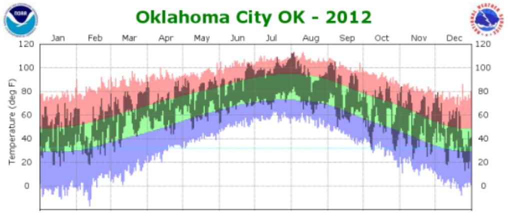
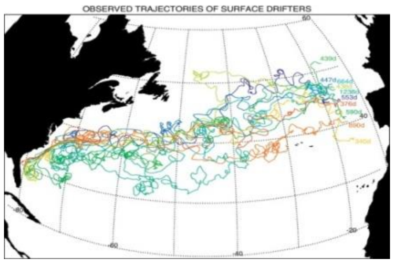
Finding Interesting Correlations with Conditional Heavy Hitters

Katsiaryna Mirylenka, Themis Palpanas (University of Trento)
Graham Cormode, Divesh Srivastava (AT&T Labs)



- Need to mine patterns from streams of updates
  - Each item in the stream gives more information
  - Stream is too large to store or forward

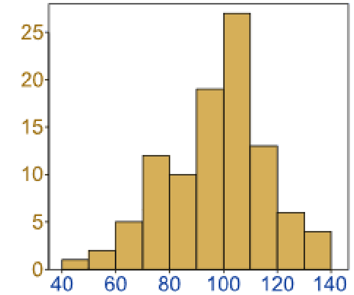


- Common application domains:
  - Network health monitoring (anomaly detection)
  - Intrusion detection over streams of events
- Prior work on stream mining in small space
  - For “heavy hitters” (frequent items, frequent itemset
  - For quantiles, entropy and other statistical quantities
  - For data mining and machine learning (clustering, cla

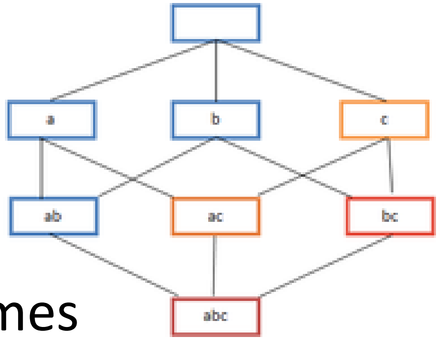
Limitations of current approaches

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



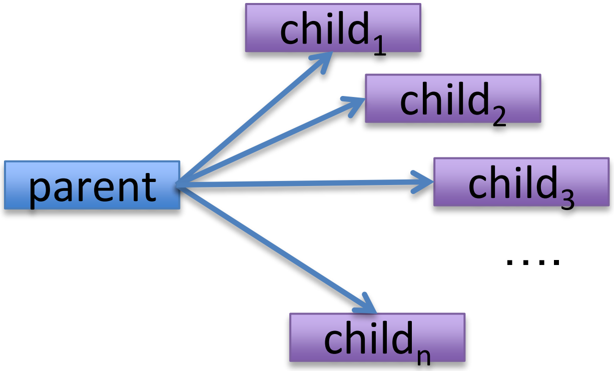
- Frequent itemset mining in intrusion detection is not scalable
  - Enormous search space of possible combinations
  - Existing algorithms need a lot of space
  - Do not offer 'real-time' performance



- Want mining primitive between these two extremes
  - Finer than heavy hitters, simpler than frequent itemsets
  - We propose **Conditional Heavy Hitters**

Conditional Heavy Hitters

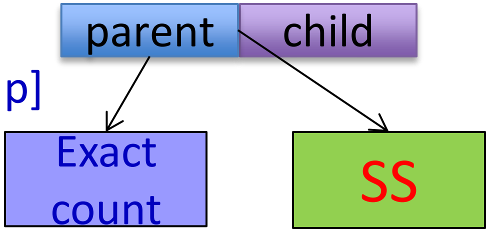
- Observation: much data can be abstracted as pairs of items
  - (Source, destination) in network data
  - (Current, next) states in Markov chain models
  - Pairs of attributes in database systems



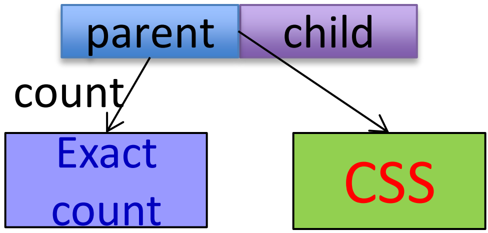
- First item is primary, other is secondary
  - Abstract as (parent, child) pairs
  - Seek (parent, child) pairs where the child is frequent given the parent
- Given parents p, and children c, define
  - f_p as the frequency (count) of parent p in the stream
  - $f_{p,c}$ as the frequency (count) of pair (p,c) in the stream
  - Pr[c|p] as the *conditional* probability of c given p, $f_{p,c}/f_p$
- Conditional heavy hitters are those (p, c) pairs with $\Pr[c|p] > \phi$
  - Define algorithms to find the top-τ based on their $f_{p,c}$ values

