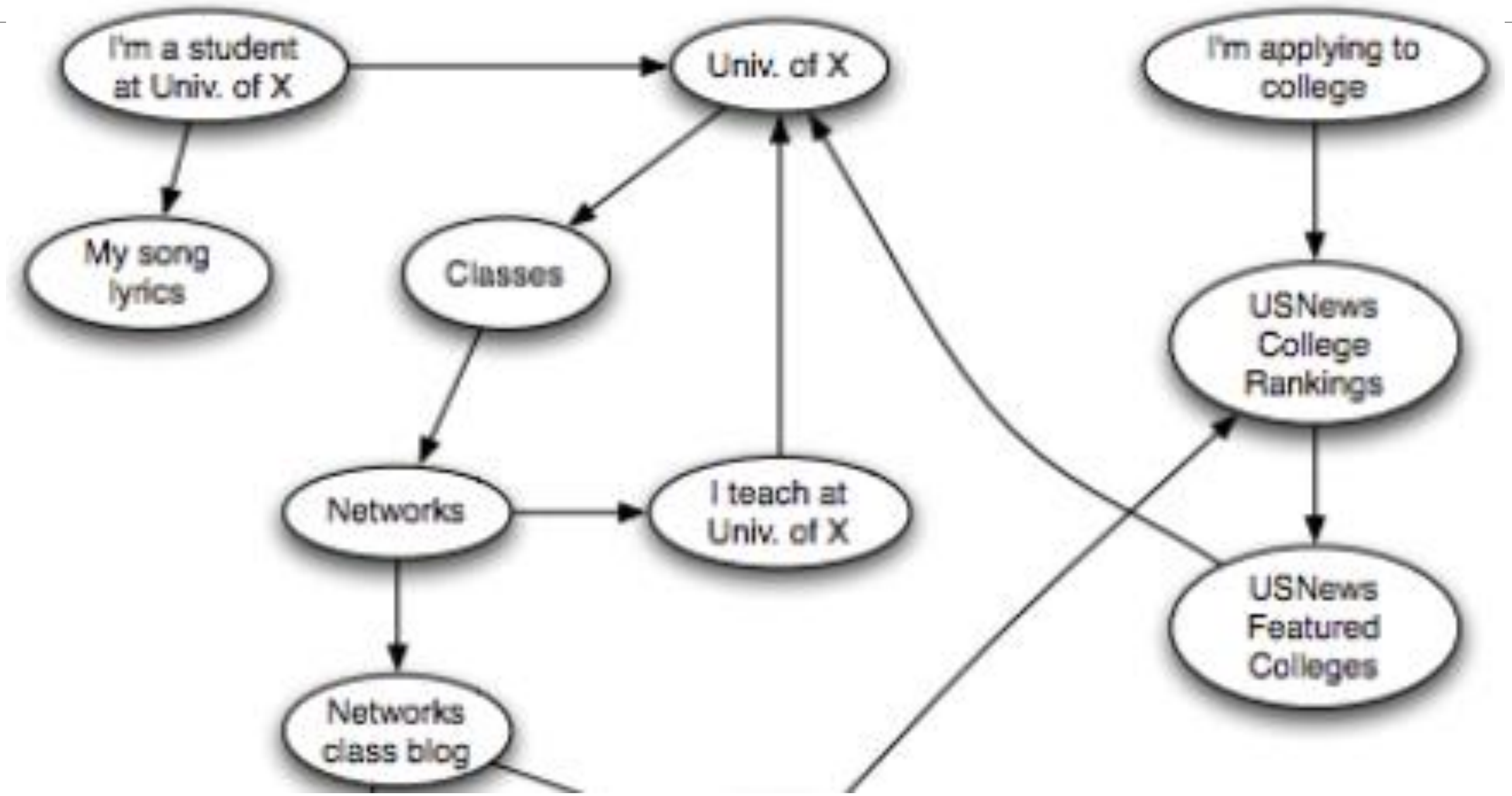
MapReduce and link analysis

ANISHA JOSEPH

Web as a Graph

- Web as a directed graph:
  - Nodes: Webpages
  - Edges: Hyperlinks

Web as a Graph



Broad Question

- **How to organize the Web?**
- **First try:** Human curated **Web directories**
  - Yahoo, DMOZ, LookSmart
- **Second try: Web Search**
  - **Information Retrieval** investigates: Find relevant docs in a small and trusted set
    - Newspaper articles, Patents, etc.
  - But: Web is **huge**, full of untrusted documents, random things, web spam, etc.

Web Search: 2 Challenges

2 challenges of web search:

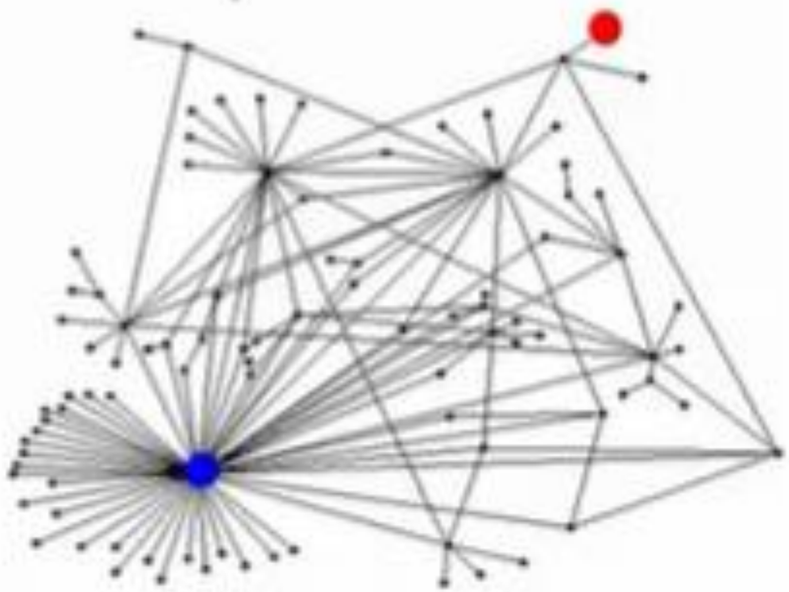
- **(1) Web contains many sources of information Who to "trust"?**
  - **Trick:** Trustworthy pages may point to each other!
- **(2) What is the "best" answer to query "newspaper"?**
  - No single right answer
  - **Trick:** Pages that actually know about newspapers might all be pointing to many newspapers

Ranking Nodes on the Graph

- **All web pages are not equally "important"**

  www.joe-schmoe.com vs. www.stanford.edu

- There is large diversity in the web-graph node connectivity.
  Let's rank the pages by the link structure!



Link Analysis Algorithms

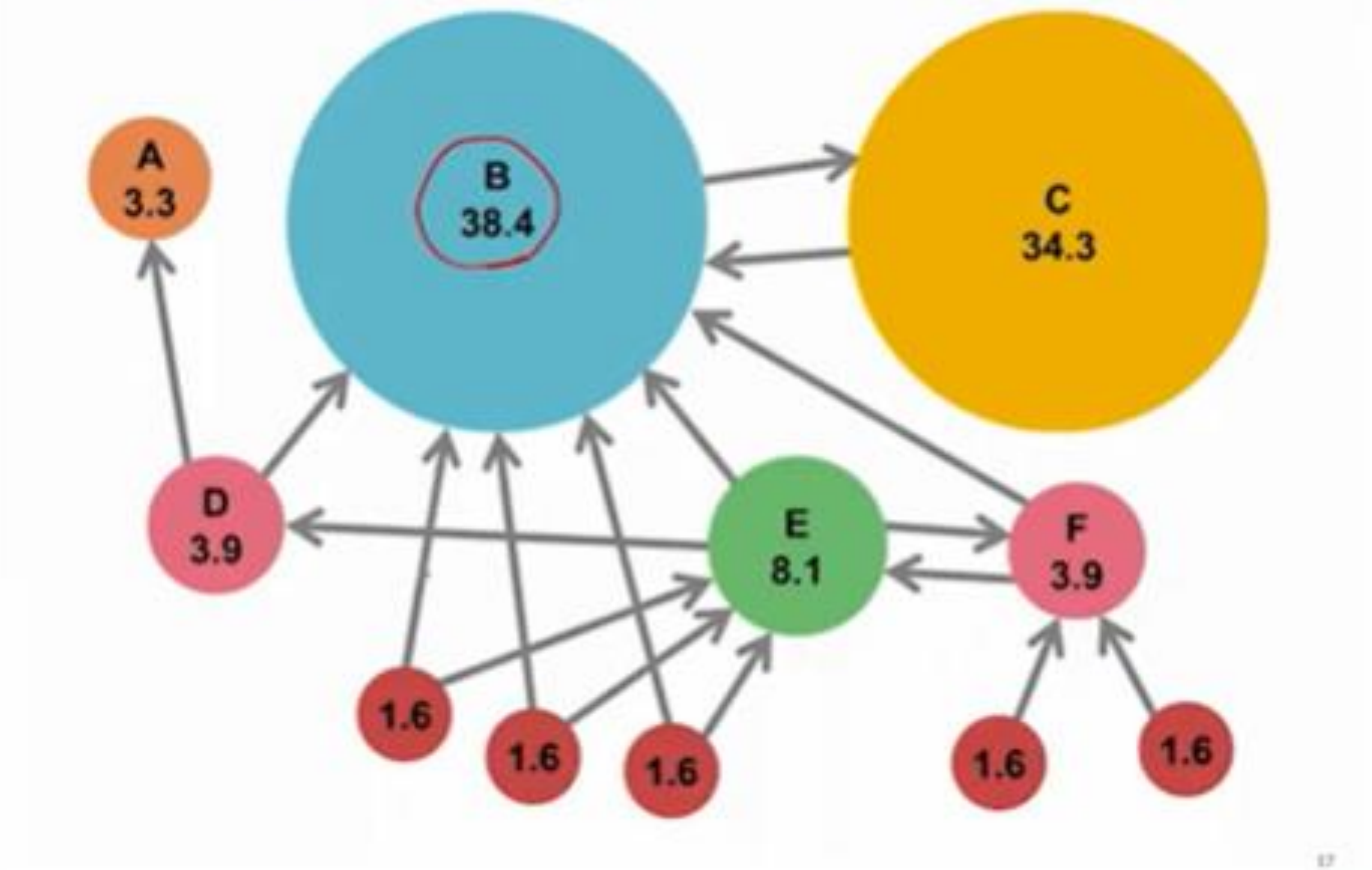
- We will cover the following **Link Analysis approaches** for computing **importances** of nodes in a graph:
  - Page Rank
  - Hubs and Authorities (HITS)
  - Topic-Specific (Personalized) Page Rank
  - Web Spam Detection Algorithms

PageRank

Links as Votes

- **Idea: Links as votes**
  - **Page is more important if it has more links**
    - In-coming links? Out-going links?
- **Think of in-links as votes:**
  - www.stanford.edu has 23,400 in-links
  - www.joe-schmoe.com has 1 in-link
- **Are all in-links are equal?**
  - Links from important pages count more
  - Recursive question!

Example: PageRank Scores

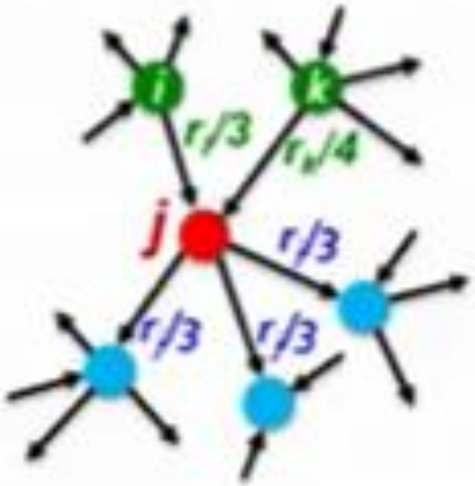


High score not only because of high in links but also high worth links.

