in V$ is the number of edges incident with it



(a)

Primitives and Notations

- G = (V, E)
  - E can also be represented as an adjacency matrix
- Undirected vs. directed graph
- Degree



(b)

For directed graphs, the *indegree* of node v_i, denoted as $id(v_i)$, is the number of edges with v_i as head, that is, the number of incoming edges at v_i. The *outdegree* of v_i, denoted $od(v_i)$, is the number of edges with v_i as the tail, that is, the number of outgoing edges from v_i.

Primitives and Notations

- G = (V, E)
  - E can also be represented as an adjacency matrix
- Undirected vs. directed graph
- Degree
- (Shortest) distance between two vertices

The *eccentricity* of a node v_i is the maximum distance from v_i to any other node in the graph:

$$\text{Eccentricity}(v) = \max_{u \neq v} dist(u, v)$$

Primitives and Notations

- G = (V, E)
  - E can also be represented as an adjacency matrix
- Undirected vs. directed graph
- Degree
- (Shortest) distance between two vertices

The *eccentricity* of a node v_i is the maximum distance from v_i to any other node in the graph:

$$\text{Eccentricity}(v) = \max_{u \neq v} dist(u, v)$$

Primitives and Notations

- G = (V, E)
  - E can also be represented as an adjacency matrix
- Undirected vs. directed graph
- Degree
- (Shortest) distance between two vertices

The *radius* of a connected graph, denoted $r(G)$, is the minimum eccentricity of any node in the graph:

$$\text{Radius}(G) = \min_{v \in V} \text{Eccentricity}(v)$$

Primitives and Notations

- G = (V, E)
  - E can also be represented as an adjacency matrix
- Undirected vs. directed graph
- Degree
- (Shortest) distance between two vertices

The *diameter*, denoted $d(G)$, is the maximum eccentricity of any vertex in the graph:

$$\text{Diameter}(G) = \max_{v \in V} \text{Eccentricity}(v)$$

Properties of Nodes

- Centrality: how “central” or important a node is in the graph
  - How close the node is to all other nodes?

$$\text{Closeness Centrality}(v) = \frac{1}{\sum_{u \neq v} dist(u, v)}$$

A node v_i with the smallest total distance, $\sum_j d(v_i, v_j)$, is called the *median node*.

Properties of Nodes

- Centrality: how "central" or important a node is in the graph
  - How close the node is to all other nodes?
  - How much is a node a "choke point"?

Betweenness centrality: How many shortest paths between all pairs of vertices include vi.

$\gamma_{jk}(v_i) = \frac{\eta_{jk}(v_i)}{\eta_{jk}}$: the fraction of shortest paths between vertices v_j and v_k through v_i

The betweenness centrality for a node v_i is defined as

$$c(v_i) = \sum_{j \neq i} \sum_{\substack{k \neq i \\ k > j}} \gamma_{jk}(v_i) = \sum_{j \neq i} \sum_{\substack{k \neq i \\ k > j}} \frac{\eta_{jk}(v_i)}{\eta_{jk}}$$

Properties of Nodes

- Clustering coefficient: how much does a node cluster with neighbors
  - Local clustering coefficient

  The **local clustering coefficient** of a vertex (node) in a graph quantifies how close its neighbors are to being a clique (complete graph).

  The proportion of links between the vertices within its neighbourhood divided by the number of links that could possibly exist between them.

Background

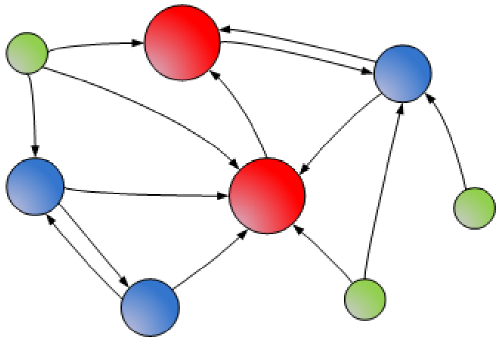
- Besides the keywords, what other evidence can one use to rate the importance of a webpage?

Background

- Besides the keywords, what other evidence can one use to rate the importance of a webpage?
- Solution: Use the hyperlink structure
- E.g. a webpage linked by many webpages is probably important.
  - but this method is not global (comprehensive).
- PageRank is developed by Larry Page in 1998.

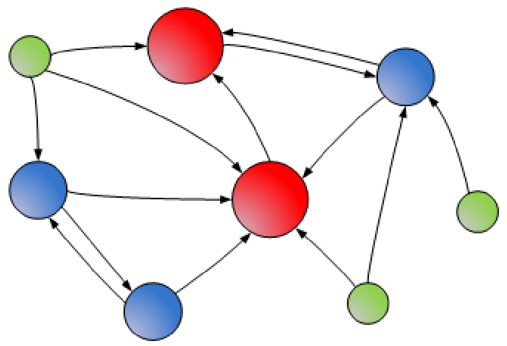
Idea

- A graph representing WWW
  - Node: webpage
  - Directed edge: hyperlink



Idea

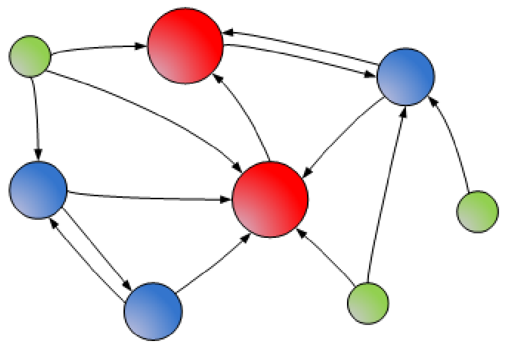
- A graph representing WWW
  - Node: webpage
  - Directed edge: hyperlink



- A user randomly clicks the hyperlink to surf WWW.
  - The probability a user stop in a particular webpage is the PageRank value.

