Data Min Knowl Disc (2018) 32:162–186
https://doi.org/10.1007/s10618-017-0526-x

Fast and accurate mining of correlated heavy hitters

Italo Epicoco[1,2] · Massimo Cafaro[1,2] ·
Marco Pulimeno[1]

Received: 15 November 2016 / Accepted: 22 June 2017 / Published online: 5 July 2017

Abstract The problem of mining correlated heavy hitters (CHH) from a two-dimensional data stream has been introduced recently, and a deterministic algorithm based on the use of the Misra–Gries algorithm has been proposed by Lahiri et al. to solve it. In this paper we present a new counter-based algorithm for tracking CHHs, formally prove its error bounds and correctness and show, through extensive experimental results, that our algorithm outperforms the Misra–Gries based algorithm with regard to accuracy and speed whilst requiring asymptotically much less space.

Keywords Data stream mining · Correlation · Heavy hitters

1 Introduction

Mining heavy hitters (also called frequent items), is a well known and studied data mining task. Algorithms for detecting heavy hitters in a data stream can be classified as being either *counter* or *sketch* based, depending on their main data structure. Counter-

Responsible editor: Toon Calders.

✉ Massimo Cafaro
massimo.cafaro@unisalento.it; massimo.cafaro@cmcc.it

Italo Epicoco
italo.epicoco@unisalento.it; italo.epicoco@cmcc.it

Marco Pulimeno
marco.pulimeno@unisalento.it

1 University of Salento, Lecce, Italy

2 Euro-Mediterranean Center on Climate Change, CMCC Foundation, Lecce, Italy

based algorithms rely on a fixed number of counters in order to keep track of stream items.

Sketch-based algorithms, as their name suggests, monitor the input data stream by using a set of counters which are stored in a sketch data structure, typically a two-dimensional array. In this case, hash functions map items to their corresponding sketch cells. Counter-based algorithms are deterministic and sketch-based ones are randomized, thus providing a probabilistic guarantee.

Regarding the counter-based algorithms, the first sequential algorithm has been designed by Misra and Gries (1982). About twenty years later, the same algorithm was rediscovered independently by both Demaine et al. (2002) (the so-called *Frequent* algorithm) and by Karp et al. (2003). Among the recently developed counter-based algorithms we recall here *Sticky Sampling*, *Lossy Counting* (Manku and Motwani 2002) and *Space Saving* (Metwally et al. 2006). Notable examples of sketch-based algorithms are provided by *CountSketch* (Charikar et al. 2002), *Group Test* (Cormode and Muthukrishnan 2005b), *hCount* (Jin et al. 2003) and *Count-Min* (Cormode and Muthukrishnan 2005a).

In some applications, frequent items must be detected with the additional constraint that recent items must be weighted more than former items. The underlying assumption is that, in these applications, recent data is more useful and valuable than older, stale data. Therefore, each item in the stream has an associated timestamp that will be used to determine its weight. In practice, instead of estimating frequency counts, we are required to estimate *decayed counts*. Two different models have been proposed in the literature: the *sliding window* and the *time fading* model.

In the sliding window model (Datar et al. 2002), freshness of recent items is captured by a time window, i.e., a temporal interval of fixed size in which only the most recent N items are taken into account; detection of frequent items is strictly related to those items falling in the window. The items in the stream become stale over time, since the window periodically slides forward. In the time fading model (Manjhi et al. 2005; Cormode et al. 2008; Chen and Mei 2014; Cafaro et al. 2016a) there is no window sliding over time; freshness of more recent items is instead emphasized by *fading* the frequency count of older items. This is achieved by using a decaying factor $0 < \lambda < 1$ to compute an item's *decayed count* (also called *decayed frequency*) through decay functions that assign greater weight to more recent elements. The older an item, the lower its decayed count is: in the case of exponential decay, the weight of an item occurred n time units in the past is $e^{-\lambda n}$, which is an exponentially decreasing quantity.

Regarding parallel algorithms, message-passing based parallel versions of the Frequent and Space Saving algorithms are provided by Cafaro and Tempesta (2011), Cafaro and Pulimeno (2016) and Cafaro et al. (2016b). Shared-memory algorithms have been designed as well, and we recall here the parallel version of Frequent (Zhang et al. 2014), the parallel version of Lossy Counting (Zhang 2012) and the parallel versions of Space Saving (Cafaro et al. 2017; Roy et al. 2012; Das et al. 2009). Novel shared-memory parallel algorithms for frequent items were recently proposed in Tangwongsan et al. (2014). Accelerator based algorithms for frequent items exploiting a GPU (Graphics Processing Unit) include Govindaraju et al. (2005); Erra and Frola (2012).

The problem of mining correlated heavy hitters has been introduced recently by Lahiri et al. (2016). Data mining problems that require to compute correlated heavy hitters may be found in the context of network monitoring and management, as well as anomaly and intrusion detection. As an example, consider the stream of pairs (source address, destination address) of IP packets passing through a router. Identifying the nodes that are responsible for the majority of the traffic through that router (frequent items over a single dimension) could be useful, but it is also interesting to discover, for all of the frequent sources, what are the destinations that receive the majority of connections by the same source. Important sources are detected as frequent items over the first dimension, then we search for the frequent destinations in the context of each one of those sources, i.e., the correlated heavy hitters of the stream. In order to formally state the problem, we recall here preliminary notation and definitions.

Definition 1 The frequency f_{xy} of the tuple (x, y) in the stream

$$\sigma =< (x_1, y_1), (x_2, y_2), \ldots, (x_n, y_n) >$$

is given by $f_{xy} = |\{i : (x = x_i) \wedge (y = y_i)\}|$.

Definition 2 The frequency f_x of an item which appears as first element in the tuple (x, y) in the stream σ is given by $f_x = |\{i : (x = x_i)\}|$.

The frequency f_x refers to the frequency of the item x disregarding the second item belonging to the tuple, i.e., the frequency computed when considering the sub-stream induced by the projection of the tuples on the first dimension, which is also referred to as the primary dimension (whose items are referred to as primary items). We are now ready to state the exact correlated heavy hitters problem.

Problem 1 Exact correlated heavy hitters problem.

In the online setting, given a data stream σ of length N made of (x, y) tuples in which the item x is drawn from the universe $\mathcal{U}_1 = \{1, \ldots, m_1\}$ and the item y is drawn from the universe $\mathcal{U}_2 = \{1, \ldots, m_2\}$, two user-defined thresholds ϕ_1 and ϕ_2 such that $0 < \phi_1 < 1$ and $0 < \phi_2 < 1$, the exact correlated heavy hitters (ECHH) problem requires determining all of the (x, y) tuples such that:

$$f_x > \phi_1 N \tag{1}$$

and

$$f_{xy} > \phi_2 f_x. \tag{2}$$

The exact correlated heavy hitters problem can not be solved using limited space and only one pass through the input stream, hence the approximate correlated heavy hitters problem (ACHH) is introduced by Lahiri et al. (2016). We state the problem as follows.

Problem 2 Approximate correlated heavy hitters problem.

Given a data stream σ of length N made of (x, y) tuples in which the item x is drawn from the universe $\mathcal{U}_1 = \{1, \ldots, m_1\}$ and the item y is drawn from the universe

$\mathcal{U}_2 = \{1, \ldots, m_2\}$, two user-defined thresholds ϕ_1 and ϕ_2 such that $0 < \phi_1 < 1$ and $0 < \phi_2 < 1$ and two error bounds ϵ_1 and ϵ_2 such that $0 < \epsilon_1 < \phi_1$ and $0 < \epsilon_2 < \phi_2$, the Approximate Correlated Heavy Hitters (ACHH) problem requires determining all of the primary items x such that

$$f_x > \phi_1 N \tag{3}$$

and no items with

$$f_x \leq (\phi_1 - \epsilon_1) N \tag{4}$$

should be reported; moreover, we are required to determine for each frequent primary candidate x, all of the tuples (x, y) such that

$$f_{xy} > \phi_2 f_x \tag{5}$$

and no tuple (x, y) such that

$$f_{xy} \leq (\phi_2 - \epsilon_2) f_x \tag{6}$$

should be reported.

In this paper we present CSSCHH (cascading space saving correlated heavy hitters) a new counter-based algorithm for tracking CHHs in a two-dimensional data stream and solving the ACHH problem, formally prove its error bounds and correctness and show, through extensive experimental results, that our algorithm outperforms the Misra–Gries based algorithm proposed by Lahiri et al. (2016) (from now on called MGCHH) with regard to accuracy and speed whilst requiring asymptotically much less space.

The rest of this manuscript is organized as follows. In Sect. 2 we provide an overview of related work, we recall in Sect. 3 the MGCHH algorithm introduced by Lahiri et al. (2016) and present our algorithm in Sect. 4. Then, we formally prove its error bound and correctness in Sect. 5. Next, we analyze our algorithm's worst case time and space complexity in Sect. 6. We compare, from a theoretical perspective, CSSCHH against MGCHH in Sect. 7. Extensive experimental results are reported and discussed in Sect. 8. We draw our conclusions in Sect. 9.

2 Related work

The problem of efficiently analyzing two-dimensional data streams in order to gain insights and compute significant statistics has been largely investigated in many forms.

Gehrke et al. (2001), Ananthakrishna et al. (2003) and Cormode et al. (2009) refer to the notion of *correlated aggregates* and present solutions tailored to different contexts. On a two-dimensional stream, i.e., a stream of items' pairs, a correlated aggregate is an aggregate value computed along the second dimension on a set of pairs defined by a particular constraint on the first dimension. A typical correlated aggregate is, for instance, the average value of the items on the second dimension computed for those pairs such that the frequency of the first dimension item is above a fixed threshold.

