

New Algorithms for Heavy Hitters in Data Streams *

David P. Woodruff
 IBM Research Almaden
 dpwoodru@us.ibm.com

Abstract

An old and fundamental problem in databases and data streams is that of finding the heavy hitters, also known as the top- k , most popular items, frequent items, elephants, or iceberg queries. There are several variants of this problem, which quantify what it means for an item to be frequent, including what are known as the ℓ_1 -heavy hitters and ℓ_2 -heavy hitters. There are a number of algorithmic solutions for these problems, starting with the work of Misra and Gries, as well as the CountMin and CountSketch data structures, among others.

In this survey paper, accompanying an ICDT invited talk, we cover several recent results developed in this area, which improve upon the classical solutions to these problems. In particular, with coauthors we develop new algorithms for finding ℓ_1 -heavy hitters and ℓ_2 -heavy hitters, with significantly less memory required than what was known, and which are optimal in a number of parameter regimes.

1 The Heavy Hitters Problem

A well-studied problem in databases and data streams is that of finding the heavy hitters, also known as the top- k , most popular items, frequent items, elephants, or iceberg queries. These can be used for flow identification at IP routers [21], in association rules and frequent itemsets [1, 25, 26, 44, 47], and for iceberg queries and iceberg datacubes [7, 22, 24]. We refer the reader to the survey [18], which presents an overview of known algorithms for this problem, from both theoretical and practical standpoints.

There are various different flavors of guarantees for the heavy hitters problem. We start with what is known as the ℓ_1 -guarantee:

Definition 1 (ℓ_1 - (ϵ, ϕ) -Heavy Hitters Problem) *In the (ϵ, ϕ) -Heavy Hitters Problem, we are given parameters $0 < \epsilon < \phi < 1$, as well as a stream a_1, \dots, a_m of items $a_j \in \{1, 2, \dots, n\}$. Let f_i denote the number of occurrences of item i , i.e., its frequency. The algorithm should make one pass over the stream and at the end of the stream output a set $S \subseteq \{1, 2, \dots, n\}$ for which if $f_i \geq \phi m$, then $i \in S$, while if $f_i \leq (\phi - \epsilon)m$, then $i \notin S$. Further, for each item $i \in S$, the algorithm should output an estimate \tilde{f}_i of the frequency f_i which satisfies $|f_i - \tilde{f}_i| \leq \epsilon m$.*

We are interested in algorithms which use as little space (i.e., memory) in bits as possible to solve the ℓ_1 - (ϵ, ϕ) -Heavy Hitters Problem. We allow the algorithm to be randomized and to succeed with probability at least $1 - \delta$, for $0 < \delta < 1$. We do not make any assumption on the

*A preliminary version of this paper appeared as an invited paper in ICDT 2016.

ordering of the stream a_1, \dots, a_m . This is desirable, as often in applications one cannot assume a best-case or even a random order. We will assume m is known in advance, though many of the algorithms below (including ours) can deal with unknown m .

The first algorithm for the ℓ_1 - (ϵ, ϕ) -**Heavy Hitters Problem** was given by Misra and Gries [39], who achieved $O(\epsilon^{-1} \log n)$ bits of space for any $\phi > 2\epsilon$. This algorithm was rediscovered by Demaine et al. [20], and again by Karp et al. [32]. Other than these algorithms, which are deterministic, there are a number of randomized algorithms, such as the CountSketch [15], Count-Min sketch [19], sticky sampling [35], lossy counting [35], space-saving [37], sample and hold [21], multi-stage bloom filters [13], and sketch-guided sampling [33]. Berinde et al. [6] show that using $O(k\epsilon^{-1} \log(mn))$ bits of space, one can achieve the stronger guarantee of reporting, for each item $i \in S$, f_i with $|\tilde{f}_i - f_i| \leq \frac{\epsilon}{k} F_1^{res(k)}$, where $F_1^{res(k)} < m$ denotes the sum of frequencies of items in $\{1, 2, \dots, n\}$ excluding the frequencies of the k most frequent items. This is particularly useful when there are only a few large frequencies, since then the error $\frac{\epsilon}{k} F_1^{res(k)}$ will depend only on the remaining small frequencies.

While the ℓ_1 -heavy hitters have a number of applications, there is also a sometimes stronger notion known as the ℓ_2 -heavy hitters, which we now define.

Definition 2 (ℓ_2 - (ϵ, ϕ) -**Heavy Hitters Problem**) *In the (ϵ, ϕ) -Heavy Hitters Problem, we are given parameters $0 < \epsilon < \phi < 1$, as well as a stream a_1, \dots, a_m of items $a_j \in \{1, 2, \dots, n\}$. Let f_i denote the number of occurrences of item i , i.e., its frequency. Let $F_2 = \sum_{i=1}^n f_i^2$. The algorithm should make one pass over the stream and at the end of the stream output a set $S \subseteq \{1, 2, \dots, n\}$ for which if $f_i^2 \geq \phi F_2$, then $i \in S$, while if $f_i^2 \leq (\phi - \epsilon) F_2$, then $i \notin S$. Further, for each item $i \in S$, the algorithm should output an estimate \tilde{f}_i of the frequency f_i which satisfies $|f_i - \tilde{f}_i| \leq \epsilon \sqrt{F_2}$.*

One of the algorithms for ℓ_1 -heavy hitters mentioned above, the CountSketch [16], refined in [46], actually solves the ℓ_2 - (ϵ, ϕ) -**Heavy Hitters Problem**. Notice that this guarantee can be significantly stronger than the aforementioned ℓ_1 -guarantee that $f_i \geq \epsilon m$. Indeed, if $f_i \geq \phi m$, then $f_i^2 \geq \phi^2 m^2 \geq \phi^2 F_2$. So, an algorithm for finding the ℓ_2 -heavy hitters, with ϕ replaced by ϕ^2 , will find all items satisfying the ℓ_1 -guarantee with parameter ϕ . On the other hand, given a stream of n distinct items in which $f_{i^*} = \sqrt{n}$ for an $i^* \in [n] = \{1, 2, 3, \dots, n\}$, yet $f_i = 1$ for all $i \neq i^*$, an algorithm satisfying the ℓ_2 -heavy hitters guarantee will identify item i with constant ϕ , but an algorithm which only has the ℓ_1 -guarantee would need to set $\phi = 1/\sqrt{n}$, therefore using $\Omega(\sqrt{n})$ bits of space. In fact, ℓ_2 -heavy hitters are in some sense the best one can hope for with a small amount of space in a data stream, as it is known for $p > 2$ that finding those i for which $f_i^p \geq \phi F_p$ requires $n^{1-2/p}$ bits of space even for constant ϕ [5, 14].

The ℓ_2 -heavy hitter algorithms of [16, 46] have broad applications in compressed sensing [23, 38, 43] and numerical linear algebra [10, 17, 36, 41], and are often used as a subroutine in other data stream algorithms, such as ℓ_p -sampling [3, 30, 40], cascaded aggregates [29], and frequency moments [9, 28].

Given the many applications of heavy hitters, it is natural to ask what the best space complexity for them is. For simplicity of presentation, we make the common assumption that the stream length m is polynomially related to the universe size n .

It is clear that for constant ϵ and ϕ , that there is an $\Omega(\log n)$ bit lower bound, as this is just the number of bits needed to specify the identity of the heavy hitter.

For constant ϵ , given the aforementioned results, this is actually tight for the ℓ_1 - (ϵ, ϕ) -**Heavy Hitters Problem**. The main focus then, for the ℓ_1 - (ϵ, ϕ) -**Heavy Hitters Problem** is on obtaining

tight bounds as a function of ϵ and ϕ .

On the other hand, for the ℓ_2 - (ϵ, ϕ) -**Heavy Hitters Problem**, even for constant ϵ and ϕ , the best previous algorithms of [16] and the followup [46] achieve $\Theta(\log^2 n)$ bits of space. It is known that if one allows deletions in the stream, in addition to insertions, then $\Theta(\log^2 n)$ bits of space is optimal [4, 30]. However, in many cases we just have a stream of insertions, such as in the model studied in the seminal paper of Alon, Matias, and Szegedy [2]. Thus, for the ℓ_2 - (ϵ, ϕ) -**Heavy Hitters Problem**, our focus will be on the regime of constant ϵ and ϕ and on understanding the dependence on n .

There are a number of other desirable properties one would want out of a heavy hitters algorithm. For instance, one is often also interested in minimizing the *update time* and *reporting time* of such algorithms. Here, the update time is defined to be the time the algorithm needs to update its data structure when processing a stream insertion. The reporting time is the time the algorithm needs to report the answer after having processed the stream. In this article we will focus primarily on the space complexity. For other very interesting recent work on improving the reporting time in a stream of insertions and deletions, see [34]. The results in this survey are focused on a stream of insertions only (for which, as mentioned above, smaller space bounds are possible).

2 Our Recent Results

In several recent works [8, 11, 12], we significantly improve known algorithms for finding both ℓ_1 -heavy hitters as well as ℓ_2 -heavy hitters. For many settings of parameters, our algorithms are optimal.

2.1 ℓ_1 -Heavy Hitters

In joint work with Bhattacharyya and Dey [8], we improve upon the basic algorithm of Misra and Gries [39] for the ℓ_1 - (ϵ, ϕ) -**Heavy Hitters Problem**, the latter achieving $O(\epsilon^{-1} \log n)$ bits of space for any $\phi \geq 2\epsilon$. There are two algorithms of [8], the first a bit simpler and already achieving a large improvement over [39], and the second an optimal algorithm. We first discuss the first algorithm.

We first recall the algorithm of Misra and Gries. That algorithm initializes a table of $1/\epsilon + 1$ pairs of (v, c) to $(\perp, 0)$, where v is an element in the universe $\{1, 2, \dots, n\} \cup \perp$, and c is a non-negative integer. When receiving a new stream insertion a_i , the algorithm checks if $v = a_i$ for some (v, c) pair in the table. If so, it replaces (v, c) with $(v, c + 1)$. Otherwise, if there is a (v, c) in the table with $v = \perp$, then the algorithm replaces that (v, c) pair with $(a_i, 1)$. If neither of the previous two cases hold, the algorithm takes each (v, c) pair in the table, and replaces it with $(v, c - 1)$. If $c - 1 = 0$, then the corresponding v is replaced with \perp .

Note that the algorithm, as described in the previous paragraph, naturally can be implemented using $O(\epsilon^{-1} \log n)$ bits of space (recall we assume the stream length m and the universe size n are polynomially related, so $\log m = \Theta(\log n)$). Moreover, a nice property is that the algorithm is deterministic.

For the correctness, note that if an item i occurs $f_i \geq 2\epsilon m$ times, then it will appear in the table at the end of the stream. Indeed, notice that for each occurrence of i in the stream, if it is not included in the table via the operation of replacing a pair (i, c) with $(i, c + 1)$ for some value of c , or replacing a pair $(\perp, 0)$ with $(i, 1)$, then this means that there were at least $1/\epsilon + 1$ stream updates that were removed from the table upon seeing this occurrence of i , since each counter c

for each (v, c) pair in the table is decremented by 1. We can therefore charge those stream updates to this occurrence of i . Moreover, if (i, c) is in the table for some value of c and is replaced with $(i, c - 1)$ or $(\perp, 0)$, this means we can charge at least $1/\epsilon$ stream updates to items not equal to i to this occurrence of i . Since we are charging distinct stream updates for each occurrence of i , we have the relationship that $f_i \cdot (1/\epsilon) \leq m$, which is a contradiction to $f_i \geq 2\epsilon m$. Therefore, i will occur in a pair in the table at the end of the stream. The same analysis in fact implies that at most ϵm occurrences of i will not be accounted for in the table at the end of the stream, which means that for the (i, c) pair in the table, we have $f_i \geq c \geq f_i - \epsilon m$. This latter guarantee enables us to solve the $\ell_1(\epsilon, \phi)$ -**Heavy Hitters Problem** for any $\phi \geq 2\epsilon$.

One shortcoming of the algorithm above is that if ϕ is much larger than ϵ , say ϕ is constant, then the above algorithm still requires $O(\epsilon^{-1} \log n)$ bits of space, that is, it is insensitive to the value of ϕ . Consider for instance, the case when $\epsilon = 1/\log n$ and $\phi = 1/10$, so one wants a very high accuracy estimate to each of the item frequencies for items occurring at least 10% of the time. The above algorithm would use $O(\log^2 n)$ bits of space for this problem. In this case, the only known lower bound is $\Omega(\log n)$ bits, which just follows from the need to return the identities of the heavy hitters. Is it possible to improve this $O(\log^2 n)$ bits of space upper bound?

This is precisely what we show in [8]. Here we sketch how to achieve a bound of $O((1/\phi) \log n + (1/\epsilon) \log(1/\epsilon))$ bits of space and refer to [8] for further optimizations as well as extensions to related problems. Note that this translates to a space bound of $O(\log n \log \log n)$ bits for the above setting of parameters.

The first observation is that if we randomly sample $r = \Theta(1/\epsilon^2)$ stream updates, then with probability 99%, simultaneously for every universe item i , if we let \hat{f}_i denote its frequency among the samples, and f_i its frequency in the original stream, then we have

$$\left| \frac{\hat{f}_i}{r} - \frac{f_i}{m} \right| \leq \frac{\epsilon}{2}.$$

This follows by Chebyshev's inequality and a union bound. Indeed, consider a given $i \in [n]$ with frequency f_i and suppose we sample each of its occurrences pairwise-independently with probability r/m , for a parameter r . Recall that pairwise independence here implies that any single occurrence is sampled with probability r/m and any two occurrences are jointly sampled with probability exactly r^2/m^2 , though we do not impose any constraints on the joint distribution of any three or more samples. Also, a pairwise independent hash function can be represented with only $O(\log n)$ bits of space. Then the expected number $\mathbf{E}[\hat{f}_i]$ of sampled occurrences is $f_i \cdot r/m$ and the variance $\mathbf{Var}[\hat{f}_i]$ is $f_i \cdot r/m(1 - r/m) \leq f_i r/m$ (here we use pairwise independence to conclude the same variance bound as if the samples were fully independent). Applying Chebyshev's inequality,

$$\Pr \left[\left| \hat{f}_i - \mathbf{E}[\hat{f}_i] \right| \geq \frac{r\epsilon}{2} \right] \leq \frac{\mathbf{Var}[\hat{f}_i]}{(r\epsilon/2)^2} \leq \frac{4f_i r}{mr^2 \epsilon^2}.$$

Setting $r = \frac{C}{\epsilon^2}$ for a constant $C > 0$ makes this probability at most $\frac{4f_i}{Cm}$. By the union bound, if we sample each element in the stream independently with probability $\frac{r}{m}$, then the probability there exists an i for which $|\hat{f}_i - \mathbf{E}[\hat{f}_i]| \geq \frac{r\epsilon}{2}$ is at most $\sum_{i=1}^n \frac{4f_i}{Cm} \leq \frac{4}{C}$, which for $C \geq 400$ is at most $\frac{1}{100}$, as desired.

After sampling so that the stream length is reduced to $O(1/\epsilon^2)$, it follows that the number of distinct items in the stream is also $O(1/\epsilon^2)$, and therefore if we hash the item identifiers to a

universe of size $O(1/\epsilon^4)$, by standard arguments with probability 99% the items will be perfectly hashed, that is, there will be no collisions. This follows even with a pairwise-independent hash function h . The high level idea then is to run the algorithm of Misra and Gries, but the pairs (v, c) correspond to the *hashed* item identity and the *count in the sampled stream*, respectively. Notice that it takes only $O(\log(1/\epsilon))$ bits to represent such pairs and so the algorithm of Misra and Gries would take $O(\epsilon^{-1} \log(1/\epsilon))$ bits of space.

However, we still want to return the actual item identifiers! To do this, we maintain a parallel data structure containing actual item identifiers in $[n]$, but the data structure only contains $O(1/\phi)$ items. In particular, these item identities correspond to the items v for which $(h(v), c)$ is stored in the algorithm of Misra and Gries, for which the c values are largest. Namely, the items with top $1/\phi$ c -values have their actual identities stored. This can be maintained under stream insertions since given a new stream update, one has the actual identity in hand, and therefore can appropriately update the identities of the items with top $O(1/\phi)$ counts. Moreover, when we subtract one from all counters in the algorithm of Misra and Gries, the only thing that changes in the top $O(1/\phi)$ identities is that some of them may now have zero frequency, and so can be thrown out. Thus, we can always maintain the actual top $O(1/\phi)$ identities in the original (before hashing) universe.

The second algorithm of [8] achieves an optimal $O(\epsilon^{-1} \log \phi^{-1} + \phi^{-1} \log n + \log \log m)$ bits of space. The algorithm can be seen as an extension of our first algorithm. The idea of the optimal algorithm, as in our first algorithm, is to have a list of the top $O(1/\phi)$ -heavy hitters with exact identities, and to use a separate data structure to approximate their individual frequencies up to $\epsilon \cdot m$. In the earlier algorithm, this was an accompanying Misra-Gries data structure on the hashed universe identities and sample count values; in the new one we optimize this data structure to use $O(\epsilon^{-1} \log \phi^{-1})$ bits instead of the earlier $O(\epsilon^{-1} \log \epsilon^{-1})$ bits. We have $O(1/\epsilon)$ counts, as before, but now in each count we spend $O(1)$ bits on average. We also eliminate the need to maintain hashed identities in the earlier algorithm by partitioning the items into $O(1/\epsilon)$ buckets using a hash function and maintaining the approximate sum in each bucket. We note that the counts need to be randomized, but in a different sense than probabilistic counters since we want them to achieve additive error $O(1/\epsilon)$ rather than the relative error guarantee of probabilistic counters. We call these *accelerated counters* since their relative error improves as the count gets larger. We are able to compress the counts since they sum up to $O(1/\epsilon^2)$, which is the length of the sampled stream. Each count is individually only correct with constant probability, so we have $O(\log(1/\phi))$ repetitions and take a median across the repetitions to get a correct count for each of the $O(1/\phi)$ heavy hitters in our list.

We refer the reader to [8] for further details about both algorithms.

2.2 ℓ_2 -Heavy Hitters

In joint work with Braverman, Chestnut, and Ivkin [12], we improve upon the CountSketch data structure [16] for the ℓ_2 - (ϵ, ϕ) -**Heavy Hitters Problem**. To illustrate the algorithm of [12], we consider ϵ and ϕ to be constants in what follows, and further, we suppose there is only a single $i^* \in [n]$ for which $f_{i^*}^2 \geq \phi F_2$ and there is no i for which $(\phi - \epsilon)F_2 \leq f_i^2 < \phi F_2$. It is not hard to reduce to this case by first hashing into $O(1)$ buckets (recall ϕ, ϵ are constants for this discussion), since the $O(1/\phi)$ heavy hitters will go to separate buckets with large constant probability (if, say, we have $\Omega(1/\phi^2)$ buckets). Thus, we focus on this case. In this case the CountSketch algorithm would use $\Theta(\log^2 n)$ bits of space, whereas in [12] we achieve $O(\log n \log \log n)$ bits of space, nearly matching the trivial $\Omega(\log n)$ bit lower bound.

We first explain the CountSketch data structure. The idea is to assign each item $i \in [n]$ a random sign $\sigma(i) \in \{-1, 1\}$. We also randomly partition $[n]$ into B buckets via a hash function h and maintain a counter $c_j = \sum_{i|h(i)=j} \sigma(i) \cdot f_i$ in the j -th bucket. Then, to estimate any given frequency f_i , we estimate it as $\sigma(i) \cdot c_{h(i)}$. Note that $\mathbf{E}[\sigma(i) \cdot c_{h(i)}] = \mathbf{E}[\sigma(i)^2 f_i + \sum_{j \neq i, h(j)=h(i)} f_j \sigma(j) \sigma(i)] = f_i$, using that $\mathbf{E}[\sigma(i)\sigma(j)] = 0$ for $i \neq j$. Moreover, by computing the variance and applying Chebyshev's inequality, one has that

$$|\sigma(i) \cdot c_{h(i)} - f_i| = O(\sqrt{F_2/B})$$

with probability at least 9/10. The intuitive explanation is that due to the random sign combination of remaining items in the same hash bucket as i , the absolute value of this linear combination concentrates to the Euclidean norm of the frequency vector of these items. The idea then is to repeat this independently $O(\log n)$ times in parallel. Then we estimate f_i by taking the median of the estimates across each of the $O(\log n)$ repetitions. By Chernoff bounds, we have that with probability $1 - 1/n^2$, say, the resulting estimate is within an additive $O(\sqrt{F_2/B})$ of the true frequency f_i . This then holds for every $i \in [n]$ simultaneously by a union bound, at which point one can then find the ℓ_2 -heavy hitters, if say, one sets $B = \Theta(1/\epsilon^2)$.

Notice that it is easy to maintain the CountSketch data structure in a data stream since we just need to hash the new item i to the appropriate bucket and add $\sigma(i)$ to the counter in that bucket, once for each of the $O(\log n)$ repetitions. The total space complexity of the CountSketch algorithm is $O(B \cdot \log^2 n)$, where the “ B ” is the number of hash buckets, one $\log n$ factor is to store the counter in each bucket, and the other $\log n$ factor is for the number of repetitions. For constant ϵ and $B = \Theta(1/\epsilon^2)$ this gives $O(\log^2 n)$ bits of space. It is also not hard to see that the CountSketch data structure can be maintained in a stream with deletions as well as insertions, since given a deletion to item i , this just corresponds to subtracting $\sigma(i)$ from the bucket i hashes to in each repetition. Moreover, as mentioned earlier, this $O(\log^2 n)$ space bound is optimal for streams with deletions.

To give some intuition for our new algorithm, let $i^* \in [n]$ be the identity of the single ℓ_2 -heavy hitter that we wish to find. Suppose first that $f_{i^*} \geq \sqrt{n} \log n$ and that $f_i \in \{0, 1\}$ for all $i \in [n] \setminus \{i^*\}$. For the moment, we are also going to ignore the issue of storing random bits, so assume we can store $\text{poly}(n)$ random bits for free (which can be indexed into using $O(\log n)$ bits of space). We will later sketch how to remove this assumption. As in the CountSketch algorithm, we again assign a random sign $\sigma(i)$ to each item $i \in [n]$. Suppose we randomly partition $[n]$ into two buckets using a hash function $h : [n] \rightarrow \{1, 2\}$, and correspondingly maintain two counters $c_1 = \sum_{i|h(i)=1} \sigma(i) \cdot f_i$ and $c_2 = \sum_{i|h(i)=2} \sigma(i) \cdot f_i$. Suppose for discussion that $h(i^*) = 1$. A natural question is what the values c_1 and c_2 look like as we see more updates in the stream.

Consider the values $c_1 - \sigma(i^*) \cdot f_{i^*}$ and c_2 . Then, since all frequencies other than i^* are assumed to be 0 or 1, and since the signs $\sigma(j)$ are independent, these two quantities evolve as random walks starting at 0 and incrementing by +1 with probability 1/2, and by -1 with probability 1/2, at each step of the walk. By standard theory of random walks (e.g., Levy's theorem), there is a constant $C > 0$ so that with probability at least 9/10, simultaneously at all times during the stream we have that $|c_1 - \sigma(i^*) \cdot f_{i^*}|$ and $|c_2|$ are upper bounded by $C\sqrt{n}$. The constant of 9/10, like typical constants in this paper, is somewhat arbitrary. This suggests the following approach to learning i^* : at some point in the stream we will have that $f_{i^*} > 2C\sqrt{n}$, and at that point $|c_1| > C\sqrt{n}$, but then we know that i^* occurs in the first bucket. This is assuming that the above event holds for the random walks. Since we split $[n]$ randomly into two pieces, this gives us 1 bit of information

about the identity of i^* . If we were to repeat this $O(\log n)$ times in parallel, we would get exactly the CountSketch data structure, which would use $\Theta(\log^2 n)$ bits of space. Instead, we get much better space by repeating $\Theta(\log n)$ times sequentially!

To repeat this sequentially, we simply wait until either $|c_1|$ or $|c_2|$ exceeds $Cn^{1/2}$, at which point we learn one bit of information about i^* . Then, we reset the two counters to 0 and perform the procedure again. Assuming $f_{i^*} = \Omega(\sqrt{n} \log n)$, we will have $\Omega(\log n)$ repetitions of this procedure, each one succeeding independently with probability 9/10. By Chernoff bounds, there will only be a single index $i \in [n]$ which match a 2/3 fraction of these repetitions, and necessarily $i = i^*$.

2.2.1 Gaussian Processes

In general we do not have $f_{i^*} = \Omega(\sqrt{n} \log n)$, nor do we have that $f_i \in \{0, 1\}$ for all $i \in [n] \setminus \{i^*\}$. We fix both problems using the theory of Gaussian processes.

Definition 3 A Gaussian process is a collection $\{X_t\}_{t \in T}$ of random variables, for an index set T , for which every finite linear combination of the random variables is Gaussian.

We assume $\mathbf{E}[X_t] = 0$ for all t , as this will suffice for our application. It then follows that the Gaussian process is entirely determined by its covariances $\mathbf{E}[X_s X_t]$. This fact is related to the fact that a Gaussian distribution is determined by its mean and covariance. The distance function $d(s, t) = (\mathbf{E}[(X_s - X_t)^2])^{1/2}$ is then a pseudo-metric on T (the only property it lacks of a metric is that $d(s, t)$ may equal 0 if $s \neq t$).

The connection to data streams is the following. Suppose we replace the signs $\sigma(i)$ with standard normal random variables $g(i)$ in our counters above, and consider a counter c at time t , denoted $c(t)$, of the form $\sum_i g(i) \cdot f_i(t)$. Here $f_i(t)$ is the frequency of item i after processing t stream insertions. The main point is that $c(t)$ is a Gaussian process! Indeed, any linear combination of the $c(t)$ values for different t is again Gaussian since the sum of normal random variables is again a normal random variable.

The reason we wish to make such a connection to Gaussian processes is the following powerful inequality called the “chaining inequality”.

Theorem 4 (Talagrand [45]) Let $\{X_t\}_{t \in T}$ be a Gaussian process and let $T_0 \subseteq T_1 \subseteq T_2 \subseteq \dots \subseteq T$ be such that $|T_0| = 1$ and $|T_i| \leq 2^{2^i}$ for $i \geq 1$. Then,

$$\mathbf{E} \left[\sup_{t \in T} X_t \right] \leq O(1) \cdot \sup_{t \in T} \sum_{i \geq 0} 2^{i/2} d(t, T_i),$$

where $d(t, T_i) = \min_{s \in T_i} d(t, s)$.

We wish to apply Theorem 4 to the problem of finding ℓ_2 -heavy hitters. Let $F_2(t)$ be the value of the second moment F_2 after seeing t stream insertions. We now describe how to choose the sets T_i in order to apply the chaining inequality; the intuition is that we recursively partition the stream based on its F_2 value.

Let a_t be the first stream update for which $F_2(m)/2 \leq F_2(t)$. Then $T_0 = \{t\}$. We then let T_i be the set of 2^{2^i} times $t_1, t_2, \dots, t_{2^{2^i}}$ in the stream for which t_j is the first point in the stream for which $j \cdot F_2(m)/2^{2^i} \leq F_2(t_j)$. Then, we have created a nested sequence of subsets $T_0 \subseteq T_1 \subseteq T_2 \subseteq \dots \subseteq T$ with $|T_0| = 1$ and $|T_i| \leq 2^{2^i}$ for $i \geq 1$.

We are now in position to apply Theorem 4. A straightforward computation based on our recursive partitioning of the stream around where F_2 changes (see [12] for details) shows that for any stream position t and set T_i we have created,

$$d(t, T_i) = \min_{s \in T_i} (\mathbf{E}[|c(t) - c(s)|^2])^{1/2} = O\left(\frac{F_2}{2^{2i}}\right)^{1/2}.$$

Applying Theorem 4, we have

$$\mathbf{E}[\sup_{t \in T} X_t] \leq O(1) \sup_{t \in T} \sum_{i \geq 0} 2^{i/2} \left(\frac{F_2}{2^{2i}}\right)^{1/2} = O(F_2^{1/2}).$$

This is exactly the same bound that the theory for random walks gave us earlier! (recall in that case $\sum_{i \neq i^*} f_i^2 < n$).

Using Gaussian processes has therefore allowed us to remove our earlier assumption that $f_i \in \{0, 1\}$ for all $i \in [n] \setminus \{i^*\}$. The same random walk based algorithm will now work; however, we still need to assume the $f_{i^*} = \Omega(\sqrt{F_2} \log n)$ in order to learn $\log n$ bits of information to identify i^* , as before. This is not satisfactory, as an ℓ_2 -heavy hitter only satisfies $f_i = \Omega(\sqrt{F_2})$ (recall we have assumed ϕ and ϵ are constants), which is weaker than the $f_{i^*} = \Omega(\sqrt{F_2} \log n)$ that the above analysis requires.

2.2.2 Amplification

To remove the assumption that $f_{i^*} = \Omega(\sqrt{F_2} \log n)$, our work [12] designs what we call an “amplification” procedure. This involves for $j = 1, 2, \dots, O(\log \log n)$, independently choosing a pairwise independent hash function $h^j : [n] \rightarrow \{1, 2\}$. For each j , we as before maintain two counters $c_1^j = \sum_{i|h^j(i)=1} g_j(i) \cdot f_i$ and $c_2^j = \sum_{i|h^j(i)=2} g_j(i) \cdot f_i$, where the $g_j(i)$ are independent standard normal random variables.

Applying the chaining inequality to each of the $O(\log \log n)$ counters created, we have that with large constant probability, in a constant fraction of the $O(\log \log n)$ pairs, both counters c_1^j and c_2^j will be bounded by $O(\sqrt{F_2})$ in magnitude. It follows that if $f_{i^*} \geq C\sqrt{F_2}$ for a sufficiently large constant $C > 0$ (which we can assume by first hashing the universe into $O(1)$ buckets before the streaming algorithm begins), then in say, a 9/10 fraction of pairs j , the counter c_k^j , $k \in \{1, 2\}$, of larger magnitude will contain i^* . Moreover, by Chernoff bounds, only a $\frac{1}{\log^c n}$ fraction of other $i \in [n]$ will hash to the larger counter in at least a 9/10 fraction of such pairs, where $c > 0$ is a constant that can be made arbitrarily large by increasing the constant in the number $O(\log \log n)$ of pairs of counters created. Now the idea is to effectively run our previous algorithm only on items which hash to the heavier counter in at least a 9/10 fraction of pairs. By definition, this will contain i^* , and now the expected second moment of the other items for which we run the algorithm on will be $F_2 / \log^c n$, which effectively makes $f_{i^*} = \Omega(\sqrt{F_2} \log n)$, where F_2 is now measured with respect to the items for which we run the algorithm on. Now we can sequentially learn $O(\log n)$ bits of information about i^* in our algorithm, as before.

One thing to note about this approach is that after seeing a sufficiently large number of insertions of i^* , i.e., $\Theta(\sqrt{F_2})$ such insertions, then most of the pairs of counters will have the property that the larger counter (in absolute value) stays larger forever. This is due to the chaining inequality. This can be used to fix the itemset for which we run the algorithm on. In fact, this is precisely why this does not result in a 2-pass algorithm, which one might expect since one does not know the

itemset to run our algorithm on in advance. However, we always run the algorithm on whichever current itemset agrees with at least a $9/10$ fraction of the larger counters, and just accept the fact that in the beginning of the stream the bits we learn about i^* are nonsense; however, after enough updates to i^* have occurred in the stream then the counters “fix” themselves in the sense that the larger counter does not change. At this point the bits we learn about i^* in our algorithm are the actual bits that we desire. At the end of the stream, we only look at a suffix of these bits to figure out i^* , thereby ignoring the nonsensical bits at the beginning of the stream. We refer the reader to [12] for more details.

2.2.3 Derandomization

The final piece of the algorithm is to account for the randomness used by the algorithm. We need to derandomize the counters, which use the theory of Gaussian processes to argue their correctness. We also cannot afford to maintain all of the hash functions that were used to learn specific bits of i^* (which we need ad the end of the stream to figure out what i^* is).

To derandomize the Gaussian processes, we use a derandomized Johnson-Lindenstrauss transform of Kane, Meka, and Nelson [31]. The rough idea is to first apply a Johnson-Lindenstrauss transform to the frequency vectors for which we take inner products with independent Gaussian random variables in our counters. This will reduce the dimension from n to $O(\log n)$, for which we can then afford to take an inner product with fully independent Gaussian random variables. The nice thing about Johnson-Lindenstrauss transforms is that they preserve all the covariances up to a constant factor in our specific Gaussian process, and therefore we can use Slepian’s Lemma (see [12] for details) to argue that the Gaussian process is roughly the same as before, since it is entirely determined by its covariances. Here the derandomized Johnson-Lindenstrauss transform of [31] can be represented using only $O(\log n \log \log n)$ bits of space. Also, instead of using Gaussian random variables, which require truncation, we can directly use sign random variables (+1 with probability $1/2$, -1 with probability $1/2$), which results in what are called Bernoulli processes, together with a comparison theorem for Bernoulli processes and Gaussian processes. This enables us to avoid arguments about truncating Gaussians.

To derandomize the hash functions, we use Nisan’s pseudorandom generator in a similar way that Indyk uses it for derandomizing his algorithms for norm estimation [27, 42]. Please see [12] for further details.

2.3 Followup Work

Very recently, in followup work by Braverman et al. [11], we improved the space bound further to the optimal $O(\log n)$ bits of space (for constant ϵ, ϕ). The high level idea of using Gaussian or Bernoulli processes is the same, but several additional insights were needed. This involves both a new algorithm which we call **BPTree**, which avoids the amplification step and Nisan’s pseudorandom generator described above, as well as a better derandomization of the Bernoulli processes using $O(1)$ -wise independence. We refer the reader to that work for further details.

3 Conclusions

We presented new algorithms for finding ℓ_1 -heavy hitters and ℓ_2 -heavy hitters in a data stream. We refer the reader to the original papers cited above for further details. As these problems are

inspired from applications in practice, it is very interesting to see how the improved theoretical algorithms perform in practice. In ongoing work we are testing these algorithms in practice on real datasets.

Another interesting aspect is that the technique of using Gaussian processes in the ℓ_2 -heavy hitters algorithm has led to a number of other improvements to data stream algorithms, including for example the ability to estimate the second moment F_2 at all times in a stream of insertions. Previously, given a stream of length n and a universe of size n , to estimate F_2 at all points in a stream up to a constant factor would require $\Theta(\log^2 n)$ bits of space, since it takes $\Theta(\log n \log(1/\delta))$ bits to estimate it at a single point with failure probability δ , and one needs to union bound over n stream positions. Using Gaussian processes, [12] achieves only $O(\log n \log \log n)$ bits of space for this task. It would be interesting to see if Gaussian processes are useful for other problems in data streams.

References

- [1] Rakesh Agrawal and Ramakrishnan Srikant. Fast algorithms for mining association rules in large databases. In *VLDB'94, Proceedings of 20th International Conference on Very Large Data Bases, September 12-15, 1994, Santiago de Chile, Chile*, pages 487–499, 1994.
- [2] Noga Alon, Yossi Matias, and Mario Szegedy. The space complexity of approximating the frequency moments. *J. Comput. Syst. Sci.*, 58(1):137–147, 1999.
- [3] Alexandr Andoni, Robert Krauthgamer, and Krzysztof Onak. Streaming algorithms via precision sampling. In *IEEE 52nd Annual Symposium on Foundations of Computer Science, FOCS 2011, Palm Springs, CA, USA, October 22-25, 2011*, pages 363–372, 2011.
- [4] Khanh Do Ba, Piotr Indyk, Eric Price, and David P. Woodruff. Lower bounds for sparse recovery. *CoRR*, abs/1106.0365, 2011.
- [5] Ziv Bar-Yossef, T. S. Jayram, Ravi Kumar, and D. Sivakumar. An information statistics approach to data stream and communication complexity. *J. Comput. Syst. Sci.*, 68(4):702–732, 2004.
- [6] Radu Berinde, Piotr Indyk, Graham Cormode, and Martin J. Strauss. Space-optimal heavy hitters with strong error bounds. *ACM Trans. Database Syst.*, 35(4):26, 2010.
- [7] Kevin S. Beyer and Raghu Ramakrishnan. Bottom-up computation of sparse and iceberg cubes. In *SIGMOD 1999, Proceedings ACM SIGMOD International Conference on Management of Data, June 1-3, 1999, Philadelphia, Pennsylvania, USA.*, pages 359–370, 1999.
- [8] Arnab Bhattacharyya, Palash Dey, and David P. Woodruff. An optimal algorithm for l_1 -heavy hitters in insertion streams and related problems. *PODS*, 2016.
- [9] Lakshminath Bhuvanagiri, Sumit Ganguly, Deepanjan Kesh, and Chandan Saha. Simpler algorithm for estimating frequency moments of data streams. In *Proceedings of the Seventeenth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2006, Miami, Florida, USA, January 22-26, 2006*, pages 708–713, 2006.

- [10] Jean Bourgain and Jelani Nelson. Toward a unified theory of sparse dimensionality reduction in euclidean space. *CoRR*, abs/1311.2542, 2013.
- [11] Vladimir Braverman, Stephen R. Chestnut, Nikita Ivkin, Jelani Nelson, Zhengyu Wang, and David P. Woodruff. BPTree: an ℓ_2 heavy hitters algorithm using constant memory. *CoRR*, abs/1603.00759, 2016.
- [12] Vladimir Braverman, Stephen R. Chestnut, Nikita Ivkin, and David P. Woodruff. Beating countsketch for heavy hitters in insertion streams. *STOC*, 2016.
- [13] Yousra Chabchoub, Christine Fricker, and Hanene Mohamed. Analysis of a bloom filter algorithm via the supermarket model. In *21st International Teletraffic Congress, ITC 2009, Paris, France, September 15-17, 2009*, pages 1–8, 2009.
- [14] Amit Chakrabarti, Subhash Khot, and Xiaodong Sun. Near-optimal lower bounds on the multi-party communication complexity of set disjointness. In *18th Annual IEEE Conference on Computational Complexity (Complexity 2003), 7-10 July 2003, Aarhus, Denmark*, pages 107–117, 2003.
- [15] Moses Charikar, Kevin Chen, and Martin Farach-Colton. Finding frequent items in data streams. *Theoretical Computer Science*, 312(1):3–15, 2004.
- [16] Moses Charikar, Kevin Chen, and Martin Farach-Colton. Finding frequent items in data streams. *Theor. Comput. Sci.*, 312(1):3–15, 2004.
- [17] Kenneth L. Clarkson and David P. Woodruff. Low rank approximation and regression in input sparsity time. In *Symposium on Theory of Computing Conference, STOC’13, Palo Alto, CA, USA, June 1-4, 2013*, pages 81–90, 2013.
- [18] Graham Cormode and Marios Hadjieleftheriou. Finding frequent items in data streams. *Proceedings of the VLDB Endowment*, 1(2):1530–1541, 2008.
- [19] Graham Cormode and S Muthukrishnan. An improved data stream summary: the count-min sketch and its applications. *Journal of Algorithms*, 55(1):58–75, 2005.
- [20] Erik D Demaine, Alejandro López-Ortiz, and J Ian Munro. Frequency estimation of internet packet streams with limited space. In *AlgorithmsESA 2002*, pages 348–360. Springer, 2002.
- [21] Cristian Estan and George Varghese. New directions in traffic measurement and accounting: Focusing on the elephants, ignoring the mice. *ACM Trans. Comput. Syst.*, 21(3):270–313, 2003.
- [22] Min Fang, Narayanan Shivakumar, Hector Garcia-Molina, Rajeev Motwani, and Jeffrey D. Ullman. Computing iceberg queries efficiently. In *VLDB’98, Proceedings of 24rd International Conference on Very Large Data Bases, August 24-27, 1998, New York City, New York, USA*, pages 299–310, 1998.
- [23] Anna C. Gilbert, Yi Li, Ely Porat, and Martin J. Strauss. Approximate sparse recovery: optimizing time and measurements. In *Proceedings of the 42nd ACM Symposium on Theory of Computing, STOC 2010, Cambridge, Massachusetts, USA, 5-8 June 2010*, pages 475–484, 2010.

- [24] Jiawei Han, Jian Pei, Guozhu Dong, and Ke Wang. Efficient computation of iceberg cubes with complex measures. In *Proceedings of the 2001 ACM SIGMOD international conference on Management of data, Santa Barbara, CA, USA, May 21-24, 2001*, pages 1–12, 2001.
- [25] Jiawei Han, Jian Pei, and Yiwen Yin. Mining frequent patterns without candidate generation. In *Proceedings of the 2000 ACM SIGMOD International Conference on Management of Data, May 16-18, 2000, Dallas, Texas, USA.*, pages 1–12, 2000.
- [26] Christian Hidber. Online association rule mining. In *SIGMOD 1999, Proceedings ACM SIGMOD International Conference on Management of Data, June 1-3, 1999, Philadelphia, Pennsylvania, USA.*, pages 145–156, 1999.
- [27] Piotr Indyk. Stable distributions, pseudorandom generators, embeddings, and data stream computation. *J. ACM*, 53(3):307–323, 2006.
- [28] Piotr Indyk and David P. Woodruff. Optimal approximations of the frequency moments of data streams. In *Proceedings of the 37th Annual ACM Symposium on Theory of Computing, Baltimore, MD, USA, May 22-24, 2005*, pages 202–208, 2005.
- [29] T. S. Jayram and David P. Woodruff. The data stream space complexity of cascaded norms. In *50th Annual IEEE Symposium on Foundations of Computer Science, FOCS 2009, October 25-27, 2009, Atlanta, Georgia, USA*, pages 765–774, 2009.
- [30] Hossein Jowhari, Mert Saglam, and Gábor Tardos. Tight bounds for ℓ_p samplers, finding duplicates in streams, and related problems. In *Proceedings of the 30th ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems, PODS 2011, June 12-16, 2011, Athens, Greece*, pages 49–58, 2011.
- [31] Daniel M. Kane, Raghu Meka, and Jelani Nelson. Almost optimal explicit johnson-lindenstrauss families. In *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques - 14th International Workshop, APPROX 2011, and 15th International Workshop, RANDOM 2011, Princeton, NJ, USA, August 17-19, 2011. Proceedings*, pages 628–639, 2011.
- [32] Richard M Karp, Scott Shenker, and Christos H Papadimitriou. A simple algorithm for finding frequent elements in streams and bags. *ACM Transactions on Database Systems (TODS)*, 28(1):51–55, 2003.
- [33] Abhishek Kumar and Jun (Jim) Xu. