Mining Massive Datasets

Lecture 12

Artur Andrzejak
http://pvs.ifi.uni-heidelberg.de

RUPRECHT-KARLS-
UNIVERSITÄT
HEIDELBERG

Note on Slides

A substantial part of these slides come (either verbatim or in a modified form) from the book Mining of Massive Datasets by Jure Leskovec, Anand Rajaraman, Jeff Ullman (Stanford University).
For more information, see the website accompanying the book: http://www.mmds.org.

Infinite Data

High dim. data	Graph data	Infinite data	Machine learning	Apps
Locality sensitive hashing	PageRank, SimRank	Filtering data streams	SVM	Recommen der systems
Clustering	Community Detection	Web advertising	Decision Trees	Association Rules
Dimensio- nality reduction	Spam Detection	Queries on streams	Perceptron, kNN	Duplicate document detection

Programming in Spark & MapReduce

Theory of Stream Processing

- **Last lecture:**
  - **Stream filtering**
  - **Sampling a fixed proportion of a stream**
    - Sample size grows as the stream grows
- **This lecture:**
  - **Sampling a fixed-size sample**
    - Reservoir sampling

Sampling from a Data Stream: Sampling a fixed-size sample

Maintaining a fixed-size sample

- **Problem 2: Fixed-size sample**
- Suppose we need to maintain a random sample S of **size exactly *s* tuples**
  - Why? Don't know length of stream in advance
- Suppose at time n we have seen n items
  - Each item is in the sample S with equal prob. s/n

How to think about the problem: say s = 2

Stream: a x c y z k c d e g…

At **n= 5,** each of the first 5 tuples is included in the sample **S** with equal prob.

At **n= 7,** each of the first 7 tuples is included in the sample **S** with equal prob.

Impractical solution would be to store all the *n* tuples seen so far and out of them pick *s* at random

Solution: Fixed Size Sample

- **Algorithm (a.k.a. Reservoir Sampling)**
  - Store all the first ***s*** elements of the stream to ***S***
  - Suppose we have seen ***n-1*** elements, and now the ***n***th element arrives (***n*** > ***s***)
    - With probability ***s/n***, keep the ***n***th element, else discard it
    - If we picked the ***n***th element, then it replaces one of the ***s*** elements in the sample ***S***, picked uniformly at random
- **Claim:** This algorithm maintains a sample ***S*** with the desired property:
  - After ***n*** elements, the sample contains each element seen so far with probability ***s/n***

Proof: By Induction

- **We prove this by induction:**
  - Assume that after ***n*** elements, the sample contains each element seen so far with probability ***s/n***
  - We need to show that after seeing element ***n+1*** the sample maintains the property
    - Sample contains each element seen so far with probability ***s/(n+1)***
- **Base case:**
  - After we see **n=s** elements the sample **S** has the desired property
    - Each out of **n=s** elements is in the sample with probability ***s/s = 1***

Proof: By Induction

- **Inductive hypothesis:** After ***n*** elements, the sample ***S*** contains each element seen so far with prob. ***s/n***
- **Now element *n+1* arrives**
- **Inductive step:** For elements already in ***S***, probability that the algorithm keeps it in ***S*** is:

$$\underbrace{\left(1-\frac{s}{n+1}\right)}_{\text{Element } \mathbf{n+1} \text{ discarded}}+\underbrace{\left(\frac{s}{n+1}\right)}_{\text{Element } \mathbf{n+1} \text{ not discarded}}\underbrace{\left(\frac{s-1}{s}\right)}_{\text{Element in the sample not picked}}=\frac{n}{n+1}$$

- So, at time ***n,*** tuples in ***S*** were there with prob. **s/n**
- Time ***n→n+1,*** tuple stayed in ***S*** with prob. **n/(n+1)**
- So prob. tuple is in ***S*** at time ***n+1*** $= \frac{s}{n} \cdot \frac{n}{n+1} = \frac{s}{n+1}$

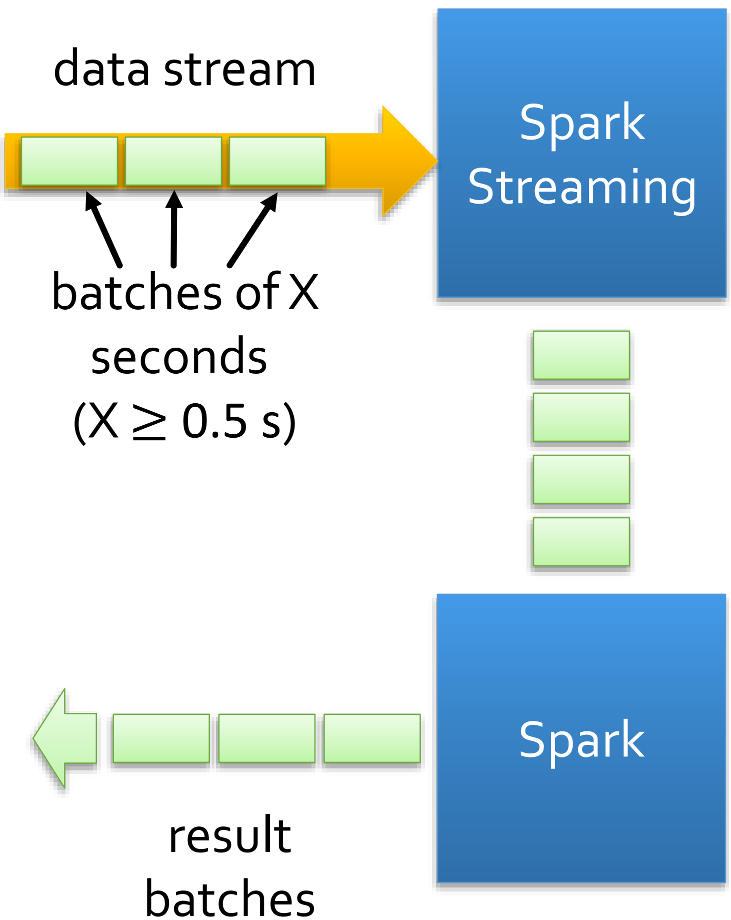
Spark Streaming

Advanced Programming

Spark Streaming: Concept

Repetition

- Process stream as a **series of small batch jobs**
  - Chop up the live stream into batches of X seconds
  - Spark treats each batch of data as an RDD and processes them using (normal) RDD operations
  - The results of the RDD operations are returned in batches



