Tracking Frequent Items Dynamically:

”What’s Hot and What’s Not”

Graham Cormode
graham@dimacs.rutgers.edu
dimacs.rutgers.edu/~graham

S. Muthukrishnan
muthu@cs.rutgers.edu

Outline

- **Problem definition and lower bounds**
- Finding Heavy Hitters via Group Testing
  - Finding a simple majority
  - Non-adaptive Group Testing
  - Experimental Evaluation
- Extensions and Conclusions

Motivating Problems

- DBMSs need to track attribute values that occur frequently in a column for query plan optimization, approximate query answering.
- Network managers want to know users using large quantities of bandwidth as connections are set up and torn down, for charging, tuning, detecting problems or abuse.
- Many other problems can be modeled as tracking frequent items in a dynamic setting.

Scenario

- Data arrives as sequence of updates: inserts and deletes in Database, SYN and ACK in networks, start and end call in telecoms
- Model state as an (implicit) vector a[1..n]
- On insert of i, add 1 to a[i], on delete of i decrement a[i]
- Only interested in "hot" entries $a[i] > \phi \|a\|_1$
- Easy for a small enough domain: challenge is from large domains: eg IP addresses $n = 2^{32}$

Previous Work

Many solutions for insertions only, old and new:

- In Algorithms: Boyer, Moore 82, Misra, Gries 82, Demaine, LopezOrtiz, Munro 02, Charikar, Chen, Farach-Colton 02
- In Databases: Fang, Shivakumar, Garcia-Molina, Motwani, Ullman 98, Manku, Motwani 02, Karp, Papadimitriou, Shenker 03
- In Networks: Estan, Varghese 02

…but (almost) nothing with deletions

Difficulty of Deletions

- Suppose we keep some currently hot items and their counts: these could all get deleted next.
- Need to recover newly hot items. Eg $\phi = 0.2$, from millions of items, all but 4 are deleted – need to find these four.
- Can't backtrack on the past without explicitly storing the whole sequence: backing sample will help, but not much...

Our solutions

- Escape lower bounds using probability and approximation.
- Our solution is based on (non-adaptive) Group Testing
- Some prior work did this kind of thing, but requires heavy duty sketches, large poly in log n time and space (eg top wavelet coefficients [Gilbert Guha Indyk Kotidis Muthukrishnan Strauss 02])

Outline

- Problem definition and lower bounds
- Finding Heavy Hitters via Group Testing
  - **Finding a simple majority**
  - Non-adaptive Group Testing
  - Experimental Evaluation
- Extensions and Conclusions

Non-adaptive Group Testing

Special case: $\phi = ½$. At most 1 item $a[i] > ½ \|a\|_1$

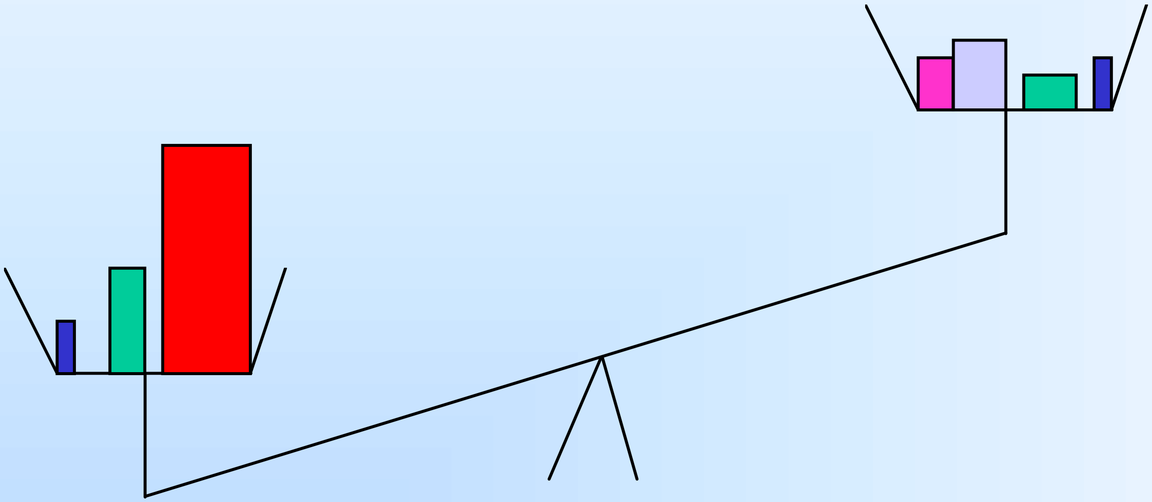
Assume there is such an item when we query, how to find it?