Mining heavy hitters can also be applied to streams of pairs, when a pair is regarded as a single item. The problem requires finding all of the items which appear in the stream with a frequency greater than a threshold. Manerikar and Palpanas (2009) and Cormode and Hadjieleftheriou (2009) present an overview and comparison of the most common frequent items algorithms, while Zhang et al. (2004) treat the case of multidimensional and hierarchical heavy hitters in the context of network traffic analysis.

A stream of tuples consisting of items can also be processed in order to find frequent itemsets, i.e., sets of items appearing together in a number of tuples above a fixed threshold. In the context of streams of pairs we are interested in finding the frequent two-itemsets, i.e., itemsets consisting of pairs of items. Also, strictly correlated is the notion of association rules, which are implications of the type *first item* $\implies$ *second item*. Given an association rule $x \implies y$, with $(x, y) \in \sigma$, the *support* of the rule is defined as the ratio $\frac{f_{xy}}{N}$, and the *confidence* of the rule is defined as the ratio $\frac{f_{xy}}{f_x}$. The problem of mining association rules entails searching for pairs of items such that their support and confidence are above fixed thresholds. We note here that the confidence constraint as defined for association rules is fully equivalent to the CHH constraint shown in Eq. (2). The difference between mining association rules involving tuples (x, y) and CHHs lies in the support constraint, that is, in the case of association rules we select all of the tuples (x, y) which are frequent, i.e., with support above a fixed threshold whilst for CHHs we select all of the tuples (x, y) in which x is frequent, i.e., its support $\frac{f_x}{N}$ is above a fixed threshold, as in Eq. (1).

Several frequent itemsets algorithm in the offline setting have been proposed, we recall here Apriori (Agrawal et al. 1993), Eclat (Zaki 2000) and FPGrowth (Han et al. 2000), while an overview of streaming algorithms that solve the frequent itemsets problem is given in Cheng et al. (2008).

Mirylenka et al. (2015) introduce the notion of *Conditional Heavy Hitters* and compare it with other related problems, such as association rules and correlated heavy hitters, highlighting how solving these problems actually leads to different outputs, each emphasizing particular aspects of the input data stream. A group of algorithms is proposed and experimentally evaluated with respect to the approximate mining of Conditional Heavy Hitters. One of these, FamilyHH, is based on the same approach we use in our algorithm, but the authors conclude that the algorithm is not particularly suitable for that problem.

Lahiri et al. (2016) introduced the notion of correlated heavy hitters and proposed an approximate solution based on the Misra–Gries algorithm. We will refer to the Misra and Gries algorithm as MG, and to their CHH algorithm as MGCHH. In this paper, we present a new counter-based algorithm for tracking CHHs, with reference to the original problem formulation introduced in Lahiri et al. (2016). Therefore, we compare our solution against MGCHH. In the next section, we recall the MGCHH algorithm.

3 The Misra–Gries based CHH algorithm

This CHH algorithm, recently introduced in Lahiri et al. (2016), is based on a nested application of the Misra and Gries algorithm (Misra and Gries 1982). Before delving into the details of MGCHH, we recall first how MG works. Being counter-based, MG keeps track of stream items by using a data structure holding k counters, i.e., pairs (item, frequency). In particular, given a stream σ of length N, a support threshold ϕ to determine frequent items (i.e., those items whose frequency exceeds ϕN), and an error threshold $0 < \epsilon < 1$, MG requires at least $k = \frac{1}{\epsilon}$ counters to estimate the frequencies of the items with an error less than or equal to ϵN.

The MG algorithm works as follows. Upon receiving an item from the stream, if one of the counters is already monitoring the item, then the counter is updated by increasing by one the frequency of the item. If none of the counters is monitoring the item but there is a counter available in the data structure (i.e., a counter which is not monitoring any item), this counter is then assigned the responsibility of monitoring the received item and its corresponding frequency is set to one. Otherwise, all of the counters in the data structure are already in charge of monitoring an item. In this case, since the number of counters can not exceed k, all of the counters' frequencies are decremented by one. As a result, the counters whose frequencies after the decrement are reset to zero become available to monitor incoming items (since their items are discarded). It is worth recalling here that MG underestimates the frequency of an item; therefore, a single pass of the algorithm over the stream is not enough for exact identification of frequent items.

MGCHH is a single-pass algorithm solving Problem 2 as follows. Basically, the algorithm estimates the frequencies of the items occurring in the stream along the primary dimension using a set of counters (primary counters) updated as in the MG algorithm; moreover, another set of counters (secondary counters) is associated to each primary counter to keep track of the frequent items occurring along the secondary dimension and correlated to the primary item. In other words, for each distinct item d along the primary dimension, the algorithm maintains its frequency estimate $\hat{f}_d$ and an embedded MG set of secondary counters related to the sub-stream σ_d induced by d: $\sigma_d = \{y | (d, y) \in \sigma\}$. The embedded secondary counters are a set of pairs $(s, \hat{f}_{d,s})$, where s is an item occurring in σ_d and $\hat{f}_{d,s}$ estimates the frequency of the tuple (d, s) in σ. Alternatively, $\hat{f}_{d,s}$ can be seen as the frequency estimate of s in σ_d. Therefore, the MGCHH actions on $\hat{f}_d$ are driven by the item d occurring in σ whilst the actions on $\hat{f}_{d,s}$ depend instead on the item s occurring in σ_d.

The MGCHH main data structure is a table H, a set of tuples of the form $(d, \hat{f}_d, H_d)$, where d is an item along the primary dimension, $\hat{f}_d$ is its estimated frequency and H_d is a secondary table which stores the secondary items occurring with d. The H_d table maintains a set of counters (item, frequency) monitoring items occurring along the secondary dimension; for each secondary item s, its frequency is determined with regard to σ_d and denoted by $\hat{f}_{d,s}$.

Let s_1 and s_2 be respectively the maximum number of primary counters in H and the maximum number of secondary counters in each H_d. In MGCHH s_1 and s_2 depend on the parameters $\phi_1, \phi_2, \epsilon_1, \epsilon_2$, and are set at the beginning of the algorithm.

When the algorithm starts, its data structures are initialized. The number of counters s_1 (for tracking the primary items) and s_2 (for tracking correlated items) is selected in order to solve the ACHH problem within the ϵ_1 and ϵ_2 error bounds; moreover, s_1 and s_2 are also chosen to minimize the total space required, as discussed in Lahiri et al. (2016). In practice, letting $\alpha = \frac{1+\phi_2}{\phi_1-\epsilon_1}$, if $\epsilon_1 \geq \frac{\epsilon_2}{2\alpha}$, then MGCHH initializes $s_1 = \frac{2\alpha}{\epsilon_2}$ counters in order to keep track of the primary frequent items and $s_2 = \frac{2}{\epsilon_2}$ counters to track correlated frequent items; otherwise, if $\epsilon_1 < \frac{\epsilon_2}{2\alpha}$, then MGCHH sets $s_1 = \frac{1}{\epsilon_1}$ and $s_2 = \frac{1}{\epsilon_2 - \alpha\epsilon_1}$.

Upon receiving a tuple (x, y) from the stream, the data structures are updated as needed. Depending on the tuple (x, y), the update process works as follows:

1. If x is in H (i.e., it is already monitored), and y is in H_x as well, then both $\hat{f}_x$ and $\hat{f}_{x,y}$ are incremented.
2. If x is in H, but y is not in H_x (i.e., it is not monitored) and there is an available counter, then y is added to H_x and its frequency is initialized to one. If no counter is available (i.e., $|H_x| = s_2$), then each counter in H_x is decremented by one. If the frequency of any monitored item goes to zero, the item is evicted from H_x. After this operation, the size of H_x is such that $|H_x| \leq s_2$.
3. If x is not in H, and $|H| < s_1$ then a counter is created for x in H setting its frequency $\hat{f}_x$ to one, and initializing H_x with the counter $(y, 1)$. If $|H| = s_1$, then for each monitored item $d \in H$, its frequency $\hat{f}_d$ is decremented by one; if this decrement causes $\hat{f}_d$ to become zero, then the counter monitoring d is discarded and removed from H. Otherwise, an arbitrary item s is randomly selected from H_d such that $\hat{f}_{d,s} > 0$ and $\hat{f}_{d,s}$ is decremented by one. If $\hat{f}_{d,s}$ goes to zero, the item s is discarded from H_d. This further decrement guarantees that the sum of the $\hat{f}_{d,s}$ frequencies within H_d is less than or equal to $\hat{f}_d$.

Finally, in order to report the CHHs in the stream, a query can be posed to the data structures as follows. For each primary item $d \in H$, if $\hat{f}_d \geq (\phi_1 - \frac{1}{s_1})N$ then the algorithm searches for the secondary items $s \in H_d$ such that $\hat{f}_{d,s} \geq (\phi_2 - \frac{1}{s_2})\hat{f}_d - \frac{N}{s_1}$ and returns the corresponding (d, s) tuples.

4 A Space Saving based algorithm

Our cascading space saving correlated heavy hitters (CSSCHH) algorithm exploits the basic ideas of the Space Saving algorithm (Metwally et al. 2006), combining two Space Saving stream summaries for tracking the primary item frequencies and the tuple frequencies. We refer to our algorithm as Cascading Space Saving since it is based on the use of two distinct and independent applications of Space Saving. Therefore, we use two independent Space Saving stream summaries as data structures. The first, denoted by $\mathcal{S}^p$, and referred to as the primary stream summary, monitors a subset of primary items which appears in the stream through the use of k_1 distinct counters. The second, denoted by $\mathcal{S}^t$, includes k_2 counters and monitors a subset of the tuples which appear in the stream.

The counters are updated in order to accurately estimate the items' frequencies and a lightweight data structure is exploited to keep the elements sorted by their estimated frequencies. A detailed description of the Space Saving algorithm is given in Metwally et al. (2006). Here, we briefly recall how Space Saving works and its main properties.

