The VLDB Journal (2015) 24:395–414
DOI 10.1007/s00778-015-0382-5

REGULAR PAPER

Conditional heavy hitters: detecting interesting correlations in data streams

Katsiaryna Mirylenka · Graham Cormode · Themis Palpanas · Divesh Srivastava

Received: 24 March 2014 / Accepted: 14 February 2015 / Published online: 26 February 2015

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 explore the relationship between conditional heavy hitters and other related notions in the literature, and show analytically and experimentally the usefulness of our approach. We introduce several algorithm variations that allow us to efficiently find conditional heavy hitters for input data with very different characteristics, and provide analytical results for their performance. Finally, we perform experimental evaluations with several synthetic and real datasets to demonstrate the efficacy of our methods and to study the behavior of the proposed algorithms for different types of data.

K. Mirylenka (✉)
The University of Trento, Trento, Italy
e-mail: kmirylenka@disi.unitn.it

G. Cormode
The University of Warwick, Coventry, UK
e-mail: G.Cormode@warwick.ac.uk

T. Palpanas
Paris Descartes University, Paris, France
e-mail: themis@mi.parisdescartes.fr

D. Srivastava
AT&T Labs, Bedminster, NJ, USA
e-mail: divesh@research.att.com

Keywords Streaming data · Online algorithms · Heavy hitters

1 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 [10,25]. 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 [3,7,9,13,23,26,27,29].

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 file sharers). As a result, tracking the heavy hitters within these 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 among 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 online 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 that between 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, 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.
- Caching and pre-fetching are widely used in Web-based systems. Their aim is to reduce the time latencies perceived by users when navigating the Web. Advanced caching and pre-fetching policies use probability graphs, access trees, and Markov chain models [31], in order to predict the least/most probable objects to be accessed next or in the near future. In this case, conditional heavy hitters can efficiently capture the main transitions from previous sequence of objects to the next, thus providing existing policies with the most relevant real-time probabilities to enable them to make better predictions.
- Within a database management system or data warehouse, there are a large number of transactions that 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 compare conditional heavy hitters with other notions of interesting elements studied in the data stream literature, such as frequent itemsets, association rules, and correlated heavy hitters;
- We develop and describe several streaming algorithms for retrieving conditional heavy hitters, and we 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 characteris-

tics 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: Sect. 2 describes background and related work for heavy hitters and frequent itemset mining, and Sect. 4 discusses the relationship between conditional heavy hitters and other notions of interesting elements in a data stream. In Sect. 3, we refine the notion of conditional heavy hitters to obtain a workable definition. In Sect. 5, we present and discuss a sequence of algorithms to find conditional heavy hitters from a stream of data. Our experimental results are shown in Sect. 6, and we conclude our discussion in Sect. 7.

2 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 [5,29], 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 [16,26,27], and randomized "sketch" methods, capable of handling negative weights [9,13]. 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 [10,25].

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 [1] and FP-Tree algorithms [21]. These concepts have been adapted to work over streams of data, generating algorithms such as FUP [8] and FP-stream [20]. A limitation of finding frequent itemsets is that the number of possibly frequent itemsets can become very large, meaning that the algorithm has to either 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 trade-offs 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 [38].

Several other variations in 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 [12,15,23,34]. 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 [35].

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 [11]. Lastly, correlated aggregates consider streams of tuples and ask for aggregates on some attributes, for the subset of tuples that meet some other conditions [19]. 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.

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 [14,24]. 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 [4], Bayesian networks [30], and others [24,36]. 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 [32] 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.

Most related to this paper is the work of Lahiri and Tirthapura [22] 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. We discuss the similarity and differences of this definition to conditional heavy hitters in Sects. 3 and 4.

3 Preliminaries

To allow our definition of conditional heavy hitters to be generally applicable, we assume that the input can be modeled as a stream of pairs of adjacent (parent, child) values $(\mathbf{p}, c)$. A parent $\mathbf{p}$ can be a single symbol, or a sequence of adjacent symbols in the stream, while a child is a single symbol.

Definition 1 (*Frequencies*) Given a stream of (parent, child) pairs whose ith element is $(\mathbf{p}_i, c_i)$, the frequency of a parent $\mathbf{p}$, $f_{\mathbf{p}}$, is defined as

$$f_{\mathbf{p}} = |\{i : \mathbf{p}_i = \mathbf{p}\}|.$$

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

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

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 $\mathbf{p}$, $\Pr[\mathbf{p}]$, is defined as $\Pr[\mathbf{p}] = f_{\mathbf{p}}/n$. The joint probability of a parent–child pair, $\Pr[\mathbf{p}, c]$, is defined as $\Pr[\mathbf{p}, c] = f_{\mathbf{p},c}/n$. The conditional probability of a child given a parent, $\Pr[c|\mathbf{p}]$, is defined as

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

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

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

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 $\mathbf{p}$ that occurs only once in the stream. Then we have $f_{\mathbf{p},c} = f_{\mathbf{p}} = 1$ for the associated child c, and so $\Pr[c|\mathbf{p}] = 1$, 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 $\mathbf{p}$ 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 $\Pr[\mathbf{p}] > \psi$ for $(\mathbf{p}, c)$ to form a conditional heavy hitter (similar to [22]). 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 $\Pr[(\mathbf{p}, c)] > \phi\psi$, have similar drawbacks. Consequently, we set up a different requirement as our goal.

Definition 4 (*Popular conditional heavy hitter*) A pair $(\mathbf{p}, c)$ 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 $f_{\mathbf{p},c}$.

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 datasets, we may expect that there will be many conditional heavy hitters with large $f_{\mathbf{p},c}$ values, which will ensure that we avoid the trivial case of $f_{\mathbf{p},c} = f_{\mathbf{p}} = 1$. 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 that 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 that finds popular conditional 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 previously 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 [2], 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 that 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 are 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 that allow us to estimate the conditional probability of parent–child pairs accurately. That is, the goal is to find an estimate $\widehat{\Pr}[c|\mathbf{p}]$ that accurately estimates $\Pr[c|\mathbf{p}]$. From this, we will be able to find candidate conditional heavy hitters. Having the candidate conditional heavy hitters, we can also extract the popular conditional heavy hitters. Our experimental study evaluates the ability of these algorithms to find such conditional heavy hitters.

4 Notions of elements of interest in a data stream

The problem of detecting "interesting" elements in data streams has attracted a lot of attention in the recent years. There can be many different interpretations of what makes an element of interest, varying across different applications. Consequently, several different notions have been proposed that are relevant to our study. In each case, there has been at least one paper describing algorithms for each of the following notions

(a) frequent items or heavy hitters [10,25];
(b) frequent itemsets [21], which are the foundation of
(c) association rules [1];
(d) correlated heavy hitters [22];
(e) elements with the highest pointwise mutual information (PMI) [18], and
(f) conditional heavy hitters [28].

In this section, we formally consider each notion or problem definition and study the relationships among them. For consistency with the rest of the paper, we consider streams of pairs $\mathbf{p}$ and c.

Notion 1. The *Frequent Itemsets* (of size two) are the $(\mathbf{p}, c)$-pairs that occur in the data stream more often than a given support threshold: $f_{\mathbf{p},c} > \phi_0, \phi_0 > 0$.

Note that *Heavy Hitters* (HHs) of the stream can be considered as frequent itemsets of size one. Conceptually, the frequent itemsets are equivalent to the heavy hitters if every $(\mathbf{p}, c)$-pair is modeled as a single element.

Notion 2. A $(\mathbf{p}, c)$-pair, or implication $\mathbf{p} \Rightarrow c$, is an *Association Rule* [1] if $f_{\mathbf{p},c} > \phi_1$ (support), $\phi_1 > 0$, and $\Pr[c|\mathbf{p}] > \phi_2$ (confidence), $0 < \phi_2 < 1$.

Notion 3. *Correlated Heavy Hitters* [22] are the pairs $(\mathbf{p}, c)$, where $f_{\mathbf{p}} > \phi_3$, $\phi_3 > 0$ and $\Pr[c|\mathbf{p}] > \phi_2$, $0 < \phi_2 < 1$.

Notion 4. A pair $(\mathbf{p}, c)$ is considered to have the highest *Pointwise Mutual Information* [18] if it is among the top-σ ($\sigma \in \mathbb{Z}_{>0}$) pairs ranked using $PMI(\mathbf{p}, c) = \log \frac{\Pr[\mathbf{p},c]}{\Pr[\mathbf{p}]\Pr[c]}$.

Notion 5.1 A pair $(\mathbf{p}, c)$ is a *Conditional Heavy Hitter* [28] if $\Pr[c|\mathbf{p}] > \phi_2$, $0 < \phi_2 < 1$.

Notion 5.2 The pair $(\mathbf{p}, c)$ is considered to be a *Popular Conditional Heavy Hitter* [28] if it is a conditional heavy hitter and it ranks among the top-τ ($\tau \in \mathbb{Z}_{>0}$) of the highest conditional heavy hitters sorted by $f_{\mathbf{p},c}$.

