Mining Data Streams II / Link Analysis I

Alexander Schönhuth

UNIVERSITÄT BIELEFELD
Faculty of Technology

Bielefeld University
June 10, 2020

TODAY

Overview

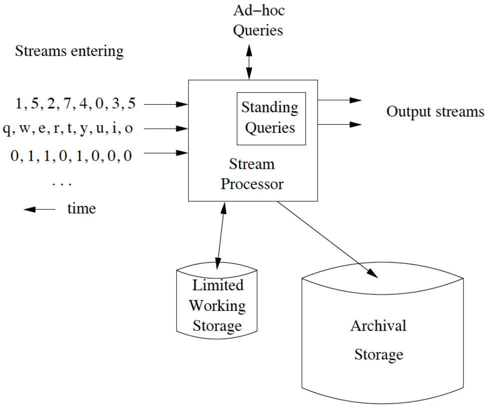
- Mining Data Streams II
  - Counting Ones in a Window: Datar-Gionis-Indyk-Motwani algorithm
- Link Analysis I
  - PageRank: Introduction, Definition
  - PageRank: Dead Ends and Spider Traps

Learning Goals: Understand these topics and get familiarized

Counting Ones in a Window

The Datar-Gionis-Indyk-Motwani Algorithm

Data Stream Management System



A data stream management system

Adopted from `mmds.org`

Data Stream Queries

Issues

- Streams deliver elements rapidly: need to act quickly
- Thus, data to work on should fit in main memory
- New techniques required:

☞ Compute approximate, not exact answers

☞ Hashing is a useful technique

Counting Ones in Window: Problem

- *Situation:*
  - Suppose we have a window of length N on a binary stream
  - Query: "how many ones are there in the last $k \leq N$ bits?"
  - We cannot afford to store entire window
  - Approximate algorithms required
- Present solution for binary streams first
- Discuss extension for summing numbers (from a stream of numbers) thereafter

The Cost of Exact Counts

- One needs to store N bits to answer count-one-queries for arbitrary $k \leq N$:
  - Assume one could use less than N bits
  - We need 2^N different representations to represent all possible 2^N bit strings of length N
  - Since we use less than N bits, there are two different bit strings $w \neq x$, for which we use the same representation
  - Let k be the first bit from the right where w and x disagree
  - *Example:*
    - For $w = 0101, x = 1010$, we have $k = 1$
    - For $w = 1001, x = 0101$, we have $k = 3$
  - So the counts of ones in the window of length k for w and x differ
  - But because we use identical representations for w and x, we will output the same count
  - This proves that one needs the full N bits to represent bit strings for exact count-one-queries.

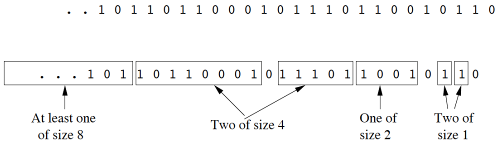
THE DATAR-GIONIS-INDYK-MOTWANI ALGORITHM

- *Situation:*
  - We consider a binary stream: elements are *bits*
  - Let each element of the stream have a *timestamp*
  - The first, *leftmost* element has timestamp 1, the second leftmost has timestamp 2, and so on
- *Goal:* We like to count the ones among the N most recent (rightmost) elements/bits
- *Space requirements:*
  - Storing timestamps modulo N, and
  - marking rightmost timestamp as most recent
  - allows to store positions of individual bits using $\log_2 N$ bits

THE DATAR-GIONIS-INDYK-MOTWANI ALGORITHM

- *Algorithm:* Divide window into *buckets*, contiguous bit substrings
- *Bucket Representation:* For identifying buckets, we store
  - The timestamp of its right end, and
  - The *size* of the bucket, as the number of 1's in the bucket
  - The size is supposed to be a power of 2
- *Bucket Space Requirements:*
  - Timestamp requires $\log_2 N$ bits
  - Size is 2^j, hence requires $\log\log_2 N$ bits (by storing $\log_2 j$ bits)
  - Requires $O(\log N)$ bits overall