A stream summary $\mathcal{S}$ is a data structure used to monitor k distinct items and includes k counters. We denote with c_j the jth counter and by $c_j.i$ and $c_j.f$ respectively the item monitored by the jth counter and its corresponding estimated frequency. When processing an item which is already monitored by a counter, its estimated frequency is incremented by one. When processing an item which is not monitored, there are two possibilities. If a counter is available, it will be in charge of monitoring the item and its estimated frequency is set to one. Otherwise, if all of the counters are already occupied (their frequencies are different from zero), the counter storing the item with minimum frequency is incremented by one. Then the monitored item is replaced by the new item. This is because an item which is not monitored can not have a frequency greater than the minimal frequency. The complexity of the Space Saving update procedure is $O(1)$ in the worst case, as proved by its authors.

Let N be the length of the input stream, $\sum_{c_i \in \mathcal{S}} c_i.f$ the sum of the counters in $\mathcal{S}$, $k = |\mathcal{S}|$ the number of counters in $\mathcal{S}$, f_v the exact frequency of an item v, $\hat{f}_v$ its estimated frequency and $\hat{f}^{min}$ the minimum frequency in $\mathcal{S}$. Then, the following relations hold for Space Saving:

$$\sum_{c_i \in \mathcal{S}} c_i.f = N,$$

$$\hat{f}_v - f_v \leq \hat{f}^{min} \leq \frac{N}{k}. \tag{7}$$

Our CSSCHH algorithm starts by initializing the $\mathcal{S}^p$ primary stream summary data structure allocating k_1 counters and the correlated $\mathcal{S}^t$ stream summary allocating k_2 counters. We shall explain in Sect. 5 how exactly the values of k_1 and k_2 are derived. Algorithm 1 presents the pseudocode related to the initialization phase of CSSCHH.

Algorithm 1 CSSCHH Init

```
Require: Threshold for primary items $\phi_1$; threshold for correlated items $\phi_2$; tolerance for primary items
    $\epsilon_1$; tolerance for correlated items $\epsilon_2$.
Ensure: Properly initialized $\mathcal{S}^p$ and $\mathcal{S}^t$ stream summaries
1: procedure CSSCHH- INIT($\phi_1, \phi_2, \epsilon_1, \epsilon_2$)
2:     $\beta \leftarrow \frac{1}{\epsilon_2 \phi_1}$
3:     $\gamma \leftarrow \frac{\epsilon_2 + \phi_2}{\epsilon_2 \phi_1}$
4:     $k_1 \leftarrow \max \left\{ \frac{1}{\epsilon_1}, \gamma + \sqrt{\beta\gamma} \right\}$
5:     $k_2 \leftarrow \beta \frac{k_1}{k_1 - \gamma}$
6:     Allocate $k_1$ counters for $\mathcal{S}^p$
7:     Allocate $k_2$ counters for $\mathcal{S}^t$
8:     return $\mathcal{S}^p$ and $\mathcal{S}^t$
9: end procedure
```

In CSSCHH algorithm, a tuple (x, y) is processed by updating two stream summaries, $\mathcal{S}^p$ and $\mathcal{S}^t$. The primary item x is used to update the primary stream summary

$\mathcal{S}^p$; the tuple (x, y) is also considered as a single item and it is used to update the correlated stream summary $\mathcal{S}^t$. Since for each tuple in the stream both stream summaries are updated by means of the Space Saving update procedure, we inherit its properties. Let f_x and $\hat{f}_x$ denote the exact and estimated frequency of the primary item x, and let f_{xy} and $\hat{f}_{xy}$ denote the exact and estimated frequency of the tuple (x, y); moreover, denoting by c_i^p and c_i^t the ith counter in the primary and in the correlated stream summary, and denoting by $\hat{f}^{p^{min}}$ and $\hat{f}^{t^{min}}$ the minimum frequency in the primary and correlated stream summary, the following relations hold:

$$\sum_{c_i^p \in \mathcal{S}^p} c_i^p.f = N, \qquad \hat{f}_x - f_x \le \hat{f}^{p^{min}} \le \frac{N}{k_1}, \tag{8}$$

$$\sum_{c_i^t \in \mathcal{S}^t} c_i^t.f = N, \qquad \hat{f}_{xy} - f_{xy} \le \hat{f}^{t^{min}} \le \frac{N}{k_2}. \tag{9}$$

The update procedure of CSSCHH is presented in Algorithm 2.

Algorithm 2 CSSCHH Update

```
Require: $x, y$, the items of a tuple.
Ensure: Update of $\mathcal{S}^p$ and $\mathcal{S}^t$ stream summaries.
1: procedure CSSCHH- UPDATE($\mathcal{S}^p, \mathcal{S}^t, x, y$)
2:     SPACESAVINGUPDATE($\mathcal{S}^p, x$)
3:     SPACESAVINGUPDATE($\mathcal{S}^t, (x, y)$)
4: end procedure
```

In order to retrieve the correlated heavy hitters, a query is posed to both stream summaries. The query procedure internally uses two lists, F and C. The former stores primary items and their estimated frequencies $(r, \hat{f}_r)$. The latter stores CHHs $(r, s, \hat{f}_{rs})$ in which r is a primary frequent item, s the correlated frequent item candidate and $\hat{f}_{rs}$ the estimated frequency of the tuple (r, s).

The query algorithm inspects all of the k_1 counters in the $\mathcal{S}^p$ stream summary. If the frequency of the monitored item is greater than the selection criterion (i.e., $c_j^p.f > \phi_1 N$), then we add the monitored item $r = c_j^p.i$ and its estimated frequency $\hat{f}_r = c_j^p.f$ to F.

The algorithm inspects now all of the k_2 counters of the $\mathcal{S}^t$ stream summary. The monitored items in $\mathcal{S}^t$ are the tuples (r, s). We check if the primary item r is a primary frequent item candidate (i.e., if $r \in F$); if this condition is true and the tuple estimated frequency is greater than the selection criterion (i.e., $c_j^t.f > \phi_2(\hat{f}_r - \frac{N}{k_1})$), then the triplet $(r, s, \hat{f}_{rs})$ is added to C. The Query procedure is presented as Algorithm 3.

Algorithm 3 CSSCHH Query

```
Require: $\mathcal{S}^p$ and $\mathcal{S}^t$ stream summaries.
Ensure: Set of correlated frequent items $C$
1: procedure CSSCHH- QUERY($\mathcal{S}^p$, $\mathcal{S}^t$)
2:     $F \leftarrow \emptyset$
3:     for each $c_j^p \in \mathcal{S}^p$ do
4:         $r \leftarrow c_j^p.i$; $\hat{f}_r \leftarrow c_j^p.f$
5:         if $\hat{f}_r > \phi_1 N$ then
6:             $F \leftarrow F \cup \{(r, \hat{f}_r)\}$
7:         end if
8:     end for
9:     for each $c_j^t \in \mathcal{S}^t$ do
10:        $(r, s) \leftarrow c_j^t.i$; $\hat{f}_{rs} \leftarrow c_j^p.f$
11:        if $r \in F \wedge (\hat{f}_{rs} > \phi_2(\hat{f}_r - \frac{N}{k_1}))$ then
12:            $C \leftarrow C \cup \{(r, s, \hat{f}_{rs})\}$
13:        end if
14:    end for
15:    return $C$
16: end procedure
```

5 Correctness

We are going to formally prove the correctness of our algorithm. The main results of this section are the following two theorems.

Theorem 1 *The CSSCHH algorithm reports all of the primary items x whose exact frequency f_x is greater than the threshold, i.e., $f_x > \phi_1 N$ and no items whose exact frequency is such that $f_x \leq (\phi_1 - \frac{1}{k_1})N$.*

Proof The algorithm determines all of the primary frequent candidates through the selection criterion $\hat{f}_x > \phi_1 N$. Since the stream summary provides an overestimation of the frequency $\hat{f}_x \geq f_x$, if the exact frequency of an item is greater than the threshold, its estimated frequency will be greater as well: $\hat{f}_x \geq f_x > \phi_1 N$, hence the item will be selected and this proves the first part of the theorem.

The second part of the theorem states that, given an item x, if its exact frequency is $f_x \leq (\phi_1 - \frac{1}{k_1})N$, which can be rewritten as

$$f_x + \frac{1}{k_1}N \leq \phi_1 N, \tag{10}$$

then the item will not be selected; hence we must prove that its estimated frequency is less than $\hat{f}_x \leq \phi_1 N$. By using the Space Saving properties we know that the estimate error provided by the stream summary $\mathcal{S}^p$ is bounded by $\frac{N}{k_1}$:

$$\hat{f}_x - f_x \leq \frac{1}{k_1}N \Rightarrow \hat{f}_x \leq f_x + \frac{1}{k_1}N, \tag{11}$$

hence, by using Eq. (10) we have

$$\hat{f}_x \leq f_x + \frac{1}{k_1}N \leq \phi_1 N, \tag{12}$$

which proves the theorem. Moreover, this theorem also implies that for all of the primary frequent candidates it holds that:

$$f_x > \left(\phi_1 - \frac{1}{k_1}\right) N. \tag{13}$$

□

Theorem 2 *All of the tuples (x, y) with the item x reported as primary frequent candidate and with exact frequency f_{xy} greater than the threshold ($f_{xy} > \phi_2 f_x$) are reported as correlated heavy hitter candidate. No tuple with a primary item x reported as frequent primary candidate and with exact frequency less than $f_{xy} \leq (\phi_2 - \frac{k_2\phi_2+k_1}{k_2(k_1\phi_1-1)}) f_x$ is reported as correlated heavy hitter candidate.*

Proof The algorithm determines a correlated heavy hitter candidate (x, y) only if the primary item x as been reported as primary frequent item candidate and if its estimated frequency is greater than the selection criterion $\hat{f}_{xy} > \phi_2(\hat{f}_x - \frac{N}{k_1})$. We must prove that those tuples whose exact frequency is greater than the threshold $f_{xy} > \phi_2 f_x$ are reported by the algorithm and hence their estimated frequency is greater than the selection criterion $\hat{f}_{xy} > \phi_2(\hat{f}_x - \frac{N}{k_1})$. If $f_{xy} > \phi_2 f_x$ is true, then $\hat{f}_{xy} > \phi_2 f_x$ is also true since the stream summary $\mathcal{S}^t$ provides an overestimation of the tuple frequency. Now, since x is reported as primary frequent candidate, from Theorem 1 we have that $f_x \geq \hat{f}_x - \frac{N}{k_1}$ and it holds that

$$\hat{f}_{xy} > \phi_2 f_x \geq \phi_2 \left(\hat{f}_x - \frac{N}{k_1}\right). \tag{14}$$

Since the frequency estimate is greater than the selection criterion, the tuple will be reported and this proves the first part of the theorem.