Idea

- A graph representing WWW
  - Node: webpage
  - Directed edge: hyperlink



- A user randomly clicks the hyperlink to surf WWW.
  - The probability a user stop in a particular webpage is the PageRank value.

- A node that is linked by many nodes with high PageRank value receives a high rank itself;
  If there are no links to a node, then there is no support for that page.

Formal Formulation

Let $G = (V, E)$ be a directed graph, with $|V| = n$. The adjacency matrix of G is an $n \times n$ asymmetric matrix $\mathbf{A}$ given as

$$\mathbf{A}(u, v) = \begin{cases} 1 & \text{if } (u, v) \in E \\ 0 & \text{if } (u, v) \notin E \end{cases}$$

Let $p(u)$ be a positive real number, called the *prestige* score for node u.

$$\begin{aligned} p(v) &= \sum_u \mathbf{A}(u, v) \cdot p(u) \\ &= \sum_u \mathbf{A}^T(v, u) \cdot p(u) \end{aligned}$$

the prestige of a node depends on the prestige of other nodes pointing to it.

Formal Formulation

Let $p(u)$ be a positive real number, called the *prestige* score for node u.

$$p(v) = \sum_u \mathbf{A}(u, v) \cdot p(u)$$

$$= \sum_u \mathbf{A}^T(v, u) \cdot p(u)$$

the prestige of a node depends on the prestige of other nodes pointing to it.

Across all the nodes, we can recursively express the prestige scores as

$$\mathbf{p}' = \mathbf{A}^T \mathbf{p}$$

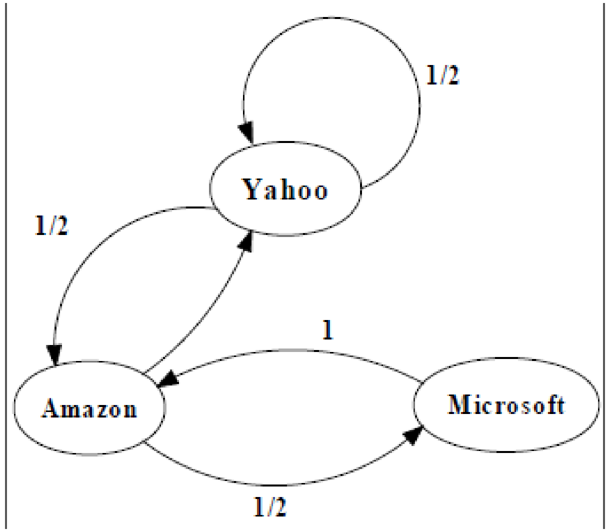
where $\mathbf{p}$ is an n-dimensional column vector corresponding to the prestige scores for each vertex.

Iterative Computation

$$
\begin{aligned}
\mathbf{p}_k &= \mathbf{A}^T \mathbf{p}_{k-1} \\
&= \mathbf{A}^T (\mathbf{A}^T \mathbf{p}_{k-2}) = (\mathbf{A}^T)^2 \mathbf{p}_{k-2} \\
&= (\mathbf{A}^T)^2 (\mathbf{A}^T \mathbf{p}_{k-3}) = (\mathbf{A}^T)^3 \mathbf{p}_{k-3} \\
&= \vdots \\
&= (\mathbf{A}^T)^k \mathbf{p}_0
\end{aligned}
$$

where $\mathbf{p}_0$ is the initial prestige vector. It is well known that the vector $\mathbf{p}_k$ converges to the dominant eigenvector of $\mathbf{A}^T$ with increasing k.

Example 1



$$M = \begin{bmatrix} 1/2 & 1/2 & 0 \\ 1/2 & 0 & 1 \\ 0 & 1/2 & 0 \end{bmatrix}$$

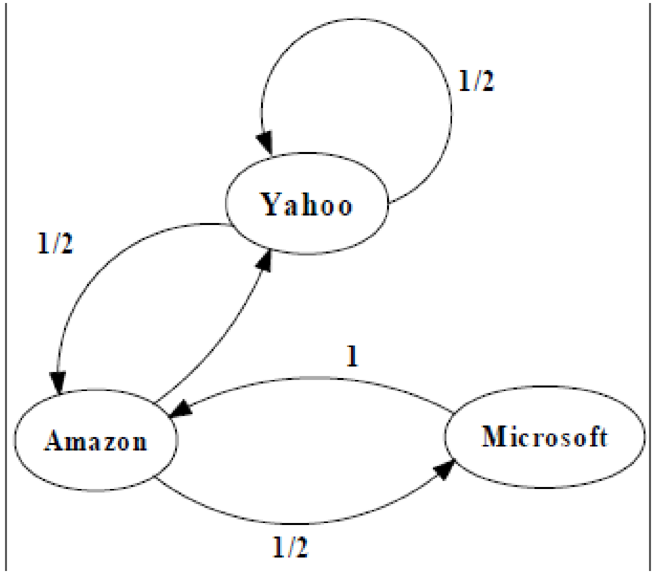
=the transpose of A (adjacency matrix)

