RankMass Crawler

A Crawler with High Personalized PageRank Coverage Guarantee

Junghoo Cho
Uri Schonfeld

Work done in UCLA

shuri@shuri.org

Motivation

- Impossible to download the entire web:
  - Example: many pages from one calendar
- When can we stop?
- How to gain the most benefit from the pages we download

Main Issues

- Crawler Guarantee:
  - guarantee on how much of the "important" part of the Web they "cover" when they stop crawling
  - If we don't see the pages, how do we know how important they are?
- Crawler Efficiency:
  - Download "important" pages early during a crawl
  - Obtain coverage with a min number of downloads

Outline

- Formalize coverage metric
- L-Neighbor: Crawling with RankMass guarantee
- RankMass: Crawling to achieve high RankMass
- Windowed RankMass: How greedy do you want to be?
- Experimental Results

Web Coverage Problem

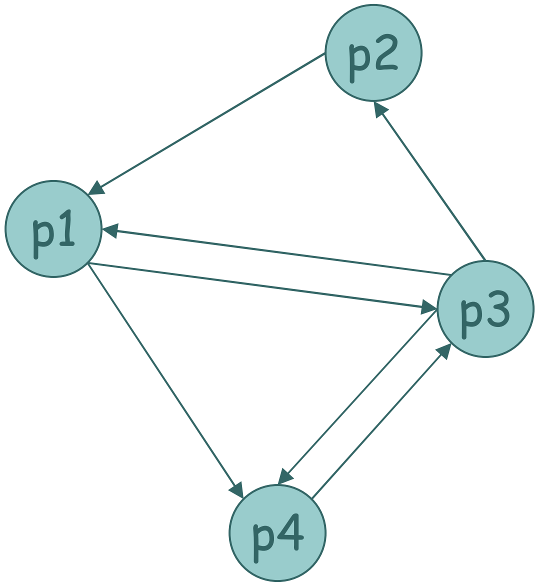
- D – The potentially infinite set of documents of the web
- D_C – The finite set of documents in our document collection
- Assign importance weights to each page

shuri@shuri.org © http://www.shuri.org

Web Coverage Problem

- What weights? Per query? Topic? Font?
- PageRank? Why PageRank?
  - Useful as importance mesure
  - Random surfer.
  - Effective for ranking.

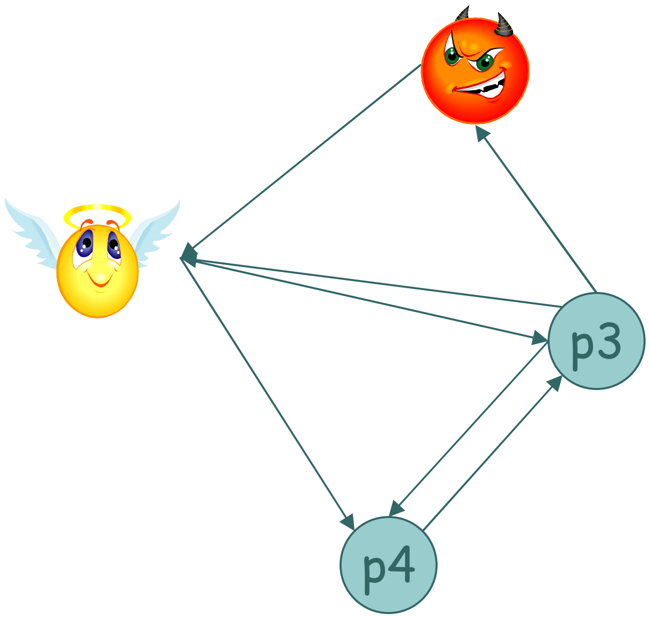
PageRank a Short Review



$$r_i = d\left[\sum_{p_{j \in I(p_i)}} \frac{r_j}{c_j}\right] + (1-d)\frac{1}{|D|}$$

Now it's Personal

- Personal, TrustRank, General



$$T = \begin{bmatrix} t_1 \\ t_2 \\ \vdots \end{bmatrix}$$

$$r_i = d\left[\sum_{p_j \in I(p_i)} \frac{r_j}{c_j}\right] + (1-d)t_i$$

RankMass Defined

- Using personalized pagerank formally define RankMass of D_C :

$$RM(D_C) = \sum_{p_i \in D_C} r_i$$

- Coverage Guarantee:
  - We seek a crawler that given □, when it stops the downloaded pages D_C:

$$RM(D_C) = \sum_{p_i \in D_C} r_i > 1 - \varepsilon$$

- Efficient crawling:
  - We seek a crawler that, for a given N, downloads $|D_C|=N$ s.t. $RM(D_C)$ is greater or equal to any other $|D_C|=N$, D_C□ D