Transformations on DStreams

- Many “normal” Spark transformations are available, and some additional ones

map	flatMap	filter
repartition	union	count
reduce	count	countByValue
reduceByKey	join	cogroup
transform	**updateStateByKey**	

Operation “transform”

- The **transform** operation applies an arbitrary RDD-to-RDD function (i.e. a transformation) on each RDD in a DStream
  - Allows using any RDD operation not in the DStream API
- Example: join each RDD in a DStream with additional (precomputed) information

```
# "Normal" RDD containing spam information
spamInfoRDD = sc.pickleFile(<loadPath>)
# join data stream with spamInfoRDD
cleanDStream = wordCounts.transform( lambda
                  rdd: rdd.join(spamInfoRDD).filter(...) )
```

Operation "updateStateByKey"

- Allows to maintain arbitrary state and update it with new data from a stream
  - Input DStream contains (key, value)-pairs
- Two components:
  - **State**: any datatype (e.g. primitive, object, list,…)
  - **Update function f** of the form:
    newState = **f** (newValues, oldState)
- **f** will be called for each key k separately;
  newValues is a sequence of new values (for k)
- Usage:
  - statesDStream = inputDStream.updateStateByKey(**f**)

The updateStateByKey method returns a new DStream which contains RDDs that has the state data of each key

Example for updateStateByKey

- We want to count number of occurrences of each word since the start of the stream
- The input DStream contains pairs (<word>, 1)
- We need a separate state for each encountered word w, namely a count of w (= integer)
- The update function in Python:

Different for each <word>!

```
def updateFunction (newValues, oldCount):
        if oldCount is None:
                oldCount = 0
        return sum (newValues, oldCount)
```

Python's sum(iterable[, start])
(https://docs.python.org/2/library/functions.html#sum)

This is a sequence of 1's for a current word w since last call of the updateFunction (for w)

Excerpt from stateful_network_wordcount.py (link)

Call: stateful_network_wordcount.py localhost 9999

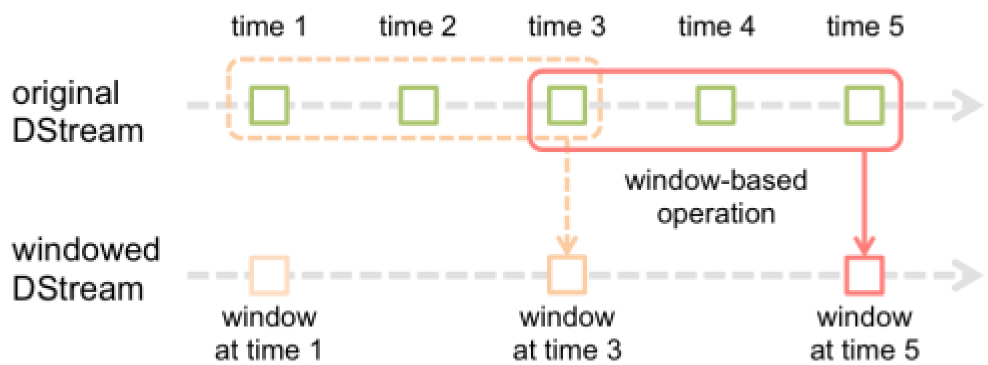
```
...
def updateFunc(new_values, last_sum):
        return sum(new_values) + (last_sum or 0)

lines = ssc.socketTextStream(sys.argv[1], int(sys.argv[2]))
running_counts = lines.flatMap(lambda line: line.split(" "))\
                        .map(lambda word: (word, 1))\
                        .updateStateByKey(updateFunc)

running_counts.pprint()

# Start the processing pipeline
ssc.start()
ssc.awaitTermination()
```

Spark Streaming

Window Operations

Window Operations

- Transformations over a **sliding window** of data, i.e. most recent stream fragment of fixed length



Spark Streaming Programming Guide v1.2,
http://spark.apache.org/docs/latest/streaming-programming-guide.html

- Here, a **windowed operation** is performed every 2 sec. (= **sliding interval**) over the last 3 sec. of data (= **window length**)
- Example: create a new DStream via

  window (windowLength, slideInterval)

Overview: Window Operations

- **window** (*windowLength*, *slideInterval*)
  - Construct a new Dstream (Example 1)
- **reduceByKeyAndWindow** (*func[, invFunc]*, *windowLength*, *slideInterval*, [*numTasks*])
  - Example 2
- **countByWindow** (*windowLength*, *slideInterval*)
  - Sliding window count of elements in the stream
- **reduceByWindow** (*func*, *windowLength*, *slideInterval*)
  - A new single-element stream from aggregating elements over a sliding interval using func

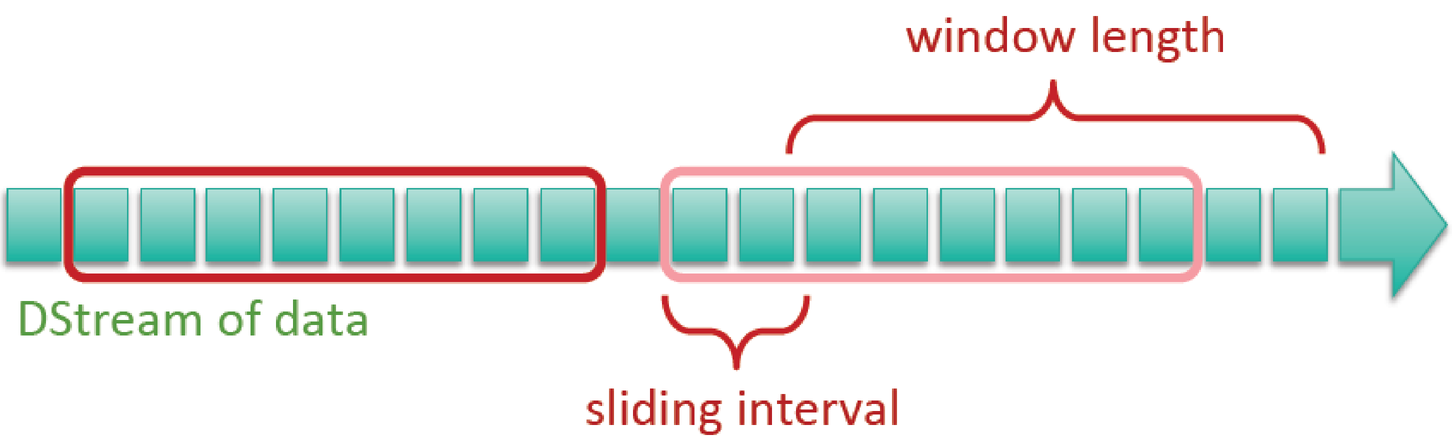
Window Op Example 1 (Scala)

```scala
val ssc = new StreamingContext (sparkContext, Seconds(1))
val hashTags = tweets.flatMap(status => getTags(status))
val tagCounts = hashTags.window(Minutes(1), Seconds(5)).countByValue()
```

