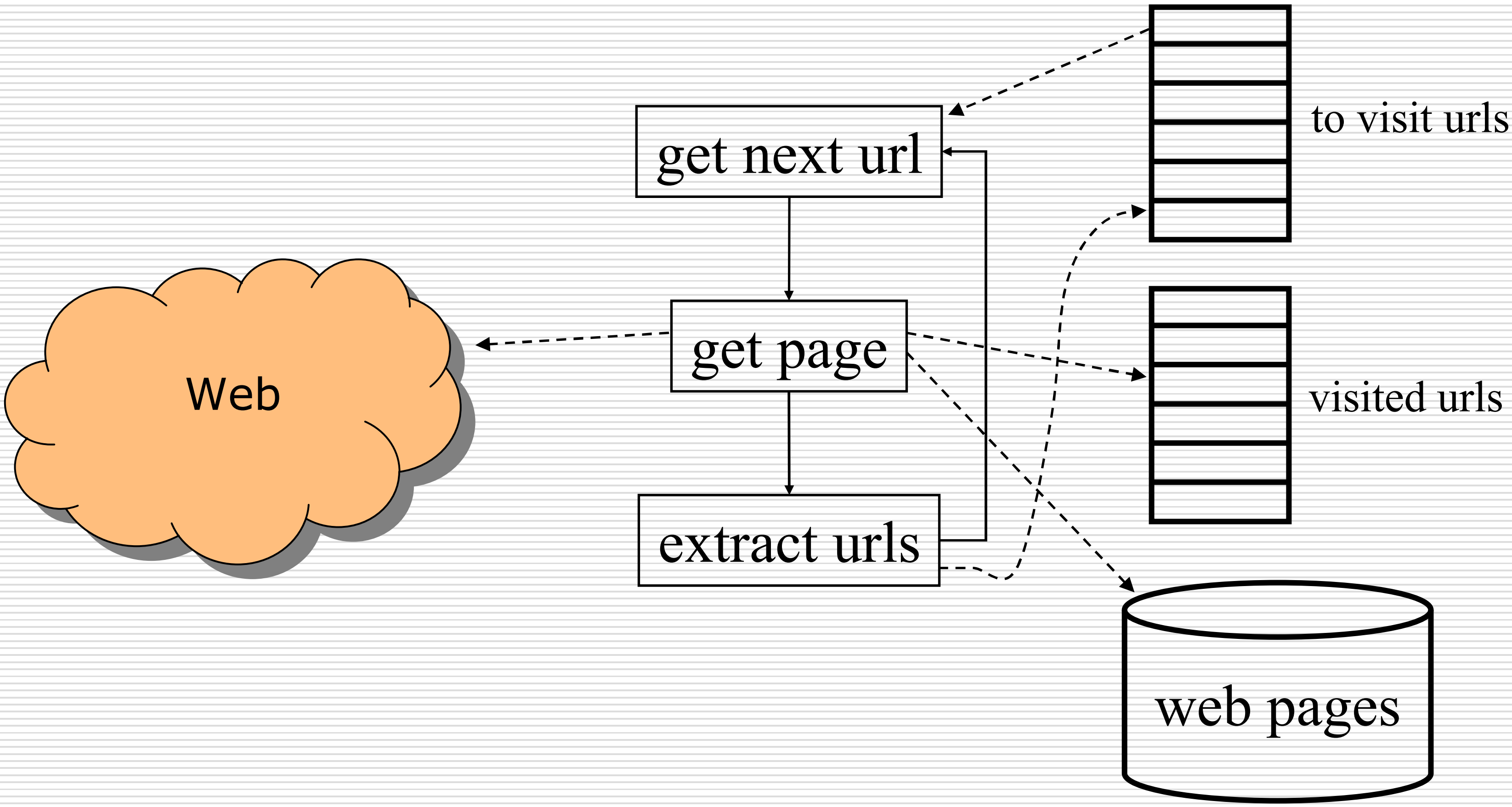
CS345
Data Mining

Crawling the Web

Web Crawling Basics

Start with a “seed set” of to-visit urls



Crawling Issues

- Load on web servers
- Insufficient resources to crawl entire web
  - Which subset of pages to crawl?
- How to keep crawled pages “fresh”?
- Detecting replicated content e.g., mirrors
- Can’t crawl the web from one machine
  - Parallelizing the crawl

