Data Min Knowl Disc (2016) 30:797–818
DOI 10.1007/s10618-015-0438-6

Identifying correlated heavy-hitters in a two-dimensional data stream

Bibudh Lahiri[1] · **Arko Provo Mukherjee**[2] · **Srikanta Tirthapura**[2]

Received: 2 October 2013 / Accepted: 15 September 2015 / Published online: 24 October 2015

Abstract We consider online mining of correlated heavy-hitters (CHH) from a data stream. Given a stream of two-dimensional data, a correlated aggregate query first extracts a substream by applying a predicate along a primary dimension, and then computes an aggregate along a secondary dimension. Prior work on identifying heavy-hitters in streams has almost exclusively focused on identifying heavy-hitters on a single dimensional stream, and these yield little insight into the properties of heavy-hitters along other dimensions. In typical applications however, an analyst is interested not only in identifying heavy-hitters, but also in understanding further properties such as: what other items appear frequently along with a heavy-hitter, or what is the frequency distribution of items that appear along with the heavy-hitters. We consider queries of the following form: "In a stream S of (x, y) tuples, on the substream H of all x values that are heavy-hitters, maintain those y values that occur frequently with the x values in H". We call this problem as CHH. We formulate an approximate

Responsible editor: Bart Goethals.

Electronic supplementary material The online version of this article (doi:10.1007/s10618-015-0438-6) contains supplementary material, which is available to authorized users.

✉ Srikanta Tirthapura
snt@iastate.edu

Bibudh Lahiri
bibudhlahiri@gmail.com

Arko Provo Mukherjee
arko@iastate.edu

[1] Impetus Technologies, Los Gatos, CA 95032, USA

[2] Department of Electrical and Computer Engineering, Iowa State University, Ames, IA 50011, USA

formulation of CHH identification, and present an algorithm for tracking CHHs on a data stream. The algorithm is easy to implement and uses workspace much smaller than the stream itself. We present provable guarantees on the maximum error, as well as detailed experimental results that demonstrate the space-accuracy trade-off.

Keywords Data stream mining · Correlation · Heavy-hitters

1 Introduction

Correlated aggregates (Ananthakrishna et al. 2003; Gehrke et al. 2001; Cormode et al. 2009) reveal interesting interactions among different attributes of a multi-dimensional dataset. They are useful in finding an aggregate on an attribute over a subset of the data, where the subset is defined by a selection predicate on a different attribute of the data. On stored data, a correlated aggregate can be computed by considering one dimension at a time, using multiple passes through the data. However, for dynamic streaming data, we often do not have the luxury of making multiple passes over the data, and moreover, the data may be too large to store and it is desirable to have an algorithm that works in a single pass through the data. Sometimes, even the substream derived by applying the query predicate along the primary dimension can be too large to store, let alone the whole dataset.

We first define the notion of a heavy-hitter on a data stream (this is considered in prior work, such as Manku and Motwani 2002; Misra and Gries 1982; Charikar et al. 2004; Cormode and Muthukrishnan 2005), and then define our notion of correlated heavy-hitters (CHH). Given a sequence of single-dimensional records $(a_1, a_2, \ldots, a_N)$, where $a_i \in \{1, \ldots, m\}$, the frequency of an item i is defined as $|\{a_j | a_j = i\}|$. Given a user-input threshold $\phi \in (0, 1)$, any data item i whose frequency is at least ϕN is termed as a ϕ-heavy-hitter. We first consider the following problem of exact identification of CHHs.

Problem 1 Exact Identification of Correlated Heavy Hitters Given a data stream S of (x, y) tuples of length N (x and y will henceforth be referred to as the "primary" and the "secondary" dimensions, respectively), and two user-defined thresholds ϕ_1 and ϕ_2, where $0 < \phi_1 < 1$ and $0 < \phi_2 < 1$, identify all (d, s) tuples such that:

$$f_d = |\{(x, y) \in S : (x = d)\}| > \phi_1 N$$

and

$$f_{d,s} = |\{(x, y) \in S : (x = d) \wedge (y = s)\}| > \phi_2 f_d$$

The above aggregate can be understood as follows. The elements d are heavy-hitters in the traditional sense, on the stream formed by projecting along the primary dimension. For each heavy-hitter d along the primary dimension, there is logically a (uni-dimensional) substream S_d, consisting of all values along the secondary dimension, where the primary dimension equals d. We require the tracking of all tuples (d, s) such that s is a heavy-hitter in S_d.

Many stream mining and monitoring problems on two-dimensional streams need the CHH aggregate, and cannot be answered by independent aggregation along single dimensions. For example, consider a network monitoring application, where a stream of (destination IP address, source IP address) pairs is being observed. The network monitor maybe interested not only in tracking those destination IP addresses that receive a large fraction of traffic (heavy-hitter destinations), but also in tracking those source IP addresses that send a large volume of traffic to these heavy-hitter destinations. This cannot be done by independently tracking heavy-hitters along the primary and the secondary dimensions. Note that in this application, we are interested not only in the identity of the heavy-hitters, but also additional information on the substream induced by the heavy-hitters.

In another example, in a stream of (server IP address, port number) tuples, identifying the heavy-hitter server IP addresses will tell us which servers are popular, and identifying frequent port numbers (independently) will tell us which applications are popular; but a network manager maybe interested in knowing which applications are popular among the heavily loaded servers, which can be retrieved using a CHH query. Such correlation queries are used for network optimization and anomaly detection (Cullingford 2009).

Another application is the recommendation system of a typical online shopping site, which shows a buyer a list of the items frequently bought with the ones she has decided to buy. Our algorithm can optimize the performance of such a system by parsing the transaction logs and identifying the items that were bought commonly with the frequently purchased items. If such information is stored in a cache with a small lookup time, then for most buyers, the recommendation system can save the time to perform a query on the disk-resident data.

Similar to the above examples, in many stream monitoring applications, it is important to track the heavy-hitters in the stream, but this monitoring should go beyond simple identification of heavy-hitters, or tracking their frequencies, as is considered in most prior formulations of heavy-hitter tracking such as Cormode and Muthukrishnan (2003), Manku and Motwani (2002), Misra and Gries (1982), Charikar et al. (2004), Estan and Varghese (2002). In this work we initiate the study of tracking additional properties of heavy-hitters by considering tracking of correlated heavy hitters.

1.1 Approximate CHH

It is easy to prove that exact identification of heavy-hitters in a single dimension is impossible using limited space, and one pass through the input. Hence, the CHH problem is also impossible to solve in limited space, using a single pass through the input. Due to this, we consider the following approximate version of the problem. We introduce additional approximation parameters, ϵ_1 and $\epsilon_2 (0 < \epsilon_1 \leq \frac{\phi_1}{2}, 0 < \epsilon_2 < \phi_2)$, which stand for the approximation errors along the primary and the secondary dimensions, respectively. We seek an algorithm that provides the following guarantees.

Problem 2 Approximate Identification of Correlated Heavy-Hitters Given a data stream S of (d, s) tuples of length N, thresholds ϕ_1 and ϕ_2:

1. Report any value d such that $f_d > \phi_1 N$ as a heavy-hitter along the primary dimension.
2. No value d such that $f_d < (\phi_1 - \epsilon_1)N$, should be reported as a heavy-hitter along the primary dimension.
3. For any value d reported above, report any value s along the secondary dimension such that $f_{d,s} > \phi_2 f_d$ as a CHH.
4. For any value d reported above, no value s along the secondary dimension such that $f_{d,s} < (\phi_2 - \epsilon_2) f_d$ should be reported as a CHH occurring along with d.

With this problem formulation, false positives are possible, but false negatives are not. In other words, if a pair (d, s) is a CHH according to the definition in Problem 1, then it is a CHH according to the definition in Problem 2, and will be returned by the algorithm. But an algorithm for Problem 2 may return a pair (d, s) that are not exact CHHs, but whose frequencies are close to the required thresholds.

1.2 Contributions

Our contributions are as follows.

