Lecture #29: Approximate Counting

C. Faloutsos

15-826 Multimedia Databases and Data Mining

Must-read material

• Christopher Palmer, Phillip B. Gibbons and Christos Faloutsos,
  ANF: A Fast and Scalable Tool for Data Mining in
  Massive Graphs, KDD 2002

• Efficient and Tunable Similar Set Retrieval, by Aristides Gionis,
  Dimitrios Gunopulos and Nikos Koudas,

• New sampling-based summary statistics for improving
  approximate query answers, by Phillip B. Gibbons and

Outline

Goal: ‘Find similar / interesting things’

• Intro to DB
• Indexing - similarity search
• Data Mining
  – ...
  – Association Rules
  – Approximate Counting
Outline

- Flajolet-Martin (and Cohen) – vocabulary size (Problem #1)
- Application: Approximate Neighborhood function (ANF)
- other, powerful approximate counting tools (Problem #2, #3)

Problem #1

- Given a multiset (eg., words in a document)
- find the vocabulary size (#, after dup. elimination)

A A B A B A C A B

Voc. Size = 3 = |{A, B, C}|
Problem #2

- Given a multiset
- compute approximate high-end histogram = hot-list query = \( (k \text{ most common words, and their counts}) \)

\[
\begin{align*}
A & \quad A & \quad A & \quad B & \quad A & \quad C & \quad A & \quad B & \quad D & \quad D & \quad D & \quad D & \quad D \\
\end{align*}
\]

(for \(k = 2:\)
- \(A\# = 6\)
- \(D\# = 5\)

Problem #3

- Given two documents
- compute quickly their similarity (\#common words/ \#total-words) \(\equiv\) Jaccard coefficient

Problem #1

- Given a multiset (eg., words in a document)
- find the vocabulary size \(V \equiv \# \text{, after dup. elimination)\}
- using space \(O(V), \text{ or } O(\log(V))\)

(Q1: Applications?)
(Q2: How would you solve it?)
Basic idea (Cohen)

large bit string, initially all zeros

A
A
C

hash!
Basic idea (Cohen)

large bit string

A
A
C

the rightmost position depends on the vocabulary size (and so does the left-most)

Repeat, with several hashing functions, and merge the estimates

Can we do it in less space??
Basic idea (Cohen)

large bit string

the rightmost position depends on the vocabulary size (and so does the left-most)

Can we do it in less space??
YES

How?

Basic idea (Flajolet-Martin)

O(log(V)) bit string (V: voc. size)

first bit: with prob. $\frac{1}{2}$
second: with prob. $\frac{1}{4}$
... 
i-th: with prob. $\frac{1}{2^i}$
Basic idea (Flajolet-Martin)

O(log(V)) bit string (V: voc. size)

again, the rightmost bit 'reveals' the vocabulary size

Eg.: V=4, will probably set the 2nd bit, etc

Flajolet-Martin

• Hash multiple values of X to same signature
  – Hash each x to a bit, using exponential distr.
  – ½ map to bit 0, ¼ map to bit 1, …
• Do several different mappings and average
  – Gives better accuracy
  – Estimate is: \(2^b / .77351 / BIAS\)
    • \(b\) ~ rightmost '1', and actually:
Flajolet-Martin

- Hash multiple values of \( X \) to same signature
  - Hash each \( x \) to a bit, using exponential distr.
  - \( \frac{1}{2} \) map to bit 0, \( \frac{1}{4} \) map to bit 1, ...
- Do several different mappings and average
  - Gives better accuracy
  - Estimate is: \( 2^b / .77351 / \text{BIAS} \)
    - \( b \): average least zero bit in the bitmask
    - bias: \( 1+.31/k \) for \( k \) different mappings
- Flajolet & Martin prove this works

FM Approx. Counting Alg.

Assume \( X = \{0, 1, ..., V-1\} \)
FOR \( i = 1 \) to \( k \) DO bitmask[\( i \)] = 0000...00
Create \( k \) random hash functions, \( \text{hash}_i \)
FOR each element \( x \) of \( M \) DO
  FOR \( i = 1 \) to \( k \) DO
  \( h = \text{hash}_i(x) \)
  bitmask[\( i \)] = bitmask[\( i \)] LOR \( h \)
Estimate: \( b = \) average least zero bit in bitmask[\( i \)]
\( 2^b / .77351 / (1+.31/k) \)

- How many bits? \( \log V + \) small constant
- What hash functions?