DATAR-GIONIS-INDYK-MOTWANI RULES



Bit stream divided into buckets following DGIM rules

From `mmds.org`

- Right end always is a 1
- Every 1 of the window is in some bucket
- Buckets do not overlap
- All sizes must be a power of 2
- For each possible size, there are either one or two buckets
- Size of buckets cannot decrease when moving

THE DATAR-GIONIS-INDYK-MOTWANI ALGORITHM

Key Ideas / Considerations

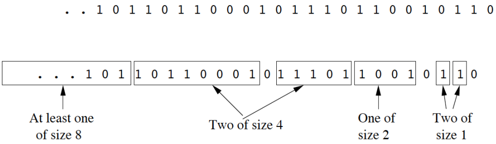
- The number of buckets representing a window must be small
- Estimate the number of 1's in the last k bits (for any k) with an error of no more than 50%
- How to maintain the DGIM Bucket Rules on new bits arriving?

The Datar-Gionis-Indyk-Motwani Algorithm

Storage Requirements

- Each bucket can be represented using $O(\log N)$ bits
- Let 2^j be size of largest bucket: $2^j < N$ implies $j \leq \log_2 N$
- So there are at most 2 buckets of sizes $2^j, j = \log_2 N, ..., 1$
- This means that there are $O(\log N)$ buckets
- Each bucket being represented by $O(\log N)$ bits requires $O(\log^2 N)$ space overall

The Datar-Gionis-Indyk-Motwani Algorithm



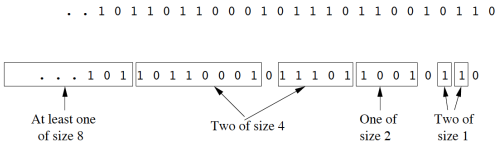
Bit stream divided into buckets following DGIM rules

From `mmds.org`

Answering Queries

- Let $1 \leq k \leq N$: how many 1's are among the last k bits?
- *Answer:*
  - Find leftmost (= with earliest timestamp) bucket b containing some of last k bits
  - *Estimate:* Sum of sizes of buckets right of b plus half the size of b

THE DATAR-GIONIS-INDYK-MOTWANI ALGORITHM



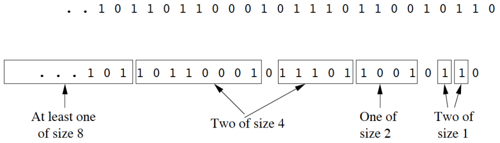
Bit stream divided into buckets following DGIM rules

From `mmds.org`

Example

- Let $k = 10$: how many 1's are among 0110010110?
- Let t be timestamp of rightmost bit
- Two buckets with one 1 each, having timestamps $t-1, t-2$ are fully included in k righmost bits
- Bucket of size 2 with timestamp $t-4$ is also included
- Bucket of size 4 with timestamp $t-8$ is only partially included
- Estimate: $1+1+2+(1/2 \times 4) = 6$, one more than true count

DGIM: Error of Estimate



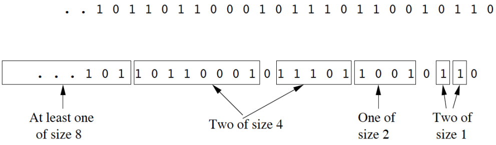
Bit stream divided into buckets following DGIM rules

From mmds.org

Case 1: estimate is less than c

- Let c be true count; let leftmost bucket b be of size 2^j
- *Worst case:* all 1's in b are among k most recent bits
- So, estimate is lower by $1/2 \times 2^j = 2^{j-1}$ than c
- Because $c \geq 2^j$, error is at most half of c