All the thresholds, ϕ_i, $i = 0, 1, 2, 3$, σ, and τ, are user defined.

We note that we consider frequent itemsets and association rules only for the case of pairs of items. The general problem of frequent itemsets is not considered here as it involves additional challenges, mainly that of pruning non-promising itemsets of varying length, creating an exponentially large search space. Moreover, we point out that Notion 1 (frequent itemsets or heavy hitters) and Notion 4 (pointwise mutual information) are very different from the others, because both these notions treat the two elements of the pair equally, thus, not taking into account the sequential nature of their relationship within the data stream. Notion 1 suffers this limitation due to its simplicity, while Notion 4 treats $\mathbf{p}$ and c symmetrically and finds those pairs where both elements occur together more often than if their occurrences were independent. Therefore, in the following we elaborate on the other three notions, which consider the sequential nature of the pair. Fig. 1 sketches these notions and the relations between them.

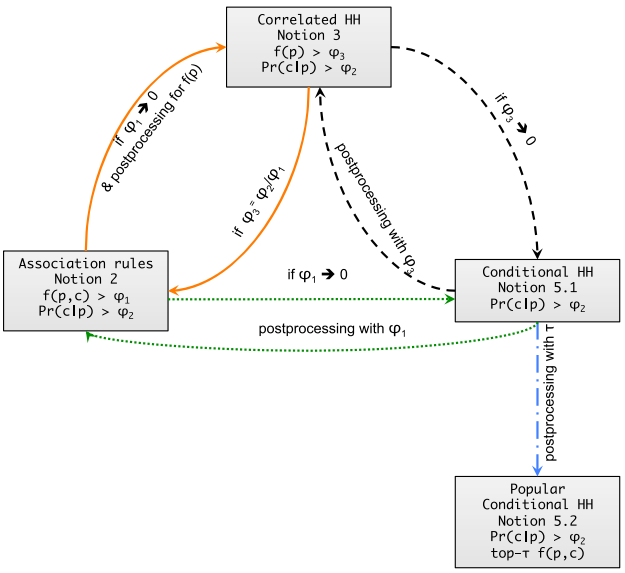

Fig. 1 Comparison of various notions of interesting data stream elements

Association rules (Notion 2) and correlated heavy hitters (Notion 3) are very similar as they both consider the conditional probability of the child given the parent. Notion 2 has an additional constraint on the frequency of the parent–child pair, while Notion 3 has a constraint on the frequency of the parent. Note that given the output of any algorithm for association rules, where $\phi_1 > \varepsilon$ with ε sufficiently close to zero, we can filter out those items where $f_{\mathbf{p}} > \phi_3$ and obtain the same results as correlated heavy hitters.

Lemma 2 *The output of the correlated heavy hitters algorithm produces the output of the association rule algorithm if* $\phi_3 = \phi_2/\phi_1$.

Proof If the pair $(\mathbf{p}, c)$ is a correlated heavy hitter, it satisfies the condition $\Pr[c|\mathbf{p}] > \phi_2$. To qualify as an association rule, the pair $(\mathbf{p}, c)$ should also satisfy the condition:

$$f_{\mathbf{p},c} > \phi_1 \tag{1}$$

Consider the definition of the conditional probability $\Pr[c|\mathbf{p}] = \Pr[\mathbf{p}, c]/\Pr[\mathbf{p}] = f_{\mathbf{p},c}/f_{\mathbf{p}}$. We then have $f_{\mathbf{p},c} = \Pr[c|\mathbf{p}] \cdot f_{\mathbf{p}}$. Since $(\mathbf{p}, c)$ is a correlated heavy hitter, $f_{\mathbf{p}} > \phi_3$ and $f_{\mathbf{p},c} = \Pr[c|\mathbf{p}] \cdot f_{\mathbf{p}} > \phi_3 \cdot \phi_2$. If $\phi_3 = \phi_2/\phi_1$, then condition (1) is satisfied.

The transformations needed to derive association rules from correlated heavy hitters and vice versa are depicted by the solid (orange) arrows in Fig. 1.

Association rules (Notion 2), correlated heavy hitters (Notion 3), and conditional heavy hitters (Notions 5.1 and 5.2) all use a threshold for the conditional probability (note that it is possible to use the same algorithms for conditional heavy hitters and popular conditional heavy hitters; in the latter case, additional ordering and pruning by the frequencies of parent–child pairs are applied). Conditional heavy hitters do not have any additional conditions on the frequencies of the elements; hence, by filtering the results based on the frequencies of parents or the frequencies of parent–child pairs, the results of conditional heavy hitters may be transformed to correlated heavy hitters or association rules, respectively. Likewise, if the thresholds on frequencies ϕ_1 and ϕ_3 are set to a sufficiently small value $\varepsilon > 0$, the results of association rules and correlated heavy hitters algorithms can be used to find the conditional heavy hitters. These relationships are shown using the dashed (black) and dotted (green) arrows in Fig. 1.

These connections show that a solution set for certain of the notions identified above can be manipulated to provide solutions for others also. However, there are overheads in simply trying to apply an algorithm for one notion to solve another: There are additional time and space costs, and it may be necessary to modify the parameters used substantially to obtain the desired output. Furthermore, the quality guarantees offered by a particular algorithm are specific to the notion and thresholds that it targets. In most cases, these guarantees do not translate into guarantees for different notions. For example, if we use the algorithm of correlated heavy hitters for conditional heavy hitters by setting $\phi_3 = 0$, the space guarantees of the algorithm (which depend on ϕ_3) no longer hold. In Sect. 6.2, we further study the relationships between heavy hitters, conditional heavy hitters, correlated heavy hitters, and association rules by experimentally evaluating their behavior in different settings.

5 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 1. We begin with algorithms for the traditional heavy hitters problem and adapt these to identify those which are conditional heavy hitters.

5.1 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 by Metwally et al. [27] 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 that 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 [10,25]: 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

Table 1 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

Algorithm 1 GlobalHH for Conditional Heavy Hitters

Input: Data stream $D = \{(\mathbf{p}_i, c_i), i = 1, 2, ...\}$.
Output: SS - SpaceSaving structure of length s for parent–child pairs
1: $f_{\mathbf{p}_i} = 0, i = 1, 2, ..., |P|$ - keeps the frequency of each parent
2: $m = 0$ - highest frequency of a pair removed from SS
3: **for** each element $(\mathbf{p}_i, c_i)$ of D **do**
4: $f_{\mathbf{p}_i} = f_{\mathbf{p}_i} + 1$
5: **if** $(\mathbf{p}_i, c_i) \in SS$ **then**
6: $SS[(\mathbf{p}_i, c_i)] = SS[(\mathbf{p}_i, c_i)] + 1$
7: **else**
8: **if** $|SS| \geq s$ **then**
9: $m = SS.deleteMin()$
10: **end if**
11: $SS.insert((\mathbf{p}_i, c_i), m + 1)$
12: **end if**
13: **end for**

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, replace it with the new item, and 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 formalize the GlobalHHalgorithm in pseudocode in Algorithm 1. Given each $(\mathbf{p}, c)$ pair in the stream, we insert it into the SS structure (lines 3–13). We also separately maintain information on the frequency of each parent (lines 1, 4). In this first algorithm, we assume that there is sufficient space to store information on all parents, which means we have the exact $f_{\mathbf{p}}$ values. If the SS structure is full (line 8), we eliminate the element with the lowest count from the structure and store its frequency in variable m (line 9). Then, a new element is inserted in SS with frequency $m + 1$ (line 11).

To identify the conditional heavy hitters and the popular conditional heavy hitters, we iterate over all items in the SS structure in decreasing order of their counts. For each stored $(\mathbf{p}, c)$ pair, we compute its estimated $\hat{f}_{p,c}$ value, which is contained in the SS structure and denoted as $SS[(\mathbf{p}_i, c_i)]$ in Algorithm 1. Then, we calculate the corresponding conditional probability $T[(\mathbf{p}_i, c_i)] = SS[(\mathbf{p}_i, c_i)]/f_{\mathbf{p}_i}$ and test whether $T[(\mathbf{p}_i, c_i)] > \phi$. If this condition is true, we output this pair as a conditional heavy hitter. The top-τ such pairs are the popular conditional heavy hitters. We refer to this algorithm as the *GlobalHH* algorithm, since it is based on finding the parent–child pairs which are global heavy hitters.

The SS structure is implemented as a min heap. Since its operations, namely Insert and Delete Minimal element, can be implemented with $O(\log s)$ time complexity, the running time of the GlobalHH algorithm for processing each new element of the stream is $O(\log s)$.

