Mining Massive Datasets

Lecture 06

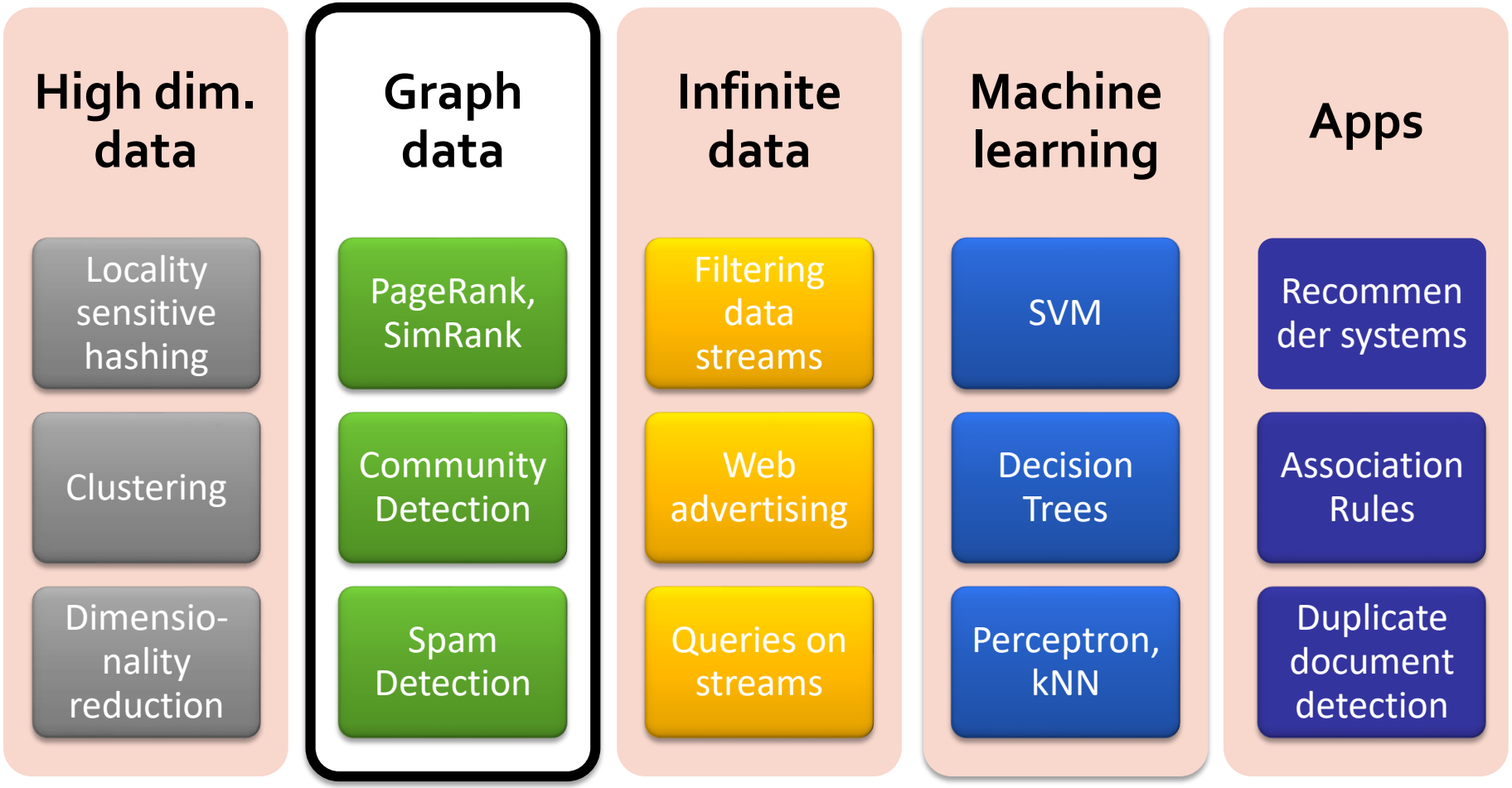
Artur Andrzejak
http://pvs.ifi.uni-heidelberg.de

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

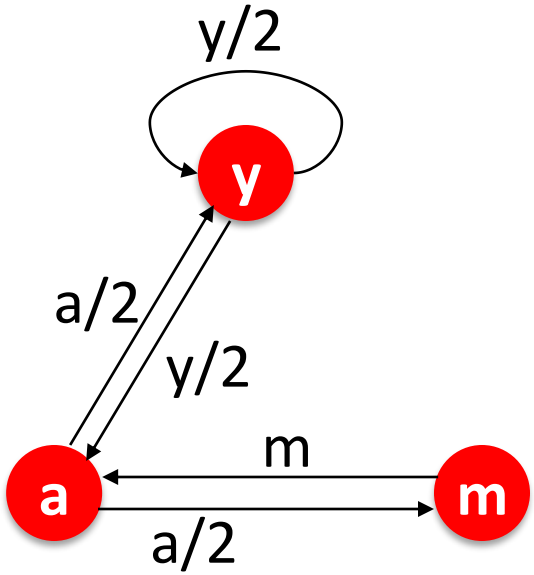


PageRank:

Recalling the most important facts from Lecture 5

Recall: 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_{rec} for page *rec*



$$r_{rec} = \sum_{sender \to rec} \frac{r_{sender}}{\mathrm{d}_{sender}}$$

d_{sender} … out-degree of node *sender*

Flow equations:

$r_y = r_y/2 + r_a/2$

$r_a = r_y/2 + r_m$

$r_m = r_a/2$

Recall: PageRank - Matrix Formulation

- Adjacency matrix M
  - Encodes the structure of the web graphl
  - Let page i has d_i out-links
  - If $i \rightarrow j$, then $M_{ji} = \frac{1}{d_i}$ else $M_{ji} = 0$
- Rank vector r: vector with an entry per page
  - r_i is the importance score of page i
  - $\sum_i r_i = 1$

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

- The flow equations can be written as

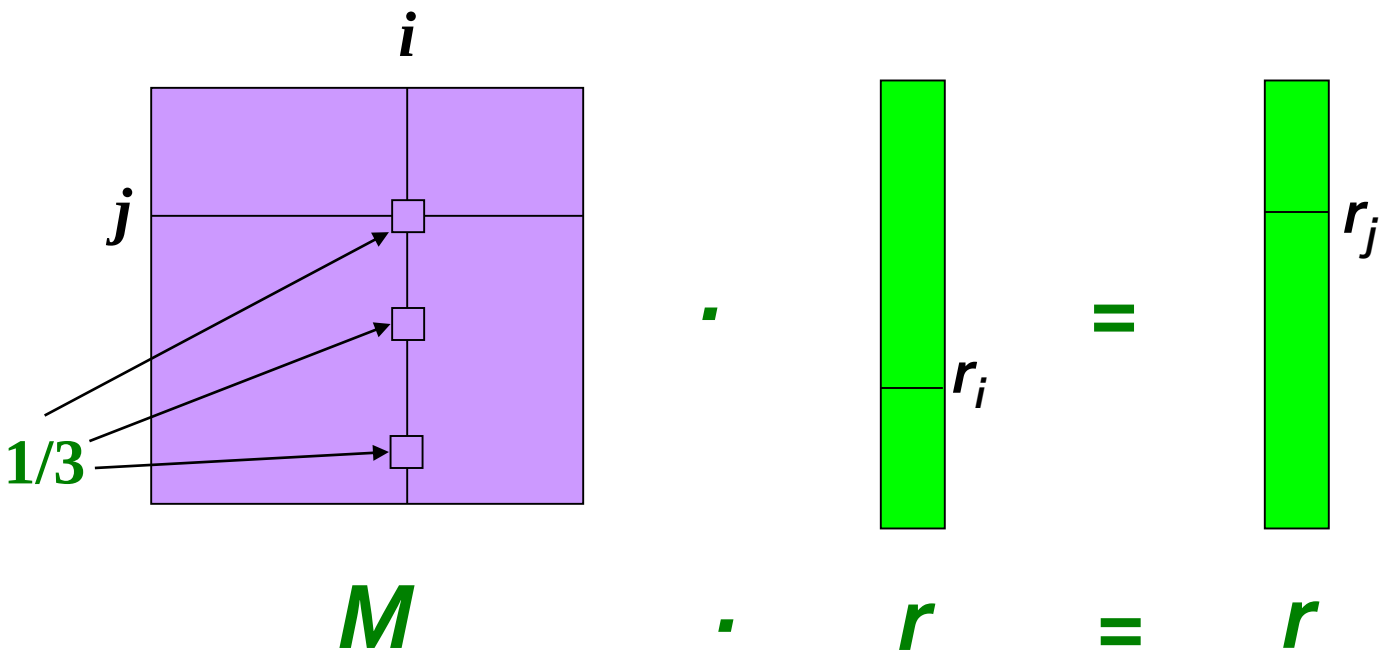
$$\boldsymbol{r} = \boldsymbol{M} \cdot \boldsymbol{r}$$

Recall: Example

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

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

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



Recall: The Google Matrix

- PageRank equation [Brin-Page, '98]

$$r_j = \sum_{i \to j} \beta \frac{r_i}{d_i} + (1 - \beta) \frac{1}{N}$$

- The Google Matrix *A*:

$$A = \beta\, M + (1 - \beta) \left[\frac{1}{N}\right]_{N \times N}$$

$[1/N]_{NxN}$…N by N matrix where all entries are 1/N

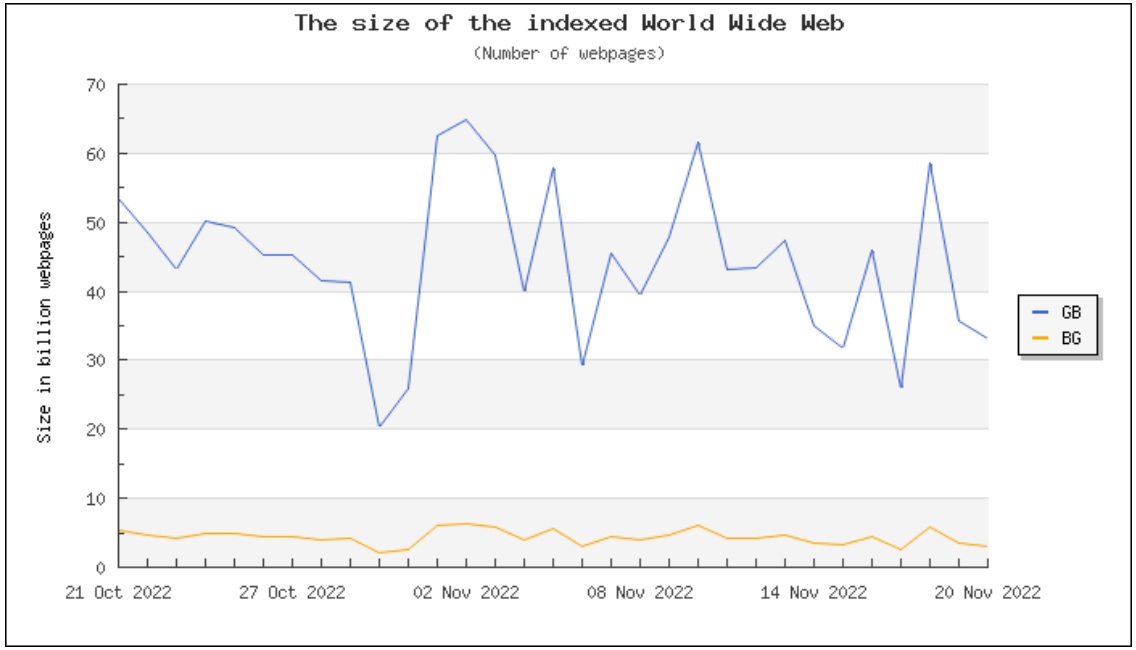
- We have now a recursive problem: $r = A \cdot r$
- Solve using the **power iteration** method:
  - (1) Init $\mathbf{r}^{(0)}$; (2) Iterate: $\mathbf{r}^{(t+1)} = \mathbf{M} \cdot \mathbf{r}^{(t)}$
  - (3) Stop when $|\mathbf{r}^{(t+1)} - \mathbf{r}^{(t)}|_1 < \varepsilon$