- We formulate exact and approximate versions of the problem of identifying CHHs in a multidimensional data stream, and present a small-space approximation algorithm for identifying approximate CHHs in a single pass. Prior literature on correlated aggregates have mostly focused on the correlated sum, and these techniques are not applicable for CHH. Our algorithm for approximate CHH identification is based on a nested application of the Misra–Gries algorithm (Misra and Gries 1982).
- We provide a provable guarantee on the approximation error. We show that there are no false negatives, and the error in the false positives is controlled. When greater memory is available, this error can be reduced. The space taken by the algorithm as well as the approximation error of the algorithm depend on the sizes of two different data structures within the algorithm. The total space taken by the sketch is minimized through solving a constrained optimization problem that minimizes the total space taken subject to providing the user-desired error guarantees.
  Specifically, Let $\alpha = \left(\frac{1+\phi_2}{\phi_1 - \epsilon_1}\right)$.
  - If $\epsilon_1 \geq \frac{\epsilon_2}{2\alpha}$, then $s_1 = \frac{2\alpha}{\epsilon}$ and $s_2 = \frac{2}{\epsilon_2}$. In this case, the space complexity is $O\left(\frac{1}{(\phi_1 - \epsilon_1)\epsilon_2^2}\right)$.
  - If $\epsilon_1 < \frac{\epsilon_2}{2\alpha}$, then $s_1 = \frac{1}{\epsilon_1}$, and $s_2 = \frac{1}{\epsilon_2 - \alpha\epsilon_1}$. In this case, the space complexity is $O(\frac{1}{\epsilon_1\epsilon_2})$.
- We present results from our simulations on (a) a stream of more than 1.4 billion (50 GB trace) anonymized packet headers from an OC48 link (collected by CAIDA https://data.caida.org/datasets/oc48/oc48-original/20020814/5min/), and (b) a sample of 240 million 2-g extracted from English fiction books (http://storage.googleapis.com/books/ngrams/books/datasetsv2.html). We compared the performance of our small-space algorithm with a slow, but exact algorithm that

goes through the input data in multiple passes. Our experiments revealed that even with a space budget of a few megabytes, the average error of our algorithm was very small, showing that it is viable in practice.

Along each dimension our algorithm maintains frequency estimates of mostly those values (or pairs of values) that occur frequently. For example, in a stream of (destination IP, source IP) tuples, for every destination that sends a significant fraction of traffic on a link, we maintain mostly the sources that occur frequently along with this destination. Note that the set of heavy-hitters along the primary dimension can change as the stream elements arrive, and this influences the set of CHHs along the secondary dimension. For example, if an erstwhile heavy-hitter destination d no longer qualifies as a heavy-hitter with increase in N (and hence gets rejected from the sketch), then a source s occurring with d should also be discarded from the sketch. This interplay between different dimensions has to be handled carefully during algorithm design.

Roadmap The rest of this paper is organized as follows. We present related work in Sect. 2. In Sect. 3.1 we present the algorithm description, followed by the proof of correctness in Sect. 3.2, and the analysis of the space complexity in Sect. 3.4. We present experimental results in Sect. 4.

2 Related work

In the data streaming literature, there is a significant body of work on correlated aggregates (Ananthakrishna et al. 2003; Gehrke et al. 2001; Cormode et al. 2009), as well as on the identification of heavy hitters (Manku and Motwani 2002; Misra and Gries 1982; Charikar et al. 2004; Cormode and Muthukrishnan 2005). See Cormode and Hadjieleftheriou (2009) for a recent overview of work on heavy-hitter identification. None of these works consider correlated heavy-hitters.

Estan et al. (2003) and Zhang et al. (2004) have independently studied the problem of identifying heavy-hitters from multi-dimensional packet streams, but they both define a multidimensional tuple as a heavy-hitter if it occurs more than ϕN times in the stream, N being the stream size—the interplay across different dimensions is not considered.

There is significant prior work on correlated aggregate computation that we now describe. The problems considered in the literature usually take the following form. On a stream of two dimensional data items (x, y) the query asks to first apply a selection predicate along the x dimension, of the form $x \geq c$ or $x < c$ (for a value c provided at query time), followed by an aggregation along the y dimension. The difference when compared with this formulation is that in our case, the selection predicate along the x dimension is one that involves frequencies and heavy-hitters, rather than a simple comparison.

Gehrke et al. (2001) addressed correlated aggregates where the aggregate along the primary dimension was an extremum (min or max) or the average, and the aggregate along the secondary dimension was sum or count. For example, given a stream S of (x, y) tuples, their algorithm could approximately answer queries of the following form: "Return the sum of y-values from S where the corresponding x values are

greater than a threshold α." They describe a data structure called *adaptive histograms*, but these did not come with provable guarantees on performance. Ananthakrishna et al. (2003) presented algorithms with provable error bounds for correlated sum and count. Their solution was based on the quantile summary of Greenwald and Khanna (2001). With this technique, exact heavy-hitter queries cannot be used as the aggregate along the primary dimension since they cannot be computed on a stream using limited space. Cormode et al. (2009) presented algorithms for maintaining the more general case of *time-decayed* correlated aggregates, where the stream elements were weighted based on the time of arrival. This work also addressed the "sum" aggregate, and the methods are not directly applicable to heavy-hitters. Other work in this direction includes (Busch and Tirthapura 2007; Xu et al. 2008). Tirthapura and Woodruff (2012) present a general method that reduces the correlated estimation of an aggregate to the streaming computation of the aggregate, for functions that admit sketches of a particular structure. These techniques only apply to selection predicates of the form $x > c$ or $x < c$, and do not apply to heavy-hitters, as we consider here.

The heavy-hitters literature has usually focused on the following problem. Given a sequence of elements $A = (a_1, a_2, \ldots, a_N)$ and a user-input threshold $\phi \in (0, 1)$, find data items that occur more than ϕN times in A. Misra and Gries (1982) presented a deterministic algorithm for this problem, with space complexity being $O(\frac{1}{\phi})$, time complexity for updating the sketch with the arrival of each element being $O(\log \frac{1}{\phi})$, and query time complexity being $O(\frac{1}{\phi})$. For exact identification of heavy-hitters, their algorithm works in two passes. For approximate heavy-hitters, their algorithm uses only one pass through the sequence, and has the following approximation guarantee. Assume user-input threshold ϕ and approximation error $\epsilon < \phi$. Note that for an online algorithm, N is the number of elements received so far.

- All items whose frequencies exceed ϕN are output. i.e. there are no false negatives.
- No item with frequency less than $(\phi - \epsilon)N$ is output.

Demaine et al. (2002a) and Karp et al. (2003) improved the sketch update time per element of the Misra–Gries algorithm from $O(\log \frac{1}{\phi})$ to $O(1)$, using an advanced data structure combining a hashtable, a linked list and a set of doubly-linked lists. Manku and Motwani (2002) presented a deterministic "Lossy Counting" algorithm that offered the same approximation guarantees as the one-pass approximate Misra–Gries algorithm; but their algorithm required $O(\frac{1}{\epsilon} \log (\epsilon N))$ space in the worst case. For our problem, we chose to extend the Misra–Gries algorithm as it takes asymptotically less space than (Manku and Motwani 2002).

3 Algorithm and analysis

3.1 Intuition and algorithm description

Our algorithm is based on a nested application of an algorithm for identifying frequent items from an one-dimensional stream, due to Misra and Gries (1982). We first describe the Misra–Gries algorithm (henceforth called the MG algorithm). Suppose we are given an input stream $a_1, a_2, \ldots$, and an error threshold ϵ, $0 < \epsilon < 1$. The algorithm

maintains a data structure $\mathcal{D}$ that contains at most $\frac{1}{\epsilon}$ (key, count) pairs. On receiving an item a_i, it is first checked if a tuple $(a_i, \cdot)$ already exists in $\mathcal{D}$. If it does, a_i's count is incremented by 1; otherwise, the pair $(a_i, 1)$ is added to $\mathcal{D}$. Now, if adding a new pair to $\mathcal{D}$ makes $|\mathcal{D}|$ exceed $\frac{1}{\epsilon}$, then for each (key, count) pair in $\mathcal{D}$, the count is decremented by one; and any key whose count falls to zero is discarded. This ensures at least the key which was most recently added (with a count of one) would get discarded, so the size of $\mathcal{D}$, after processing all pairs, would come down to $\frac{1}{\epsilon}$ or less. Thus, the space requirement of this algorithm is $O(\frac{1}{\epsilon})$. The data structure $\mathcal{D}$ can be implemented using hashtables or height-balanced binary search trees. At the end of one pass through the data, the MG algorithm maintains the frequencies of keys in the stream with an error of no more than ϵn, where n is the size of the stream. The MG algorithm can be used in exact identification of heavy hitters from a data stream using two passes through the data.

