Finding Interesting Correlations with Conditional Heavy Hitters

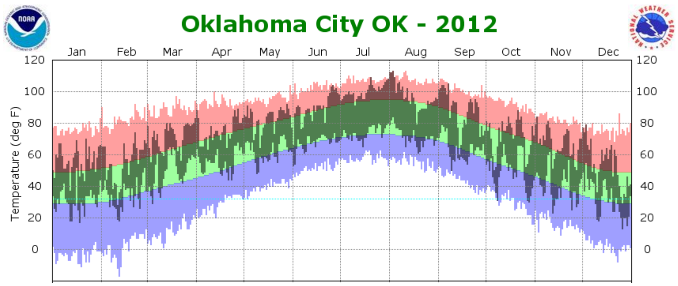
Katsiaryna Mirylenka (University of Trento)

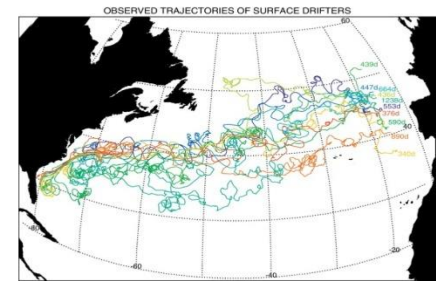
Themis Palpanas (University of Trento)

Graham Cormode (AT&T Labs)

Divesh Srivastava (AT&T Labs)

Streaming Data Processing



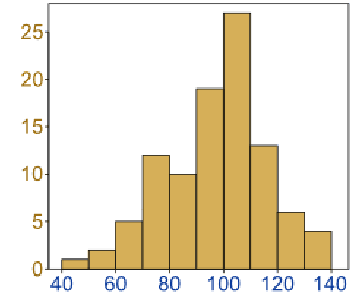


- Much big data arrives in the form of streams of updates
  - Each item in the stream gives more information
  - Stream is too large to store or forward
- Much prior work on streaming algorithms using small space
  - For “heavy hitters” (frequent items, frequent itemsets)
  - For quantiles, entropy and other statistical quantities
  - For data mining and machine learning (clustering, classifiers)
- Common application domains:
  - Network health monitoring (anomaly detection)
  - Intrusion detection over streams of events

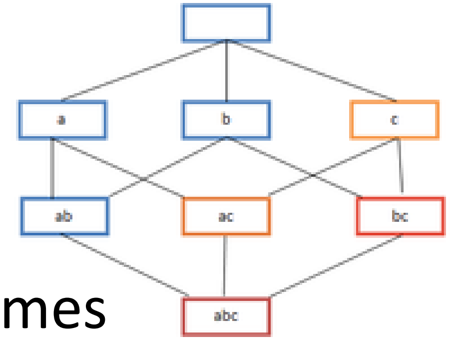
Limitations of current approaches

Existing streaming primitives not always suited to these cases:

- Tracking heavy hitters in network monitoring is too crude
  - Some sources or destinations are always popular
  - These may drown out the informative cases
  - Want to study data at a finer level of detail



- Frequent itemset mining in intrusion detection is not scalable
  - Enormous search space of possible combinations
  - Existing algorithms need a lot of space
  - Do not offer ‘real-time’ performance



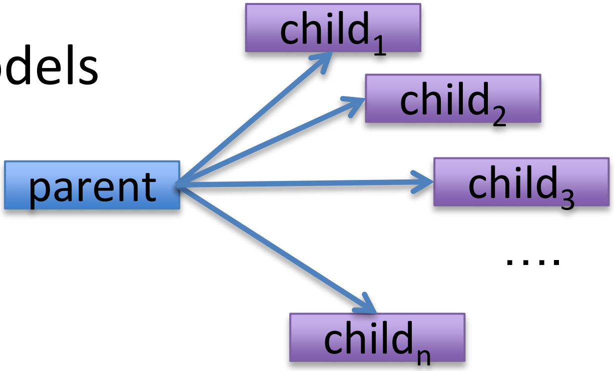
- Want mining primitive between these two extremes
  - Finer than heavy hitters, simpler than frequent itemsets