Simple Recursive Formulation

- Each link's vote is proportional to the **importance** of its source page
- If page ***j*** with importance r_j has ***n*** out-links, each link gets r_j / n votes
- Page ***j***'s own importance is the sum of the votes on its in-links

$$r_j = r_i/3 + r_k/4$$

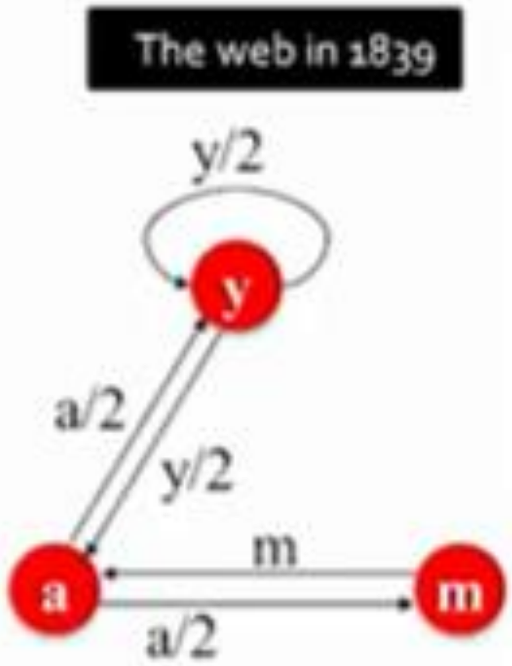


PageRank: The "Flow" Model

- A "vote" from an important page is worth more
- A page is important if it is pointed to by other important pages
- Define a "rank" r_j for page j

$$r_j = \sum_{i \to j} \frac{r_i}{\mathrm{d_i}}$$

d_i … out-degree of node i

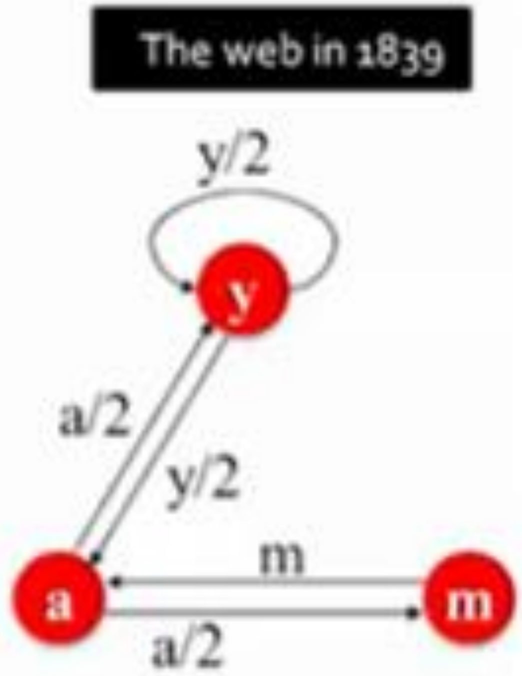


PageRank: The "Flow" Model

- A "vote" from an important page is worth more
- A page is important if it is pointed to by other important pages
- Define a "rank" r_j for page j

$$\boxed{r_j = \sum_{i \to j} \frac{r_i}{d_i}}$$

d_i … out-degree of node i



"Flow" equations:

$r_y = r_y/2 + r_a/2$

$r_a = r_y/2 + r_m$

$r_m = r_a/2$

Solving the Flow Equations

Flow equations:

$r_y = r_y/2 + r_a/2$

$r_a = r_y/2 + r_m$

$r_m = r_a/2$

- 3 equations, 3 unknowns, no constants
  - No unique solution
  - All solutions equivalent modulo the scale factor
- Additional constraint forces uniqueness:
  - $r_y + r_a + r_m = 1$
  - Solution: $r_y = \frac{2}{5}$, $r_a = \frac{2}{5}$, $r_m = \frac{1}{5}$
- Gaussian elimination method works for small examples, but we need a better method for large web-size graphs
- We need a new formulation!

PageRank: Matrix Formulation

- **Stochastic adjacency matrix M**
  - Let page i has d_i out-links
  - If $i \rightarrow j$, then $M_{ji} = \frac{1}{d_i}$ else $M_{ji} = 0$
    - M is a **column stochastic matrix**
      - Columns sum to **1**
- **Rank vector r:** vector with an entry per page
  - r_i is the importance score of page i
  - $\sum_i r_i = 1$
- **The flow equations can be written**

$$r_j = \sum_{i \rightarrow j} \frac{r_i}{\mathrm{d_i}}$$

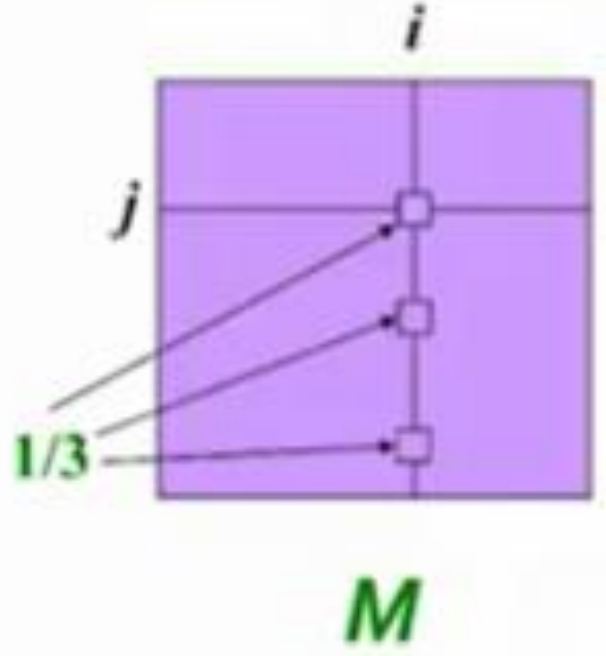
$$r = M \cdot r$$

Example

- Remember the flow equation: $r_j = \sum_{i \to j} \frac{r_i}{\mathrm{d}_i}$
- Flow equation in the matrix form

$$M \cdot r = r$$

  - Suppose page i links to 3 pages, including j



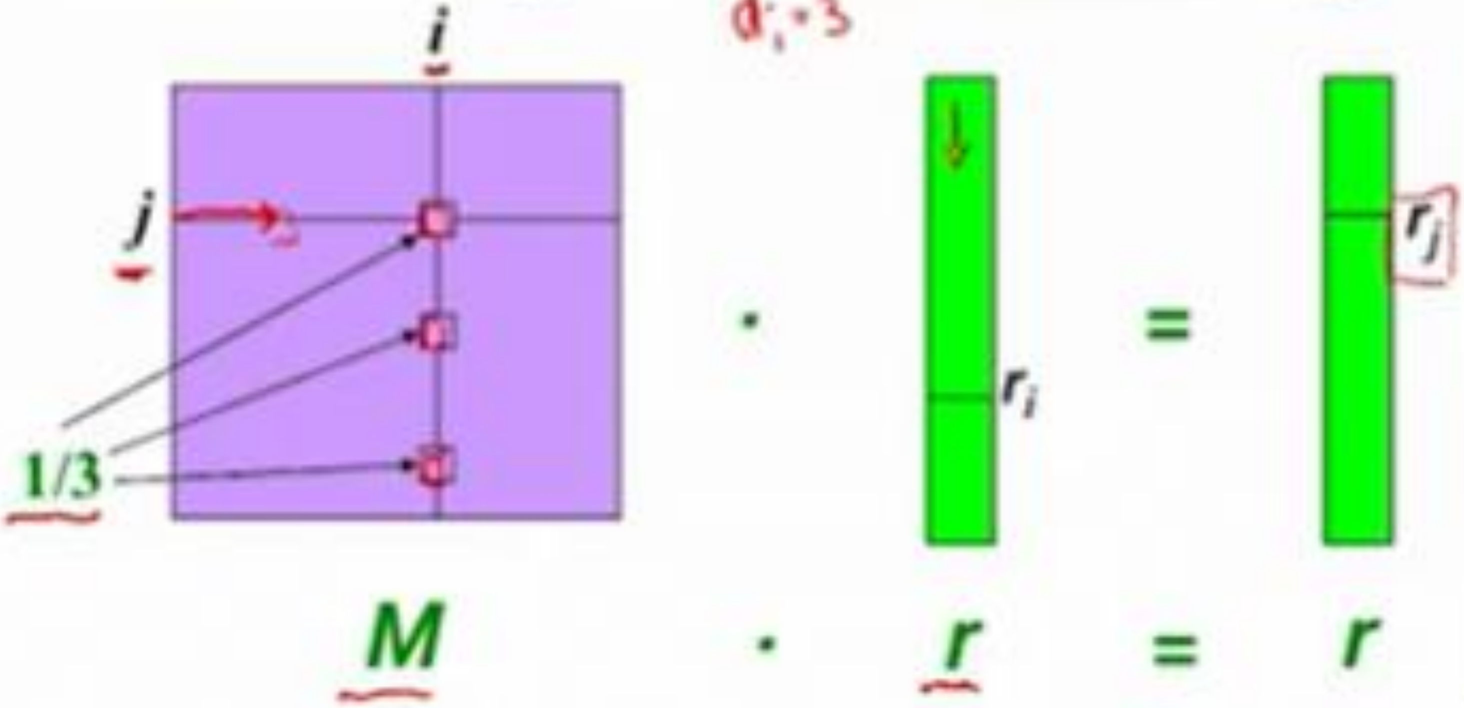
Example

- Remember the flow equation: $\boxed{r_j = \sum_{i \to j} \frac{r_i}{d_i}}$
- Flow equation in the matrix form

$$M \cdot r = r$$

  - Suppose page *i* links to 3 pages, including *j*

$d_i = 3$



Eigenvector Formulation

- The flow equations can be written

$$r = M \cdot r$$

NOTE: x is an eigenvector with the corresponding eigenvalue λ if:

$$Ax = \lambda x$$

- So the **rank vector** r is an **eigenvector** of the stochastic web matrix M
  - In fact, its first or principal eigenvector, with corresponding eigenvalue *1*
    - Largest eigenvalue of M is **1** since M is column stochastic
      - **Why?** *We know* **r** *is a stochastic vector and each column of* **M** *sums to one, so* $\mathbf{Mr} \leq 1$

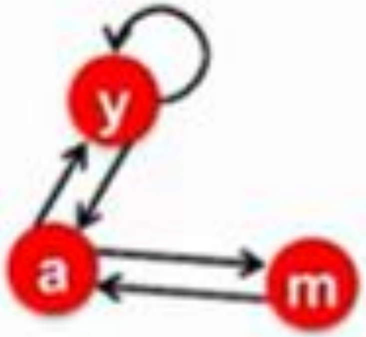
Eigen vector Formulation

- Here λ is 1
- A square matrix A is stochastic if all of its entries are nonnegative, and the entries of each column sum to 1.

- **We can now efficiently solve for *r*!**
  The method is called Power iteration

- Efficient method to find eigen vector for M.

Example: Flow Equations & M



$$
M = \begin{array}{c|ccc} & y & a & m \\ \hline y & \frac{1}{2} & \frac{1}{2} & 0 \\ a & \frac{1}{2} & 0 & 1 \\ m & 0 & \frac{1}{2} & 0 \end{array}
$$

$$\boldsymbol{r = M \cdot r}$$

$$
\begin{aligned}
r_y &= r_y/2 + r_a/2 \\
r_a &= r_y/2 + r_m \\
r_m &= r_a/2
\end{aligned}
\longrightarrow
\begin{bmatrix} y \\ a \\ m \end{bmatrix} = \begin{bmatrix} \frac{1}{2} & \frac{1}{2} & 0 \\ \frac{1}{2} & 0 & 1 \\ 0 & \frac{1}{2} & 0 \end{bmatrix} \begin{bmatrix} y \\ a \\ m \end{bmatrix}
$$

Definition of PageRank- Example

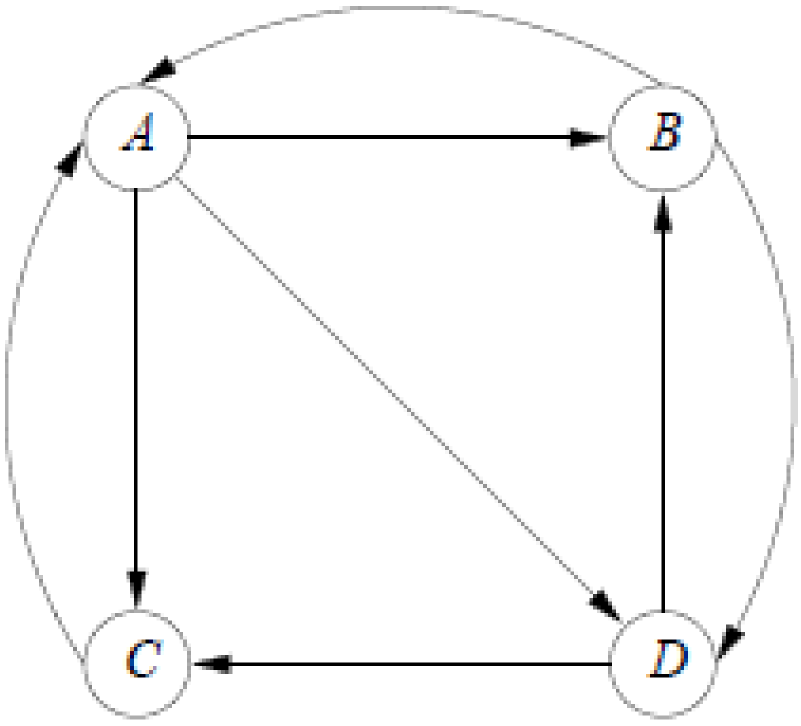


Figure 5.1: A hypothetical example of the Web

The transition matrix for the Web

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

Definition of PageRank

This sort of behavior is an example of the ancient theory of **Markov processes.** It is known that the distribution of the surfer approaches a limiting distribution r that satisfies r = Mr, provided two conditions are met:

1. The graph is **strongly connected;** that is, it is possible to get from any node to any other node.
2. There are **no dead ends**: nodes that have no arcs out.