Exact Parent Algorithms

1. GlobalHH algorithm for the 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
   - Error in estimate of Pr[c|p] is at most $n/(k\ f_p)$



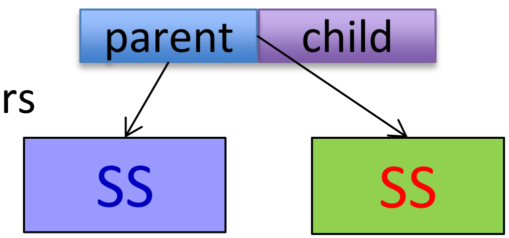
2. ConditionalSpaceSaving (CSS) algorithm is tuned to CHH definition:
   - Keeps information about k different items and their counts
   - If next item in stream is stored, update its count
   - If not, overwrite item **with lowest Pr[c|p] estimate**, update count
   - Use some implementation tricks to make fast to update
   - CondHH algorithm: uses CSS to estimate Pr[c|p]



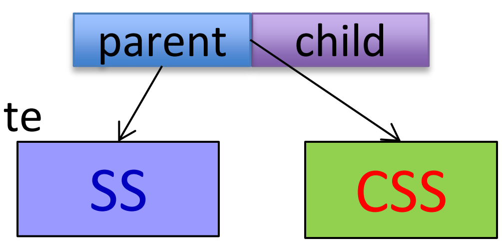
Approximate Parent Algorithms

- Previous algorithms assumed we could store all parents
  - Not realistic as the domain of parents increases, so keep approximate statistics

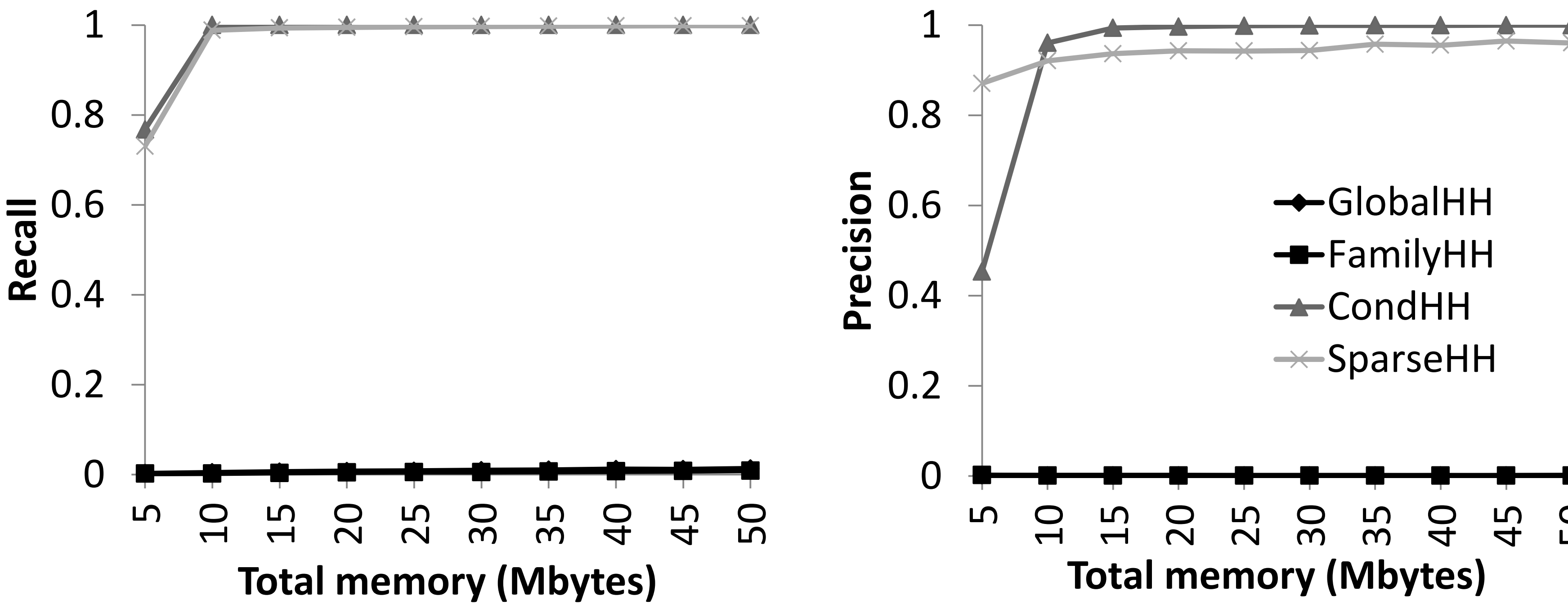
3. FamilyHH: natural generalization of GlobalHH
   - Keep SS for parents, and another SS for (parent,child) pairs
   - Use both approximate counts to estimate Pr[c|p]
   - Given O(k) space, error in Pr[c|p] is at most $n/(k\ f_p)$



4. SparseHH algorithm is the most involved
   - Keep SS on parents, CSS on parent, child pairs
   - Given new (parent, child) pair, must initialize its $f_{p,c}$ estimate
   - Use hashing/Bloom filter techniques for these estimates
   - Experimentally determine how to divide available memory