The second part of the theorem states that those items with an exact frequency such that $f_{xy} \leq (\phi_2 - \frac{k_2\phi_2+k_1}{k_2(k_1\phi_1-1)}) f_x$ will not be reported, hence we must prove that their estimate frequency is less than or equal to the selection criterion i.e., $\hat{f}_{xy} \leq \phi_2(\hat{f}_x - \frac{N}{k_1})$. Since the algorithm first filters the tuples retaining only the ones whose primary item belongs to the primary frequent item candidates, from Theorem 1 it follows that for all of the primary frequent candidates it holds that:

$$f_x > \left(\phi_1 - \frac{1}{k_1}\right) N \Rightarrow f_x > \frac{\phi_1 k_1 - 1}{k_1} N. \tag{15}$$

To prove the theorem we start assuming the exact frequency is such that

$$f_{xy} \leq \left(\phi_2 - \frac{k_2\phi_2 + k_1}{k_2(k_1\phi_1 - 1)}\right) f_x. \tag{16}$$

Due to the $\mathcal{S}^t$ stream summary properties, the error of the tuple frequency estimate is bounded by $\frac{N}{k_2}$, so that $\hat{f}_{xy} - \frac{N}{k_2} \leq f_{xy}$ and it holds that:

$$\hat{f}_{xy} - \frac{N}{k_2} \leq f_{xy} \leq \phi_2 f_x - \frac{k_2\phi_2 + k_1}{k_2(k_1\phi_1 - 1)} f_x. \tag{17}$$

Since the frequency estimate is an overestimation (i.e., $f_x \leq \hat{f}_x$),

$$\hat{f}_{xy} - \frac{N}{k_2} \leq \phi_2 \hat{f}_x - \frac{k_2\phi_2 + k_1}{k_2(k_1\phi_1 - 1)} f_x; \tag{18}$$

using Eq. (15) we can write:

$$\begin{aligned}
\hat{f}_{xy} - \frac{N}{k_2} &\leq \phi_2 \hat{f}_x - \frac{k_2\phi_2 + k_1}{k_2(k_1\phi_1 - 1)} \frac{\phi_1 k_1 - 1}{k_1} N, \\
\hat{f}_{xy} - \frac{N}{k_2} &\leq \phi_2 \hat{f}_x - \frac{k_2\phi_2 + k_1}{k_2 k_1} N, \\
\hat{f}_{xy} - \frac{N}{k_2} &\leq \phi_2 \hat{f}_x - \frac{\phi_2}{k_1} N - \frac{N}{k_2}, \\
\hat{f}_{xy} &\leq \phi_2 \left(\hat{f}_x - \frac{N}{k_1} \right).
\end{aligned} \tag{19}$$

Taking into account that the estimated frequency is less than the selection criterion, the corresponding tuple will not be reported, proving the theorem. Moreover, this theorem also implies that for all of the correlated heavy hitter candidates it holds that:

$$f_{xy} > \left(\phi_2 - \frac{k_2\phi_2 + k_1}{k_2(k_1\phi_1 - 1)} \right) f_x. \tag{20}$$

□

Theorems 1 and 2 can be used for tuning the stream summary sizes k_1 and k_2. The ACHH problem poses a constraint about the tolerance ϵ_1 on the number of primary frequent false positives and a corresponding constraint about the tolerance ϵ_2 on the number of correlated heavy hitter false positives. Therefore, k_1 and k_2 are also subject to the following constraints:

C1 The ACHH problem does not admit false negatives, hence all of the real primary frequent items must be reported. The maximum number of primary frequent items is $\frac{1}{\phi_1}$ hence

$$k_1 \geq \frac{1}{\phi_1}. \tag{21}$$

C2 The ACHH problem allows primary false positives only with a tolerance given by $\epsilon_1 N$, hence for all of the primary frequent candidates it must be $f_x \geq (\phi_1 - \epsilon_1)N$.

Using Theorem 1 we need to impose:

$$\frac{1}{k_1} \leq \epsilon_1 \Rightarrow k_1 \geq \frac{1}{\epsilon_1}. \tag{22}$$

C3 The ACHH problem does not admit false negatives, hence all of the real correlated heavy hitters must be reported. The maximum number of correlated heavy hitters is $\frac{1}{\phi_1\phi_2}$, hence

$$k_2 \geq \frac{1}{\phi_1\phi_2}. \tag{23}$$

C4 The ACHH problem allows correlated false positives only with a tolerance given by $\epsilon_2 f_x$, hence for all of the correlated heavy hitter candidates it must be $f_{xy} \geq (\phi_2 - \epsilon_2) f_x$. Using Theorem 2 we need to impose:

$$\frac{k_2\phi_2 + k_1}{k_2(k_1\phi_1 - 1)} \leq \epsilon_2. \tag{24}$$

Solving Eq. (24) w.r.t. k_2, we have

$$k_2(\epsilon_2 k_1\phi_1 - \epsilon_2 - \phi_2) \geq k_1 \tag{25}$$

Since k_2 and k_1 represent the sizes of the stream summaries, both are positives integers. Therefore, a solution to Eq. (25) is feasible only when the left hand side term is positive. To summarize, the current constraint is expressed by the following equations:

$$\begin{aligned} \epsilon_2 k_1\phi_1 - \epsilon_2 - \phi_2 > 0 \Rightarrow & k_1 > \frac{\epsilon_2 + \phi_2}{\epsilon_2\phi_1}, \\ & k_2 \geq \frac{k_1}{\epsilon_2 k_1\phi_1 - \epsilon_2 - \phi_2}. \end{aligned} \tag{26}$$

Constraint C1 can be ignored since it is already embodied by constraint C2; indeed, the problem requires that $\epsilon_1 < \phi_1$. Constraint C3 can be ignored as well since it can be easily proved that $\frac{k_1}{\epsilon_2 k_1\phi_1 - \epsilon_2 - \phi_2} > \frac{1}{\phi_1\phi_2}$ for any value of the input parameters.

Introducing the terms $\beta = \frac{1}{\epsilon_2\phi_1}$ and $\gamma = \frac{\epsilon_2 + \phi_2}{\epsilon_2\phi_1}$, we have to determine k_1 and k_2 such that their sum is minimized subject to the following constraints:

$$\begin{aligned} & k_1 \geq \frac{1}{\epsilon_1}, \\ & k_1 > \gamma, \\ & k_2 \geq \beta\frac{k_1}{k_1 - \gamma}. \end{aligned} \tag{27}$$

The optimal values for k_1 and k_2 are obtained by solving a constrained minimization problem. Introducing a variable substitution $r = k_1 + k_2$ and $s = k_1$ the constrained minimization problem is formulated as follows:

$$\begin{array}{ll} \text{minimize} & r \\ \text{subject to} & s \geq \frac{1}{\epsilon_1}, \\ & s > \gamma, \\ & r \geq s + \beta \frac{s}{s-\gamma}. \end{array} \tag{28}$$

From the last constraint we can deduce that the minimum value of r must belong to the curve $r = s + \beta \frac{s}{s-\gamma}$ which attains its minimum for $s = \gamma + \sqrt{\beta\gamma}$. Taking into account both constraints on s, the minimum is reached when

$$\begin{aligned} s &= \max \left\{ \frac{1}{\epsilon_1}, \gamma + \sqrt{\beta\gamma} \right\}, \\ r &= s + \beta \frac{s}{s-\gamma}. \end{aligned} \tag{29}$$

Therefore, the corresponding k_1 and k_2 values are

$$\begin{aligned} k_1 &= \max \left\{ \frac{1}{\epsilon_1}, \gamma + \sqrt{\beta\gamma} \right\}, \\ k_2 &= \beta \frac{k_1}{k_1 - \gamma}. \end{aligned} \tag{30}$$

These are the values set by Algorithm 1 in order to initialize the CSSCHH stream summaries data structure by using the minimum number of counters, and, consequently, of space required to solve the ACHH problem.

6 Space and time complexity

In this section, we analyze the worst case time and space complexity of our algorithm. Regarding the initialization phase (Algorithm 1), the worst case complexity is clearly $O(1)$ since initialization consists of just a few assignments, each one requiring at most $O(1)$ time.

The update procedure (Algorithm 2) requires constant time as well. Indeed, each one of the two calls to SPACESAVINGUPDATE requires at most $O(1)$.

Finally, a query (Algorithm 3) requires time at most $O(k_1 + k_2)$. Indeed, the first part of the query is just a linear scan of the k_1 counters related to the primary stream summary $\mathcal{S}^p$, in which we check, for each counter, if the corresponding monitored item's frequency exceed the selection criterion. When the check succeeds, the item and its estimated frequency are added to the hash table F. Since checking the condition can be done in constant time, this part of the query requires time at most $O(k_1)$.

