Finding Hierarchically Correlated Heavy Hitters Summary from Two Dimensional Data Streams

Zubair Shah Abdun Naser Mahmood Michael Barlow

University of New South Wales, Australia
zubair.shah@student.adfa.edu.au, {a.mahmood,b.barlow}@adfa.edu.au

Technical Report
UNSW-CSE-TR-201605
March 2016

THE UNIVERSITY OF NEW SOUTH WALES

School of Computer Science and Engineering
The University of New South Wales
Sydney 2052, Australia

Abstract

While most applications work on traditional "flat" data, many domains contain hierarchical data, such as time, geographic locations, IP addresses etc. Flat methods are generally not suitable for hierarchical data, and existing hierarchical approaches–such as, hierarchical heavy hitters, multilevel and cross-level association rules–cannot capture the semantics we require when we monitor data in the form of hierarchically correlated pairs. Therefore, in this work, we introduce the concept of Hierarchically Correlated Heavy Hitters (HCHH), which captures the sequential nature between pairs of hierarchical items at multiple concept levels. Specifically, the approach finds the correlation between items corresponding to hierarchically discounted frequency counts. We have provided formal definition for the proposed concept, and developed algorithmic approaches for solving HCHH efficiently in data streams. The proposed HCHH algorithms have deterministic error guarantees, and space bounds. They require $O(\frac{\eta}{\epsilon_p \epsilon_s})$ memory, where η is a small constant, and $\epsilon_p \in [0,1]$, $\epsilon_s \in [0,1]$ are user defined parameters. We have compared the proposed concept of HCHH with other existing similar hierarchical notions; experimental analysis shows that HCHH identifies more interesting patterns that other hierarchical notions cannot capture. Furthermore, experimental results demonstrate that the proposed HCHH algorithm is much more efficient in terms of memory usage and output quality compared to benchmark algorithm.

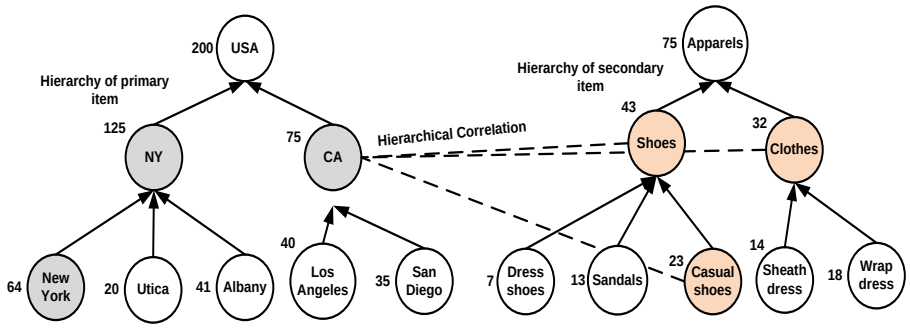


Figure 1.1: An example of hierarchically correlated heavy hitters, where location is primary item and product is secondary item.

1 Introduction

One of the widely used approaches in data stream mining is computing an online summary that offers tremendous potential for extracting useful intelligence and actionable rules. Such summaries are useful in many applications that require fast analysis based on the statistical properties of real-time streaming data [13, 30, 17]. For example a heavy hitters summary is used in database joins, data warehousing (e.g., OLAP), web caching and hits, search engine queries, network usage monitoring, DDoS detection [7, 5] and more. There has been a great deal of work in discovering algorithms to find heavy hitters summary under a variety of scenarios and data arrival models [24, 29, 3, 20].

While most applications work on traditional "flat" data, many domains contain hierarchical data, such as time, geographic locations, and IP addresses. Techniques developed for flat data summarization are not suitable to capture the important characteristics of hierarchical data. For example, besides finding relationship between flat data such as milk and bread, it is often more desirable to also show their hierarchical relationship on a different concept level, such as between wheat bread and 2% milk [15]. The relationship in the latter statement is expressed at a lower concept level but often carries more specific information than the former. Summaries with such hierarchical characteristics deliver substantial benefits for marketing, decision making, and business management [16, 15].

A related more interesting and important stream summarization problem is to discover the semantics of the data when we monitor data in the form of hierarchically correlated pairs. For instance, observing clickstreams containing pairs of items, such as $(location, product)$ from an e-commerce website, it would be interesting to find correlation between location and products at various concept levels. This problem can be best described using Fig. 1.1. The figure shows taxonomies of both locations and products, with the numbers beside the circles indicating the count of visitors from those locations and the number of visits to the products. Note, the number of visits to the products are with respect to certain location (e.g., in this case CA). An interesting query on this data could be to find a summary of highly visited products (or categories such as shoes), for all cities (states or countries) that constitute major proportion of the website visitors. For instance, Fig. 1.1 shows that for a 25% threshold; New

York, NY and CA are the major locations of the website visitors (marked Grey), and for a 30% threshold, highly visited products among CA visitors are casual shoes, shoes and clothes (marked Orange). This summary describes correlation between products and users with different demographic backgrounds. It can be used for dynamic customers segmentation to help marketing efforts serve better, by targeting specific, smaller groups with messages that those customers would find relevant and lead them to buy something. Moreover, it can help with both deeper understanding of customers' preferences with various demographic backgrounds, and discovery of what each segment finds most valuable to more accurately tailor marketing materials toward that segment.

In this work, we introduce the concept of Hierarchically Correlated Heavy Hitters (HCHH) for data streams containing pairs of items. We refer to the first and second items of the pair as *primary* and *secondary* items respectively. Each of these items are attributes that are possibly derived from different concept hierarchies, e.g., both NY, CA belongs to the location hierarchy, and shoes, shirts belong to the product hierarchy. Our aim is to find a summary consisting of a set of correlated pairs, which captures the sequential nature or correlation between pairs of items at multiple concept levels. Below we describe some motivating scenarios to explain the utility of HCHH; formal definition of HCHH is provided in Section 2.4.

Motivating Scenarios: Following are the motivating scenarios of the HCHH concept.

- *Network Monitoring:* We can find HCHH over packets of streams within the network, where in each packet the destination address is the primary item, and the source address is the secondary item. Here HCHH can find those sources (a source can be a single IP address or network address) that co-occur most frequently for their corresponding destinations (again a destination can be a single IP address or network address). For instance, at higher level of IP hierarchy, this can reveal interesting correlations between sources (e.g., universities, research labs, and large organizations) and their corresponding popular destinations (e.g., search engines, publishers, social networks, and video providers). This can be useful in traffic planning, identifying interesting trends, and shifts in popularity of sources or destinations, especially when several IP sources are seen to share the same primary item (i.e., destination IP address) as a HCHH.

- *Clickstream Analysis:* Websites and online advertisers spend a large sum of money for market research on potential online customers; including their demographics, their interest on specific products and their frequency of visits and purchases. This information can be obtained through Clickstream analysis; a process of analysing users clickstream (information about whatever parts of a web page user clicks on)– which typically contains user id (if logged in), timestamp, visitors IP address (raw and geocoded), visited URL, and navigation of visitors within their website.

  Using hierarchically correlated heavy hitters over clickstreams can reveal interesting patterns as explained above with reference to Fig. 1.1. Another use of HCHH over clickstreams is to predict a list of web links that a particular user or a group of users will follow with a high probability. By

applying HCHH on stored navigational patterns (e.g., A→ B→ C, which records the successive links clicked by a user), we can obtain a list of highly correlated links for frequently visited web pages (e.g., in the form of pairs (A,B), (A,C), (B,C) etc.). This can be useful for understanding user behavior, e.g., which link majority of the users come into a site, highly visited pages, where the visitors left, and where people are clicking and where they are not. This would help in placing the marketing material at the right place, and to build far better, more usable, and revenue-generating website.

- *Medical Domain:* In medical domain a huge amount of patient prescription data is collected by government health agencies and research organizations in the form of $(examination, prescription)$. There can be many different examinations suggested to patients. Depending on the results of an examination there can be many different prescriptions for a patient. Health care agencies are interested to find out what are the frequently advised examinations and for any frequent examination, what are the frequently prescribed medications. Generally, this huge data is very sparse, hence meaningful rules cannot be derived from the available data [4]. However, analyzing this data using HCHH at different abstraction levels allows experts to discover interesting and actionable knowledge which may otherwise remain hidden (e.g., in the flat data).

- *Recommender System:* In online shopping websites, personalized product recommendation is very important for a business to increase revenue and for a user to have a rich experience in shopping. The online HCHH algorithm can parse the transaction as it arrives, to identify a list of additional products (from various categories) that were bought together with the most frequently purchased products (at various concept levels). The performance of a recommender system can be optimize by storing an updated list of HCHH in a cache with a small lookup time, and then perform recommendation to buyers using HCHH summary rather than performing recommendation from large disk resident data.

Contributions: The contributions of this paper are as follows:

1. We introduce the concept of HCHH, which can be applied in a variety of scenarios to discover interesting and actionable patterns. We provide formal definition for our proposed concept, and develop algorithmic approaches for solving HCHH problem efficiently in data streams.

2. We show that the proposed HCHH algorithm provides deterministic error guarantees, and space bounds, specifically it requires $O(\frac{\eta}{\epsilon_p \epsilon_s})$ memory, where η is a small constant, and $\epsilon_p \in [0,1]$, $\epsilon_s \in [0,1]$ are user defined parameters.

3. To develop HCHH algorithm, we first show how to find flat correlated heavy hitters optimally. We give a space optimal flat correlated heavy hitter algorithm that has $O(\frac{1}{\epsilon_p \epsilon_s})$ memory bound. Next, we extend the flat correlated heavy hitter algorithm to develop the HCHH algorithm.

4. Evaluation of the proposed HCHH algorithms with existing similar hierarchical notions shows that HCHH identifies interesting patterns that

are missed by the other hierarchical notions. Furthermore, experimental results demonstrate that the proposed HCHH algorithm is much more efficient in terms of memory usage and output quality compared to benchmark algorithm.

The remainder of this paper is organized as follows; In Section 3 we describe our proposed algorithms and provide the pseudo−codes and theoretical proofs on bounds in space and accuracy guarantees. The implementation details and experimental results are shown in Section 4. Section 5 provides a brief insight on existing work in the area of data streams, specifically on summarization, and the paper concludes in Section 6.

2 Problem Definition

2.1 Notation

Let the set of transactions $T = \{(\tau_1, w_1), (\tau_2, w_2), (\tau_3, w_3) \cdots\}$ be a continuous stream, where each transaction τ_i is a tuple $(p_i,\ s_i)$ of pairs such that p is a primary item and s is a secondary item, w_i is the weight associated with τ_i, and N is the sum of the weights of the transactions processed so far.

2.2 Correlated Heavy Hitters

The concept of Correlated Heavy Hitters (CHH) can formally be defined as:

Definition 1 *(Correlated Heavy Hitters) Let the frequency of p, f_p be defined as,*

$$f_p = \sum_{\forall p_i = p} w_i$$

The frequency of a (p, s) pair, $f_{p,s}$, is defined as;

$$f_{p,s} = \sum_{\forall p_i = p \wedge \forall s_i = s} w_i$$

The user define parameters are $\phi_p \in [0, 1]$ and $\phi_s \in [0, 1]$. The CHH problem is to compute a set F_c so that for all pairs,

$$(p, s) \in F_c, (f_p \geq \phi_p N) \wedge (f_{p,s} \geq \phi_s f_p)$$

The exact set of CHH cannot be found with limited memory and using one pass over the data (i.e., stream computational model), hence, the concept of online identification of CHH is introduced next.

Definition 2 *(Online CHH identification problem) Given user defined parameters ϵ_p, ϵ_s, $\phi_p \in [\epsilon_p, 1]$ and $\phi_s \in [\epsilon_s, 1]$, the online CHH identification problem is to compute a set F_c so that the following conditions are met;*

1. Accuracy:

$$\hat{f}_p \leq f_p + \epsilon_p N$$

$$\hat{f}_{p,s} \leq f_{p,s} + \epsilon_s f_p$$

2. Coverage:

$$\forall (p^{'}, s^{'}) \notin F_c, (f_{p'} < \phi_p N) \wedge (f_{p',s'} < \phi_s f_p)$$

The conditions state that the accuracy of the estimated frequency $\hat{f}_p$ is within $\epsilon_p N$ of the true frequency f_p and the estimated frequency $\hat{f}_{p,s}$ is within $\epsilon_s f_p$ of the true frequency $f_{p,s}$.

Lemma 1 *Using formulation of CHH from definition 2, false positives are possible, but false negatives are not.*

In other words, while no correlated heavy hitters will be missed, due to limited memory for storing the unbounded stream, some non-correlated heavy hitters may be reported. This error is controlled using parameters ϵ_p and ϵ_s.