Polite Crawling

- Minimize load on web servers by spacing out requests to each server
  - E.g., no more than 1 request to the same server every 10 seconds
- Robot Exclusion Protocol
  - Protocol for giving spiders (“robots”) limited access to a website
  - www.robotstxt.org/wc/norobots.html

Crawl Ordering

- Not enough storage or bandwidth to crawl entire web
- Visit “important” pages first
- Importance metrics
  - In-degree
    - More important pages will have more inlinks
  - Page Rank
    - To be discussed later
    - For now, assume it is a metric we can compute

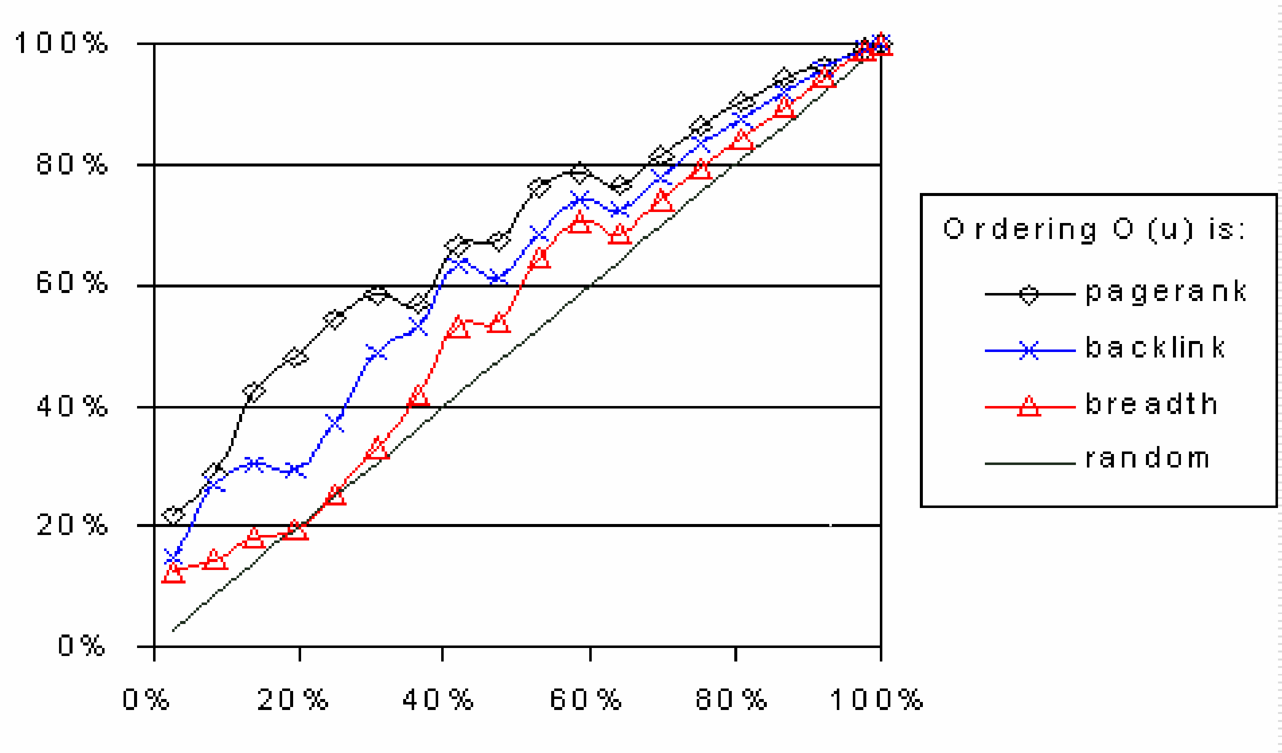
Crawl Order

- Problem: we don’t know the actual in-degree or page rank of a page until we have the entire web!
- Ordering heuristics
  - Partial in-degree
  - Partial page rank
  - Breadth-first search (BFS)
  - Random Walk -- baseline