Formulate as a group testing problem.

Arrange items 1..n into (overlapping) groups, keep counts: every time an item from a group arrives, increment group's count, decrement for departures. Also keep count of all items.

Test: Is the count of the group $> ½ \|a\|_1$?

Weighing up the odds



If there is an item with weighing over half the total weight, it will always be in the heavier pan...

Log Groups

- Keep log n groups, one for each bit position
- If j'th bit of i is 1, put item i is group j
- Can read off index of majority item
- log n bits clearly necessary, get 1 bit from each counter comparison.
- Order of insertions and deletions doesn't matter, since addition/subtraction commute

Outline

- Problem definition and lower bounds
- Finding Heavy Hitters via Group Testing
  - Finding a simple majority
  - **Non-adaptive Group Testing**
  - Experimental Evaluation
- Extensions and Conclusions

Group Testing

Want to extend this approach to arbitrary ϕ
– want to find up to $k = 1/\phi$ items

Need a construction of groups so can use "weight" tests to find hot items.

There are deterministic group constructions which use superimposed codes of order k

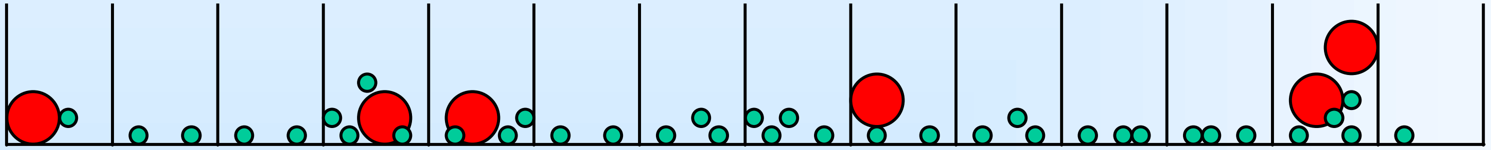
These are too costly to decode: need to consider n codewords, and n is large

Randomized Construction

- Use randomized group construction (with limited randomness)
- Idea: generate groups randomly which have at most 1 hot item in whp
- If one hot item and little else in a group, then it is majority, use majority method to find it.
- Need to reason about false positives (reporting infrequent items) and false negatives (missing hot items)

Multiple Buckets

Multiple buckets spread the weight out:



- Hot items are unlikely to collide
- Isn't too much weight from other items

So, there's a good chance that each hot item will be in the majority for its bucket