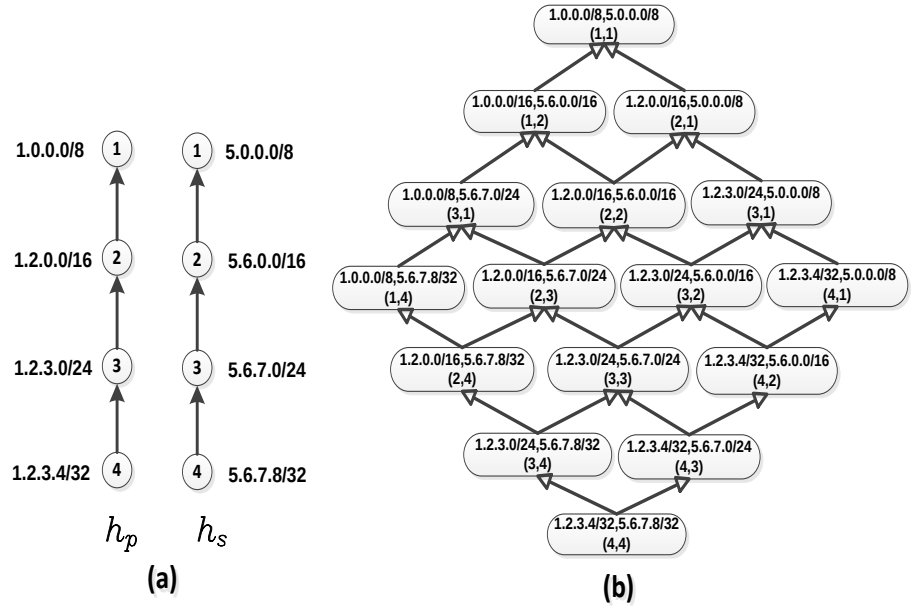


Figure 2.1: A lattice like structure build from two dimensional IP addresses (e.g., source IP address and destination IP address).

2.3 Generalizing item hierarchies

Here we illustrate the concept of generalization using hierarchies h_p, and h_s, as an example, we consider a transaction from network IP data with destination IP address as p and source IP address as s. This is because the company might be interested in finding out which of its internet connected servers are receiving most visitors, and what is the location of these visitors that are most occurring for the heavily visited servers. This information may be used to improve visitor experience by creating, relocating or adding redundant servers at closer geo-location, or customizing the offered services tailored to the frequent visitors.

Consider a example transaction (1.2.3.4/32, 5.6.7.8/32) where 1.2.3.4 and 5.6.7.8 are two IP addresses used for illustration only. The notation x.x.x.x/b is often used to represent IP addresses where b indicates the number of bits important to distinguish a given IP address. For example, if b = 32, then all the 4x8 = 32 bits are significant. If b = 24, then only the first 24 bits (the 24 MSB bits) are significant and the remaining 8-bits are disregarded. The height

of hierarchy of both destination and source IP is the same $h_p = 4$ and $h_s = 4$, when a byte-wise hierarchy is considered, shown in Fig. 2.1.

Each transaction τ can be generalized using item hierarchies. We refer to the generalized transaction τ_a as the ancestor of the transaction τ from which it has been generalized. The generalization can be performed using function *GeneralizeTo*(.). For instance, *GeneralizeTo*(τ , L) is a generalization step of transaction τ to label L. We represent each label of the hierarchy by L, as a vector of two integers, where entries $L_p \in L$ and $L_s \in L$ represent the levels of hierarchies of the items p and s respectively. For example, L =(4, 4) shows the transaction, where both items are not generalized. Similarly, L =(3, 3) represents an ancestor transaction τ_a where both p and s are generalized to level 3. Thus, function *GeneralizeTo*(1.2.3.4/32,5.6.7.8/32, (3, 2)) returns a transaction τ_a = (1.2.3.0/24,5.6.0.0/16), where item p is at level 3 and item s is at level 2 of their hierarchies. It is important to note that the generalization, such as *GeneralizeTo*(1.2.3.0/24,5.6.0.0/16, (4, 2)) can not be computed, because the transaction (1.2.3.0/24,5.6.0.0/16) is currently at label (3, 2), and the function *GeneralizeTo*(1.2.3.0/24,5.6.0.0/16, (4, 2)) requires to *roll down* (a concept used in OLAP [23]) in the hierarchy. However, in this case rolling down the hierarchy is not possible because transactions at higher level of generalization could be rolled down to a large numbers of possible entries (i.e., there is no single descendant).

By enumerating each possible combination of L_p and L_s using *GeneralizeTo*(.) function, we can build a lattice like structure as shown in Fig. 2.1. In general, the number of nodes η in lattice build from h_p and h_s are $\eta = h_p \times h_s$, and the height λ of the lattice built from h_p and h_s is $\lambda = h_p + h_s - 1$. The levels of the lattice (not to confuse with the levels of p and s) are numbered 1 (top most node of the lattice) to λ (bottom most node of the lattice), and represented by an integer l.

2.4 Hierarchically Correlated Heavy Hitters

This section provides the formulations of the HCHH. Let p_a and s_a be the ancestors of p and s at level L respectively, such that (p_a, s_a) =*GeneralizeTo*(p, s, L). For the sake convenience, let the notation $\prec$ represents ancestor–descendant relationship, for instance $p \prec p_a$ represents that p_a is the generalization of p. Also, let $p \preccurlyeq p_a$ represents $(p \prec p_a) \vee (p = p_a)$, then the frequency of p_a, f_{p_a} and frequency of the pair (p_a, s_a), f_{p_a,s_a} can be defined as follows;

Definition 3 *(Hierarchically Correlated Heavy Hitters) Let $\phi_p \in [0,1]$ and $\phi_s \in [0,1]$ are user defined parameters, F_m is the set of all HCHH, $F_{m_l} \subseteq F_m$ is the set of HCHH at level l of the hierarchy, where $\lambda \geq l \geq 1$, then:*
1. F_{m_λ} is the set of HCHH, which is computed by simply computing the CHH of T (using Definition 2) at level λ, and
2. F_{m_l} is the set of HCHH at level l of the hierarchy, which is computed as follows:

$$(f_{p_a} \geq \phi_p N) \wedge (f_{p_a,s_a} \geq \phi_s f_{p_a})$$

where

$$f_{p_a} = \sum_{(p \preccurlyeq p_a) \wedge (p \notin F_{m_{l+1}})} f_p$$

$$f_{p_a,s_a} = \sum_{(p \preccurlyeq p_a) \wedge (s \preccurlyeq s_a) \wedge ((p,s) \notin F_{m_{l+1}})} f_{p,s}$$

3. $F_m = \bigcup_{\forall l} F_{m_l}$

A naïve approach would be to apply the CHH definition at each level of generalization; at the leaves, but also for each internal node in the lattice. However, this definition will identify a large number of CHH at the ancestor level, simply because they have a descendant CHH. This fails to capture the complexity of the data distribution in the hierarchy. For example, is a node marked as CHH merely because it has a descendant which is CHH, or because the aggregation of its descendants makes it CHH? Therefore, the above definition of HCHH identifies those nodes whose frequencies do not include the frequencies of its descendant CHH pairs. In other words, any node can be a HCHH but only after discounting the frequencies of its descendant nodes that are themselves HCHH.

The Definitions 3 require space linearly proportional to the size of input for computing HCHH with exact frequencies. In the data stream model with constraint on the resources, an online version of this problem is defined as follows;

Definition 4 *(Online HCHH identification problem) Given a data stream of transactions* $\tau_i \in T$*, and user define parameters* ϵ_p*,* ϵ_s*,* $\phi_p \in [\epsilon_p, 1]$ *and* $\phi_s \in [\epsilon_s, 1]$*, then HCHH identification problem is to output a set* F_m *of correlated pairs of prefixes* $(p_a, s_a) : (p \preccurlyeq p_a) \wedge (s \preccurlyeq p_a)$*, and approximate bounds on the frequency of each item* $p_a \in F_m$ *and* $s_a \in F_m$*, such that the following conditions are satisfied;*

1. Accuracy:

$$\hat{f}_{p_a} \leq f_{p_a} + \epsilon_p N$$

$$\hat{f}_{p_a,s_a} \leq f_{p_a,s_a} + \epsilon_s f_{p_a}$$

2. Coverage:

$$\forall\ (p_a, s_a)\ \notin\ F_m, (f_{p_a} < \phi_p N) \wedge (f_{p_a,s_a} < \phi_s f_{p_a})$$

3 Proposed Algorithms

This section describes the proposed algorithms for flat (i.e., CHH) as well as hierarchical correlated heavy hitters (i.e., HCHH). First, we propose an algorithm for flat datasets (e.g., when p, and s are not hierarchical items), which we call OCHHA that has neither of the problem described above; it does not require the values of ϕ_p and ϕ_s apriori, has lower false positive rate, is memory optimal, and can be extended to design algorithm for HCHH computation. Next, we propose a hierarchy-aware algorithm, which is based on OCHHA, and we call it HCHHA.

In the following lemma, we show the memory bound for any optimal online algorithm for solving CHH problem in two dimensional data streams.

Lemma 2 *Any algorithm that solves approximate CHH problem from definition 2 requires at least* $\Omega(\frac{1}{\epsilon_p} + \frac{1}{\epsilon_p \epsilon_s})$ *memory.*

The standard definition of finding a set $\mathcal{H}$ of heavy hitters from a stream of N items is;

$$(\forall e \in \mathcal{H} \Rightarrow f_e \geq (\phi - \epsilon)N) \wedge (\forall e^{'} \notin \mathcal{H} \Rightarrow f_{e'} < \phi N)$$

which requires all items e with $f_e \geq \epsilon N$ to be maintained. It has been shown in [5] that a summary of size $\Omega(\frac{1}{\epsilon})$ is needed to maintain all items e with $f_e \geq \epsilon N$, and to ensure ϵN error guarantee. Now, the definition of finding a set F_c of CHH from a stream of N pairs is;

$$(\forall p \in F_c \Rightarrow f_p \geq (\phi_p - \epsilon_p)N) \wedge (\forall p^{'} \notin F_c \Rightarrow f_{p'} < \phi_p N) \wedge \tag{3.1}$$

$$(\forall (p,s) \in F_c \Rightarrow f_{p,s} \geq (\phi_s - \epsilon_s)f_p) \wedge (\forall (p^{'},s^{'}) \notin F_c \Rightarrow f_{p',s'} < \phi_s f_p) \tag{3.2}$$

Equation 3.1 is exactly same as the definition of heavy hitters set $\mathcal{H}$, which require $\Omega(\frac{1}{\epsilon_p})$ memory. By analogy to $\mathcal{H}$ equation 3.2 requires $(f_{p,s} \geq \epsilon_s f_p)$ to be also satisfied by CHH algorithm. Now using $f_p \geq \epsilon_p N$, we can get the following facts;

$$f_p \geq \epsilon_p N \Rightarrow \epsilon_s f_p \geq \epsilon_s \epsilon_p N$$

$$f_{p,s} \geq \epsilon_s f_p \geq \epsilon_s \epsilon_p N$$

Which shows that $(f_{p,s} \geq \epsilon_s f_p)$ requirements can also be satisfied by $(f_{p,s} \geq \epsilon_s \epsilon_p N)$, which needs at least $\Omega(\frac{1}{\epsilon_s \epsilon_p})$ memory. Thus, to satisfy equation 3.1 and 3.2, we require $\Omega(\frac{1}{\epsilon_p}) + \Omega(\frac{1}{\epsilon_s \epsilon_p}) = \Omega(\frac{1}{\epsilon_p} + \frac{1}{\epsilon_s \epsilon_p})$.

3.1 Optimal CHH Algorithm (OCHHA)

This section describes OCHHA in detail, and develops theoretical bounds on memory usage and deterministic error guarantees in answer qualities. OCHHA creates two sketches, $\mathcal{SS}_{p,s}$ and $\mathcal{SS}_p$. The sketch $\mathcal{SS}_{p,s}$ contains a set of, at most, $k_s = \frac{1}{\epsilon_s \epsilon_p}$ tuples of the form $[(p,s), g_{p,s}, \Delta_{p,s}]$, where pair (p,s) is the key of the tuple, $g_{p,s}$ is the approximate frequency of pair (p,s), $\Delta_{p,s}$ is the error in approximation of frequency of pair (p,s) in stream S. Similarly, the sketch $\mathcal{SS}_p$ contains a set of, at most, $k_p = \frac{1}{\epsilon_p}$ tuples of the form $[p, g_p, \Delta_p]$, where primary item p is the key of the tuple, g_p is the approximate frequency of item p, Δ_p is the error in approximation of frequency of item p.

High level concept of OCHHA: we process each pair (p,s) from the stream S using Space Saving [25] sketch $\mathcal{SS}_{p,s}$, which maintains a summary of size k_s. The main idea behind OCHHA is to process, any infrequent pair that is deleted from the sketch $\mathcal{SS}_{p,s}$ by another Space Saving sketch $\mathcal{SS}_p$, which maintains a summary of size k_p. The difference between sketch $\mathcal{SS}_{p,s}$ and sketch $\mathcal{SS}_p$ is that the sketch $\mathcal{SS}_p$ only processes the infrequent primary items p that are deleted from sketch $\mathcal{SS}_{p,s}$, hence, sketch $\mathcal{SS}_p$ contains primary items p that belong to the tail/infrequent pairs of stream S.