How to Calculate RankMass

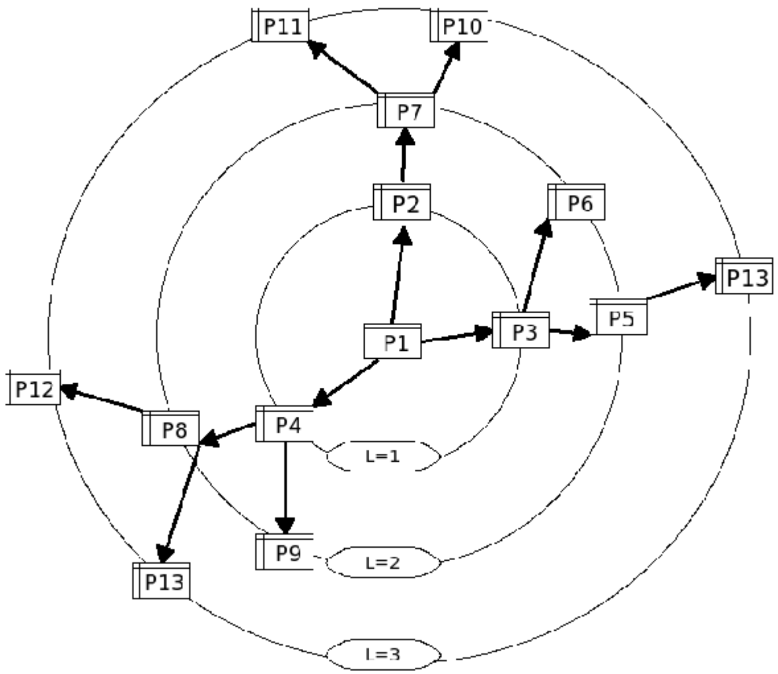
- Based on PageRank
- How do you compute RM(Dc) without downloading the entire web
- We can't compute the exact but can lower bound
- Let's a start a simple case

shuri@shuri.org http://www.shuri.org

Single Trusted Page

- $t_1=1$; $t_i = 0$ $i \neq 1$
- Always jump to p_1 when bored
- We can place a lowerbound on being within L of P_1
- $N_L(p_1)$=pages reachable from p_1 in L links

Single Trusted Page



Lower bound guarantee: Single Trusted

- Theorem 1:
  - Assuming the trust vector $T^{(1)}$, the sum of the PageRank values of all L-neighbors of p1 is at least d^{L+1} close to 1.. That is:

$$\boxed{\sum_{p_i \in N_L(p_1)} r_i \geq 1 - d^{L+1}}$$

Lower bound guarantee: General Case

- The RankMass of the L-neighbors of the group of all trusted pages G, $N_L(G)$, is at least d^{L+1} close to 1. That is:

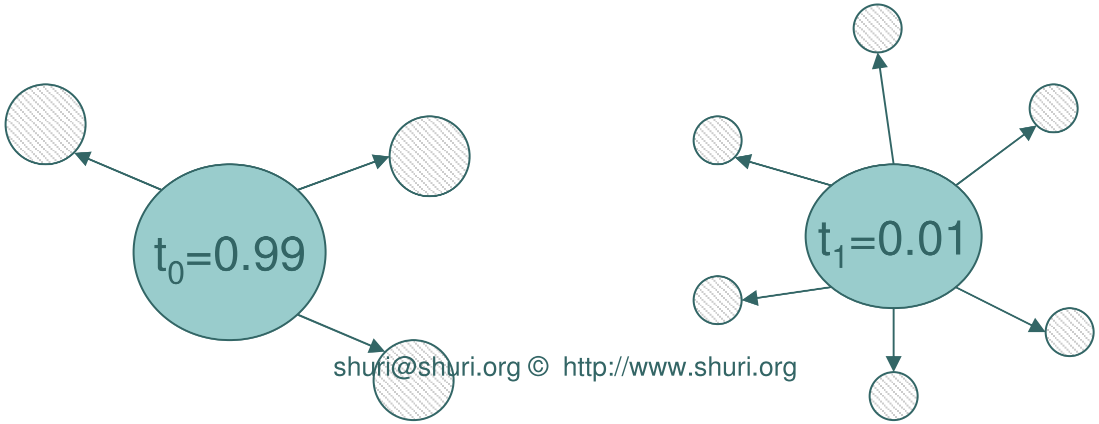
$$\boxed{\sum_{p_i \in N_L(G)} r_i \geq 1 - d^{L+1}}$$

The L-Neighbor Crawler

1. $L := 0$
2. $N[0] = \{pi|ti > 0\}$ // Start with the trusted pages
3. While (□ < d^{L+1})
   1. Download all uncrawled pages in N[L]
   2. $N[L + 1] = \{$all pages linked to by a page in $N[L]\}$
   3. $L = L + 1$

But what about efficency?

- L-Neighbor similar to BFS
- L-Neighbor simple and efficient
- May wish to prioritize further certain neighborhoods first
- Page level prioritization.