Lemma 3 *Given space $O(s+|P|)$, the GlobalHH algorithm guarantees that each candidate $(\mathbf{p}, c)$ pair output will have $\Pr[c|\mathbf{p}] \leq \widehat{\Pr}[c|\mathbf{p}] \leq \Pr[c|\mathbf{p}] + \frac{1}{s \Pr[\mathbf{p}]}$. When the distribution of $(\mathbf{p}, c)$ pairs follows a Zipfian distribution with parameter $z > 1$, the error bound is improved to $\frac{1}{s^z \Pr[\mathbf{p}]}$.*

Proof Since the $f_{\mathbf{p}}$ values are found exactly, the uncertainty in the estimated conditional probability, $\widehat{\Pr}[c|\mathbf{p}]$, is due to the error from the SS algorithm. This guarantees that our estimate of $f_{\mathbf{p},c}$ is an overestimate by at most n/s for arbitrary streams. We output $\hat{f}_{p,c}/f_{\mathbf{p}}$, which overestimates by at most $\frac{n}{sf_{\mathbf{p}}} = \frac{1}{s \Pr[\mathbf{p}]}$. 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 $f_{\mathbf{p},c}$. 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} [27], improving the error bound to $\frac{1}{s^z \Pr[\mathbf{p}]}$ as claimed.

5.2 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.

Algorithm 2 describes ParentHH. For each parent $\mathbf{p}$ observed in the stream, we maintain an instance of the $SS_{\mathbf{p}}$ structure of size $s/|P|$, dedicated to the children c that arrive as part of a pair for this $\mathbf{p}$ (line 2). For each pair $(\mathbf{p}, c)$ that arrives, we insert c in the corresponding $SS_{\mathbf{p}}$ (line 7).

The output of the ParentHH algorithm, T, which is the set of the s highest conditional heavy hitters, is calculated using the SS_P structure in the following way: For each parent $\mathbf{p}_i$, the corresponding $SS_{\mathbf{p}_i}$ is retrieved. All the pairs $(\mathbf{p}_i, c_i)$, where $c_i \in SS_{\mathbf{p}_i}$ are placed in the answer set T with the corresponding conditional probabilities $T[(\mathbf{p}_i, c_i)]$, are set equal to $SS_{\mathbf{p}_i}(c_i)/f_{\mathbf{p}_i}$. In order to recover the conditional heavy hitters given a threshold ϕ, we consider the set T in decreasing order of the conditional probabilities, and output $(\mathbf{p}, c)$ as an estimated conditional heavy hitter if $\hat{f}_{p,c}/f_{\mathbf{p}} \geq \phi$. The first τ such pairs are popular conditional heavy hitters.

The reintroduction strategy in this algorithm is similar to the GlobalHH algorithm considered above, but in this case, the $SS_{\mathbf{p}}$ structure and reintroduction frequencies $m_{\mathbf{p}}$ are specific for each distinct parent. In our implementation, $SS_{\mathbf{p}}$ is realized using a heap, and all heaps are kept in a hash table with a parent ID as a key. Since in this case we consider a fixed amount of parents, the operations on the hash table take constant time and the time required by the ParentHH

Algorithm 2 ParentHH for Conditional Heavy Hitters

Input: Data stream $D = \{(\mathbf{p}_i, c_i), i = 1, 2, ...\}$.
Output: $SS_{\mathbf{p}_i}$ - SpaceSaving structure of length $s/|P|$ for parent–child pairs assigned to each parent $\mathbf{p}_i, i = 1, 2, ..., |P|$.
1: $f_{\mathbf{p}_i} = 0, i = 1, 2, ..., |P|$ - keeps the frequency of each parent
2: $m_{\mathbf{p}}$ - highest frequency of a pair removed from $SS_{\mathbf{p}}$
3: $m_{\mathbf{p}_i} = 0, i = 1, 2, ..., |P|$
4: **for** each element $(\mathbf{p}_i, c_i)$ of D **do**
5: $f_{\mathbf{p}_i} = f_{\mathbf{p}_i} + 1$
6: **if** $c_i \in SS_{\mathbf{p}_i}$ **then**
7: $SS_{\mathbf{p}_i}(c_i) = SS_{\mathbf{p}_i}(c_i) + 1$
8: **else**
9: **if** $|SS_{\mathbf{p}_i}| \geq s/|P|$ **then**
10: $m_{\mathbf{p}_i} = SS_{\mathbf{p}_i}.deleteMin()$
11: **end if**
12: $SS_{\mathbf{p}_i}.insert(c_i, m_{\mathbf{p}_i} + 1)$
13: **end if**
14: **end for**

algorithm for processing a new element from the stream is $O(\log s)$.

Lemma 4 *Given space $O(\min(s, n))$ (for $s > |P|$), the ParentHH algorithm guarantees that each candidate $(\mathbf{p}, c)$ pair output will have*

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

Proof From the $SS_{\mathbf{p}}$ structure, we obtain an estimate of $f_{\mathbf{p},c}$ which has error proportional to the number of items passed to the structure, which is $f_{\mathbf{p}}$, the number of occurrences of $\mathbf{p}$. So the amount by which $\hat{f}_{p,c}$ overestimates is at most $f_{\mathbf{p}}|P|/s$. When we estimate $\widehat{\Pr}[c|\mathbf{p}]$, the error is $(f_{\mathbf{p}}|P|/s)/f_{\mathbf{p}} = |P|/s$. 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 $f_{\mathbf{p},c}$, 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).

5.3 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 use their (estimated) conditional probability. Since this aligns with the overall goal, it may lead to more accurate behavior. We call this algorithm *CondHH* since it treats the conditional probability as a first class citizen.

In the CondHH algorithm (which is written out in pseudocode in Algorithm 3), we keep a collection of $(\mathbf{p}, c)$ pairs in the CSS (line 3) structure, along with a count for each pair. The algorithm proceeds similarly to GlobalHH: for each $(\mathbf{p}, c)$ pair that arrives, it checks whether it is stored in CSS, and if so, it increases its associated count. If it is not stored, then it evicts some $(\mathbf{p}, c)$ pair from CSS and replaces it with the new pair. Under the GlobalHH semantics, we would apply the SS algorithm and evict the pair with the least frequency. But in the CondHH algorithm, we find the pair with the lowest conditional probability, i.e., with the smallest value of $\widehat{\Pr}[c|\mathbf{p}] = \hat{f}_{p,c}/f_{\mathbf{p}}$, and evict it (line 9). The algorithm also keeps track of the maximum value of $\hat{f}_{p,c}$ for all children of parent $\mathbf{p}$ that have been deleted so far; this value is stored in $m_{\mathbf{p}}$ (line 2). When we need to remove an old pair $(\mathbf{p}', c')$ from the data structure in order to insert a new pair, we update $m_{\mathbf{p}'}$ if needed and insert the new pair $(\mathbf{p}, c)$ with an estimated count $\hat{f}_{p,c} = m_{\mathbf{p}} + 1$ (line 12).

The set of conditional heavy hitters and their probabilities can then be calculated on demand as follows: For each $(\mathbf{p}_i, c_i) \in CSS$, its conditional probability is $T[(\mathbf{p}_i, c_i)] = CSS[(\mathbf{p}_i, c_i)]/f_{\mathbf{p}_i}$. The conditional heavy hitters and the popular conditional heavy hitters are then computed in the same way as in the algorithms described earlier.

Directly implementing this algorithm could be slow, due to the need to find the item with the lowest $\widehat{\Pr}[c|\mathbf{p}]$ on each eviction. 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 [27]. Then the parents are stored in sorted order of $\min_c(\hat{f}_{p,c})/f_{\mathbf{p}}$,

Algorithm 3 CondHH for Conditional Heavy Hitters

Input: Data stream $D = \{(\mathbf{p}_i, c_i), i = 1, 2, ...\}$.
Output: CSS - Conditional SpaceSaving structure of length s for parent–child pairs
1: $f_{\mathbf{p}_i} = 0, i = 1, 2, ..., |P|$ - keeps the frequency of each parent
2: $m_{\mathbf{p}_i} = 0$ - highest frequency of a pair removed from CSS per parent $\mathbf{p}_i, i = 1, 2, ..., |P|$
3: **for** each element $(\mathbf{p}_i, c_i)$ of D **do**
4: $f_{\mathbf{p}_i} = f_{\mathbf{p}_i} + 1$
5: **if** $(\mathbf{p}_i, c_i) \in CSS$ **then**
6: $CSS[(\mathbf{p}_i, c_i)] = CSS[(\mathbf{p}_i, c_i)] + 1$
7: **else**
8: **if** $|CSS| \geq s$ **then**
9: $m_{p_j}^{candidate} = CSS.deleteMinCondProb()$
10: $m_{p_j} = \max(m_{p_j}, m_{p_j}^{candidate})$
11: **end if**
12: $CSS.insert((\mathbf{p}_i, c_i), m_{\mathbf{p}_i} + 1)$
13: **end if**
14: **end for**

i.e., the estimated conditional probability of their least frequent child, via a standard data structure such as a heap or search tree if we need also fast search. 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 $f_{\mathbf{p}}$ and so may also require us to move the parent around to restore the heap property on $\min_c(\hat{f}_{p,c})/f_{\mathbf{p}}$. Likewise, whenever a child of $\mathbf{p}$ is removed from this structure (because it is chosen for eviction), its successor in the sorted order of $\hat{f}_{p,c}$ becomes the new least frequent child of $\mathbf{p}$, which may also require restoring the heap property. At the same time, we can update $m_{\mathbf{p}}$. Thus, implementing this algorithm requires a constant number of pointer operations ($O(1)$) and a constant number of heap or tree operations per update (each taking time $O(\log s)$), leading to an overall time complexity of $O(\log s)$ per element.

Lemma 5 *The CondHH algorithm guarantees that each candidate* $(\mathbf{p}, c)$ *pair output will have*

$$\Pr[c|\mathbf{p}] \leq \widehat{\Pr}[c|\mathbf{p}] \leq \Pr[c|\mathbf{p}] + \frac{m_{\mathbf{p}}+1}{f_{\mathbf{p}}}.$$

Proof For each parent $\mathbf{p}$, $m_{\mathbf{p}}$ is an upper bound on the maximum value of $\hat{f}_{p,c}$ of a $(\mathbf{p}, c)$ pair that has been deleted. Inductively, this is also an upper bound on any $f_{\mathbf{p},c}$ deleted $(\mathbf{p}, c)$ pair: This is true initially when $m_{\mathbf{p}} = f_{\mathbf{p},c} = 0$ for all $(\mathbf{p}, c)$ pairs and is maintained by every operation. Therefore, we have that whenever any new pair is inserted with $\hat{f}_{p,c} = m_{\mathbf{p}} + 1$, we have that $0 \leq f_{\mathbf{p},c} \leq \hat{f}_{p,c} = m_{\mathbf{p}} + 1$, and hence (while it remains in the data structure) $0 \leq \hat{f}_{p,c} - f_{\mathbf{p},c} \leq m_{\mathbf{p}} + 1$. Consequently, we have $0 \leq \widehat{\Pr}[c|\mathbf{p}] - \Pr[c|\mathbf{p}] \leq (m_{\mathbf{p}} + 1)/f_{\mathbf{p}}$. 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 $m_{\mathbf{p}} + 1$, to obtain a lower bound on $f_{\mathbf{p},c}$ and hence $\Pr[c|\mathbf{p}]$.

Note that in general this bound might not be very strong: We may see cases where $m_{\mathbf{p}} + 1 = f_{\mathbf{p}}$, and so we do not obtain a useful guarantee. However, in practice we expect to obtain useful guarantees for the popular conditional heavy hitters.

5.4 FamilyHH algorithm

All the algorithms proposed above make the assumption that we can track detailed information for each parent (such as $f_{\mathbf{p}}$, heavy hitter children for each $\mathbf{p}$, etc.). However, this assumption is not always reasonable. For example, in the network traffic example in Sect. 1, 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 GlobalHHand 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 $f_{\mathbf{p}}$, we will instead use $\hat{f}_{\mathbf{p}}$, an approximate version of the frequency.

The resulting algorithm is called FamilyHH as it keeps track of reintroduction frequencies, one for the parent elements, $m_{\mathbf{p}}$, and one for the parent–child pairs, m_c. FamilyHH, shown in Algorithm 4, uses two separate instances of the SS data structure, namely $SS_{\mathbf{p}}$ for the parents with space t and SS_c for the parent–child pairs with space s (lines 4–1). The insertion and eviction mechanisms are similar to the ones presented in the previous algorithms. When a parent or a child is evicted from the corresponding structure, reintroduction frequencies are updated by its current frequency plus one (lines 9, 11 and 17, 19 correspondingly for a parent and a child).

In order to identify the candidate conditional heavy hitters on demand, we iterate over the heavy hitter parent–child pairs in decreasing order of frequency, and from each find $\widehat{\Pr}[c|\mathbf{p}] = \hat{f}_{p,c}/\hat{f}_{\mathbf{p}}$, or fill in the set T with the corresponding probabilities $T[(\mathbf{p}_i, c_i)]$ set equal to $SS_c[(\mathbf{p}_i, c_i)]/SS_{\mathbf{p}}[\mathbf{p}_i]$.

Algorithm 4 FamilyHH for Conditional Heavy Hitters

Input: Data stream $D = \{(\mathbf{p}_i, c_i), i = 1, 2, ...\}$.
Output: SS_c - SpaceSaving structure of length s for parent–child pairs

1: $SS_{\mathbf{p}}$ - SpaceSaving structure of length t for parents
2: $m_c = 0$ - highest frequency of a pair removed from SS_c
3: $m_{\mathbf{p}} = 0$ - highest frequency of a parent removed from $SS_{\mathbf{p}}$
4: **for** each element $(\mathbf{p}_i, c_i)$ of D **do**
5: **if** $\mathbf{p}_i \in SS_{\mathbf{p}}$ **then**
6: $SS_{\mathbf{p}}[\mathbf{p}_i] = SS_{\mathbf{p}}[\mathbf{p}_i] + 1$
7: **else**
8: **if** $|SS_{\mathbf{p}}| \geq t$ **then**
9: $m_{\mathbf{p}} = SS_{\mathbf{p}}.deleteMin()$
10: **end if**
11: $SS_{\mathbf{p}}.insert(\mathbf{p}_i, ++m_{\mathbf{p}})$
12: **end if**
13: **if** $(\mathbf{p}_i, c_i) \in SS_c$ **then**
14: $SS_c[(\mathbf{p}_i, c_i)] = SS_c[(\mathbf{p}_i, c_i)] + 1$
15: **else**
16: **if** $|SS_c| \geq s$ **then**
17: $m_c = SS_c.deleteMin()$
18: **end if**
19: $SS_c.insert((\mathbf{p}_i, c_i), ++m_c)$
20: **end if**
21: **end for**

Both $SS_{\mathbf{p}}$ and SS_c are implemented as heaps, which leads to $O(\log(t+s))$ running time complexity for processing every element in the data stream.

Lemma 6 *The FamilyHH algorithm guarantees that each candidate* $(\mathbf{p}, c)$ *pair output will have*

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

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

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

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

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.

5.5 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 $(\mathbf{p}, c)$ deleted that came from that group. For example, if the $\mathbf{p}$ 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 $(\mathbf{p}, c)$ 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 [6]. 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 $(\mathbf{p}, c)$ into a Bloom Filter indexed by i. When a new pair $(\mathbf{p}, c)$ is inserted into the data structure, we scan through the Bloom Filters, testing whether $(\mathbf{p}, c)$ 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. SparseHH is described in Algorithm 5. The reintroduction data structures for parents (line 2) and parent–child pairs (line 3) follow one of the strategies listed above and get updated according to this strategy every time there is an eviction (lines 10–11) or an insertion (lines 16–17). As in the previous algorithms, the set of potential heavy hitters is computed on demand and consists of all the pairs $(\mathbf{p}_i, c_i) \in CSS$ with their conditional probabilities $T[(\mathbf{p}_i, c_i)] = CSS(\mathbf{p}_i, c_i)]/f_{ap}(\mathbf{p}_i)$.

The R_c and $R_{\mathbf{p}}$ structures can be implemented with data structures that support the search, insert, and delete operations have (average) complexity of $O(1)$, such as those based on hash tables. The CSS structure, similar to the CondHH algorithm, can be implemented using a double linked list and a heap or a balanced search tree structure. Therefore, the overall running time of SparseHH is $O(\log s)$ per element.

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

Algorithm 5 SparseHH for Conditional Heavy Hitters

Input: Data stream $D = \{(\mathbf{p}_i, c_i), i = 1, 2, ...\}$.
Output: CSS - SpaceSaving structure of length s for parent–child pairs
f_{ap} - frequencies of parents which are active in CSS structure
$R_{\mathbf{p}}$ - reintroduction frequency for a parent
R_c - reintroduction frequency for a pair CSS
for each element $(\mathbf{p}_i, c_i)$ of D **do**
  if $(\mathbf{p}_i, c_i) \in CSS$ **then**
    $f_{ap}(\mathbf{p}_i) = f_{ap}(\mathbf{p}_i) + 1$
    $CSS[(\mathbf{p}_i, c_i)] = CSS[(\mathbf{p}_i, c_i)] + 1$
  else
    if $|CSS| \geq s$ **then**
      $CSS.deleteMinCondProb()$
      Update f_{ap}, R_c and $R_{\mathbf{p}}$
    end if
    if $\mathbf{p}_i \notin f_{ap}$ **then**
      $f_{ap}(\mathbf{p}_i) = R_{\mathbf{p}}(\mathbf{p}_i)$
    end if
    $CSS.insert((\mathbf{p}_i, c_i), R_c[(\mathbf{p}_i, c_i)] + 1)$
    $f_{ap}(\mathbf{p}_i) = f_{ap}(\mathbf{p}_i) + 1$
  end if
end for

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 Sect. 6.

5.6 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.

6 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 $\Pr[c|\mathbf{p}] \geq \phi$ (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 $r = 1, 2, ..., \tau$.