Note: A directed graph is strongly connected if there is a path between any two pair of vertices.

Power Iteration Method

- **Given a web graph with *N* nodes, where the nodes are pages and edges are hyperlinks**
- **Power iteration:** a simple iterative scheme
  - Suppose there are ***N*** web pages
  - Initialize: $\mathbf{r}^{(0)} = [1/N,\ldots,1/N]^T$
  - Iterate: $\mathbf{r}^{(t+1)} = \mathbf{M} \cdot \mathbf{r}^{(t)}$
  - Stop when $|\mathbf{r}^{(t+1)} - \mathbf{r}^{(t)}|_1 < \varepsilon$

$$r_j^{(t+1)} = \sum_{i \to j} \frac{r_i^{(t)}}{d_i}$$

d_i …. out-degree of node i

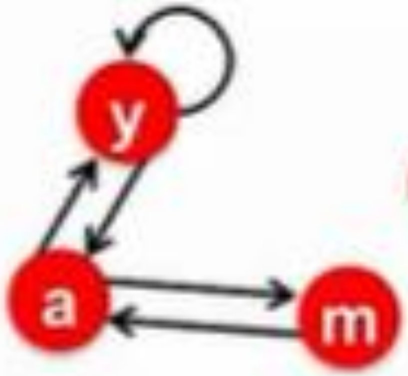
$|\mathbf{x}|_1 = \sum_{1 \le i \le N} |x_i|$ is the L_1 norm
Can use any other vector norm, e.g., Euclidean

PageRank: How to solve?

- **Power Iteration:**
  - Set $r_j = 1/N$
  - **1:** $r'_j = \sum_{i \to j} \frac{r_i}{d_i}$
  - **2:** $r = r'$
  - If not converged: goto **1**
- **Example:**

$$\begin{bmatrix} r_y \\ r_a \\ r_m \end{bmatrix} = \begin{matrix} 1/3 \\ 1/3 \\ 1/3 \end{matrix}$$

Iteration 0, 1, 2, …



$$M = \begin{array}{c} \\ y \\ a \\ m \end{array} \begin{array}{c} \begin{array}{ccc} y & a & m \end{array} \\ \begin{bmatrix} \frac{1}{2} & \frac{1}{2} & 0 \\ \frac{1}{2} & 0 & 1 \\ 0 & \frac{1}{2} & 0 \end{bmatrix} \end{array}$$

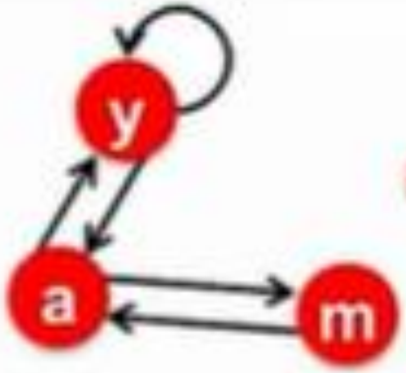
$r_y = r_y/2 + r_a/2$

$r_a = r_y/2 + r_m$

$r_m = r_a/2$

PageRank: How to solve?

- **Power Iteration:**
  - Set $r_j = 1/N$
  - **1:** $r'_j = \sum_{i \to j} \frac{r_i}{d_i}$
  - **2:** $r = r'$
  - If not converged: goto **1**
- **Example:**



$$M = \begin{array}{c|ccc} & y & a & m \\ \hline y & 1/2 & 1/2 & 0 \\ a & 1/2 & 0 & 1 \\ m & 0 & 1/2 & 0 \end{array}$$

$r_y = r_y/2 + r_a/2$

$r_a = r_y/2 + r_m$

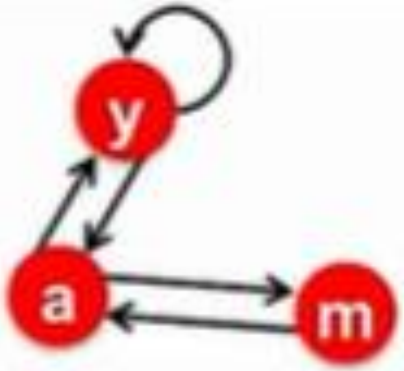
$r_m = r_a/2$

$$\begin{bmatrix} r_y \\ r_a \\ r_m \end{bmatrix} = \quad \begin{matrix} 1/3 & 1/3 \\ 1/3 & 3/6 \\ 1/3 & 1/6 \end{matrix}$$

Iteration 0, 1, 2, ...

PageRank: How to solve?

- **Power Iteration:**
  - Set $r_j = 1/N$
  - **1:** $r'_j = \sum_{i \to j} \frac{r_i}{d_i}$
  - **2:** $r = r'$
  - If not converged: goto **1**
- **Example:**



	y	a	m
y	½	½	0
a	½	0	1
m	0	½	0

$r_y = r_y/2 + r_a/2$
$r_a = r_y/2 + r_m$
$r_m = r_a/2$

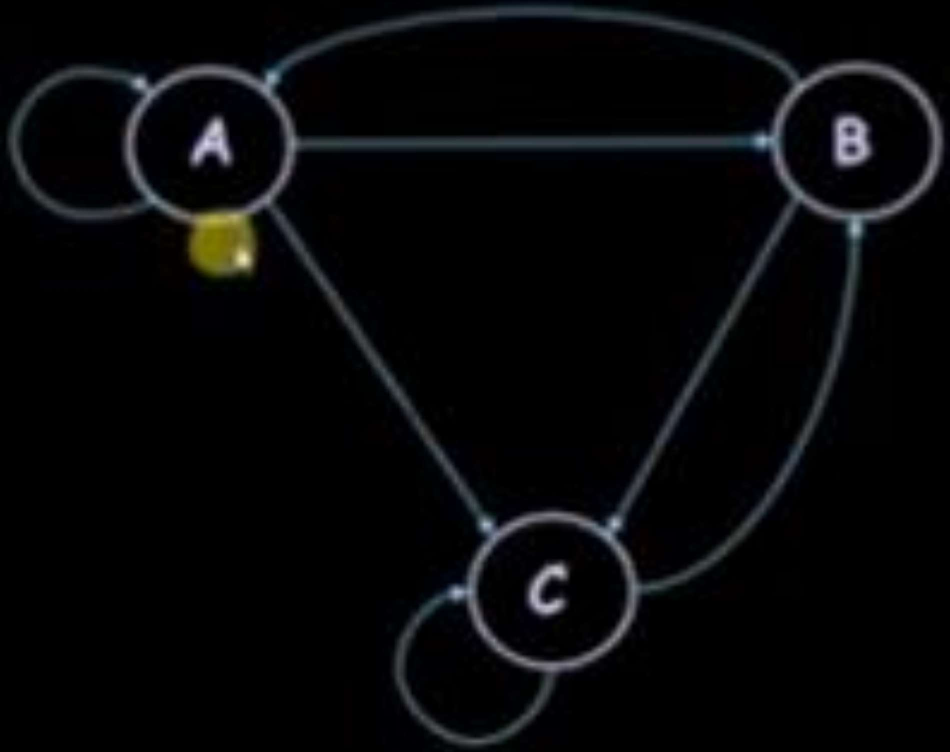
$$\begin{bmatrix} r_y \\ r_a \\ r_m \end{bmatrix} = \begin{matrix} 1/3 & 1/3 & 5/12 & 9/24 & & 6/15 = \frac{2}{5} \\ 1/3 & 3/6 & 1/3 & 11/24 & \dots & 6/15 \\ 1/3 & 1/6 & 3/12 & 1/6 & & 3/15 \end{matrix}$$

Iteration 0, 1, 2, …

PageRank Algorithm

Example

Compute the PageRank of each page in the following graph after 3 iterations



PageRank Iteration Using MapReduce

It is a Matrix Vector Multiplication. So we can use map reduce functions discussed before.

Matrix-Vector Multiplication by MapReduce

we have an n×n matrix M, whose element in row i and column j will be denoted mij . Suppose we also have a vector v of length n, whose jth element is vj .

$$x_i = \sum_{j=1}^{n} m_{ij} v_j$$

If n = 100, we do not want to use a DFS or MapReduce for this calculation.

Matrix-Vector Multiplication by MapReduce

n is large, but vector v can fit in main memory and thus be available to every Map task.

The matrix M and the vector v each will be stored in a file of the DFS.
We assume that the row-column coordinates of each matrix element will be discoverable, either from its position in the file, or because it is stored with explicit coordinates, as a triple (i, j,mij).

Matrix-Vector Multiplication by MapReduce

The Map Function:

Each Map task will operate on a chunk of the matrix M. From each matrix element mij it produces the key-value pair (i,mijvj). Thus, all terms of the sum that make up the component xi of the matrix-vector product will get the same key, i.

The Reduce Function: The Reduce function simply sums all the values associated with a given key i. The result will be a pair (i, xi).