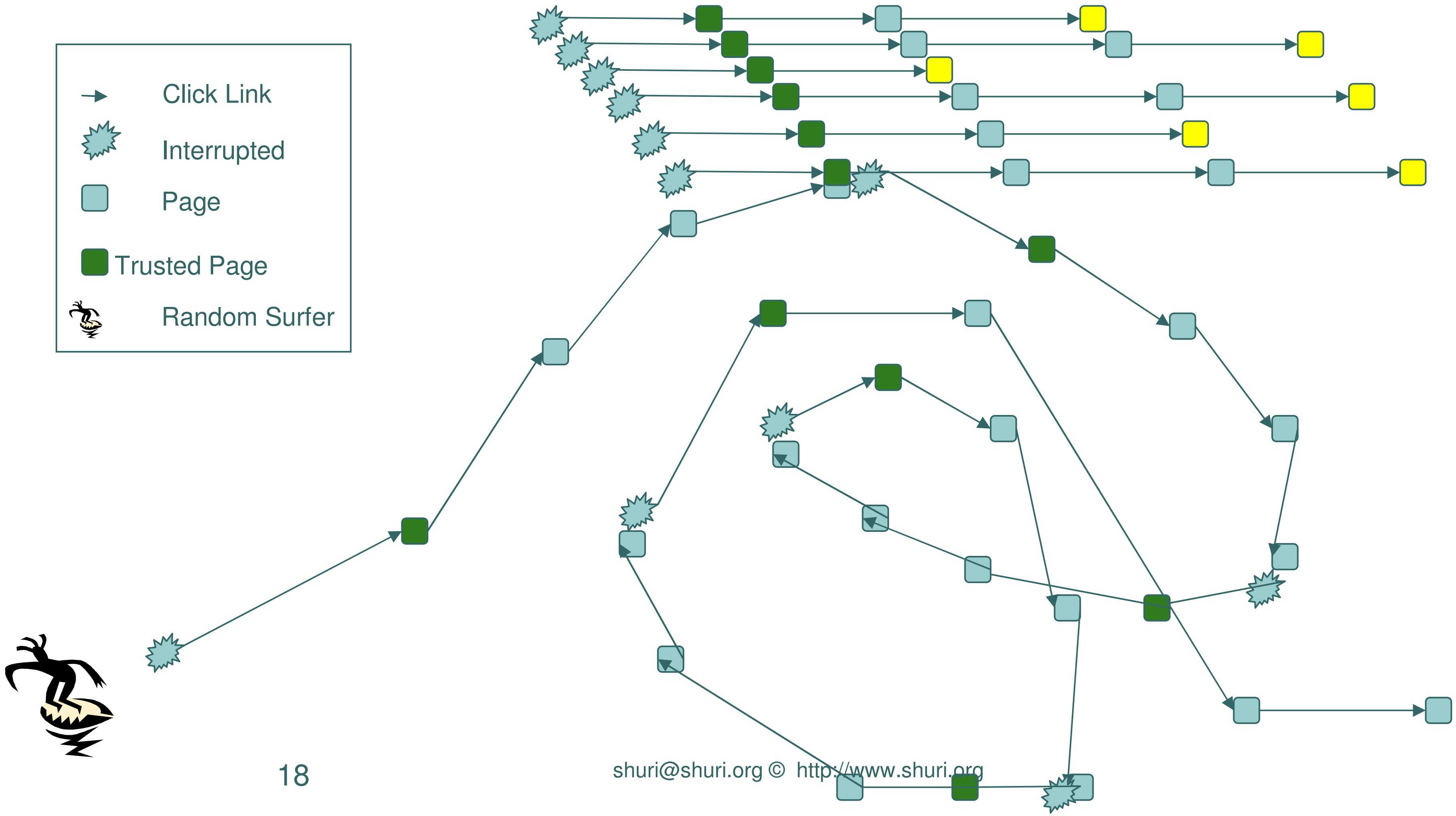


Page Level Prioritizing

- We want a more fine-grained page-level priority
- The idea:
  - Estimate PageRank on a page basis
  - High priority for pages with a high estimate of PageRank
- We cannot calculate exact PageRank
- Calculate PageRank lower bound of undownloaded pages