Sliding window operation

Window **length**

Sliding **interval**



From: Tathagata Das, Spark Streaming, Spark Summit 2014

Word Count in a Window (Ex. 2)

- We do count of word occurrences every 10 sec. over the last 30 sec. of stream data
  - Again, input stream contains pairs (<word>, 1)

```
windowedWordCounts =
pairs.reduceByKeyAndWindow( lambda x, y: x + y,
lambda x, y: x - y, 30, 10)
```

- Here: reduceByKeyAndWindow (func, invFunc, windowLength, slideInterval, [numTasks])
- func is clear (= adding up counts), but why invFunc?
- invFunc “substracts” values which leave the window
  - => Efficient handling by reusing the result of previous window

Link Analysis

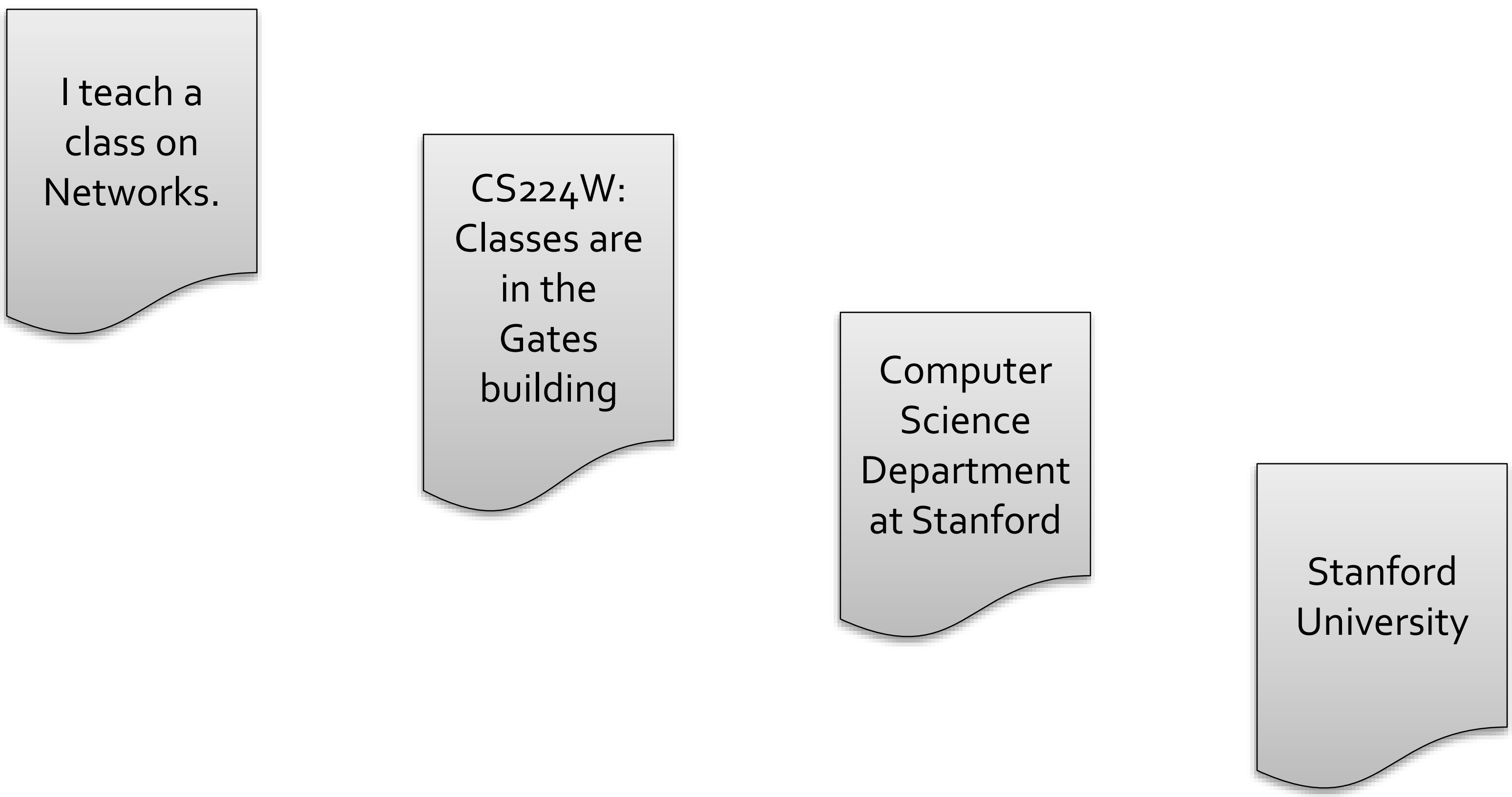
Infinite Data

High dim. data	Graph data	Infinite data	Machine learning	Apps
Locality sensitive hashing	PageRank, SimRank	Filtering data streams	SVM	Recommen der systems
Clustering	Community Detection	Web advertising	Decision Trees	Association Rules
Dimensio-nality reduction	Spam Detection	Queries on streams	Perceptron, kNN	Duplicate document detection