6.1 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 $\Pr[c_h|\mathbf{p}] \geq 0.6$ while $\Pr[c_i|\mathbf{p}] = \frac{1-\Pr[c_h|\mathbf{p}]}{|A|-1}$, where

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

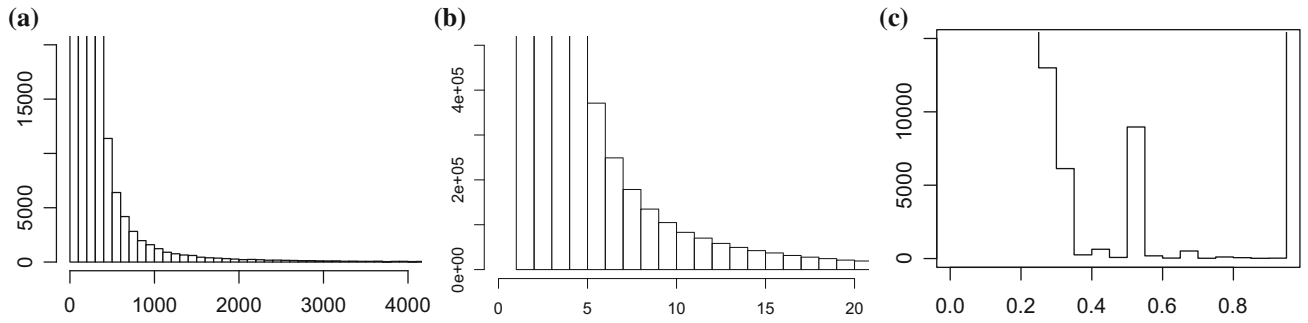

Fig. 2 Descriptive statistics for World Cup 1998 data. **a** Parent frequencies, **b** parent–child pair frequencies, **c** conditional heavy hitter probabilities

$c_i \in A, c_i \neq c_h$. 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 $\mathbf{p}$ is chosen to be heavy, then we pick n_c children C at random and set their transition probabilities $\Pr[c \in C|\mathbf{p}] = 0.8/n_c$, while for the others $\Pr[c \notin C|\mathbf{p}] = 0.2/(|A| - n_c)$. 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 consist 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 $[37.6 \ldots 37.835]$, which covers 26 km and longitude in the range $[-122.52 \ldots -122.35]$, which covers 15 km. 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 are 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 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.

World Cup 1998 data The World Cup data[2] contain 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 these 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 these 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 (Fig. 2a, 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

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

probabilities of conditional heavy hitters is shown in Fig. 2c and shows that there is sufficient probability mass associated with higher conditional probabilities.

6.2 Comparison with association rules, simple, and correlated heavy hitters

In the first set of experiments, we compare conditional heavy hitters to simple heavy hitters, correlated heavy hitters, and association rules, and demonstrate that conditional heavy hitters constitute a distinct set of elements that cannot be identified by existing methods. In these experiments, we computed the top-τ heavy hitters, correlated heavy hitters, and conditional heavy hitters, and then computed the Jaccard distances between the conditional heavy hitters and the other three approaches. We recall that the Jaccard distance between two sets X and Y is defined as $1 - \frac{|X \cap Y|}{|X \cup Y|}$. The distance is 0 if $X = Y$, and the closer the distance gets to 1, the more the two sets are different (distance 1 means that the sets are disjoint).

Since the set of conditional heavy hitters and the set of association rules cannot be directly compared, we performed the experiment as follows. We first identified the top-τ frequent itemsets of sizes two and three combined, as both are needed in order to compute conditional heavy hitters that model a Markov chain of order 2 (i.e., the parent consists of two elements of the data stream). From these τ itemsets, we were able to compute r conditional heavy hitters, and we compared those to the set of top-r conditional heavy hitters, which were extracted from the τ results computed by our approach. The results of this comparison on the World Cup 1998 dataset are shown in Fig. 3 (the plot is broken in two pieces to aid readability: Fig. 3a shows the results for small values of τ, while Fig. 3b plots the trends as τ increases to large values). The numbers (in red) marked on the "CondHH vs. Association Rules" curve denote the value of r for the different experiments.

These graphs show that the set of conditional heavy hitters is very different from the sets produced by the other approaches. This is especially true when we compare traditional heavy hitters to conditional heavy hitters, indicating that the two definitions are truly describing distinct phenomena. Likewise, there is little similarity between the results found for correlated heavy hitters and conditional heavy hitters. We also observe that the number r of parent–child pairs discovered by association rules is very low compared with τ in all cases. Although the most frequent parent–child pairs are similar to the most popular conditional heavy hitters for some of the small values of r, in general, the set of association rules is very different from that of conditional heavy hitters. Thus, current approaches for finding association rules cannot help in retrieving conditional heavy hitters, for which we need new algorithms in order to efficiently identify.

Utility of Conditional HH We now study the elements that are found as conditional heavy hitters and interpret them in a domain where the semantics are known. We compare to the elements found as correlated heavy hitters and show that there is value in both sets discovered, but conditional heavy hitters can provide more useful insights than correlated heavy hitters. For this experiment, we use the Taxicab GPS data and compare the top-25 popular conditional heavy hitters with the top-25 correlated heavy hitters. Following Notion 3, correlated heavy hitters are sorted in descending order of the parent frequency. There is some overlap between these two sets, indicating items that are reported as significant under both definitions. However, there are 14 elements found outside the overlap: 7 specific to each definition. We plot each of these sets of 7 elements overlaid on the San Francisco area maps from which they are drawn, in Fig. 4.

The plots represent the San Francisco region, where the taxicab data was collected from, discretized into a 100×100 grid. Here, a parent is defined as two successive positions (this helps to establish direction of travel, for example), and the child is the subsequent location. In some cases, two out of the three cells intersect. We plot first parent cells (the first in a sequence) using a black border, second parent cells (the second in a sequence) using a blue border, and child cells

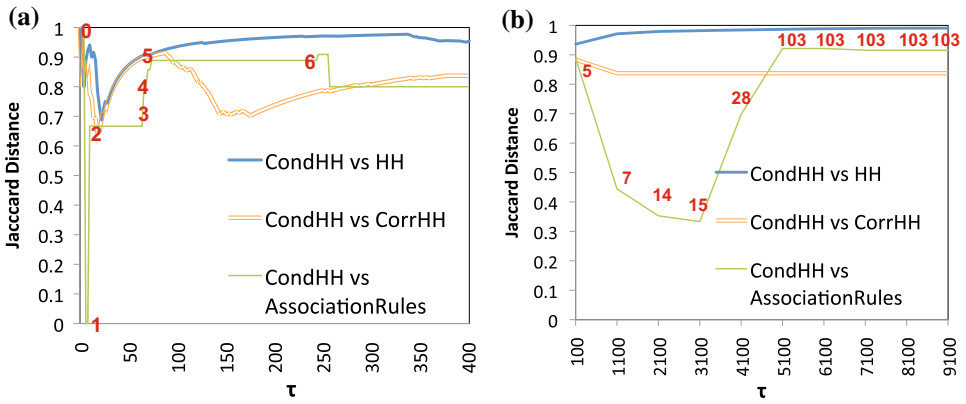

Fig. 3 Jaccard distance between top-τ conditional (CondHH) and simple (HH) heavy hitters, between CondHH and correlated (CorrHH) heavy hitters, and between CondHH and association rules, for which the red numbers marked on the curve correspond to r (the number of retrieved frequent item sets that are triples). **a** τ from 1 to 400, **b** τ from 100 to 9100 (color figure online)

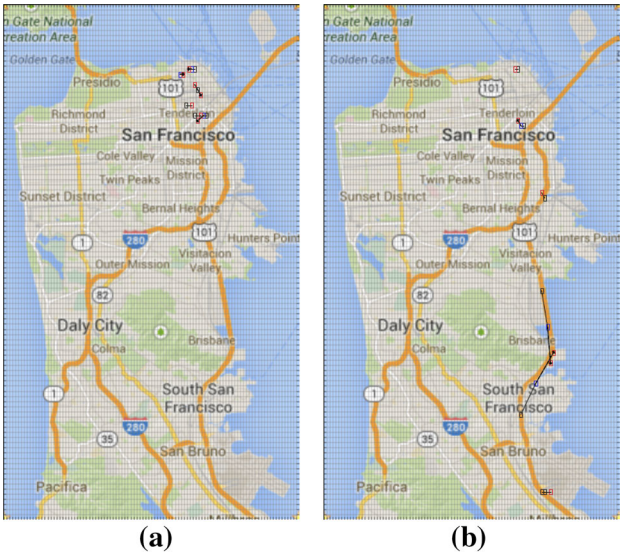

Fig. 4 Unique correlated and conditional heavy hitters not detected by the other technique. **a** Correlated HH, **b** Conditional HH

(the third in a sequence) using a red border. There is a line between the first parent cell and the second parent cell, and an arrow between a second parent cell and a child, if they do not overlap. Figure 4a shows the correlated HH that are not found with the conditional HH definition, while Fig. 4b shows the conditional HH that are not detected as correlated HH.

We interpret these results based on our study of features on the map such as highways and tourist attractions—note that the algorithms do not possess any such knowledge. We observe that the unique correlated HH are primarily short trajectories concentrated around the city center, indicating slow moving traffic around popular points of interest. Meanwhile, the conditional HH are found around the city center but also on the highways further outside the city. Of particular interest are trajectories around the airport (SFO), which show journeys that turn off the highway to go to the airport.

Although similar, the two definitions emphasize different aspects. Correlated heavy hitters are ones which have high support for the parent. In this instance, they have shown that it is common to make slow progress in the heart of the city (parent), in which case it is quite common to continue making slow progress in the same direction (child). While helpful in identifying "pinch points" in the traffic, this does not provide much unexpected information. Conditional heavy hitters place more emphasis on having a high conditional probability of the child, given the parents. In this instance, they highlight that traffic traveling south on highway 101 is likely to stay on (rather than take an exit) and also that traffic on the highway close to the airport is very likely to go to the airport. This highlights the importance of the airport as a destination from the highway. Such insights can be of greater use to traffic planners and city architects in understanding typical journey and behavior around intersections. We obtain similar results when we look at different sized sets, such as the top-20 and top-30 sets. These results demonstrate that the conditional HH concept can reveal interesting and meaningful patterns, distinct from those found by correlated HH.

6.3 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 5 shows the accuracy over the World Cup 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 (Fig. 5a). The ancestor strategy can obtain good results, but only when a larger total amount of memory is made available (Fig. 5b). The Bloom filter strategy, while achieving high recall, always has very poor precision (Fig. 5c). Based on this and other

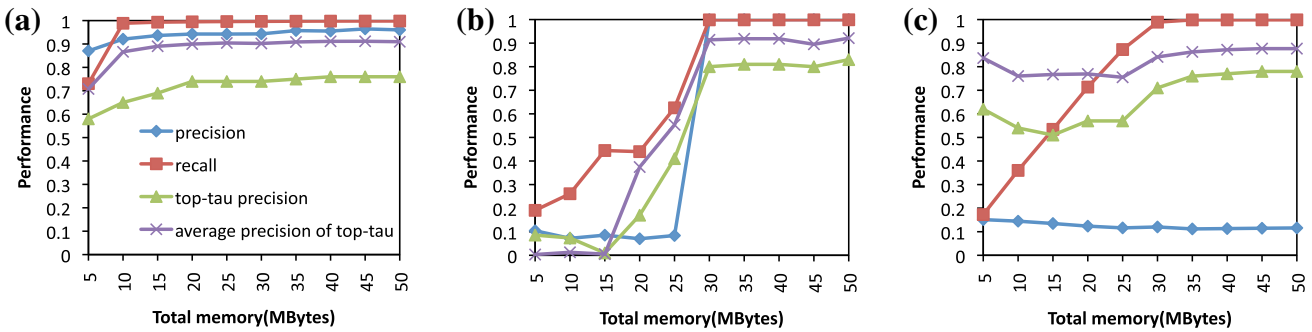

Fig. 5 SparseHH accuracy for conditional heavy hitter recovery on World Cup data under different reintroduction strategies. **a** Hash partition strategy, **b** ancestor strategy, **c** Bloom Filter strategy

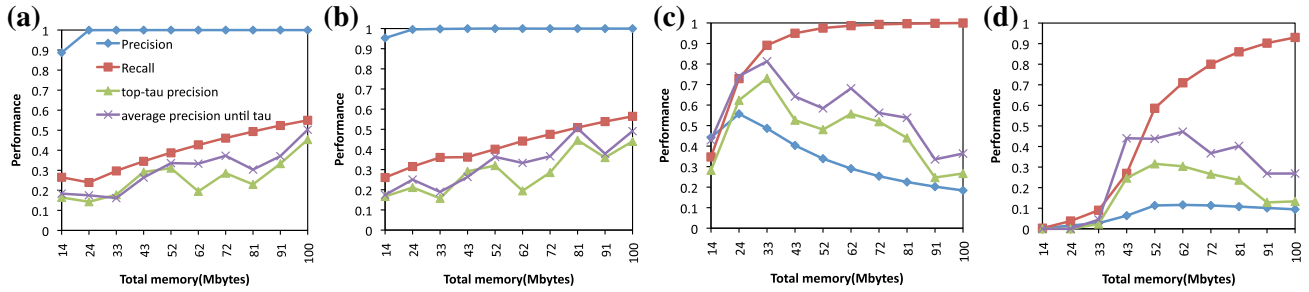

Fig. 6 Accuracy on sparse synthetic data using SparseHH. **a** Favor precision, $\rho = 0.9$, **b** favor precision, $\rho = 0.95$, **c** favor recall, $\rho = 0.5$, **d** favor recall, $\rho = 0.9$

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 Sect. 5.5, 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 Fig. 6 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 we favor precision, recall tends to improve as we allocate more memory (Fig. 6a, b), while favoring recall tends to cause precision to drop off as more memory is used (Fig. 6c, d).

To investigate this further, we fix the available memory and vary the ratio ρ. The results on the same data are shown in Fig. 7. We observe that when we favor precision, the precision is always near perfect (Fig. 7a). 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 (Fig. 7b). Still, it is hard to obtain both good recall and 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.

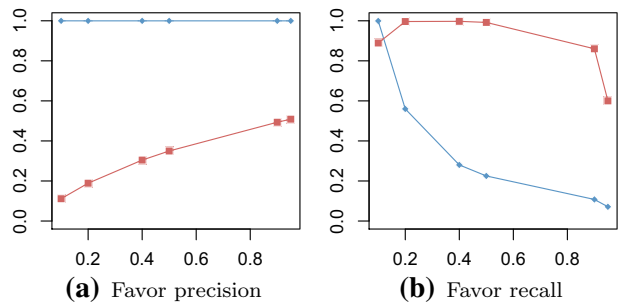

Fig. 7 Precision (*blue diamonds*) and recall (*red squares*) of SparseHH variations on sparse synthetic data as ρ varies. **a** Favor precision, **b** favor recall (color figure online)

6.4 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) World Cup data. In other experiments, we also looked for correlations on other attribute combinations, such as ServerID and ObjectID. The results there were broadly similar and so are omitted for brevity.

Figure 8 shows results on precision and recall for recovering the conditional heavy hitters for these 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.

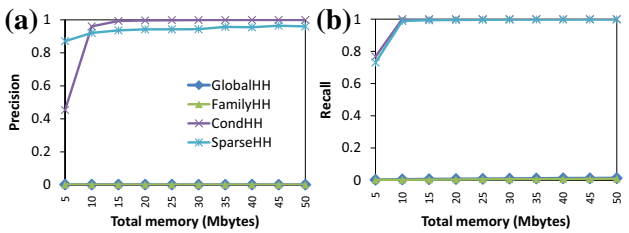

Fig. 8 Precision and recall on the World Cup data. **a** Precision, **b** recall

This observation suggests that such a pruning strategy can be effective at retaining the most promising pairs in memory. For these 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 these 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 9 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.

Pruning Strategy With Combined Eviction Criteria We now evaluate the effect of using a combined eviction criteria, based on both the conditional probability and the frequency of the parent–child pairs, for both of which we fix thresholds. When the memory budget is reached, the eviction strategy works as follows:

1. evict the pair for which both thresholds do not hold;
2. otherwise, evict the pair for which the threshold on the conditional probability does not hold;
3. otherwise, evict the pair for which the threshold on the parent–child frequency does not hold;
4. otherwise, evict the pair with the lowest conditional probability.

This eviction strategy was implemented for both IP (intermediate parent) and Hash reintroduction strategies of SparseHH, and was tested on the World Cup data. The experiments were done using the same settings as for the prior reintroduction strategies and for different frequency thresholds to check their influence on the results. The new strategy does not lead to significant differences: Precision and recall are the same (and up to 10 % smaller for small memory budgets), while top-τ and average τ precision are 10 % higher (and up to 18 % for small memory budgets) for the modified versions. We tried with values 10, 100, and 1000 as the frequency thresholds used for eviction and observed little sensitivity of the results; the reason is that low conditional probability is still an important criterion of eviction. The changes in the Hash version of SparseHH algorithm are even less pronounced, though the general trend is the same: Precision and recall results are a bit lower, while top-τ and average top-τ precisions are higher for the modified algorithm.