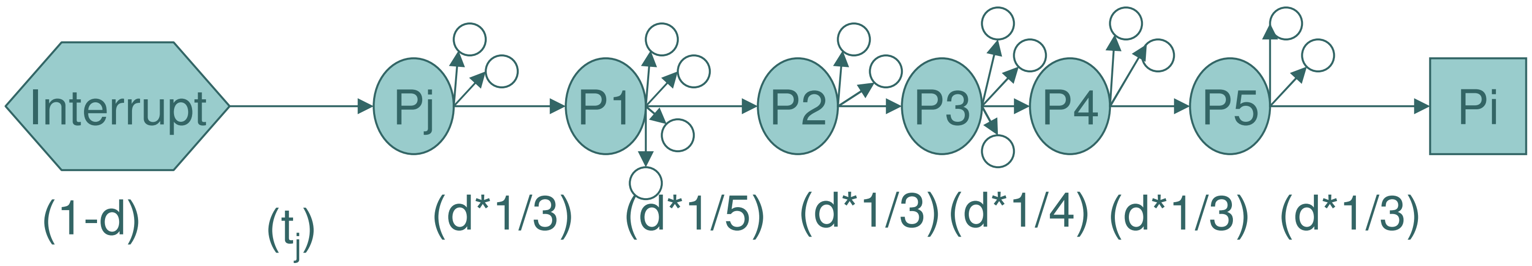
shuri@shuri.org © http://www.shuri.org

Probability of being at Page P

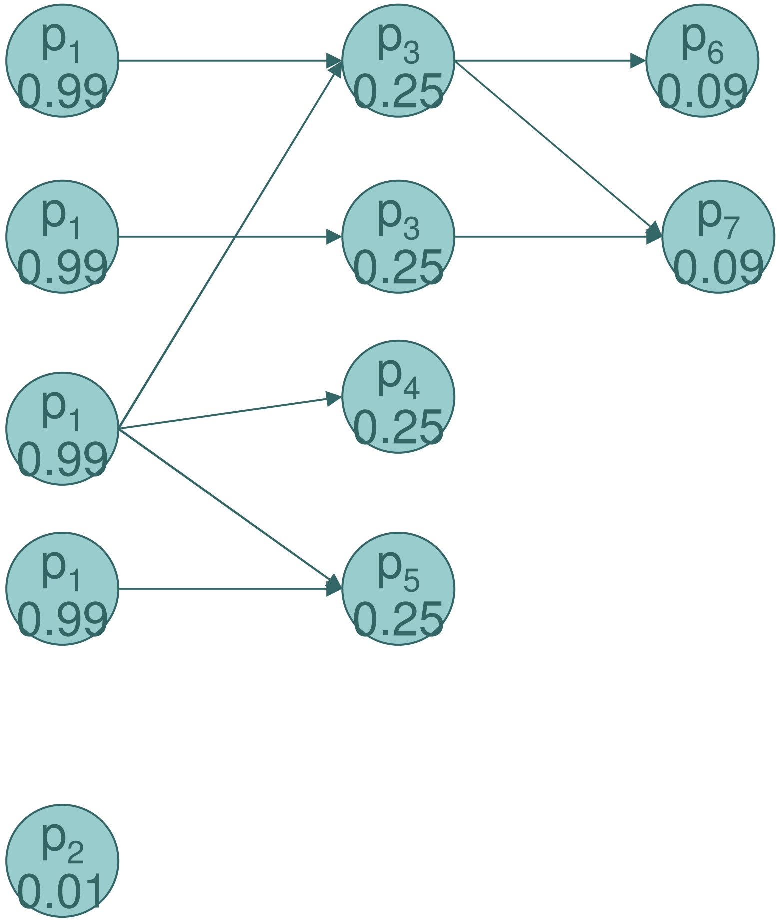


Calculating PageRank Lower Bound

- PageRank(p) = Probability Random Surfer in p
- Breakdown path by "interrupts", jumps to a trusted page
- Sum up all paths that start with an interrupt and end with p



RankMass Basic Idea



RankMass Crawler: High Level

- But that sounds complicated?!
- Luckily we don't need all that
- Based on this idea:
  - Dynamically update lower bound on PageRank
  - Update total RankMass
  - Download page with highest lower bound

shuri@shuri.org http://www.shuri.org

RankMass Crawler (Shorter)

- Variables:
  - CRM: RankMass lower bound of crawled pages
  - rmi: Lower bound of PageRank of pi.

- RankMassCrawl()
  - CRM = 0
  - $rm_i = (1 - d)t_i$ for each $t_i > 0$
  - While (CRM < 1 – □):
    - Pick p_i with the largest rm_i.
    - Download pi if not downloaded yet
    - $CRM = CRM + rm_i$
    - Foreach p_j linked to by p_i:
      - $rm_j = rm_j + d/c_i\, rm_i$
    - $rm_i = 0$

Experimental Setup

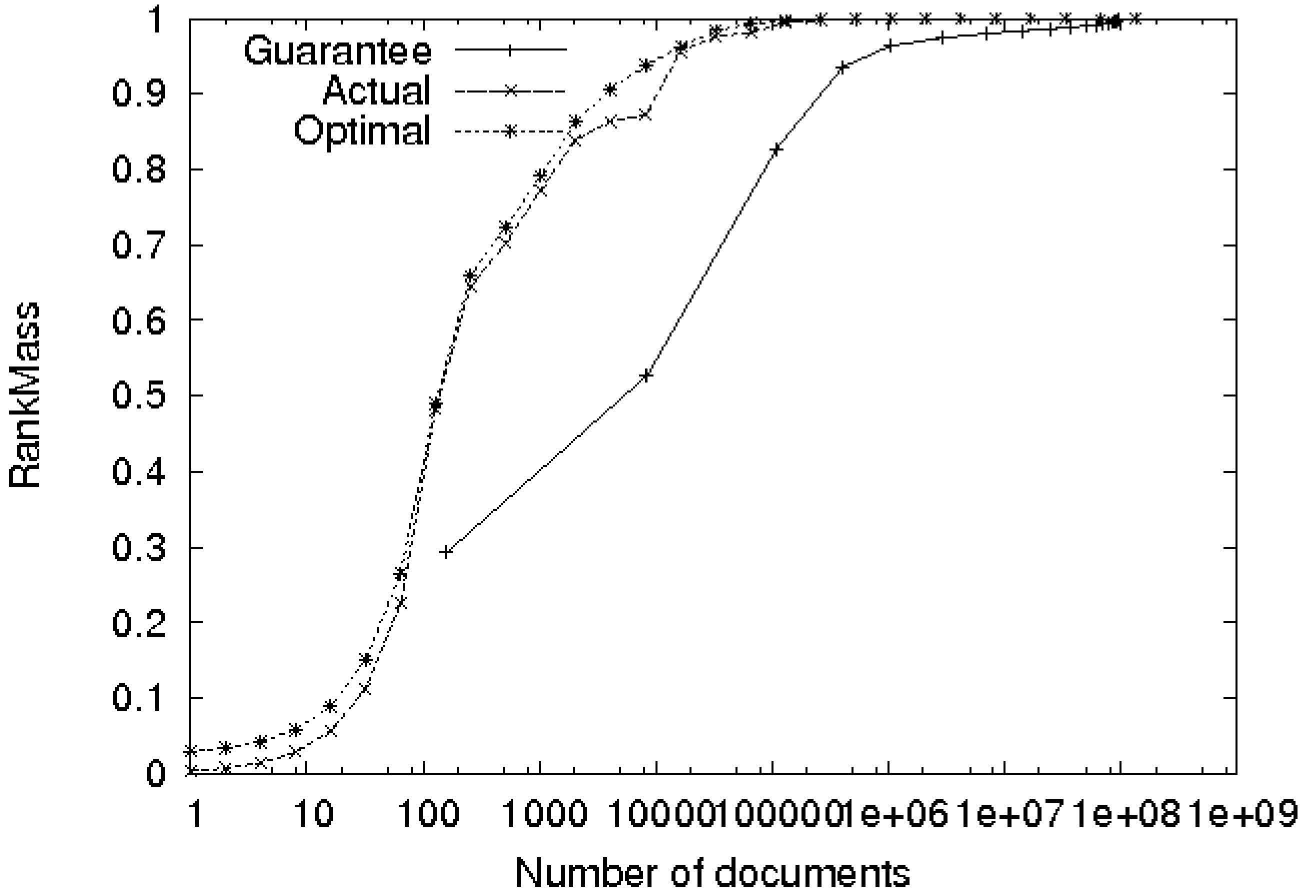
- HTML files only
- Algorithms simulated over web graph
- Crawled between Dec' 2003 and Jan' 2004
- 141 millon URLs span over 6.9 million host names
- 233 top level domains.