In the scenario of limited memory, the MG algorithm can be used to solve Problem 1 in three passes through the data, as follows. We first describe a four pass algorithm. In the first two passes, heavy-hitters along the primary dimension are identified, using memory $O(1/\phi_1)$. Note that this is asymptotically the minimum possible memory requirement of any algorithm for identifying heavy-hitters, since the size of output can be $\Omega\left(\frac{1}{\phi_1}\right)$. In the next two passes, heavy-hitters along the secondary dimension are identified for each heavy-hitter along the primary dimension. This takes space $O\left(\frac{1}{\phi_2}\right)$ for each heavy-hitter along the primary dimension. The total space cost is $O\left(\frac{1}{\phi_1\phi_2}\right)$, which is optimal, since the output could be $\Omega\left(\frac{1}{\phi_1\phi_2}\right)$ elements. The above algorithm can be converted into a *three* pass exact algorithm by combining the second and third passes.

Next let us consider Problem 2. Note that the MG algorithm cannot be used to solve this problem in one pass due to the following reason. Let us consider running the MG Algorithm using $\epsilon = \phi_1\phi_2$. Let us also consider an element (x, y) such that the frequency of the element (x, y) is greater than ϵ. However, this doesn't guarantee that the frequency of x in first dimension is greater than ϕ_1 or the frequency of y in sub-stream of x is greater than ϕ_2. If such is the case, then this element will be reported by the MG Algorithm, leading to a false positive. To validate this, we generated a synthetic dataset containing some elements that have relative frequency in first dimension less than ϕ_1 but have overall frequency higher than ϵ. Next we implemented the MG algorithm and ran it using the synthetic dataset, with $\epsilon = \phi_1\phi_2$. As predicted, the elements having frequency less than ϕ_1 in first dimension but overall frequency greater than ϵ got reported. To overcome this limitation, we designed a novel single-pass algorithm for Problem 2.

The high-level idea behind our single-pass algorithm for Problem 2 is as follows. The MG algorithm for an one-dimensional stream, can be viewed as maintaining a small space "sketch" of data that (approximately) maintains the frequencies of each distinct item d along the primary dimension; of course, these frequency estimates are useful only for items that have very high frequencies. For each distinct item d along the primary dimension, apart from maintaining its frequency estimate $\hat{f}_d$, our algorithm maintains an embedded MG sketch of the substream S_d induced by d, i.e.

$S_d = \{(x, y)|((x, y) \in S) \wedge (x = d)\}$. The embedded sketch is a set of tuples of the form $(s, \hat{f}_{d,s})$, where s is an item that occurs in S_d, and $\hat{f}_{d,s}$ is an estimate of the frequency of the pair (d, s) in S (or equivalently, the frequency of s in S_d). While the actions on $\hat{f}_d$ (increment, decrement, discard) depend on how d and the other items appear in S, the actions on $\hat{f}_{d,s}$ depend on the items appearing in S_d. Further, the sizes of the tables that are maintained have an important effect on both the correctness and the space complexity of the algorithm.

We now present a more detailed description. The algorithm maintains a table H, which is a set of tuples $(d, \hat{f}_d, H_d)$, where d is a value along the primary dimension, $\hat{f}_d$ is the estimated frequency of d in the stream, and H_d is another table that stores the values of the secondary attribute that occur with d. H_d stores its content in the form of (key, count) pairs, where the keys are values (s) along the secondary attribute and the counts are the frequencies of s in S_d, denoted as $\hat{f}_{d,s}$, along with d. The tables H and H_d can be implemented using the data structures we describe in detail in Sect. 3.3.

The maximum number of tuples in H is s_1, and the maximum number of tuples in each H_d is s_2. The values of s_1 and s_2 depend on the parameters $\phi_1, \phi_2, \epsilon_1, \epsilon_2$, and are decided at the start of the algorithm. Since s_1 and s_2 effect the space complexity of the algorithm, as well as the correctness guarantees provided by it, their values are set based on an optimization procedure, as described in Sect. 3.4.

The formal description is presented in Algorithms 1, 2 and 3. Before a stream element is received, Algorithm 1 Sketch-Initialize is invoked to initialize the data structures. Algorithm 2 Sketch-Update is invoked to update the data structure as each stream tuple (x, y) arrives. Algorithm 3 Report-CHH is used to answer queries when a user asks for the CHHs in the stream so far.

Algorithm 1: Sketch-Initialize($\phi_1, \phi_2, \epsilon_1, \epsilon_2$)

Input: Threshold for primary dimension ϕ_1; Threshold for secondary dimension ϕ_2; Tolerance for primary dimension ϵ_1; Tolerance for secondary dimension ϵ_2

1 $H \leftarrow \Phi$

2 Set $\frac{1}{s_1} \leq \epsilon_1$;

3 Set $\frac{1}{s_2} + \frac{1+\phi_2}{s_1(\phi_1-\epsilon_1)} \leq \epsilon_2$

On receiving an element (x, y) of the stream, the following three scenarios may arise. We explain the action taken in each.

1. If x is present in H, and y is present in H_x, then both $\hat{f}_x$ and $\hat{f}_{x,y}$ are incremented.
2. If x is present in H, but y is not in H_x, then y is added to H_x with a count of 1. If this addition causes $|H_x|$ to exceed its space budget s_2, then for each (key, count) pair in H_x, the count is decremented by 1 (similar to the MG algorithm). If the count of any key falls to zero, the key is dropped from H_x. Note that after this operation, the size of H_x will be at most s_2.
3. If x is not present in H, then an entry is created for x in H by setting $\hat{f}_x$ to 1, and by initializing H_x with the pair $(y, 1)$. If adding this entry causes $|H|$ to exceed s_1, then for each $d \in H$, f_d is decremented by 1. If the decrement causes $\hat{f}_d$ to be zero, then we simply discard the entry for d from H.

Algorithm 2: Sketch-Update(x, y)

Input: Element along primary dimension x; Element along secondary dimension y

if $x \in H$ **then**
  $\hat{f}_x \leftarrow \hat{f}_x + 1$;
  if $y \in H_x$ **then**
    /* Both x and y are present */
    Increment $\hat{f}_{x,y}$ in H_x by 1;
  else
    /* $x \in H$, but $y \notin H_x$ */
    Add the tuple $(y, 1)$ to H_x;
    if $|H_x| > s_2$ **then**
      foreach $(s, \hat{f}_{d,s}) \in H_x$ **do**
        $\hat{f}_{d,s} \leftarrow \hat{f}_{d,s} - 1$;
        if $\hat{f}_{d,s} = 0$ **then**
          discard $(s, \hat{f}_{d,s})$ from H_x;
else
  /* Neither of x or y is present */
  $H_x \leftarrow \Phi$; Add $(y, 1)$ to H_x; $\hat{f}_x \leftarrow 1$;
  if $|H| > s_1$ **then**
    foreach $d \in H$ **do**
      $\hat{f}_d \leftarrow \hat{f}_d - 1$;
      if *there exists s such that* $\hat{f}_{d,s} > 0$ **then**
        Choose an arbitrary $(s, \hat{f}_{d,s}) \in H_d$ such that $\hat{f}_{d,s} > 0$;
        $\hat{f}_{d,s} \leftarrow \hat{f}_{d,s} - 1$;
        if $\hat{f}_{d,s} = 0$ **then**
          discard $(s, \hat{f}_{d,s})$ from H_d;
      if $\hat{f}_d = 0$ **then**
        Discard (d, H_d) from H;

Algorithm 3: Report-CHH(N)

Input: Size of the stream N

foreach $d \in H$ **do**
  if $\hat{f}_d \geq (\phi_1 - \frac{1}{s_1})N$ **then**
    Report d as a frequent value of the primary attribute;
    foreach $(s, \hat{f}_{d,s}) \in H_d$ **do**
      if $\hat{f}_{d,s} \geq (\phi_2 - \frac{1}{s_2})\hat{f}_d - \frac{N}{s_1}$ **then**
        Report s as a CHH occurring with d;

Otherwise, when f_d is decremented, the algorithm keeps the sum of the $\hat{f_{d,s}}$ counts within H_d equal to f_d; the detailed correctness is proved in Sect. 3.4. To achieve this, an arbitrary key s is selected from H_d such that such that $\hat{f}_{d,s} > 0$, and $\hat{f}_{d,s}$ is decremented by 1. If $\hat{f}_{d,s}$ falls to zero, s is discarded from H_d.