$$\begin{bmatrix} \text{yahoo} \\ \text{Amazon} \\ \text{Microsoft} \end{bmatrix} = \begin{bmatrix} 1/3 \\ 1/3 \\ 1/3 \end{bmatrix}$$

$$\begin{bmatrix} 1/3 \\ 1/2 \\ 1/6 \end{bmatrix} = \begin{bmatrix} 1/2 & 1/2 & 0 \\ 1/2 & 0 & 1 \\ 0 & 1/2 & 0 \end{bmatrix} \begin{bmatrix} 1/3 \\ 1/3 \\ 1/3 \end{bmatrix}$$

PageRank Calculation: first iteration

Example 1



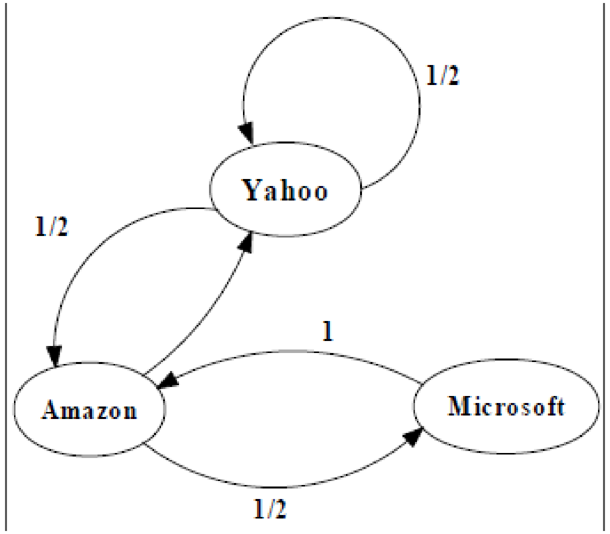
$$M = \begin{bmatrix} 1/2 & 1/2 & 0 \\ 1/2 & 0 & 1 \\ 0 & 1/2 & 0 \end{bmatrix}$$

$$\begin{bmatrix} \text{yahoo} \\ \text{Amazon} \\ \text{Microsoft} \end{bmatrix} = \begin{bmatrix} 1/3 \\ 1/3 \\ 1/3 \end{bmatrix}$$

$$\begin{bmatrix} 5/12 \\ 1/3 \\ 1/4 \end{bmatrix} = \begin{bmatrix} 1/2 & 1/2 & 0 \\ 1/2 & 0 & 1 \\ 0 & 1/2 & 0 \end{bmatrix} \begin{bmatrix} 1/3 \\ 1/2 \\ 1/6 \end{bmatrix}$$

PageRank Calculation: second iteration

Example 1



$$M = \begin{bmatrix} 1/2 & 1/2 & 0 \\ 1/2 & 0 & 1 \\ 0 & 1/2 & 0 \end{bmatrix}$$

$$\begin{bmatrix} \text{yahoo} \\ \text{Amazon} \\ \text{Microsoft} \end{bmatrix} = \begin{bmatrix} 1/3 \\ 1/3 \\ 1/3 \end{bmatrix}$$

$$\begin{bmatrix} 3/8 \\ 11/24 \\ 1/6 \end{bmatrix} \begin{bmatrix} 5/12 \\ 17/48 \\ 11/48 \end{bmatrix} \ldots \begin{bmatrix} 2/5 \\ 2/5 \\ 1/5 \end{bmatrix}$$

Convergence after some iterations

A simple version

$$R(u) = \sum_{v \in B_u} \frac{R(v)}{N_v}$$

- u: a webpage
- B_u: the set of u's backlinks
- N_v: the number of forward links of page v

- Initially, R(u) is 1/N for every webpage
- Iteratively update each webpage's PR value until convergence.

A little more advanced version

- Adding a damping factor *d*
- Imagine that a surfer would stop clicking a hyperlink with probability 1-*d*

$$R(u) = \frac{(1-d)}{N-1} + d\sum_{v \in B_u} \frac{R(v)}{N_v}$$

- R(u) is at least (1-d)/(N-1)
  - N is the total number of nodes.

Other applications

- Social network (Facebook, Twitter, etc)
  - Node: Person; Edge: Follower / Followee / Friend
  - Higher PR value: Celebrity
- Citation network
  - Node: Paper; Edge: Citation
  - Higher PR values: Important Papers.
- Protein-protein interaction network
  - Node: Protein; Edge: Two proteins bind together
  - Higher PR values: Essential proteins.