shuri@shuri.org © http://www.shuri.org

Metrics Of Evaluation

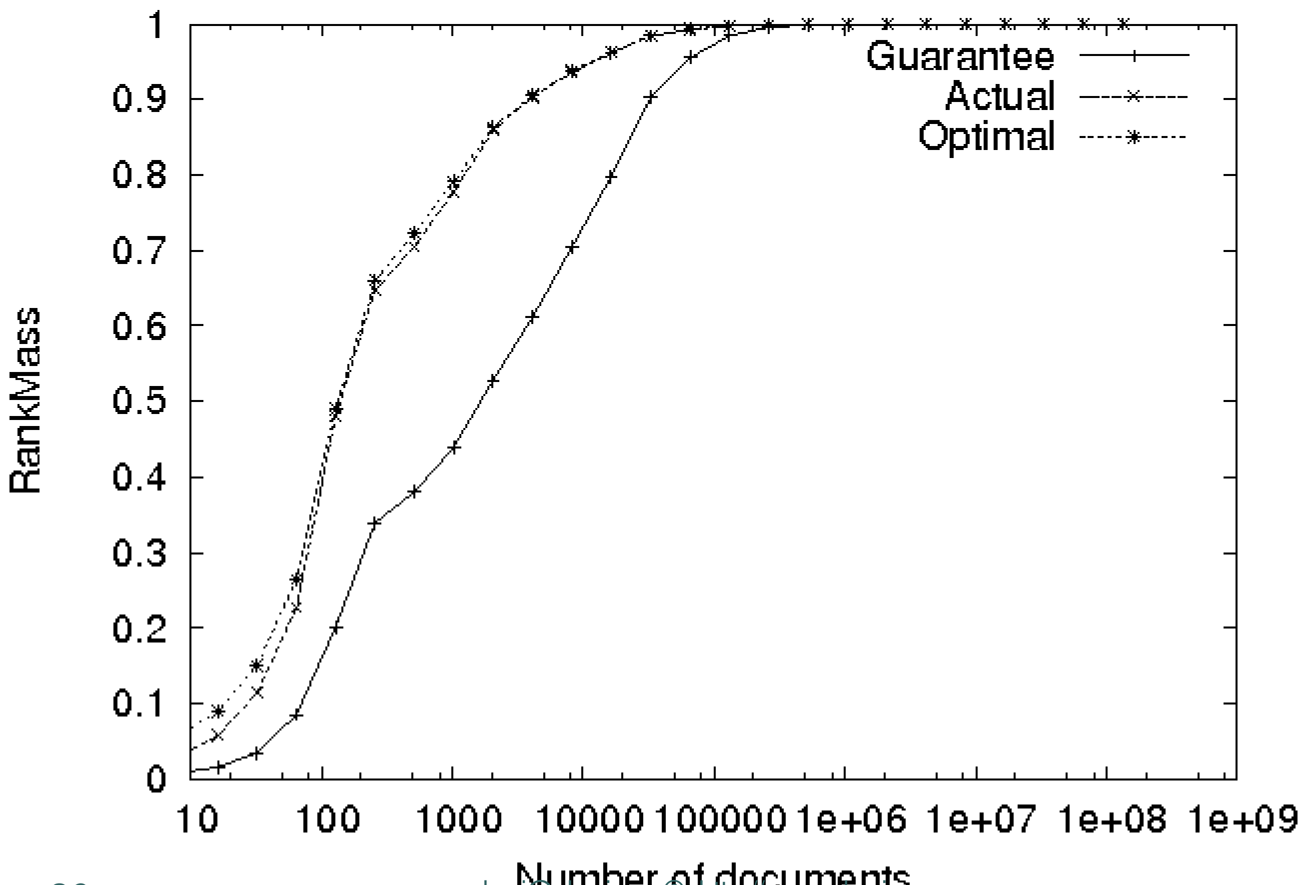
1. How much RankMass is collected during the crawl
2. How much RankMass is "known" to have been collected during the crawl
3. How much computational and performance overhead the algorithm introduces.

L-Neighbor



Guarantee
Actual
Optimal
RankMass
Number of documents
0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9
1
1
10
100
1000
10000
100000
1e+06
1e+07
1e+08
1e+09

RankMass



Guarantee
Actual
Optimal
RankMass
1
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0
10
100
1000
10000
100000
1e+06
1e+07
1e+08
1e+09
Number of documents

Algorithm Efficiency

Algorithm	Downloads required for above 0.98% guaranteed RankMass	Downloads required for above 0.98% actual RankMass
L-Neighbor	7 million	65,000
RankMass	131,072	27,939
Windowed-RankMass	217,918	30,826
Optimal	27,101	27,101

Algorithm Running Time

Window	Hours	Number of Iterations	Number of Documents
L-Neighbor	1:27	13	83,638,834
20%-Windowed	4:39	44	80,622,045
10%-Windowed	10:27	85	80,291,078
5%-Windowed	17:52	167	80,139,289
RankMass	25:39	Not comparable	10,350,000

shuri@shuri.org

Thank you

- Thank you

Greedy vs Simple

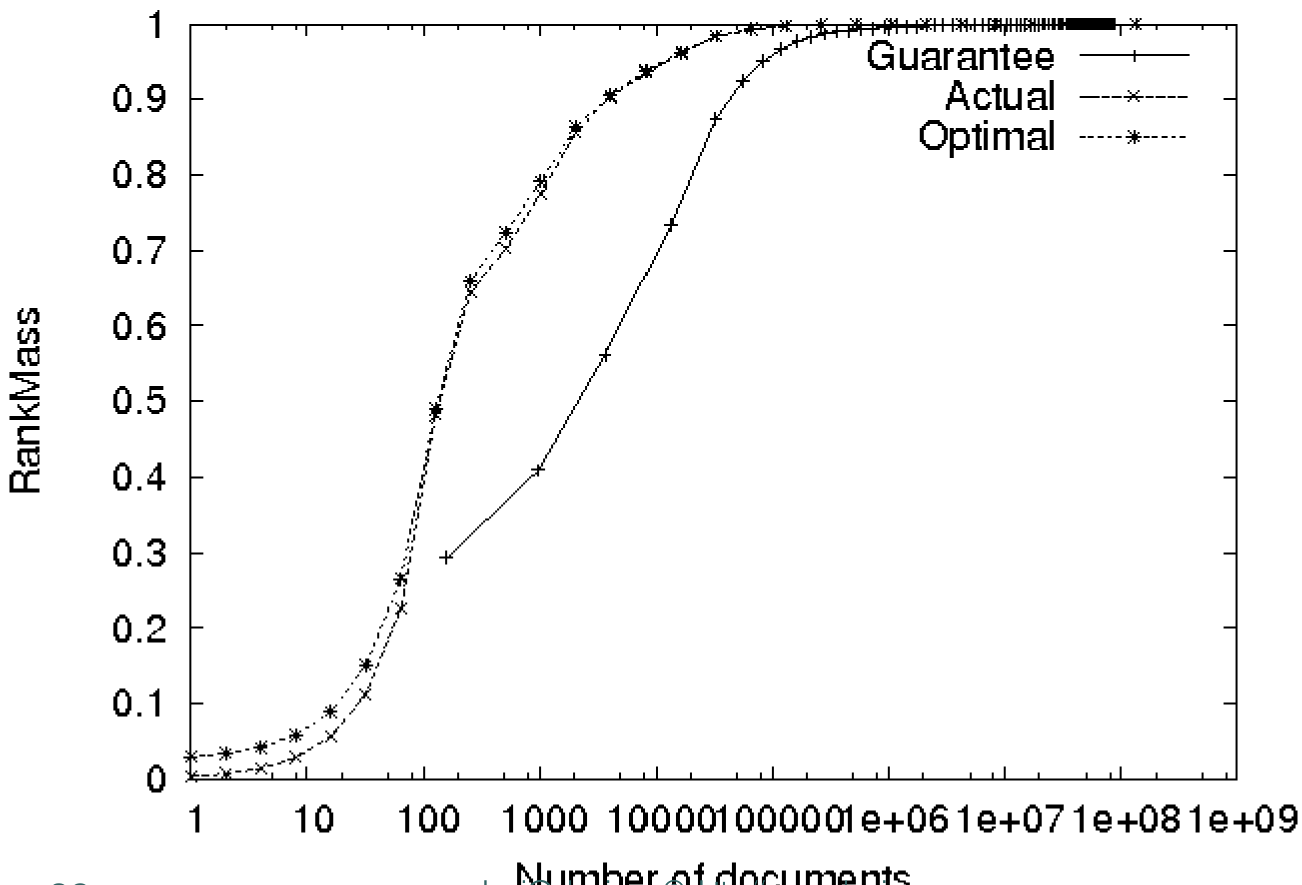
- L-Neighbor is simple
- RankMass is very greedy.
- Update expensive: random access to web graph
- Compromise?
- Batching
  - downloads together
  - updates together