If the Vector v Cannot Fit in Main Memory

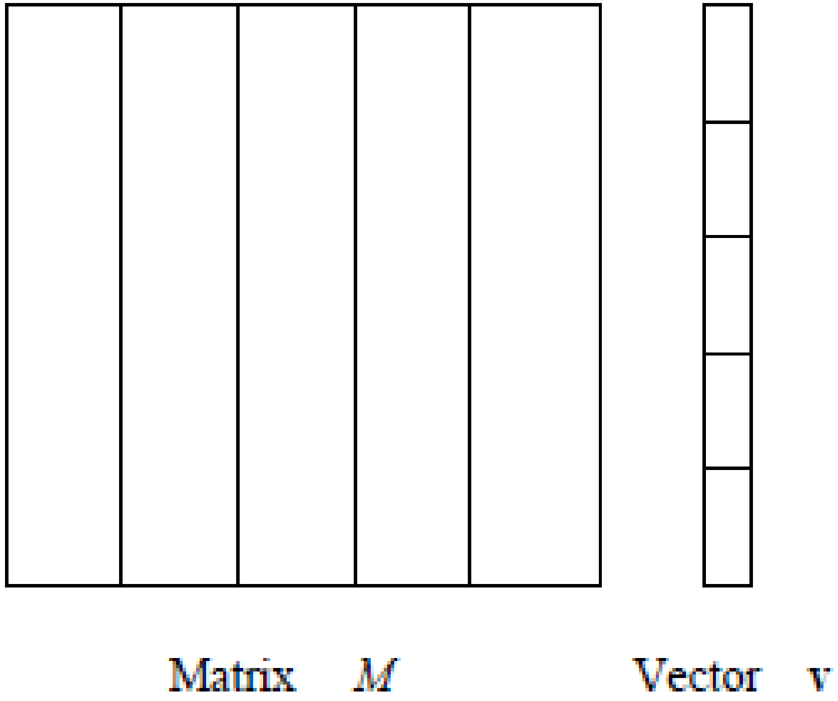


Figure 2.4: Division of a matrix and vector into five stripes

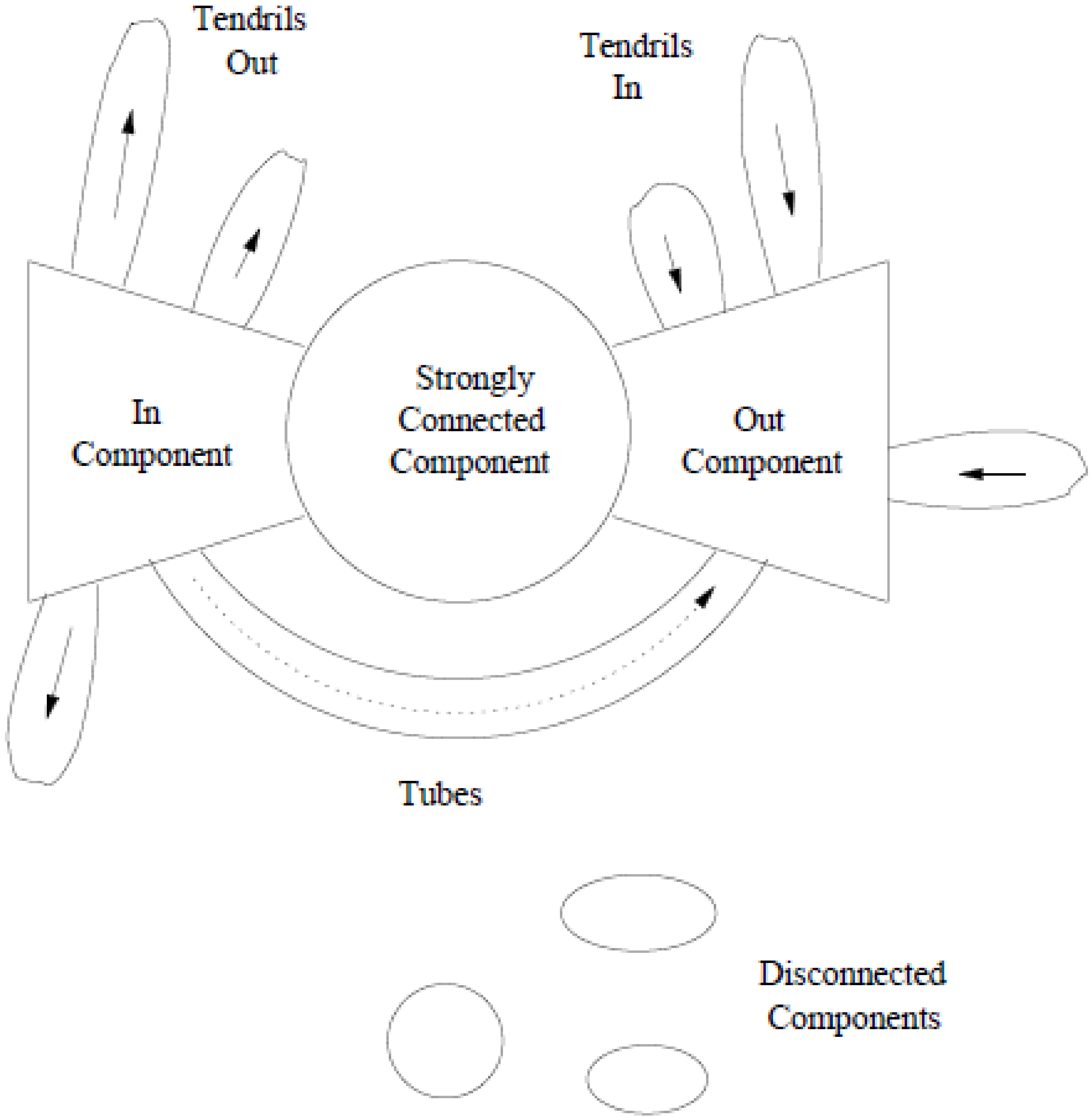
If the Vector v Cannot Fit in Main Memory

The ith stripe of the matrix multiplies only components from the ith stripe of the vector. Thus, we can divide the matrix into one file for each stripe, and do the same for the vector.

Each Map task is assigned a chunk from one of the stripes of the matrix and gets the entire corresponding stripe of the vector.

The Map and Reduce tasks can then act exactly as was described above for the case where Map tasks get the entire vector

Structure of the Web



Structure of the Web

- The **in-component,** consisting of pages that could reach the SCC by following links, but were not reachable from the SCC.
- The **out-component**, consisting of pages reachable from the SCC but unable to reach the SCC.
- **Tendrils**, which are of two types. Some tendrils consist of pages reachable from the in-component but not able to reach the in-component. The other tendrils can reach the out-component, but are not reachable from the out-component.

Structure of the Web

Tubes, which are pages reachable from the in-component and able to reach the out-component, but unable to reach the SCC or be reached from the SCC.

Isolated components that are unreachable from the large components (the SCC, in- and out-components) and unable to reach those components.

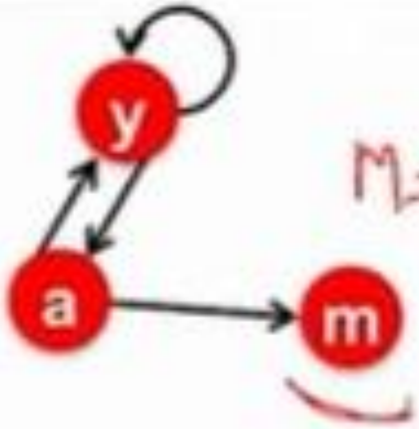
PageRank Problems

2 problems:

- **(1)** Some pages are **dead ends** (have no out-links)
  - Such pages cause importance to "leak out"
- **(2)** **Spider traps** (all out-links are within the group)
  - Eventually spider traps absorb all importance

Problem: Dead Ends

- **Power Iteration:**
  - Set $r_j = 1$
  - $r_j = \sum_{i \to j} \frac{r_i}{d_i}$
    - And iterate



$$M = \begin{array}{c|ccc} & y & a & m \\ \hline y & \frac{1}{2} & \frac{1}{2} & 0 \\ a & \frac{1}{2} & 0 & 0 \\ m & 0 & \frac{1}{2} & 0 \end{array}$$

$r_y = r_y/2 + r_a/2$

$r_a = r_y/2$

$r_m = r_a/2$

- **Example:**

$$\begin{pmatrix} r_y \\ r_a \\ r_m \end{pmatrix} = \begin{matrix} 1/3 & 2/6 & 3/12 & 5/24 & & 0 \\ 1/3 & 1/6 & 2/12 & 3/24 & \ldots & 0 \\ 1/3 & 1/6 & 1/12 & 2/24 & & 0 \end{matrix}$$

Iteration 0, 1, 2, …

Dead ends

If we allow dead ends, the transition matrix of the Web is no longer stochastic, since some of the columns will sum to 0 rather than 1.

A matrix whose column sums are at most 1 is called **substochastic.**

Dead ends

There are two approaches to dealing with dead ends.

1. We can drop the dead ends from the graph, and also drop their incoming arcs. Doing so may create more dead ends, which also have to be dropped, recursively. However, eventually we wind up with a strongly-connected component, none of whose nodes are dead ends.

2.We can modify the process by which random surfers are assumed to move about the Web. This method, which we refer to as "taxation," also solves the problem of spider traps.

Solution: Always Teleport

- **Teleports:** Follow random teleport links with probability **1.0** from dead-ends
  - Adjust matrix accordingly

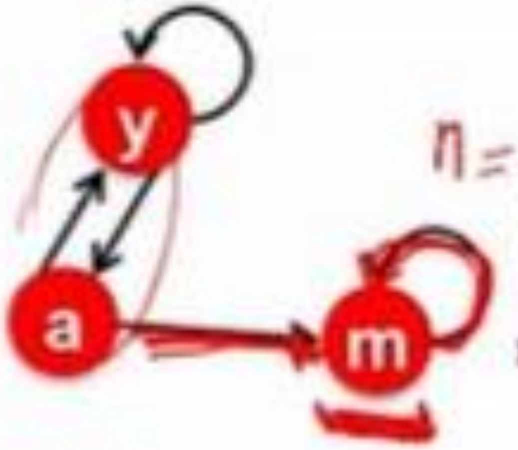
M =

	y	a	m
y	½	½	0
a	½	0	0
m	0	½	0

	y	a	m
y	½	½	⅓
a	½	0	⅓
m	0	½	⅓

Problem: Spider Traps

- **Power Iteration:**
  - Set $r_j = 1$
  - $r_j = \sum_{i \to j} \frac{r_i}{d_i}$
    - And iterate



M =

	y	a	m
y	½	½	0
a	½	0	0
m	0	½	1

$r_y = r_y/2 + r_a/2$

$r_a = r_y/2$

$r_m = r_a/2 + r_m$

- **Example:**

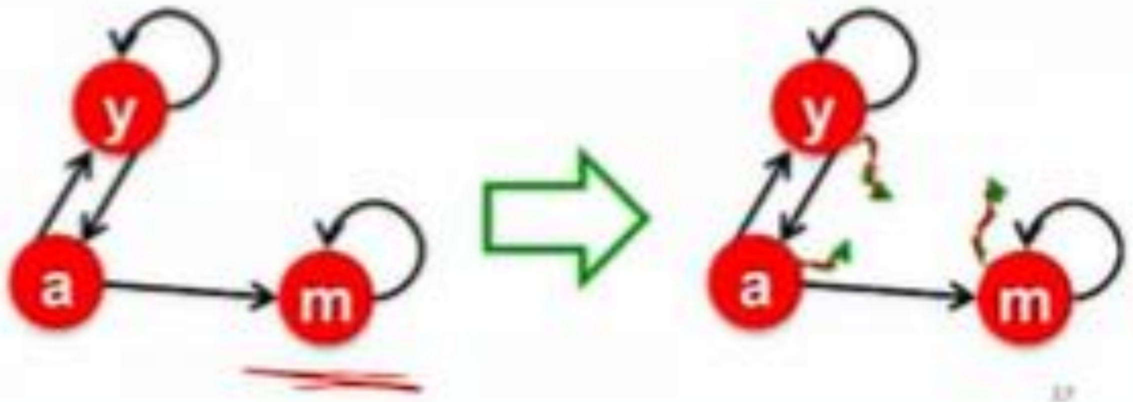
$$\begin{pmatrix} r_y \\ r_a \\ r_m \end{pmatrix} = \begin{matrix} 1/3 & 2/6 & 3/12 & 5/24 & & \boxed{0} \\ 1/3 & 1/6 & 2/12 & 3/24 & \ldots & \boxed{0} \\ 1/3 & 3/6 & 7/12 & 16/24 & & \boxed{1} \end{matrix}$$

