Finding Interesting Correlations with Conditional Heavy Hitters

Katsiaryna Mirylenka [#], Themis Palpanas [#], Graham Cormode [*], Divesh Srivastava [*]

[#] *University of Trento,*
Trento, Italy
{kmirylenka, themis}@disi.unitn.eu

[*] *AT&T Labs – Research,*
Florham Park, NJ, USA
{graham,divesh}@research.att.com

***Abstract*— The notion of *heavy hitters*—items that make up a large fraction of the population—has been successfully used in a variety of applications across sensor and RFID monitoring, network data analysis, event mining, and more. Yet this notion often fails to capture the semantics we desire when we observe data in the form of correlated pairs. Here, we are interested in items that are *conditionally* frequent: when a particular item is frequent within the context of its parent item. In this work, we introduce and formalize the notion of Conditional Heavy Hitters to identify such items, with applications in network monitoring, and Markov chain modeling. We introduce several streaming algorithms that allow us to find conditional heavy hitters efficiently, and provide analytical results. Different algorithms are successful for different input characteristics. We perform experimental evaluations to demonstrate the efficacy of our methods, and to study which algorithms are most suited for different types of data.**

I. Introduction

Within applications that generate large quantities of data, it is often important to identify particular entities that are associated with a large fraction of the data items [1], [2]. For example, in a network setting, we often want to find which users are responsible for sending or receiving a large fraction of the traffic. In monitoring updates to a large database table, it is important to know which attribute values predominate, for query planning and approximate query answering purposes. This notion has been abstracted as the idea of "heavy hitters" or "frequent items". There has been much effort spent in finding algorithms to track these under a variety of scenarios and data arrival models [3], [4], [5], [6], [7], [8], [9].

However, the concept of heavy hitters can on occasion be quite a blunt one. Consider again the network health monitoring scenario. Here, it is well-known that in any measurement, there will be some destinations that are globally popular (search engines, social networks, video providers); likewise, so will be certain users (large organizations behind a single IP address, heavy downloaders and filesharers). As a result, tracking the heavy hitters within this data is not always informative, as they reveal only knowledge which is relatively slow changing, and not actionable. Rather, we would like to find which are sources or destinations that are significantly *locally* popular. That is, find those (source, destination) pairs where the destination is a heavy hitter amongst the connections of the same source.

Another application area is in security, in the intrusion detection domain. Here, a large number of different actions are observed, and the goal is to sift for unusual patterns of activity. The canonical approach is based around association rule and frequent itemset mining. These methods identify subsets of activities whose joint occurrence frequency exceeds some given threshold. While popular, this methodology has its limitations. The enormous search space implied by all possible combinations of actions typically requires a lengthy off-line search to identify the patterns of interest. While there are some on-line algorithms, these still require substantial resources to track sufficient statistics for the potentially frequent subsets. As a result, this kind of mining tends to be costly, and to deliver results significantly after the event.

Essentially, these two approaches (frequent items and frequent itemsets) fall at two ends of the spectrum: the frequent items approach is not rich enough to identify behavior of interest, while the frequent itemsets are potentially too rich, and too costly to find. In this work, we propose an intermediate goal, which teases out more correlations between items than simple heavy hitters, but is lightweight enough to permit efficient streaming algorithms. We dub this concept "conditional heavy hitters", and work to provide meaningful definitions and a suite of algorithmic approaches to find them.

Specifically, we model data that can be abstracted as pairs of items, which we refer to as *parent* and *child* items. The central concept of conditional heavy hitters is to find those parent-child pairs that are most frequent, relative to the frequency of the parent. The reason for referring to this as "conditional" is by analogy to conditional probabilities: essentially, we seek children whose probability is high, conditioned on the parent. These should be distinct from the parent-child pairs which are overall most frequent, since these can be found by using existing heavy hitter algorithms. While this is a natural goal, it turns out that there are several ways to formalize this, which we discuss in more detail in subsequent sections.

Equipped with the concept of conditional heavy hitters, we can now apply it to a variety of settings:

- In a network monitoring setting, we can look for the conditional heavy hitters over packets within the network,

978-1-4673-4910-9/13/$31.00 © 2013 IEEE

where for each packet the source address is considered as the parent, and the destination address is the child. Then the conditional heavy hitters identify those destinations which occur most frequently for their corresponding sources. This can be useful in identifying local trends, shifts in popularity, and traffic planning, especially when several sources are seen to share the same child as a conditional heavy hitter.

- When modeling many real systems and processes, it is common to use a Markov chain to capture the transition behavior between states. However, many of the systems have large numbers of possible states (e.g. modeling traffic flow in a large city), and so it can be costly to maintain complete statistics. Instead it suffices if we can just measure the most important observed transition probabilities from state to state from a stream of state occupancies. That is, we identify the large probabilities of moving from one state to another—these are exactly the conditional heavy hitters, with the parent being the current state, and the child being the transition taken. Thus, tracking the conditional heavy hitters over a long sequence of observations of state transitions can capture the essential parameters of the Markov chain.
- Within a database management system or data warehouse, there are a large number of transactions which affect the overall data distribution. Such systems commonly keep statistics on individual attributes, to capture the number of distinct values, frequent items, and so on. Given the large number of columns, it is infeasible to keep detailed statistics on all combinations of columns. However, a suitable compromise is to keep some summary statistics on pairs of columns which commonly co-occur (in join paths, say). Finding the conditional heavy hitters within such pairs captures information about the correlations between them, and can allow improved selectivity estimation.

Contributions. The main contributions of our work include:

- We define the concept of conditional heavy hitters, which can be applied in a variety of settings;
- We develop and describe several streaming algorithms for retrieving conditional heavy hitters and analyze their applicability for data with varying characteristics;
- The algorithms developed are evaluated on a mixture of real and synthetic datasets. We observe that certain algorithms can retrieve the conditional heavy hitters with high accuracy while retaining a compact amount of historical information. We observe that different algorithms achieve the best results depending on simple characteristics of the data: essentially, whether the number of conditional heavy hitters is comparable to the number of parents, or whether it is much lower.

The rest of the paper is organized as follows: Section II describes background and related work for heavy hitters and frequent itemset mining. In Section III we refine the notion of conditional heavy hitters to obtain a workable definition. In Section IV we present and discuss a sequence of algorithms to find conditional heavy hitters from a stream of data. Our experimental results are shown in Section V, and we conclude our discussion in Section VI.

II. Related Work

The notions of heavy hitters and frequent itemsets have been heavily studied in the database and data mining literature. Interest in finding the heavy hitters in streams of data goes back to the early eighties [4], [10], where simple algorithms based on tracking items and counts were developed. Thanks to the interest in algorithms for streams of data, improved methods were developed over the course of the last decade. These included variants of methods which track items and corresponding estimated counts [11], [3], [5], and randomized "sketch" methods, capable of handling negative weights [8], [9]. These methods can all provide the guarantee that given a parameter ϵ, they can find all items in a stream of length n which occur more than ϵn times, while maintaining a summary of size $O(1/\epsilon)$. Equivalently, they estimate the frequency of any given item with additive error ϵn. For further details and empirical comparison of methods, see the surveys [1], [2].

The heavy hitters are a special case of frequent itemsets: they are the frequent 1-itemsets. Further, all larger frequent itemsets consist of subsets of the heavy hitters. There has been much work to find frequent itemsets (and their variations) in the off-line setting, often starting from the A priori [12] and FP-Tree algorithms [13]. These concepts have been adapted to work over streams of data, generating algorithms such as FUP [14], and FP-stream [15]. A limitation of finding frequent itemsets is that the number of possibly frequent itemsets can become very large, meaning that the algorithm either has to track information about many candidates, or else aggressively prune the retained data, and risk missing out on some frequent itemsets. In formalizing conditional heavy hitters, one aim is to form a compromise between heavy hitters (which are simple and for which space/accuracy tradeoffs can be provided) and frequent itemsets (which are much more complex, for which no tight space guarantees are provided). Additional background on itemset mining in streams is given by Yu and Chi [16].