dbTrento

at&t

Conditional Heavy Hitters

- Observation: much data can be abstracted as pairs of items
  - (Source, destination) in network data
  - (Current, next) states in Markov chain models
  - Pairs of attributes in database systems
- First item is primary, other is secondary
  - Abstract as (parent, child) pairs
- Introduce the notion of conditional heavy hitters:
  - (parent, child) pairs where the child is frequent given the parent
  - We formalize this definition, and give algorithms to find them

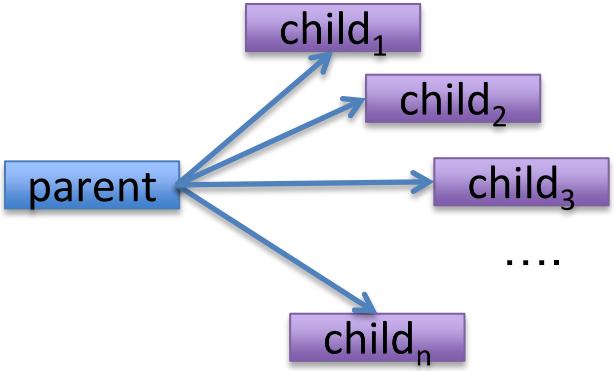


Conditional Heavy Hitters Definitions

- Given parents p, and children c, define
  - f_p as the frequency (count) of parent p in the stream
  - $f_{p,c}$ as the frequency (count) of pair (p,c) in the stream
  - $Pr[p]$ as the probability of p, f_p/n
  - $Pr[c|p]$ as the *conditional* probability of c given p, $f_{p,c}/f_p$
- Conditional heavy hitters are those (p, c) pairs with $Pr[c|p] > \phi$
  - Needs refinement: if $f_p = f_{p,c} = 1$, then $Pr[c|p]=1$
  - Restrict attention to those with the top-τ largest $f_{p,c}$ values
- Still a technically difficult problem
  - Lower bound shows a lot of space needed to give guarantees

Outline

- Introduce a sequence of four algorithms to find Conditional Heavy Hitters (CHH)
- Initial two algorithms store information on all parents
- Subsequent two algs track approximate information on parents
- Experimental study identifies where each algorithm performs best

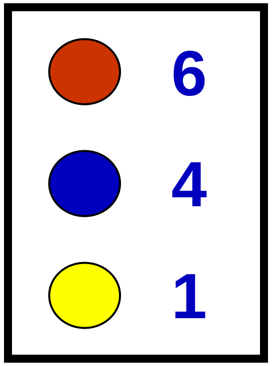


dbTrento

at&t
Your world. Delivered.

Space Saving Algorithm for HH

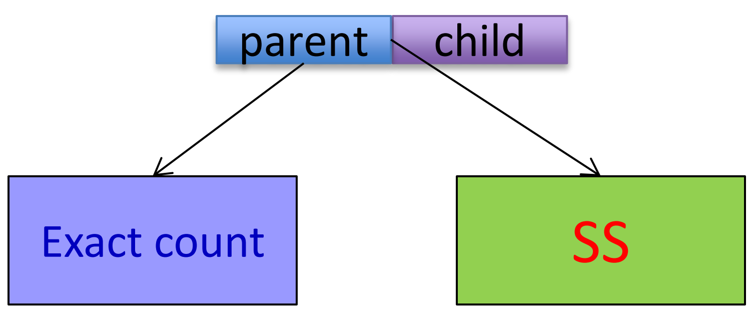
- Basic building block is an algorithm for heavy hitters (HH)
- SpaceSaving is an efficient HH algorithm [Metwally et al ‘05]
- Keeps information about k different items and their counts
  - If next item in stream is stored, update its count
  - If not, overwrite least frequent item and update count
- Guarantees error at most (n/k) on any count
- SpaceSaving (SS) often performs very well in practice