3.2 Algorithm correctness

In this section, we show the correctness of the algorithm, subject to the following constraints on s_1 and s_2. In Sect. 3.4, we assign values to s_1 and s_2 in such a manner that the space taken by the data structure is minimized.

Constraint 1

$$\frac{1}{s_1} \leq \epsilon_1$$

Constraint 2

$$\frac{1}{s_2} + \frac{1+\phi_2}{s_1(\phi_1 - \epsilon_1)} \leq \epsilon_2$$

Consider the state of the data structure after a stream S of length N has been observed. Consider a value d of the primary dimension, and s of the secondary dimension. Let f_d and $f_{d,s}$ be defined as in Sect. 1. Our analysis focuses on the values of variables $\hat{f}_d$ and $\hat{f}_{d,s}$, which are updated in Algorithms 2 and used in Algorithm 3. For convenience, if d is not present in H then we define $\hat{f}_d = 0$. Similarly, if d is not present in H, or if (d, s) is not present in H_d, then we define $\hat{f}_{d,s} = 0$.

Lemma 1

$$\hat{f}_d \geq f_d - \frac{N}{s_1}$$

Proof The total number of increments in the s_1 counters that keep track of the counts of the different values of the primary dimension is N. Each time there is a decrement to $\hat{f}_d$ (in Line 20 of Algorithm 2), $s_1 + 1$ different counters are decremented. The total number of decrements, however, cannot be more than the total number of increments, and hence is at most N. So the number of times the block of lines 19–31 in Algorithm 2 gets executed is at most $\frac{N}{s_1+1} < \frac{N}{s_1}$. We also know that $\hat{f}_d$ is incremented exactly f_d times, hence the final value of $\hat{f}_d$ is greater than $f_d - \frac{N}{s_1}$. Note that this analysis is obtained from the standard analysis for the Misra–Gries frequent items algorithm. □

Lemma 2 *Assume that Constraint 1 is true. If $f_d > \phi_1 N$, then d is reported by Algorithm 3 as a frequent item. Further, if $f_d < (\phi_1 - \epsilon_1)N$, then d is not reported as a frequent item.*

Proof Suppose $f_d \geq \phi_1 N$. From Lemma 1, $\hat{f}_d \geq f_d - \epsilon_1 N \geq \phi_1 N - \epsilon_1 N$. Hence Algorithm 3 will report d (see Lines 2 and 3). Next, suppose that $f_d < (\phi_1 - \epsilon_1)N$. Since $\hat{f}_d \leq f_d$, Algorithm 3 will not report d as a frequent item. □

Lemma 3

$$\sum_{(s,\cdot)\in H_d} \hat{f}_{d,s} \leq \hat{f}_d$$

Proof Let $\Sigma_d = \sum_{(s,\cdot)\in H_d} \hat{f}_{d,s}$. Let $C(n)$ denote the condition $\Sigma_d \leq \hat{f}_d$ after n stream elements have been observed. We prove $C(n)$ by induction on n. The base case is when $n = 0$, and in this case, $\hat{f}_{d,s} = \hat{f}_d = 0$ for all d, s, and $C(0)$ is trivially true. For the inductive step, assume that $C(k)$ is true, for $k \geq 0$. Consider a new element that arrives, say (x, y), and consider Algorithm 2 applied on this element. We consider four possible cases.

(I) If $x = d$, and $d \in H$, then $\hat{f}_d$ is incremented by 1, and it can be verified (Lines 3–11) that Σ_d increases by at most 1 (and may even decrease). Thus $C(k+1)$ is true.

(II) If $x = d$, and $d \notin H$, then initially, $\hat{f}_d$ and Σ_d are both 1 (line 17). If $|H| \leq s_1$, then both $\hat{f}_d$ and Σ_d remain 1, and $C(k+1)$ is true. Suppose $|H| > s_1$, then both $\hat{f}_d$ and Σ_d will go down to 0, since H_d will be discarded from H. Thus $C(k+1)$ is true.

(III) If $x \neq d$, and $x \in H$, then neither $\hat{f}_d$ nor Σ_d change.

(IV) Finally, if $x \neq d$ and $x \notin H$, then it is possible that $\hat{f}_d$ is decremented (line 20). In this case, if $\Sigma_d > 0$, then Σ_d is also decremented (line 22), and $C(k+1)$ is satisfied. If $\Sigma_d = 0$, then $C(k+1)$ is trivially satisfied since $\hat{f}_d \geq 0$. □

Lemma 4 *Subject to Constraint 1,* $\hat{f}_{d,s} \geq f_{d,s} - \epsilon_2 f_d - \epsilon_1 N$.

Proof Note that each time the tuple (d, s) occurs in the stream, $\hat{f}_{d,s}$ is incremented in Algorithm 2. But $\hat{f}_{d,s}$ can be less than $f_{d,s}$ because of decrements in Lines 9 or 19 in Algorithm 2. We consider these two cases separately.

Let $\Sigma_d = \sum_{(s,\cdot)\in H_d} \hat{f}_{d,s}$. For decrements in Line 9, we observe that each time this line is executed, Σ_d reduces by $s_2 + 1$. From Lemma 3, we know that $\Sigma_d \leq \hat{f}_d \leq f_d$. Thus the total number of times $\hat{f}_{d,s}$ is decremented due to Line 9 is no more than $\frac{f_d}{s_2+1}$. From Constraint 2, we know $\frac{1}{s_2} < \epsilon_2$, and $\frac{f_d}{s_2+1} < \epsilon_2 f_d$.

For decrements in Line 23, we observe that $\hat{f}_{d,s}$ is decremented in Line 23 no more than the number of decrements to $\hat{f}_d$, which was bounded by $\frac{N}{s_1}$ in Lemma 1. From Constraint 1, this is no more than $\epsilon_1 N$. □

Lemma 5 *For any value d that gets reported in line 3 of Algorithm 3, any value s of the secondary dimension that occurs with d such that $f_{d,s} > \phi_2 f_d$, will be identified by line 6 of Algorithm 3 as a CHH occurring along with d.*

Proof From Lemma 4,

$$\begin{aligned}\hat{f}_{d,s} &\geq f_{d,s} - \epsilon_2 f_d - \epsilon_1 N \\ &> \phi_2 f_d - \epsilon_2 f_d - \epsilon_1 N \\ &= (\phi_2 - \epsilon_2) f_d - \epsilon_1 N \\ &\geq (\phi_2 - \epsilon_2) \hat{f}_d - \epsilon_1 N\end{aligned}$$

where we have used $f_d \geq \hat{f}_d$. The lemma follows since $(\phi_2 - \epsilon_2)\hat{f}_d - \epsilon_1 N$ is the threshold used in line 5 of Algorithm 3 to report a value of the secondary dimension as a CHH. □

Lemma 6 *Under Constraints 1 and 2, for any value of d that is reported as a heavy-hitter along the primary dimension, then for a value s' along the secondary dimension, if $f_{d,s'} < (\phi_2 - \epsilon_2) f_d$, then the pair (d, s') will not be reported as a CHH.*

Proof We will prove the contrapositive of the above statement. Consider a value s such that (d, s) is reported as a CHH. Then, we show that $f_{d,s} \geq (\phi_2 - \epsilon_2) f_d$.