Several other variations of heavy hitters on streams have been proposed in the literature. Where the stream is time-varying, it is sometimes of interest to monitor only the heavy hitters within a recent time window, or with some other time-decay [17], [18], [19]. The 'distinct heavy hitters' are found over pairs of items (a, b), as those items a associated with a large number of distinct values b [20]. The notion of hierarchical heavy hitters says that when items fall in a hierarchy (or combination of hierarchies), it is interesting to find nodes in the hierarchy that are heavy from aggregating their descendants [23]. Lastly, correlated aggregates consider streams of tuples, and ask for aggregates on some attributes, for the subset of tuples that meet some other conditions [24]. In this regard, our concept of conditional heavy hitters can be seen as a generalized version of a kind of correlated aggregate, albeit one that has not been studied previously.

Most related is the work of Lahiri and Tirthapura [21] which considers the problem of 'correlated heavy hitters' over a stream of tuples (a, b). Here, (a, b) is a correlated heavy

hitter if a is a simple heavy hitter (frequency exceeds ψ) in a sequence of single-dimensional records and b is a heavy hitter in the subset of tuples where a appears. However, as we discuss below, this definition may miss interesting correlations, since it restricts the a records to be heavy hitters.

Our notion of conditional heavy hitters is related to models of (temporal) correlation in data, as captured by Markov chains. That is, given a sequence of items, the kth order transition probabilities are defined as the (marginal) probability of seeing each character, given the history of the k prior characters. In our terminology, setting the child as the new character and the parent as the concatenation of the k previous characters means that finding the conditional heavy hitters maps on to finding the high transition probabilities in this Markov chain. The importance of considering correlations has been recently motivated within several domains [22], [27]. There has been much prior work on capturing correlations in data via different Markov-style models, such as homogeneous Markov chains of high order, hidden Markov Models [25], Bayesian networks [26] and others [27], [28]. However, fitting these increasingly complex models requires a lot of CPU and I/O time and multiple passes over the data, and hence it is infeasible to estimate them in a streaming setting. For example, the simple Mixture Transition Distribution [29] aims to approximate the transition probabilities with a smaller number of parameters, but requires multiple iterations over the data to do so. By focusing on the conditional heavy hitters, we also identify a small number of parameters to describe the distribution, but can recover these efficiently in a single pass over the data.

III. Preliminaries

To allow our definition to be generally applicable, we assume that the input can be modeled as a stream of pairs of (parent, child) values (p, c).

Definition 1 (Frequencies): Given a stream of (parent, child) pairs whose ith element is (p_i, c_i), the frequency of a parent p, f_p, is defined as

$$f_p = |\{i : p_i = p\}|.$$

The frequency of a (parent, child) pair, $f_{p,c}$, is defined as

$$f_{p,c} = |\{i : p_i = p \wedge c_i = c\}|.$$

From these frequencies, we can define (empirical) probabilities associated with items and pairs.

Definition 2 (Probabilities): Given a stream of n (parent, child) pairs, the empirical probability of a parent p, $\Pr[p]$, is defined as $\Pr[p] = f_p/n$. The joint probability of a parent-child pair, $\Pr[p, c]$, is defined as $\Pr[p, c] = f_{p,c}/n$. The conditional probability of a child given a parent, $\Pr[c|p]$, is defined as

$$\Pr[c|p] = \frac{\Pr[p, c]}{\Pr[p]} = \frac{f_{p,c}}{f_p}.$$

We can now define a first notion of conditional heavy hitters.

Definition 3 (Conditional Heavy Hitter): We say that a pair (p, c) is a conditional heavy hitter with respect to a threshold $0 < \phi < 1$ if $\Pr[c|p] \geq \phi$.

This definition has the advantage of clarity and simplicity. However, on further consideration, there are some drawbacks associated with this formulation. Firstly, observe that when a parent is rare, it is more likely to generate conditional heavy hitters. As a clear example, consider the case of a parent p that occurs only once in the stream. Then we have $f_{p,c} = f_p = 1$ for the associated child c, and so $\Pr[c|p] = 1$, making it automatically a conditional heavy hitter. While this is a valid application of the definition—the (empirical) conditional probability of this child truly is 1—we might still object that this is not a particularly significant association, due to the limited support of this item. Second, for related reasons, the total number of conditional heavy hitters meeting this definition can be large. Specifically, in an extreme case a given parent p can have $\Theta(1/\phi)$ distinct children which are all conditional heavy hitters. So if there are $|P|$ distinct parents, there can be a total of $\Theta(|P|/\phi)$ distinct conditional heavy hitters—a very large amount—many of which may be infrequent and uninformative.

A natural way to avoid these issues is to place an additional constraint on the frequency of the parent, thus limiting the number of parents which can contribute to conditional heavy hitters. One solution would be to additionally require that $\Pr[p] > \psi$ for (p, c) to form a conditional heavy hitter (similar to [21]). Certainly, this has the desired effect: the number of conditional heavy hitters can now be at most $\Omega(1/\phi\psi)$, and parents with very small count can no longer contribute conditional heavy hitters. However, we argue that this definition is overly restrictive: it restricts attention to only those parents who are ψ-heavy hitters, and so misses those pairs which may have a significant correlation despite a lower total frequency. Other formulations, such as requiring $\Pr[(p, c)] > \phi\psi$ have similar drawbacks. Consequently, we set up a different requirement as our goal.

Definition 4 (Popular conditional heavy hitter): A pair (p, c) is a *popular* conditional heavy hitter if it is a conditional heavy hitter with respect to ϕ, and it ranks among the top-τ of the conditional heavy hitters, ordered by $f_{p,c}$.

This says that we seek parent-child pairs that are conditional heavy hitters, with a preference to those that have a higher occurrence within the observed data. In realistic data sets, we may expect that there will be many conditional heavy hitters with large $f_{p,c}$ values, which will ensure that we avoid the trivial case of $f_{p,c} = f_p = 1$. Consequently, this represents a workable definition, that avoids this unwanted case, while also avoiding ruling out interesting cases.

Given this definition, it is not possible to provide algorithms which guarantee to always find exactly those items counted as popular conditional heavy hitters while also using small space, as shown by the following lemma:

Lemma 1: Any (randomized) one-pass algorithm which promises to find all popular conditional heavy hitters must use $\Omega(\min(n, |P|))$ space, where n is the length of the stream, and $|P|$ is the number of distinct parents.

Proof: Consider a stream of items, $x_1, x_2, \ldots x_n$, and suppose we have an algorithm which finds popular conditional

TABLE I

MAIN CHARACTERISTICS OF THE PROPOSED ALGORITHMS

Algorithm	Parents	Summary structure	Eviction order
GlobalHH	all	global	parent-child frequency
ParentHH	all	local	parent-child frequency
CondHH	all	global	conditional probability
FamilyHH	partial	global	parent-child frequency
SparseHH	partial	global	conditional probability

heavy hitters. From this stream, we generate a new stream of parent-child pairs, $(x_1, 0), (x_2, 0), \ldots (x_n, 0)$. Then each distinct pair is a conditional heavy hitter: $P[0|p] = 1$. Thus, the algorithm must find the top-τ most frequently occurring parents. But observe that these correspond exactly to the top-τ most frequently occurring items in the original stream. It has been shown that accurately solving this problem requires space $\Omega(\min(n, U))$ over a set of U possible items, even just to find the most frequent item [30], which gives the claimed lower bound. ■

In some cases, $|P|$ is not so large, and so we can look for algorithms which use this much space. In other cases, $|P|$ may be very large, and this amount of space is impractical. However, this bound should not cause us to abandon hope of finding methods which are effective in practice. The kinds of sequences which are used to construct the worst-case examples in the lower-bounds are very artificial, where all items occur only once or twice within the data, forcing any correct algorithm to keep enough information to distinguish which items occur more than once. Realistic data is more varied, and so there is more evidence spread throughout the stream to help identify the conditional heavy hitters.

Our goal will be to design algorithms which allow us to estimate the conditional probability of parent-child pairs accurately. That is, the goal is find an estimate $\widehat{\Pr}[c|p]$ that accurately estimates $\Pr[c|p]$. From this, we will be able to find candidate conditional heavy hitters. Our experimental study evaluates the ability of these algorithms to find such conditional heavy hitters.

IV. ALGORITHMS FOR CONDITIONAL HEAVY HITTERS

In this section, we describe a variety of algorithms to help us identify the conditional heavy hitters within a stream of data. These are summarized in Table I. We begin with algorithms for the traditional heavy hitters problem, and adapt these to identify those which are conditional heavy hitters.