dbTrento

1. GlobalHH Algorithm

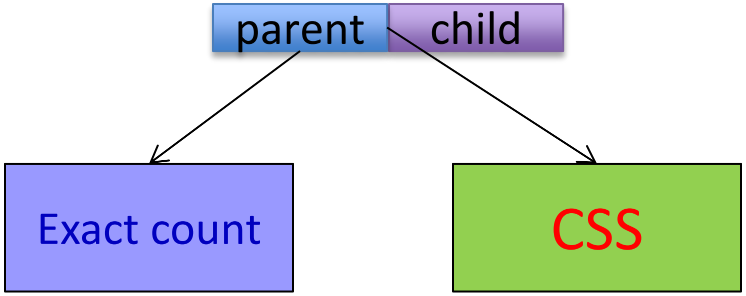
- Natural first approach to CHH problem:
  - Keep exact statistics on parent frequencies
  - Keep approximate counts of (parent, child) pairs via SS
  - Use approximate and exact information to estimate Pr[c|p]
  - Output CHHs based on these estimates
- Provides guarantees on estimated values:
  - Error in estimate of Pr[c|p] is at most $n/(k\ f_p)$
  - Error improves if distribution is skewed



dbTrento

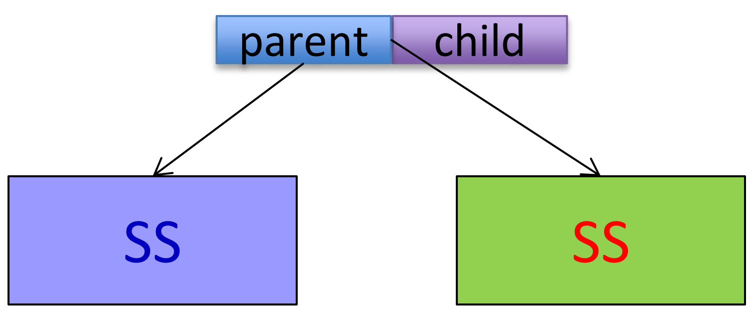
2. CondHH Algorithm

- Previous algorithm is not tuned to the CHH definition
  - SS algorithm prunes based on raw frequency
  - Instead, CondHH algorithm prunes based on (estimated) Pr[c|p]
- Introduce ConditionalSpaceSaving (CSS) algorithm:
  - Keeps information about k different items and their counts
  - If next item in stream is stored, update its count
  - If not, overwrite item **with lowest Pr[c|p] estimate**, update count
  - Use some implementation tricks to make fast to update
- CondHH: use CSS for (parent, child) pairs to estimate Pr[c|p]



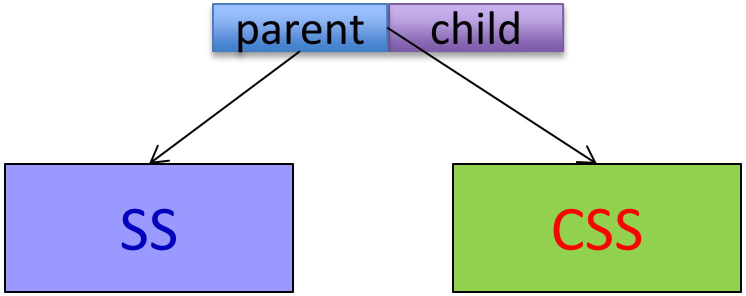
3. FamilyHH Algorithm

- Previous algorithms assumed we could store all parents
  - Not realistic as the domain of parents increases
- FamilyHH: natural generalization of GlobalHH
  - Keep SS for parents, and another SS for (parent,child) pairs
  - Use both approximate counts to estimate Pr[c|p]
- Similar worst case guarantees to GlobalHH
  - Given O(k) space, error in Pr[c|p] is at most $n/(k\ f_p)$