If (d, s) is reported, then it must be true that $\hat{f}_{d,s} \geq (\phi_2 - \frac{1}{s_2}) \hat{f}_d - \frac{N}{s_1}$ (Algorithm 3, line 5). Using $f_{d,s} \geq \hat{f}_{d,s}$, and $\hat{f}_d \geq f_d - \frac{N}{s_1}$, we get:

$$
\begin{aligned}
f_{d,s} &\geq \hat{f}_{d,s} \\
&\geq \left(\phi_2 - \frac{1}{s_2}\right) \hat{f}_d - \frac{N}{s_1} \\
&\geq \left(\phi_2 - \frac{1}{s_2}\right) \left(f_d - \frac{N}{s_1}\right) - \frac{N}{s_1} \\
&= \left(\phi_2 - \frac{1}{s_2}\right) f_d - \frac{N}{s_1} \left(1 + \phi_2 - \frac{1}{s_2}\right) \\
&\geq \left(\phi_2 - \frac{1}{s_2}\right) f_d - \frac{f_d}{(\phi_1 - \epsilon_1) s_1} \left(1 + \phi_2 - \frac{1}{s_2}\right) \\
&\quad \left(\text{since } d \text{ gets reported, by Lemma 2, } f_d \geq (\phi_1 - \epsilon_1) N \Rightarrow N \leq \frac{f_d}{\phi_1 - \epsilon_1}\right) \\
&= \left(\phi_2 - \frac{1}{s_2} - \frac{1}{(\phi_1 - \epsilon_1) s_1} \left(1 + \phi_2 - \frac{1}{s_2}\right)\right) f_d \\
&\geq f_d (\phi_2 - \epsilon_2) (\text{using Constraint 2})
\end{aligned}
$$

□

Lemmas 6, 5, and 2 together yield the following.

Theorem 1 *If Constraints 1 and 2 are satisfied, then Algorithms 1, 2 and 3 satisfy all the four requirements of Problem 2.*

3.3 Implementation

We discuss a data structure for implementing an element update in $O(1)$ time, based on the idea discussed by Demaine et al. (2002b). The data structure is a fixed pool of counters, all of which start in the same "group" but eventually get clustered into different groups. All counters in the same group are connected in a doubly linked list, and all counters in the same group have the same frequency, so the frequency can actually be stored on a per-group basis, and the individual counters need only store the identifiers of the items they keep track of. The first group has the frequency of its elements stored explicitly, and all the other groups maintain the difference between the frequency of the items in that group and the frequency of the items in the previous group. Groups are maintained in sorted order of the frequencies. This way, the task of decrementing the frequency of all items can be performed by simply decrementing the (absolute) frequency of the first group. Details are discussed in Sect. 3.3 of Demaine et al. (2002b).

We note that a group is dropped when (a) it runs out of all counters it started with, which happens with the initial group where all counters in the pool get placed into

different groups, or, (b) it becomes the first group in the (sorted) order of groups at some point but its "value" drops to 0 eventually. Since newly created groups need to be inserted between existing groups (if the difference between "value" s of two consecutive groups are more than 1), we suggest linking the groups also in a doubly linked list, which can be accomplished by making the pointer from the last counter in a group point to the first counter in the next group, and making a pointer from the first counter in a group point to the last counter in the previous group.

Demaine et al. (2002b) it is assumed that given an item, its corresponding counter can be looked up in $O(1)$ time. To achieve this, we store the items in a hash table where in the (key, value) pair in the hashtable entry, the "value" is a pointer to the counter that stores the frequency of that item. Before a group is deleted, the corresponding items should be first deleted from the hash table to maintain consistency between the hashtable and the group-based data structure.

We demonstrate in Table 1 how the data structure changes as elements arrive in a simple example stream: 10, 8, 9, 10, 8, 7, 6. Note that the decrement of counters is accomplished by the decrement of the value of the first group in $O(1)$. If the value of the first group falls to 0 after decrement, then we will need to drop the items in the first group from the hashtable, but it will typically be a few items for most streams.

3.4 Analysis

We analyze the space complexity of the algorithm. In Theorem 1, we showed that the Algorithms 2 and 3 solve the Approximate CHH detection problem, as long as Constraints 1 and 2 are satisfied.

Space Complexity in terms of s_1 and s_2 In our algorithm, we maintain at most s_2 counters for each of the (at most) s_1 distinct values of the primary dimension in H. Hence, the size of our sketch is $O(s_1 + s_1 s_2) = O(s_1 s_2)$. We now focus on the following question. *What is the setting of s_1 and s_2 so that the space complexity of the sketch is minimized while meeting the constraints required for correctness.?*

Theorem 2 *Let $\alpha = \left(\frac{1+\phi_2}{\phi_1-\epsilon_1}\right)$. Subject to Constraints 1 and 2, the space of the data structure is minimized by the following settings of s_1 and s_2.*

- *If $\epsilon_1 \geq \frac{\epsilon_2}{2\alpha}$, then $s_1 = \frac{2\alpha}{\epsilon}$ and $s_2 = \frac{2}{\epsilon_2}$. In this case, the space complexity is $O\left(\frac{1}{(\phi_1-\epsilon_1)\epsilon_2^2}\right)$.*
- *If $\epsilon_1 < \frac{\epsilon_2}{2\alpha}$, then $s_1 = \frac{1}{\epsilon_1}$, and $s_2 = \frac{1}{\epsilon_2-\alpha\epsilon_1}$. In this case, the space complexity is $O(\frac{1}{\epsilon_1\epsilon_2})$.*

Proof Let $\sigma_1 = \frac{1}{s_1}, \sigma_2 = \frac{1}{s_2}$. The problem is now to maximize $\sigma_1\sigma_2$. Constraints 1 and 2 can be rewritten as follows.

- **Constraint 1:** $\sigma_1 \leq \epsilon_1$
- **Constraint 2:** $\alpha\sigma_1 + \sigma_2 \leq \epsilon_2$

Table 1 Change in data structure as stream elements arrive

Stream Item	Starting state of data structure	Action taken	Final state of data structure
None		Create a group G_1 with four counters. Initialize the value of G_1 to 0. Create a blank hashtable	Group 1 has four counters, none of which maintains the count of any element. The hashtable has no entries
10	Group 1 has four counters, none of which maintains the count of any element. The hashtable has no entries	Take one of the counters from group G_1, assign the new item 10 to it. Put it in a new group G_2, and initialize the value of G_2 to 1. Put a (key, value) pair in the hashtable, the key being 10 and the value being a pointer to the counter in G_2 for item 10	Groups G_1 and G_2, with values 0 and 1 respectively. The value 1 for G_2 represents that item 10 has a frequency of 1
8	Groups G_1 and G_2, with values 0 and 1 respectively. 3 counters in G_1 with no items assigned, 1 counter in G_2 for 10	Take a counter from G_1, assign it to item 8 and put it in G_2. Put 8 in hashtable and make the corresponding pointer point to the counter for 8	Groups G_1 and G_2, with values 0 and 1 respectively. 2 counters in G_1 with no items assigned, 2 counters in G_2 for 10 and 8
9	Groups G_1 and G_2, with values 0 and 1 respectively. 2 counters in G_1 with no items assigned, 2 counters in G_2 for 10 and 8	Take a counter from G_1, assign it to item 9 and put it in G_2. Put 9 in hashtable and make the corresponding pointer point to the counter for 9	Groups G_1 and G_2, with values 0 and 1 respectively. 1 counter in G_1 with no items assigned, 3 counters in G_2 for 10, 8 and 9
10	Groups G_1 and G_2, with values 0 and 1 respectively. 1 counter in G_1 with no items assigned, 3 counters in G_2 for 10, 8 and 9	Take the counter for 10 from G_2, put it in a new group G_3 with value = 1, make G_3 the next group after G_2	Groups G_1, G_2 and G_3, with values being 0, 1 and 1 respectively. The value 1 for G_3 indicates that 10 has a frequency of 1 + 1 = 2. G_1 still has one counter with no item assigned
8	Groups G_1, G_2 and G_3, with values being 0, 1 and 1 respectively	Move the counter for 8 from G_2 to G_3	Groups G_1, G_2 and G_3, with values being 0, 1 and 1 respectively. G_1 has a counter with no value assigned, G_2 has the single counter for 9, and G_3 has counters for 8 and 10
7	Groups G_1, G_2 and G_3, with values being 0, 1 and 1 respectively	Take the single counter from G_1, assign it to 7, and put it in group G_2. Since G_1 has no more counters after this, we delete the group G_1, and G_2 becomes the first group now, with value 1	Groups G_1 and G_2, with values 1 for both, G_1 having counters for 7 and 9, G_2 having counters for 8 and 10
6	Groups G_1 and G_2, with values 1 for both, G_1 having counters for 7 and 9, G_2 having counters for 8 and 10	There are no more counters to assign to 6. So, we decrement the value of G_1 by 1. Since the value for G_1 drops to 0, we drop the items 7 and 9 from the hashtable, and drop group G_1	Group G_2 with value 1. Since G_2 is the first group now, this implies 8 and 10 have a (decremented) frequency estimate of 1 each

First, we note that any assignment $(\sigma_1, \sigma_2) = (x, y)$ that maximizes $\sigma_1\sigma_2$ must be tight on Constraint 2, i.e. $\alpha x + y = \epsilon_2$. This can be proved by contradiction. Suppose not, and $\alpha x + y < \epsilon_2$, and xy is the maximum possible. Now, there is a solution $\sigma_1 = x$, and $\sigma_2 = y'$, such that $y < y'$, and Constraints 1 and 2 are still satisfied. Further, $xy' > xy$, showing that the solution (x, y) is not optimal.