Next, we inspect the correlated stream summary $\mathcal{S}^t$. Again, this is just a linear scan. For each counter, we retrieve the stored tuple (r, s), and its estimated frequency. Then, we search for the primary item r in F; if the item belongs to F and if the condition required for the tuple (r, s) to be considered a CHH is verified, then we update the list C holding the CHHs that shall be returned to the user. Searching in F requires constant time (since F is implemented as an hash table), and verifying the

CHH condition requires constant time as well, the second part of the query requires time at most $O(k_2)$. It follows that, overall, the query requires in the worst case time at most $O(k_1 + k_2)$.

Regarding the space complexity, it is clear that the total space required is at most $O(k_1 + k_2)$, since we use k_1 counters for $\mathcal{S}^p$ and k_2 counters for $\mathcal{S}^t$. More specifically, in order to express the space complexity with regard to the input parameters, we distinguish two cases as in Eq. (30). When $1/\epsilon_1 \leq \gamma + \sqrt{\beta\gamma}$, we have

$$k_1 + k_2 = \frac{2\sqrt{\epsilon_2 + \phi_2} + \epsilon_2 + \phi_2 + 1}{\epsilon_2 \phi_1} = O\left(\frac{1}{\epsilon_2 \phi_1}\right); \tag{31}$$

otherwise, for $1/\epsilon_1 > \gamma + \sqrt{\beta\gamma}$, it holds that

$$k_1 + k_2 = \frac{1}{\epsilon_1} + \frac{1}{\epsilon_1 \epsilon_2 \phi_1 \left(\frac{1}{\epsilon_1} - \frac{\epsilon_2 + \phi_2}{\epsilon_2 \phi_1}\right)} < \frac{1}{\epsilon_1} + \frac{1}{\epsilon_1 \sqrt{\epsilon_2 + \phi_2}} = O\left(\frac{1}{\epsilon_1 \sqrt{\epsilon_2}}\right). \tag{32}$$

7 Theoretical comparison

In this section we compare MGCHH and our algorithm CSSCHH from a theoretical perspective, before presenting the results of the experiments that we have carried out. We begin by comparing the space complexity and how many counters are required by both algorithms to guarantee their error bounds.

Let $\alpha = \frac{1+\phi_2}{\phi_1 - \epsilon_1}$, then, if $\epsilon_1 \geq \frac{\epsilon_2}{2\alpha}$, MGCHH requires $s_1 = \frac{2\alpha}{\epsilon_2}$ counters in order to keep track of the primary frequent items, and $s_2 = \frac{2}{\epsilon_2}$ counters to track correlated frequent items; otherwise (if $\epsilon_1 < \frac{\epsilon_2}{2\alpha}$), $s_1 = \frac{1}{\epsilon_1}$ and $s_2 = \frac{1}{\epsilon_2 - \alpha\epsilon_1}$.

In the former case the space complexity of MGCHH is $O(\frac{1}{(\phi_1 - \epsilon_1)\epsilon_2^2})$, and in the latter case its space complexity is $O(\frac{1}{\epsilon_1 \epsilon_2})$ (Lahiri et al. 2016).

MGCHH requires a total of $s_1 + (s_1 s_2)$ counters, since the algorithm consists of a nested application of the Frequent algorithm: besides the s_1 counters for primary frequent items, there are $s_1 s_2$ counters for correlated items. Indeed, for each primary counter there is an entire summary consisting of s_2 counters.

Our CSSCHH algorithm requires $k_1 = \max\left\{\frac{1}{\epsilon_1}, \gamma + \sqrt{\beta\gamma}\right\}$ counters for the primary frequent items and $k_2 = \beta \frac{k_1}{k_1 - \gamma}$ for the correlated frequent items, where $\beta = \frac{1}{\epsilon_2 \phi_1}$ and $\gamma = \frac{\epsilon_2 + \phi_2}{\epsilon_2 \phi_1}$.

As shown in the previous section, the space complexity of our algorithm is $O\left(\frac{1}{\epsilon_2 \phi_1}\right)$ when $1/\epsilon_1 \leq \gamma + \sqrt{\beta\gamma}$, and it is $O\left(\frac{1}{\epsilon_1 \sqrt{\epsilon_2}}\right)$ when $1/\epsilon_1 > \gamma + \sqrt{\beta\gamma}$. It is immediate verifying that our algorithm requires asymptotically less space than MGCHH. From a practical perspective, it's worth noting here that MGCHH is also subject to the constraint $\epsilon_1 < \phi_1/2$, whilst our algorithm is not.

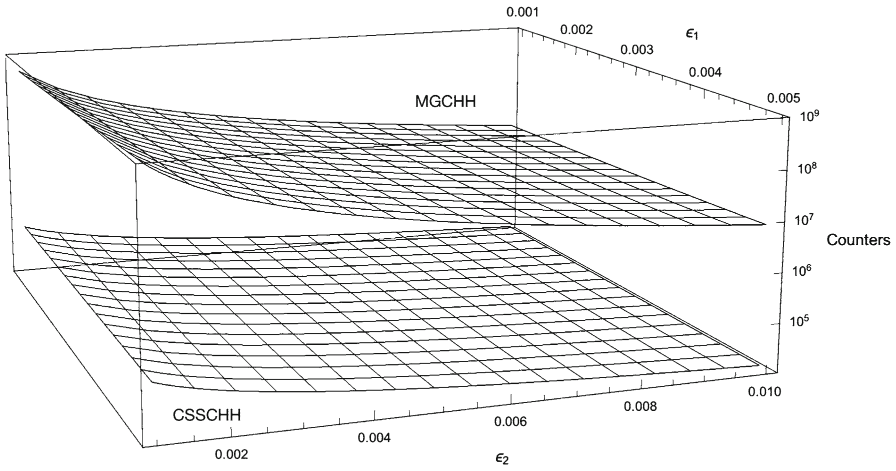

Fig. 1 Counters required by MGCHH and CSSCHH

Figure 1 depicts the number of counters required by MGCHH and CSSCHH (using a logarithmic scale on the z axis) fixing $\phi_1 = 0.01$, $\phi_2 = 0.01$ and letting ϵ_1 varying up to $\phi_1/2$, ϵ_2 varying up to ϕ_2.

As shown by the two surfaces, CSSCHH requires several orders of magnitude less counters than MGCHH. Moreover, we expect MGCHH to be slower with regard to CSSCHH. Indeed, every update step, in which the incoming primary stream item is not monitored and all of the s_1 counters are full, requires not only decrementing all of the s_1 counters (as in the Frequent algorithm), but also, for each one of them, MGCHH must randomly select a correlated item and decrease its frequency as well. Now, we consider and discuss the accuracy of MGCHH. Being based on the Frequent algorithm, we know that, overall, its accuracy shall be lower than the accuracy provided by CSSCHH which is, instead, based on the Space Saving algorithm. Indeed, it is well known that Space Saving is more accurate than Frequent (Cormode and Hadjieleftheriou 2009; Manerikar and Palpanas 2009). In the next section we shall experimentally see how much faster and accurate CSSCHH is with regard to MGCHH.

8 Experimental results

In order to compare and evaluate our CSSCHH algorithm against MGCHH we have implemented them in C++. The source code has been compiled using the clang c++ compiler v8.0 on Mac OS X v10.12 with the following flags: -Os -std=c++14. We recall here that, on Mac OS X, the optimization flag -Os provides better optimization than the -O3 flag and is the standard for building the release build of an application. The tests have been carried out on a machine equipped with 16 GB of RAM and a 3.2 GHz quad-core Intel Core i5 processor with 6 MB of cache level 3. The source code is freely available for inspection and for reproducibility of results contacting the authors by email.

Table 1 Experiments carried out (synthetic datasets)

Items (n, millions)	Skew (ρ)	Space used (MBytes)	Threshold (ϕ_1)
$\{5, 50, 500, 1000\}$	$\{1, 1.4, 1.8, 2.2\}$	$\{1.009, 3.941, 15.573, 61.908\}$	$\{0.1, 0.01, 0.001\}$

Table 2 Counters corresponding to the space used (synthetic datasets)

Space (MB)	$k_1 = s_1$	k_2	s_2
1.009	4200	63,000	20
3.941	8400	252,000	40
15.573	16,800	1,008,000	80
61.908	33,600	4,032,000	160

The items in the synthetic datasets used in our experiments are distributed according to the Zipf distribution. In each one of the experiments, the execution have been repeated 10 times using a different seed for the pseudo-random number generator used for creating the input data stream (using the same seeds in the corresponding executions of different algorithms). For each input distribution generated, the results have been averaged over all of the runs. The input items are 32 bits unsigned integers. Table 1 reports all of the experiments on synthetic datasets that have been carried out.

We have also experimented using a real dataset, namely Worldcup'98. This dataset is publicly available[1] and stores information related to the requests made to the World Cup web site during the 1998 tournament. For each request, the dataset includes a ClientID (which is a unique integer identifier for the client that issued the request) and an ObjectID (again, a unique integer identifier for the requested URL). In this experiment we determine correlated heavy hitters between ClientID and ObjectID pairs, treating ClientID as the primary items, and ObjectID as the secondary item. Owing to the huge size of the full dataset, we used a subset of the available data, i.e., the data from day 41 to day 46 of the competition. Table 3 reports the statistical characteristics of the real dataset.