DGIM: Error of Estimate



Bit stream divided into buckets following DGIM rules

From `mmds.org`

Case 2: estimate is larger than c

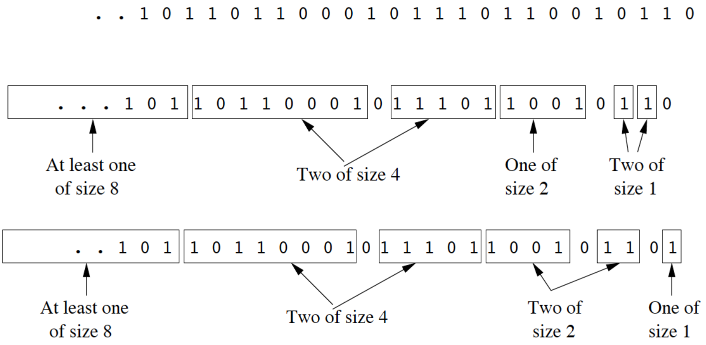
- Let c be true count; let leftmost bucket b be of size 2^j
- *Worst case:* only rightmost bit of b is among k most recent bits, and
- There is only one bucket for each of sizes $2^{j-1}, ..., 1$
- That yields $c = 1 + 2^{j-1} + ... + 1 = 1 + 2^j - 1 = 2^j$
- Estimate is $2^{j-1} + 2^{j-1} + ... + 1 = 2^{j-1} + 2^j - 1$, so
- Error $\frac{2^{j-1}+2^j-1}{2^j}$ is no greater than 50% of true count

Maintaining DGIM Rules

Upon a new bit with timestamp t having arrived:

- Check timestamp s of leftmost bucket b:
  - if $s \leq t - N$, drop b from list of buckets
- If the new bit is 0, do nothing
- If the new bit is 1, do
  - Create new bucket with timestamp t and size 1
  - On increasing size, starting with size 1, while there are three buckets of the same size, do
    - keep the rightmost bucket of that size as is
    - join the two left buckets into one of double the size
    - where the timestamp is that of the rightmost bit
  - *For example:* joining the two left of the three buckets of size 1 into a bucket of size 2 may create a third bucket of size 2, and so on
- *Runtime:* Need to look at $O(\log N)$ buckets, joining is constant time, so processing new bit requires $O(\log N)$ time overall

The Datar-Gionis-Indyk-Motwani Algorithm
Part VI



Bit stream divided into buckets following DGIM rules (top), with new 1 arriving (bottom)

From `mmds.org`

DGIM Algorithm: Reducing the Error

- For some $r > 2$, allow either r or $r-1$ buckets of the same size
- Allow this for all but size 1 and largest size, whose numbers may be any of $1, \ldots, r$
- Compute estimate as before
- Extend maintaining the DGIM Bucket Rules in the obvious way
- *Recall:* largest error $\frac{2^{j-1}+2^j-1}{2^j}$ was made when only one 1 from leftmost bucket b was within window
- *New error:*
  - True count is at most $1+(r-1)(2^{j-1}+\ldots+1) = 1+(r-1)(2^j-1)$
  - Estimate is $2^{j-1}-1+(r-1)(2^j-1)$, so fractional error is

    $$\frac{2^{j-1}-1}{1+(r-1)(2^j-1)}$$

    which is upper bounded by $1/2(r-1)$
  - Picking large r can limit error to any $\epsilon > 0$

DGIM Algorithm: Extensions

- DGIM can be extended to integers instead of bits
- Question is to estimate the sum of last $k \leq N$ integers from a window of N integers overall
- However, DGIM cannot be extended to streams containing negative integers
- Consider case of integers in range of 1 to 2^m, so represented by m bits
- *Solution:*
  - Treat each bit of integers as separate stream
  - Apply DGIM algorithm to each of m streams, yielding estimate c_i for i-th stream
  - *Overall estimate:*

$$\sum_{i=0}^{m-1} c_i 2^i$$

  - If error is at most ϵ for all i, overall error is also at most ϵ

PageRank
Introduction

PageRank: Overview

- Motivation of PageRank definition: history of search engines
- Concept of *random surfers* foundation of PageRank's effectiveness
- *Taxation* ("recycling of random surfers") allows to deal with problematic web structures