Programming in Spark & MapReduce

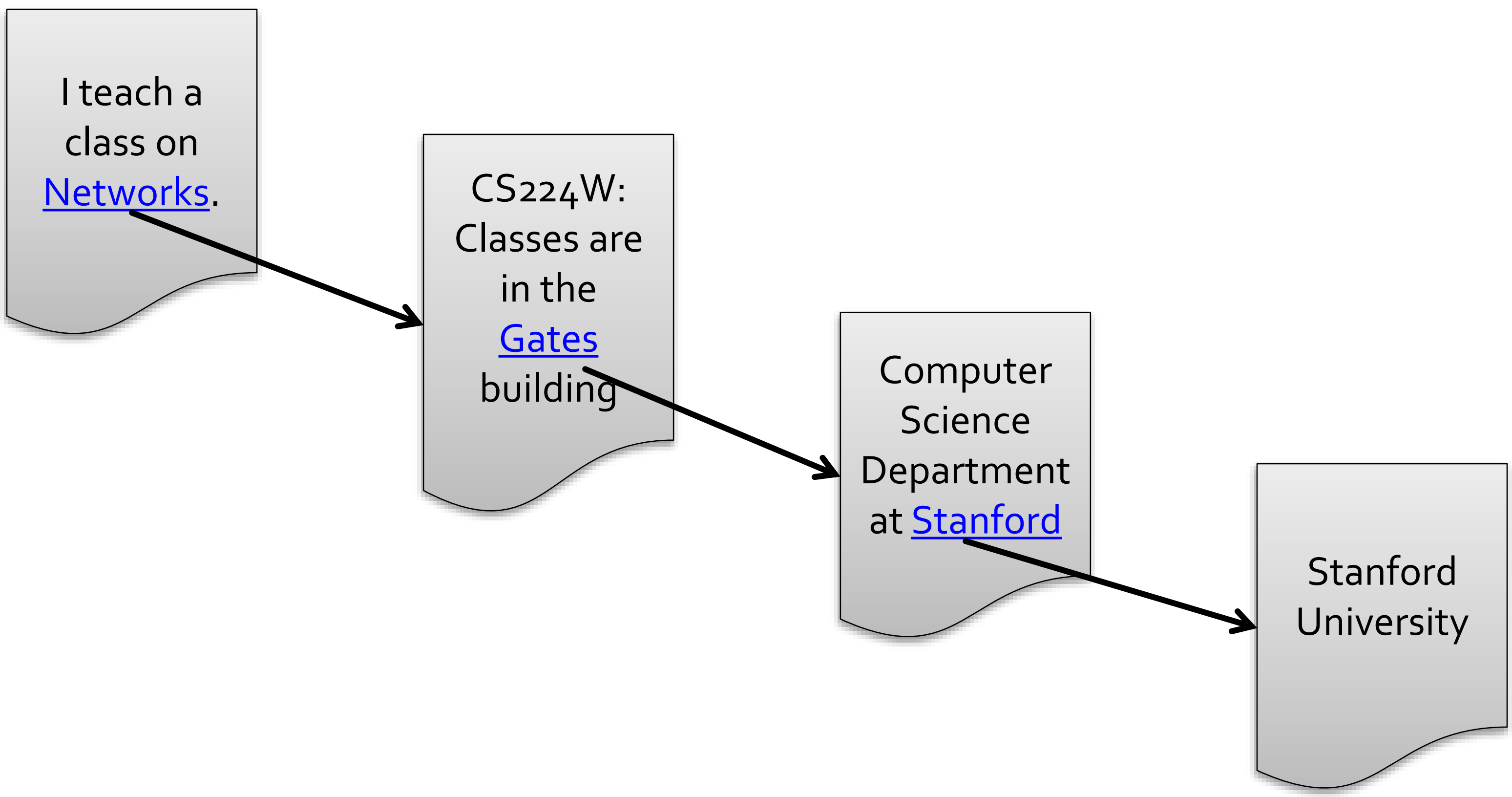
Web as a Graph

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

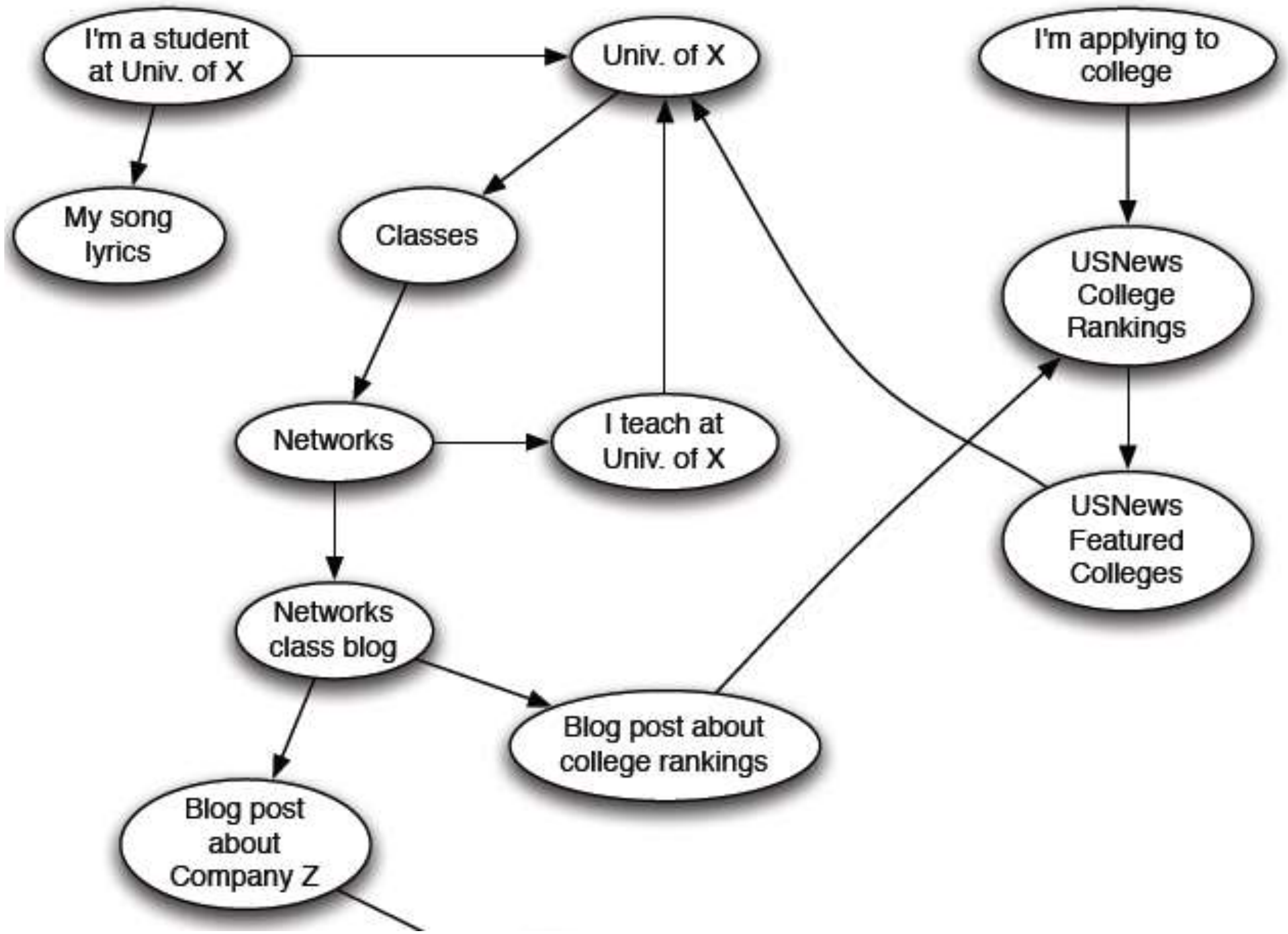


Web as a Graph

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



Web as a Directed 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

Two challenges of web search:

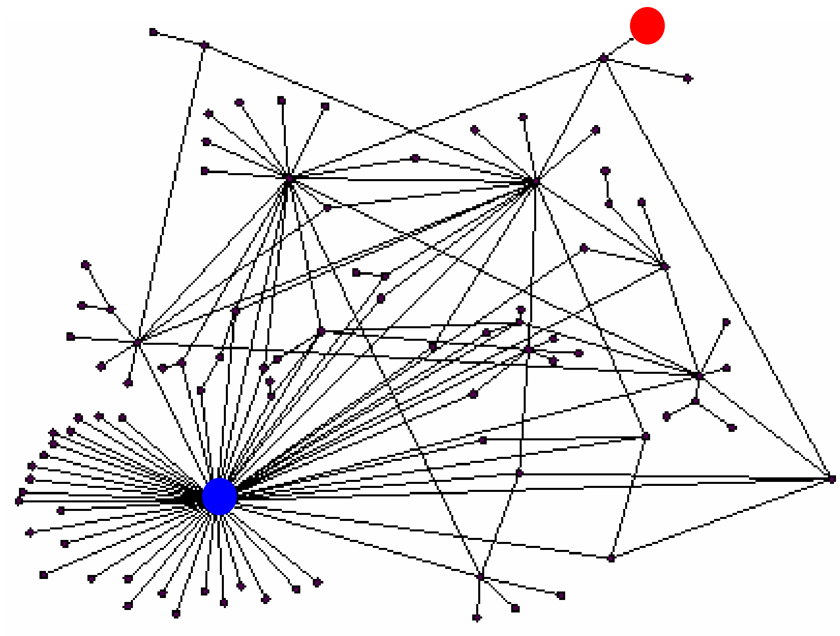
- **(1) Web contains many sources of information Who to "trust"?**
  - **Trick:** Trustworthy pages may point to each other!
- **(2) What is the "best" answer to query "newspaper"?**
  - No single right answer
  - **Trick:** Pages that actually know about newspapers might all be pointing to many newspapers

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!