PageRank:

How do we actually compute the PageRank?

How many pages are on the Internet?

- Indexed Web contains *at least* **2.1 billion** pages (Monday, 21 November, 2022)
  - Data by Tilburg University (updated daily)
- The actual size seems to be >**50 billion** pages



GB = Sorted on Google and Bing
BG = Sorted on Bing and Google
Source: https://www.worldwidewebsize.com/

Computing Page Rank

- Key step is **matrix-vector multiplication**
  - $r^{new} = A \cdot r^{old}$
- Easy if we have enough main memory to hold **A**, $\mathbf{r}^{old}$, $\mathbf{r}^{new}$
- **Say N = 1 billion pages**
  - We need 4 bytes for each entry (say)
  - 2 billion entries for vectors, approx 8GB
  - Matrix **A** has N^2 entries
    - 10^{18} is a large number!
  - Insight: **M** is sparse, **A** is not!
  - Goal: Find a recursive update step which uses only sparse matrices!

$$\mathbf{A} = \beta \cdot \mathbf{M} + (1-\beta)\,[1/N]_{NxN}$$

$$A = 0.8 \begin{bmatrix} ½ & ½ & 0 \\ ½ & 0 & 0 \\ 0 & ½ & 1 \end{bmatrix} + 0.2 \begin{bmatrix} 1/3 & 1/3 & 1/3 \\ 1/3 & 1/3 & 1/3 \\ 1/3 & 1/3 & 1/3 \end{bmatrix}$$

$$= \begin{bmatrix} 7/15 & 7/15 & 1/15 \\ 7/15 & 1/15 & 1/15 \\ 1/15 & 7/15 & 13/15 \end{bmatrix}$$

Matrix Formulation

- Suppose there are ***N*** pages
- Consider page ***i***, with $\mathbf{d}_i$ out-links
- We have $M_{ji} = 1/|d_i|$ when $i \rightarrow j$
  and $M_{ji} = 0$ otherwise
- The random teleport is equivalent to:
  - Adding a **teleport link** from ***i*** to every other page and setting transition probability to $(1-\beta)/N$
  - Reducing the probability of following each out-link from $1/|d_i|$ to $\beta/|d_i|$
  - **Equivalent:** ***Tax*** each page a fraction $(1-\beta)$ of its score and redistribute evenly

Rearranging the Equation

- $\boldsymbol{r} = \boldsymbol{A} \cdot \boldsymbol{r}$, **where** $\boldsymbol{A_{ji}} = \boldsymbol{\beta}\, \boldsymbol{M_{ji}} + \frac{\boldsymbol{1-\beta}}{\boldsymbol{N}}$
- $r_j = \sum_{i=1}^{N} A_{ji} \cdot r_i$
- $r_j = \sum_{i=1}^{N} \left[\beta\, M_{ji} + \frac{1-\beta}{N}\right] \cdot r_i$ # use def of A_{ji}

  $= \sum_{i=1}^{N} \beta\, M_{ji} \cdot r_i + \frac{1-\beta}{N} \sum_{i=1}^{N} r_i$

  $= \sum_{i=1}^{N} \beta\, M_{ji} \cdot r_i + \frac{1-\beta}{N}$ # use $\sum r_i = 1$
- **So we get:** $\boldsymbol{r} = \boldsymbol{\beta}\, \boldsymbol{M} \cdot \boldsymbol{r} + \left[\frac{\boldsymbol{1-\beta}}{\boldsymbol{N}}\right]_{\boldsymbol{N}}$

Note: Here we assumed **M** has no dead-ends for $\sum r_i = 1$

$[x]_N$ … a vector of length N with all entries x

Sparse Matrix Formulation

- We just rearranged the **PageRank equation**

$$\boldsymbol{r} = \boldsymbol{\beta M} \cdot \boldsymbol{r} + \left[\frac{\boldsymbol{1-\beta}}{\boldsymbol{N}}\right]_{\boldsymbol{N}}$$

  - where $\mathbf{[(1-\beta)/N]_N}$ is a vector with all $\boldsymbol{N}$ entries **(1-β)/N** (= const)

- $\boldsymbol{M}$ is a **sparse matrix!** (with no dead-ends)
  - 10 links per node, approx 10N entries
- So in each iteration, we need to:
  - Compute $\boldsymbol{r}^{new} = \beta \boldsymbol{M} \cdot \boldsymbol{r}^{old}$
  - Add a constant value **(1-β)/N** to each entry in $\boldsymbol{r}^{new}$
    - Note: if M contains dead-ends then $\sum_j r_j^{new} < 1$ and we also have to **renormalize** r^{new} so that it sums to 1