History: Early Search Engines

- *Early search engines*
  - Crawl the (entire) web
  - List all terms encountered in an *inverted index*
  - An inverted index is a data structure that, given a term, provides pointers to all places where term occurs
- On a *search query* (a list of terms)
  - pages with those terms are extracted from the index
  - ranked according to use of terms within pages
  - E.g. the term appearing in the header renders page more important
  - or the term appearing very often

Term Spam

- *Spammers* exploited this to their advantage
- *Simple strategy:*
  - Add terms thousands of times to own webpages
  - Terms can be made hidden by using background color
  - So pages are listed in searches that do not relate to page contents
  - Example: add term “movie” 1000 times to page that advertizes shirts
- *Alternative strategy:*
  - Carry out web search on term
  - Copy-paste highest ranked page into own page
  - Upon new search on term, own page will be listed high up
- Corresponding techniques are referred to as *term spam*

PageRank's Motivation: Fighting Term Spam

Idea:

- Simulate *random web surfers*
  - They start at random pages
  - They randomly follow web links leaving the page
  - Iterate this procedure sufficiently many times
  - Eventually, they gather at "important" pages
- Judge page also by *contents of surrounding pages*
  - Difficult to add terms to pages not owned by spammer

PageRank's Motivation: Fighting Term Spam

Justification

- Ranking web pages by number of in-links does not work
  - Spammers create "spam farms" of dummy pages all linking to one page
- *But*, spammers' pages do not have in-links from elsewhere

☞ Random surfers do not wind up at spammers' pages

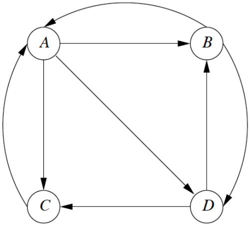
- (Non-spammer) page owners place links to pages they find helpful
- Random surfers indicate which pages are likely to visit
  ☞ Users are more likely to visit useful pages

PageRank: Definition

- PageRank is a function that assigns a real number to each (accessible) web page
- *Intuition:* The higher the PageRank, the more important the page
- There is not one fixed algorithm for computing PageRank
- There are many variations, each of which caters to particular issue

PageRank: Definition

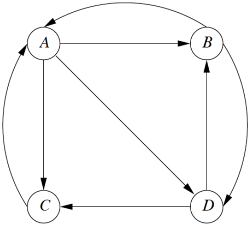
- Consider the web as a directed graph
  - Nodes are web pages
  - Directed edges are links leaving from and leading to web pages



Hypothetical web with four pages

Adopted from `mmds.org`

PageRank: Definition



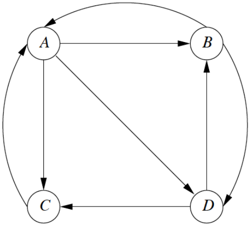
Random walking a web with four pages

Adopted from `mmds.org`

- For example, a *random surfer* starts at node A
- Walks to B, C, D each with probability $1/3$
- So has probability 0 to be at A after first step

UNIVERSITÄT BIELEFELD

PageRank: Definition



Random walking a web with four pages

Adopted from `mmds.org`

- *Random surfer* at B, for example, in next step
  - is at A, D each with probability $1/2$
  - is at B, C with probability 0

WEB TRANSITION MATRIX: DEFINITION

DEFINITION [WEB TRANSITION MATRIX]:

- Let n be the number of pages in the web
- The *transition matrix* $M = (m_{ij})_{1 \leq i,j \leq n} \in \mathbb{R}^{n \times n}$ has n rows and columns
- For each $(i, j) \in \{1, ..., n\} \times \{1, ..., n\}$
  - $m_{ij} = 1/k$, if page j has k arcs out, of which one leads to page i
  - $m_{ij} = 0$ otherwise