Backup slides

LSH: First Cut

2	1	4	1
1	2	1	2
2	1	2	1

- **Goal:** Find documents with Jaccard similarity at least *s* (for some similarity threshold, e.g., *s*=0.8)
- **LSH – General idea:** Use a function ***f(x,y)*** that tells whether ***x*** and ***y*** is a ***candidate pair***: a pair of elements whose similarity must be evaluated
- **For Min-Hash matrices:**
  - Hash columns of signature matrix ***M*** to many buckets
  - Each pair of documents that hashes into the same bucket is a **candidate pair**

Candidates from Min-Hash

2	1	4	1
1	2	1	2
2	1	2	1

- **Pick a similarity threshold *s* (0 < s < 1)**

- Columns ***x*** and ***y*** of ***M*** are a **candidate pair** if their signatures agree on at least fraction ***s*** of their rows:
  M (i, x) = M (i, y) for at least frac. ***s*** values of ***I***
  - We expect documents ***x*** and ***y*** to have the same (Jaccard) similarity as their signatures

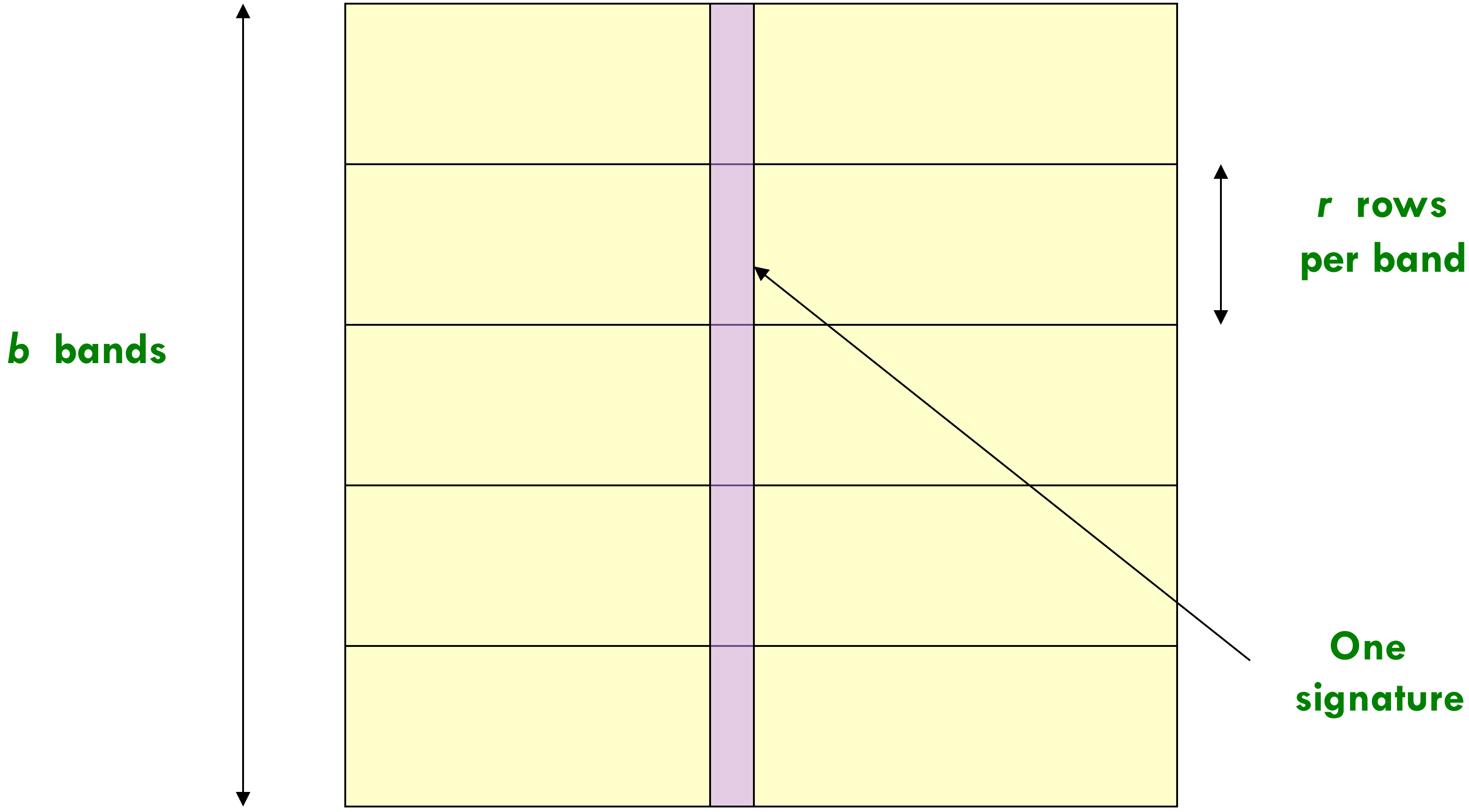
2	1	4	1
1	2	1	2
2	1	2	1

LSH for Min-Hash

- **Big idea: Hash columns of signature matrix *M* several times**
- Arrange that (only) **similar columns** are likely to **hash to the same bucket**, with high probability
- **Candidate pairs are those that hash to the same bucket**