PageRank: The Complete Algorithm

- **Input: Graph G and parameter β**
  - Directed graph G (can have **spider traps** and **dead ends**)
  - Parameter β
- **Output: PageRank vector r^{new}**
  - **Set:** $r_j^{old} = \frac{1}{N}$
  - **repeat until convergence:** $\sum_j \left| r_j^{new} - r_j^{old} \right| < \varepsilon$
    - A. $\forall j$: $r'^{new}_j = \sum_{i \to j} \beta \frac{r_i^{old}}{d_i}$ (this is $\beta M \cdot r$)
      $r'^{new}_j = 0$ if in-degree of j is 0
    - B. Re-insert the "leaked" (due to β and dead ends) PageRank:
      $\forall j$: $r_j^{new} = r'^{new}_j + \frac{1-\beta S}{N}$ where: $S = \sum_j r'^{new}_j$ (after step A)
    - $r^{old} = r^{new}$

If the graph has no dead-ends then the amount of "leaked" PageRank is **1-β**. But since we have dead-ends the amount of leaked PageRank may be larger. We have to explicitly account for it by computing **S**.

PageRank:

Computing the PageRank with Memory Constraints

Sparse Matrix Encoding

- Encode sparse matrix using only nonzero entries
  - Space proportional roughly to number of links
  - Say 10N => (4 bytes)*10*10 billion = 400GB
  - Still won't fit in memory, but will fit on disk
  - **In 2022: 1 TB RAM server ~20k EUR => all in RAM**

Data structure for matrix M

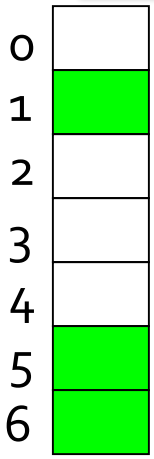
source node	degree	destination nodes
0	3	1, 5, 7
1	5	17, 64, 113, 117, 245
2	2	13, 23

Basic Algorithm: Update Step

- **Assume enough RAM to fit r^{new} into memory**
  - Store r^{old} and matrix **M** on disk
- **Now one iteration (step A) of power iteration is:**

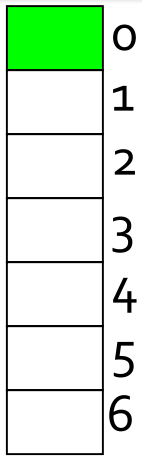
Initialize all entries of $\mathbf{r^{new}} = \mathbf{(1-\beta) / N}$
For each page ***i*** (of out-degree d_i):
  Read into memory: i, d_i, $dest_1, \ldots, dest_{d_i}$, $r^{old}(i)$
  For $\mathbf{j = 1 \ldots d_i}$ (this is $\beta M \cdot r$)
    $\mathbf{r^{new}(dest_j) \mathrel{+}= \beta\, r^{old}(i) / d_i}$

Assume that $S = \sum_j r_j^{new} = 1$ to simplify, i.e. no step B (= no dead-ends)