Thus, we have:

$$\sigma_2 = \epsilon_2 - \alpha\sigma_1 \tag{1}$$

Thus the problem has reduced to: **Maximize** $f(\sigma_1) = \sigma_1 (\epsilon_2 - \alpha\sigma_1)$ **subject to** $\sigma_1 \leq \epsilon_1$.

Consider

$$f'(\sigma_1) = \epsilon_2 - 2\alpha\sigma_1$$

We consider two cases.

– **Case I:** $\epsilon_1 \geq \frac{\epsilon_2}{2\alpha}$.
Setting $f'(\sigma_1) = 0$, we find that the function reaches a fixed point at $\sigma_1 = \frac{\epsilon_2}{2\alpha}$. At this point, $f''(\sigma_1) = -2\alpha$, which is negative. Hence $f(\sigma_1)$ is maximized at $\sigma_1 = \frac{\epsilon_2}{2\alpha}$. We note that this value of σ_1 does not violate Constraint 1, and hence this is a feasible solution. In this case, the optimal settings are: $\sigma_1 = \frac{\epsilon_2}{2\alpha}$ and $\sigma_2 = \frac{\epsilon_2}{2}$. Thus $s_1 = \frac{2\alpha}{\epsilon}$ and $s_2 = \frac{2}{\epsilon_2}$. The space complexity is $O(\frac{1}{\sigma_1\sigma_2}) = O(\frac{4\alpha}{\epsilon_2^2})$.

– **Case II:** $\epsilon_1 < \frac{\epsilon_2}{2\alpha}$
The function $f(\sigma_1)$ is increasing for σ_1 from 0 to $\frac{\epsilon_2}{2\alpha}$. Hence this will be maximized at the point $\sigma_1 = \epsilon_1$. Thus, in this case the optimal settings are $\sigma_1 = \epsilon_1$, and $\sigma_2 = \epsilon_2 - \alpha\epsilon_1$. Thus, $s_1 = \frac{1}{\epsilon_1}$, and $s_2 = \frac{1}{\epsilon_2 - \alpha\epsilon_1}$. The space complexity is: $O(\frac{1}{\epsilon_1(\epsilon_2 - \alpha\epsilon_1)})$.
We note that since $\epsilon_2 > 2\alpha\epsilon_1$, we have $(\epsilon_2 - \alpha\epsilon_1) > \frac{\epsilon_2}{2}$, and hence the space complexity is $O(\frac{1}{\epsilon_1\epsilon_2})$. □

Theorem 3 *The time taken to update the sketch on receiving an element of the stream is* $O(1)$.

Proof The analysis is based on the data structure discussed in Sect. 3.3. While we discussed it for one-dimensional streams, such a data structure can be maintained for each substream $S_d = \{(x, y) | ((x, y) \in S) \wedge (x = d)\}$ induced by a distinct item d in the primary dimension. In processing an element (x, y) of the stream by Algorithm 2, the following three scenarios may arise.

1. x is present in H, and y is present in H_x. The time taken to look up and increment $\hat{f}_x$ from H and $\hat{f}_{x,y}$ from H_x is O(1).
2. x is present in H, but y is not in H_x. Then y needs to be inserted into H_x, which can be done in $O(1)$ time, as explained in Sect. 3.3.
3. x is not present in H. Then x needs to be inserted into H, which again takes time $O(1)$, as explained in Sect. 3.3.

The time complexity to update the sketch on receiving each element is the maximum of these three, which establishes the claim. □

4 Experiments

We implemented our algorithm for finding CHH using Java, and evaluated it using two datasets.

- **NGram** is the "English fiction" 2-g dataset based on the Google n-gram dataset (http://storage.googleapis.com/books/ngrams/books/datasetsv2.html), extracted from books predominantly in the English language that a library or publisher identified as fiction. We took a uniform random sample of size 944,598,580 from this dataset. We will refer to the two elements of a 2-g as the "first gram" and the "second gram" respectively.
- **Synthetic** We generated a synthetic dataset of 100 million tuples, each having a pair of elements. The distribution of the primary dimension was as follows: there were 500 items designated as heavy-hitters, each of them having a frequency of approximately 9000. We introduced some randomness to make the actual frequencies vary a little around 9000. For each heavy-hitter, we had 30 CHHs, and each of the CHHs had a frequency of approximately 270. Once again, some randomness was used to make the actual frequencies of the CHHs vary a little around 270. We filled in the remaining of the stream with non-heavy hitters to make the distribution a long-tailed one.

Objective The goal of our experiments were twofold. First, to learn about typical frequency distributions along both the dimensions in real two-dimensional data streams; second, to demonstrate how the space budget (and hence, the allocated memory) influences the accuracy of our algorithm in practice.

For the first objective, we ran a naive algorithm on the "NGram" dataset, where all the distinct first grams were stored, and for each distinct first gram, all the distinct second grams were stored. We identified (exactly) the frequent values along both the dimensions for $\phi_1 = 0.001$ and $\phi_2 = 0.001$. Only 91 of the 514,249 distinct first grams were reported as heavy-hitters. For the secondary dimension, we ranked the first grams based on the number of distinct second grams they co-occurred with, and the number of distinct second grams for the top eight are shown in Fig. 1. The number of distinct second grams, co-occurring with the first grams, varies between 10 million and 100 million, but the number of CHH second grams vary between 10 and 100 only, orders of magnitude lower than the number of distinct values of the second grams. Note that the Y-axis in Fig. 1 is in log scale. This shows that the distribution of the primary attribute values, as well as that of the secondary attribute values for a given value of the primary attribute, are very skewed, and hence call for the design of small-space approximation algorithms like ours.

Since the "NGram" dataset is based on English fiction text, we observed some interesting patterns while working with the dataset: pairs of words that occur frequently together, as reported by this dataset, are indeed words whose co-occurrence intuitively look natural. We present some examples in Table 2, along with their frequencies:

For the second objective, we tested the small-space algorithm on both the datasets (with different values of s_1 and s_2): "NGram" and "Synthetic". To test the accuracy of our small-space algorithm, we derived the "ground truth", i.e., a list of the *actual* heavy-hitters along both the dimensions along with their *exact* frequencies, by employing

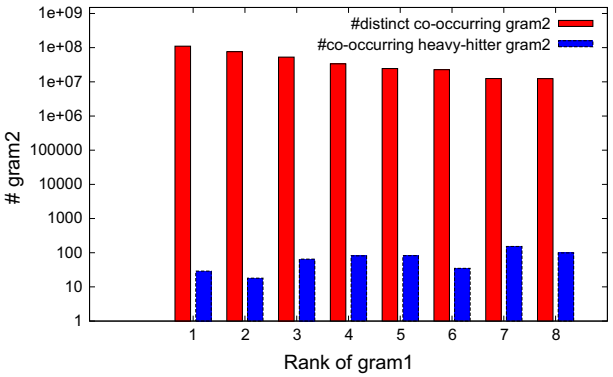

Fig. 1 Basic statistics for "NGram". On the X-axis are the ranks of the eight (heavy-hitter) first gram values, that co-appear with maximum number of distinct second grams. For each first gram, the Y-axis shows (1) the number of distinct second grams co-occurring with it, (2) the number of heavy-hitter second grams co-appearing with it. Note that the Y-axis is logarithmic

Table 2 Pairs of words frequently occurring together

Gram1	Frequency of Gram1	Gram2	Frequency of Gram2 along with Gram1
Are	1989774	Hardly	4717
Are	1989774	Meant	5031
Still	1601172	Remained	4798
Out	1777906	Everything	5497
Was	2373607	Present	7932
Was	2373607	Deserted	7641
Look	1226326	Outside	2052
Could	1215055	Suggest	5081

the naive algorithm we have already mentioned above. For the "Synthetic" dataset, the parameters in the naive algorithm were set to $\phi_1 = 8.9 \times 10^{-2}$ and $\phi_2 = 7.8 \times 10^{-3}$.