Regarding synthetic datasets, we vary the input data stream size (n, in millions), the skew of the zipfian distribution (ρ), the total space used (measured in MegaBytes) and the ϕ_1 and ϕ_2 support thresholds. The value of the remaining parameters are fixed and reported in each individual plot. Regarding the total space used, it is worth noting here that, for MGCHH, a counter (related to either a primary or a correlated item) requires 4 bytes to store the monitored item (an unsigned int) and 8 bytes to store its estimated frequency (a long int), for a total of 12 bytes. On the other hand, for CSSCHH a counter related to primary items requires 12 bytes as well, but a counter monitoring a tuple (x, y) requires instead 4 bytes for x, 4 bytes for y and 8 bytes to store the estimated frequency, for a total of 16 bytes. Therefore, the total space used by MGCHH is $12(s_1 + s_1 s_2)$ bytes, whilst the total space used by CSSCHH is

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

Table 3 Statistical characteristics of the real dataset (Worldcup'98)

	Primary	Secondary
Count	104,271,758	104,271,758
Distinct items	539,464	21,605
Min	1	0
Max	1,375,004	40,317
Mean	549,924	9774.51
Median	578,870	887
SD	427,859	11,094.8
Skewness	0.146792	0.449196

Table 4 Counters corresponding to the space used (Worldcup'98)

Space (MB)	$k_1 = s_1$	k_2	s_2
11.558	10,000	750,000	100
36.048	25,000	2,343,750	125
86.402	50,000	5,625,000	150
230.026	100,000	15,000,000	200

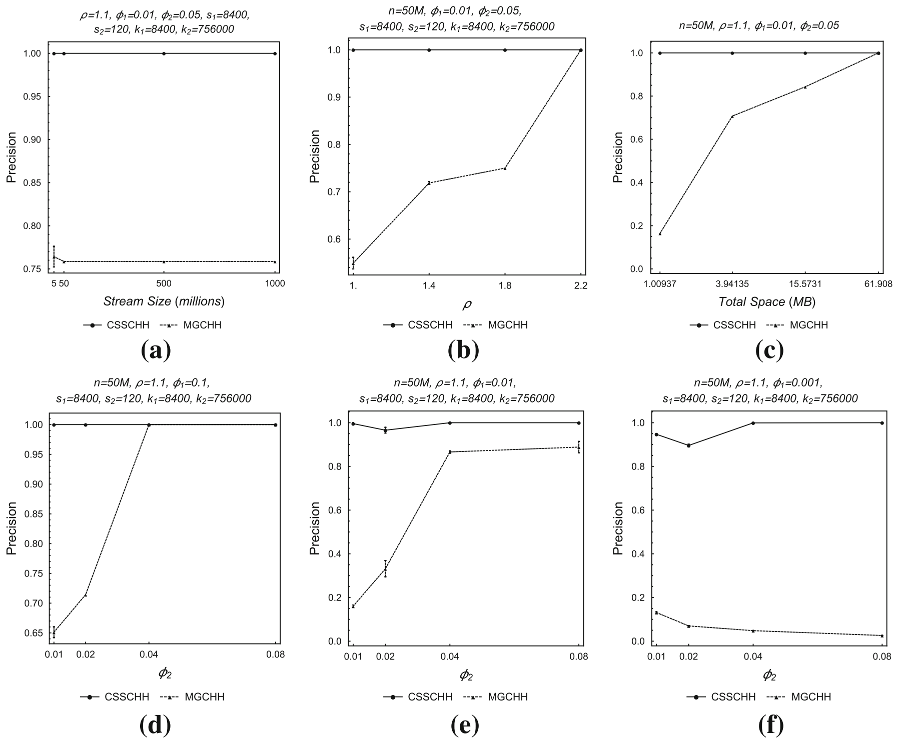

Fig. 2 Synthetic datasets: Precision (mean and confidence interval), **a** varying n, **b** varying ρ, **c** varying the space used, **d** $\phi_1 = 0.1$, **e** $\phi_1 = 0.01$, **f** $\phi_1 = 0.001$

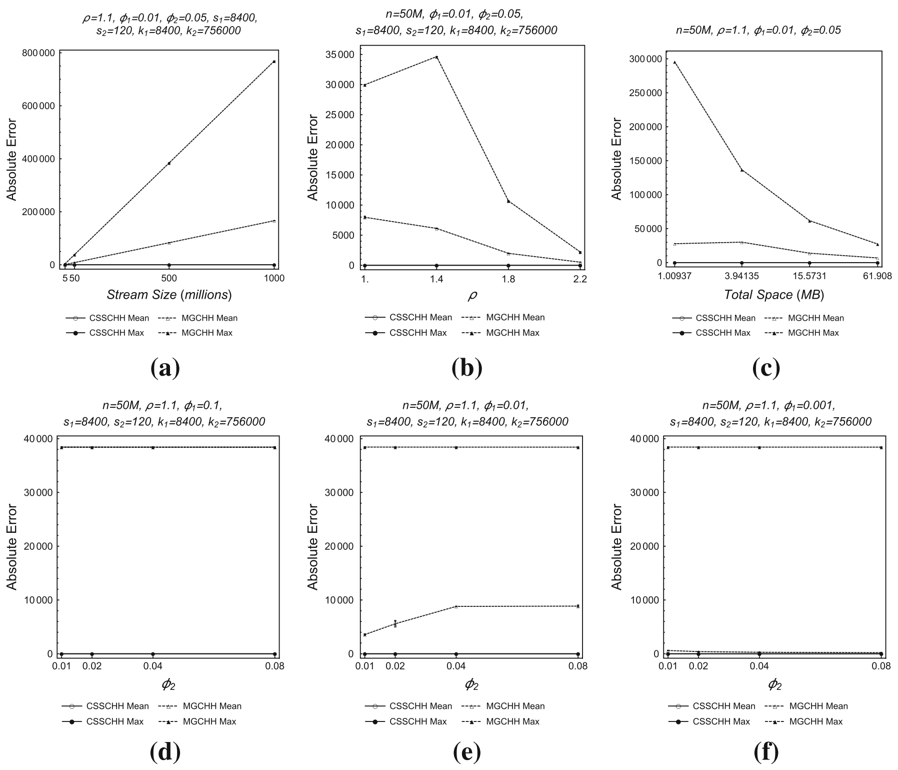

Fig. 3 Synthetic datasets: Absolute error (mean and confidence interval), **a** varying n, **b** varying ρ, **c** varying the space used, **d** $\phi_1 = 0.1$, **e** $\phi_1 = 0.01$, **f** $\phi_1 = 0.001$

$12k_1 + 16k_2$ bytes. The test related to the space used is carried out by assigning to both algorithm exactly the same space (measured in MegaBytes) in order to fairly compare both algorithms and to understand how the algorithms behave when performing under exactly the same conditions (with regard to the space used). Therefore, the allocated space determines the number of counters to be used correspondingly by the algorithms. In all of the other tests, we preserved this property choosing s_1, s_2, k_1 and k_2 such that $12k_1 + 16k_2 = 12(s_1 + s_1 s_2)$ and $k_1 = s_1$. Table 2 reports the counters corresponding to the space used in the experiments.

We begin our analysis discussing the results for synthetic datasets. The *recall* is the total number of true frequent items reported over the number of frequent items given by an exact algorithm. Therefore, an algorithm is correct iff its *recall* is equal to one. Since the algorithms under test are based respectively on Frequent (MGCHH) and on Space Saving (CSSCHH), their *recall* is always one if they are allowed to use enough counters. In all of the test we used a number of counters s_1 and k_1 greater than $\frac{1}{\phi_1}$ for the primary items and a number of counters for correlated items greater that $\frac{1}{\phi_2}$ for s_2 and greater than $\frac{1}{\phi_1 \phi_2}$ for k_2. This guarantees both algorithms to reach a *recall* equal to one in every case. The plots on the *recall* have not been reported in the paper.

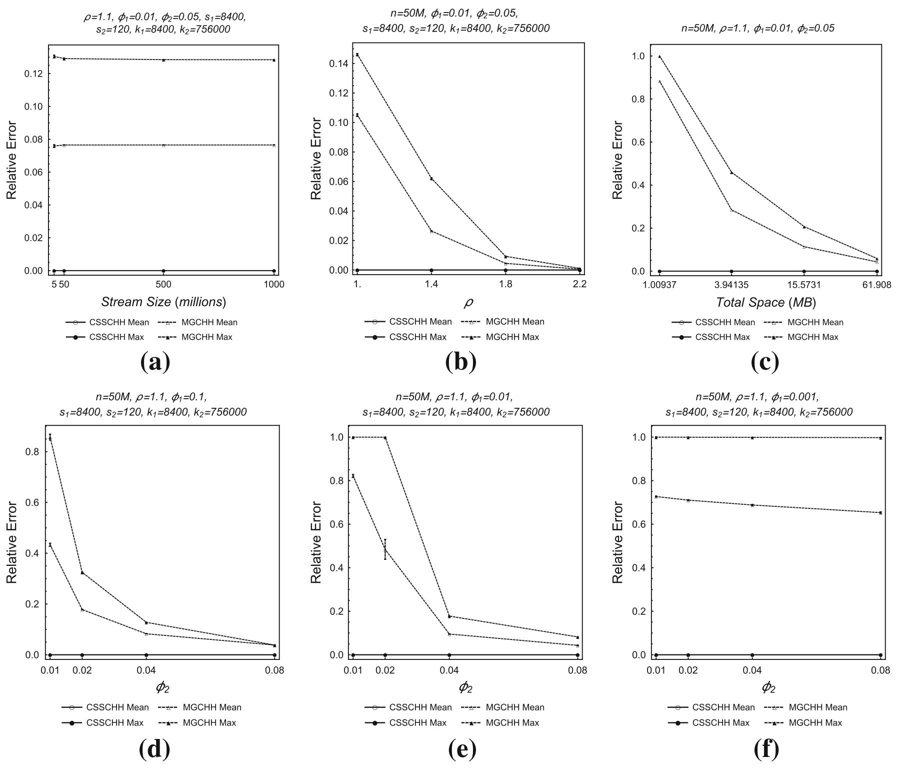

Fig. 4 Synthetic datasets: Relative error (mean and confidence interval), **a** varying n, **b** varying ρ, **c** varying the space used, **d** $\phi_1 = 0.1$, **e** $\phi_1 = 0.01$, **f** $\phi_1 = 0.001$

Next, we analyze the accuracy, beginning with the *precision* attained (with regard to the CHHs). Since *precision* is defined as the total number of true frequent items reported over the total number of items reported, this metric quantifies the number of false positives outputted by an algorithm. It follows that, from this point of view, the algorithm's *precision* should ideally be one. As shown in Fig. 2, CSSCHH clearly outperforms MGCHH with regard to the *precision*. Indeed, CSSCHH is consistently able to provide one or near one *precision* in all of the tests carried out, whilst MGCHH lags far behind.