A. GlobalHH Algorithm

Our first algorithm aims to identify all parent-child pairs that occur frequently, so that we can extract the subset that are conditional heavy hitters. For this task, we make use of existing algorithms to find the heavy hitters from a stream of items. The *SpaceSaving* algorithm due to Metwally *et al.* [5] has been widely used for this problem. Given an amount of space s, it guarantees to find all items in a stream of length n which occur more than n/s times (and, moreover, to provide an estimate of their frequency which is accurate up to an n/s additive error). For streams which obey a frequency distribution that follows a long-tail distribution, formalized as a Zipfian distribution, it further guarantees to provide accurate recovery of the head of the distribution. The algorithm behaves very well in practice, finding accurate estimates of frequencies of items across a variety of data sets [1], [2]: it exhibits the most stable behavior among all heavy hitter algorithms.

The algorithm works as follows: it maintains a collection SS of k items and associated counts. For simplicity, assume that the structure is initialized with k arbitrary items with count 0. For each item x seen in the stream, if it is currently stored in the collection, the associated count $\hat{f}_x$ is incremented. Otherwise, we find the item with the current smallest count in the collection, and replace it with the new item, then increment its count. At any time, we can estimate the frequency of any item x with the associated count in the collection $\hat{f}_x$ if the item is stored, and 0 otherwise.

We adapt this algorithm to help us find conditional heavy hitters. Given each (p, c) pair in the stream, we insert it into the SS structure. We also separately maintain information on the frequency of each parent. In this first algorithm, we assume that there is sufficient space to store information on all parents, and so we have f_p exactly. To identify the popular conditional heavy hitters, we step through all items in the SS structure in order of count. For each stored (p, c) pair, we compute its estimated $\hat{f}_{p,c}$ value, and test whether $\hat{f}_{p,c}/f_p > \phi$, then output it if so. We stop when we have output τ estimated popular conditional heavy hitters, or when the SS data structure is exhausted. We refer to this algorithm as the *GlobalHH* algorithm, since it is based on finding the parent-child pairs which are global heavy hitters.

Lemma 2: Given space $O(s+|P|)$, the GlobalHH algorithm guarantees that each candidate (p, c) pair output will have $\Pr[c|p] \leq \widehat{\Pr}[c|p] \leq \Pr[c|p] + \frac{1}{s \Pr[p]}$. When the distribution of (p, c) pairs follows a Zipfian distribution with parameter $z > 1$, the error bound is improved to $\frac{1}{s^z \Pr[p]}$.

Proof: Since the f_p values are found exactly, the uncertainty in the estimated conditional probability, $\widehat{\Pr}[c|p]$ is due to the error from the SS algorithm. This guarantees that our estimate of $f_{p,c}$ is an overestimate by at most n/s for arbitrary streams. We output $\hat{f}_{p,c}/f_p$, which overestimates by at most $\frac{n}{s f_p} = \frac{1}{s \Pr[p]}$. Therefore, we have the bound stated. This guarantees to overestimate the conditional probability, and so will ensure good recall. Alternately, we could provide an underestimate of the conditional probability by using a lower bound on the estimate of $f_{p,c}$. In this case, we ensure good precision, but do not guarantee recall. For streams with Zipfian frequency distribution, the error bound is tightened to ns^{-z} [5], improving the error bound to $\frac{1}{s^z \Pr[p]}$ as claimed. ■

B. ParentHH Algorithm

Our second algorithm takes a parent-centric view of the problem. Again, making the assumption that it is feasible to retain information about each distinct parent observed, we consider the case of keeping information about the set of children associated with each parent. That is, we keep a

separate instance of the SS structure for each distinct parent. Clearly, this can use a lot of space, but will allow very accurate recovery of conditional heavy hitters. We call this the *ParentHH* algorithm, since it retains heavy hitter information for each parent.

Specifically, for each parent p observed, we maintain an instance of the SS structure of size $s/|P|$ for just those children c that arrive as part of a pair for that p. For each pair (p, c) that arrives, we insert c into SS_p, the structure for that parent. To recover the estimated conditional heavy hitters, we consider each parent in turn, and find $\hat{f}_{p,c}$ for each child c stored in SS_P. We consider these values in sorted order of $\hat{f}_{p,c}$, and output (p, c) as an estimated conditional heavy hitter if $\hat{f}_{p,c}/f_p \geq \phi$, and at most τ such pairs have been output.

Lemma 3: Given space $O(\min(s, n))$ (for $s > |P|$), the ParentHH algorithm guarantees that each candidate (p, c) pair output will have

$$\Pr[c|p] \leq \widehat{\Pr}[c|p] \leq \Pr[c|p] + \frac{|P|}{s}.$$

Proof: From the SS_p structure, we obtain an estimate of $f_{p,c}$ which has error proportional to the number of items passed to the structure, which is just f_p, the number of occurrences of p. So the amount by which $\hat{f}_{p,c}$ overestimates is at most $f_p|P|/s$. When we estimate $\widehat{\Pr}[c|p]$, the error is $(f_p|P|/s)/f_p = |P|/s$. The space bound follows immediately: it is bounded by n, since each item in the stream can increase the number of tuples stored by at most a constant amount. Using this overestimate favors recall, at the cost of precision. It is possible to instead use an underestimate of $f_{p,c}$, by subtracting an appropriate amount. In this case, we obtain good precision, but without guarantees on recall. ∎

Clearly, this algorithm provides accurate estimated conditional probabilities, but at a cost: we devote up to $s/|P|$ space for each parent, which seems excessive for parents that turn out to be relatively infrequent (and hence their children are unlikely to appear as true conditional heavy hitters).

C. CondHH algorithm

Our third algorithm also keeps a summary structure similar to the previous algorithms, but with a different goal. Instead of using the absolute frequency to determine which items to retain detailed information for, we instead use their (estimated) conditional probability. Since this aligns with the overall goal, it may lead to more accurate behavior.

In the CondHH algorithm, we keep a collection of (p, c) pairs in a data structure we refer to as CSS, along with a count for each, intended to serve as an estimate of $f_{p,c}$. The algorithm proceeds similarly to the GlobalHH case: for each (p, c) pair that arrives, we see if it is stored in CSS, and if so, we increase its associated count. If it is not stored, then we will eject some (p, c) pair from CSS, and replace it with the new pair. Under the GlobalHH semantics, we would apply the SS algorithm, and eject the pair with the least frequency. But in the CondHH algorithm, we attempt to find the pair with the lowest conditional probability, i.e. with the smallest value of $\widehat{\Pr}[c|p] = \hat{f}_{p,c}/f_p$. In our algorithm, we also keep track of the maximum value of $\hat{f}_{p,c}$ of any child of p that has been deleted so far, as m_p. When we remove a pair (p', c') from the data structure, we update $m_{p'}$ if needed, and insert the new pair (p, c) with estimated count $\hat{f}_{p,c} = m_p$. To find conditional heavy hitters, we use the estimated counts as in the previous algorithms.

Directly implementing this algorithm could be slow, due to the need to find the item with the lowest $\widehat{\Pr}[c|p]$ on each ejection. However, this can be made fast by keeping the data in an appropriate data structure. Specifically, we can index the stored parent-child pairs in a two-level data structure. For each parent, we keep its children stored in sorted ascending order of $\hat{f}_{p,c}$. This can be maintained efficiently using data structures based on doubly-linked lists as described in [5]. Then the parents are stored in sorted order of $\min_c(\hat{f}_{p,c})/f_p$, i.e. the estimated conditional probability of their least frequent child, via a standard data structure such as a heap. This structure means that we can quickly find the parent-child pair with the smallest overall estimated conditional probability, based on the observation that for each parent we only need to consider the probability of its least frequent child.

Whenever a child frequency is increased, we can quickly update the estimated $\hat{f}_{p,c}$, and ensure that the sortedness condition on the children is maintained. This update also affects f_p, and so may also require us to move the parent around to restore the heap property on $\min_c(\hat{f}_{p,c})/f_p$. Likewise, whenever a child of p is removed from this structure (because it is chosen for ejection), its successor in the sorted order of $\hat{f}_{p,c}$ becomes the new least frequent child of p, which may also require restoring the heap property. At the same time we can update m_p. Thus, implementing this algorithm requires a constant number of pointer operations and heap operations per update, and so can be done quickly.