Partition M into b Bands

2	1	4	1
1	2	1	2
2	1	2	1



Signature matrix *M*

Partition M into Bands

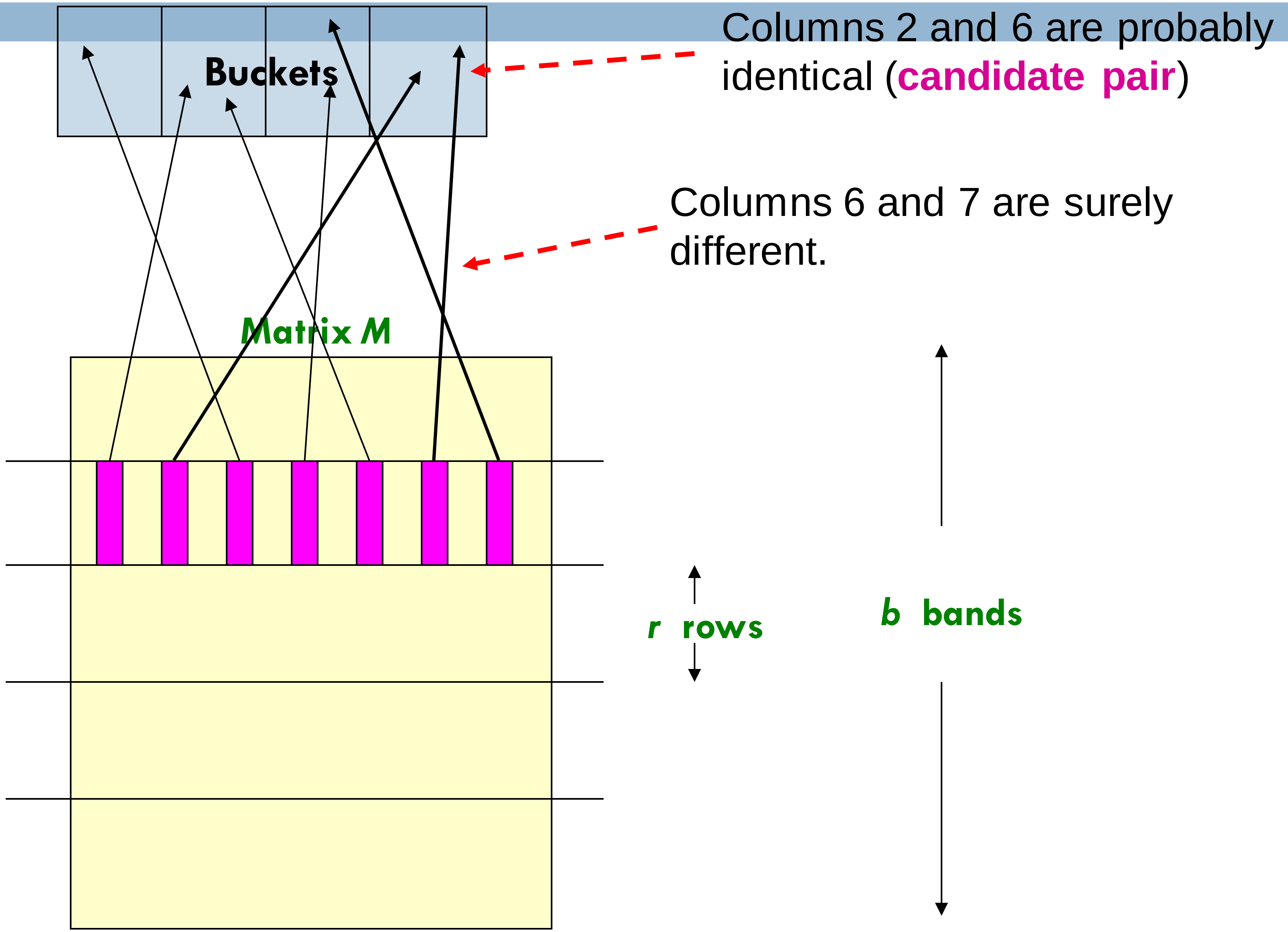
- Divide matrix ***M*** into ***b*** bands of ***r*** rows

- For each band, hash its portion of each column to a hash table with ***k*** buckets
  - Make ***k*** as large as possible

Partition M into Bands

- Divide matrix ***M*** into ***b*** bands of ***r*** rows
- For each band, hash its portion of each column to a hash table with ***k*** buckets
  - Make ***k*** as large as possible
- ***Candidate*** column pairs are those that hash to the same bucket for ≥ 1 band
- Tune ***b*** and ***r*** to catch most similar pairs, but few non-similar pairs

Hashing Bands



Simplifying Assumption

- There are **enough buckets** that columns are unlikely to hash to the same bucket unless they are **identical** in a particular band
- Hereafter, we assume that "**same bucket**" means "**identical in that band**"
- Assumption needed only to simplify analysis, not for correctness of algorithm

Example of Bands

2	1	4	1
1	2	1	2
2	1	2	1

Assume the following case:

- Suppose 100,000 columns of ***M*** (100k docs)
- Signatures of 100 integers (rows)
- Therefore, signatures take 40Mb
- Choose b = 20 bands of r = 5 integers/band

- **Goal:** Find pairs of documents that are at least $s = 0.8$ similar

2	1	4	1
1	2	1	2
2	1	2	1

C_1, C_2 are 80% Similar

- **Find pairs of** ≥ *s*=0.8 similarity, set **b**=20, **r**=5

- **Assume:** sim(C_1, C_2) = 0.8
  - Since sim(C_1, C_2) ≥ **s**, we want C_1, C_2 to be a **candidate pair**: We want them to hash to at **least 1 common bucket** (at least one band is identical)