shuri@shuri.org

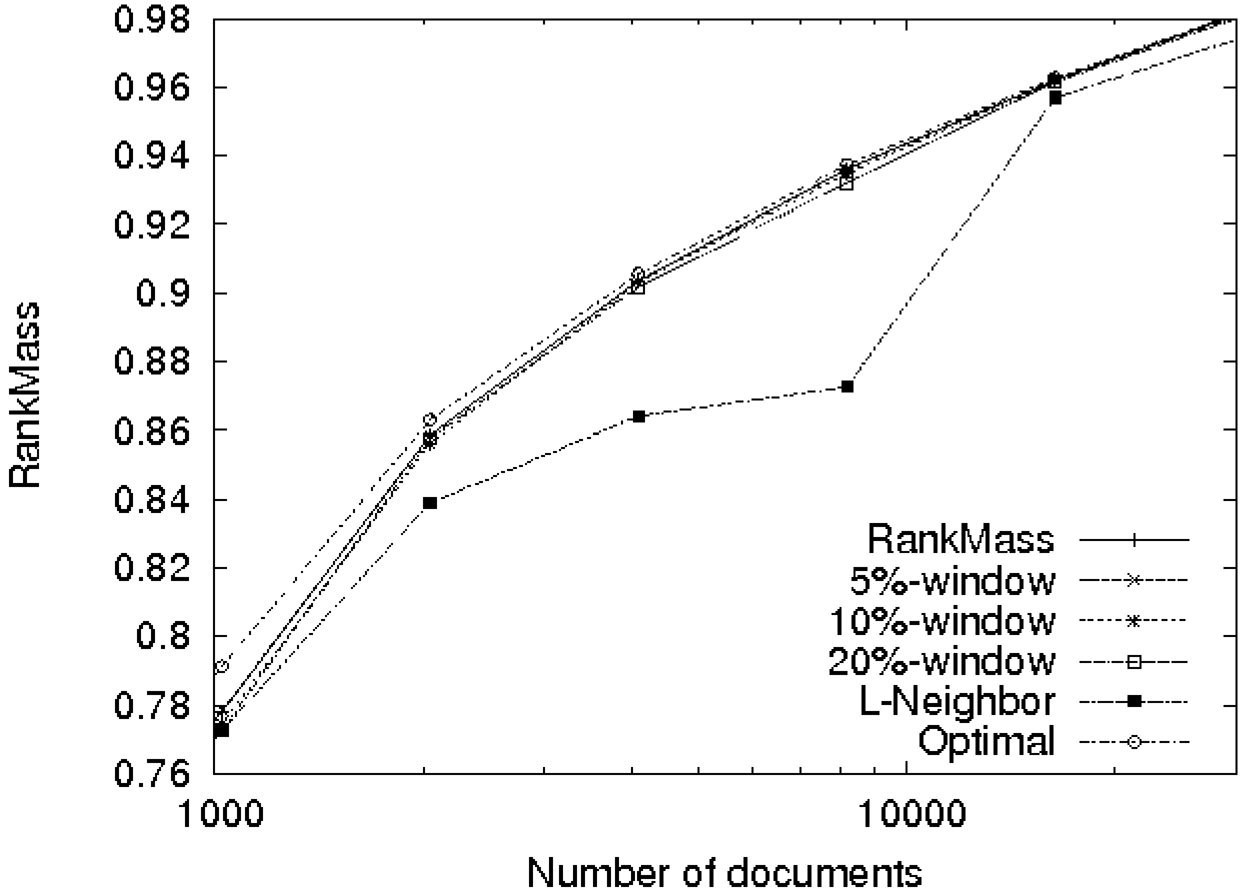
Windowed RankMass

- Variables:
  - CRM: RankMass lower bound of crawled pages
  - rm_i: Lower bound of PageRank of p_i.
- Crawl()
  - rmi = (1 – d)ti for each ti > 0
  - While (CRM < 1 – □):
    - Download **top window%** pages according to rm_i
    - Foreach page p_i □ D_C
      - CRM = CRM + rm_i
      - Foreach p_j linked to by p_i:
      - $rm_j = rm_j + d/c_i\ rm_i$
    - $rm_i = 0$

Windowed RankMass



Window Size



0.98
0.96
0.94
0.92
0.9
0.88
0.86
0.84
0.82
0.8
0.78
0.76
RankMass
1000
10000
Number of documents
RankMass
5%-window
10%-window
20%-window
L-Neighbor
Optimal