Lemma 4: The CondHH algorithm guarantees that each candidate (p, c) pair output will have

$$\Pr[c|p] \leq \widehat{\Pr}[c|p] \leq \Pr[c|p] + \frac{m_p}{f_p}.$$

Proof: For each parent p, m_p is an upper bound on the maximum value of $\hat{f}_{p,c}$ of a (p, c) pair that has been deleted. Inductively, this is also an upper bound on any $f_{p,c}$ deleted (p, c) pair: this is true initially when $m_p = f_{p,c} = 0$ for all (p, c) pairs, and is maintained by every operation. Therefore, we have that whenever any new pair is inserted with $\hat{f}_{p,c} = m_p$, we have that $0 \leq f_{p,c} \leq \hat{f}_{p,c} = m_p$, and hence (while it remains in the data structure) $0 \leq \hat{f}_{p,c} - f_{p,c} \leq m_p$. Consequently we have $0 \leq \widehat{\Pr}[c|p] - \Pr[c|p] \leq m_p/f_p$. As with the previous algorithms, this tends to overestimate the true conditional probability, leading to higher recall, but weaker precision. This can be reversed by manipulating the estimate of $\hat{f}_{p,c}$, by subtracting m_p, to obtain a lower bound on $f_{p,c}$ and hence $\Pr[c|p]$. ∎

Note that in general this bound might not be very strong: we may see cases where $m_p = f_p$, and so we do not obtain a useful guarantee. However, in practice we expect to obtain useful guarantees for the popular conditional heavy hitters.

D. FamilyHH Algorithm

All the algorithms proposed above make the assumption that we can track detailed information for each parent (such as f_p, heavy hitter children for each p, etc.). However, this assumption is not always reasonable. For example, in the network traffic example in Section I, the number of parents is equal to the number of possible children (both are equal to the number of IP addresses, which is 2^{32} under IPv4). In some cases the number of parents actually observed in the data will be small enough to track exactly. But in general, we should also provide algorithms for when this is not so.

Our next algorithm generalizes GlobalHH, and so keeps sparse information about both parents and children by maintaining just the heavy hitter parents, and the heavy hitter parent-child pairs. So instead of tracking f_p, we will instead use $\hat{f}_p$, an approximate version of the frequency. The resulting FamilyHH algorithms uses two separate instances of the SS data structure, using space t and s respectively. To identify the candidate conditional heavy hitters, we pass through the heavy hitter parent-child pairs in order of (estimated) frequency, and from each find $\widehat{\Pr}[c|p] = \hat{f}_{p,c}/\hat{f}_p$. The space for this algorithm is bounded by $O(t+s)$.

Lemma 5: The FamilyHH algorithm guarantees that each candidate (p, c) pair output will have

$$\widehat{\Pr}[c|p] = \Pr[c|p] \pm 1/(\min(s,t)\Pr[p]).$$

Proof: From the properties of the heavy hitters algorithm, we have that $f_p \leq \hat{f}_p \leq f_p + n/t$ and $f_{p,c} \leq \hat{f}_{p,c} \leq f_{p,c} + n/s$. Consequently, we have

$$\begin{aligned}
\widehat{\Pr}[c|p] &= \frac{\hat{f}_{p,c}}{\hat{f}_p} \leq \frac{f_{p,c} + n/s}{f_p} = \Pr[c|p] + \frac{1}{s\Pr[p]} \\
\widehat{\Pr}[c|p] &= \frac{\hat{f}_{p,c}}{\hat{f}_p} \geq \frac{f_{p,c}}{f_p + n/t} = \frac{\Pr[c|p]}{1 + n/(f_p t)} \\
&\geq \Pr[c|p](1 - \frac{n}{f_p t}) = \Pr[c|p] - \frac{\Pr[c|p]}{t\Pr[p]} \\
&\geq \Pr[c|p] - \frac{1}{t\Pr[p]}
\end{aligned}$$

Based on this analysis, we choose to set $t = s$, so that the error bound is $\Pr[c|p] \pm 1/(s\Pr[p])$. ∎

This estimate may sometimes overestimate, and sometimes underestimate, depending on the errors of the component estimates. We can make it always overestimate or always underestimate by scaling these values appropriately.

E. SparseHH Algorithm

Our final, most involved, algorithm also keeps only approximate information on the set of parents. We call this the SparseHH algorithm, as it keeps only sparse information on the parents. For the parent-child pairs, we make use of the CSS structure from the CondHH algorithm, which attempts to retain those pairs with the highest conditional probability. However, now that we are retaining only a subset of the parents, we need to modify this slightly. We will aim to maintain frequency information on only those parents that have an active child in the data structure. Now, when we come to insert a new parent-child pair p, c into CSS, we must decide what bound on the frequency to use. We suggest some possibilities for this "reintroduction strategy":

- *Global.* Maintain a global value m on the max (estimated) frequency of any (p, c) pair that has been deleted so far.
- *Ancestor.* If there is some reason to believe that parents with similar labels (e.g. in a hierarchy) have similar frequency, then we can maintain a small number g of different groups of parents, and retain for each the maximum frequency of any (p, c) deleted that came from that group. For example, if the p values are drawn from a hierarchy, we can choose a high level in the hierarchy, and create groups based on this.
- *Hash Partition.* For other cases, we can create a random partitioning of parents into g groups based on a hash function, and maintain the maximum frequency of any (p, c) pair belonging to that group. In the case of a single group, this becomes equivalent to the global strategy.
- *Bloom Filter.* We can keep a compact summary of items deleted with high values of $\hat{f}_{p,c}$, say, in the form of a Bloom Filter [31]. That is, when we delete a pair with frequency $\hat{f}_{p,c}$, we compute an index from this as $i = \lceil \log_b \hat{f}_{p,c} \rceil$ for a parameter b (for concreteness, consider $b = 2$). Then we insert (p, c) into a Bloom Filter indexed by i. When a new pair (p, c) is inserted into the data structure, we scan through the Bloom Filters, testing if (p, c) is present in each. If the test indicates it is in the ith Bloom Filter, then we instantiate $\hat{f}_{p,c} = b^i$. Note that Bloom Filters may lead to false positives: in this case, we will result in an overestimate of the frequency. This may lead to false positives in the estimated conditional heavy hitters, but will ensure that we do not underestimate the conditional probability of any pair.

Depending on the circumstances, any of these reintroduction strategies may be better, and indeed, we can run multiple of these in parallel, and choose the one that gives the smallest estimated value of $\hat{f}_{p,c}$ each time. There are several other implementation choices for SparseHH:

- Different reintroduction strategies may offer either upper or lower bounds on estimated counts; upper bounds favor recall, while lower bounds favor precision.
- We divide the memory available to the algorithm into two pieces: the memory used for the main tracking of counts (which in turn is split into information kept for parents and for approximate (p, c) frequencies); and the memory used for estimating counts when an item is introduced into the structure. We use a parameter ρ to describe this split: a ρ fraction of the available memory is given to the main structure, and $1 - \rho$ to the reintroduction structure.

We compare these choices empirically in Section V.

F. Discussion

We have proposed a variety of algorithms. They fall into two main classes: those that keep some information about each parent (GlobalHH, ParentHH, and CondHH); and those that do not (FamilyHH and SparseHH). Each algorithm aims to accurately approximate the conditional probability of pairs,

based on different priorities for what information to retain given limited space. Among these, we are most interested in the behavior of CondHH and SparseHH, since these most directly capture the nature of conditional heavy hitters by focusing on the (estimated) conditional probability of items. Meanwhile, GlobalHH, ParentHH and FamilyHH are based on the raw frequencies of items. A priori, it is unclear which algorithm will perform best for the task of retrieving conditional heavy hitters from a stream, so we will compare them empirically to determine the relative performance.

V. Experimental Results

All our experiments were conducted on a single 2.67GHz core of a Linux server with a large total amount of available memory. In evaluating the quality of our algorithms for recovering conditional heavy hitters, we make use of several measures of accuracy:

- The Precision and Recall of the recovered conditional heavy hitter pairs relative to the "true" set that are greater than a threshold $\Pr[c|p] \geq \phi$ (Definition 3);
- The Precision of the top-τ popular conditional probabilities (Definition 4)[1];
- The Average Precision for the popular conditional probabilities, where the average is taken over all top-r sets of popular conditional heavy hitters, for $r = 1, 2, ..., \tau$.

A. Data analysis and Experimental setup

We applied the above algorithms for several real and artificial datasets, namely (1) simulated Markov chains, to estimate the largest elements of the matrix of transition probabilities; (2) requests made to the World Cup 1998 Web site to detect conditional heavy hitters between clientID and objectID of the requests; (3) GPS trajectories of taxis in San Francisco to detect the most probable position of the vehicle taking into account two previous positions. We describe the datasets in more detail in the following sections. ϕ was chosen in each case according to the characteristics of the data in order to have reasonable number of conditional heavy hitters. We distinguish between cases where the data is sparse—few parents have conditional heavy hitter children—and dense—most parents have conditional heavy hitter children.

Markov chain artificial data. As discussed in the Introduction, one application of finding the conditional heavy hitters is to model the transition probabilities of a Markov chain. The goal is then to estimate the entries in this transition probability matrix, by finding the large values (and assuming the rest to be uniform). To model a Markov chain of order k, we concatenate the k most recent observations together to form a parent, and take the next observation as the corresponding child. In general, given an alphabet A, it is not feasible to track all the $|A|^{k+1}$ transition probabilities exactly, due to the high resource costs to do so. Hence, we instead use our algorithms to find and estimate the highest and the most important elements of transition probability matrix. In our experiments we use an alphabet size $|A| = 10^3$ and model a Markov chain of order $k = 2$. This means there are one million parents and one billion possible parent-child pairs.

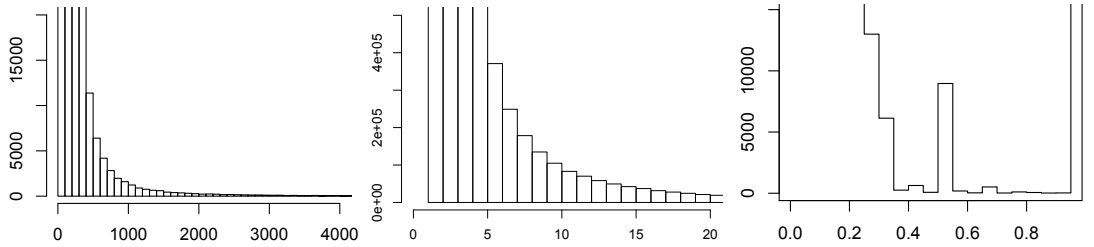
We use two types of generation process for the data. The first case generates "dense" sequences so that each parent (P) has exactly one "heavy" child (C_h) with conditional probability chosen (randomly) to be greater than 0.6. The rest of the probability mass is uniformly distributed among the other possible edges. More formally, $\forall P \in A \times A$, $\exists C_h \in A$, such that $\Pr[C_h|P] \geq 0.6$ while $\Pr[C_i|P] = \frac{1-\Pr[C_h|P]}{|A|-1}$, where $C_i \in A$, $C_i \neq C_h$. In this setting, there are 1 million conditional heavy hitters out of 1 billion possibilities.

The second generation process produces a "sparse" sequence with a predefined number of conditional heavy hitters that is smaller than the number of parents. We identify a subset of parents to have one or more heavy children. We determine the number of heavy children n_c for a "heavy" parent by picking n_c from a truncated normal distribution with mean 3 and standard deviation 2. In our experiments we created a total of 200K conditional heavy hitters, so on average, only 1 in 15 parents has conditional heavy hitter children. Each "heavy" parent shares a total transition probability equal to 0.8 among its conditional heavy hitter children. The rest of the probability mass 0.2 is divided uniformly among the other edges. More formally, if a parent P is chosen to be heavy, then we pick n_c children C at random, and set their transition probabilities $\Pr[c \in C|P] = 0.8/n_c$, while for the others $\Pr[c \notin C|P] = 0.2/(|A| - n_c)$. We set the number of conditional heavy hitters to recover as the true number of conditional heavy hitters, i.e. 200K.

Taxicab GPS data. The Taxicab data consists of about 20 million GPS points for a fleet of taxis, collected over the course of a month, obtained from `cabspotting.org`. To go from the fine-grain GPS locations to streams of values, we performed preprocessing to clip the data to a bounded region and coarsen to a grid. The region of the measurements is restricted to a rectangle in the area of San Francisco, with latitude in the range $[37.6...37.835]$, which covers 26km and longitude in the range $[-122.52... - 122.35]$, which covers 15km. This clipping was performed to remove a few incorrect readings which were far outside this region.

This space was partitioned into 10,000 rectangles using a 100×100 grid. Given the readings within this grid, we proceeded to define trajectories from the data as a sequence of grid cells occupied by the same cab. We considered a new trajectory to begin if there was a gap of more than 30 minutes between successive observations. Following this definition, we extracted 54,308 trajectories. We model the trajectory data as a second order Markov chain, on the grounds that knowing the previous two steps is likely to be indicative of where the next step will take us. A first order model would only have the previous location, and so would not capture in what direction the vehicle was traveling. This model generates around 160,000 distinct parents and a million distinct

[1]Note that when restricting output to have size exactly τ, precision and recall are identical, so we do not duplicate this measurement.



(a) Parent frequencies (b) Parent-child pair frequencies (c) Conditional heavy hitter probabilities

Fig. 1. Descriptive statistics for WorldCup'98 data.

parent-child pairs; our experiments with finer grids (omitted for brevity) had even higher dimensionality. With a default ϕ value of 0.8, we observed 63,721 conditional heavy hitters. This means that about 2 out of every 5 parents have conditional heavy hitter children, so we consider this a dense dataset.

Worldcup'98 data. The Worldcup data[2] contains information about the requests made to the World Cup Web site during the 1998 tournament. Each request contains a ClientID (a unique integer identifier for the client that issued the request) and an ObjectID (a unique integer identifier for the requested URL). We are interested in finding conditional heavy hitters between ClientID and ObjectID pairs, where ClientID is treated as the parent, and ObjectID as the child. That is, we are interested in detecting (ClientID, ObjectID) pairs where the requested child is particularly popular for that user.

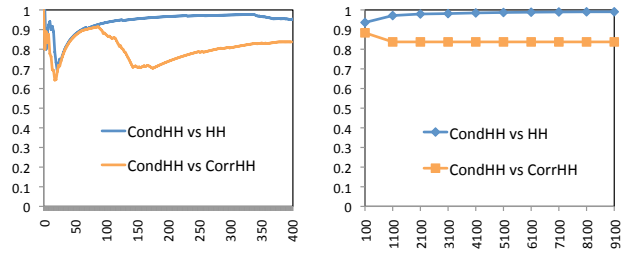
We used data from day 41 to day 46 of the competition. The total number of records in this period is around 105 million; the number of distinct parent-child pairs is around 59 million; and the number of distinct parents is 540K. In this data we look for the conditional heavy hitters that have a probability of occurrence greater then $\phi = 0.25$. The total number of such conditional heavy hitters is in excess of fifty thousand. About 1 in 10 parents has a conditional heavy hitter child, making this data relatively sparse.

The frequency distributions are skewed: there are many parents and parent-child pairs that are found only once or a small number of times in the dataset (Figure 1(a) and Figure 1(b)). Although most of the parent-child pairs occur once—52 million out of the 59 million distinct pairs—still, there are many pairs that occur a greater number of times. The distribution of probabilities of conditional heavy hitters is shown in Figure 1(c), and shows that there is sufficient probability mass associated with higher conditional probabilities.

B. Comparison with simple and correlated heavy hitters.

Our first set of experiments demonstrate that our definition of conditional heavy hitters is distinct from the traditional definition of heavy hitters and correlated heavy hitters, and therefore we do need a new approach to recovering them. For this study, we computed the exact set of the k most significant heavy hitters, correlated heavy hitters (setting the parameter to detect a heavy hitter as $\psi = 0.00001$) [21] and

[2]http://ita.ee.lbl.gov/html/contrib/WorldCup.html



(a) k from 1 to 400. (b) k from 100 to 9100.

Fig. 2. Jaccard distance for top-k conditional (CondHH) and simple (HH) or correlated(CorrHH) heavy hitters

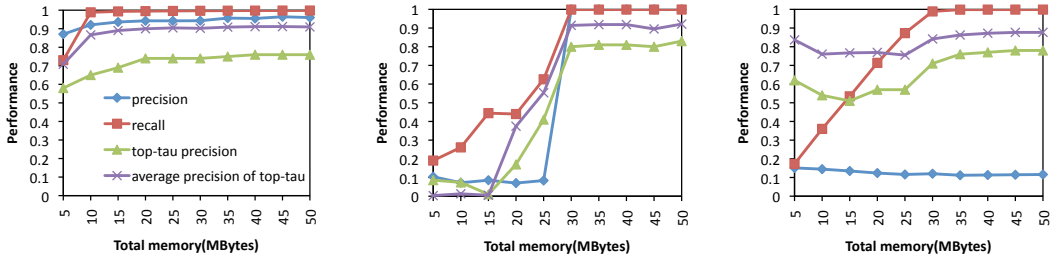
conditional heavy hitters (for $\phi = 0.25$), and computed the Jaccard distances between conditional and simple heavy hitters and conditional and correlated heavy hitters. Jaccard distance between two sets A and B is $1 - \frac{|A \cap B|}{|A \cup B|}$. The distance is 0 if $A = B$, and is 1 if the two sets are completely disjoint.