stanford.edu experiment

- 179K pages

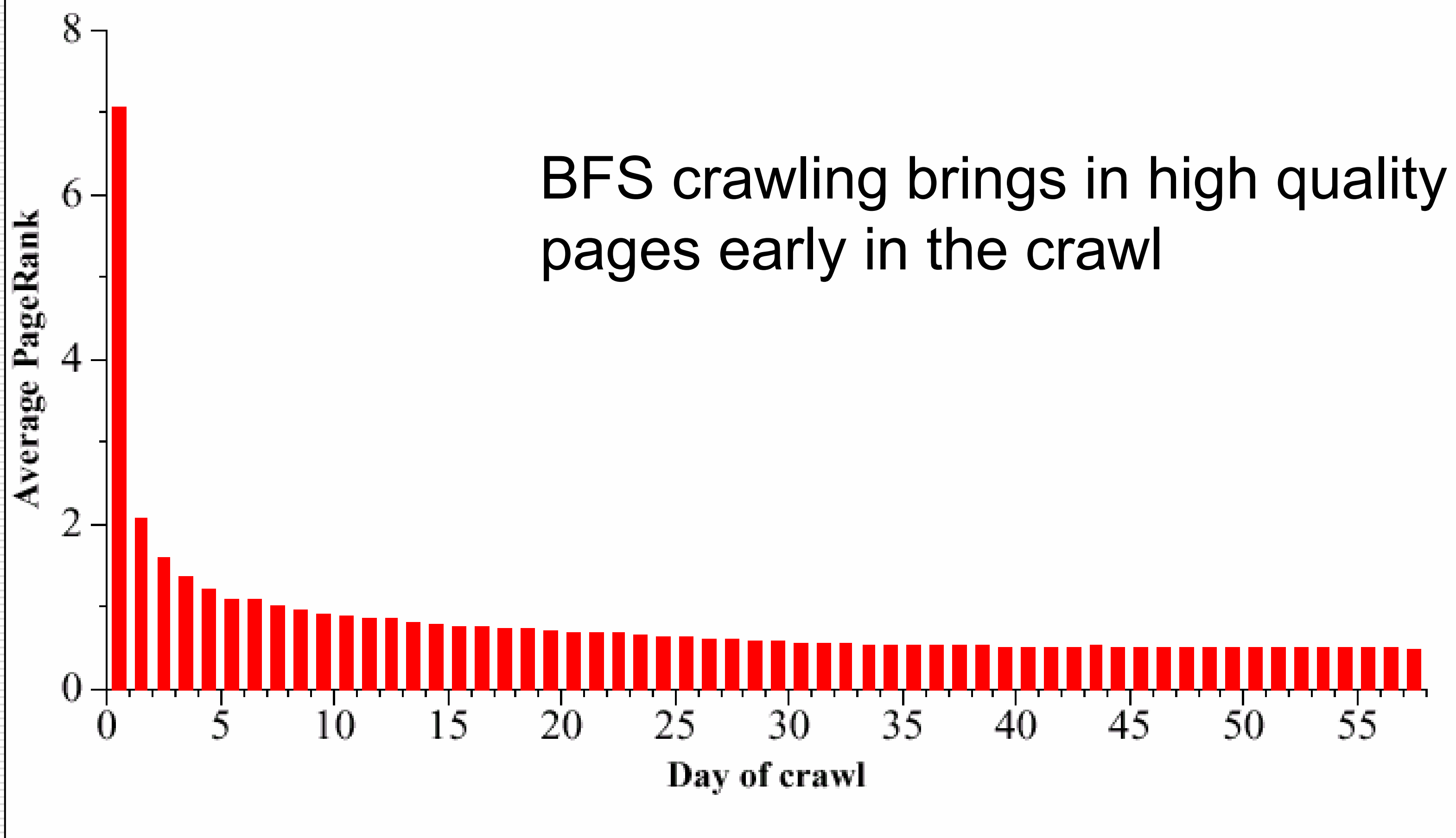
Overlap with best x% by indegree



x% crawled by O(u)

Source: Cho et al (1998)

Larger study (328M pages)



Average PageRank score by day of crawl

Source: Najork and Wiener (2001)