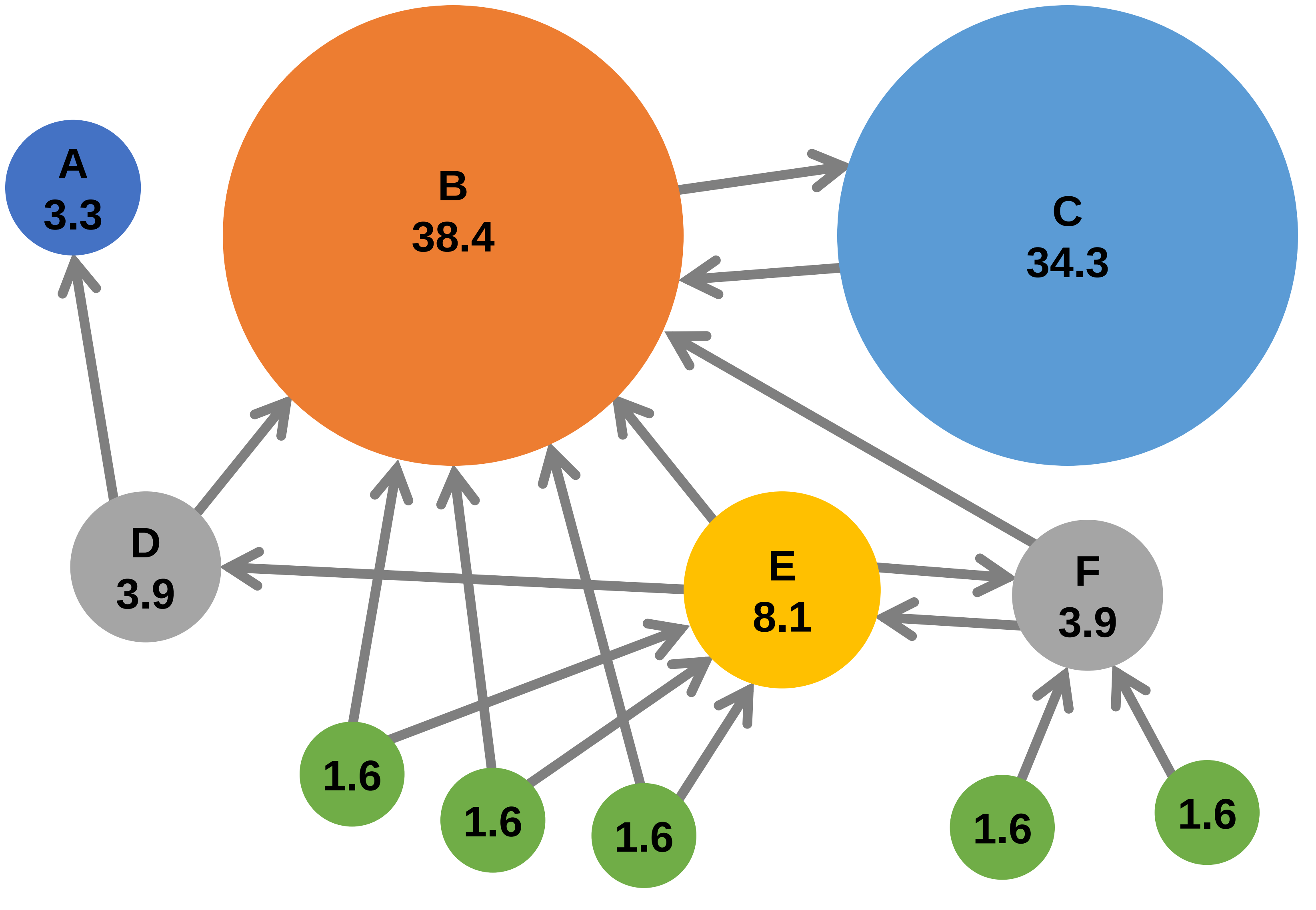
PageRank

The “Flow” Formulation

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



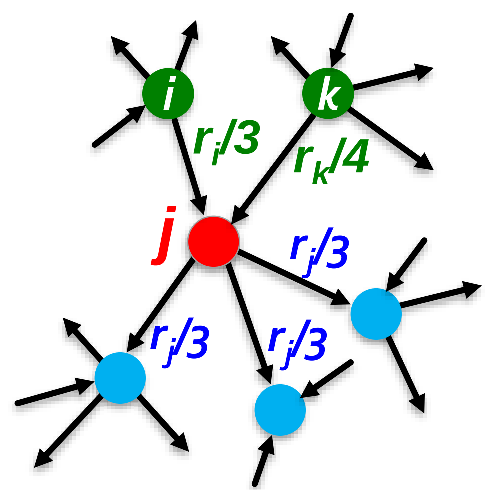
Simple Recursive Formulation

- 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

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

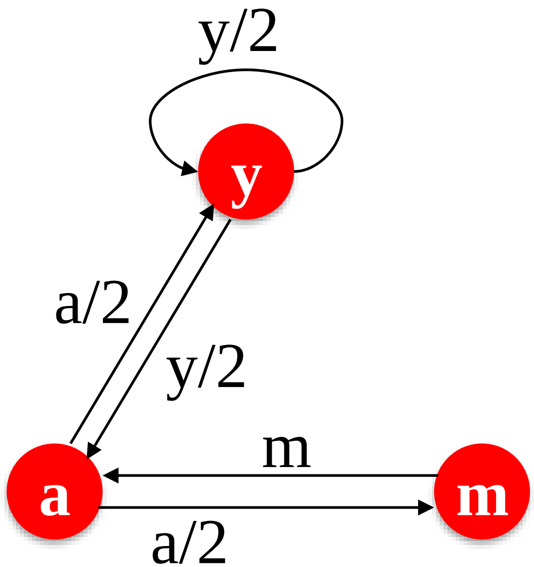


PageRank: The "Flow" Model

- 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



"Flow" equations:

$\mathbf{r_y} = \mathbf{r_y}/2 + \mathbf{r_a}/2$

$\mathbf{r_a} = \mathbf{r_y}/2 + \mathbf{r_m}$

$\mathbf{r_m} = \mathbf{r_a}/2$

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:**
  - $\boldsymbol{r_y + r_a + r_m = 1}$