Now we describe OCHHA in more detail. For each incoming tuple (p_i,s_i,w_i), OCHHA performs one of the following actions (see Algorithm 1);

- if the pair (p_i, s_i) is present in $\mathcal{SS}_{p,s}$, OCHHA increments the corresponding counter using $g_{p_i,s_i} += w_i$;
- if the pair (p_i, s_i) is not present in $\mathcal{SS}_{p,s}$, then the pair with the smallest counter in $\mathcal{SS}_{p,s}$ (say (p,s)) is removed from $\mathcal{SS}_{p,s}$; next, it is replaced by the pair (p_i, s_i); next, the counter of the pair (p_i, s_i) is set to $g_{p_i,s_i} = g^{'}_{p,s} + w_i$; and finally, the error of the pair (p_i, s_i) is set to $\Delta_{p_i,s_i} = g^{'}_{p,s}$. Next, the deleted pair (p,s) from $\mathcal{SS}_{p,s}$ is processed by the second sketch $\mathcal{SS}_p$ as follows: the primary item p is taken out from the deleted pair

Algorithm 1 OCHHA

1: **procedure** INITIALIZE(ϵ_p, ϵ_s)
2: Set $k_p = \left\lceil \frac{1}{\epsilon_p} \right\rceil$, $k_s = \left\lceil \frac{1}{\epsilon_p \epsilon_s} \right\rceil$;
3: **end procedure**
4: **function** UPDATE-$\mathcal{SS}_{p,s}(p_i, s_i, w_i)$
5: **if** $(p_i, s_i) \in \mathcal{SS}_{p,s}$ **then**
6: $g_{p_i,s_i} += w_i$;
7: **else**
8: let $g'_{p,s} = min_{\forall (p,s) \in \mathcal{SS}_{p,s}}(g_{p,s})$;
9: replace (p_i, s_i) with (p, s) in $[(p, s), g_{p,s}, \Delta_{p,s}]$;
10: $c = g'_{p,s} - \Delta_{p,s}$;
11: Insert(p, c);
12: $\Delta_{p_i,s_i} = g'_{p,s}$;
13: $g_{p_i,s_i} = g'_{p,s} + w_i$;
14: **end if**
15: **Return**(p,s,c);
16: **end function**
17: **function** UPDATE-$\mathcal{SS}_p(p_i, w)$
18: **if** $p_i \in \mathcal{SS}_p$ **then**
19: $g_{p_i} += w$;
20: **else**
21: let $g'_p = min_{\forall p \in \mathcal{SS}_p}(g_p)$;
22: replace p_i with p in $[p, g_p, \Delta_p]$;
23: $c = g'_p - \Delta_p$;
24: $\Delta_{p_i} = g'_p$;
25: $g_{p_i} = g'_p + w$;
26: **end if**
27: **Return**(p,c)
28: **end function**
29: **procedure** OUTPUT(ϕ_p, ϕ_s)
30: $F_c = \{\}, F_p = 0$
31: $F_p = \sum_{\forall (p_i,s_i) \in \mathcal{SS}_{p,s} : p_i = p} (g_{p,s} - \Delta_{p,s})$
32: $F_p += g_p$
33: **for each** $(p, s) \in \mathcal{SS}_{p,s}$ **do**
34: **if** $(F_p \geq \phi_p N) \wedge (g_{p,s} \geq \phi_s (F_p - \epsilon_p N^{del}))$ **then**
35: Print$(p, F_p, \Delta_p, s, g_{p,s}, \Delta_{p,s})$
36: $F_c = F_c \cup (p, s)$
37: **end if**
38: **end for**
39: **end procedure**

(p, s); the count c of the pair (p, s) is calculated from sketch $\mathcal{SS}_{p,s}$ using $c = g'_{p,s} - \Delta_{p,s}$; and then the primary item p with count c is given to sketch $\mathcal{SS}_p$ (see lines 7-8 of Algorithm 1), where item p is processed by $\mathcal{SS}_p$ in the same manner as the pair (p_i, s_i) is processed by sketch $\mathcal{SS}_{p,s}$.

Example: It is easier to understand how OCHHA works with an example. Figure 3.1 illustrates the runtime mechanism of OCHHA for processing the stream of pairs. For this example, we assume $|\mathcal{SS}_{p,s}| = \frac{1}{\epsilon_p \epsilon_s} = 4$ and $|\mathcal{SS}_p| = \frac{1}{\epsilon_p} = 2$. Next, we explain each step of the algorithm in detail.

1. Figure 3.1a shows the current status of OCHHA, where $\mathcal{SS}_{p,s}$ is full and $\mathcal{SS}_p$ is empty. The OCHHA is ready to process the next pair {(x, d), 3} from the stream.

2. Figure 3.1b shows that the OCHHA has received the pair {(x, d), 3}; first, it compares {(x, d)} with tuples in $\mathcal{SS}_{p,s}$. Since, there is no tuple in $\mathcal{SS}_{p,s}$ that matches the key (x, d) and $\mathcal{SS}_{p,s}$ is full, therefore, OCHHA finds a tuple [(x, a), 1, 0] in $\mathcal{SS}_{p,s}$ with the smallest count, and replaces it with

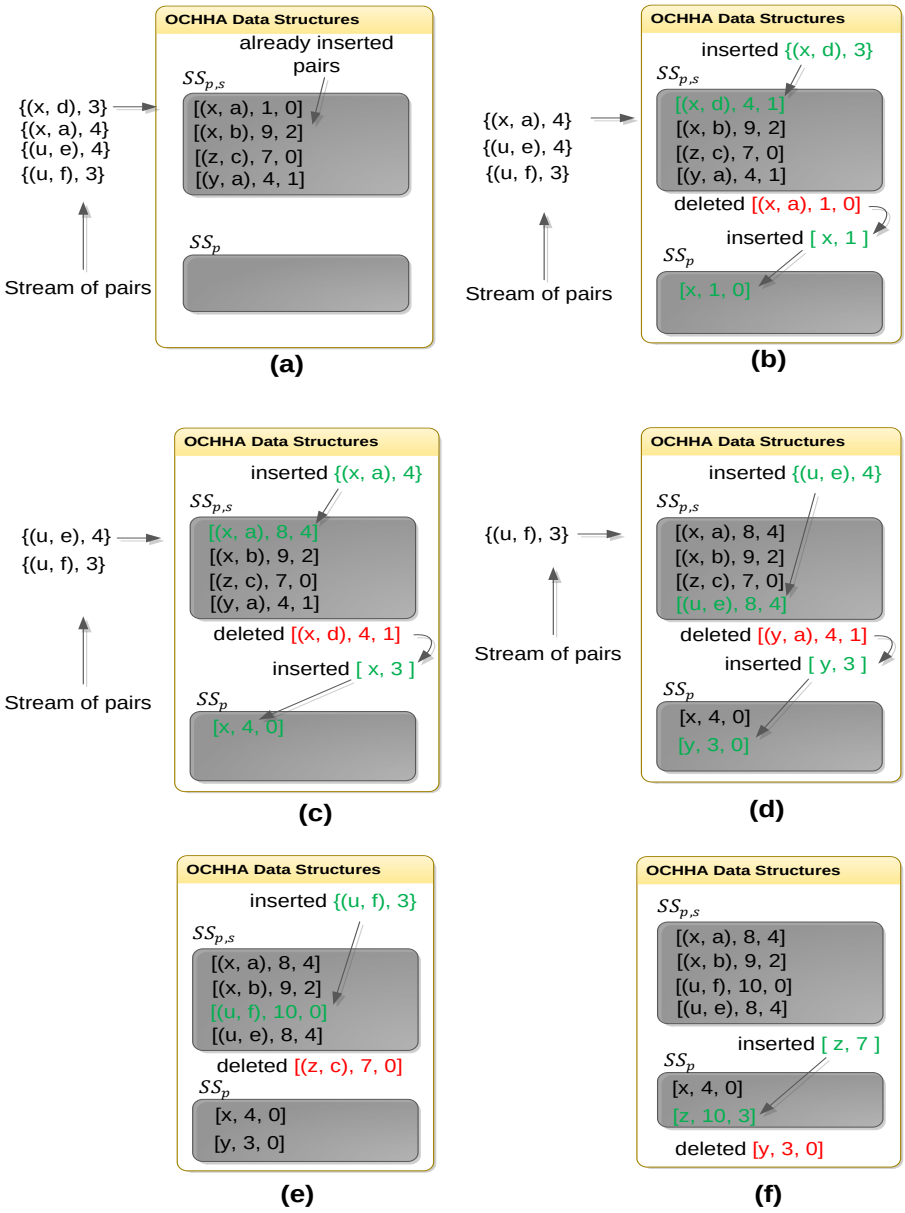


Figure 3.1: An illustration of various steps of OCHHA to process a stream of pairs that contain entries $\{(p, s), w\}$, where p is the primary item, s is the secondary item and w is the frequency of the pair (p, s).

[(x, d), 4, 1], where $g_{x,d} = 4$ and $\Delta_{x,d} = 1$. Next, OCHHA computes {x, 1} from the deleted tuple [(x, a), 1, 0] and inserts {x, 1} into $\mathcal{SS}_p$.

3. In Figure 3.1c the OCHHA receives the pair {(x, a), 4} and compared {(x, a)} with tuples in $\mathcal{SS}_{p,s}$. Since, there is no tuple in $\mathcal{SS}_{p,s}$ that matches the key (x, d) and $\mathcal{SS}_{p,s}$ is full, therefore, OCHHA finds a tuple [(x, d), 4, 1] in $\mathcal{SS}_{p,s}$ with the smallest count, and replaces it with [(x, d), 8, 4],

where $g_{x,a} = 8$ and $\Delta_{x,a} = 4$. Next, OCHHA computes {x, 3} from the deleted tuple [(x, d), 4, 1] and inserts {x, 3} into $\mathcal{SS}_p$. As x is found in $\mathcal{SS}_p$, therefore, OCHHA just increments the count of x using $g_x = 1+3$.

4. In Figure 3.1d the OCHHA inserts incoming pair {(u, e), 4} that cause tuple [(y, a), 4, 1] to be deleted from $\mathcal{SS}_{p,s}$. The OCHHA computes {y, 3} from the deleted tuple [(y, a), 4, 1] and inserts {y, 3} into $\mathcal{SS}_p$.

5. In Figure 3.1e the OCHHA inserts the incoming pair {(u, f), 3} that cause tuple [(z, c), 7, 0] to be deleted from $\mathcal{SS}_{p,s}$.

6. Figure 3.1f shows that the OCHHA computes {z, 7} from the deleted tuple [(z, c), 7, 0] and compares z with tuples in $\mathcal{SS}_p$. Since, there is no tuple in $\mathcal{SS}_p$ that matches the key z and $\mathcal{SS}_p$ is full, therefore, OCHHA finds a tuple [y, 3, 0] in $\mathcal{SS}_p$ with the smallest count, and replaces it with [z, 10, 3], where $g_z = 10$ and $\Delta_z = 3$.

As it can be seen a primary item p may exist in both sketches $\mathcal{SS}_{p,s}$ and $\mathcal{SS}_p$ (e.g., x exists in both sketches $\mathcal{SS}_{p,s}$ and $\mathcal{SS}_p$, see Figure 3.1), therefore, to find the approximate frequency of primary item p at query time, we have to calculate the estimated frequency using both sketches (see lines 25-26 of Algorithm 1);

$$\hat{f}_p = \sum_{\forall (p_i,s_i) \in \mathcal{SS}_{p,s} : p_i = p} (g_{p_i,s_i} - \Delta_{p_i,s_i}) + g_{p_i}$$

The approximate frequency of pair (p,s) can be found simply from sketch $\mathcal{SS}_{p,s}$ using $\hat{f}_{p,s} = g_{p,s}$. For example, if we want to find $\hat{f}_x$, OCHHA calculates it as follows.

$$\hat{f}_x = (g_{x,a} - \Delta_{x,a}) + (g_{x,b} - \Delta_{x,b}) + g_x = (8-4) + (9-2) + 4 = 15$$

To find the approximate frequency of a pair, say (x, a), OCHHA finds it directly from sketch $\mathcal{SS}_{p,s}$, which is $\hat{f}_{x,a} = g_{x,a} = 8$.