Accuracy is also related to the absolute and relative errors on the frequency estimate committed by the algorithms. Denoting with f the exact frequency of a CHH and with $\hat{f}$ the corresponding frequency reported by an algorithm, then the absolute error is, by definition, the difference $\left|f - \hat{f}\right|$.

Similarly, the relative error is defined as $\frac{\left|f-\hat{f}\right|}{f}$ and the average relative error is derived by averaging the relative errors over all of the measured frequencies. Figures 3 and 4 depict respectively absolute and relative errors committed by the algorithms (with regard to CHHs). Each single plot reports both the maximum and the mean values attained by both algorithms. Again, it is immediate verifying that CSS-

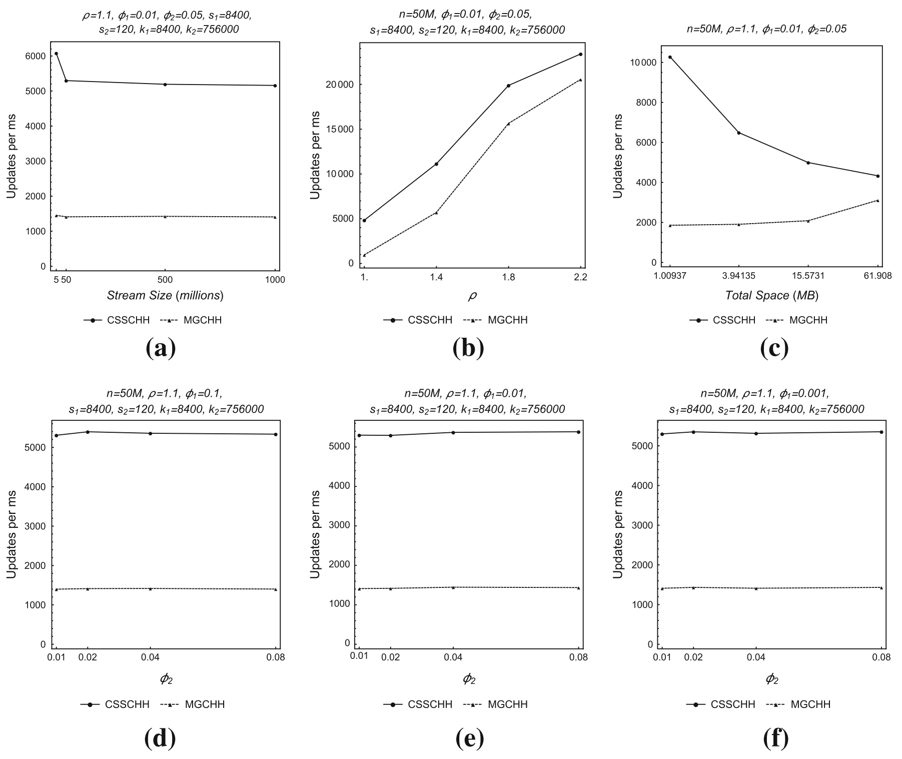

Fig. 5 Synthetic datasets: Updates/ms (mean and confidence interval), **a** varying n, **b** varying ρ, **c** varying the space used, **d** $\phi_1 = 0.1$, **e** $\phi_1 = 0.01$, **f** $\phi_1 = 0.001$

CHH is extremely accurate in all of the cases, with absolute and relative errors always equal to zero. On the contrary, MGCHH estimates are clearly affected by significant error.

Finally, we evaluated the algorithm with regard to their speed in processing stream items. Figure 5 shows the speed attained, reported as updates per millisecond. Again, CSSCHH outperforms MGCHH, being consistently much faster in all of the cases. In particular, CSSCHH is more than three times faster in all of the tests that have been carried out, except the tests in which we vary the skew of the input distribution and the space we allow to be used. Anyway, as shown by the plots, CSSCHH is always faster than MGCHH.

Next, we compare MGCHH and CSSCHH using the real dataset Worldcup'98. Figures 6 and 7 depict the results obtained respectively when varying the space allowed and the ϕ_2 threshold (fixing the ϕ_1 threshold). Table 4 reports the counters corresponding to the space used. It is worth noting here that, owing to the skewness of the dataset being really low (see Table 3), we fixed ϕ_1 to a low value (0.001) and ϕ_2 varying from 0.001 to 0.01, since otherwise no (or just a few) correlated heavy hitters exist. As shown, both algorithms exhibit the same behaviour already observed on synthetic

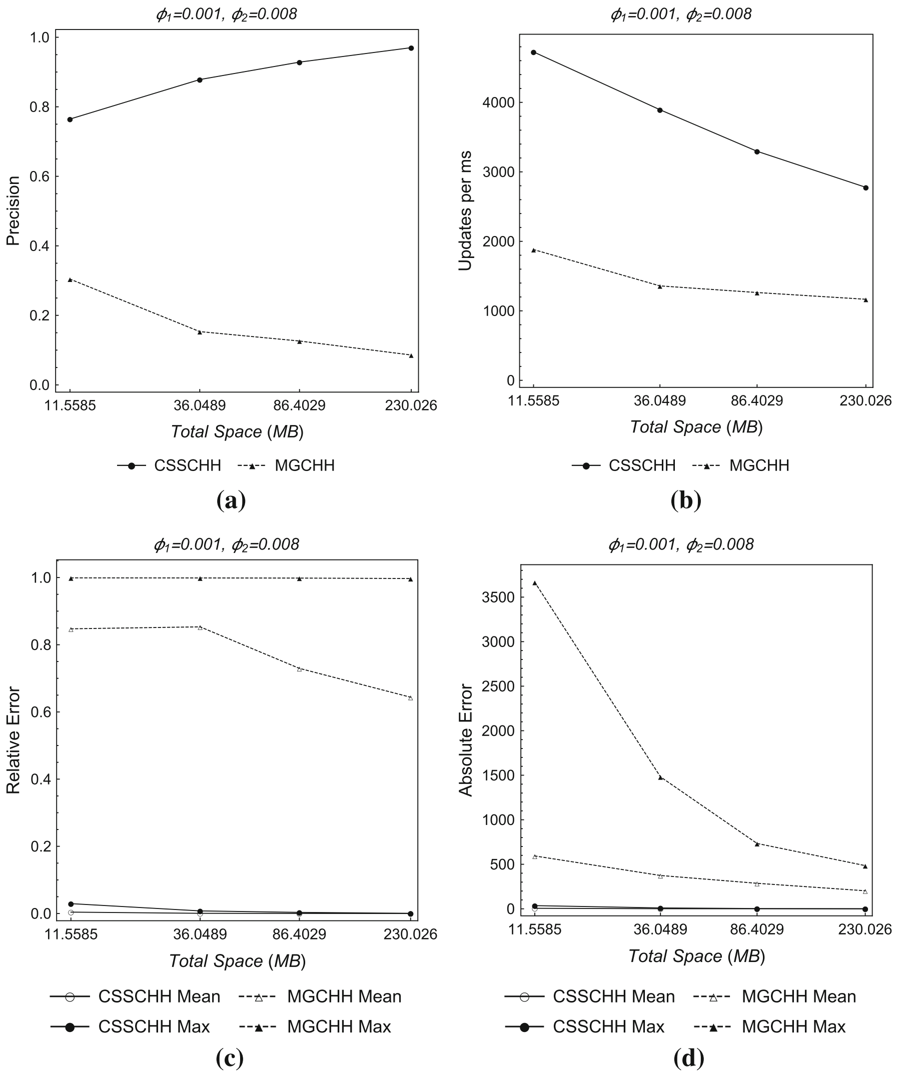

Fig. 6 Worldcup'98 results obtained varying the space allowed (mean and confidence interval), **a** precision, **b** updates/ms, **c** relative error, **d** absolute error

datasets, with CSSCHH clearly outperforming MGCHH with regard to every metric under consideration.

We conclude by noting that the experimental results fully confirm our theoretical expectations reported in the previous section. CSSCHH is more accurate in terms of both precision, absolute and relative error committed. Our algorithm is also faster than MGCHH. Therefore, CSSCHH is a better alternative to MGCHH for mining correlated heavy hitters.