4. SparseHH Algorithm

- Last algorithm is the most involved
  - Keep SS on parents, CSS on parent, child pairs
- Given new (parent, child) pair, need to initialize its $f_{p,c}$ estimate
  - Can use additional data structures to track this information
  - Use hashing/Bloom filter techniques to minimize space
  - Experimentally determine how to divide available memory
- No worst-case guarantees on performance,
  - So we compare all algorithms empirically



Algorithm Summary

Algorithm	Parent	Parent,Child
1. GlobalHH	Exact	SS
2. CondHH	Exact	CSS
3. FamilyHH	SS	SS
4. SparseHH	SS	CSS

- Other algorithms proposed, performed less well
- For more details, see paper

dbTrento

at&t
Your world. Delivered.

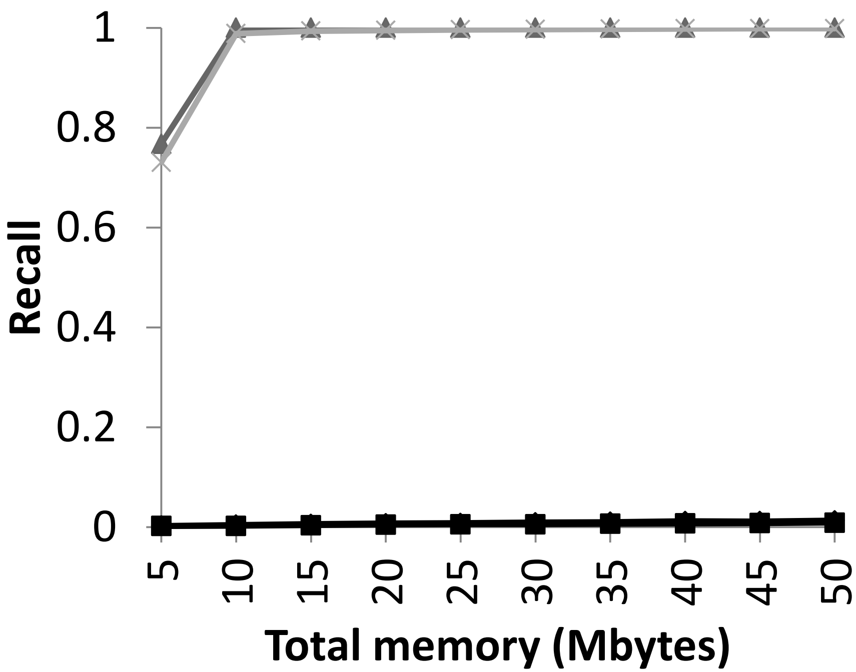
Experimental Study

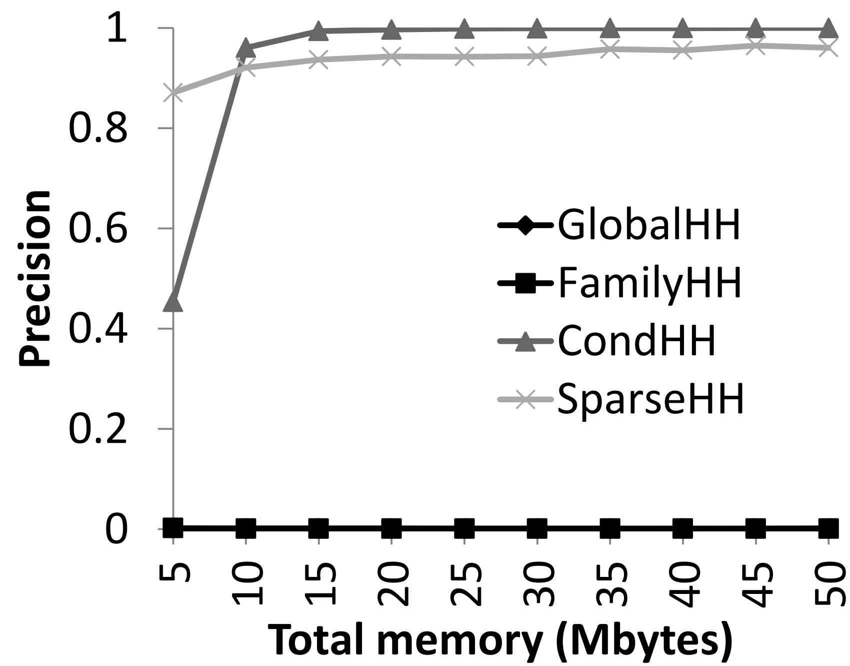
- Implemented and evaluated on variety of data
  - WorldCup data of (ClientID, ObjectID) request pairs
  - Taxicab GPS data: 54K trajectories in a 2^{nd} order Markov model
- Distinguish between data that is sparse and dense
  - Sparse data has few distinct children per parent (on average)
  - Dense data has many distinct children per parent (on average)
- Measure precision and recall of CHH recovery