r^{new}

source	degree	destination
0	3	1, 5, 6
1	4	17, 64, 113, 117
2	2	13, 23

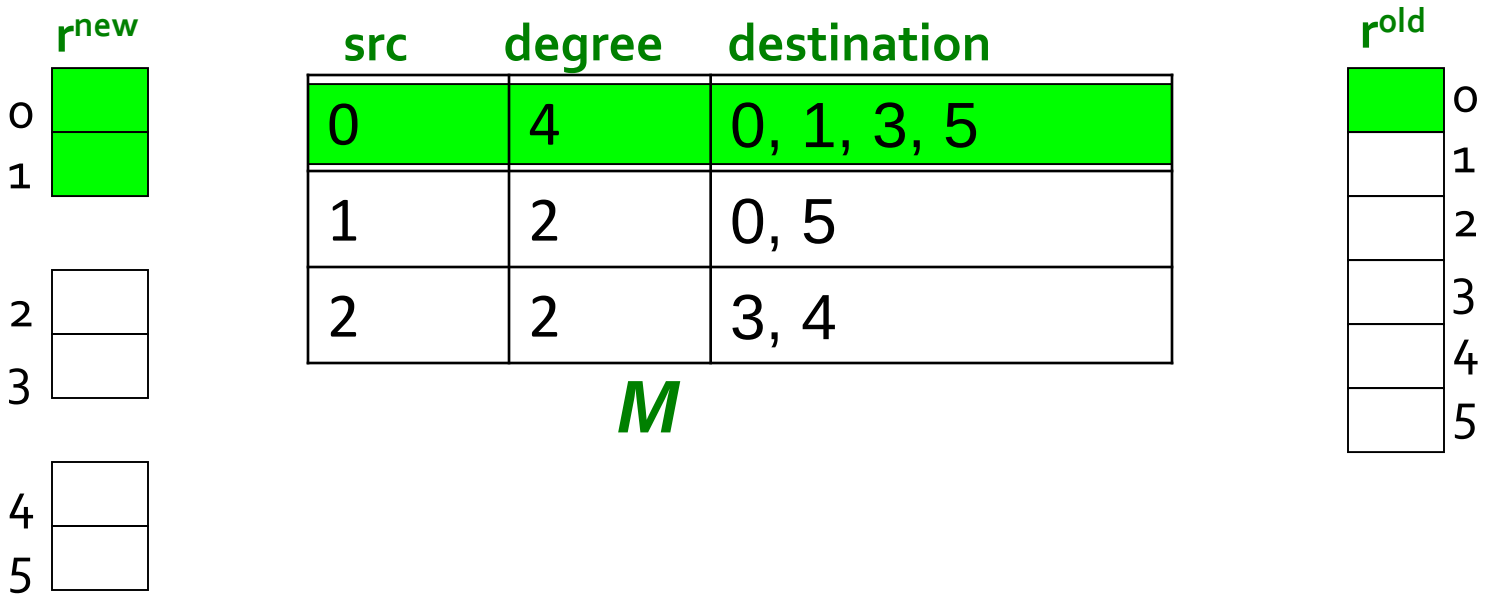


r^{old}

Analysis

- **Assume enough RAM to fit r^{new} into memory**
  - Store $\boldsymbol{r}^{old}$ and matrix $\boldsymbol{M}$ on disk
- **In each iteration, we have to:**
  - Read $\boldsymbol{r}^{old}$ and $\boldsymbol{M}$
  - Write $\boldsymbol{r}^{new}$ back to disk
  - **Cost per iteration of Power method:**
    $= 2|\boldsymbol{r}| + |\boldsymbol{M}|$
- **Question:**
  - What if we could not even fit $\boldsymbol{r}^{new}$ in memory?

Block-based Update Algorithm

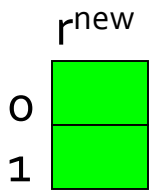


- Break r^{new} into k blocks that fit in memory
- Scan M and r^{old} once for each block

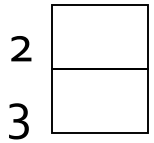
Analysis of Block Update

- Similar to nested-loop join in databases
  - Break $\boldsymbol{r}^{new}$ into $\boldsymbol{k}$ blocks that fit in memory
  - Scan $\boldsymbol{M}$ and $\boldsymbol{r}^{old}$ once for each block
- Total cost:
  - $\boldsymbol{k}$ scans of $\boldsymbol{M}$ and $\boldsymbol{r}^{old}$
  - **Cost per iteration of Power method:**
    $k(|\boldsymbol{M}| + |\boldsymbol{r}|) + |\boldsymbol{r}| = \boldsymbol{k|M| + (k+1)|r|}$
- Can we do better?
  - **Hint:** $\boldsymbol{M}$ is much bigger than $\boldsymbol{r}$ (approx 10-20x), so we must avoid reading it $\boldsymbol{k}$ times per iteration

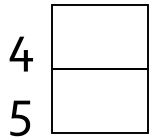
Block-Stripe Update Algorithm



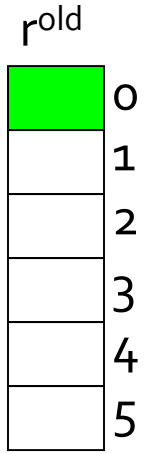
src	degree	destination
0	4	0, 1
1	3	0
2	2	1



0	4	3
2	2	3



0	4	5
1	3	5
2	2	4



Break *M* into stripes! Each stripe contains only destination nodes in the corresponding block of $\boldsymbol{r}^{new}$

Block-Stripe Analysis

- Break ***M*** into stripes
  - Each stripe contains only destination nodes in the corresponding block of $\boldsymbol{r}^{new}$
- Some additional overhead per stripe
  - But it is usually worth it
- => Cost per iteration of Power method:
  $= |M|(1+\varepsilon) + (k+1)|r|$

Some Problems with Page Rank

- Measures generic popularity of a page
  - Biased against topic-specific authorities
  - **Solution:** Topic-Specific PageRank
- Uses a single measure of importance
  - Other models of importance
  - **Solution:** Hubs-and-Authorities
- Susceptible to Link spam
  - Artificial link topographies created in order to boost page rank
  - **Solution:** TrustRank

Topic-Specific PageRank

(Short)

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

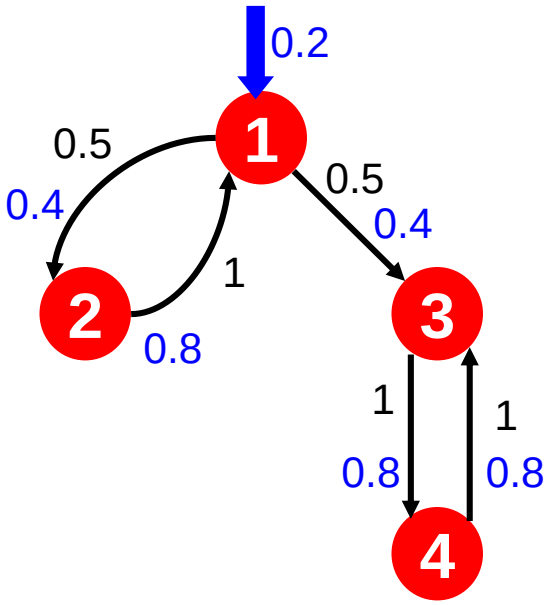
Matrix Formulation

- To make this work all we need is to update the teleportation part of the PageRank formulation:

$$A_{ij} = \begin{cases} \beta M_{ij} + (1-\beta)/|S| & \text{if } i \in S \\ \beta M_{ij} + 0 & \text{otherwise} \end{cases}$$

  - ***A*** is stochastic!
  - We weighted all pages in the teleport set ***S*** equally, but this can be changed
- **Compute as for regular PageRank:**
  - Multiply by ***M***, then add a vector
  - Maintains sparseness