Iteration 0, 1, 2, …

Solution: Random Teleports

- The Google solution for spider traps: At each time step, the random surfer has two options
  - With prob. β, follow a link at random
  - With prob. **1-**β, jump to some random page
  - Common values for β are in the range 0.8 to 0.9
- **Surfer will teleport out of spider trap within a few time steps**

$$\mathbf{v}' = \beta M\mathbf{v} + (1-\beta)\mathbf{e}/n$$



Example

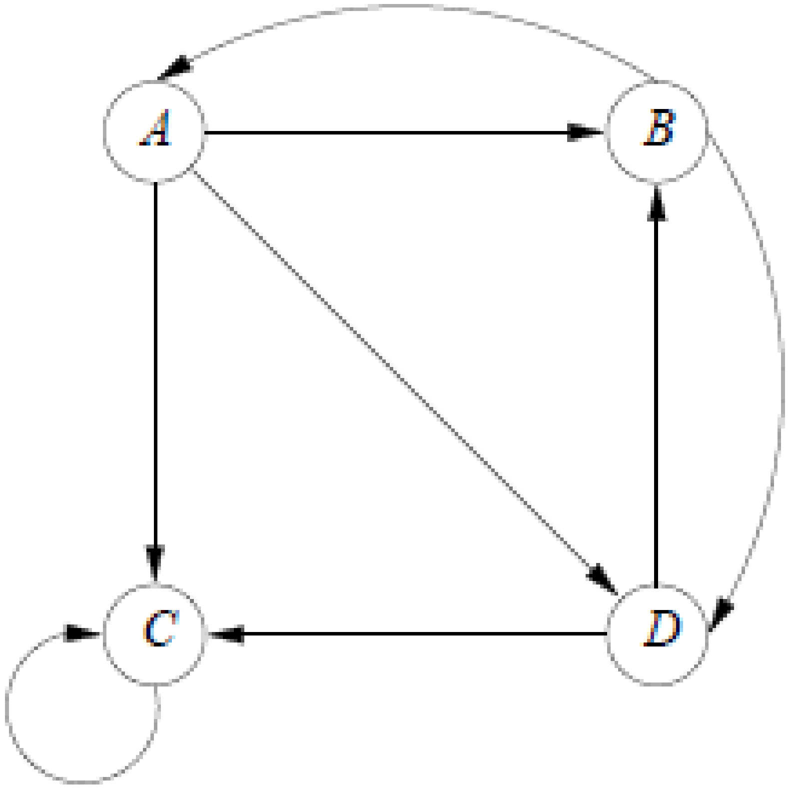


Figure 5.6: A graph with a one-node spider trap

The transition matrix for Fig. 5.6 is

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

Spider Traps and Taxation

get

$$\begin{bmatrix} 1/4 \\ 1/4 \\ 1/4 \\ 1/4 \end{bmatrix}, \begin{bmatrix} 3/24 \\ 5/24 \\ 11/24 \\ 5/24 \end{bmatrix}, \begin{bmatrix} 5/48 \\ 7/48 \\ 29/48 \\ 7/48 \end{bmatrix}, \begin{bmatrix} 21/288 \\ 31/288 \\ 205/288 \\ 31/288 \end{bmatrix}, \cdots, \begin{bmatrix} 0 \\ 0 \\ 1 \\ 0 \end{bmatrix}$$

- We shall use $\beta = 0.8$ in this example. This method, which we refer to as "taxation,"
- Notice that we have incorporated the factor β into M by multiplying each of its elements by 4/5. The components of the vector $(1 - \beta)e/n$ are each 1/20, since $1 - \beta = 1/5$ and $n = 4$. Here are the first few iterations.

Spider Traps and Taxation

$$\mathbf{v}' = \begin{bmatrix} 0 & 2/5 & 0 & 0 \\ 4/15 & 0 & 0 & 2/5 \\ 4/15 & 0 & 4/5 & 2/5 \\ 4/15 & 2/5 & 0 & 0 \end{bmatrix} \mathbf{v} + \begin{bmatrix} 1/20 \\ 1/20 \\ 1/20 \\ 1/20 \end{bmatrix}$$

Here are the first few iterations:

$$\begin{bmatrix} 1/4 \\ 1/4 \\ 1/4 \\ 1/4 \end{bmatrix}, \begin{bmatrix} 9/60 \\ 13/60 \\ 25/60 \\ 13/60 \end{bmatrix}, \begin{bmatrix} 41/300 \\ 53/300 \\ 153/300 \\ 53/300 \end{bmatrix}, \begin{bmatrix} 543/4500 \\ 707/4500 \\ 2543/4500 \\ 707/4500 \end{bmatrix}, \ldots, \begin{bmatrix} 15/148 \\ 19/148 \\ 95/148 \\ 19/148 \end{bmatrix}$$

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

Topic-Specific PageRank

- 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

Matrix Formulation

- 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 weighted all pages in the teleport set ***S*** equally
  - **Could also assign different weights to pages!**
- **Random Walk with Restart: S** is a single element
- **Compute as for regular PageRank:**
  - Multiply by ***M***, then add a vector
  - Maintains sparseness

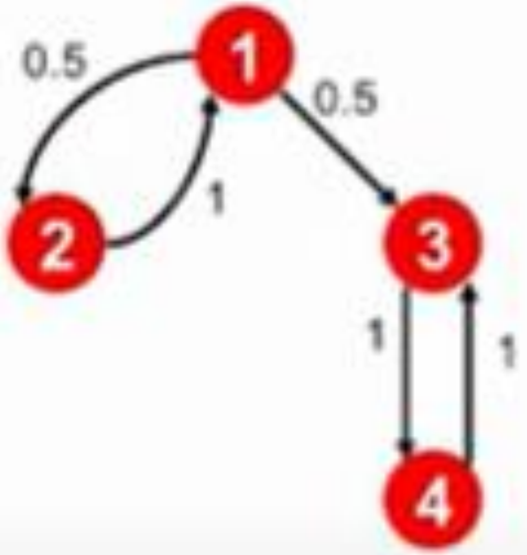
Topic-Sensitive PageRank

- The mathematical formulation for the iteration that yields topic-sensitive PageRank is similar to the equation we used for general PageRank.
- The only difference is how we add the new surfers.
- Suppose S is a set of integers consisting of the row/column numbers for the pages we have identified as belonging to a certain topic (called the **teleport** set).
- Let eS be a vector that has 1 in the components in S and 0 in other components. Then the topic-sensitive Page- Rank for S is the limit of the iteration.

$$\mathbf{v}' = \beta M \mathbf{v} + (1 - \beta)\mathbf{e}_S/|S|$$

Example

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

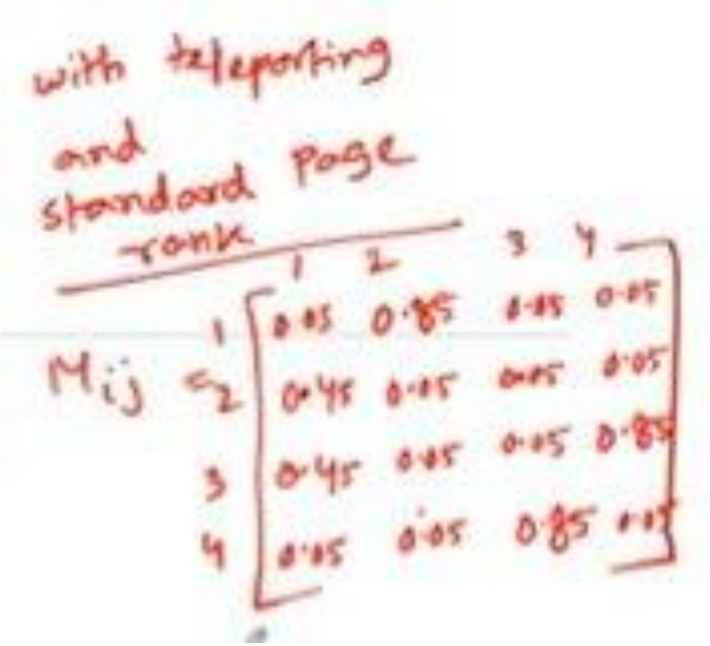


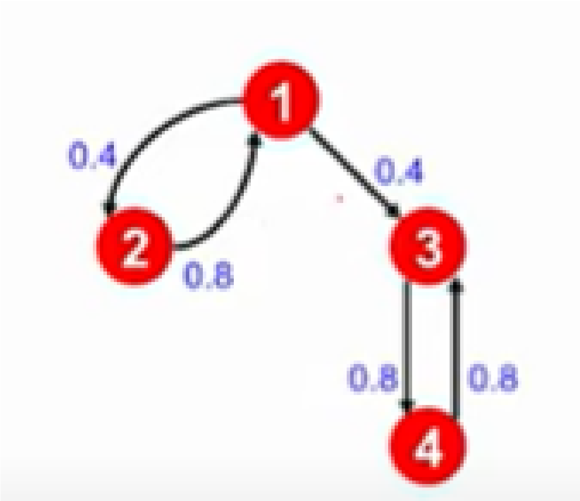
Example – without teleporting

without teleporting

$$M_{ij} \rightarrow \begin{array}{c|cccc} & 1 & 2 & 3 & 4 \\ \hline 1 & 0 & 1 & 0 & 0 \\ 2 & 0.5 & 0 & 0 & 0 \\ 3 & 0.5 & 0 & 0 & 1 \\ 4 & 0 & 0 & 1 & 0 \end{array}$$

Example –with teleporting – standard page rank





Example – page specific rank

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

$$M_{ij} = \begin{bmatrix} 0.2 & 1 & 0.2 & 0.2 \\ 0.4 & 0 & 0 & 0 \\ 0.4 & 0 & 0 & 0.8 \\ 0 & 0 & 0.8 & 0 \end{bmatrix}$$

$$r_0 = \begin{bmatrix} 0.25 \\ 0.25 \\ 0.25 \\ 0.25 \end{bmatrix}$$

Node	Iteration 0	1	2	…	stable
1	0.25	0.4	0.28		0.294
2	0.25	0.1	0.16		0.118
3	0.25	0.3	0.32		0.327
4	0.25	0.2	0.24		0.261

Example – page specific rank

- $S = \{1,2,3,4\}$, $\beta = 0.8$
  - $r = [0.13, 0.10, 0.39, 0.36]$
- $S = \{1,2,3\}$, $\beta = 0.8$
  - $r = [0.17, 0.13, 0.38, 0.3]$
- $S = \{1,2\}$, $\beta = 0.8$
  - $r = [0.26, 0.20, 0.29, 0.23]$
- $S = \{1\}$, $\beta = 0.8$
  - $r = [0.29, 0.11, 0.32, 0.26]$

- $S = \{1\}$, $\beta = 0.8$
  - $r = [0.29, 0.11, 0.32, 0.26]$
- $S = \{1\}$, $\beta = 0.9$
  - $r = [0.17, 0.07, 0.4, 0.36]$
- $S = \{1\}$, $\beta = 0.7$
  - $r = [0.39, 0.14, 0.24, 0.19]$

Example – page specific rank

[Topic specific page rank (part 1)]

[topic specific page rank (part 2)]

[Topic-specific page rank (part 3)]

Example

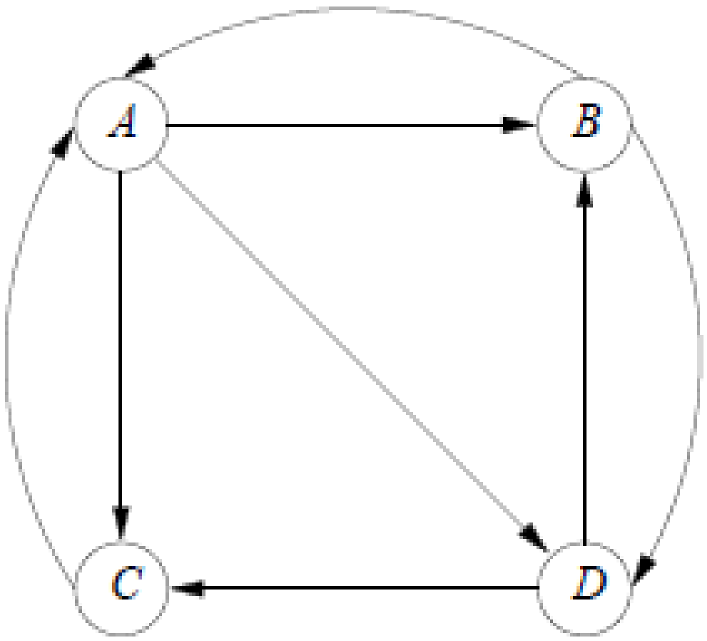


Figure 5.15: Repeat of example Web graph

Example

Example 5.10 : Let us reconsider the original Web graph we used in Fig. 5.1, which we reproduce as Fig. 5.15. Suppose we use $\beta = 0.8$. Then the transition matrix for this graph, multiplied by β, is

$$\beta M = \begin{bmatrix} 0 & 2/5 & 4/5 & 0 \\ 4/15 & 0 & 0 & 2/5 \\ 4/15 & 0 & 0 & 2/5 \\ 4/15 & 2/5 & 0 & 0 \end{bmatrix}$$

Example

- Suppose that our topic is represented by the teleport set $S = \{B,D\}$. Then the vector $(1 - \beta)e_S/|S|$ has 1/10 for its second and fourth components and 0 for the other two components.
- The reason is that $1 - \beta = 1/5$, the size of S is 2, and e_S has 1 in the components for B and D and 0 in the components for A and C.