Next, in order to provide bounds on the accuracy of OCHHA, we proceed as follows: let N^{del} represent the sum of all items processed by $\mathcal{SS}_p$ (i.e., the sum of count of deleted pairs from sketch $\mathcal{SS}_{p,s}$), then OCHHA has the following estimation guarantees;

Lemma 3 (Accuracy)

$$\hat{f}_p \leq f_p + \epsilon_p N^{del}$$

$$\hat{f}_{p,s} \leq f_{p,s} + \epsilon_s f_p$$

Observe that, the only error that we have in the estimation of primary item p is due to items that are deleted from sketch $\mathcal{SS}_p$, thus, the error in estimation of frequency of primary item p is relative to N^{del}. Since, sketch $\mathcal{SS}_p$ uses $O(\frac{1}{\epsilon_p})$ memory, therefore, this error is bounded by $\epsilon_p N^{del}$, i.e., $\hat{f}_p \leq f_p + \epsilon_p N^{del}$. Similarly, the sketch $\mathcal{SS}_{p,s}$ provides $\hat{f}_{p,s} \leq f_{p,s} + \epsilon_p \epsilon_s N$ guarantees due to Space Saving algorithm [25]. Since, $\epsilon_s f_p \geq \epsilon_s \epsilon_p N$ (see lemma 2), therefore, the sketch $\mathcal{SS}_{p,s}$ provides $\hat{f}_{p,s} \leq f_{p,s} + \epsilon_s f_p$ guarantees. The overall memory required by OCHHA is $\frac{1}{\epsilon_p} + \frac{1}{\epsilon_s \epsilon_p} = O(\frac{1}{\epsilon_s \epsilon_p})$, which is optimal. For arbitrary positive

counter updates, using a suitable min-heap based implementation of Space Saving, OCHHA updates will take $O(\log \frac{1}{\epsilon_s \epsilon_p})$ time, and lookups will require O(1) time. When all updates are unitary (of the form $w_i = 1$), both insertions and lookups can be processed in $O(1)$ time using the Stream Summary data structure [25].

Based on the information available in two sketches $\mathcal{SS}_{p,s}$ and $\mathcal{SS}_p$, OCHHA outputs any pair (p,s) when the frequency of the pair (p,s) satisfies the following threshold (see lines 28 of Algorithm 1);

$$(F_p \geq \phi_p N) \wedge (g_{p,s} \geq \phi_s (F_p - \epsilon_p N^{del}))$$

where $F_p = \hat{f}_p$ and $g_{p,s} = \hat{f}_{p,s}$. The following lemma provides useful observation on properties of the CHH pairs that are output by OCHHA.

Lemma 4 (Coverage) *OCHHA satisfies the coverage property from definition 2.*

The criteria for a pair $(p, s) \in F_c$ used in the output subroutine of OCHHA can be written as follows (see lines 26-30 of Algorithm 1);

$$(F_p \geq \phi_p N) \wedge (g_s \geq \phi_s (F_p - \epsilon_p N^{del}))$$

$$\Rightarrow (\hat{f}_p \geq \phi_p N) \wedge (\hat{f}_{p,s} \geq \phi_s (\hat{f}_p - \epsilon_p N^{del})) \tag{3.3}$$

Now using $f_p \leq \hat{f}_p$ and bounds from lemma 3, the above equation becomes;

$$\Rightarrow (f_p + \epsilon_p N^{del} \geq \phi_p N) \wedge (f_{p,s} + \epsilon_s f_p \geq \phi_s (f_p - \epsilon_p N^{del}))$$

$$\Rightarrow (f_p \geq \phi_p N - \epsilon_p N^{del}) \wedge (f_{p,s} \geq (\phi_s - \epsilon_s) f_p - \phi_s \epsilon_p N^{del}) \tag{3.4}$$

Similarly, in order to find the bounds on pairs $(p, s) \notin F_c$, we use the inverse of equation 3.3, i.e.,

$$\Rightarrow (\hat{f}_p < \phi_p N) \wedge (\hat{f}_{p,s} < \phi_s (\hat{f}_{p,s} - \epsilon_p N^{del}))$$

Again using the bounds from lemma 3, the above equation becomes;

$$\Rightarrow (f_p < \phi_p N) \wedge (f_{p,s} < \phi_s f_p) \tag{3.5}$$

Hence, we conclude from equation 3.4 that $\forall (p, s) \in F_c$, $((f_p \geq \phi_p N - \epsilon_p N^{del}) \wedge (f_{p,s} \geq (\phi_s - \epsilon_s) f_p))$, and from equation 3.5, $\forall\ (p^{'}, s^{'}) \notin F_c, ((f_{p'} < \phi_p N) \wedge (f_{p',s'} < \phi_s f_p))$.

Theorem 1 *OCHHA satisfies all the requirements of definition 2 using memory* $O(\frac{1}{\epsilon_p \epsilon_s})$.

Theorem 1 follows directly from lemmas 3 and 4.

3.2 Hierarchical Algorithms

A naïve way of computing HCHH, using OCHHA, would be to find CHH over all prefixes of all the pairs in the data stream and then to discard extraneous pairs in a post processing step. We argue that this naïve strategy can be improved substantially in practice (in terms of the space usage and the answer quality)

by designing the hierarchy-aware algorithms that we design next. In Section 3.2 we present the naïve algorithm to compute hierarchically correlated heavy hitters, and in Section 3.2 we develop an efficient hierarchy-aware algorithm (i.e., HCHHA) with deterministic error guarantees. We consider cash register stream computational model, where new transactions only arrive and there are no deletions of previously seen transactions.

Naïve Algorithm

We first describe a naïve algorithm that uses OCHHA as a subroutine. We will use naïve algorithm as a baseline to compare various results. Basically, this algorithm maintains information for every label (node) in the lattice, by maintaining η independent instances of OCHHA, where each one computes (approximately) CHH for that node in the lattice (see Fig 2.1 for an example of lattice). The naïve algorithm works as follows. For every update τ, we compute all generalizations of τ and insert each one separately into respective HCHHA in the lattice. The output of the naïve algorithm is the union of all the outputs from each instance of HCHHA. The output from the naïve algorithm is a superset of the approximate HCHH (see Definition 4), and satisfies the accuracy and coverage requirements for HCHH. Nevertheless, it is a costly strategy in terms of space usage, as well as update costs.

Since, we use η independent instances of OCHHA and place each update into each of these η instances, the naïve algorithm has $O(\frac{\eta}{\epsilon_p \epsilon_s})$ overall space bound. Moreover, since OCHHA processes each update in constant time, the naïve algorithm requires $O(\eta)$ updates to process each τ.

Hierarchically CHH Algorithm (HCHHA)

The main idea behind the HCHHA is to create a two-dimensional lattice (similar to Fig 2.1b) with η nodes, and in each node maintain a list of CHH using a frequency discounting strategy (more details to follow). For a given HCHH query, the algorithm outputs HCHH by starting from the lowest level of the lattice and progressing towards the top. At each level, it outputs all the CHH pairs by properly discounting the frequencies of CHH maintained at lower levels of the lattice using Definition 4.

HCHHA maintains a sketch $\mathcal{SS}^L_{p,s}$ (that we have described for OCHHA algorithm) at each node of the lattice, where L represents the label of the node in the lattice. Notice , unlike the naïve algorithm, the HCHHA does not maintain $\mathcal{SS}^L_p$ sketches at each node of the lattice, except the nodes on the left most diagonal of the lattice (e.g., nodes (4,4), (3,4), (2,4) and (1,4) in Fig 2.1b). This is because in the naïve algorithm the sketch $\mathcal{SS}^L_p$ maintains exactly the same information at each diagonal of the lattice. Since we maintain two sketches (similar to OCHHA) at various nodes in the lattice, therefore, we first explain the operations on sketch $\mathcal{SS}^L_{p,s}$ in the lattice and then explain operations on sketch $\mathcal{SS}^L_p$ in the lattice. Each incoming pair from the stream is inserted into sketch $\mathcal{SS}^{(4,4)}_{p,s}$ at the lowest level of the lattice (e.g., node (4,4) in Fig 2.1b). Whenever a pair becomes infrequent and is deleted from sketch $\mathcal{SS}^{(4,4)}_{p,s}$ at the lower level, its frequency is added to its ancestors in sketches $\mathcal{SS}^{(3,4)}_{p,s}$ and $\mathcal{SS}^{(4,3)}_{p,s}$ at the immediate higher level; such process is known as generalization as explained in

Algorithm 2 Update procedure of HCHHA

Require: ϵ_p and ϵ_s to initialize sketches $\mathcal{SS}^L_{p,s}$ with $\frac{1}{\epsilon_p \epsilon_s}$ counters at each node L of the lattice, and $\mathcal{SS}^L_p$ with $\frac{1}{\epsilon_p}$ counters at nodes $L = \{(4,4),(3,4),(2,4),(1,4)\}$ of the lattice.

procedure UPDATE(p_i, s_i, w_i)
  $N+ = w_i$;
  $L = getLevelVector(p_i, s_i)$;
  (p, s, c) = Update-$\mathcal{SS}^L_{p,s}(p_i, s_i, w_i)$; ▷ See Update-$\mathcal{SS}_{p,s}(.)$ of Algorithm 1
  Update$_{p,s}(p, s, c)$;
  Update$_p(p, c)$;
end procedure
procedure UPDATE$_p(p, c)$
  $N^{del}+ = c$;
  $L = getLevel(p)$
  D = Update-$\mathcal{SS}^L_p(p, c)$; ▷ See Update-$\mathcal{SS}_p(.)$ of Algorithm 1
  for each (p, c) in D **do**
    $p_a = GetAncestor_p(p)$
    if p_a is not NULL **then**
      $L = getLevel(p_a)$
      D_p =Update-$\mathcal{SS}^L_p(p, c)$;
      Add tuples from D_p to the end of D;
    end if
  end for
end procedure
procedure UPDATE$_{p,s}$(p,s,c)
  $D = \{p, s, c\}$;
  for each (p, s, c) in D **do**
    Let $A = \{\}$ is a multidimensional set, initially empty
    $A = GetAncestors_{p,s}(p, s, c)$
    for each (p_a, s_a, c) in A **do**
      $L = getLevelVector(p_a, s_a)$
      $D_{p,s}$ = Update-$\mathcal{SS}^L_{p,s}(p_a, s_a, c)$;
      Add tuples from $D_{p,s}$ to the end of D;
    end for
  end for
end procedure
function GETANCESTORS$_{p,s}(p_i, s_i, c)$
  $(L_p, L_s) = getLevelVector(p_i, s_i)$
  Let $A = \{\}$ is a multidimensional set, initially empty
  $L = (L_p - 1, L_s)$
  if p_i is generalizable to L **then**
    $(p_a, s) = GeneralizeTo(p_i, s_i, L)$;
    $A = A \bigcup \{p_a, s, +c\}$
  end if
  $L = (L_p, L_s - 1)$
  if s_i is generalizable to L **then**
    $(p, s_a) = GeneralizeTo(R_i, L)$;
    $A = A \bigcup \{p, s_a, +c\}$
  end if
  $L = (L1 + 1, L2 + 1)$
  if (p_i, s_i) is generalizable to L **then**
    $(p_a, s_a) = GeneralizeTo(p_i, s_i, L)$;
    $A = A \bigcup \{p_a, s_a, -c\}$
  end if
  Return A;
end function
function GETANCESTOR$_p(p_i)$
  if p_i is generalizable to $L_p - 1$ **then**
    Return $GeneralizeTo(p_i, L_p - 1)$;
  else
    Return NULL;
  end if
end function

Section 2.3. The deletion can happen at higher levels too when a pair becomes infrequent there, which continues upwards towards the root node.

Since, the deleted pair has to pass its count to two immediate ancestors (or at most n ancestors for n-dimensional data), which introduces the "over-counting"

Algorithm 3 Output procedure of HCHHA

```
1: procedure OUTPUT(φ_p, φ_s)
2:     F_m = {};
3:     F_p = {(p, g_p) : g_p = Σ_{∀(p_i,s_i)∈SS^(4,4)_{p,s} : p_i=p} (g_{p,s} − Δ_{p,s})};
4:     for l = λ to 0 do
5:         for each L ∈ Level(l) do
6:             F_{p_a} = Σ_{(p∈SS^L_p ∨ p∈F_p)∧(p∉F_m)∧(p⋠p_a∈F_m)} g_p;
7:             f_{p_a} = Σ_{(p∈SS^L_p ∨ p∈F_p)} g_p;
8:             if F_{p_a} ≥ φ_p N then
9:                 for each (p, s) ∈ SS^L_{p,s} do
10:                    (p_a, s_a) = GeneralizeTo(p, s, L)
11:                    f_{p_a,s_a} += g_{p,s};
12:                    if ∄(p_h, s_h) ∈ F_m : ((p, s) ≼ (p_h, s_h)) ∧ ((p_h, s_h) ≼ (p_a, s_a)) then
13:                        F_{p_a,s_a} += g_{p,s};
14:                    end if
15:                end for
16:                for each (p_h, s_h) ∈ F_m do
17:                    (p_a, s_a) = ancestor((p_h, s_h), L)
18:                    if ∄(p_q, s_q) ∈ F_m : ((p_h, s_h) ≼ (p_q, s_q)) ∧ ((p_q, s_q) ≼ (p_a, s_a)) then
19:                        F_{p_a,s_a} += −g_{p,s};
20:                    end if
21:                end for
22:                for each (p_a, s_a) ∈ Level(l) do
23:                    if F_{p_a,s_a} ≥ φ_s f_p then
24:                        F_m = F_m ⋃{(p_a, s_a)};
25:                        Print (p_a, f_{p_a}, s_a, f_{p_a,s_a});
26:                    end if
27:                end for
28:            end if
29:        end for
30:    end for
31: end procedure
```

problem; well known in multidimensional hierarchical heavy hitters computation [31, 10]. The problem occurs when the count of a descendant node traverses up through multiple paths of a lattice and ends up in a single ancestor node. This means that the same child node's frequency is counted multiple times at an upper level node, which is clearly an error. This over-counting can be controlled using a inclusion-exclusion strategy defined in [31, 10], which inserts the count of each deleted pair into its immediate parents, and subtracts its count from common grandparent. This is done for each deleted pair while moving up in the lattice hierarchy to effectively solve the over-counting problem.