Sparse Synthetic Data We now compare all the algorithms on the truly sparse synthetic data, for a stream of length 10^8. These data have a much smaller number of conditional heavy hitters compared with 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 Fig. 10: 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 (Fig. 10b). 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 72 MB. Interestingly, the preci-

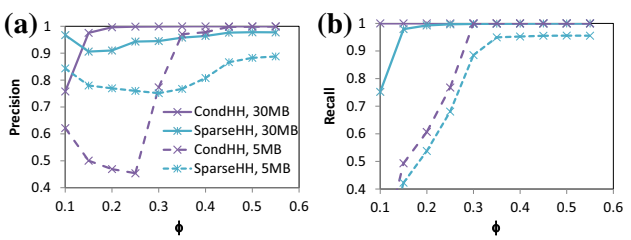

Fig. 9 Accuracy as ϕ varies on World Cup data. **a** Precision, **b** recall

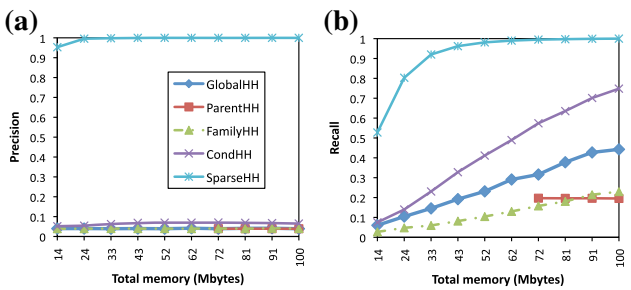

Fig. 10 Accuracy on sparse synthetic data as memory varies. **a** Precision, **b** recall

sion performance of all algorithms apart from SparseHH is very poor: Much more memory is needed before these can achieve good precision (Fig. 10a). This is in part because even the highest amount of memory shown in Fig. 10 represents less than 5 % of the 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, Fig. 11 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.

6.5 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 second-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 $\phi = 0.5$, Fig. 12 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.

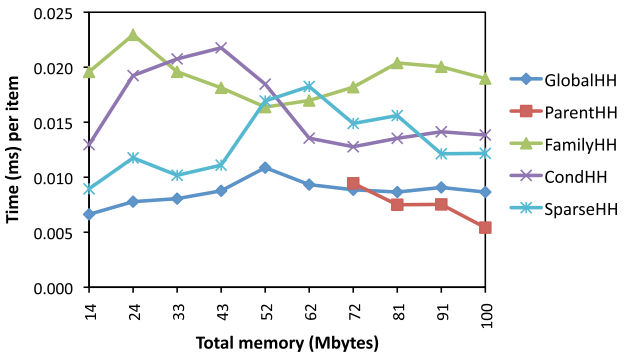

Fig. 11 Time cost of algorithms on sparse synthetic data as memory varies

The results show that the GlobalHH algorithm performs poorly, with only moderate precision and recall on this relatively "easy" dataset (Fig. 12a, 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 these 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. Fig. 12c 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 are quite dense: Many parents have a conditional heavy hitter child. Figure 13 provides precision and recall results on these 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 (Fig. 13a). 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, approaching 1, while SparseHH achieved only 0.25.

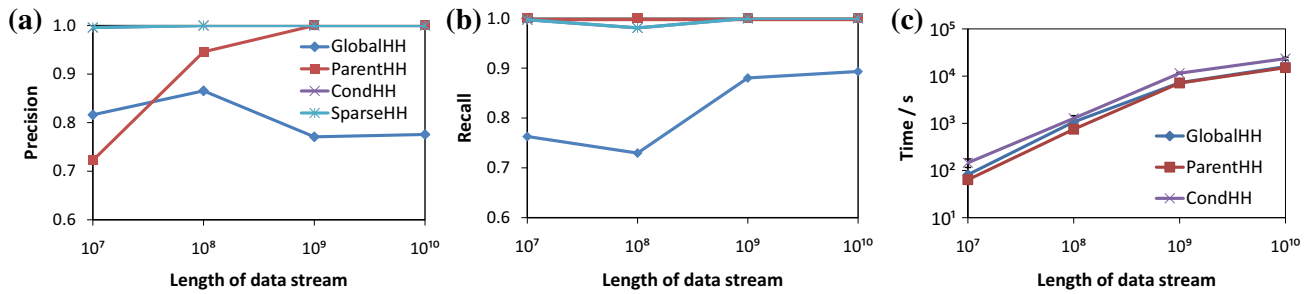

Fig. 12 Accuracy and timing results for algorithms on dense synthetic data. **a** Precision, **b** recall, **c** time

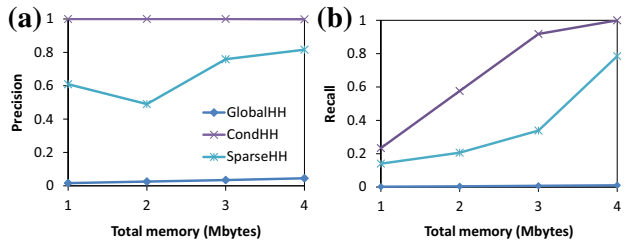

Fig. 13 Accuracy on Taxicab data as memory varies. **a** Precision, **b** recall

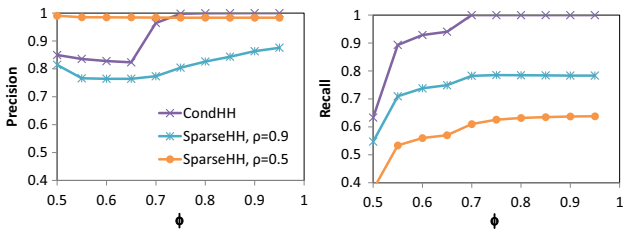

Fig. 14 Accuracy as ϕ varies on Taxicab data

To better understand the relative behavior of these two competitive algorithms, Fig. 14 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.

6.6 Markov model estimation

In this section, we experimentally evaluate how well conditional heavy hitters can approximate a Markov chain model. The premise is that the conditional probabilities we derive from the conditional heavy hitters can be used to estimate the largest elements of the transition probability matrix. We computed the conditional heavy hitters in the Taxicab dataset using the CondHH algorithm, using 3MB of total memory and $\phi = 0.8$ (the corresponding precision and recall values are shown in Fig. 13).

First, we check whether conditional heavy hitters can adequately describe the generation process of the real trajectories in the Taxicab data. To study this, we compare the heatmaps of the trajectories generated by the exact Markov model and by the recovered Markov model, i.e., the model approximated by the conditional heavy hitters results, depicted in Fig. 15. The heatmap indicates the number of times different trajectories have passed through a spatial cell. The "hottest" cells are colored in red, while the "coolest" in blue. The resulting heatmaps show that the trajectories built using conditional heavy hitters accurately reflect the hot regions of the trajectories built using the exact transition probability matrix, which indicates that conditional heavy hitters indeed capture the highest transition probabilities.

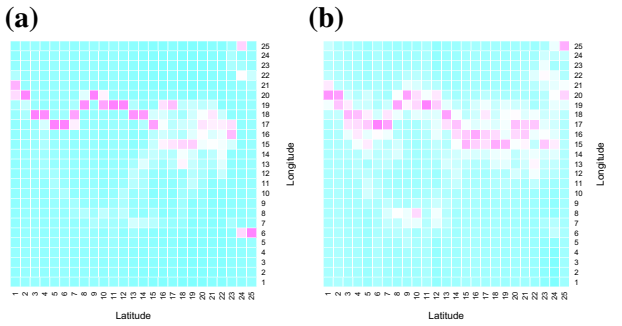

Fig. 15 Heatmaps of trajectories modeled using exact transition probability matrix and recovered matrix based on conditional heavy hitters. **a** Exact Markov model, **b** recovered Markov model based on conditional heavy hitters (color figure online)

Table 2 Earth Mover's Distance between the heatmaps of trajectories

Heatmaps	Exact Markov model
Conditional HH	1.98
Supervised random	4.13
Random	7.40

Earth Mover's Distance comparison In order to quantify the distance between the two heatmaps, we employ the Earth Mover's Distance (EMD)[33]. EMD measures the difference of probability masses, multiplied by the distance that the probability mass has to be moved in order to derive one probability density function from the other. In our setting, we compute EMD over the probability density functions (or the corresponding histograms) that are represented by the heatmaps. We calculated the EMD between the heatmaps of trajectories generated by the exact Markov model and conditional heavy hitters, as well as two baseline models, random and supervised random. The *Random* model corresponds to a random assignment for the next state of a trajectory at each step. The *Supervised Random* assigns the next state in the same way as our approach does when a particular prefix (i.e., parent) is missing. That is, it distributes the probability mass among the neighborhood of this prefix, assigning slightly more mass in the direction of movement (according to the previous state). The results, reported in Table 2, show that conditional heavy hitters are more than two times closer to the exact Markov model than the supervised random assignment, and more than 3.5 times closer than Random.

Kernel density comparison We also conducted a *Kernel density-based two-sample comparison (KDE)* test [17] to compare the position distributions of the trajectories generated by the exact and recovered Markov models. The null hypothesis of this test was that the distributions were equal. We set the significance level $\alpha = 0.05$ and obtained a p-value for the test equal to 0.51, which means that there is not enough evidence to reject the hypothesis that these two distributions are equal.

Prediction accuracy Besides the heatmaps, we also assess the prediction errors of the exact and recovered Markov models. Our goal is to check whether the recovered model can be used to predict the next state of a Taxicab given its two previous states. We also compare the results with the error of random and supervised random models described above. Each model defines a probability distribution for the next state given two previous states. The prediction is performed using this probability distribution.