Example

- Thus, the equation that must be iterated is

$$\mathbf{v}' = \begin{bmatrix} 0 & 2/5 & 4/5 & 0 \\ 4/15 & 0 & 0 & 2/5 \\ 4/15 & 0 & 0 & 2/5 \\ 4/15 & 2/5 & 0 & 0 \end{bmatrix} \mathbf{v} + \begin{bmatrix} 0 \\ 1/10 \\ 0 \\ 1/10 \end{bmatrix}$$

Example

- Here are the first few iterations of this equation.
- We have also started with the surfers only at the pages in the teleport set. Although the initial distribution has no effect on the limit, it may help the computation to converge faster.

$$\begin{bmatrix} 0/2 \\ 1/2 \\ 0/2 \\ 1/2 \end{bmatrix}, \begin{bmatrix} 2/10 \\ 3/10 \\ 2/10 \\ 3/10 \end{bmatrix}, \begin{bmatrix} 42/150 \\ 41/150 \\ 26/150 \\ 41/150 \end{bmatrix}, \begin{bmatrix} 62/250 \\ 71/250 \\ 46/250 \\ 71/250 \end{bmatrix}, \ldots, \begin{bmatrix} 54/210 \\ 59/210 \\ 38/210 \\ 59/210 \end{bmatrix}$$

On-Line Algorithms

- Before addressing the question of matching advertisements to search queries, we shall digress slightly by examining the general class to which such algorithms belong.
- This class is referred to as **“on-line,”** and they generally involve an approach called **“greedy.”**
- All the data needed by the algorithm is presented initially. The algorithm **can access the data in any order.** At the end, the algorithm produces its answer. Such an algorithm is called **off-line.**

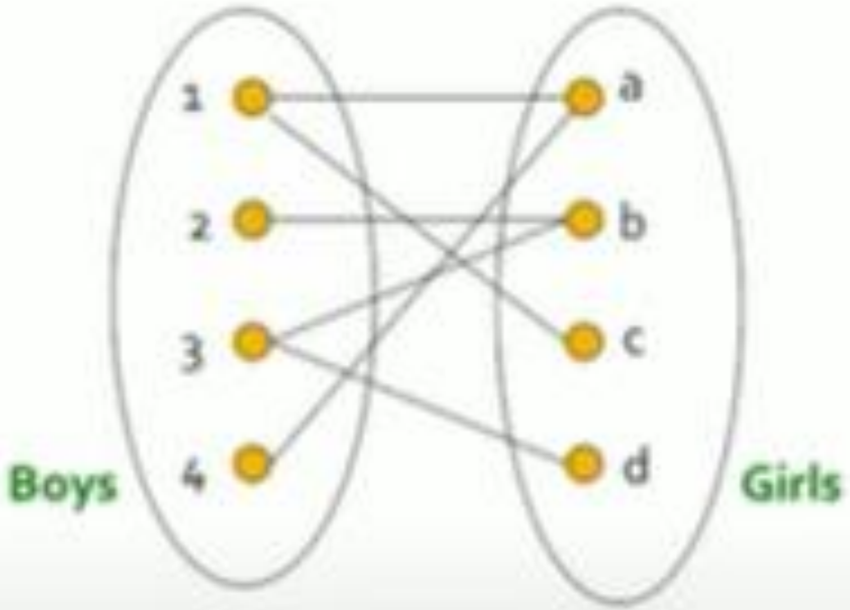
On-Line Algorithms

- There is an extreme form of stream processing, where we must respond with an output after each stream element arrives.
- We thus must decide about each stream element knowing nothing at all of the future. Algorithms of this class are called **on-line algorithms.**

The Matching Problem

- The problem of **matching ads to search queries.**
- This problem, called **“maximal matching,”** is an abstract problem involving bipartite graphs (graphs with two sets of nodes – left and right – with all edges connecting a node in the left set to a node in the right set.

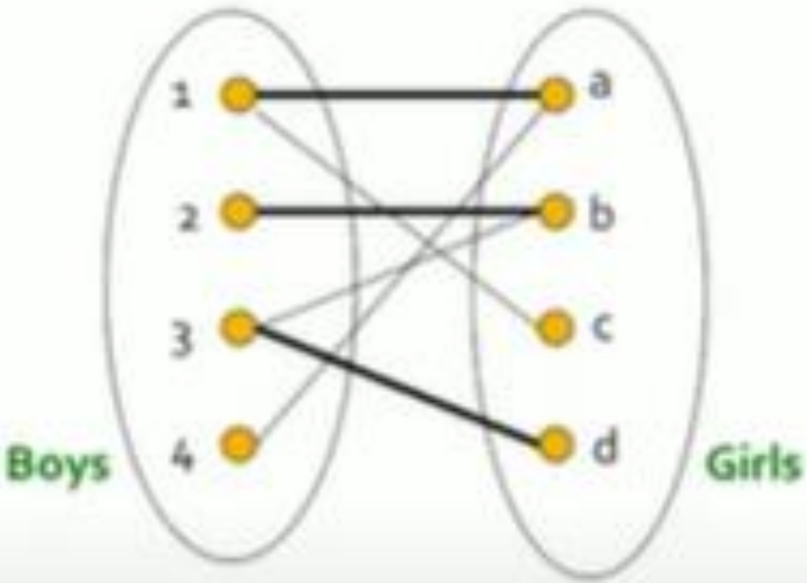
Example: Bipartite Matching



Nodes: Boys and Girls; Edges: Compatible Pair

Goal: Match as many compatible pairs as pos

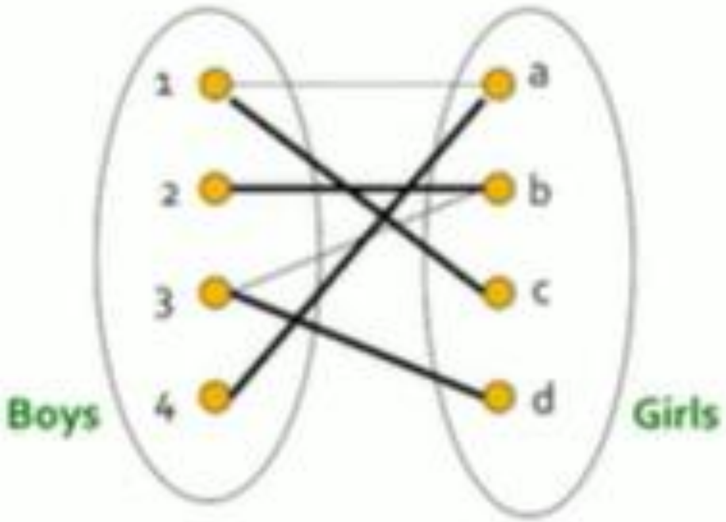
Example: Bipartite Matching



M = {(1,a),(2,b),(3,d)} is a **matching**

Cardinality of matching = |M| = 3

Example: Bipartite Matching



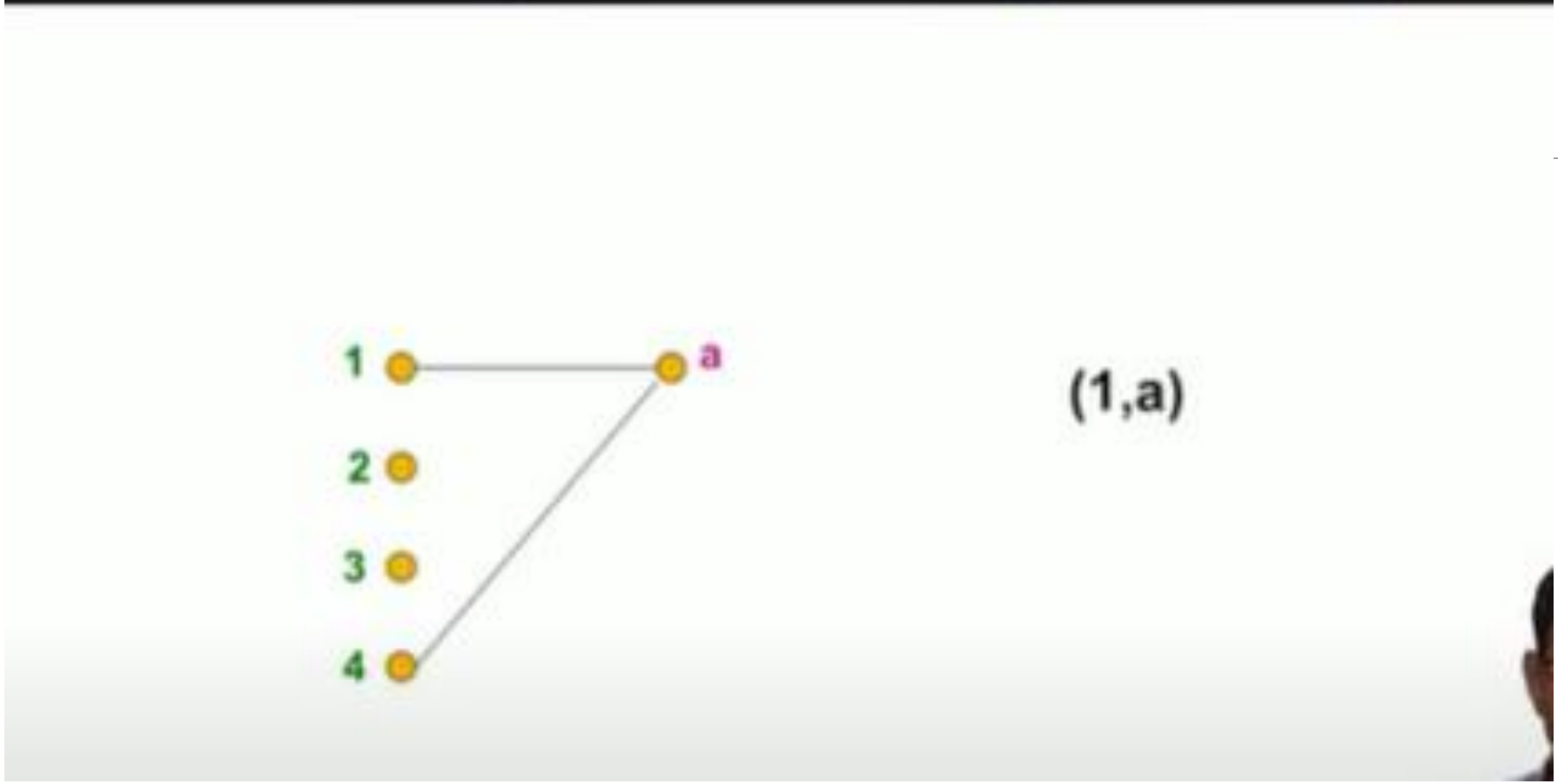
M = {(1,c),(2,b),(3,d),(4,a)} is a **perfect matching**

Perfect matching … all vertices of the graph are matched
Maximum matching … a matching that contains the largest possible number of matches

Matching Algorithm

- **Problem: Find a maximum matching for a given bipartite graph**
  - A perfect one if it exists
- There is a polynomial-time offline algorithm based on augmenting paths (Hopcroft & Karp 1973, see http://en.wikipedia.org/wiki/Hopcroft-Karp_algorithm)

Online Graph Matching: Example



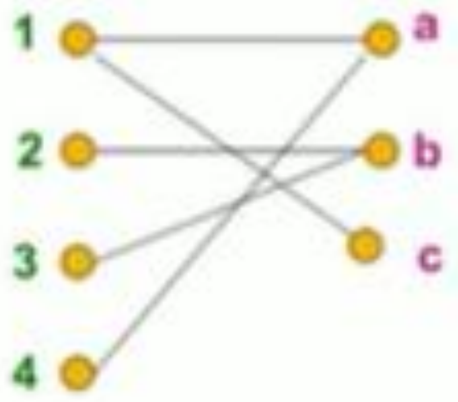
a can be matched with 1 and 4, but we choose **(1,a)**

Online Graph Matching: Example

(1,a)
(2,b)

b can be matched with 2 and 3, but we choose **(2,b)**

Online Graph Matching: Example



(1,a)
(2,b)

c can be matched with 1, but we cannot choose **(1,c)**

Online Graph Matching: Example

(1,a)
(2,b)
(3,d)

- d can be matched with 3, but we choose **(3,d)**
- It is not a perfect matching or best matching. So this Matching problem is a heuristic problem.