dbTrento

at&t
Your world. Delivered.

Sparse Data Results



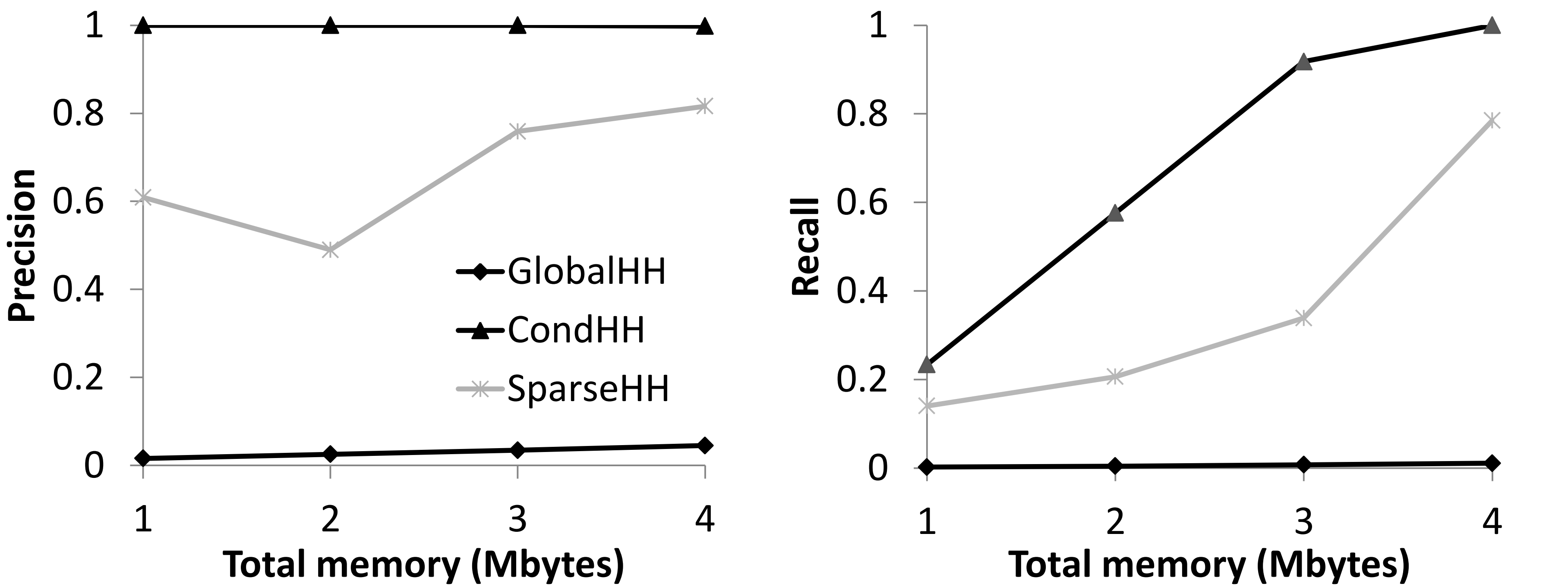


- World Cup data is sparse: 1/10 parents have a CHH child
- CondHH and SparseHH do well, both based on CSS
  - Keep very similar information internally
  - Other methods not competitive