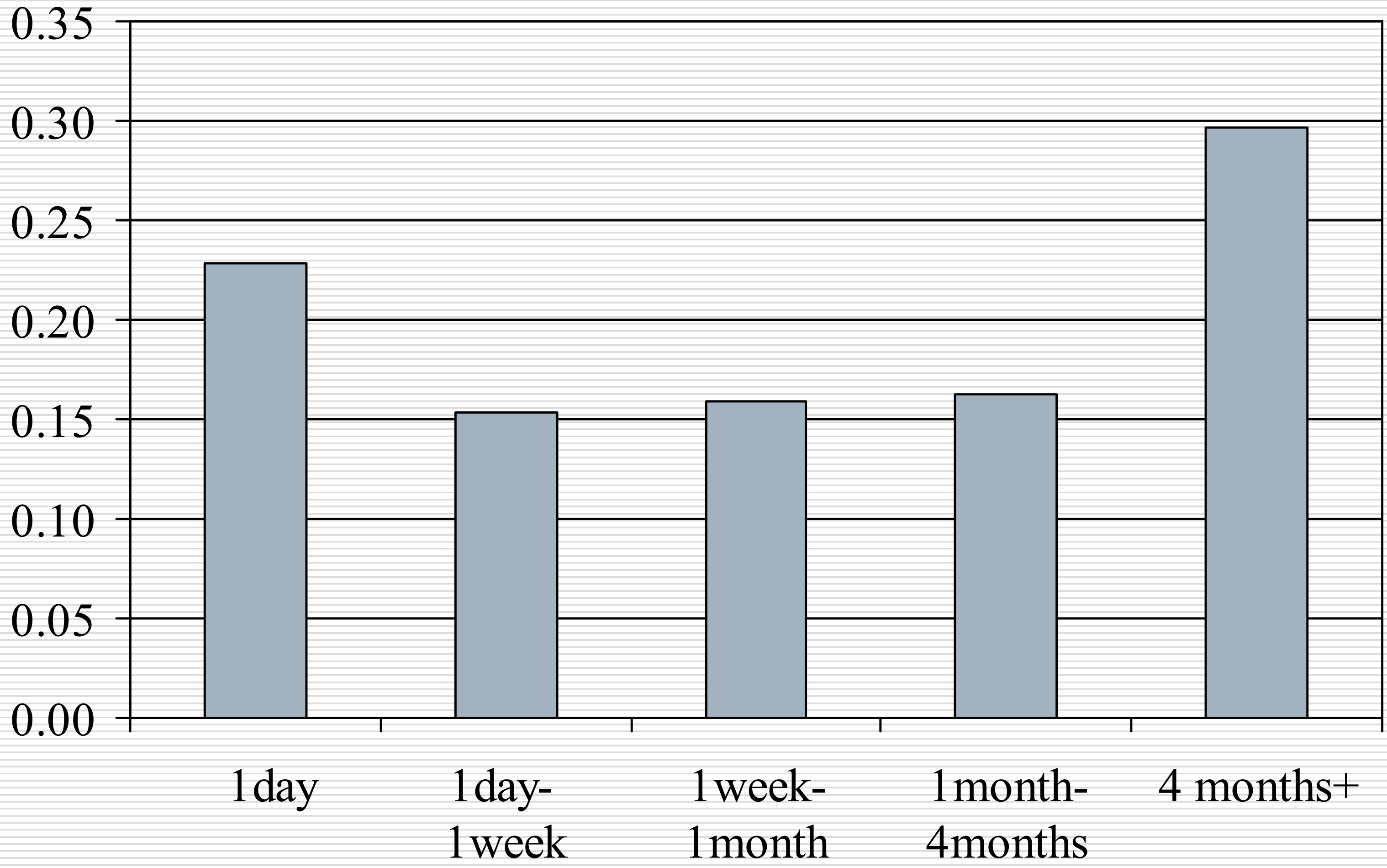
Maintaining freshness

- How often do web pages change?
- What do we mean by freshness?
- What strategy should we use to refresh pages?

How often do pages change?

- Cho et al (2000) experiment
- 270 sites visited (with permission)
  - identified 400 sites with highest "PageRank"
  - contacted administrators
- 720,000 pages collected
  - 3,000 pages from each site daily
  - start at root, visit breadth first (get new & old pages)
  - ran only 9pm - 6am, 10 seconds between site requests

Average change interval



Source: Cho et al (2000)

Modeling change

- Assume changes to a web page are a sequence of random events that happen independently at a fixed average rate
- Poisson process with parameter λ
- Let $X(t)$ be a random variable denoting the number of changes in any time interval t

  $Pr[X(t)=k] = e^{-\lambda t}(\lambda t)^k/k!$ for $k = 0,1,\ldots$

- "Memory-less" distribution

Poisson processes

- Let us compute the expected number of changes in unit time

  $E[X(1)] = \sum_k k e^{\lambda} \lambda^k / k! = \lambda$

- λ is therefore the average number of changes in unit time
- Called the rate parameter