Example: Topic-Specific PageRank



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

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

S={1}, β=0.90:
r=[0.17, 0.07, 0.40, 0.36]
S={1} , β=0.8:
r=[0.29, 0.11, 0.32, 0.26]
S={1}, β=0.70:
r=[0.39, 0.14, 0.27, 0.19]

S={1,2,3,4}, β=0.8:
r=[0.13, 0.10, 0.39, 0.36]
S={1,2,3} , β=0.8:
r=[0.17, 0.13, 0.38, 0.30]
S={1,2} , β=0.8:
r=[0.26, 0.20, 0.29, 0.23]
S={1} , β=0.8:
r=[0.29, 0.11, 0.32, 0.26]

Discovering the Topic Vector S

- **Create different PageRanks for different topics**
  - The 16 DMOZ top-level categories:
    - arts, business, sports,…
- **Which topic ranking to use?**
  - User can pick from a menu
  - Classify query into a topic
  - Can use the **context** of the query
    - E.g., query is launched from a web page talking about a known topic
    - History of queries e.g., “basketball” followed by “Jordan”
  - User context, e.g., user’s bookmarks, …

Link: https://pingo.coactum.de/147633

Pingo Quiz

- Mark true statements (multiple are possible)

✘ 1. If we cannot fit the “new” rank vector into RAM, we get only approximate result for $\boldsymbol{r}^{\text{new}} = \boldsymbol{A} \cdot \boldsymbol{r}^{\text{old}}$

✓ 2. In the „Block-based Update Algorithm“ we don‘t need to preprocess „link matrix M“ before start

✘ 3. The “Block-Stripe Update Algorithm” stores the targets of some links in a redundant way

✓ 4. In the “Topic-Specific PageRank”, pages not in the teleport set S have smaller prob. as teleport goals

TrustRank

Spam Farming

What is Web Spam?

- **Spamming:**
  - Any deliberate action to boost a web page's position in search engine results, incommensurate with page's real value
- **Spam:**
  - Web pages that are the result of spamming
- This is a very broad definition
  - **SEO** industry might disagree!
  - SEO = search engine optimization
- Approximately **10-15%** of web pages are spam

Google vs. Spammers

- Once Google became the dominant search engine, spammers began to work out ways to fool Google
- **Spam farms** were developed to concentrate PageRank on a single page
- **Link spam:**
  - Creating link structures that boost PageRank of a particular page