Greedy Algorithm for Maximal Matching

- In particular, the greedy algorithm for maximal matching works as follows.
- We consider the edges in whatever order they are given. **When we consider (x, y), add this edge to the matching if neither x nor y are ends of any edge selected for the matching so far. Otherwise, skip (x, y).**

Competitive Ratio

- For input I, suppose greedy produces matching M_{greedy} while an optimal matching is M_{opt}

Competitive ratio =

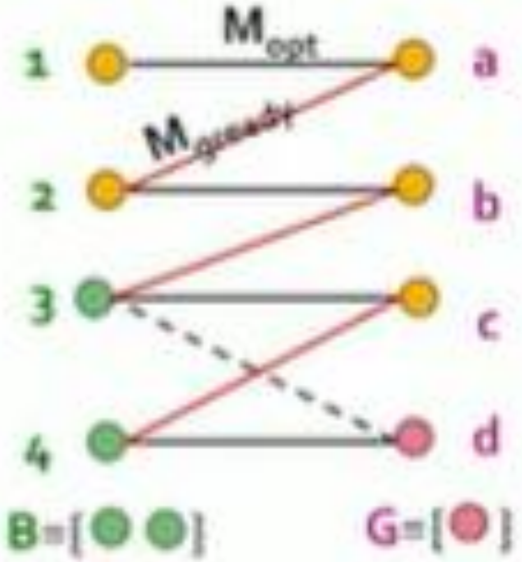
$$min_{all\ possible\ inputs\ I}\ (|M_{greedy}|/|M_{opt}|)$$

(what is greedy's <u>worst</u> performance <u>over all possible</u> inputs I)

It is defined as the worst-case ratio between the cost of the solution produced by the online algorithm and the cost of an optimal solution, over all possible inputs.

Analyzing the Greedy Algorithm

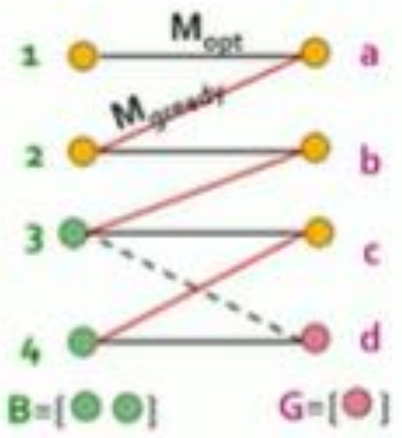
- Suppose $M_{greedy} \neq M_{opt}$
- Consider the set G of girls matched in M_{opt} but not in M_{greedy}
- (1) $|M_{opt}| \leq |M_{greedy}| + |G|$



- Here, Cardinality of greedy is 3 and cardinality of optimal is 4
- So, 4<=3+1

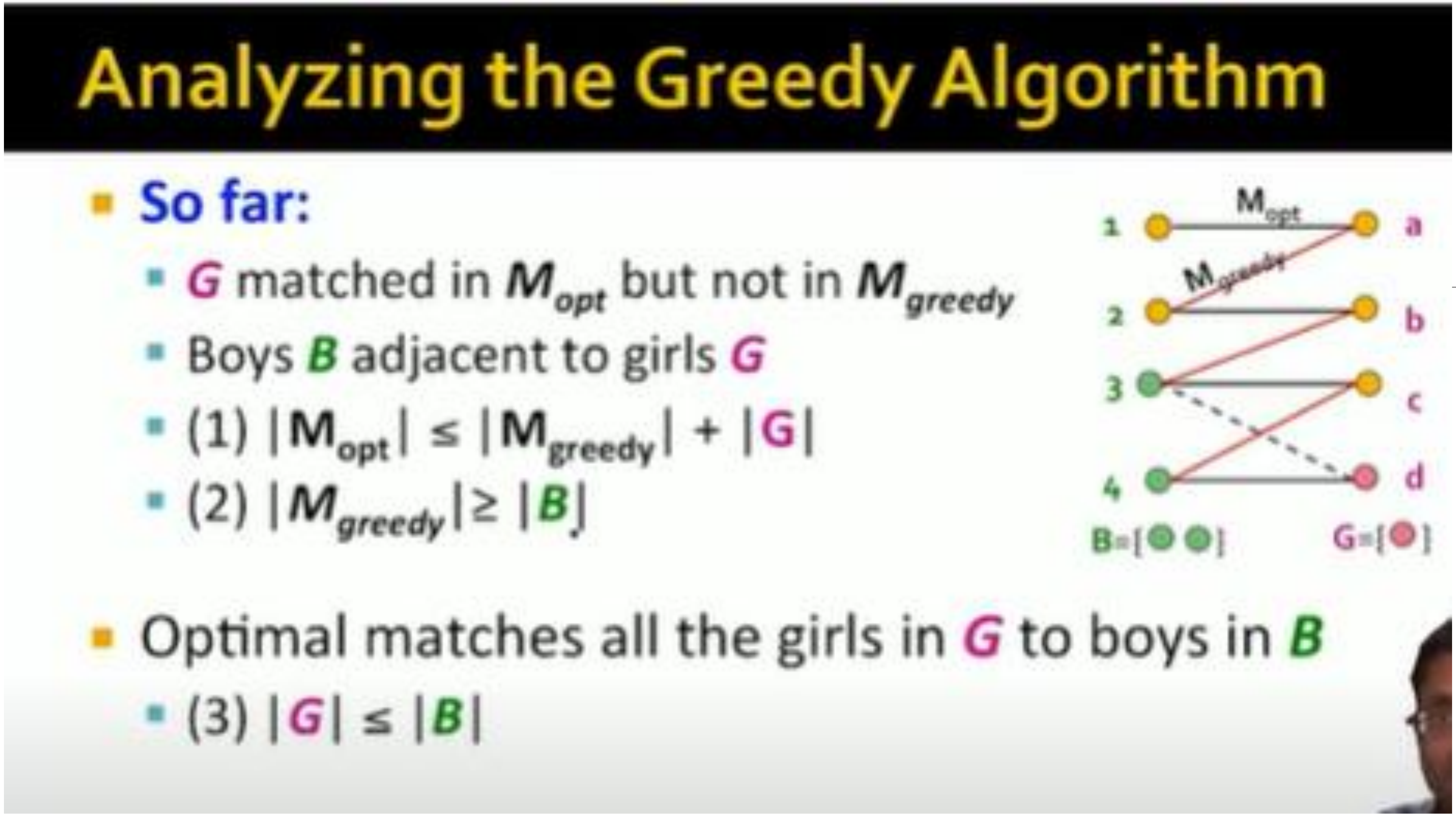
Analyzing the Greedy Algorithm

- Suppose $M_{greedy} \neq M_{opt}$
- Consider the set G of girls matched in M_{opt} but not in M_{greedy}
- (1) $|M_{opt}| \leq |M_{greedy}| + |G|$



- Every boy B adjacent to girls in G is already matched in M_{greedy}
- (2) $|M_{greedy}| \geq |B|$

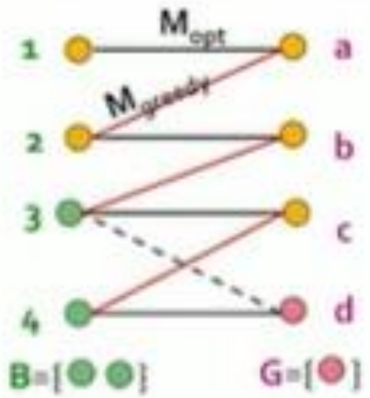
- Here, Cardinality of greedy is 3 and cardinality of optimal is 4
- So, 3>=2



- Here, Cardinality of greedy is 3 and cardinality of optimal is 4
- So, 1<=2

Analyzing the Greedy Algorithm

- **So far:**
  - *G* matched in M_{opt} but not in M_{greedy}
  - Boys *B* adjacent to girls *G*
  - (1) $|M_{opt}| \le |M_{greedy}| + |G|$
  - (2) $|M_{greedy}| \ge |B|$



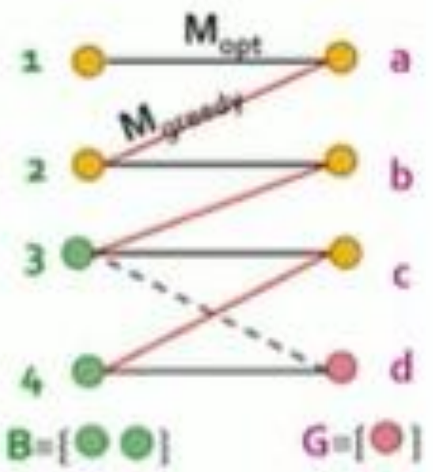
- Optimal matches all the girls in *G* to boys in *B*
  - (3) $|G| \le |B|$

- Combining (2) and (3):
  - (4) $|G| \le |B| \le |M_{greedy}|$

- I.e , 1<=2<=3

Analyzing the Greedy Algorithm

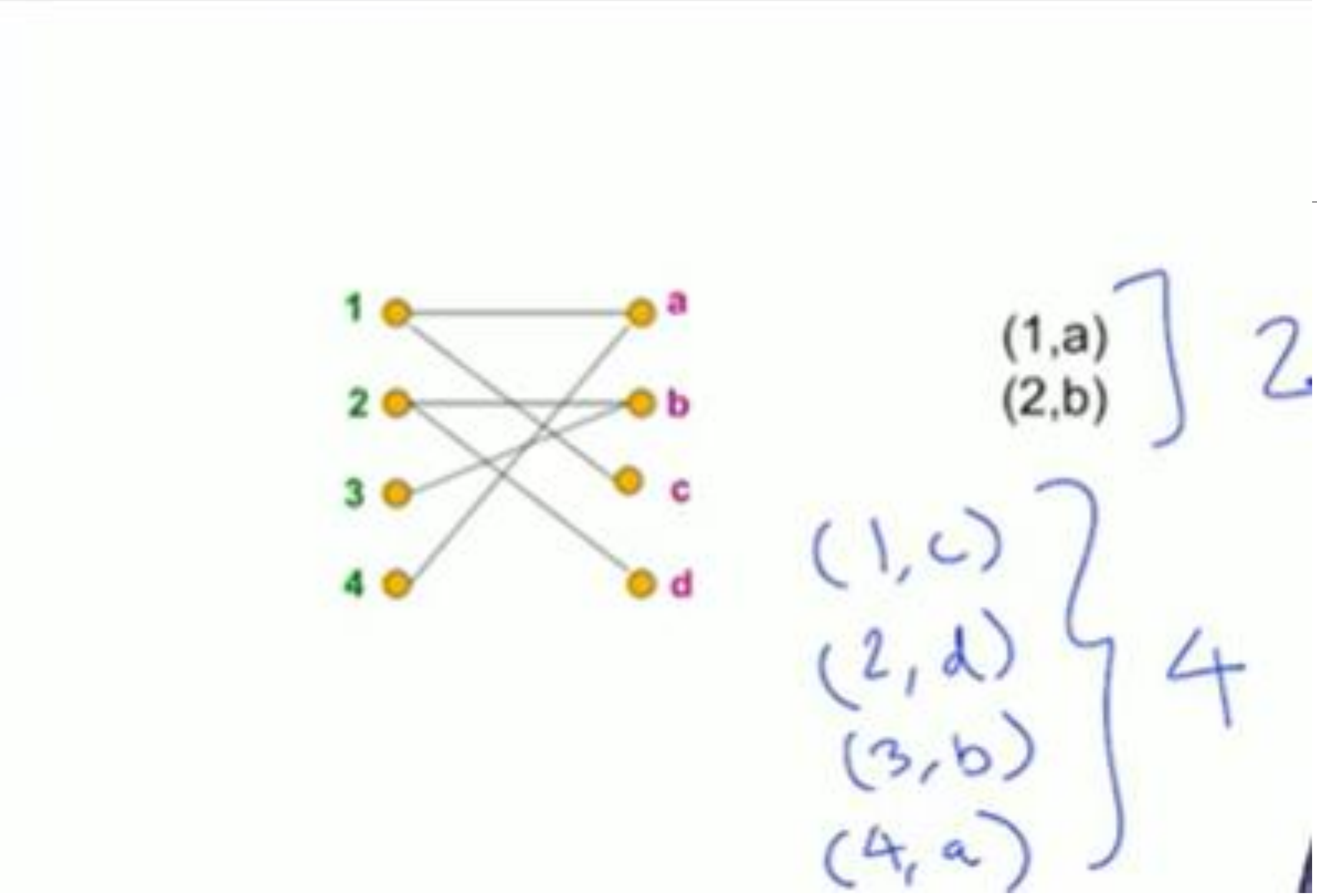
- So we have:
  - (1) $|M_{opt}| \leq |M_{greedy}| + |G|$
  - (4) $|G| \leq |B| \leq |M_{greedy}|$



- Combining (1) and (4):
  - $|M_{opt}| \leq |M_{greedy}| + |M_{greedy}|$
  - $|M_{opt}| \leq 2|M_{greedy}|$
  - $|M_{greedy}|/|M_{opt}| \geq 1/2$

- I.e competitive ratio of greedy algorithm is at-least ½.