dbTrento

at&t

Dense Data Results

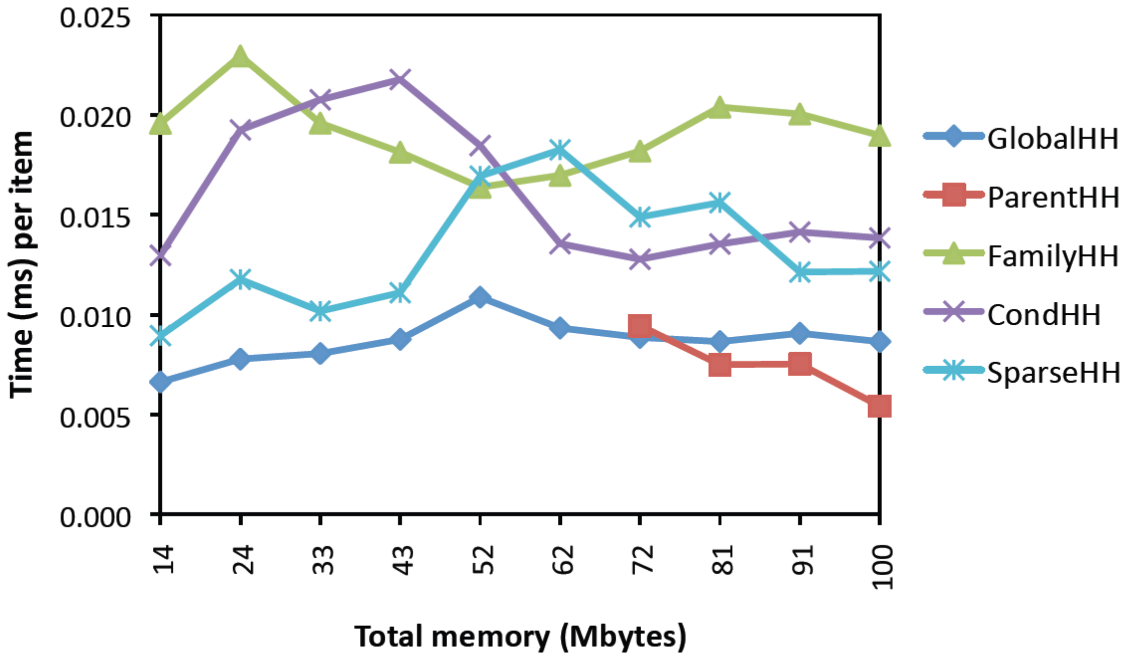


- Taxicab data is relatively dense, many parents have CHH child
- CondHH can take more advantage of available memory
- SparseHH converges on CondHH as more memory is used
- Other algorithms are not competitive

dbTrento

at&t

Throughput and Performance



- Not much variation as memory increases
- CondHH and SparseHH are slightly more expensive, due to more complex processing
- Throughput is still 5×10^5 items / second per core

dbTrento

Concluding Remarks

- High precision and recall of CHHs is possible on data streams
  - SparseHH algorithm works well over a variety of data types
  - CondHH is preferred when the data is more dense
- Future work:
  - Evaluate for Markov Chain parameter estimation
  - Compare to other recently proposed definitions

dbTrento

at&t

Your world. Delivered.

ParentHH Algorithm

- Keep small amount of information for each parent about its child distribution
  - Run an instance of SS for each parent
  - Track child distribution accurately
  - Use stored information to estimate Pr[c|p] and output CHHs
- Also provides guarantees on accuracy
  - Given total space k, error in estimate of Pr[c|p] is |P|/s
  - P denotes total number of parents

dbTrento

at&t
Your world. Delivered.