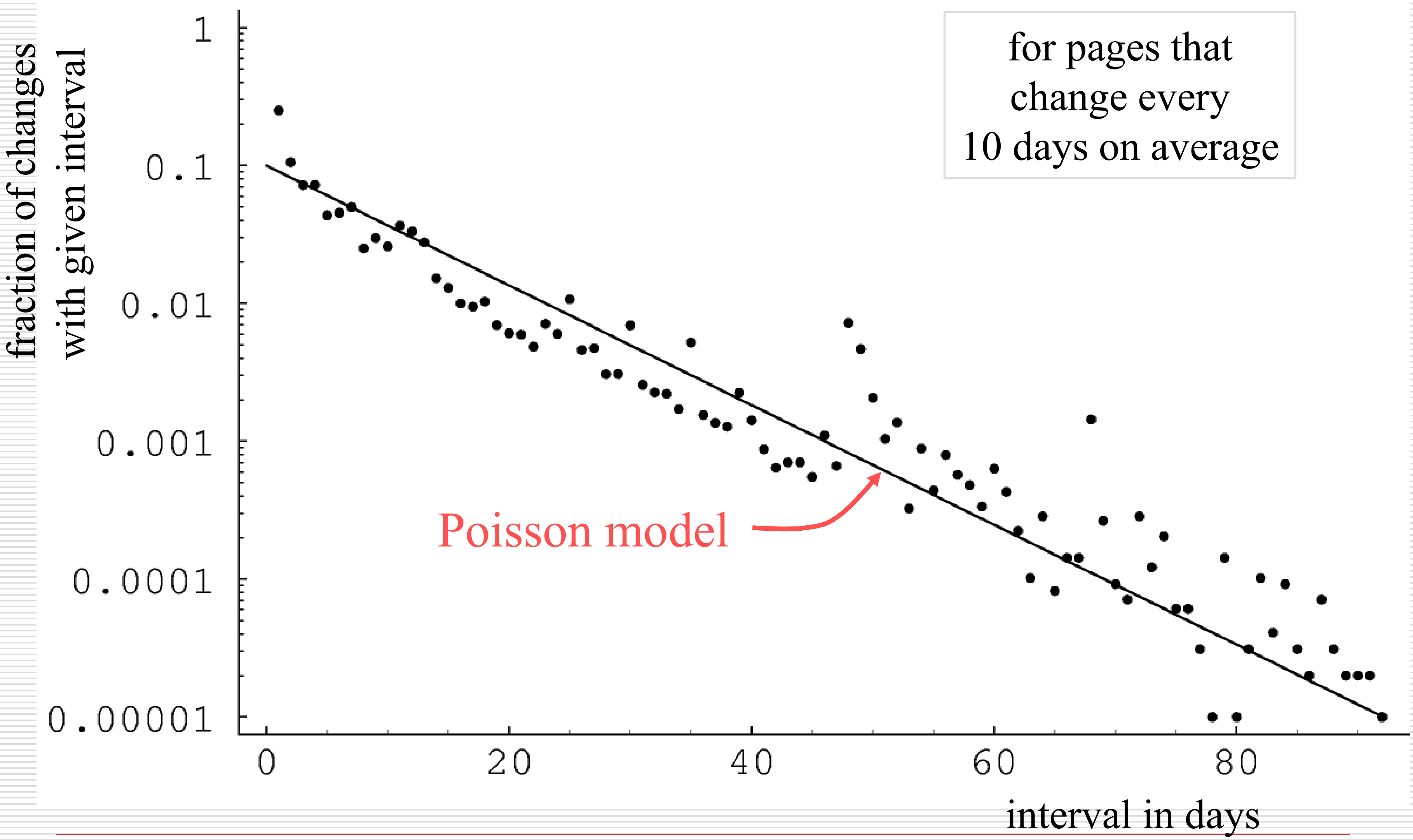
Time to next event

- Let T be a random variable denoting the time to the next event
- Verify that

  $Pr[T>t] = e^{-\lambda t} \quad (t>0)$
- The corresponding density function is

  $f(t) = \lambda e^{-\lambda t}$
- The distribution of change intervals should follow an exponential distribution

Change intervals of pages



Source: Cho et al (2000)

Change Metrics (1) - Freshness

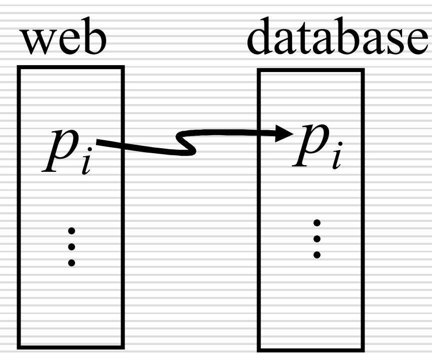
- Freshness of element e_i at time t is

$$F(p_i;t) = \begin{cases} 1 & \text{if } e_i \text{ is up-to-date at time } t \\ 0 & \text{otherwise} \end{cases}$$