  - **Solution:** $\boldsymbol{r_y = \frac{2}{5},\ r_a = \frac{2}{5},\ r_m = \frac{1}{5}}$
- **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 $\boldsymbol{M}$**
  - Let page i has d_i out-links
  - If $i \to j$, then $M_{ji} = \frac{1}{d_i}$ else $M_{ji} = 0$
    - $\boldsymbol{M}$ is a **column stochastic matrix** (i.e. columns sum to 1)
- **Rank vector $\boldsymbol{r}$:** vector with an entry per page
  - r_i is the importance score of page i
  - $\sum_i r_i = 1$

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

- **The flow equations can be written as**

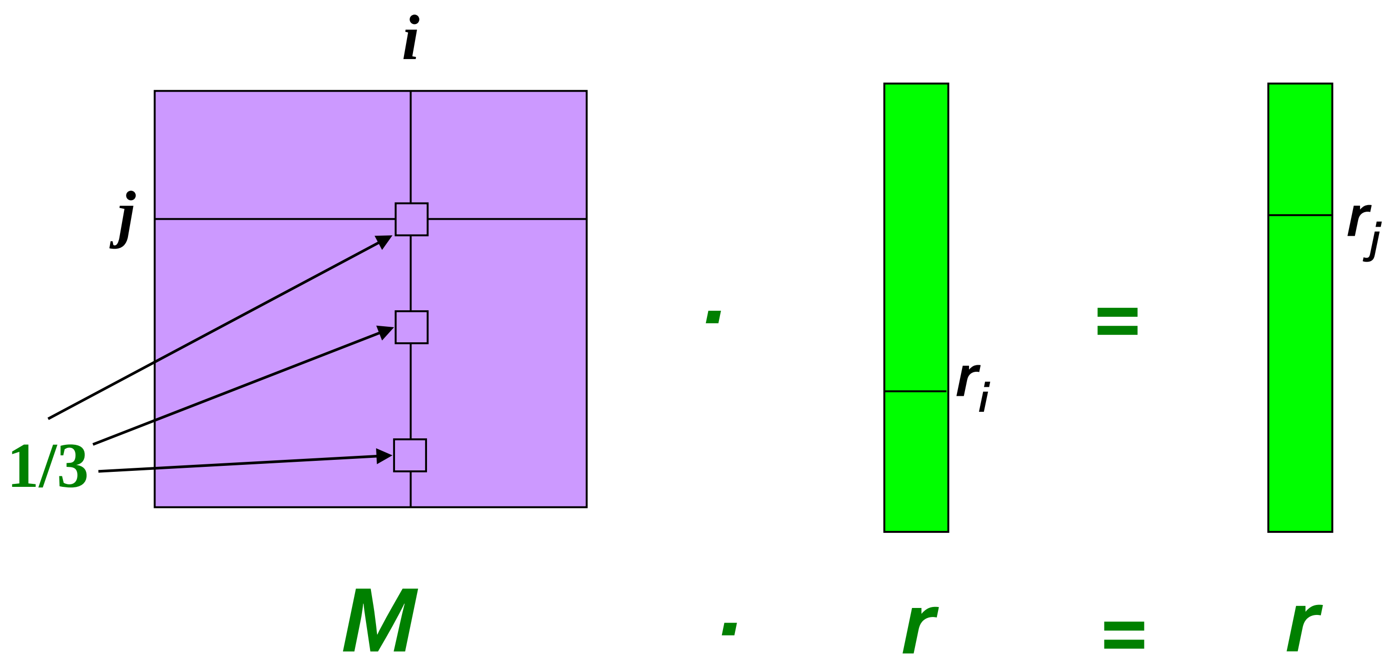
$$\boldsymbol{r} = \boldsymbol{M} \cdot \boldsymbol{r}$$