Observations We define the error statistic in estimating the frequency of a heavy-hitter value d of the primary attribute as $\frac{f_d - \hat{f}_d}{N}$, and in Figs. 2 and 3, for each value of s_1, we plot the maximum and the average of this error statistic over all the heavy-hitter values of the primary attribute. We observed that both the maximum and the average fell sharply as s_1 increased. For "NGram", even by using a space budget (s_1) as low as 1000, the maximum error statistic was only 0.03 %. For "Synthetic", however, we had to use larger values of s_1, because ϕ_1 was orders of magnitude lower. Intuitively, when ϕ_1 is lower, the result-set can have more heavy-hitters, and hence we need a higher space budget to accommodate the heavy-hitters. As we have already discussed, we had only 91 heavy-hitters in "NGram" but 5000 of them in "Synthetic". However, even with the lowest value of s_1(20,000), the average error for the first attribute for "Synthetic" was as low as 4.88972×10^{-5}. Some theoretical lower bound for s_1 actually follows from Constraint 1 in Sect. 3.4.

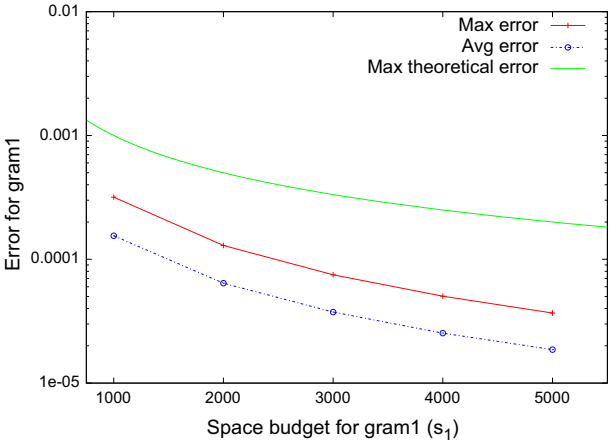

Fig. 2 Error statistic in estimating the frequencies of the heavy-hitter first grams from "NGram"

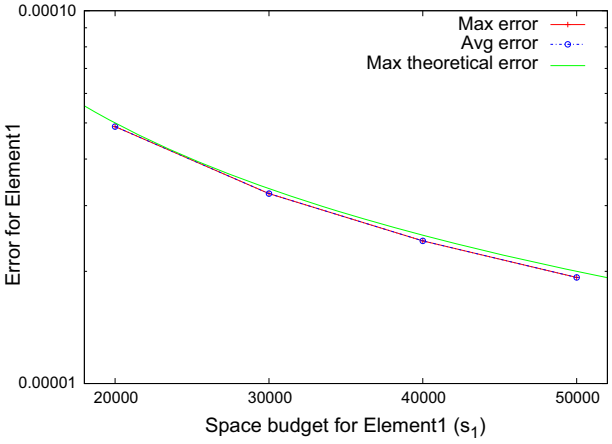

Fig. 3 Error statistic in estimating the frequencies of the heavy-hitters along the first dimension from "Synthetic"

The graphs in Fig. 4 show the results of running our small-space algorithm with different values of s_1 as well as s_2. We define the error statistic in estimating the frequency of a CHH s (that occurs along with a heavy-hitter primary attribute d) as $\frac{f_{d,s}-\hat{f}_{d,s}}{f_d}$, and for each combination of s_1 and s_2, we plot the theoretical maximum, the experimental maximum and the average of this error statistic over all CHH attributes. Here also, we observed that both the maximum and the average fall sharply as s_1 increases. Also, when we see the errors upon varying s_2 for identical values of s_1 in Fig. 4, we see that the error for the second attribute decreases with increasing s_2. For example, for the "NGram" data, for $s_1 = 1000$, the average value of $\frac{f_{d,s}-\hat{f}_{d,s}}{f_d}$ is 0.002777 for $s_2 = 100$, whereas it falls to 0.0018 for $s_2 = 300$. The low error even for $s_2 = 100$ suggests like it is a reasonable value for space budget in a practical setting.

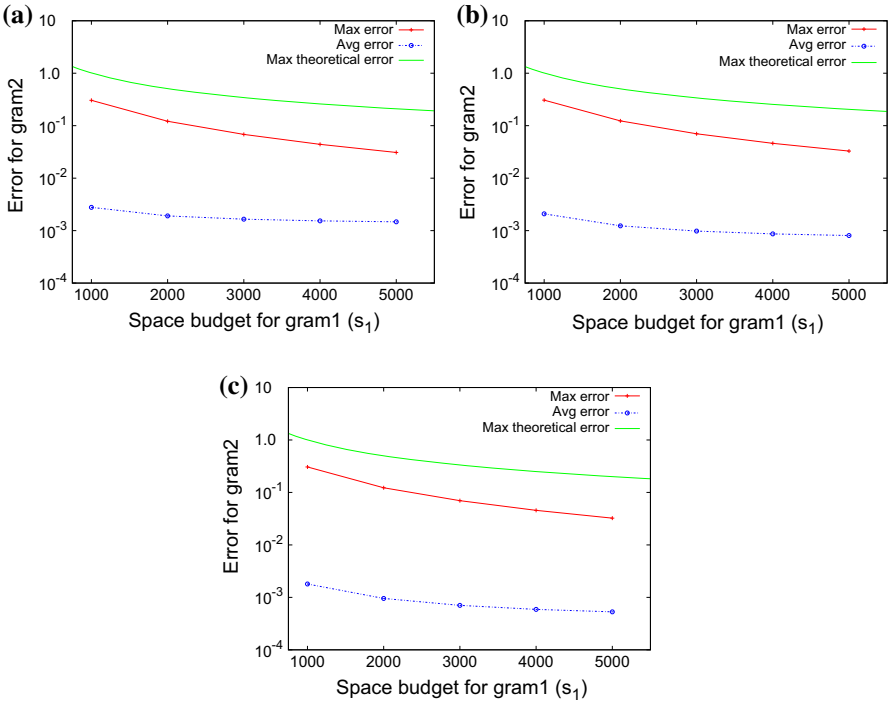

Fig. 4 Error statistic in estimating the frequencies of the CHH second grams in "NGram". The three graphs are for $s_2 = 100$, $s_2 = 200$ and $s_2 = 300$

For "Synthetic", in Fig. 5, we present the change in the error statistic for the CHHs as s_2 increases, and the three subplots are each for a different value of s_1. We see that the error statistic decreases steadily as s_2 increases, e.g., with $s_1 = 30{,}000$, the average error statistic is 0.02949 for $s_2 = 20$, but falls to 0.012578 for $s_2 = 50$. This shows that although the s_1 values are an order of magnitude higher for "Synthetic" than that for "NGram", the s_2 values for "Synthetic" could be very well kept under 100 for errors less than 2.0 %.