Freshness of the database S at time t is

$$F(S;t) = \frac{1}{N} \sum_{i=1}^{N} F(p_i;t)$$

(Assume "equal importance" of pages)



Change Metrics (2) - Age

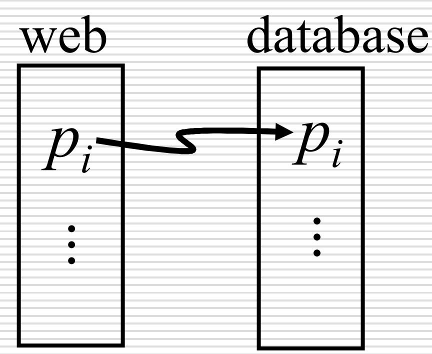
- Age of element e_i at time t is

$$A(p_i\,;\,t) = \begin{cases} 0 & \text{if } e_i \text{ is up-to-date at time } t \\ t - (\text{modification time } p_i) & \text{otherwise} \end{cases}$$

Age of the database S at time t is

$$A(S\,;\,t) = \frac{1}{N}\sum_{i=1}^{N} A(p_i\,;\,t)$$

(Assume "equal importance" of pages)

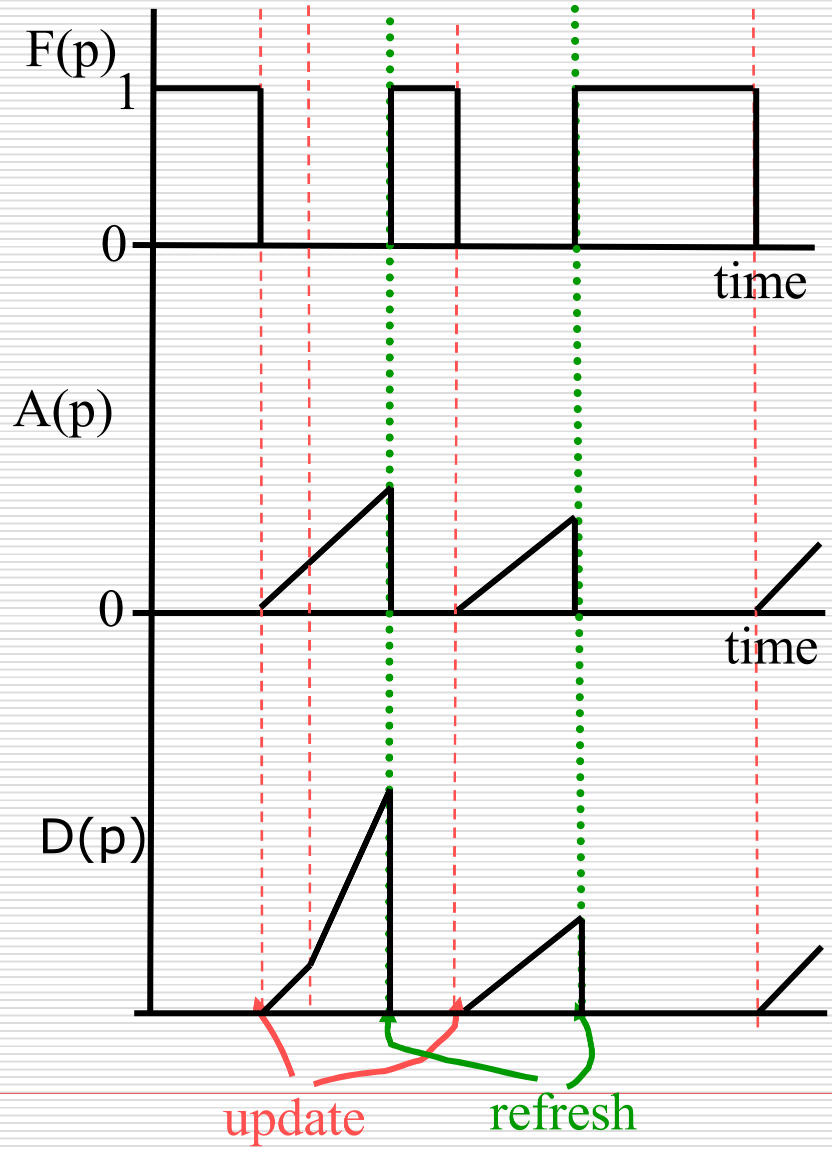


Change Metrics (3) - Delay

- Suppose crawler visits page p at times τ_0, τ_1,…
- Suppose page p is updated at times t_1, t_2,… t_k between times τ_0 and τ_1
- The delay associated with update t_i is