Example

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

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

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



Eigenvector Formulation

- The flow equations can be written

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

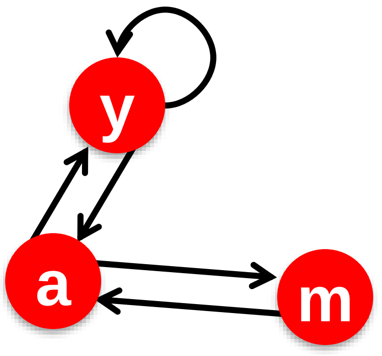
- So the **rank vector** ***r*** is an **eigenvector** of the stochastic web matrix ***M***
  - In fact, it is its first or **principal** eigenvector, with corresp. eigenvalue ***1***
    - Largest eigenvalue of ***M*** is **1** since ***M*** is column stochastic (with non-negative entries)
      - *We know* ***r*** *is unit length and each column of* ***M*** *sums to one, so* $\boldsymbol{Mr} \leq \mathbf{1}$

NOTE: ***x*** is an **eigenvector** with the corres-ponding **eigenvalue** **λ** if:

$$\boldsymbol{Ax} = \boldsymbol{\lambda x}$$

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

Example: Flow Equations & M



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

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

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

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

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 $|\mathbf{r}^{(t+1)} - \mathbf{r}^{(t)}|_1 < \varepsilon$

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

d_i …. out-degree of node i

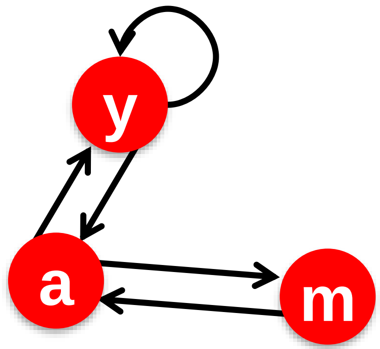
$|\mathbf{x}|_1 = \sum_{1 \le i \le N} |x_i|$ is the $\mathbf{L_1}$ norm
Can use any other vector norm, e.g., Euclidean

PageRank: How to solve?

- **Power Iteration:**
  - Set $r_j = 1/N$
  - **1:** $r'_j = \sum_{i \to j} \frac{r_i}{d_i}$
  - **2:** $r = r'$
  - 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, …



	y	a	m
y	½	½	0
a	½	0	1
m	0	½	0

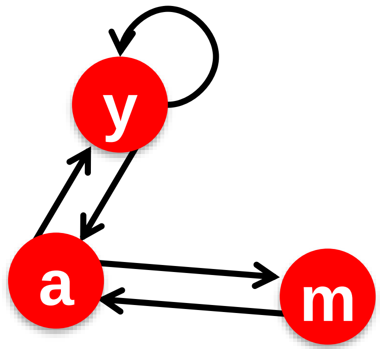
$\mathbf{r_y = r_y/2 + r_a/2}$

$\mathbf{r_a = r_y/2 + r_m}$

$\mathbf{r_m = r_a/2}$

PageRank: How to solve?

- **Power Iteration:**
  - Set $r_j = 1/N$
  - **1:** $r'_j = \sum_{i \to j} \frac{r_i}{d_i}$
  - **2:** $r = r'$
  - Goto **1**
- **Example:**



	y	a	m
y	½	½	0
a	½	0	1
m	0	½	0

$r_y = r_y/2 + r_a/2$

$r_a = r_y/2 + r_m$

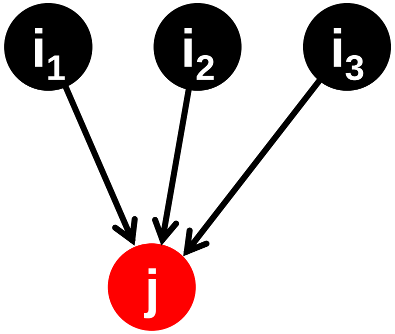
$r_m = r_a/2$

$$\begin{pmatrix} r_y \\ r_a \\ r_m \end{pmatrix} = \begin{matrix} 1/3 & 1/3 & 5/12 & 9/24 & & 6/15 \\ 1/3 & 3/6 & 1/3 & 11/24 & \ldots & 6/15 \\ 1/3 & 1/6 & 3/12 & 1/6 & & 3/15 \end{matrix}$$

Iteration 0, 1, 2, …

Random Walk Interpretation

- **Imagine a random web surfer:**
  - At any time $\boldsymbol{t}$, surfer is on some page $\boldsymbol{i}$
  - At time $\boldsymbol{t+1}$, the surfer follows an out-link from $\boldsymbol{i}$ uniformly at random
  - Ends up on some page $\boldsymbol{j}$ linked from $\boldsymbol{i}$
  - Process repeats indefinitely
- **Let:**
  - $\boldsymbol{p}(\boldsymbol{t})$ … vector whose $\boldsymbol{i}^{\text{th}}$ coordinate is the prob. that the surfer is at page $\boldsymbol{i}$ at time $\boldsymbol{t}$
  - So, $\boldsymbol{p}(\boldsymbol{t})$ is a probability distribution over pages