More specifically, for each incoming $\tau_i = (p_i, s_i, w_i)$, HCHHA inserts (p_i, s_i) into $\mathcal{SS}^{(4,4)}_{p,s}$ at the lowest node of the lattice, if $\mathcal{SS}^{(4,4)}_{p,s}$ returns no deleted pair then HCHHA continues to process next pair from the stream. Otherwise, HCHHA performs the following actions (see Algorithm 2); let $\mathcal{SS}^{(4,4)}_{p,s}$ returns a deleted pair (p, s) having count c (where c is defined previously in OCHHA) then:

1. The deleted pair (p, s) from $\mathcal{SS}^{(4,4)}_{p,s}$ is generalized to parent labels $L = (3, 4)$ and $L = (4, 3)$ and grandparent label $L = (3, 3)$. Then the generalized pair (p_a, s) and (p, s_a) with count c are given to sketches $\mathcal{SS}^{(3,4)}_{p,s}$ and $\mathcal{SS}^{(4,3)}_{p,s}$ respectively at the parent nodes of the lattice, and the generalized pair (p_a, s_a) with count $-c$ is given to sketch $\mathcal{SS}^{(3,3)}_{p,s}$ at the grandparent node of the lattice. The insertion of a pair (p, s) into $\mathcal{SS}^{L}_{p,s}$ at any ancestor label L may cause $\mathcal{SS}^{L}_{p,s}$ to return another deleted pair, which is generalized to its parent and grandparent labels in similar manner as explained

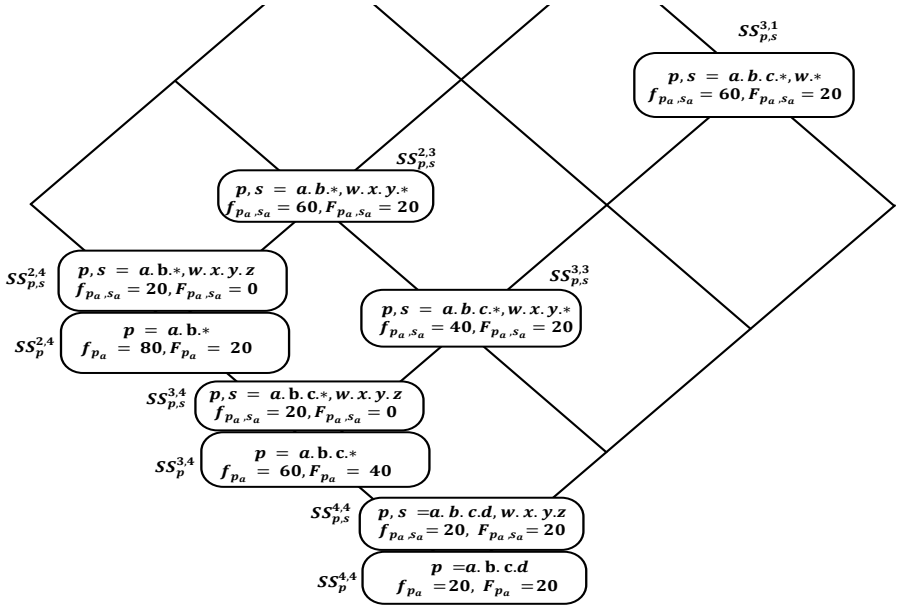


Figure 3.2: An illustration of HCHHA to process and output HCHH from a two-dimensional stream of IP address pairs that contain entries $\{(p, s)\}$, where p is the primary item corresponding to destination IP address and s is the secondary item corresponding to source IP address. Both p and s are hierarchical items at octet-based granularity. The exact HCHH contain a set of ordered pairs of destination-source IP address prefixes. The HCHH set contains those sources which co-occur most frequently for their corresponding destinations at various levels of their hierarchies. The terms f_{p_a} and F_{p_a} represent the full and discounted count of primary item p respectively. Similarly, the terms f_{p_a,s_a} and F_{p_a,s_a} represent the full and discounted count of pair p_a, s_a respectively. The stream contains 20 repetitions of the pairs (a.b.c.d, w.x.y.z), followed by pairs (a.b.c.i, w.x.y.i), (a.b.i.d, w.x.y.i), and (a.b.c.i, w.i.y.z), where $i = 1, 2, , \cdots, 20$. The alphabets a, b, w, y etc. are representing different integers between 10 and 255. To demonstrate the main concept, we have skipped the steps of insertion of pairs and only highlighted the HCHH pairs found for the specific stream described above (where $N = 80$) by the HCHHA at various nodes of the lattice for $\phi_p = 0.25$ and $\phi_s = 0.5$.

for deleted pair (p, s) above.

2. Also, the primary item p is taken out from the pair (p, s) which is deleted from $\mathcal{SS}^{(4,4)}_{p,s}$, and then the primary item p with count c is given to sketch $\mathcal{SS}^{(4,4)}_{p}$. Note that this operation is specific to node (4,4) only; i.e., at no other node this operation happens. However, when infrequent items p are deleted from sketch $\mathcal{SS}^{(4,4)}_{p}$, they are generalized and inserted to sketch $\mathcal{SS}^{(3,4)}_{p}$. Similarly, infrequent items p deleted from sketch $\mathcal{SS}^{(3,4)}_{p}$ are generalized and inserted to sketch $\mathcal{SS}^{(2,4)}_{p}$, and so on.

The output procedure of HCHHA (see Algorithm 3) is a complex subroutine that outputs HCHH pairs after properly discounting the counts of HCHH at lower level of the lattice (see Figure 3.2). It iterates through all the levels of the lattice, starting at the bottom level of the lattice and work towards the top, using the inclusion-exclusion strategy mentioned above. At each level it iterates through all the labels (nodes) belonging to that level. At each label (node) it iterates through all the items maintained by sketches $\mathcal{SS}^L_{p,s}$ and $\mathcal{SS}^L_p$ within that label of the lattice. The subroutine output any prefix pair whose estimated count exceeds the thresholds ϕ_p and ϕ_s.

For any record (p_a, s_a) that belongs to any level and any label at that level, the subroutine computes F_{p_a} and F_{p_a,s_a}, which are the sum of the counts of all descendants $p : p \not\preccurlyeq p_a \in F_m$ and $(p, s) : (p, s) \not\preccurlyeq (p_a, s_a) \in F_m$ respectively. In the case of computing F_{p_a,s_a} the subroutine checks if any pair has contributed twice to an ancestor pair, then the subroutine removes that count of the repeated pair from the ancestor pair (see loop at lines 17-22 of the pseudo code given in Algorithm 3). Notice that this is not required for computing F_{p_a} since an item p forms a tree instead of a lattice, and the sketches $\mathcal{SS}^L_p$ for storing items p are maintained at the nodes on the left most diagonal of the lattice (e.g., nodes (4,4), (3,4), (2,4) and (1,4) in Fig 2.1b).

Next, we provide proofs on computing discounted counts for HCHH identification by the proposed algorithm HCHHA.

Lemma 5 *If F_{p_a} is the count of items p or their descendants, such that:*

$$F_{p_a} = \sum_{(g_p < \phi_p N) \wedge (p \notin F_m) \wedge (p \not\preccurlyeq p_a \in F_m)} g_p$$

then

$$F_{p_a} \geq \phi_p N$$

provides the discounted count of p_a.

Since, at the lowest node (4,4) of lattice there are no descendants of p, therefore, F_{p_a} provides the discounted frequency of p_a.

At other upper nodes (e.g., (3,4),(2,4) and (1,4)) F_{p_a} is computed from items $p : (p \notin F_m) \wedge (p \not\preccurlyeq p_a \in F_m)$, which means neither p nor any ancestor of p belongs to set F_m, that is:

$$F_{p_a} = \sum_{(p \notin F_m) \wedge (p \not\preccurlyeq p_a \in F_m)} g_p$$

Since,

$$\forall p \in F_m, g_p \geq \phi_p N \Rightarrow \forall p^{'} \notin F_m, g_{p'} < \phi_p N$$

therefore,

$$F_{p_a} = \sum_{(g_p < \phi_p N) \wedge (p \notin F_m) \wedge (p \not\preccurlyeq p_a \in F_m)} g_p$$

Hence, the lemma follows.

Lemma 6 *If F_{p_a,s_a} is the count of pairs (p, s) or their descendants, such that:*

$$F_{p_a,s_a} = \sum_{(g_{p,s} < \phi_s f_p) \wedge ((p,s) \notin F_m) \wedge ((p,s) \prec (p_a,s_a) \notin F_m)} g_{p,s}$$

then

$$F_{p_a,s_a} \geq \phi_s f_p$$

provides the discounted count of pair (p_a, s_a).

Since by definition F_{p_a,s_a} is computed using:

$$F_{p_a,s_a} = \sum_{\nexists(p_h,s_h)\in F_m \,:\, ((p,s)\preccurlyeq(p_h,s_h))\wedge((p_h,s_h)\preccurlyeq(p_a,s_a))} g_{p,s} - \sum_{\nexists(p_q,s_q)\in F_m \,:\, ((p_h,s_h)\preccurlyeq(p_q,s_q))\wedge((p_q,s_q)\preccurlyeq(p_a,s_a))} g_{p,s}$$

See conditions at line 12 and 18 in Algorithm 3. Since;

$$\sum_{\nexists(p_h,s_h)\in F_m:((p,s)\preccurlyeq(p_h,s_h))\wedge((p_h,s_h)\preccurlyeq(p_a,s_a))} g_{p,s} = \sum_{((p,s)\notin F_m)} g_{p,s} \tag{3.6}$$

and,

$$\sum_{\nexists(p_q,s_q)\in F_m \,:\, ((p_h,s_h)\preccurlyeq(p_q,s_q))\wedge((p_q,s_q)\preccurlyeq(p_a,s_a))} g_{p,s} = \sum_{((p,s)\prec(p_a,s_a)\in F_m)} g_{p,s} \tag{3.7}$$

therefore, the subtraction of equation 3.7 from equation 3.6 implies;

F˙p˙a,s˙a $= \sum_{(g_{p,s}<\phi_s f_p)\wedge((p,s)\notin F_m)\wedge((p,s)\prec(p_a,s_a)\notin F_m)} g_{p,s}$ Hence, F_{p_a,s_a} provides the discounted count of pair (p, s).

Theorem 2 *HCHHA requires* $O(\frac{\eta}{\epsilon_p\epsilon_s})$ *space to satisfy the Accuracy and Coverage requirements from definition 4 and performs* $O(\eta)$ *updates per transaction from the stream S. The output operation takes* $O(\frac{\eta}{\epsilon_p\epsilon_s})$ *time.*

Since we initialize each instance of sketch $\mathcal{SS}_p^L$ by $\frac{1}{\epsilon_p}$ and $\mathcal{SS}_{p,s}^L$ by $\frac{1}{\epsilon_p\epsilon_s}$, therefore, the overall space required by the lattice is $\frac{h_p}{\epsilon_p} + \frac{\eta}{\epsilon_p\epsilon_s}$, where $\frac{h_p}{\epsilon_p}$ is the space required for maintaining $\mathcal{SS}_p^L$ sketches and $\frac{\eta}{\epsilon_p\epsilon_s}$ the space required for maintaining $\mathcal{SS}_{p,s}^L$ sketches. The $\mathcal{SS}_p^L$ sketches can estimate the frequency of each item p inserted into it within an error $\epsilon_p N$ and the $\mathcal{SS}_{p,s}^L$ sketches can estimate the frequency of each pair (p, s) inserted into it within an error $\epsilon_s f_p$. As a result, the Accuracy requirements from Definition 4 is satisfied using overall $\frac{h_p}{\epsilon_p} + \frac{\eta}{\epsilon_p\epsilon_s} = O(\frac{\eta}{\epsilon_p\epsilon_s})$ space. Finally, with this space, combined with lemmas 5 and 6 the Coverage requirements are satisfied.