Two kinds of errors are considered:

1. The mean Euclidean distance (MED) between the estimated cell $\hat{c}_i$ and true cell c_i, $i = 1, 2, ..., T$, where T is the number of predictions over Taxicab trajectories, T is larger than:

$$\begin{aligned} \text{MED} &= \frac{1}{T}\sum_{i=1}^{T} \text{dist}(\hat{c}_i, c_i) \\ &= \frac{1}{T}\sum_{i=1}^{T} \sqrt{(\hat{c}_i^x - c_i^x)^2 + (\hat{c}_i^y - c_i^y)^2}, \end{aligned}$$

   where c^x and c^y are spatial coordinated of the cells in the 100×100 grid that correspond to initial longitude and latitude of trajectory points;
2. The misclassification error (ME) or ratio of cells which were incorrectly estimated:

$$ME = \frac{1}{T}\sum_{i=1}^{T} \mathbb{1}\{\hat{c}_i \neq c_i\}.$$

Table 3 summarizes the results, which are averaged over 10 runs. The differences between all pairs of errors are statistically significant according to Welch two-sample t test [37], for which significance level α was set to 0.01. The results show that the accuracy of the exact model compared to the accuracy of the recovered model is 1 and 5.6 % higher according to MED and ME correspondingly. This was expected as the recovered model is just an approximation of the exact model. At the same time, the performance of the recovered model is 23 and 6.2 % better than the performance of random and supervised random models according to ME (2521 and 19 % better according to MED). The prediction made with the recovered model for the states with known prefix behaves even better.

Table 3 Mean Euclidean Distance (MED) and Misclassification Error (ME) of different prediction models for Taxicab dataset

Prediction Model	MED	ME
Exact model	1.77	0.80
Recovered model	1.87	0.81
Recovered model only with known prefix	1.59	0.74
Random	47.15	1.00
Supervised random	2.23	0.86

We note that both the heatmaps and the prediction errors can be improved if we better model the cases where the prefix is not among the conditional heavy hitters. This can be achieved using domain knowledge, such as the road maps in the region of the Taxicab dataset. Nevertheless, even without domain knowledge, we have shown that the trajectories built using the conditional heavy hitters are a fairly accurate representation of the original data.

7 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 are 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 distrib-

uted 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. Agrawal, R., Imielinski, T., Swami, A.N.: Mining association rules between sets of items in large databases. In: ACM SIGMOD International Conference on Management of Data, pp. 207–216 (1993)
2. Alon, N., Matias, Y., Szegedy, M.: The space complexity of approximating the frequency moments. In: ACM Symposium on Theory of Computing, pp. 20–29 (1996)
3. Arasu, A., Manku, G.S.: Approximate counts and quantiles over sliding windows. In: Proceedings of the Twenty-Third ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems, pp. 286–296. ACM (2004)
4. Baum, L.E., Petrie, T.: Statistical inference for probabilistic functions of finite state Markov chains. Ann. Math. Stat. **37**(6), 1554–1563 (1966)
5. Boyer, B., Moore, J.: A fast majority vote algorithm. Tech. Rep. ICSCA-CMP-32. Institute for Computer Science, University of Texas (1981)
6. Broder, A., Mitzenmacher, M.: Network applications of bloom filters: a survey. Internet Math. **1**(4), 485–509 (2005)
7. Budak, C., Georgiou, T., Agrawal, D., El Abbadi, A.: Geoscope: online detection of geo-correlated information trends in social networks. PVLDB **7**(4), 229–240 (2013)
8. Chang, J.H., Lee, W.S.: Finding recent frequent itemsets adaptively over online data streams. In: KDD, pp. 487–492 (2003)
9. Charikar, M., Chen, K., Farach-Colton, M.: Finding frequent items in data streams. In: Proceedings of the International Colloquium on Automata, Languages and Programming (ICALP) (2002)
10. Cormode, G., Hadjieleftheriou, M.: Finding frequent items in data streams. In: International Conference on Very Large Data Bases (2008)
11. Cormode, G., Korn, F., Muthukrishnan, S., Srivastava, D.: Finding hierarchical heavy hitters in data streams. In: International Conference on Very Large Data Bases, pp. 464–475 (2003)
12. Cormode, G., Korn, F., Tirthapura, S.: Time decaying aggregates in out-of-order streams. In: Proceedings of the Twenty-Seventh ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems, pp. 89–98. ACM (2008)
13. Cormode, G., Muthukrishnan, S.: An improved data stream summary: the count-min sketch and its applications. J. Algorithm. **55**(1), 58–75 (2005)
14. Dallachiesa, M., Nushi, B., Mirylenka, K., Palpanas, T.: Uncertain time-series similarity: return to the basics. PVLDB **5**(11), 1662–1673 (2012)
15. Dallachiesa, M., Palpanas, T.: Identifying streaming frequent items in ad hoc time windows. Data Knowl. Eng. **87**, 66–90 (2013)
16. Demaine, E., López-Ortiz, A., Munro, J.I.: Frequency estimation of internet packet streams with limited space. In: European Symposium on Algorithms (ESA) (2002)
17. Duong, T., Goud, B., Schauer, K.: Closed-form density-based framework for automatic detection of cellular morphology changes. Proc. Natl. Acad. Sci. **109**(22), 8382–8387 (2012)
18. Durme, B.V., Lall, A.: Streaming pointwise mutual information. In: Advances in Neural Information Processing Systems, pp. 1892–1900 (2009)
19. Gehrke, J., Korn, F., Srivastava, D.: On computing correlated aggregates over continual data streams. In: ACM SIGMOD International Conference on Management of Data, pp. 13–24 (2001)
20. Giannella, C., Han, J., Pei, J., Yan, X., Yu, P.S.: Mining frequent patterns in data streams at multiple time granularities. In: Kargupta, H., Joshi, A., Sivakumar, K., Yesha, Y. (eds.) Next Generation Data Mining, pp. 191–212 (2003)
21. Han, J., Pei, J., Yin, Y.: Mining frequent patterns without candidate generation. In: SIGMOD Conference, pp. 1–12 (2000)
22. Lahiri, B., Tirthapura, S.: Finding correlated heavy-hitters over data streams. In: IEEE 28th International Conference on Performance Computing and Communications (IPCCC), pp. 307–314. IEEE (2009)
23. Lee, L-K., Ting, H.F.: A simpler and more efficient deterministic scheme for finding frequent items over sliding windows. In: Proceedings of the Twenty-Fifth ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems, pp. 290–297. ACM (2006)
24. Letchner, J., Ré, C., Balazinska, M., Philipose, M.: Approximation trade-offs in Markovian stream processing: an empirical study. In: IEEE 26th International Conference on Data Engineering (ICDE), pp. 936–939. IEEE (2010)
25. Manerikar, N., Palpanas, T.: Frequent items in streaming data: an experimental evaluation of the state-of-the-art. Data Knowl. Eng. **68**(4), 415–430 (2009)
26. Manku, G., Motwani, R.: Approximate frequency counts over data streams. In: International Conference on Very Large Data Bases, pp. 346–357 (2002)
27. Metwally, A., Agrawal, D., Abbadi, A.E.: Efficient computation of frequent and top-k elements in data streams. In: International Conference on Database Theory (2005)
28. Mirylenka, K., Cormode, G., Palpanas, T., Srivastava, D.: Finding interesting correlations with conditional heavy hitters. In: International Conference on Data Engineering (ICDE) (2013)
29. Misra, J., Gries, D.: Finding repeated elements. Sci. Comput. Program. **2**, 143–152 (1982)
30. Pearl, J.: Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference. Morgan Kaufmann Publishers Inc., Los Altos (1988)
31. Rabinovich, M., Spatschek, O.: Web Caching and Replication. Addison-Wesley Longman Publishing Co., Inc, Boston (2002)
32. Raftery, A.E.: A model of high-order Markov chains. J. R. Stat. Soc. Series B Methodol. **47**(3), 528–539 (1985)
33. Rubner, Y., Tomasi, C., Guibas, L.: The earth mover's distance as a metric for image retrieval. Int. J. Comput. Vision **40**(2), 99–121 (2000)
34. Tantono, F.I., Manerikar, N., Palpanas, T.: Efficiently discovering recent frequent items in data streams. In: Scientific and Statistical Database Management, pp. 222–239. Springer, Berlin, Heidelberg (2008)
35. Venkataraman, S., Song, D.X., Gibbons, P.B., Blum, A.: New streaming algorithms for fast detection of superspreaders. In: Network and Distributed System Security Symposium NDSS (2005)
36. Wang, P., Wang, H., Wang, W.: Finding semantics in time series. In: ACM SIGMOD International Conference on Management of Data, pp. 385–396 (2011)
37. Welch, B.L.: The generalization of 'student's' problem when several different population variances are involved. Biometrika **34**(1/2), 28–35 (1947)
38. Yu, P.S., Chi, Y.: Association rule mining on streams. In: Encyclopedia of Database Systems, pp. 136–139. Springer-Verlag (2009)