Random Hash Functions

- Can use linear hash functions. Pick random \((a_i, b_i)\) and then the hash function is:
  - \( lhash_i(x) = a_i \cdot x + b_i \)
- Gives uniform distribution over the bits
- To make this exponential, define
  - \( hash_i(x) = \) least zero bit in \( lhash_i(x) \)
- Hash functions easy to create and fast to use
Conclusions

• Want to measure # of distinct elements
• Approach #1: (Flajolet-Martin)
  – Map elements to random bits
  – Keep bitmask of bits
  – Estimate is $O(2^b)$ for least zero-bit $b$
• Approach #2: (Cohen)
  – Create random permutation of elements
  – Keep least element seen
  – Estimate is: $O(1/le)$ for least rank $le$

Approximate counting

• Flajolet-Martin (and Cohen) – vocabulary size
• Application: Approximate Neighborhood function (ANF)
• other, powerful approximate counting tools

Fast Approximation of the “neighborhood” Function for Massive Graphs

Christopher R. Palmer
Phillip B. Gibbons
Christos Faloutsos

KDD 2001
**Motivation**

- What is the diameter of the Web?
- What is the effective diameter of the Web?
- Are the telephone caller-callee graphs for the U.S. similar to the ones in Europe?
- Is the citation graph for physics different from the one for computer science?
- Are users in India further away from the core of the Internet than those in the U.S.?

---

**Proposed Tool: neighborhood**

Given graph $G=(V,E)$

$N(h)$ = # pairs within $h$ hops or less

= neighborhood function

$N(u,h)$ = # neighbors of node $u$, within $h$ hops or less
Example of neighborhood details

Example of neighborhood ~diameter of graph
details

Requirements (for massive graphs)

• Error guarantees
• Fast: (and must scale linearly with graph)
• Low storage requirements: massive graphs!
• Adapts to available memory
• Sequential scans of the edges
• Also estimates individual neighborhood functions |S(u,h)|
  – These are actually quite useful for mining
How would you compute it?

- Repeated matrix multiply
  - Too slow $O(n^{2.38})$ at the very least
  - Too much memory $O(n^2)$
- Breadth-first search
  FOR each node $u$ DO
    bf-search to compute $S(u,h)$ for each $h$
  - Best known exact solution!
  - We will use this as a reference
- Approximations? Only 1 that we know of which we will discuss when we evaluate it.

Intuition

- Guess what we'll use?
  - Approximate Counting!
- Use very simple algorithm:
  FOR each node $u$ DO $S(u,0) = \{(u,u)\}$ initialize to self-only
  FOR $h = 1$ to diameter of $G$ DO
    FOR each node $u$ DO $S(u,h) = S(u,h-1)$ can reach same things
    FOR each edge $(u,v)$ in $G$ DO
      $S(u,h) = S(u,h) \cup \{(u,v') : (v,v') \in S(v,h-1)\}$ add one more step

Intuition

- Guess what we'll use?
  - Approximate Counting!
- Use very simple algorithm:
  FOR each node $u$ DO $S(u,0) = \{(u,u)\}$ initialize to self-only
  FOR $h = 1$ to diameter of $G$ DO
    FOR each node $u$ DO $S(u,h) = S(u,h-1)$ can reach same things
    FOR each edge $(u,v)$ in $G$ DO
      $S(u,h) = S(u,h) \cup \{(u,v') : (v,v') \in S(v,h-1)\}$ add one more step

  # (distinct) neighbors of $u$, within $h$ hops

  # (distinct) neighbors of $v$, within $h-1$ hops
Trace

\( h=0 \)

\{ (1,1) \}
\{ (2,2) \}
\{ (3,3) \}
\{ (4,4) \}

---

Trace

\( h=0 \quad h=1 \)

\{ (1,1) \} \quad \{ (1,1) \}
\{ (2,2) \} \quad \{ (2,2) \}
\{ (3,3) \} \quad \{ (3,3) \}
\{ (4,4) \} \quad \{ (4,4) \}

---

Trace

\( h=0 \quad h=1 \)

\{ (1,1) \} \quad \{ (1,1) \}
\{ (2,2) \} \quad \{ (2,2) \}
\{ (3,3) \} \quad \{ (3,3) \}
\{ (4,4) \} \quad \{ (4,4) \}
Trace

h=0    h=1

\{(1,1)\} \quad \{(1,1), (1,2)\}
\{(2,2)\} \quad \{(2,2)\}
\{(3,3)\} \quad \{(3,3)\}
\{(4,4)\} \quad \{(4,4)\}