SPAM FARMING

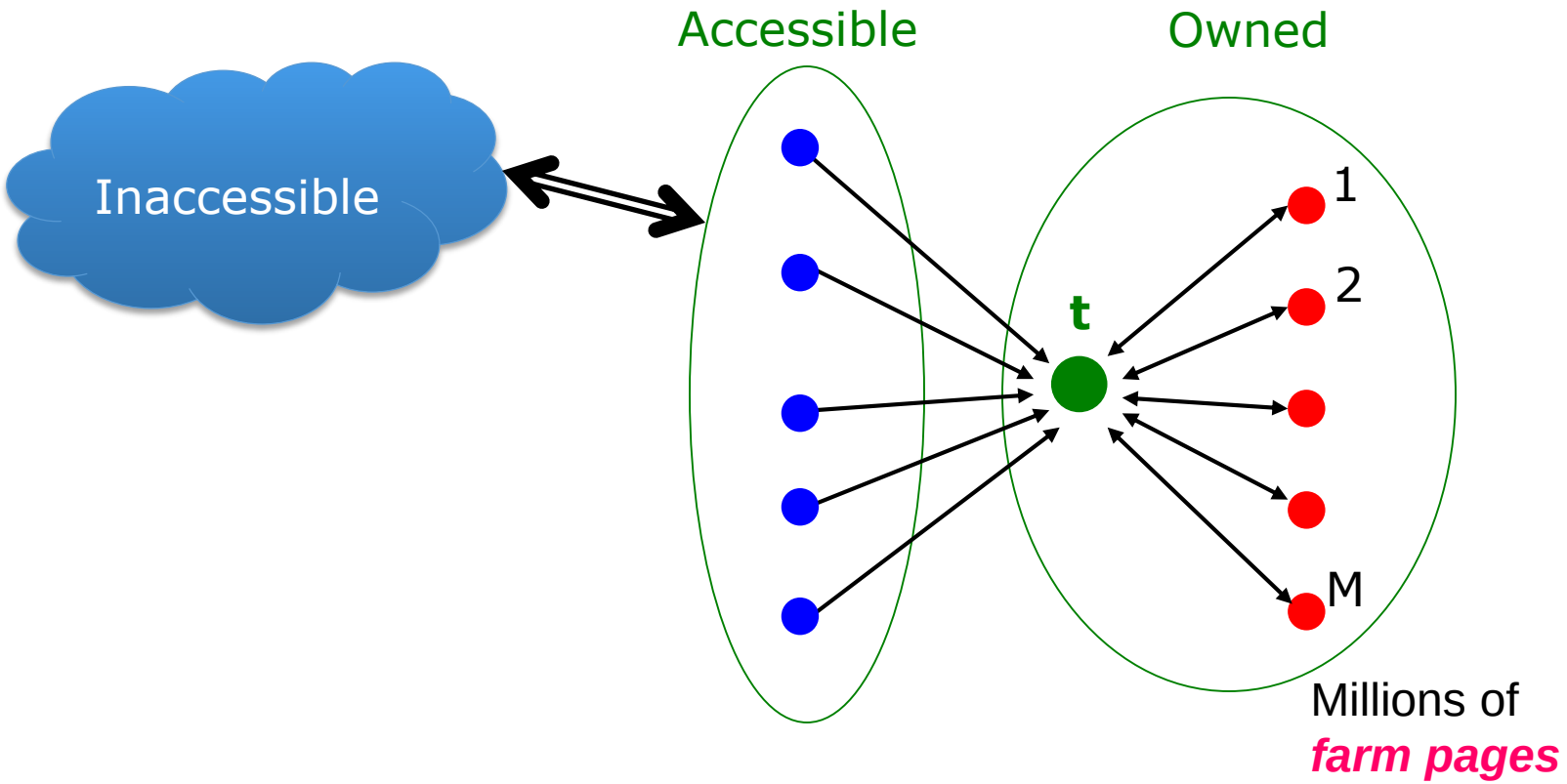
Link Spamming

- Three kinds of web pages from a spammer's point of view
  - **Inaccessible pages**
    - Visible but controlled by non-spammers
  - **Accessible pages**
    - e.g., blog comments pages
    - spammer can post links to his pages
  - **Owned pages**
    - Completely controlled by spammer
    - May span multiple domain names

Link Farms

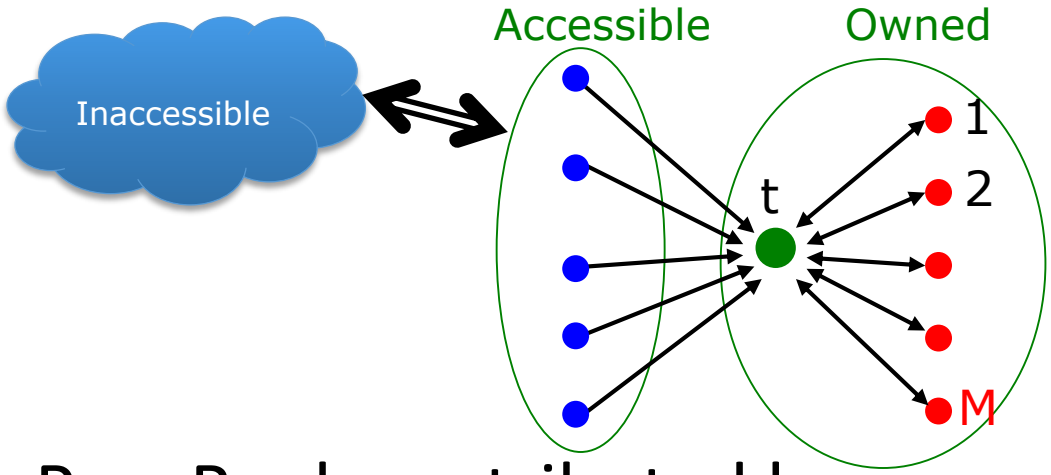
- Spammer's goal:
  - Maximize the PageRank of target page ***t***
- Technique:
  - Get as many links from accessible pages as possible to target page ***t***
  - Construct "link farm" to get PageRank multiplier effect

Link Farms



One of the most common and effective organizations for a link farm

Analysis



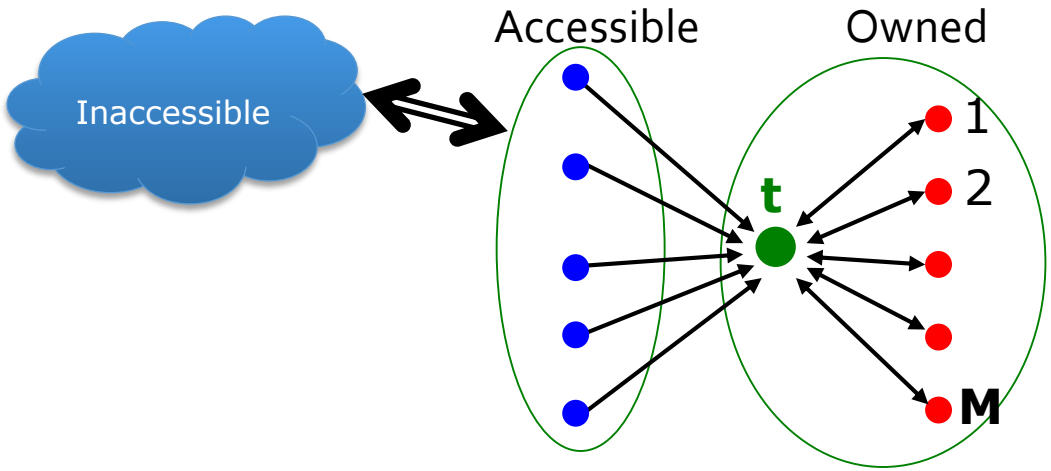
N…# pages on the web
M…# of pages spammer owns (not a matrix!)

- **x**: PageRank contributed by accessible pages
- **y**: PageRank of target page ***t***
- Rank of each "farm" page $= \frac{\beta y}{M} + \frac{1-\beta}{N}$
- $y = x + \beta M \left[\frac{\beta y}{M} + \frac{1-\beta}{N}\right] + \frac{1-\beta}{N}$

  $= x + \beta^2 y + \frac{\beta(1-\beta)M}{N} + \boxed{\frac{1-\beta}{N}}$

  Very small; ignore
  Now we solve for ***y***
- $y = \frac{x}{1-\beta^2} + c\frac{M}{N}$ where $c = \frac{\beta}{1+\beta}$

Analysis



N…# pages on the web
M…# of pages spammer owns

- $y = \frac{x}{1-\beta^2} + c\frac{M}{N}$ where $c = \frac{\beta}{1+\beta}$
- For β = 0.85, $1/(1-\beta^2) = 3.6$

- x estimates the **multiplier effect** for acquired PageRank; only partially controlled!
- But: By making M large, we can make y as large as we want!