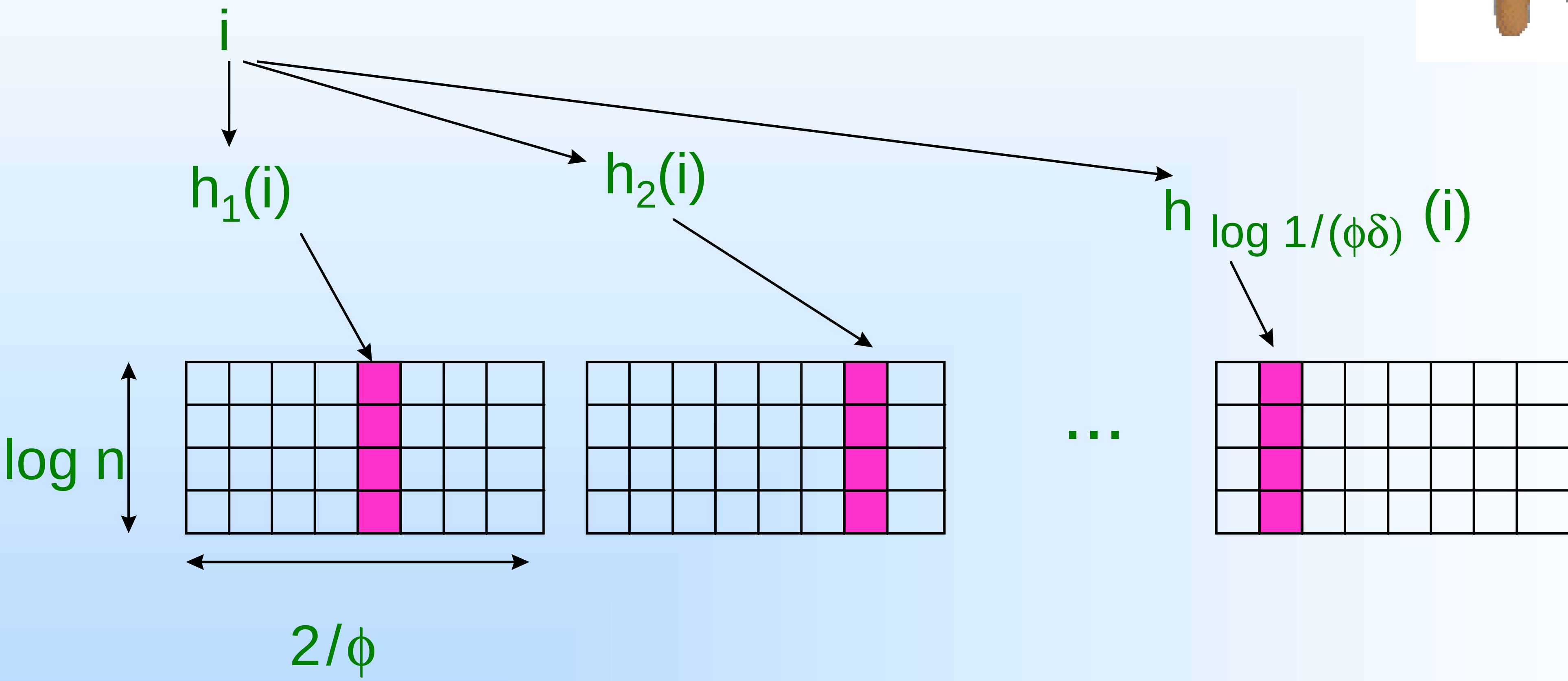
Randomized Construction

- Partition universe uniformly randomly to c/ϕ groups, $c > 1$
- Include item i in group j with probability ϕ/c
- Repeat enough times, each hot item is a majority in its group in some partition with high probability
- Storing description of groups explicitly is too expensive, so define groups by hash functions: but how strong hash functions?

Small space construction

- Pairwise independent hash function suffices, and these are easy to compute with.
- Range of hash fn is $2/\phi$, defines $2/\phi$ groups, group j holds all items i such that $h(i)=j$
- Use $\log 1/(\phi\delta)$ hash functions to get prob of success $= 1-\delta$
- In each group keep $\log n$ counters as before so can find the majority of items in group

Data Structure



Space used is $(2/\phi)*\log(n)*\log(1/(\phi\delta))$

Easy to update counts for inserts, deletes

Search Procedure

If group count is $> \phi \|a\|_1$ assume hot item is in there, and search subgroups

For each of log n splits, reject some bad cases:

- if both halves of the split $> \phi\|a\|_1$, could be 2 hot items in the same set, so abort
- if both halves of the split $< \phi\|a\|_1$, cannot be hot item in the set, so abort
- Else, find index of candidate hot item

Avoiding False Positives

Some danger of including an infrequent item in the output, so for each candidate:

- check the candidate hashes to the group that produced that candidate
- check each group it is in to ensure every one passes threshold.

Together these will guarantee chance of false positive is small.

Recap

- Find heavy items using Group Testing
- Spread items out into groups using hash fns
- If there is 1 hot item and little else in a group, it is majority, find using log groups
- Want to analyze probability each hot item lands in such a group (so no false negatives)
- Can also bound probability of false positives, but skipped for this talk.

Probability of Success

For each hot item, can identify if its group does not contain much additional weight.

That is, if total other weight $\leq \phi \, ||a||_1$ it is majority

By pairwise independence, linearity of expectation, expected weight in same bucket:

$$E(wt) \leq \Sigma\, a[i]\phi/2 \leq \phi||a||_1/2$$

By Markov inequality, $\Pr[wt > \phi \, ||a||_1] < ½$

So constant probability of success. Repeat for $\log 1/(\phi\delta)$ hash functions, gives probability $1 - \delta$ every hot item is in output

Time and Space Costs

- Update cost: Compute $\log 1/(\phi\delta)$ hash functions, update $\log(n) \log 1/(\phi\delta)$ counters
- Space is small: $2/\phi \log(n) \log 1/(\phi\delta)$ counts, decoding requires a linear scan of counts.
- Bonus: can specify $\phi' > \phi$ at query time
- Results do not depend on order of updates

Outline

- Problem definition and lower bounds
- Finding Heavy Hitters via Group Testing
  - Finding a simple majority
  - Non-adaptive Group Testing
  - **Experimental Evaluation**
- Extensions and Conclusions