  $D(t_k) = \tau_1 - t_i$
- The total delay associated with the changes to page p is $D(p) = \sum_i D(t_i)$
- At time time τ_1 the delay drops back to 0
- Total crawler delay is sum of individual page delays

Comparing the change metrics



Resource optimization problem

- Crawler can fetch M pages in time T
- Suppose there are N pages $p_1,\ldots,p_N$ with change rates $\lambda_1,\ldots,\lambda_N$
- How often should the crawler visit each page to minimize delay?
- Assume uniform interval between visits to each page

A useful lemma

Lemma.

For page p with update rate λ, if the interval between refreshes is τ, the expected delay during this interval is $\lambda\tau^2/2$.

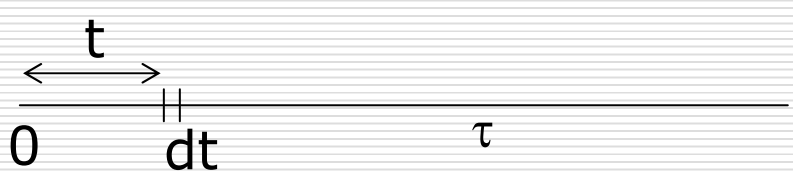
Proof.

Number of changes generated at between times t and t+dt = λ dt

Delay for these changes = τ-t

Total delay = $\int_0^{\tau} \lambda(\tau - t)\, dt$

= $\lambda\tau^2/2$



Optimum resource allocation

Total number of accesses in time T = M

Suppose allocate m_i fetches to page p_i

Then $\sum_{i=1}^{N} m_i = M$

Interval between fetches of page p_i = T/m_i

Delay for page p_i between fetches = $\frac{\lambda_i (T/m_i)^2}{2}$

Total delay for page p = $\frac{\lambda_i T^2}{2m_i}$

Minimize $f = \sum_{i=1}^{N} \frac{\lambda_i T^2}{2m_i}$

subject to $g = M - \sum_{i=1}^{N} m_i = 0$

Method of Lagrange Multipliers

To maximize or minimize a function $f(x_1,\ldots x_n)$ subject to the constraint $g(x_1,\ldots x_n)=0$

Introduce a new variable μ and define

$$h = f - \mu g$$

Solve the system of equations:

$$\frac{\partial h}{\partial x_i} = 0 \qquad \text{for } i = 1,\ldots,n$$

$$g(x_1,\ldots x_n)=0$$

n+1 equations in n+1 variables

Optimum refresh policy

Applying the Lagrange multiplier method to our problem, we have

$$h = \sum_{i=1}^{N}\left(\frac{\lambda_i T^2}{2m_i} + \mu m_i\right) - M$$

$$\frac{\partial h}{\partial m_i} = \frac{-\lambda_i T^2}{2m_i^2} + \mu m_i$$

$$m_i = \sqrt{\frac{\lambda_i T^2}{2\mu}} = k\sqrt{\lambda_i}$$

Optimum refresh policy

- To minimize delay, we must allocate to each page a number of visits proportional to the square root of its average rate of change
- Very different answer to minimze the freshness and age metrics; see references.

Estimating the rate parameter λ

- Simple estimator
  - Visit page N times in time interval T
  - Suppose we see that page has changed X times
  - Estimate $\lambda = X/T$
- What is the problem with this estimator?

A better estimator

- The page may have actually changed more than once between visits
  - We therefore tend to understimate λ
- A better estimator is $\lambda = \log(\frac{N+0.5}{N-X+0.5})$
  - For N=10, X=3, we get:
    - λ = 0.3 with the simple estimator
    - λ = 0.34 with the improved estimator.
- Details in Cho and Garcia-Molina (2000)

Detecting duplicate pages

- Duplicates waste crawler resources
- We can quickly test for duplicates by computing a fingerprint for every page (e.g., MD5 hash)
- Make fingerprint long enough to make collisions very unlikely
- Problem: pages may differ only in ads or other formatting changes
- Also an issue in counting changes for rate estimation

Detecting approximate duplicates

- Can compare the pages using known distance metrics e.g., edit distance
- Takes far too long when we have millions of pages!
- One solution: create a sketch for each page that is much smaller than the page
- Assume: we have converted page into a sequence of tokens
  - Eliminate punctuation, HTML markup, etc