The results of this comparison on the WorldCup data are shown in Figure 2. The plot is broken into two pieces to aid readability: Figure 2(a) shows the distance for k up to 400, while Figure 2(b) shows the results as k grows up to 9000. From these plots we see that the distances between the two sets does not decrease below 0.6. In the low range (Figure 2(a)), the average distance between conditional and simple heavy hitters is 0.94, and it approaches 1 as k increases. This indicates that the two definitions are truly describing distinct phenomena, due to the high distance between the sets. The average distance between conditional and correlated heavy hitters in the low range is 0.8 and it approaches 0.84 as k increases. Though these two definitions are more similar than the previous case, they still describe two different phenomena as the distance between the sets is high. We note in passing that the results of conditional and correlated heavy hitters become increasingly similar as the parameter ψ approaches zero.

C. Parameter setting for SparseHH

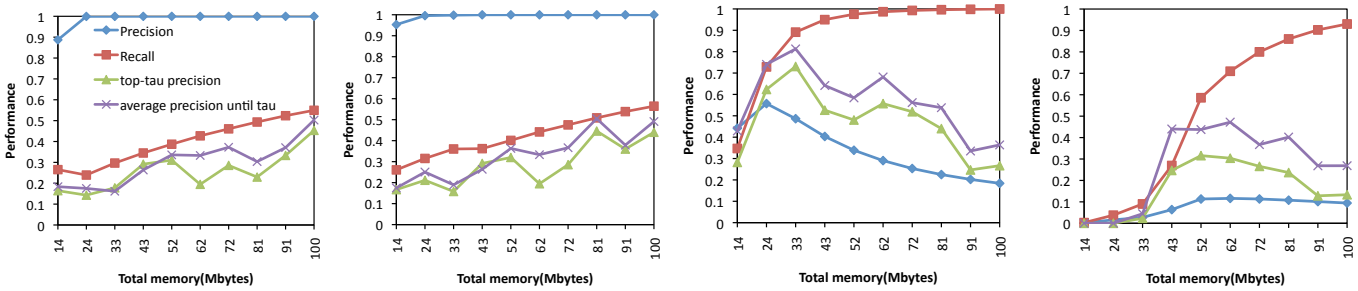
The SparseHH algorithm has several parameters and choices that affect its performance. Here, we investigate how to set these parameters before comparing with other algorithms.

Choice of reintroduction strategy. We compare the different choices of reintroduction strategy: hash partitioning, ancestor, and Bloom Filter. Figure 3 shows the accuracy over the



(a) Hash Partition strategy (b) Ancestor strategy (c) Bloom Filter strategy

Fig. 3. SparseHH accuracy for conditional heavy hitter recovery on Worldcup data under different reintroduction strategies.



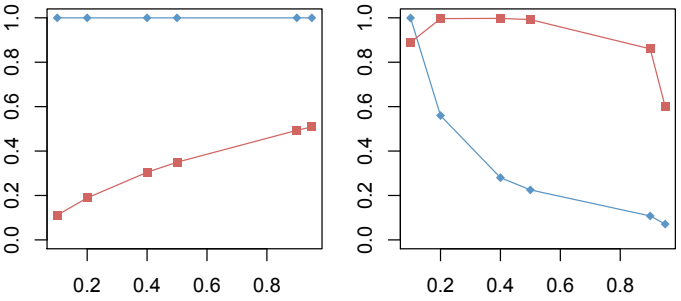
(a) Favor precision, $\rho = 0.9$. (b) Favor precision, $\rho = 0.95$. (c) Favor recall, $\rho = 0.5$. (d) Favor recall, $\rho = 0.9$.

Fig. 4. Accuracy on sparse synthetic data using SparseHH.

Worldcup data, where we set $\tau = 100$ and $\phi = 0.25$ to define the (popular) conditional heavy hitters.

Here, the ratio of memory allocated to the main structure, ρ, was set to 0.9, with the remainder used to help reintroduce items to the data structure. We observe that the hash partitioning strategy performs the best across all metrics (Figure 3(a)). The ancestor strategy can obtain good results, but only when a larger total amount of memory is made available (Figure 3(b)). The Bloom Filter strategy, while achieving high recall, always has very poor precision (Figure 3(c)). Based on this and other results, we adopt the Hash partitioning strategy as the method of choice for SparseHH: while the Ancestor method is sometimes competitive, this can be seen as a special case of the Hash partition method with a structured choice of hash function, so we do not further distinguish these methods.

Choice of memory ratio. As noted in Section IV-E, we can adjust the estimated counts in the algorithm to give either upper or lower bounds, and hence to favor precision or to favor recall. We compared the impacts of this choice in our experiments, shown in Figure 4 for the sparse synthetic data. We set $\phi = 0.05$, sufficient to distinguish the conditional heavy hitters from the other pairs. In the plots, we pick a representative selection of parameter settings, as we vary the ratio ρ that governs the division of memory between the main and reintroduction structures, and whether the algorithm favors precision or recall. Across these, we first observe that this choice does indeed behave as advertised: favoring precision obtains near perfect precision, while favoring recall allows recall to grow as total memory increases. However, when

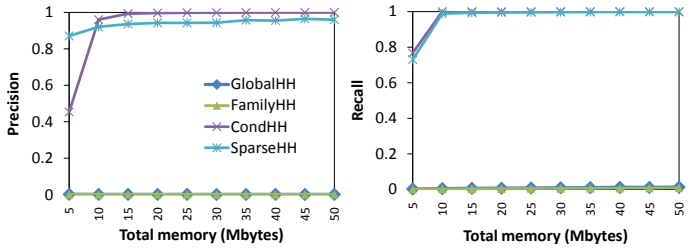


(a) Favor precision (b) Favor recall

Fig. 5. Precision (blue diamonds) and Recall (red squares) of SparseHH variations on sparse synthetic data as ρ varies.

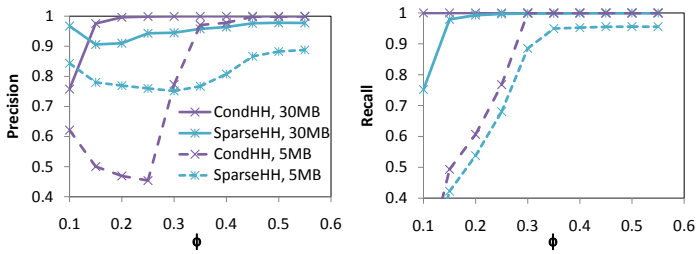
we favor precision, recall tends to improve as we allocate more memory (Figures 4(a) and 4(b)), while favoring recall tends to cause precision to drop off as more memory is used (Figures 4(c) and 4(d)).

To investigate this further, we fix the available memory, and vary the ratio ρ. The results on the same data are shown in Figure 5. We observe that when we favor precision, the precision is always near perfect (Figure 5(a)). The benefit of giving more memory to the main structure outweighs the loss from reducing space for the reintroduction strategy, so a large ρ value gives the best recall. Contrarily, favoring recall has generally good recall, but gets the best precision when almost all of the memory is turned over to the reintroduction strategy (Figure 5(b)). Still, it is hard to obtain both good recall and



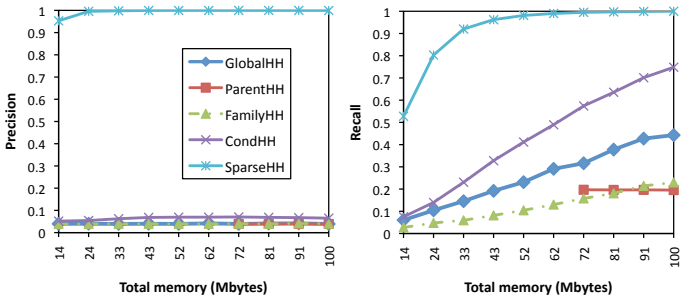
(a) Precision (b) Recall

Fig. 6. Precision and Recall on the WorldCup data.