We experimented with a wide range of values for s_2 and reported only a subset of them here. The rate of change in the error statistic for the second attribute depends on a number of factors, for example, on the distribution of the first attribute values and the distribution of the second attribute values occurring with different first attribute values. Intuitively, once s_2 becomes so large that the inner data structure H_x in Algorithm 2 does not need to go through the decrement and deletion steps in lines 7–11 very often, then we will not see much difference between the $f_{d,s}$ and the $\hat{f_{d,s}}$ values, and hence the error statistic for the second dimension will reach saturation. Another intuitive explanation is as follows: the maximum theoretical error for the second attribute is $\frac{1}{\phi_1 s_1} + \frac{1}{s_2}$. If we call this function $f(s_1, s_2)$, then $f(s_1, s_2)$ is a monotonically decreasing function of s_2, and $\frac{\partial f}{\partial s_2} = -\frac{1}{{s_2}^2}$, which implies the absolute value of the rate of change of $f(s_1, s_2)$ also decreases with increasing s_2. For example, for the "Synthetic" dataset,

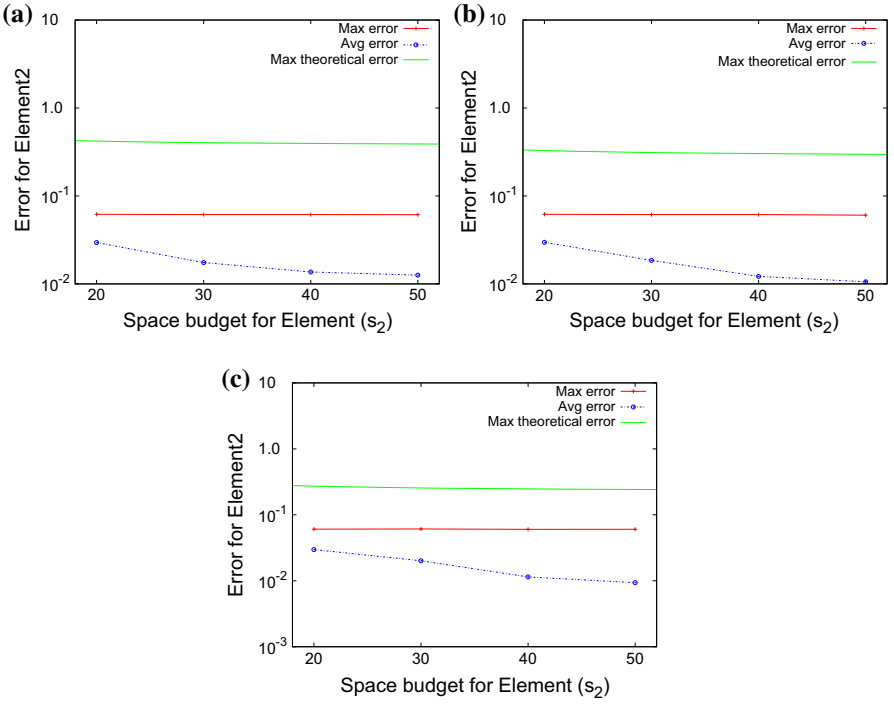

Fig. 5 Error statistic in estimating the frequencies of the CHH second grams in "Synthetic". The three graphs are for $s_1 = 30{,}000$, $s_1 = 40{,}000$ and $s_1 = 50{,}000$. Note that the maximum theoretical error and the actual maximum error look almost horizontal because the Y-axis is in the log scale

for $s_1 = 30{,}000$, with $s_2 = 20$, the average error statistic for the second attribute is 0.02949, and for $s_2 = 50$, it is 0.012578, so the relative change is $(0.02949 - 0.012578)/0.02949 = 57.34\,\%$. However, with $s_1 = 30{,}000$, when s_2 is changed from 70 to 100 (an increase of 30 points again), the error changes from 0.002554 to 0.001449, so the relative change is 43.26 %, so the decrease in the rate of change already starts showing.

5 Conclusion and future work

For two-dimensional data streams, we presented a small-space approximation algorithm to identify the heavy-hitters along the secondary dimension from the substreams induced by the heavy-hitters along the primary. We theoretically studied the relationship between the maximum errors in the frequency estimates of the heavy-hitters and the space budgets; computed the minimum space requirement along the two dimensions for user-given error bounds; and tested our algorithm to show the space-accuracy tradeoff for both the dimensions.

Identifying the heavy-hitters along any one dimension allows us to split the original stream into several important substreams; and take a closer look at each one to identify

the properties of the heavy-hitters. In future, we plan to work on computing other properties of the heavy-hitters. For example, as we have already discussed in Sect. 4, our experiments with the naive algorithm (on both the datasets) revealed that the number of *distinct* secondary dimension values varied quite significantly across the different (heavy-hitter) values of the primary dimension. For any such data with high variance, estimating the variance in small space (Babcock et al. 2003; Zhang and Guan 2007) over a sliding window is an interesting problem in itself. Moreover, for data with high variance, the simple arithmetic mean is not an ideal central measure, so finding different quantiles, once again in small space, can be another problem worth studying.

Acknowledgments The authors were supported in part by the National Science Foundation through Grants NSF CNS-0834743 and CNS-0831903.

References

Ananthakrishna R, Das A, Gehrke J, Korn F, Muthukrishnan S, Srivastava D (2003) Efficient approximation of correlated sums on data streams. IEEE Trans Knowl Data Eng 15(3):569–572

Babcock B, Datar M, Motwani R, O'Callaghan L (2003) Maintaining variance and k-medians over data stream windows. In: Proceedings of the twenty-second ACM SIGACT-SIGMOD-SIGART symposium on principles of database systems (PODS), pp 234–243

Busch C, Tirthapura S (2007) A deterministic algorithm for summarizing asynchronous streams over a sliding window. In: STACS

CAIDA: OC48 traces dataset. https://data.caida.org/datasets/oc48/oc48-original/20020814/5min/

Charikar M, Chen K, Farach-Colton M (2004) Finding frequent items in data streams. Theor Comput Sci 312(1):3–15

Cormode G, Muthukrishnan S (2003) What's hot and what's not: tracking most frequent items dynamically. In: Proceedings of the 22nd ACM SIGMOD international conference on management of data/principles of database systems (PODS), pp 296–306

Cormode G, Muthukrishnan S (2005) An improved data stream summary: the count-min sketch and its applications. J Algorithms 55(1):58–75

Cormode G, Hadjieleftheriou M (2009) Finding the frequent items in streams of data. Commun ACM 52(10):97–105

Cormode G, Tirthapura S, Xu B (2009) Time-decayed correlated aggregates over data streams. In: Proceedings of the SIAM international conference on data mining (SDM), pp 269–280

Cullingford RE (2009) Correlation and collaboration in anomaly detection. In: Cybersecurity applications & technology conference for homeland security (CATCH), pp 251–254

Demaine ED, López-Ortiz A, Munro JI (2002a) Frequency estimation of internet packet streams with limited space. In: Proceedings of the 10th annual european symposium (ESA), pp 348–360

Demaine ED, López-Ortiz A, Munro JI (2002b) Frequency estimation of internet packet streams with limited space. Tech rep

Estan C, Savage S, Varghese G (2003) Automatically inferring patterns of resource consumption in network traffic. In: Proceedings of the ACM SIGCOMM 2003 conference on applications, technologies, architectures, and protocols for computer communication (SIGCOMM), pp 137–148

Estan C, Varghese G (2002) New directions in traffic measurement and accounting. In: Proceedings of the ACM SIGCOMM 2002 conference on applications, technologies, architectures, and protocols for computer communication (SIGCOMM), pp 323–336

Gehrke J, Korn F, Srivastava D (2001) On computing correlated aggregates over continual data streams. In: Proceedings of the 20th ACM SIGMOD international conference on management of data (SIGMOD), pp 13–24

Google: Google n-grams dataset. http://storage.googleapis.com/books/ngrams/books/datasetsv2.html

Greenwald M, Khanna S (2001) Space-efficient online computation of quantile summaries. In: Proceedings of the 20th ACM SIGMOD international conference on management of data (SIGMOD), pp 58–66

Karp RM, Shenker S, Papadimitriou CH (2003) A simple algorithm for finding frequent elements in streams and bags. ACM Trans Database Syst 28:51–55

Manku GS, Motwani R (2002) Approximate frequency counts over data streams. In: Proceedings of 28th international conference on very large data bases (VLDB), pp 346–357

Misra J, Gries D (1982) Finding repeated elements. Sci Comput Program 2(2):143–152

Tirthapura S, Woodruff DP (2012) A general method for estimating correlated aggregates over a data stream. In: Proceedings of the ICDE, pp 162–173

Xu B, Tirthapura S, Busch C (2008) Sketching asynchronous data streams over sliding windows. Distrib Comput 20(5):359–374

Zhang L, Guan Y (2007) Variance estimation over sliding windows. In: Proceedings of the twenty-sixth ACM SIGACT-SIGMOD-SIGART symposium on principles of database systems (PODS), pp 225–232

Zhang Y, Singh S, Sen S, Duffield NG, Lund C (2004) Online identification of hierarchical heavy hitters: algorithms, evaluation, and applications. In: Internet measurement conference (IMC), pp 101–114