C_1, C_2 are 80% Similar

2	1	4	1
1	2	1	2
2	1	2	1

- **Find pairs of** $\geq s=0.8$ similarity, set $\mathbf{b}=20$, $\mathbf{r}=5$
- **Assume:** $\text{sim}(C_1, C_2) = 0.8$
  - Since $\text{sim}(C_1, C_2) \geq \mathbf{s}$, we want C_1, C_2 to be a **candidate pair**: We want them to hash to at **least 1 common bucket** (at least one band is identical)
- **Probability C_1, C_2 identical in one particular band:** $(0.8)^5 = 0.328$
- Probability C_1, C_2 are ***not*** similar in all of the 20 bands: $(1-0.328)^{20} = 0.00035$
  - i.e., about 1/3000th of the 80%-similar column pairs are **false negatives** (we miss them)
  - **We would find 99.965% pairs of truly similar documents**

2	1	4	1
1	2	1	2
2	1	2	1

C_1, C_2 are 30% Similar

- **Find pairs of** $\geq s=0.8$ similarity, set **b**=20, **r**=5
- **Assume:** sim(C_1, C_2) = 0.3
  - Since sim(C_1, C_2) < **s** we want C_1, C_2 to hash to **NO common buckets** (all bands should be different)

C_1, C_2 are 30% Similar

2	1	4	1
1	2	1	2
2	1	2	1

- **Find pairs of** $\geq$ s=0.8 similarity, set **b**=20, **r**=5

- **Assume:** sim(C_1, C_2) = 0.3
  - Since sim(C_1, C_2) < **s** we want C_1, C_2 to hash to **NO common buckets** (all bands should be different)

- **Probability C_1, C_2 identical in one particular band:** $(0.3)^5 = 0.00243$
- Probability C_1, C_2 identical in at least 1 of 20 bands: $1 - (1 - 0.00243)^{20} = 0.0474$
  - In other words, approximately 4.74% pairs of docs with similarity 30% end up becoming **candidate pairs**
    - They are **false positives** since we will have to examine them (they are candidate pairs) but then it will turn out their similarity is below threshold **s**

LSH Involves a Tradeoff

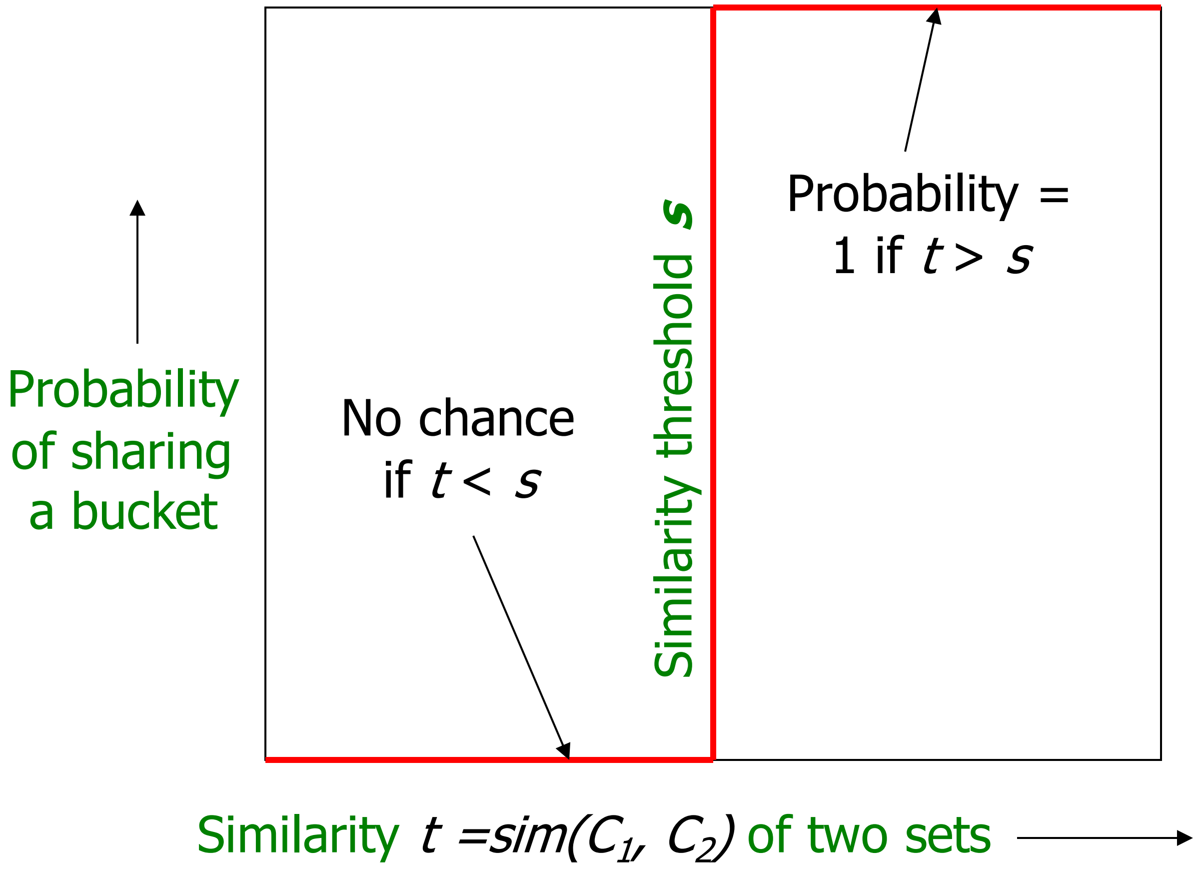
2	1	4	1
1	2	1	2
2	1	2	1

- **Pick:**
  - The number of Min-Hashes (rows of ***M***)
  - The number of bands ***b***, and
  - The number of rows ***r*** per band