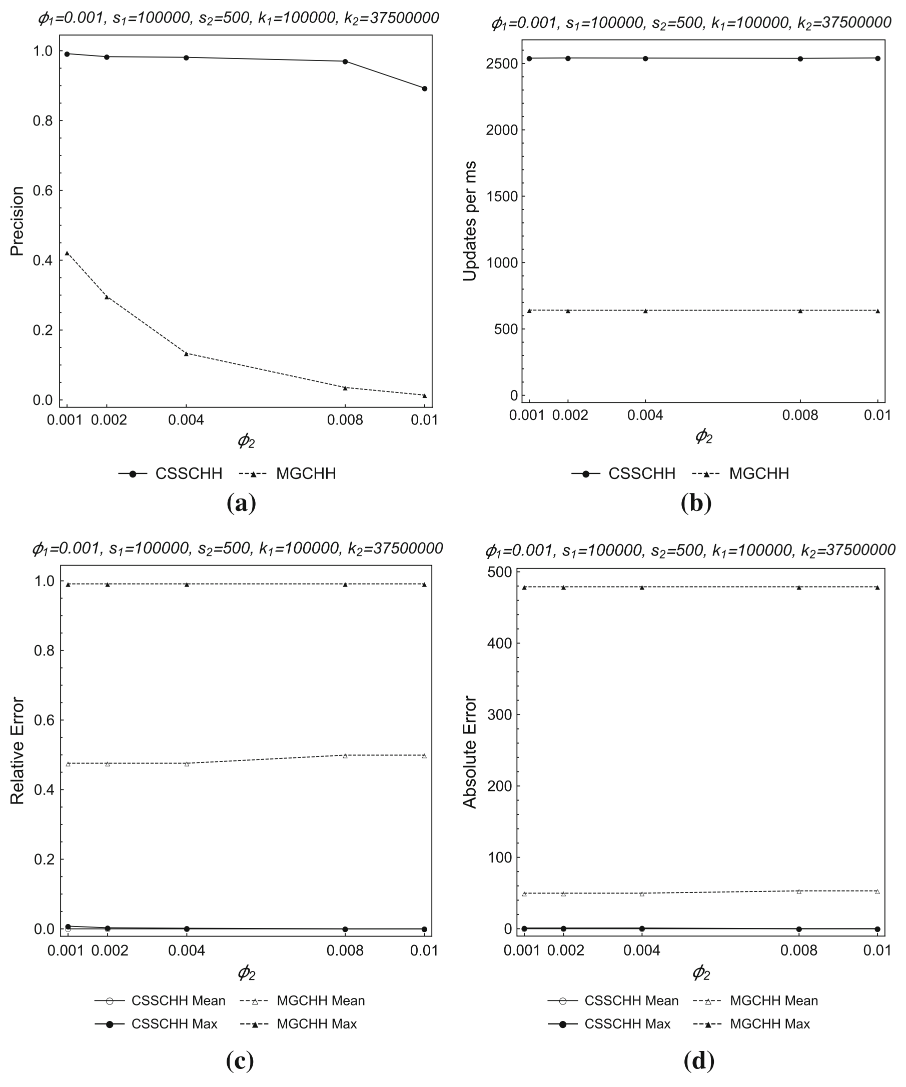

Fig. 7 Worldcup'98 results obtained varying the ϕ_2 threshold (mean and confidence interval), **a** precision, **b** updates/ms, **c** relative error, **d** absolute error

9 Conclusions

In this paper, we have studied the problem of mining correlated heavy hitters from a two-dimensional data stream. We have presented CSSCHH, a new counter-based algorithm for tracking CHHs, and formally proved its error bounds and correctness. We have compared our algorithm to MGCHH, a recently designed deterministic algorithm based on the Misra–Gries algorithm both from a theoretical point of view and through extensive experimental results, and we have shown that our algorithm outperforms it with regard to accuracy and speed whilst requiring asymptotically much less space.

References

Agrawal R, Imieliński T, Swami A (1993) Mining association rules between sets of items in large databases. ACM SIGMOD Rec 22(2):207–216

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

Cafaro M, Pulimeno M (2016) Merging frequent summaries. In: Proceedings of the 17th Italian conference on theoretical computer science (ICTCS 2016), vol 1720, CEUR proceedings, pp 280–285

Cafaro M, Tempesta P (2011) Finding frequent items in parallel. Concurr Comput Pract Exp 23(15):1774–1788. doi:10.1002/cpe.1761

Cafaro M, Pulimeno M, Epicoco I, Aloisio G (2016a) Mining frequent items in the time fading model. Inf Sci 370–371:221–238. doi:10.1016/j.ins.2016.07.077

Cafaro M, Pulimeno M, Tempesta P (2016b) A parallel space saving algorithm for frequent items and the hurwitz zeta distribution. Inf Sci 329:1–19. doi:10.1016/j.ins.2015.09.003, http://www.sciencedirect.com/science/article/pii/S002002551500657X

Cafaro M, Pulimeno M, Epicoco I, Aloisio G (2017) Parallel space saving on multi- and many-core processors. Concurr Comput Pract Exp. doi:10.1002/cpe.4160

Charikar M, Chen K, Farach-Colton M (2002) Finding frequent items in data streams. In: ICALP '02: proceedings of the 29th international colloquium on automata. Languages and programming. Springer-Verlag, pp 693–703

Chen L, Mei Q (2014) Mining frequent items in data stream using time fading model. Inf Sci 257:54–69. doi:10.1016/j.ins.2013.09.007, http://www.sciencedirect.com/science/article/pii/S0020025513006403

Cheng J, Ke Y, Ng W (2008) A survey on algorithms for mining frequent itemsets over data streams. Knowl Inf Syst 16(1):1–27. doi:10.1007/s10115-007-0092-4

Cormode G, Hadjieleftheriou M (2009) Finding the frequent items in streams of data. Commun ACM 52(10):97–105. doi:10.1145/1562764.1562789

Cormode G, Muthukrishnan S (2005a) An improved data stream summary: the count-min sketch and its applications. J Algorithms 55(1):58–75. doi:10.1016/j.jalgor.2003.12.001

Cormode G, Muthukrishnan S (2005b) What's hot and what's not: tracking most frequent items dynamically. ACM Trans Database Syst 30(1):249–278. doi:10.1145/1061318.1061325

Cormode G, Korn F, Tirthapura S (2008) Exponentially decayed aggregates on data streams. In: IEEE 24th international conference on data engineering, 2008, ICDE 2008, pp 1379–1381, doi:10.1109/ICDE.2008.4497562

Cormode G, Tirthapura S, Xu B (2009) Time-decayed correlated aggregates over data streams. Stat Anal Data Min 2(5–6):294–310

Das S, Antony S, Agrawal D, El Abbadi A (2009) Thread cooperation in multicore architectures for frequency counting over multiple data streams. Proc VLDB Endow 2(1):217–228. doi:10.14778/1687627.1687653

Datar M, Gionis A, Indyk P, Motwani R (2002) Maintaining stream statistics over sliding windows: (extended abstract). In: Proceedings of the thirteenth annual ACM-SIAM symposium on discrete algorithms, SODA '02. Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, pp 635–644

Demaine ED, López-Ortiz A, Munro JI (2002) Frequency estimation of internet packet streams with limited space. In: ESA, pp 348–360

Erra U, Frola B (2012) Frequent items mining acceleration exploiting fast parallel sorting on the gpu. Proc Comput Sci 9(0):86–95, doi:10.1016/j.procs.2012.04.010, http://www.sciencedirect.com/science/article/pii/S1877050912001317, proceedings of the international conference on computational science, ICCS 2012

Gehrke J, Korn F, Srivastava D (2001) On computing correlated aggregates over continual data streams. SIGMOD Rec 30(2):13–24. doi:10.1145/376284.375665

Govindaraju NK, Raghuvanshi N, Manocha D (2005) Fast and approximate stream mining of quantiles and frequencies using graphics processors. In: Proceedings of the 2005 ACM SIGMOD international conference on management of data, SIGMOD '05, ACM, pp 611–622, doi:10.1145/1066157.1066227

Han J, Pei J, Yin Y (2000) Mining frequent patterns without candidate generation. ACM SIGMOD Rec 29(2):1–12

Jin C, Qian W, Sha C, Yu JX, Zhou A (2003) Dynamically maintaining frequent items over a data stream. In: Proceedings of CIKM, ACM Press, pp 287–294

Karp RM, Shenker S, Papadimitriou CH (2003) A simple algorithm for finding frequent elements in streams and bags. ACM Trans Database Syst 28(1):51–55. doi:10.1145/762471.762473

Lahiri B, Mukherjee AP, Tirthapura S (2016) Identifying correlated heavy-hitters in a two-dimensional data stream. Data Min Knowl Discov 30(4):797–818. doi:10.1007/s10618-015-0438-6

Manerikar N, Palpanas T (2009) Frequent items in streaming data: an experimental evaluation of the state-of-the-art. Data Knowl Eng 68(4):415–430. doi:10.1016/j.datak.2008.11.001

Manjhi A, Shkapenyuk V, Dhamdhere K, Olston C (2005) Finding (recently) frequent items in distributed data streams. In: Proceedings of 21st international conference on data engineering, 2005, ICDE 2005, pp 767–778, doi:10.1109/ICDE.2005.68

Manku GS, Motwani R (2002) Approximate frequency counts over data streams. In: VLDB, pp 346–357

Metwally A, Agrawal D, Abbadi AE (2006) An integrated efficient solution for computing frequent and top-k elements in data streams. ACM Trans Database Syst 31(3):1095–1133. doi:10.1145/1166074.1166084

Mirylenka K, Cormode G, Palpanas T, Srivastava D (2015) Conditional heavy hitters: detecting interesting correlations in data streams. VLDB J 24(3):395–414. doi:10.1007/s00778-015-0382-5

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

Roy P, Teubner J, Alonso G (2012) Efficient frequent item counting in multi-core hardware. In: Proceedings of the 18th ACM SIGKDD international conference on knowledge discovery and data mining, KDD '12, ACM, pp 1451–1459, doi:10.1145/2339530.2339757

Tangwongsan K, Tirthapura S, Wu KL (2014) Parallel streaming frequency-based aggregates. In: Proceedings of the 26th ACM symposium on parallelism in algorithms and architectures, SPAA '14, ACM, pp 236–245, doi:10.1145/2612669.2612695

Zaki MJ (2000) Scalable algorithms for association mining. IEEE Trans Knowl Data Eng 12(3):372–390

Zhang Y (2012) Parallelizing the weighted lossy counting algorithm in high-speed network monitoring. In: Second international conference on instrumentation, measurement, computer, communication and control (IMCCC), pp 757–761, doi:10.1109/IMCCC.2012.183

Zhang Y, Singh S, Sen S, Duffield N, Lund C (2004) Online identification of hierarchical heavy hitters: algorithms, evaluation, and applications. In: Proceedings of the 4th ACM SIGCOMM conference on Internet measurement, ACM, pp 101–114

Zhang Y, Sun Y, Zhang J, Xu J, Wu Y (2014) An efficient framework for parallel and continuous frequent item monitoring. Concurr Comput Pract Exp 26(18):2856–2879. doi:10.1002/cpe.3182