Experiments

Wanted to test the recall and precision of the different methods

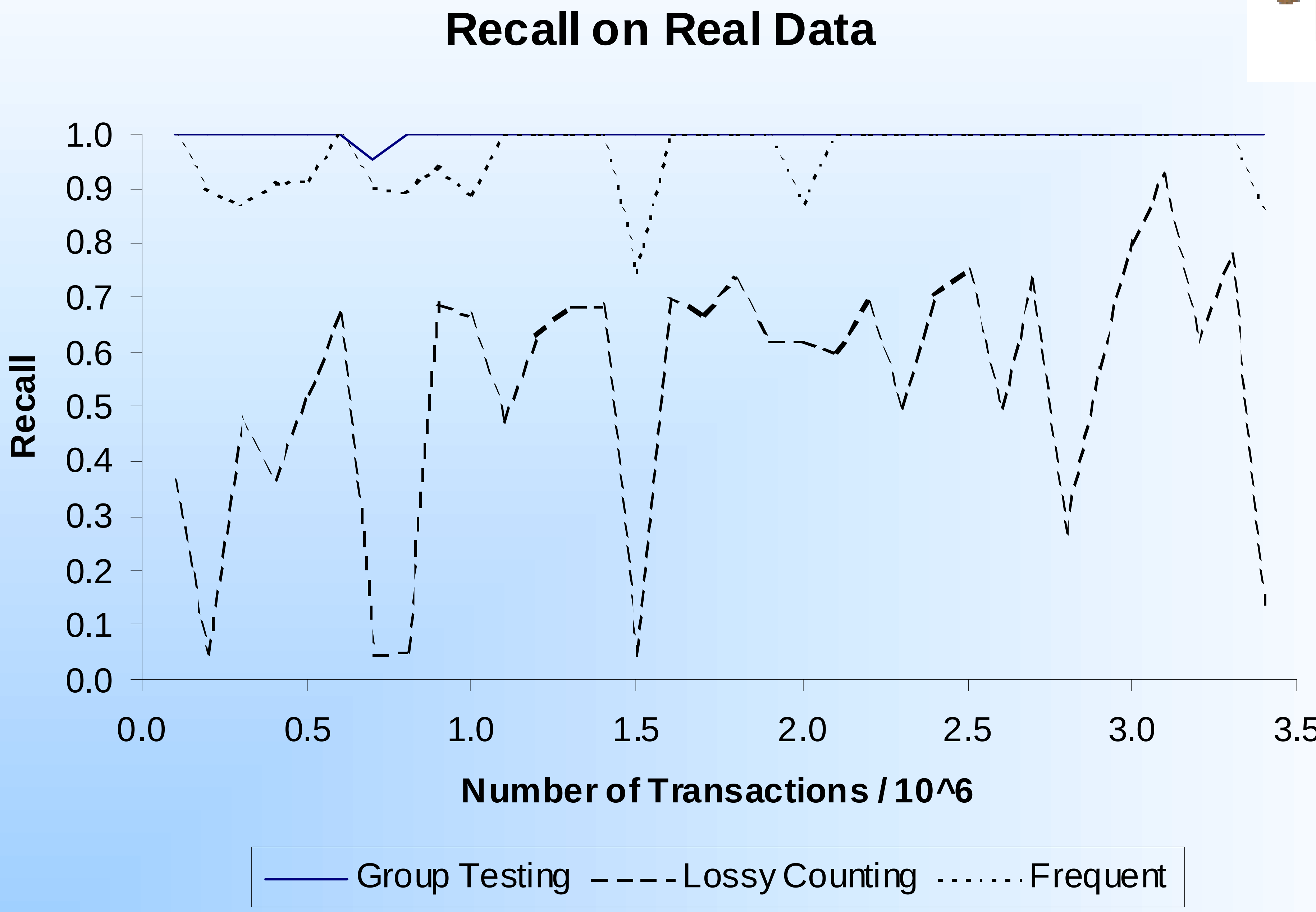
Recall = % of frequent items found

Precision = % of found items frequent

A relatively small experiment... processed a few million phone calls (from one day)