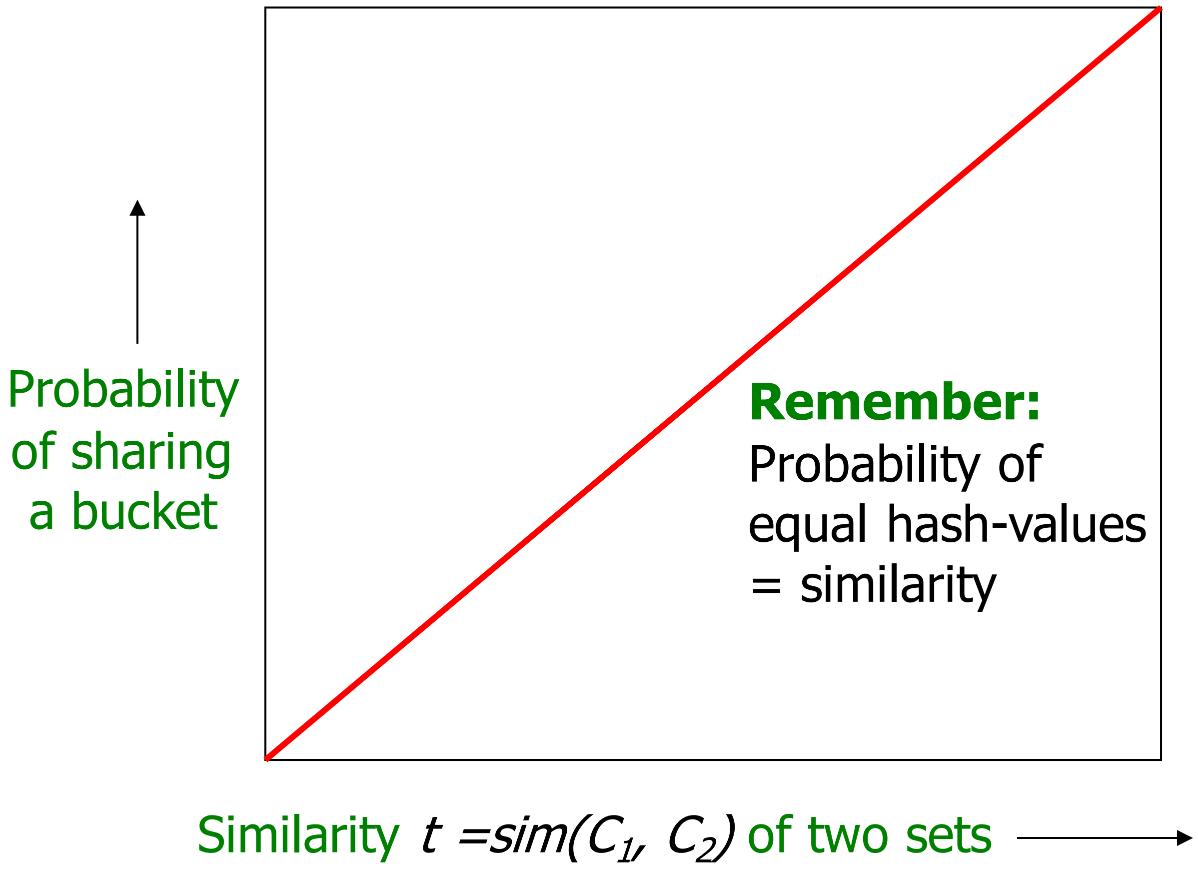
  to balance false positives/negatives

- **Example:** If we had only 15 bands of 5 rows, the number of false positives would go down, but the number of false negatives would go up