Next, to show the update cost, we proceed as follows. As described in algorithm 2, only deleted pairs from lower nodes are inserted into upper nodes. In the worst case, a pair may be deleted from each node and eventually find its way to reach the top most node. In practice, we expect this to happen very infrequently and only for a fraction of pairs, however, the worst case bound is that items may traverse all the nodes, i.e., η nodes, so the worst case update cost is $O(\eta)$ per incoming pair from the stream S, because each insertion for sketches roughly requires $O(1)$ time (if stream updates are unitary).

For the output operation the algorithm needs to scan all the tuples maintained in the sketches at different nodes of the lattice. Even though, there are searches involving the scan of HCHH output set F_m, the overall cost of output is dominated by the linear scan of the sketches at different nodes of lattice, which requires $O(\frac{\eta}{\epsilon_p\epsilon_s})$ time.

4 Implementation and Evaluation

We have implemented all our proposed algorithms and existing CHH algorithm [19, 18] using Java (v 1.8); for the sake of clarity, we have labeled existing CHH algorithm as EXIST. Our implementation of EXIST is based on the pseudo code provided in [18], including all the optimizations described in [18]. For testing, we have used an Intel Core i7 machine with 3.4GHz processor, 16 GB RAM, and 64 bit Windows operating system installed on it. We have compared CHHA, ICHHA, OCHHA and EXIST for a range of parameters i.e., ϵ_p, ϵ_s, ϕ_p and ϕ_s. The data structures used are based on hashing techniques; it requires one hashing operation to lookup a particular item in the data structure.

Evaluation Criteria: To measure the ***efficiency*** and ***effectiveness*** of our proposed algorithms, we have considered different factors including ***memory usage***, and ***quality*** of the output using exact answer as a frame of reference. The exact answer is computed for benchmarking purposes by an algorithm described in [19, 18]. The memory usage is compared using the maximum number of bytes used by the algorithms during runtime. The quality of output is compared using a number of measures, including False Positive Rate, Precision, Recall, Dice Coefficient, Average and Maximum Error.

In the first set of experiments, we provide comparisons and experimental results of CHH algorithms. In the next set of experiments, we provide comparisons and experimental results of HCHH algorithms.

4.1 Experiments on Flat Data

In this section, we compare OCHHA and EXIST for various parameters settings.

Datasets: In these experiments, we have used real Internet traffic traces datasets [26] that are openly available from the WAND Network Research Group from the University of Waikato, New Zealand. Each of these datasets contain 30 minutes trace of network traffic traces in tcpdump[1] format. Wireshark[2] was used to read the tcpdump format data to extract source and destination IP pairs as two dimensional data for testing; where destination IP is considered as primary item p and source IP is considered as secondary item s. Note that we have performed additional experiments on a number of other real word datasets including Google n-grams dataset[3] and other traces to confirm that the results are demonstrative. The results of those experiments are not included due to redundancy and space, since those experiments illustrate similar results.

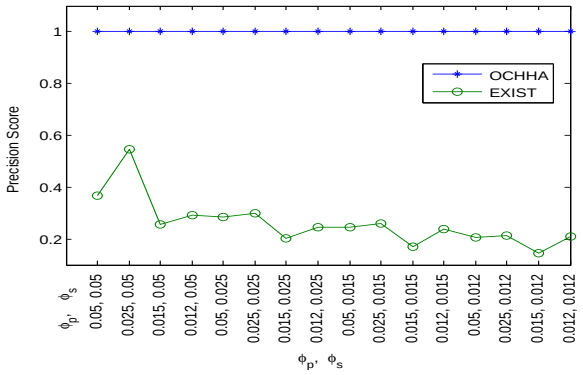
Notice that the proposed algorithms can handle both unitary as well as arbitrary updates, however, EXIST algorithm can only handle unitary updates. Therefore, to ensure consistency, in all our experiments, we have used the frequency of IP address pair to be the number of packets associated with that pair, instead of the number of bytes of raw IP packet. Consequently, N refers to the length of stream or the number of packets processed thus far.

We have evaluated the proposed algorithms for various parameter settings, and plotted the results against these settings as indicated at the x-axis of all the figures. In all our experiments, we have used various levels of the threshold ϕ_p for primary item p and ϕ_s for secondary item s. In each experiment variations

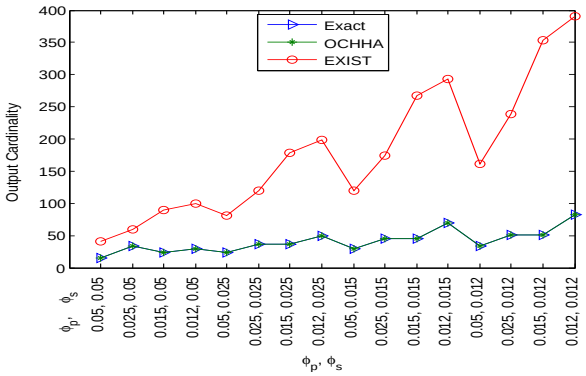
[1] http://www.tcpdump.org/manpages/tcpdump.1.html, Accessed: 23/02/2015
[2] https://www.wireshark.org/, Accessed: 23/02/2015
[3] http://storage.googleapis.com/books/ngrams/books/data setsv2.html

in ϵ_p and ϵ_s are achieved by assigning them to different ϕ_p and ϕ_s values as $\epsilon_p = \frac{\phi_p}{5}$ and $\epsilon_s = \frac{\phi_s}{5}$. The thresholds ϕ_p and ϕ_s are gradually decreased, as can be seen from figures, to observe the performance of algorithms for a variety of thresholds.

Overall, the OCHHA outperforms the EXIST algorithm both in terms of output quality and memory usage. Below, we discuss each aspect in more detail.



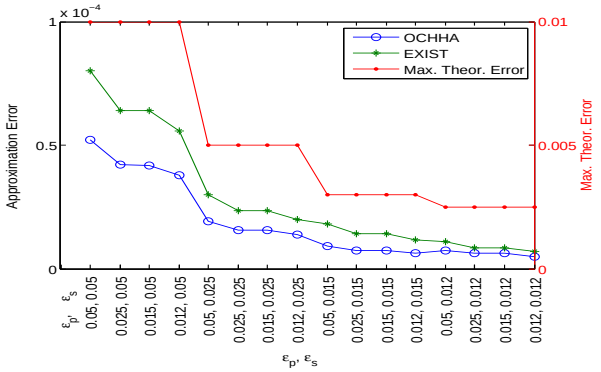
Comparison in terms of precision
(a)



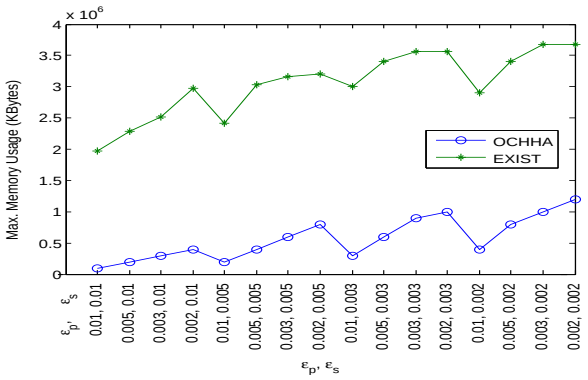
Comparison in terms of output cardinality
(b)

Figure 4.1: Evaluation of OCHHA and EXIST algorithm for the following parameters settings; $\epsilon_p = \frac{\phi_p}{5}$, $\epsilon_s = \frac{\phi_s}{5}$.

Output Quality: One of the most important issues with approximate algorithms, such as CHH algorithms, is that these techniques are not guaranteed to find the exact set of CHH. Thus to evaluate the proposed algorithms, we have compared them using Precision, Recall, and Dice coefficient measures using exact answer as a frame of reference. Let $\mathcal{C}_o$ represents the number of true CHH pairs (i.e., a pair output by the algorithm is present in the exact answer) output by the algorithm, $\mathcal{C}_t$ represents the number of exact CHH pairs in the stream, and $\mathcal{C}_a$ represents the total number of CHH pairs (i.e., both true and false pairs) output by the algorithm, then in the context of CHH, Precision,



Comparison in terms of average approximation error
(a)



Comparison in terms of memory requirements
(b)

Figure 4.2: Evaluation of OCHHA and EXIST algorithm for the following parameters settings; $\epsilon_p = \frac{\phi_p}{5}$, $\epsilon_s = \frac{\phi_s}{5}$.

Recall, and Dice coefficient can formally be defined as:

$$Precision = \frac{\mathcal{C}_o}{\mathcal{C}_a}, Recall = \frac{\mathcal{C}_o}{\mathcal{C}_t}, Dice = \frac{2\mathcal{C}_o}{\mathcal{C}_t + \mathcal{C}_a}$$

Recall is the measure of the ability of an algorithm to output all CHH pairs, while precision is the measure of the ability of an algorithm to output only CHH pairs. Recall of an algorithm is high if it does not miss any CHH pair (even if it outputs a lot of non-CHH pairs), and precision of an algorithm is high if it does not output any non-CHH pair (even if it misses a lot of true CHH pairs). The relative importance of recall and precision is subjective; in the context of document retrieval high precision is desirable, since missing a relevant document is acceptable as compared to overwhelming the user with a lot of irrelevant documents. Similarly, in the context of intrusion detection high recall is desirable, since detecting a legitimate event as intrusion (i.e., accepting false positives) is acceptable as compared to missing an intrusion altogether.

An interesting observation is (following lemma 1), approximate algorithms for computing CHH in data streams are guaranteed to have 100% recall, because they do not miss any true CHH pair, however, 100% precision is not guaranteed. Therefore, we have evaluated the proposed algorithms in terms of precision, dice (which is the harmonic mean of precision and recall) and output cardinality.

Figure 4.1a compare the proposed CHH algorithm against EXIST in terms of the precision measure. Precision score varies between 0 and 1; where the lower score indicates poor performance and the higher score indicates better performance. For Figure 4.1a, it can be seen that OCHHA performs superior than EXIST. Precision score of EXIST gradually decreases with lower thresholds of ϕ_p and ϕ_s, while for OCHHA it remains unchanged for different thresholds of ϕ_p and ϕ_s. This clearly demonstrates that EXIST outputs a significant number of false CHH pairs (in addition to true CHH pairs); while OCHHA outputs virtually no false CHH pairs. This can also be verified from figures 4.1b, where we have plotted the output cardinality for the same settings as those of Figure 4.1a.

In general, the number of CHH pairs increases with lower values of ϕ_p and ϕ_s because many correlated pairs satisfies the threshold. Approximate algorithms require to lower the thresholds ϕ_p and ϕ_s further by a factor of ϵ_p and ϵ_s in order to safeguard against missing the true CHH pairs, due to uncertainty in the estimated frequencies of items maintained in summary. Consequently, such approach may cause the algorithm to output some false positive CHH pairs, however, we have observed this effect to be worse for EXIST than OCHHA.

We have also evaluated the output quality of the propose algorithms in terms of average relative estimation error in the frequencies of CHH pairs output by the algorithms. We define the average relative estimation error $\mathcal{A}$ of the output as;

$$\mathcal{A}_p = \frac{\sum_{p \in F_c} \frac{|\hat{f}_p - f_p|}{N}}{\mathcal{C}_a}, \mathcal{A}_{p,s} = \frac{\sum_{(p,s) \in F_c} \frac{|\hat{f}_{p,s} - f_{p,s}|}{f_{p,s}}}{\mathcal{C}_a}$$

where the subscript p and p, s indicates primary and secondary items respectively. Clearly, $\mathcal{A}_p$ is less than the maximum theoretical error ϵ_p, and $\mathcal{A}_{p,s}$ is less than the maximum theoretical error ϵ_s. Here, we provide results for $\mathcal{A}_{p,s}$ only, because in all our experiments there is little difference in the values of $\mathcal{A}_p$.

Figure 4.2a compare the algorithms in terms of $\mathcal{A}_{p,s}$ and $\mathcal{M}_{p,s}$ measures. Note that, $\mathcal{A}_{p,s}$ and $\mathcal{M}_{p,s}$ are plotted on the left y-axis and maximum theoretical error ϵ_s on the right y-axis of figures 4.2a . This is because the relative estimation error for both the algorithms is very small compared to maximum theoretical error, and plotting them on the same scale (or even on the log scale) would not show the difference between algorithms. From Figure 4.2a, OCHHA has the lowest average relative estimation error; while, CHHA has the highest average relative estimation error. The relative estimation error of EXIST is higher than OCHHA.

In summary, the output quality of OCHHA is superior to the existing algorithm, in terms of all the measures, i.e., precision, recall, dice, output cardinality, and average estimation error.