Trace

h=0    h=1

\{(1,1)\} \quad \{(1,1), (1,2), (1,3)\}
\{(2,2)\} \quad \{(2,2)\}
\{(3,3)\} \quad \{(3,3)\}
\{(4,4)\} \quad \{(4,4)\}

Trace

h=0    h=1

\{(1,1)\} \quad \{(1,1), (1,2), (1,3)\}
\{(2,2)\} \quad \{(2,2), (2,1), (2,3)\}
\{(3,3)\} \quad \{(3,3), (3,1), (3,2), (3,4)\}
\{(4,4)\} \quad \{(4,4), (4,3)\}
• Guess what we’ll use?
  – Approximate Counting!
• Use very simple algorithm:
  FOR each node u DO \( S(u,0) = \{(u,u)\} \)
  FOR each node u DO \( S(u,h) = S(u,h-1) \)
  FOR each edge \((u,v)\) in \(G\) DO
    \( S(u,h) = S(u,h) \cup \{(u,v') : (v,v') \in S(v,h-1)\} \)

Intuition

# (distinct) neighbors of u, within h hops
initialize to self-only

For reach same things
and add one more step

# (distinct) neighbors of u, within h hops
initialize to self-only

Too slow and requires too much memory

Replace expensive set ops with bit ops

ANF Algorithm #1

FOR each node, \(u\), DO
  \(M(u,0) = \text{concatenation of } k \text{ bitmasks of length } \log n + r \)
  each bitmask has 1 bit set (exp. distribution)
DONE

FOR \(h = 1 \text{ to diameter of } G\) DO
  FOR each node, \(u, \text{ DO } M(u,h) = M(u,h-1)\)
  FOR each edge \((u,v)\) in \(G\) DO
    \( M(u,h) = (M(u,h) \text{ OR } M(v,h-1)) \)

Estimate \(N(h) = \text{Sum}(N(u,h)) = \text{Sum} 2^{b(u)} .77351 / (1 + 34/h) \)
  where \(b(u) = \text{average least zero bit in } M(u,0)\)
DONE
ANF Algorithm #1

FOR each node, \( u \), DO
\[ M(u,0) = \text{concatenation of } k \text{ bitmasks of length } \log n + r \]
each bitmask has 1 bit set (exp. distribution)
DONE

FOR \( h = 1 \) to diameter of \( G \) DO
FOR each node, \( u \), DO \( M(u,h) = M(u,h-1) \)
FOR each edge \((u,v)\) in \( G \) DO
\[ M(u,h) = (M(u,h) \text{ OR } M(v,h-1)) \]
Estimate \( N(h) = \sum N(u,h) = \sum 2^b(u) / .77351 / (1+.31/k) \)
where \( b(u) = \text{average least zero bit in } M(u,h) \)
DONE

ANF Algorithm #1

\[ u \quad v \]
\[ \text{whatever } u \text{ can reach with } h \text{ hops} \]
\[ \text{plus whatever } v \text{ can reach with } h-1 \text{ hops} \]
\[ M(u,h) = (M(u,h) \text{ OR } M(v,h-1)) \]
Duplicates: automatically eliminated!

Properties

• Has error guarantees: (from F&M)
• Is fast: \( O((n+m)d) \) for \( n \) nodes, \( m \) edges, diameter \( d \) (which is typically small)
• Has low storage requirements: \( O(n) \)
• Easily parallelizable: Partition nodes among processors, communicate after full iteration
• Does sequential scans of edges.
• Estimates individual neighborhood functions
• DOES NOT work with limited memory
Conclusions

- Approximate counting (ANF / Martin-Flajolet) take minutes, instead of hours
- and discover interesting facts quickly

Outline

- Flajolet-Martin (and Cohen) – vocabulary size (Problem #1)
- Application: Approximate Neighborhood function (ANF)
- other, powerful approximate counting tools (Problem #2, #3)

Problem #2

- Given a multiset
- compute approximate high-end histogram = hot-list query = (k most common words, and their counts)

\[ \text{AAABABACBDDDD} \]

(for \( k = 2 \):
- A#: 6
- D#: 5)
**Hot-list queries**

- Given a stream of product ids (with duplicates)
- Compute
  - the $k$ most frequent products,
  - and their counts
- with a SINGLE PASS and $O(k)$ memory

\[ A A B A C A B C A A D E A C A \]