Analysis of LSH – What We Want



What 1 Band of 1 Row Gives You



b bands, *r* rows/band

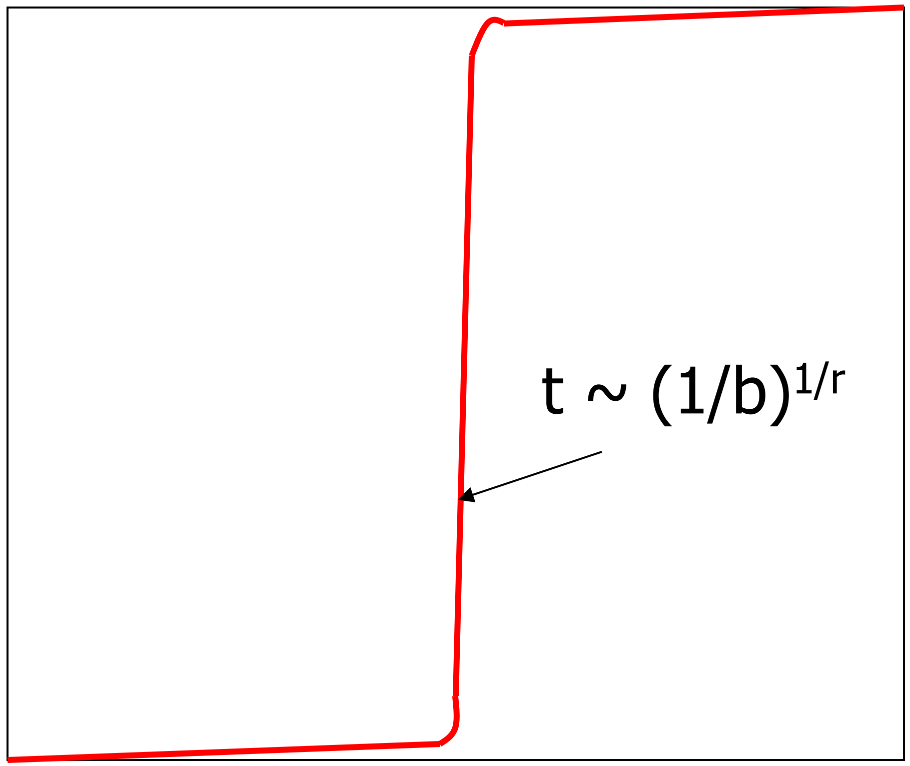
- Columns C_1 and C_2 have similarity ***t***
- Pick any band (***r*** rows)
  - Prob. that all rows in band equal = t^r
  - Prob. that some row in band unequal = $1 - t^r$
- Prob. that no band identical = $(1 - t^r)^b$
- Prob. that at least 1 band identical = $1 - (1 - t^r)^b$

What *b* Bands of *r* Rows Gives You

At least one band identical

$$1 - (1 - t^r)^b$$

Probability of sharing a bucket



Similarity $t = sim(C_1, C_2)$ of two sets

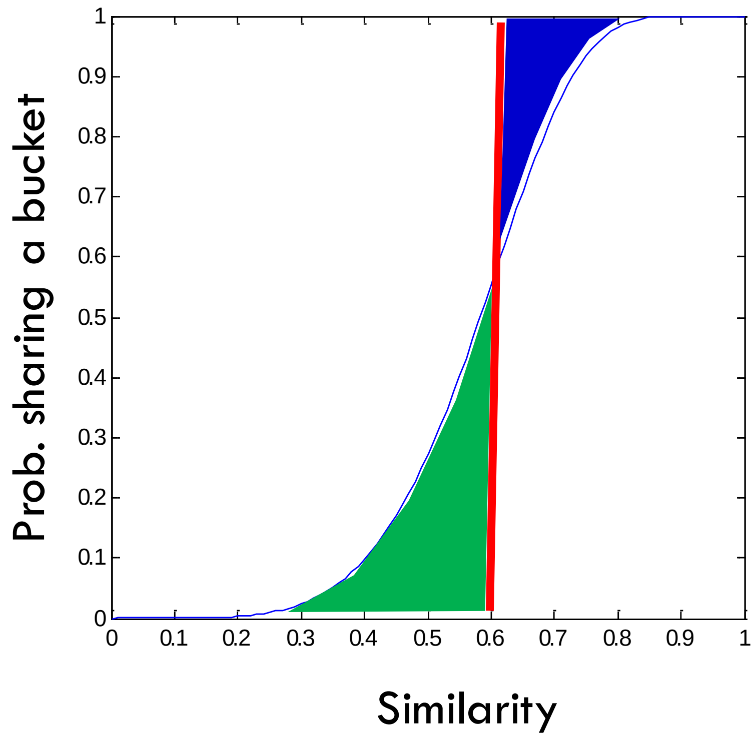
Example: $b = 20; r = 5$

- Similarity threshold s
- Prob. that at least 1 band is identical:

s	$1-(1-s^r)^b$
.2	.006
.3	.047
.4	.186
.5	.470
.6	.802
.7	.975
.8	.9996

Picking *r* and *b*: The S-curve

- **Picking *r* and *b* to get the best S-curve**
  - 50 hash-functions (r=5, b=10)



Blue area: False Negative rate
Green area: False Positive rate

LSH Summary

- Tune ***M*, *b*, *r*** to get almost all pairs with similar signatures, but eliminate most pairs that do not have similar signatures
- Check in main memory that **candidate pairs** really do have **similar signatures**
- **Optional:** In another pass through data, check that the remaining candidate pairs really represent similar documents

Summary: 3 Steps

- **Shingling:** Convert documents to sets
  - We used hashing to assign each shingle an ID
- **Min-Hashing:** Convert large sets to short signatures, while preserving similarity
  - We used **similarity preserving hashing** to generate signatures with property $\mathbf{Pr}[h_\pi(C_1) = h_\pi(C_2)] = sim(C_1, C_2)$
  - We used hashing to get around generating random permutations
- **Locality-Sensitive Hashing:** Focus on pairs of signatures likely to be from similar documents
  - We used hashing to find **candidate pairs** of similarity $\geq$ **s**