(a) Precision (b) Recall

Fig. 7. Accuracy as ϕ varies on Worldcup data



(a) Precision (b) Recall

Fig. 8. Accuracy on sparse synthetic data as memory varies

good precision from this strategy: although we see some good behavior for very small values of ρ here, this was not stable across other datasets. Consequently, we conclude that it is preferable to favor precision, and adopt this with $\rho = 0.9$ as the default in all other experiments.

D. Performance on sparse data

We now compare all the proposed algorithms, initially on sparse data, and subsequently on more dense data.

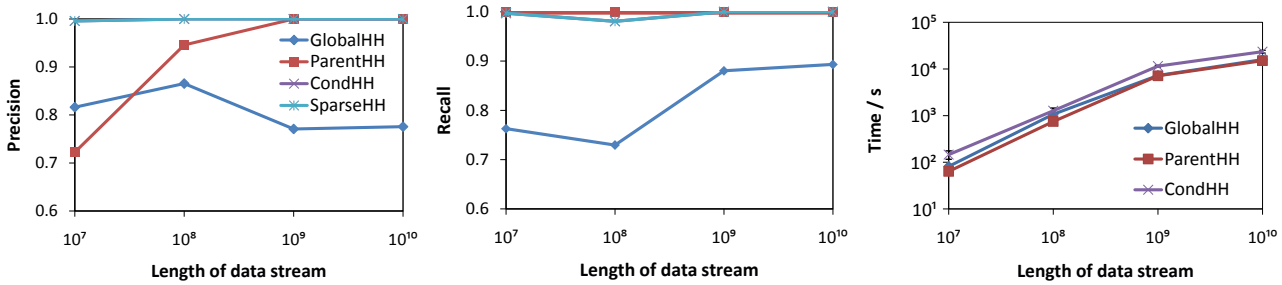
World Cup Data. We present results for recovering (ClientID, ObjectID) conditional heavy hitters from the (relatively sparse) Worldcup data. In other experiments, we also looked for correlations on other attribute combinations, such as (ServerID, ObjectID). The results there were broadly similar, and so are omitted for brevity.

Figure 6 shows results on precision and recall for recovering the conditional heavy hitters for this data. Here, the CondHH and SparseHH (using Hash partition reintroduction) methods perform the best for both precision and recall. These two algorithms both make use of an eviction strategy that picks the parent-child pair with the lowest (estimated) conditional probability to be deleted from the main structure. This observation suggests that such a pruning strategy can be effective at retaining the most promising pairs in memory. For this data, the number of parents is not so large, and so it is feasible to retain information on all parents. Thus, CondHH is not penalized for this choice here, although we see examples later where there are too many parent items to track effectively. These methods also achieved high top-τ precision, over 0.8, indicating over 80% agreement between the top 100 reported conditional heavy hitters and the true most popular heavy hitters. On this data, we observe that other approaches suggested—GlobalHH FamilyHH and ParentHH (omitted from the plots)—are unable to provide useful results: although they provide accuracy guarantees as a function of the space available, it turns out that these guarantees do not become useful until much more memory is available. In this case, the successful algorithms (CondHH and SparseHH) are able to achieve near-perfect precision and recall using less than 10% of the memory required to represent the data exactly.

Figure 7 shows the accuracy of CondHH and SparseHH as we vary ϕ, the threshold for defining a conditional heavy hitter. We see that for large ϕ values and moderate memory (30MB), CondHH is preferable, and achieves near-perfect precision and recall. As ϕ is decreased, there are more conditional heavy hitters to recover, and when memory is constrained to only 5MB (the dashed lines), recall necessarily falls: the algorithms are unable to retain information about all conditional heavy hitters. However, in the low ϕ, low memory setting, SparseHH is able to maintain higher precision, while the precision of CondHH falls off.

Sparse Synthetic Data. We now compare all the algorithms on the truly sparse synthetic data, for a stream of length 10^8. This data has a much smaller number of conditional heavy hitters compared to the number of parent items. Consequently, we expect the algorithms which try to keep information on all parents to perform poorly here, since this will occupy most of their available resources.

This conjecture is confirmed in Figure 8: only SparseHH is able to obtain both good precision and good recall for the range of memory provided. It also has accuracy as measured by top-τ precision and average precision up to τ: both around 0.9 (plots omitted for space reasons). Among the other algorithms, CondHH shows the best improvement in recall as more memory is made available, with GlobalHH and FamilyHH improving more slowly (Figure 8(b)). The ParentHH algorithm can only produce results when enough memory is available to keep a (very small) summary structure for each parent—in this case, above 72MB. Interestingly, the precision performance of all algorithms apart from SparseHH is very poor: much more memory is needed before these can achieve good precision (Figure 8(a)). This is in part because even the highest amount of memory shown in Figure 8 represents less than 5% of the



(a) Precision (b) Recall (c) Time

Fig. 10. Accuracy and timing results for algorithms on dense synthetic data

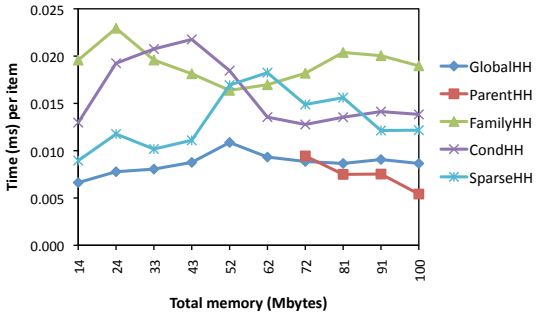
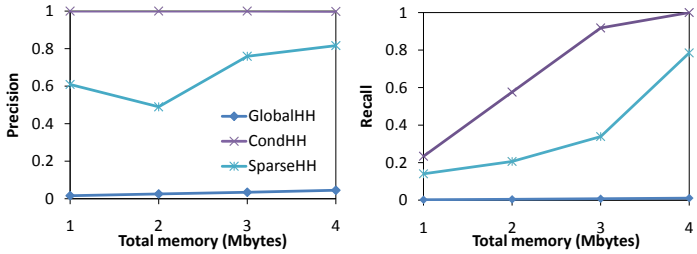


Fig. 9. Time cost of algorithms on sparse synthetic data as memory varies



(a) Precision (b) Recall

Fig. 11. Accuracy on Taxicab data as memory varies

space to record the exact statistics for the given data. In terms of the original application, of approximating the Markov chain transition matrix, the results are also strong: the L_1 difference between the distributions is about 0.01, where 0 would be perfect recovery, and 1 represents the worst case. We conclude that over sparse data, the SparseHH algorithm has the best performance and is the method of choice.

In terms of the time cost of the algorithms, Figure 9 shows that there is little systematic variation as a function of the size of the summary structure. The simpler GlobalHH and ParentHH algorithms are the faster ones, but all algorithms have performance measured in the hundreds of thousands of updates per second to process 10^8 items.

E. Performance on dense data

Dense synthetic data. In the dense synthetic data, each parent has at least one child that is a conditional heavy hitter. We generate a stream of data from the 2nd order Markov chain of different lengths, between 10^7 and 10^{10} observations. We allocate an amount of space equivalent to twice the number of possible parent items, and evaluate their accuracy in terms of precision and recall on streams of varying lengths. With the threshold $\phi = 0.5$, Figure 10 shows the average time and accuracy achieved over 10 independently chosen streams. The observed standard deviation over these repetitions was very low, around 10^{-3} for all precision and recall computations.

The results show that the GlobalHH algorithm performs poorly, with only moderate precision and recall on this relatively "easy" data set (Figures 10(a) and 10(b)). The ParentHH algorithm has near perfect recall, and precision improves as the stream gets longer (and so the signal becomes easier to detect). However, again, the CondHH algorithm has the best accuracy, getting near perfect precision and recall throughout. The SparseHH algorithm had identical results to CondHH on this data. On closer inspection of the data structures, we observed that this was because SparseHH has sufficient memory to keep frequency information on all parents. Consequently, it can store the same information as CondHH and so finds the same estimated frequencies. For similar reasons FamilyHH kept the same information as GlobalHH and so is omitted from the plots. In terms of scalability, all algorithms are similar. Figure 10(c) shows that the CondHH algorithm is slightly slower in our implementation, due to the more involved data structure maintenance process. However, the difference is not substantial, and could be improved by a more engineered solution. Even here, the throughput is nearly half a million updates per second on a single core.