TrustRank

Combating the Web Spam

Combating Spam

- **Combating term spam**
  - Analyze text using statistical methods
  - Similar to email spam filtering
  - Also useful: Detecting approximate duplicate pages
- **Combating link spam:**
- **Detection and blacklisting of structures that look like spam farms**
  - Leads to another war – hiding and detecting spam farms
- Better: **TrustRank** = topic-specific PageRank with a teleport set of **trusted pages**
  - Example: .edu domains, similar domains for non-US schools

TrustRank: Idea

- **Basic principle: Approximate isolation**
  - It is rare for a "good" page to point to a "bad" (spam) page
- Sample a set of **seed pages** from the web
- Have an oracle (human) to identify the good pages and the spam pages in the seed set
  - Expensive task, so we must make seed set as small as possible

Trust Propagation

- Call the subset of seed pages that are identified as **good** the **trusted pages**
- **Key idea**: Perform a (topic-sensitive) PageRank with **teleport set = trusted pages**
  - **Propagate trust through links:**
    - Each page gets a trust value between **0** and **1**
- **Option 1**: Use a threshold value and mark all pages below the trust threshold as spam

Simple Model: Trust Propagation

- Set trust of each trusted page to 1
- Suppose trust of page p is t_p
  - Page p has a set of out-links o_p
- For each $q \in o_p$, p **confers the trust** to q
  - $\beta t_p / |o_p|$ for $0 < \beta < 1$
- Trust is additive
  - Trust of p is the sum of the trust conferred on p by all its in-linked pages

Why is it a good idea?

- **Trust attenuation:**
  - The degree of trust conferred by a trusted page decreases with the distance in the graph
- **Trust splitting:**
  - The larger the number of out-links from a page, the less scrutiny the page author gives each out-link
  - Trust is **split** across out-links

Picking the Seed Set

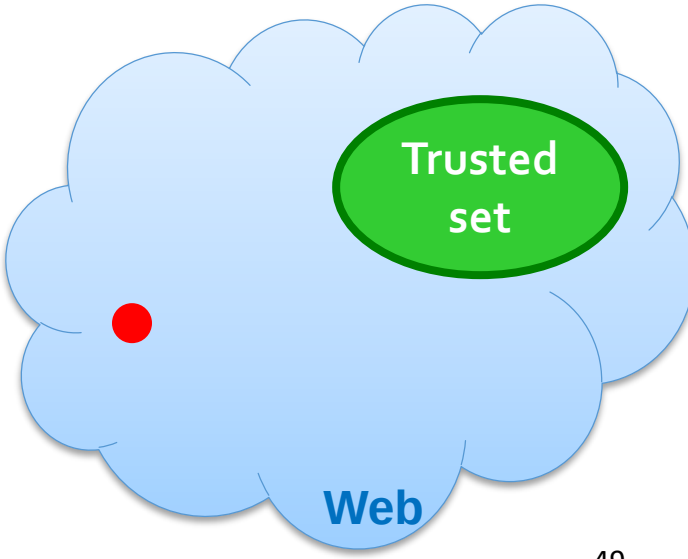
- **Two conflicting considerations:**
  - Human has to inspect each seed page, so seed set must be as small as possible
  - Must ensure every **good page** gets adequate trust rank, so need make all good pages reachable from seed set by short paths

Approaches to Picking Seed Set

- Suppose we want to pick a seed set of ***k*** pages
- **How to do that?**
- **(1) PageRank:**
  - Pick the top ***k*** pages by PageRank
  - Theory is that you can't get a bad page's rank really high
- **(2) Use trusted domains** whose membership is controlled, like .edu, .mil, .gov

Spam Mass

- In the **TrustRank** model, we start with good pages and propagate trust
- **Complementary view:**
  What fraction of a page's PageRank comes from **spam** pages?
- In practice, we don't know all the spam pages, so we need to estimate



Spam Mass Estimation

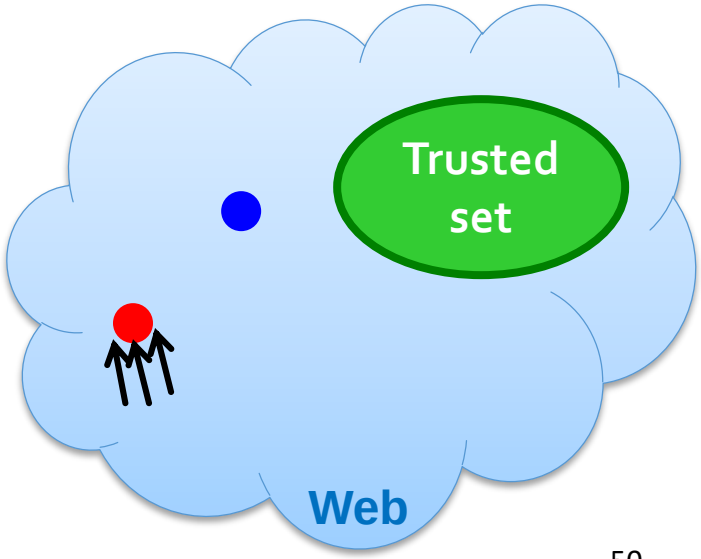
Option 2:

- r_p = PageRank of page p
- r_p^+ = PageRank of p with teleport into **trusted** pages only

- **Then:** What fraction of a page's PageRank comes from spam pages?

$$r_p^- = r_p - r_p^+$$

- **Spam mass of p = $\frac{r_p^-}{r_p}$**
  - Pages with high spam mass are spam



Thank you.

Questions?