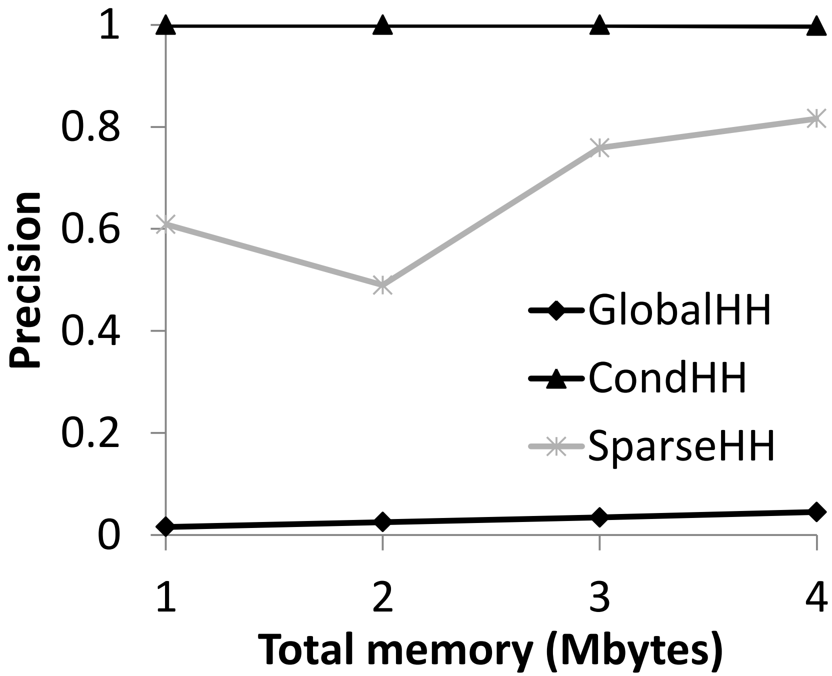


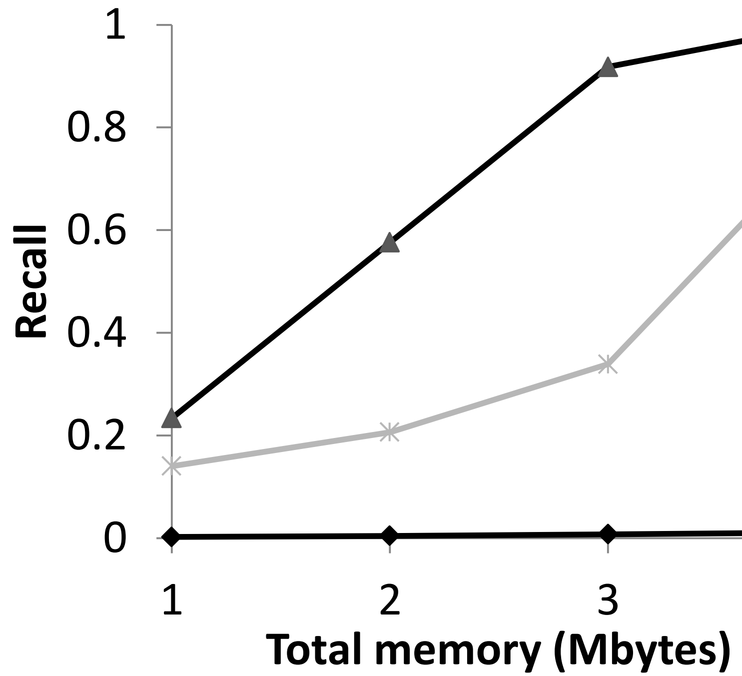
Sparse Data Results



- 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

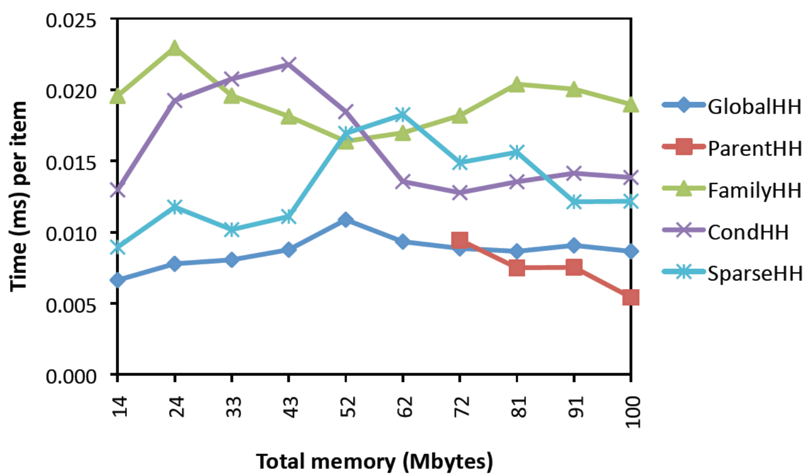
Dense Data Results





- 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 and variations are not competitive

Throughput and Conclusions



- Algs have good throughput
  - 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

- 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