Worst-case Scenario



- M Opt - (1,c), (2,d), (3,b) and (4,a)
- Mgreedy = (1,a), (2,b)

History of Web Advertising

- **Banner ads** (1995-2001)
  - Initial form of web advertising
  - Popular websites charged *X*$ for every 1,000 "impressions" of the ad
    - Called "**CPM**" rate (Cost per thousand impressions)
    - Modeled similar to TV, magazine ads
  - From **untargeted** to **demographically targeted**
  - **Low click-through rates**
    - Low ROI for advertisers

Performance-based Advertising

- **Introduced by Overture around 2000**
  - Advertisers **bid** on **search keywords**
  - When someone searches for that keyword, the **highest bidder's ad is shown**
  - Advertiser is charged only if the ad is clicked on
- Similar model adopted by Google with some changes around 2002
  - Called **Adwords**

Algorithmic Challenges

- **Performance-based advertising works!**
  - Multi-billion-dollar industry
- **What ads to show for a given query?**
  - (Today's lecture)
- **If I am an advertiser, which search terms should I bid on and how much should I bid?**
  - (Not focus of today's lecture)

AdWords Problem

- A stream of queries arrives at the search engine: $q_1, q_2, \ldots$
- Several advertisers bid on each query
- When query q_i arrives, search engine must pick a subset of advertisers whose ads are shown
- **Goal: Maximize search engine's revenues**
- **Clearly we need an online algorithm!**

Expected Revenue

Advertiser	Bid
A	$1.00
B	$0.75
C	$0.50

- A has high Bid, So Search Engine may use A.
- Sorted order is A, B, C

Expected Revenue

Advertiser	Bid	CTR	Bid * CTR
A	$1.00	1%	1 cent
B	$0.75	2%	1.5 cents
C	$0.50	2.5%	1.125 cents
		Click through rate	Expected revenue

- Google observed that taking expected revenue gives more revenue.
- Sorted order is B, C, A

Adwords Problem

- **Given:**
  - A set of bids by advertisers for search queries
  - A click-through rate for each advertiser-query pair
  - A budget for each advertiser (say for 1 day, month...)
  - A limit on the number of ads to be displayed with each search query
- **Respond to each search query with a set of advertisers such that:**
  - The size of the set is no larger than the limit on the number of ads per query
  - Each advertiser has bid on the search query
  - Each advertiser has enough budget left to pay for the ad if it is clicked upon

Limitations of Simple Algorithm

Instead of sorting advertisers by bid, sort by expected revenue!

Advertiser	Bid	CTR	Bid * CTR
B	$0.75	2%	1.5 cents
C	$0.50	2.5%	1.125 cents
A	$1.00	1%	1 cent

- CTR of an ad is unknown
- Advertisers have limited budgets and bid on multiple ads (BALANCE algorithm)

Estimating CTR

- Clickthrough rate (CTR) for a query-ad pair is measured historically
  - Averaged over a time period
- Some complications we won't cover in this lecture
  - CTR is position dependent
    - Ad #1 is clicked more than Ad #2
  - Explore v Exploit: Keep showing ads we already know the CTR of, or show new ads to estimate their CTR?

Estimating CTR

- Clickthrough rate (CTR) for a query-ad pair is measured historically
  - Averaged over a time period
- Some complications we won't cover in this lecture
  - CTR is position dependent
    - Ad #1 is clicked more than Ad #2
  - Explore v Exploit: Keep showing ads we already know the CTR of, or show new ads to estimate their CTR?

Among set of ads for a search query, New Ad may have less CTR.

Dealing with Limited Budgets

- Our setting: Simplified environment
  - There is **1** ad shown for each query
  - All advertisers have the same budget ***B***
  - All ads are equally likely to be clicked
  - Value of each ad is the same (=**1**)
- Simplest algorithm is greedy:
  - For a query pick any advertiser who has bid **1** for that query
  - Competitive ratio of greedy is 1/2

Bad Scenario for Greedy

- Two advertisers A and B
  - *A* bids on query ***x***, *B* bids on ***x*** and ***y***
  - Both have budgets of **$4**
- Query stream: ***x x x x y y y y***
  - Worst case greedy choice: ***B B B B*** _ _ _ _
  - Optimal: **A A A A B B B B**
  - **Competitive ratio = ½**
- This is the worst case!
  - **Note:** Greedy algorithm is deterministic – it always resolves draws in the same way

BALANCE Algorithm [MSVV]

- **BALANCE** Algorithm by Mehta, Saberi, Vazirani, and Vazirani
  - **For each query, pick the advertiser with the largest unspent budget**
  - Break ties arbitrarily (**but in a deterministic way)**

Example: BALANCE

- Two advertisers A and B
  - **A** bids on query ***x***, **B** bids on ***x*** and ***y***
  - Both have budgets of **$4**
- Query stream: ***x x x x y y y y***

A B

3 4

A

Example: BALANCE

- Two advertisers A and B
  - **A** bids on query ***x***, **B** bids on ***x*** and ***y***
  - Both have budgets of **$4**
- Query stream: ***x x x x y y y y***

A B
3 ~~4~~3

A B

Example: BALANCE

- Two advertisers A and B
  - **A** bids on query $\boldsymbol{x}$, **B** bids on $\boldsymbol{x}$ and $\boldsymbol{y}$
  - Both have budgets of **\$4**
- Query stream: $\boldsymbol{x\ x\ x\ x\ y\ y\ y\ y}$

A B

2 ~~3~~ ~~4~~ ~~3~~ ~~2~~

~~1~~ 0

A B A B B B _ _ →

Example: BALANCE

- Two advertisers A and B
  - **A** bids on query ***x***, **B** bids on ***x*** and ***y***
  - Both have budgets of **$4**
- Query stream: *x x x x y y y y*

A B

2 ~~3~~ ~~4~~ ~~3~~ ~~2~~

~~1~~ 0

A B A B B B _ _

$$\frac{\text{BAL} = \$6}{\text{OPT} = \$8} = \frac{3}{4}$$

Competitive ratio is ¾ , which is better that ½.

Example: BALANCE

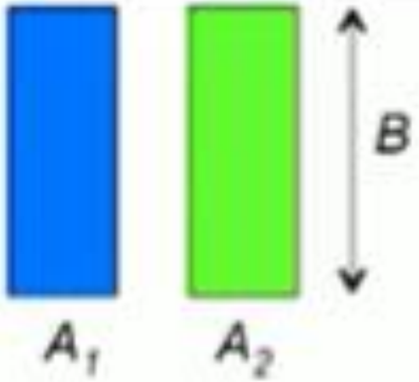
- Two advertisers A and B
  - A bids on query *x*, B bids on *x* and *y*
  - Both have budgets of **$4**
- Query stream: *x x x x y y y y*
- BALANCE choice: A B A B B B _ _
  - Optimal: **A A A A B B B B**
- Competitive ratio = ¾
  - **For BALANCE with 2 advertisers**

Analyzing 2-advertiser BALANCE

- Consider simple case
  - **2** advertisers, A_1 and A_2, each with budget B (≥ 1)
  - Optimal solution exhausts both advertisers' budgets

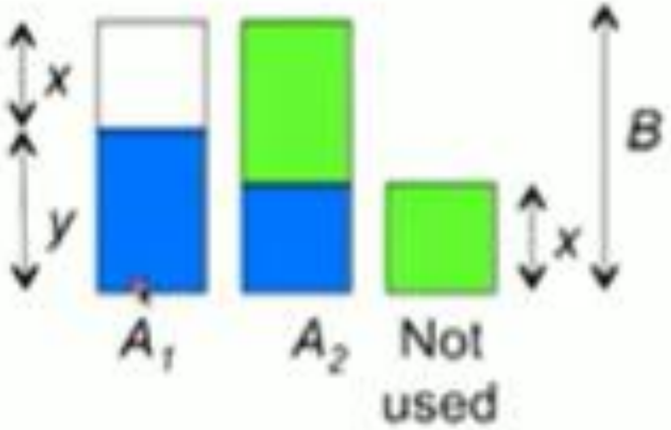
- **BALANCE must exhaust at least one advertiser's budget:**
  - **If not, we can allocate more queries**
  - Assume BALANCE exhausts A_2's budget

Analyzing Balance



Queries allocated to $\boldsymbol{A_1}$ in the optimal solution

Queries allocated to $\boldsymbol{A_2}$ in the optimal solution



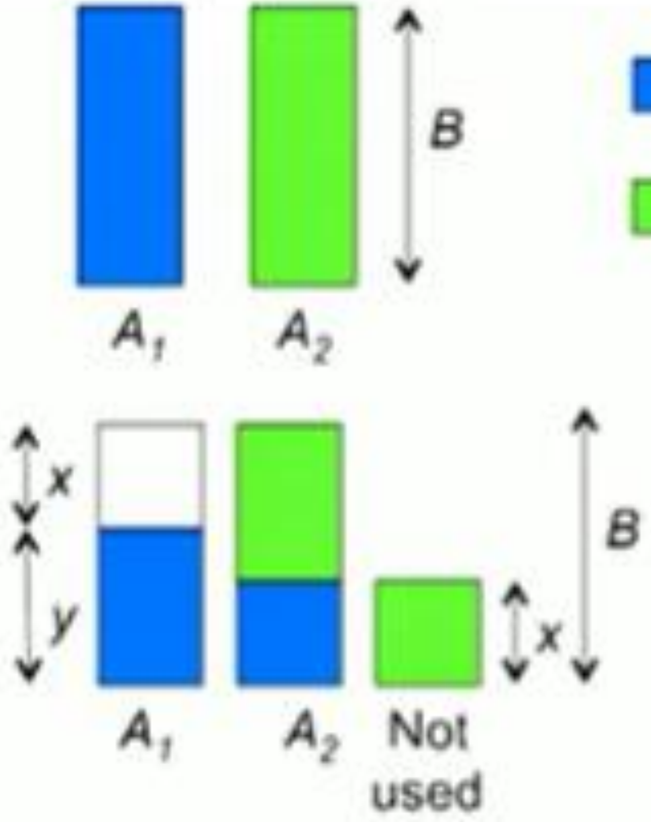
Optimal revenue = **2B**
Balance revenue = **B+y**
Goal: Show that $y \geq B/2$

Case 1: BALANCE assigns at least B/2 blue queries to A_1. So $y \geq B/2$.

Case 2: BALANCE assigns more than B/2 blue queries to A_2.

Case 2: BALANCE assigns more than B/2 blue queries to A_2.
Consider the last blue query assigned to A_2.
At that time, A_2's unspent budget must have been at least as big as A_1's.
That means at least as many queries have been assigned to A_1 as to A_2.
At this point, we have already assigned at least B/2 queries to A_2.

Analyzing BALANCE



Queries allocated to A_1 in the optimal solution

Queries allocated to A_2 in the optimal solution

Optimal revenue OPT = **2B**
Balance revenue BAL = **B+y**

We have shown that $y \geq B/2$
BAL $\geq$ B+B/2 = 3B/2
BAL/OPT $\geq$ 3/4

BALANCE: General Result

- In the general case, worst competitive ratio of BALANCE is 1–1/e = approx. 0.63
  - Interestingly, no online algorithm has a better competitive ratio!
- Let's see the worst case example that gives this ratio