Shingling

- Given document D, a w-shingle is a contiguous subsequence of w tokens
- The w-shingling S(D,w) of D, is the set of all w-shingles in D
- e.g., D=(a,rose,is,a,rose,is,a,rose)
- S(D,W) = {(a,rose,is,a),(rose,is,a,rose),(is,a,rose,is)}
- Can also define S(D,w) to be the bag of all shingles in D
  - We'll use sets in the lecture to keep it simple

Resemblance

- Let us define the resemblance of docs A and B as:

$$r_w(A, B) = \frac{|S(A,w) \cap S(B,w)|}{|S(A,w) \cup S(B,w)|}$$

- In general, $0 \leq r_w(A,B) \leq 1$
- Note $r_w(A,A) = 1$
- But $r_w(A,B)=1$ does not mean A and B are identical!
- What is a good value for w?

Sketches

- Set of all shingles is still large
- Let Ω be the set of all shingles
- Let $\pi:\Omega\rightarrow\Omega$ be a random permutation of Ω
- Assume Ω is totally ordered
- For a set of shingles W and parameter s

$$MIN_s(W) = \begin{cases} \text{set of smallest s elements of W, if } |W| \geq s \\ W, \quad \text{otherwise} \end{cases}$$

- The sketch of document A is

$$M(A) = MIN_s(\pi(S(A,w)))$$

Estimating resemblance

- Define r'(A,B) as follows:

$$r'(A,B) = \frac{|\mathsf{MIN}_s(M(A) \cup M(B)) \cap M(A) \cap M(B)|}{|\mathsf{MIN}_s(M(A) \cup M(B))|}$$

- Easy to show that r'(A,B) is a good estimate of the resemblance r_w(A,B)
- E[r'(A,B)] = r_w(A,B)

Proof

$$\begin{aligned}\mathsf{MIN}_s(M(A) \cup M(B)) &= \mathsf{MIN}_s(\pi(S(A,w)) \cup \pi(S(B,w))) \\ &= \mathsf{MIN}_s(\pi(S(A,w) \cup S(B,w)))\end{aligned}$$

Let α be the smallest element in $\pi(S(A,w) \cup S(B,w))$. Then

$$\begin{aligned}\mathbf{Pr}(\alpha \in M(A) \cap M(B)) &= \mathbf{Pr}(\pi^{-1}(\alpha) \in S(A,w) \cap S(B,w)) \\ &= \frac{|S(A,w) \cap S(B,w)|}{|S(A,w) \cup S(B,w)|} \\ &= r_w(A,B)\end{aligned}$$

We can repeat this argument for each element of $\mathsf{MIN}_s(\pi(S(A,w) \cup S(B,w)))$ to prove the result.

Implementation

- By increasing sample size (s) we can make it very unlikely r'(A,B) is significantly different from r_w(A,B)
  - 100-200 shingles is sufficient in practice
- Size of each shingle is still large
  - e.g., each shingle = 7 English words = 40-50 bytes
  - 100 shingles = 4K-5K
- Compute a fingerprint f for each shingle (e.g., Rabin fingerprint)
  - 40 bits is usually enough to keep estimates reasonably accurate
  - Fingerprint also eliminates need for random permutation

Finding all near-duplicates

- Naïve implementation makes O(N^2) sketch comparisons
  - Suppose N=100 million
- Divide-Compute-Merge (DCM)
  - Divide data into batches that fit in memory
  - Operate on individual batch and write out partial results in sorted order
  - Merge partial results

Finding all near-duplicates

1. Calculate a sketch for each document
2. For each document, write out the pairs <shingle_id, docId>
3. Sort by shingle_id (DCM)
4. In a sequential scan, generate triplets of the form <docId1,docId2,1> for pairs of docs that share a shingle (DCM)
5. Sort on <docId1,docId2> (DCM)
6. Merge the triplets with common docids to generate triplets of the form <docId1,docId2,count> (DCM)
7. Output document pairs whose resemblance exceeds the threshold

Some optimizations

- Step 4 is the most expensive
- We can speed it up eliminating very common shingles
  - Common headers, footers, etc.
  - Do it as a preprocessing step
- Also, eliminate exact duplicates up front

Mirrors

- Replication at a higher level
  - Entire collections of documents
  - Java APIs, perl manuals,...
- Use document comparison as a building block