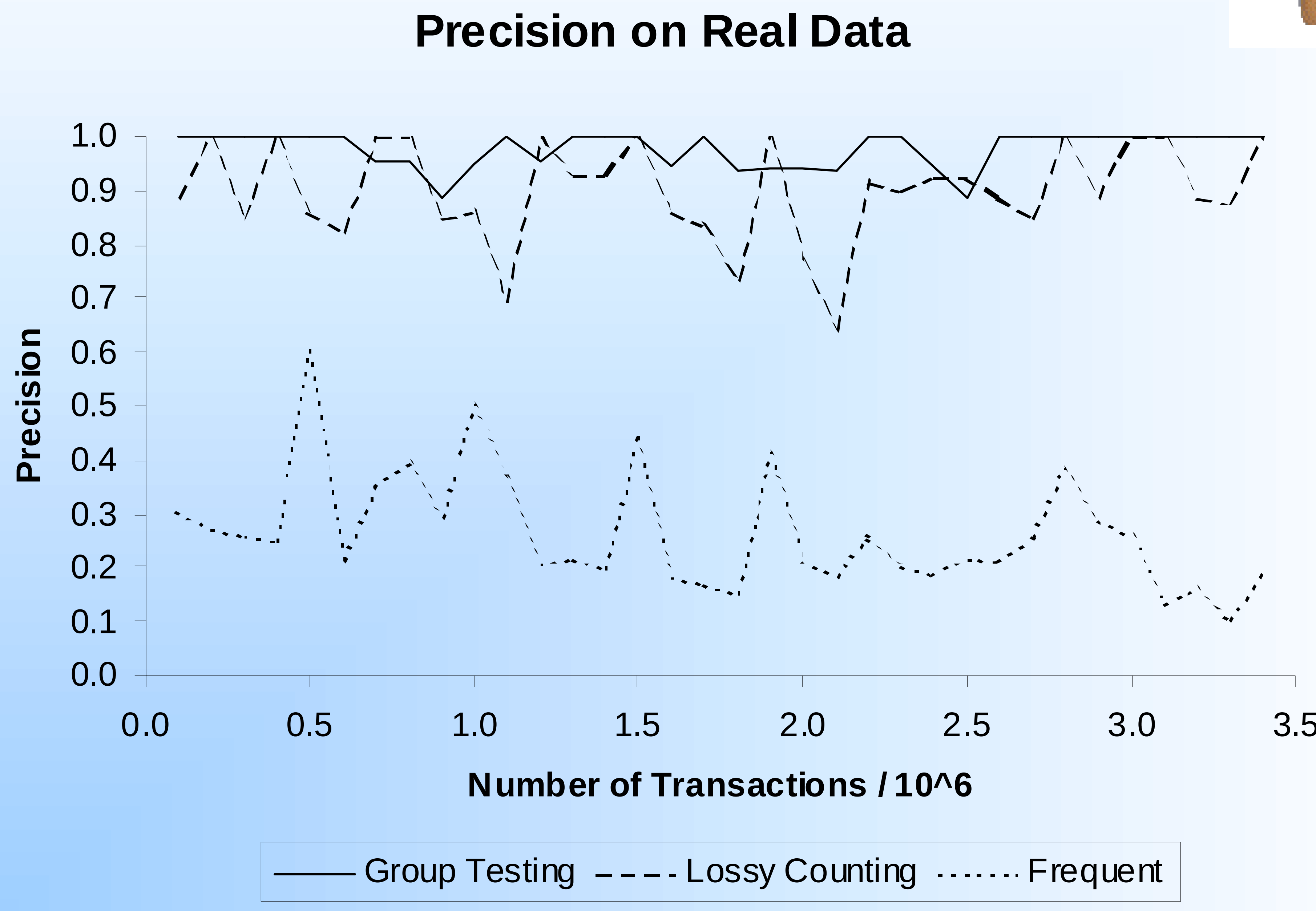
Compared to algorithms for inserts only, modified to handle deletions heuristically.

Recall



Recall on Real Data
1.0
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0.0
Recall
0.0
0.5
1.0
1.5
2.0
2.5
3.0
3.5
Number of Transactions / 10^6
Group Testing
Lossy Counting
Frequent

Precision


Precision on Real Data
1.0
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0.0
Precision
0.0
0.5
1.0
1.5
2.0
2.5
3.0
3.5
Number of Transactions / 10^6
Group Testing
Lossy Counting
Frequent

Outline

- Problem definition and lower bounds
- Finding Heavy Hitters via Group Testing
  - Finding a simple majority
  - Non-adaptive Group Testing
  - Experimental Evaluation
- **Extensions and Conclusions**

Conclusions

- The result is a pretty fast, pretty simple solution: just keep counts.
- Sketch based solutions are more costly, both in O() and in constants: here size is around a few hundred Kb.
- Seems to work well in practice.

Extensions in Progress

- An adaptive group testing solution, with slightly improved guarantees and costs (as a tech report)
- Finding hot items in hierarchies (with Korn and Srivastava, VLDB 03)
- Find large abolute or relative changes in item counts (eg between yesterday and today): conceptually, hot items relative to a vector of differences (in progress)

Open Problems

- Deterministic solutions exist for inserts only, is randomness necessary here?
- What if data is multidimensional: what are hot items here, and how to find them?
- In some sense hot items are “anomalies”, but are they really anomolous? Are anomalies always hot items?