RankMass Lower Bound

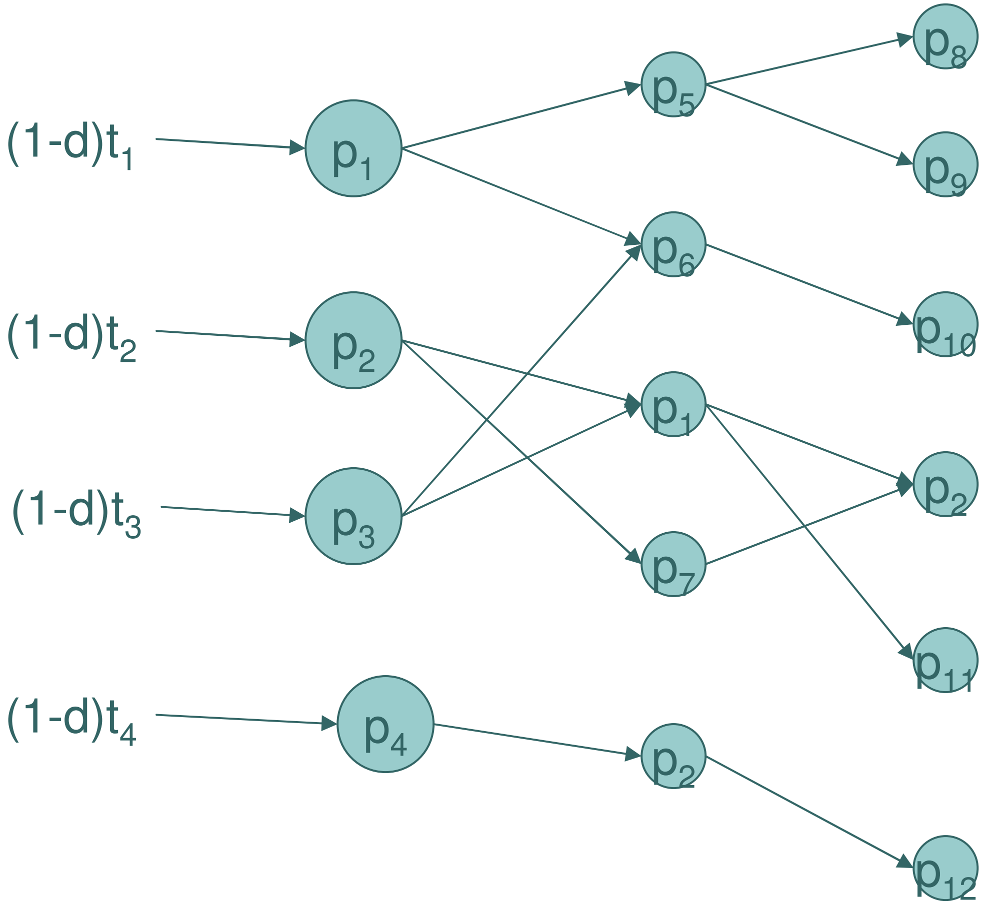
- Lower bound given a single trusted page

$$\sum_{p_i \in N_L(p_1)} r_i \geq 1 - d^{L+1}$$

- That's the basis of the crawling algorithm with a coverage guarantee
- Extension: Given a set of trusted pages G

$$\sum_{p_i \in N_L(G)} r_i \geq 1 - d^{L+1}$$

Power Method of Calculating PageRank



shuri@shuri.org http://www.shuri.org

RankMass Algorithm

- **Variables:**
  - **UnexploredPaths:** *List of unexplored paths and their path probabilities*
  - **sumPathProb$_i$:** *Sum of the probabilities of all unexplored paths leading to pi*
  - **r_i:** *Partial sum of the probability of being in pi*

- **RankMassCrawl()**
  - **// Initialize:**
  - **r_i = 0 for each i** *// Set initial probability sum to be zero.*
  - **UnexploredPaths = {}** *// Start with empty set of paths.*
- **Foreach (t_i > 0):** *// Add initial paths of jumping to a trusted page and*
  - **Push [path: {pi}, prob: (1 – d)t_i] to UnexploredPaths** *// the probability of the random jump.*
  - **sumPathProb$_i$ = (1 – d)t_i** *// For every trust page pi, we currently have only one path {pi}*

RankMass Algorithm

- While ($\sum_i r_i < 1 -$ □):
- Pick p_i with the largest $sumPathProb_i$. // Get the page with highest sumPathProbi.
- Download p_i if not downloaded yet // Crawl the page.
- // Now expand all paths that end in pi
- PathsToExpand = Pop all paths ending with pi // Get all the paths leading to pi,
- from UnexploredPaths
- Foreach p_j linked to from p_i // and expand them by adding pi's children to the paths.
  - Foreach [path, prob] 2 PathsToExpand
    - path' = path · pj // Add the child pj to the path,
    - prob' = d/c_{ii} prob // compute the probability of this expanded path,
    - Push [path', prob'] to UnexploredPaths // and add the expanded path to UnexploredPaths.
  - sumPathProbj = sumPathProbj + d/ci sumPathProbi // Add the path probabilities of the newly added paths to pj .

- Add the probabilities of just explored paths to r_i
- $r_i = r_i + sumPathProb_i$ // We just explored all paths to pi. Add their probabilities
- sumPathProbi = 0 // to r_i.