$k=2$

\[ A C \]

**Applications?**

- Best selling products
- most common words
- most busy IP destinations/sources (DoS attacks)
- summarization / synopses of datasets
- high-end histograms for DBMS query optimization
Hot-list queries

• Given a stream of product ids (with duplicates)
• Compute
  • the $k$ most frequent products,
  • and their counts
• with a SINGLE PASS and $O(k)$ memory

A A B A C A B C A A D E A C A

Exact: impossible
Thus: approximate

$k=2$

A C

8 3

Hot-list queries - idea

• Keep the (approx.) $k$ best so far, plus counts
• for a new item, if it is in the hot list
  – increment its count

A A B A C A B C A A D E A C A

$k=2$

A B

2 1

Hot-list queries - idea

• Keep the (approx.) $k$ best so far, plus counts
• for a new item, if it is in the hot list
  – increment its count

A A B A C A B C A A D E A C A

$k=2$

A B

3 1
Hot-list queries - idea

- Keep the (approx.) \( k \) best so far, plus counts
- for a new item, if it is in the hot list
  - increment its count
  - else ??

A A B A C A B C A A D E A C A

\[ k=2 \]

A B

3 1

Biased coin - what are the Head/Tail prob.?  

A A B A C A B C A A D E A C A

\[ k=2 \]

A B

6 2

3 1
Hot-list queries - idea

- Biased coin - what are the Head/Tail prob.?
- A: depends on count(weakest)

A A B A C B C A A D E A C A

\[ k=2 \]

A B

2

6

Hot-list queries - idea

- Biased coin - what are the Head/Tail prob.?
- A: depends on count(weakest)
- and the new item ('D'), if it wins, it gets the count of the item it displaced.

See [Gibbons+Matias 98] for proofs
Outline

- Flajolet-Martin (and Cohen) – vocabulary size (Problem #1)
- Application: Approximate Neighborhood function (ANF)
- Other, powerful approximate counting tools
  - Problem #2,
  - Problem #3

Problem #3

- Given two documents
- Compute quickly their similarity (#common words/ #total-words) == Jaccard coefficient

Problem #3’

- Given a query document q
- And many other documents
- Compute quickly the k nearest neighbors of q, using the Jaccard coefficient

\[
D1: \{A, B, C\} \quad q: \{A, C, D, W\} \\
D2: \{A, D, F, G\} \\
\ldots
\]
Applications?

- Set comparisons e.g.,
  - snail-mail address (set of trigrams)
- search engines - ‘similar pages’
- social networks: people with many joint friends (facebook recommendations)

Problem #3’

- Given a query document $q$
- and many other documents
- compute quickly the $k$ nearest neighbors of $q$, using the Jaccard coefficient

Q: how to extract a fixed set of numerical features, to index on?
Answer

- Approximation / hashing - Cohen:

Basic idea (Cohen)

- Large bit string
- For each document and for a given h.f. return the position of first ‘1’
- Repeat for k h.f. -> each document becomes k numbers

Idea

- Doc1: n1, n2, …, nk
- Doc2: n1’, n2’, …, nk’
**Idea**

- Doc1: $n_1, n_2, ..., n_k$
- Doc2: $n'_1, n'_2, ..., n'_k$
- say they agree on $m$ values

**Intuition behind proof**

- Venn diagram
  
  - voc. terms of Doc.#1
  - voc. terms of Doc.#2

Andrew Tomkins
Intuition behind proof

- Venn diagram

\[ \text{voc. terms of Doc.#1} \quad \text{voc. terms of Doc.#2} \]

Intuition behind proof

- Venn diagram - let \( w \) be the voc. word with the overall smallest hash value, for h.f.#1

\[ \text{voc. terms of Doc.#1} \quad \text{voc. terms of Doc.#2} \]

Intuition behind proof

- Prob. that \( w \) is smallest on both is exactly Jaccard: \#common / \#union

\[ \text{voc. terms of Doc.#1} \quad \text{voc. terms of Doc.#2} \]
Conclusions

• Approximations can achieve the impossible!
• MF and ANF for neighborhood function
• hot-lists
• Jaccard coeff. / ‘similar pages’

References


References (cont’d)

Aristides Gionis, Dimitrios Gunopulos, Nikos Koudas, Efficient and Tunable Similar Set Retrieval, ACM SIGMOD 2001, Santa Barbara, California

References (cont’d)


References (cont’d)