Taxicab data. The Taxicab data is quite dense: many parents have a conditional heavy hitter child. Figure 11 provides precision and recall results on this data for $\phi = 0.8$. As in other experiments, GlobalHH does not provide useful recovery of conditional heavy hitters with such low memory. SparseHH achieves good precision, but CondHH has enough memory to obtain perfect precision (Figure 11(a)). The story is similar for recall: SparseHH improves as more memory is available, but is consistently dominated by CondHH, until SparseHH is given enough memory to store all parents. Moreover, for the top-τ precision the results for CondHH were much stronger,

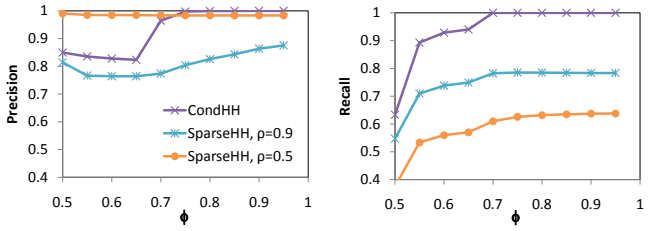


Fig. 12. Accuracy as ϕ varies on Taxicab data

approaching 1, while SparseHH achieved only 0.25.

To better understand the relative behavior of these two competitive algorithms, Figure 12 shows the case as we vary ϕ, the threshold for conditional heavy hitters, while holding the total memory constant at 4MB. As ϕ decreases, there are more pairs passing the threshold, and so the problem becomes harder. The precision of SparseHH tends to remain constant, while there is a more notable dip in the precision of CondHH. Interestingly, adjusting the memory available for the reintroduction strategy of SparseHH by adjusting ρ has a marked effect: putting more memory to this end improves precision, but reduces recall. We conclude that for dense data, CondHH is the method of choice, provided we can afford to store all parents.

VI. Concluding Remarks

In this paper, we have introduced the notion of conditional heavy hitters as a useful concept that is distinct from prior notions of heavy hitters, correlated heavy hitters and frequent itemsets. We introduced a sequence of algorithms that build on existing techniques, but target the new definition. Our empirical study demonstrated that among these, it is those that most directly target the new definition, by preferentially retaining items with high (estimated) conditional probability and pruning those with low conditional probability, that perform the best. Specifically, the SparseHH algorithm, which keeps an approximate summary of both the parent-child pairs as well as the parent items, generally performs the best across a range of sparse datasets and parameter settings. In particular, it achieves high precision and recall on the set of conditional heavy hitters while retaining only 5-10% of the space of storing exact statistics. When the data is more dense and there is sufficient memory, CondHH is the preferred method. If we do not know the nature of the data in the advance, we can simply run SparseHH, since it will keep information on parents exactly while there is room, and so behave more like CondHH. Future work will identify further applications for conditional heavy hitters, and evaluate their efficacy in those settings. Our algorithms are defined in the streaming model, which captures the challenging case of high-speed arrival of data. As the scale of data increases, it will become necessary to adapt these algorithms to a distributed setting, where multiple streams are observed, and the collected summaries can be combined to give a summary of the union of all the input data.

References

[1] G. Cormode and M. Hadjieleftheriou, "Finding frequent items in data streams," in *VLDB*, 2008.

[2] N. Manerikar and T. Palpanas, "Frequent items in streaming data: An experimental evaluation of the state-of-the-art," *Data Knowl. Eng.*, vol. 68, no. 4, pp. 415–430, 2009.

[3] G. Manku and R. Motwani, "Approximate frequency counts over data streams," in *VLDB*, 2002, pp. 346–357.

[4] J. Misra and D. Gries, "Finding repeated elements," *Science of Computer Programming*, vol. 2, pp. 143–152, 1982.

[5] A. Metwally, D. Agrawal, and A. E. Abbadi, "Efficient computation of frequent and top-k elements in data streams," in *ICDT*, 2005.

[6] A. Arasu and G. S. Manku, "Approximate counts and quantiles over sliding windows," in *ACM PODS*, 2004.

[7] L. Lee and H. Ting, "A simpler and more efficient deterministic scheme for finding frequent items over sliding windows," in *ACM PODS*, 2006.

[8] M. Charikar, K. Chen, and M. Farach-Colton, "Finding frequent items in data streams," in *ICALP*, 2002.

[9] G. Cormode and S. Muthukrishnan, "An improved data stream summary: The count-min sketch and its applications," *J. Alg*, vol. 55, no. 1, pp. 58–75, 2005.

[10] B. Boyer and J. Moore, "A fast majority vote algorithm," Institute for Computer Science, U. Texas, Tech. Rep. ICSCA-CMP-32, Feb. 1981.

[11] E. Demaine, A. López-Ortiz, and J. I. Munro, "Frequency estimation of internet packet streams with limited space," in *ESA*, 2002.

[12] R. Agrawal, T. Imielinski, and A. N. Swami, "Mining association rules between sets of items in large databases," in *ACM SIGMOD*, 1993, pp. 207–216.

[13] J. Han, J. Pei, and Y. Yin, "Mining frequent patterns without candidate generation," in *ACM SIGMOD*, 2000, pp. 1–12.

[14] J. H. Chang and W. S. Lee, "Finding recent frequent itemsets adaptively over online data streams," in *KDD*, 2003, pp. 487–492.

[15] C. Giannella, J. Han, J. Pei, X. Yan, and P. S. Yu, "Mining frequent patterns in data streams at multiple time granularities," in *Data Mining: Next Generation Challenges and Future Directions*, 2004.

[16] P. S. Yu and Y. Chi, "Association rule mining on streams," in *Encyclopedia of Database Systems*. Springer-Verlag, 2009, pp. 136–139.

[17] L. Lee and H. Ting, "A simpler and more efficient deterministic scheme for finding frequent items over sliding windows," in *ACM PODS*, 2006.

[18] G. Cormode, F. Korn, and S. Tirthapura, "Time decaying aggregates in out-of-order streams," in *ACM PODS*, 2008.

[19] F. I. Tantono, N. Manerikar, and T. Palpanas, "Efficiently discovering recent frequent items in data streams," in *SSDBM*, 2008, pp. 222–239.

[20] S. Venkataraman, D. X. Song, P. B. Gibbons, and A. Blum, "New streaming algorithms for fast detection of superspreaders," in *Network and Distributed System Security Symposium NDSS*, 2005.

[21] B. Lahiri and S. Tirthapura, "Finding correlated heavy-hitters over data streams," in *IPCCC*, 2009, pp. 307–314.

[22] M. Dallachiesa, B. Nushi, K. Mirylenka and T. Palpanas, "Uncertain Time-Series Similarity: Return to the Basics," in *PVLDB*, vol. 5, no. 11, 2012, pp. 1662–1673.

[23] G. Cormode, F. Korn, S. Muthukrishnan, and D. Srivastava, "Finding hierarchical heavy hitters in data streams," in *VLDB*, 2003, pp. 464–475.

[24] J. Gehrke, F. Korn, and D. Srivastava, "On computing correlated aggregates over continual data streams," in *ACM SIGMOD*, 2001, pp. 13–24.

[25] L. E. Baum and T. Petrie, "Statistical Inference for Probabilistic Functions of Finite State Markov Chains," *The Annals of Mathematical Statistics*, vol. 37, no. 6, pp. 1554–1563, 1966.

[26] J. Pearl, *Probabilistic reasoning in intelligent systems: networks of plausible inference*. Morgan Kaufmann Publishers Inc., 1988.

[27] J. Letchner, C. Re, M. Balazinska, and M. Philipose, "Approximation trade-offs in markovian stream processing: An empirical study," in *ICDE*, 2010, pp. 936–939.

[28] P. Wang, H. Wang, and W. Wang, "Finding semantics in time series," in *ACM SIGMOD*, 2011, pp. 385–396.

[29] A. E. Raftery, "A model of high-order markov chains," *Journal of the Royal Statistical Society (Series B Methodological)*, vol. 47, no. 3, pp. 528–539, 1985.

[30] N. Alon, Y. Matias, and M. Szegedy, "The space complexity of approximating the frequency moments," in *ACM STOC*, 1996, pp. 20–29.

[31] A. Broder and M. Mitzenmacher, "Network applications of bloom filters: A survey," *Internet Mathematics*, vol. 1, no. 4, pp. 485–509, 2005.