Memory Usage: In this section we explain the memory usage of the algorithms. Although, we have given theoretical memory usage in terms of maximum number of tuples that our algorithms can maintain, we are providing results on practical memory usage in terms of maximum actual bytes used. Figure 4.2b shows the memory usage of OCHHA and EXIST for various values of ϵ_p and

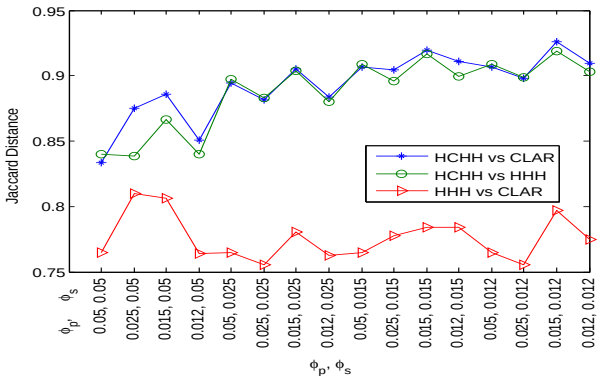


Figure 4.3: Comparisons of HCHH, cross-level association rules (CLAR) and hierarchical heavy hitters (HHH) in terms of jaccard distance for various values of ϕ_p and ϕ_s.

ϵ_s. Note that the memory usage is independent of ϕ_p and ϕ_s. From Figure 4.2b, OCHHA uses smallest amount of memory than EXIST algorithm. The memory requirements for both the algorithms increase, as expected, with the lower values of ϵ_p and ϵ_s, however, for EXIST this increase is more prominent than OCHHA.

We conclude that, OCHHA is better than EXIST algorithm in terms of memory usage and output quality. OCHHA is based on a simple design; it can be implemented using any available off-the-shelf implementation of heavy hitter algorithms. Thus, it offers an easy to implement tool for finding CHH from two dimensional data streams.

4.2 Experiments on Hierarchical Data

This section presents experimental results of HCHH algorithms. First, we compare our newly proposed HCHH notion with other hierarchical notions of interesting elements, such as cross-level association rules mining and hierarchical heavy hitters. Next, we compare HCHHA with naïve algorithm in terms of output quality and memory usage.

Comparisons of HCHH, cross-level association rules and hierarchical heavy hitters

Datasets: In these experiments, we have used a clickstream dataset from an anonymous e-commerce website, which contains an information trail a user leaves behind during her visit to the website. The dataset is semi-structured website log files that contain timestamp, IP address of the visitor (both raw and geocoded), user ID, visited URL and products. We extracted addresses of the visitors containing information about city, state and country, and used this as primary items p. The hierarchy of item p is of the form city$\rightarrow$ state$\rightarrow$ country. We also extracted visited products that contains information about product category such as clothing, handbags, computers, electronics etc, and used this as secondary items s.

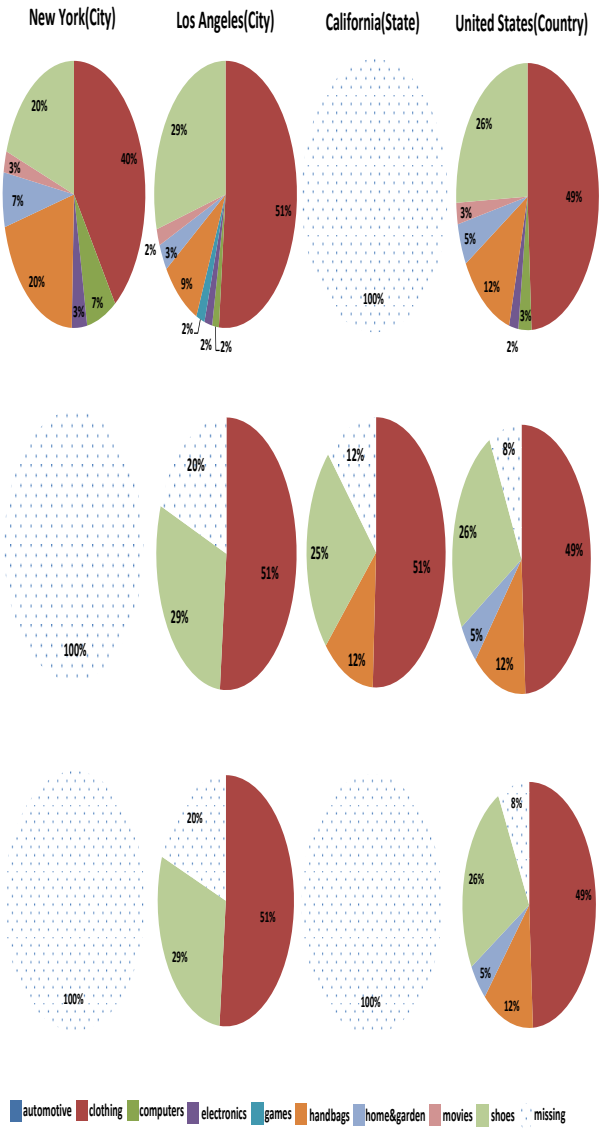
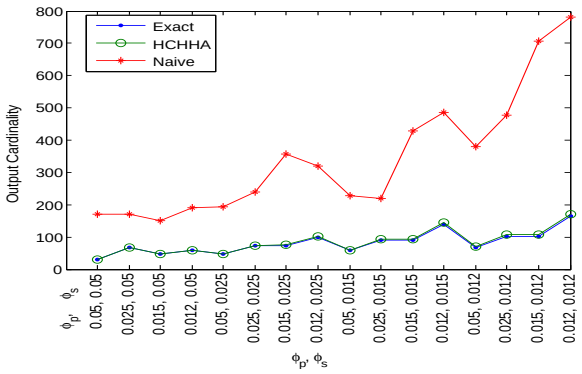
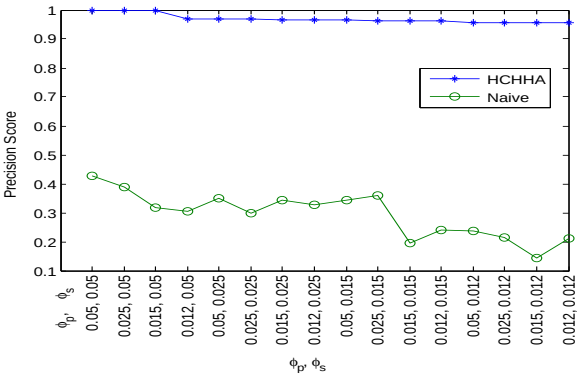


Figure 4.4: Comparisons of HCHH, cross-level association rules (CLAR) and hierarchical heavy hitters (HHH) for $\phi_p = 0.1$, $\phi_s = 0.01$. (Top row) pattern found by HCHHA, (middle row) pattern found by CLAR mining algorithm, and (bottom row) pattern found by HHH algorithm.

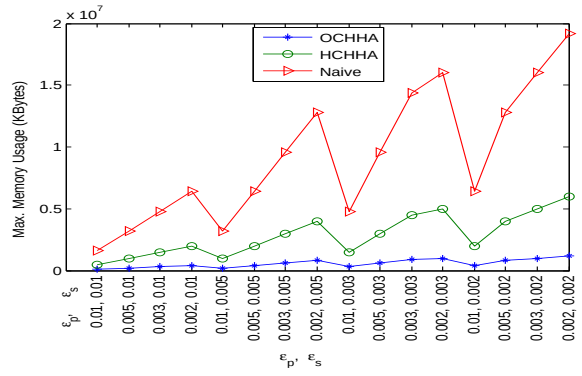
In the first set of experiments, we compare HCHH to cross-level association rules and hierarchical heavy hitters. We found that HCHH make up a distinct set of pairs that cannot be found by other hierarchical notions of interesting elements. The best way to demonstrate this is to use some measures that can find distance between two sets, such as Jaccard distance, which finds the distance between two sets $\mathcal{A}$ and $\mathcal{B}$ to be $1 - \frac{\mathcal{A} \cap \mathcal{B}}{\mathcal{A} \cup \mathcal{B}}$. The distance 0 means that the two sets are exactly the same and the distance 1 means they are totally disjoined.



Comparison in terms of output cardinality
(a)



Comparison in terms of precision
(b)



Comparison in terms of max memory usage
(c)

Figure 4.5: Evaluation of HCHHA and Naïve algorithm for the following parameters settings; $\epsilon_p = \frac{\phi_p}{5}$, $\epsilon_s = \frac{\phi_s}{5}$.

In Fig 4.3, HCHH, cross-level association rules and hierarchical heavy hitters sets are found as follows: various values of ϕ_p and ϕ_s are used as thresholds for HCHH sets; for cross-level association rules ϕ_p is used as support and ϕ_s is used as confidence; for hierarchical heavy hitters ϕ_p is used as threshold. From Fig 4.3, we can see that the set of HCHH is very dissimilar to both the sets produced by cross-level association rules and hierarchical heavy hitters approaches. In our experiment we found that the results produced by cross-level association rules and hierarchical heavy hitters are on average 76% dissimilar, however, the results produced by HCHH are on average 92% dissimilar to both cross-level association rules and hierarchical heavy hitters; demonstrating that these three definitions are strictly describing very distinct phenomena. This indicates that the notion of the proposed HCHH is different from existing hierarchical notions, i.e., hierarchical heavy hitters and cross-level association rules.

Semantic analysis of HCHH: In this set of experiments, we demonstrate the utility of HCHH notion through interpretation of the results where we know the semantics of the data. Semantic analysis on the output generated by HCHH, cross-level association rules and hierarchical heavy hitters demonstrate that HCHH offers additional value and insights into certain type of data relationships.

Figure 4.4 shows interesting statistics of the users clicks of an e-commerce website; it groups the users according to their geographic locations at various concept levels and shows the percentage of visitors to the website that are interested in different products within that group. Particularly, it highlights the locations (the heading of pie charts) the majority of the visitors are coming from; and for each of those major areas a numerical proportion of visitors that are interested in different products. As an example, consider the top left pie chart, it shows that among all the visitors from New York city, 40% visitors are interested in clothes, 20% in shoes, 3% in movies, 7% in home and gardens, 20% in handbags, 3% in electronics and 7% in computers. Note that these percentages are not relative to the total visitors of the website but are relative to the total visitors from New York city only. Other pie charts have the same interpretations. In Figure 4.4, the top, middle and bottom rows are the small subsets of the results produced by HCHH, cross-level association rules and hierarchical heavy hitters approaches respectively, where a dotted pie represents the patterns missed by the respective approaches.

In order to find the above rules (e.g., (New York, Cloth, 40%) or (New York, shoes, 20%)) we can use cross-level association rules. Using cross-level association rules we can find all frequent 1-itemsets and 2-itemsets (e.g., greater 10% threshold) from the two-dimensional dataset from which we can generate the above rules. However, cross-level association rules does not output any two-itemset that contains (New York, Cloth) or (New York, shoes) since, their frequencies are 3% and 2% in the context of the overall dataset, hence below the desired threshold of 10%. Clearly, cross-level association rules misses these items because there may be a large number of such items involving New York, but each of these items is infrequent. Similarly, hierarchical heavy hitters also suffers from the same problem. On the other hand, HCHH first finds the frequent primary item (e.g., New York) in the context of the dataset, then it finds the frequent secondary item (Cloths, Shoes) in the context of the given primary item (New York), so the derived rule (New York, Cloth) has a 40% confidence in the context of all the other rules involving New York as a primary item.

We can see that HCHH provides useful insight about the major locations and interests of users from those locations, however, majority of the interesting patterns are missed by both cross-level association rules and hierarchical heavy hitters (see middle and bottom rows). As an example, for New York city, there was no rules from either cross-level association rules or hierarchical heavy hitters. The reasons is that both these notions treat the two elements of the pair equally (i.e., they do not split the pair into separate items), hence, looking for the pairs such as (New York, shoes) or (New York, clothing) to be higher than the threshold– a pair whose count is more than 10% of all the visitors– which is clearly very inadequate to find interesting local patterns within in the context of some global popular items (e.g., finding the patterns of the visitors' interests within certain locations rather than finding the visitors' interests globally). In contrast, the HCHH notion takes into account the linear relationship of the items within the pair, thus, looking for the first item from the pairs (New York, shoes) or (New York, clothing) to be higher than the threshold 10% of all the visitors, and finds the second items from the pairs that are popular with respect to the first item, not with respect to all the visitors (e.g., globally). Consequently, the results from these approaches are very different, which can be observed from Figure 4.4; each pie chart represents a significantly interesting patterns from the data but many of these patterns are missing by other two approaches. Finally, we can see a missing pattern of California for both HCHH and hierarchical heavy hitters, the reason is that both HCHH and hierarchical heavy hitters use a discounting strategy; a pattern at higher level in the hierarchy is considered significant only if it is within the given thresholds after deducting the count of its descendant patterns that are also significant. Since both of these approaches already output the descendant pattern (i.e., Los Angeles), so both these approaches do not output the ancestor pattern as it provides little further information. This strategy is not used in cross-level association rules mining, hence it outputs many redundant ancestor patterns. Note that there are other approaches [16, 22, 21] proposed in literature to avoid such redundancy in cross-level association rules mining, but they were not considered here.

Thus, we conclude that existing hierarchical notions cannot capture the semantics of hierarchical correlation between items.