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

Transition matrix for web from slides before

Adopted from `mmds.org`

PageRank Function: Definition

Definition [PageRank Function]:

- Let n be the number of pages in the web
- Let $p_i^t, i = 1, ..., n$ be the probability that the random surfer is at page i after t steps
- The *PageRank function* for $t \geq 0$ is defined to be the vector

$$p^t = (p_1^t, p_2^t, ..., p_n^t) \in [0, 1]^n$$

PageRank Function: Interpretation

- Usually, $p^0 = (1/n, ...1/n)$ for each $i = 1, ..., n$
- So before the first iteration, the random surfer is at each page with equal probability
- The probability to be at page i in step $t + 1$ is the sum of probabilities to be at page j in step t times the probability to move from page j to i
- That is, $p_i^{t+1} = \sum_{j=1}^{n} m_{ij} p_j^t$ for all i, t, or, in other words

$$p^{t+1} = Mp^t \quad \text{for all } t \geq 0 \tag{1}$$

- So, applying the web transition matrix to a PageRank function yields another one

PageRank Function: Markov Processes

$$\boxed{p^{t+1} = Mp^t \quad \text{for all } t \geq 0}$$

- This relates to the theory of *Markov processes*
- Given that the web graph is *strongly connected*
  - That is: one can reach any node from any other node
  - In particular, there are no *dead ends*, nodes with no arcs out
- it is known that the surfer reaches a *limiting distribution* $\bar{p}$, characterized by

$$M\bar{p} = \bar{p} \tag{2}$$

PageRank Function: Markov Processes

$$\boxed{M\bar{p} = \bar{p}}$$

- Further, because M is *stochastic* (= columns each add up to one)
  - $\bar{p}$ is the *principal eigenvector*, which is
  - the eigenvector associated with the largest eigenvalue, which is one
- $\bar{p}_i$ is the probability that the surfer is at page i after a long time
- Principal eigenvector of M expresses where the surfer will end up
- *Reasoning:* The greater $\bar{p}_i$, the more important page i

PageRank Function: Computation

$$\boxed{M\bar{p} = \bar{p}}$$

- It holds that

$$M^t p^0 \underset{t\to\infty}{\longrightarrow} \bar{p} \tag{3}$$

- So, for *computing* $\bar{p}$, apply iterative matrix-vector multiplication until (approximate) convergence
- *Example:* Iterative application of transition matrix from above

$$\begin{bmatrix} 1/4 \\ 1/4 \\ 1/4 \\ 1/4 \end{bmatrix}, \begin{bmatrix} 9/24 \\ 5/24 \\ 5/24 \\ 5/24 \end{bmatrix}, \begin{bmatrix} 15/48 \\ 11/48 \\ 11/48 \\ 11/48 \end{bmatrix}, \begin{bmatrix} 11/32 \\ 7/32 \\ 7/32 \\ 7/32 \end{bmatrix}, \dots, \begin{bmatrix} 3/9 \\ 2/9 \\ 2/9 \\ 2/9 \end{bmatrix}$$

Convergence to limiting distribution for four-node web graph

Adopted from `mmds.org`

PageRank Function: Computation

$$\boxed{M\bar{p} = \bar{p}}$$

- It holds that
$$M^t p_0 \quad \underset{t \to \infty}{\longrightarrow} \quad \bar{p} \tag{4}$$
- So, for *computing* $\bar{p}$, apply iterative matrix-vector multiplication until (approximate) convergence
- In practice, working real web graphs
  - 50-75 iterations do just fine
  - For *efficient computation*, recall MapReduce based matrix-vector multiplication techniques

Materials / Outlook

- See *Mining of Massive Datasets*, chapter 4.6; 5.1
- As usual, see `http://www.mmds.org/` in general for further resources
- Next lecture: “Link Analysis II”
  - See *Mining of Massive Datasets* 5.2–5.5