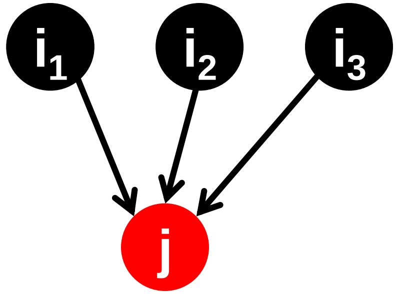


$$r_j = \sum_{i \to j} \frac{r_i}{\mathrm{d}_{\mathrm{out}}(\mathrm{i})}$$

The Stationary Distribution

- **Where is the surfer at time *t+1*?**
  - Follows a link uniformly at random

    $$\boldsymbol{p}(\boldsymbol{t}+\mathbf{1}) = \boldsymbol{M} \cdot \boldsymbol{p}(\boldsymbol{t})$$



$$p(t+1) = \mathrm{M} \cdot p(t)$$

- Suppose the random walk reaches a state

  $$\boldsymbol{p}(\boldsymbol{t}+\mathbf{1}) = \boldsymbol{M} \cdot \boldsymbol{p}(\boldsymbol{t}) = \boldsymbol{p}(\boldsymbol{t})$$

  then $\boldsymbol{p}(\boldsymbol{t})$ is **stationary distribution** of a random walk
- **Our original rank vector** $\boldsymbol{r}$ satisfies $\boldsymbol{r} = \boldsymbol{M} \cdot \boldsymbol{r}$
  - **So, r is a stationary distribution for the random walk**

Existence and Uniqueness

- A central result from the theory of random walks (a.k.a. Markov processes):

> For graphs that satisfy **certain conditions**, the **stationary distribution is unique** and eventually will be reached no matter what the initial probability distribution at time **t = 0**

Thank you.

Questions?

Additional Slides

Why Power Iteration works? (1)

Details!

- **Power iteration:**
  A method for finding dominant eigenvector (the vector corresponding to the largest eigenvalue)
  - $\boldsymbol{r}^{(1)} = \boldsymbol{M} \cdot \boldsymbol{r}^{(0)}$
  - $\boldsymbol{r}^{(2)} = \boldsymbol{M} \cdot \boldsymbol{r}^{(1)} = \boldsymbol{M}\left(\boldsymbol{M}\boldsymbol{r}^{(1)}\right) = \boldsymbol{M}^{2} \cdot \boldsymbol{r}^{(0)}$
  - $\boldsymbol{r}^{(3)} = \boldsymbol{M} \cdot \boldsymbol{r}^{(2)} = \boldsymbol{M}\left(\boldsymbol{M}^{2}\boldsymbol{r}^{(0)}\right) = \boldsymbol{M}^{3} \cdot \boldsymbol{r}^{(0)}$
- **Claim:**
  Sequence $\boldsymbol{M} \cdot \boldsymbol{r}^{(0)}, \boldsymbol{M}^{2} \cdot \boldsymbol{r}^{(0)}, \ldots \boldsymbol{M}^{k} \cdot \boldsymbol{r}^{(0)}, \ldots$ approaches the dominant eigenvector of $\boldsymbol{M}$

Why Power Iteration works? (2)

Details!

- **Claim:** Sequence $\boldsymbol{M} \cdot \boldsymbol{r}^{(0)}, \boldsymbol{M}^{2} \cdot \boldsymbol{r}^{(0)}, \ldots \boldsymbol{M}^{k} \cdot \boldsymbol{r}^{(0)}, \ldots$ approaches the dominant eigenvector of $\boldsymbol{M}$
- **Proof:**
  - Assume ***M*** has ***n*** linearly independent eigenvectors, $x_1, x_2, \ldots, x_n$ with corresponding eigenvalues $\lambda_1, \lambda_2, \ldots, \lambda_n$, where $\lambda_1 > \lambda_2 > \cdots > \lambda_n$
  - Vectors $x_1, x_2, \ldots, x_n$ form a basis and thus we can write: $r^{(0)} = c_1\, x_1 + c_2\, x_2 + \cdots + c_n\, x_n$
  - $$\begin{aligned}\boldsymbol{M}\boldsymbol{r}^{(0)} &= \boldsymbol{M}(\boldsymbol{c_1}\, \boldsymbol{x_1} + \boldsymbol{c_2}\, \boldsymbol{x_2} + \cdots + \boldsymbol{c_n}\, \boldsymbol{x_n}) \\ &= c_1(Mx_1) + c_2(Mx_2) + \cdots + c_n(Mx_n) \\ &= c_1(\lambda_1 x_1) + c_2(\lambda_2 x_2) + \cdots + c_n(\lambda_n x_n)\end{aligned}$$
  - **Repeated multiplication on both sides produces** $$M^k r^{(0)} = c_1(\lambda_1^k x_1) + c_2(\lambda_2^k x_2) + \cdots + c_n(\lambda_n^k x_n)$$

Why Power Iteration works? (3)

Details!

- **Claim:** Sequence $\boldsymbol{M} \cdot \boldsymbol{r}^{(0)}, \boldsymbol{M}^{2} \cdot \boldsymbol{r}^{(0)}, \ldots \boldsymbol{M}^{k} \cdot \boldsymbol{r}^{(0)}, \ldots$ approaches the dominant eigenvector of $\boldsymbol{M}$
- **Proof (continued):**
  - Repeated multiplication on both sides produces $M^k r^{(0)} = c_1(\lambda_1^k x_1) + c_2(\lambda_2^k x_2) + \cdots + c_n(\lambda_n^k x_n)$
  - $M^k r^{(0)} = \lambda_1^k \left[c_1 x_1 + c_2 \left(\frac{\lambda_2}{\lambda_1}\right)^k x_2 + \cdots + c_n \left(\frac{\lambda_2}{\lambda_1}\right)^k x_n \right]$
  - Since $\lambda_1 > \lambda_2$ then fractions $\frac{\lambda_2}{\lambda_1}, \frac{\lambda_3}{\lambda_1} \ldots < 1$ and so $\left(\frac{\lambda_i}{\lambda_1}\right)^k = 0$ as $k \to \infty$ (for all $i = 2 \ldots n$).
  - **Thus:** $\boldsymbol{M}^k \boldsymbol{r}^{(0)} \approx \boldsymbol{c}_1(\boldsymbol{\lambda}_1^k \boldsymbol{x}_1)$
    - Note if $c_1 = 0$ then the method won't converge