Comparisons of HCHHA and Naïve algorithm

This section presents experimental results of HCHH algorithms and comparison with naïve algorithm (which generates similar output to HCHH) in terms of output quality and memory usage. These results are computed using real Internet traffic traces dataset described in Section 4.1.

Output Cardinality: First we compare the output sizes of HCHHA and naïve algorithm using the exact output size as a frame of reference. Figure 4.5a provides the results of HCHHA and naïve algorithm for various thresholds ϕ_p and ϕ_s. The naive algorithm gives the largest output size and the difference in the output sizes of the algorithms increases with lower thresholds. The output sizes from HCHHA are close to exact output sizes and are better than the naïve algorithm. Generally, the naïve algorithm produces outputs that are an order of magnitude larger than those of HCHHA for all the thresholds; the quality of output gradually decreases with decrease in thresholds.

Output Quality: We have also studied the quality of output from HCHHA and naïve algorithm keeping exact answer as a frame of reference. Figure 4.5b plots the precision score against a range of ϕ_p and ϕ_s values. The precision score clearly demonstrates that the output of HCHHA is of high quality even with gradual decrease in thresholds.

Memory Usage: Figure 4.5c compares the efficiency of HCHHA with naïve algorithm in terms of memory requirements. It clearly demonstrates that HCHHA needs less memory than naïve algorithm for various parameter settings. In general, for lower values of ϵ_p and ϵ_s, the memory requirements of the approximate algorithms increase, however, this increase is worse for naïve algorithm compared to HCHHA. The Figure 4.5c also provides memory requirements of OCHHA as a frame of reference, which considers no hierarchy in the data. We can see that the HCHHA has a significant memory advantage over the other hierarchical naïve approach, and require only a little higher memory than the OCHHA, even with complex internal design to accommodate the hierarchical nature of the data. Thus, We emphasize that the naïve strategy can be improved substantially in practice in terms of the space usage and answer quality by designing the hierarchy-aware algorithms like HCHHA.

5 Related Work

The concept of heavy hitters has been extensively studied in the data mining and database literature, which can be tracked back to the early eighties [29, 6], where simple algorithms were developed that are based on tracking items and their counts. The concept has been extended to develop algorithms for finding heavy hitters over a stream of data; these algorithms provide approximate answers (heavy hitters) that provide either deterministic or probabilistic guarantees in answer qualities. For example, [25, 24, 29, 3] are deterministic algorithms those provide guarantees in terms of user defined parameter ϵ; it maintains all items whose frequency is more than ϵN, where N is the number of items in the stream so far. Majority of these stream processing algorithms require $O(\frac{1}{\epsilon})$ memory, which is the minimum memory required for tracking every item with frequencies higher than or equal to ϵN. Similarly, [8, 1, 11, 9] are randomized algorithms that provide probabilistic guarantees in answer qualities.

Several studies on computing heavy hitters have evolved from simple tracking of items and their frequencies to more complex notions; such as hierarchical heavy hitters (when items belongs to a concept hierarchy. The idea behind computing heavy hierarchical hitters is to find nodes in the hierarchy that are heavy hitters by aggregating frequency of their descendants) [10, 31]; computing correlated aggregates, e.g., *min, max* and *average* [14, 32]; time-decayed correlated aggregates [12]; correlated sum and count [2], correlated and conditional heavy hitters [18, 19, 27, 28].

Previous studies on correlated aggregates considered queries in a different form than correlated heavy hitters queries. For instance, the correlated aggregate query [19] first applies a selection predicate along primary dimension p such as $p \geq c$ or $p < c$, where c is provided at query time. Next, the query needs to perform selection on the secondary dimension s. The difference between previous definition and the new correlated heavy hitters concept is that, in correlated heavy hitters the selection predicate along primary dimension involves frequen-

cies f_p and extracts heavy hitters, rather than a simple comparison, i.e., $p \geq c$ or $p < c$. Gehrke et al [14] proposed technique based on adaptive histograms to find correlated aggregates, such that the aggregate along the primary dimension is min, max or average, and the aggregate along the secondary dimension is sum or count. The drawback of this technique is that it does not provide any provable guarantees. Ananthakrishna et al [2] has proposed algorithm based on quantile summary to find correlated sum or counts with provable error bounds, however, the approach is not suitable for finding correlated heavy hitters.

Another interesting idea has been considered by Cormode et al [12]; they focused on time-decayed correlated aggregates, where items in the stream are weighted based on arrival time of the items. Although their work considered *sum* aggregates, it cannot be directly applied to find heavy hitters. In this paper, we explore limitations of existing algorithms of correlated heavy hitters over a stream of two dimensional data, and provide algorithms for finding correlated heavy hitters in data streams.

Although there exists a correlated heavy hitters algorithm to compute flat HHH [19, 18] we have discovered several significant problems that prohibits its use for developing HCHH algorithm. First, the existing correlated heavy hitters algorithm may suffer from high false positive rate due to lack of threshold adjustments, which is important when constraints on memory usage of the algorithm are optimized. Second, the memory requirements of the existing correlated heavy hitters algorithm is $O(\frac{1}{\epsilon_p \epsilon_s^2})$, which is very high for many streaming applications. Third, the existing correlated heavy hitters algorithm needs the values of ϕ_p and ϕ_s apriori, which we believe is very restrictive for a variety of real life applications that require analysis of the data streams for various thresholds. Finally, the existing correlated heavy hitters algorithm cannot be extended to design an algorithm for hierarchical datasets (e.g., when p, and s are hierarchical items), because our definition of approximate HCHH requires a bottom-up approach, which means that infrequent items deleted (due to space constraint of stream) from lowest concept level needs to be combined at higher concept level. Therefore, at the time of deletion the count of all the deleted items for the period since they were inserted needs to be known, which is used to aggregate them at higher concept level for measuring the correlation there. However, the algorithm proposed in [19, 18] does not provide the count of deleted items; deleted items always have zero count.

Apart from the above, there are some techniques that summarize hierarchical data, such as hierarchical heavy hitters [31, 10], multilevel and cross-level association rules [16]. However, these notions treat the two elements of the pair equally–not taking into account the sequential nature of items' relationship within the pairs–thus cannot discover the hierarchical correlation semantics between items at multiple concept levels. Therefore, in this work, we introduce the concept of HCHH that captures the correlation between pairs of items at multiple concept levels (see Section 2.4 for formal definition of HCHH).

6 Conclusion

Finding HCHH are useful for many online applications that require to capture the semantics in the form of hierarchically correlated pairs. It can be useful in network monitoring, anomaly detection and data analysis for business intelli-

gence and planing. This paper described a new notion of hierarchically correlated heavy hitters and proposed proposed algorithms to solve HCHH problem in two dimensional streams, where items have been drawn from different hierarchies. The proposed algorithms are based on approximation with deterministic provable error guarantees. Experimental results on real benchmark datasets have confirmed the utility of the proposed algorithms. In future, we aim to extend our algorithms to the sliding window computational model and compare with other hierarchical notions.

Bibliography

[1] Noga Alon, Yossi Matias, and Mario Szegedy. The space complexity of approximating the frequency moments. In *Proc. of Theory of computing*, pages 20–29, 1996.

[2] Rohit Ananthakrishna, Abhinandan Das, Johannes Gehrke, Flip Korn, S Muthukrishnan, and Divesh Srivastava. Efficient approximation of correlated sums on data streams. *TKDE*, 15(3):569–572, 2003.

[3] Arvind Arasu and Gurmeet Singh Manku. Approximate counts and quantiles over sliding windows. In *Proc. of Principles of DB systems*, pages 286–296. ACM, 2004.

[4] Elena Baralis, Luca Cagliero, Tania Cerquitelli, Silvia Chiusano, and Paolo Garza. Digging deep into weighted patient data through multiple-level patterns. *Information Sciences*, 322:51–71, 2015.

[5] Radu Berinde, Piotr Indyk, Graham Cormode, and Martin J Strauss. Space-optimal heavy hitters with strong error bounds. *(TODS)*, 35(4):26, 2010.

[6] RS Boyer and JS Moore. *A fast majority vote algorithm*. SRI International. Computer Science Laboratory, 1981.

[7] Christian Callegari, Stefano Giordano, Michele Pagano, and Teresa Pepe. Detecting anomalies in backbone network traffic: a performance comparison among several change detection methods. *Int. Jour of Sensor Networks*, 11(4):205–214, 2012.

[8] Moses Charikar, Kevin Chen, and Martin Farach-Colton. Finding frequent items in data streams. In *Automata, Languages and Programming*, pages 693–703. Springer, 2002.

[9] Graham Cormode and Minos Garofalakis. Sketching streams through the net: Distributed approximate query tracking. In *Proc. of VLDB*, pages 13–24, 2005.

[10] Graham Cormode, Flip Korn, S Muthukrishnan, and Divesh Srivastava. Finding hierarchical heavy hitters in streaming data. *(TKDD)*, 1(4):2, 2008.

[11] Graham Cormode and S Muthukrishnan. An improved data stream summary: the count-min sketch and its applications. *Journal of Algorithms*, 55(1):58–75, 2005.

[12] Graham Cormode, Srikanta Tirthapura, and Bojian Xu. Time-decayed correlated aggregates over data streams. *Statistical Analysis and Data Mining: The ASA Data Science Journal*, 2(5-6):294–310, 2009.

[13] Cristian Estan, Stefan Savage, and George Varghese. Automatically inferring patterns of resource consumption in network traffic. In *Pro of Appl, tech, arch, and prot for comp com*, pages 137–148. ACM, 2003.

[14] Johannes Gehrke, Flip Korn, and Divesh Srivastava. On computing correlated aggregates over continual data streams. In *ACM SIGMOD Record*, volume 30, pages 13–24. ACM, 2001.

[15] Jiawei Han and Yongjian Fu. Discovery of multiple-level association rules from large databases. In *VLDB*, volume 95, pages 420–431, 1995.

[16] Jiawei Han and Yongjian Fu. Mining multiple-level association rules in large databases. *TKDE*, 11(5):798–805, 1999.

[17] Demetris Hoplaros, Zahir Tari, and Ibrahim Khalil. Data summarization for network traffic monitoring. *JNCA*, 37:194–205, 2014.

[18] Bibudh Lahiri, ArkoProvo Mukherjee, and Srikanta Tirthapura. Identifying correlated heavy-hitters in a two-dimensional data stream. *Data Mining and Knowledge Discovery*, pages 1–22, 2015.

[19] Bibudh Lahiri and Srikanta Tirthapura. Finding correlated heavy-hitters over data streams. In *(IPCCC)*, pages 307–314. IEEE, 2009.

[20] Lap-Kei Lee and HF Ting. A simpler and more efficient deterministic scheme for finding frequent items over sliding windows. In *Proc. of Principles of DB systems*, pages 290–297. ACM, 2006.

[21] Cane Wing-ki Leung, Stephen Chi-fai Chan, and Fu-lai Chung. An empirical study of a cross-level association rule mining approach to cold-start recommendations. *Knowledge-Based Systems*, 21(7):515–529, 2008.

[22] Weiyang Lin, Sergio A Alvarez, and Carolina Ruiz. Efficient adaptive-support association rule mining for recommender systems. *Data mining and knowledge discovery*, 6(1):83–105, 2002.

[23] Elzbieta Malinowski and Esteban Zimányi. Olap hierarchies: A conceptual perspective. In *Advanced Information Systems Engineering*, pages 477–491. Springer, 2004.

[24] Gurmeet Singh Manku and Rajeev Motwani. Approximate frequency counts over data streams. In *Proc. VLDB*, pages 346–357, 2002.

[25] Ahmed Metwally, Divyakant Agrawal, and Amr El Abbadi. Efficient computation of frequent and top-k elements in data streams. In *ICDT*, pages 398–412. Springer, 2005.

[26] Jörg Micheel, Ian Graham, and Nevil Brownlee. The auckland data set: an access link observed. In *Proc. of access networks and systems*, pages 19–30, 2001.

[27] Katsiaryna Mirylenka, Graham Cormode, Themis Palpanas, and Divesh Srivastava. Conditional heavy hitters: detecting interesting correlations in data streams. *VLDB Journal*, 24(3):395–414, 2015.

[28] Katsiaryna Mirylenka, Themis Palpanas, Graham Cormode, and Divesh Srivastava. Finding interesting correlations with conditional heavy hitters. In *(ICDE)*, pages 1069–1080. IEEE, 2013.

[29] Jayadev Misra and David Gries. Finding repeated elements. *Science of computer programming*, 2(2):143–152, 1982.

[30] S Muthukrishnan. *Data streams: Algorithms and applications*. Now Publishers Inc, 2005.

[31] Zubair Shah, Abdun Naser Mahmood, and Michael Barlow. Computing discounted multidimensional hierarchical aggregates using modified misra gries algorithm. *Performance Evaluation*, 91:170–186, 2015.

[32] Srikanta Tirthapura and David P Woodruff. A general method for estimating correlated aggregates over a data stream. In *(ICDE)*, pages 162–173. IEEE, 2012.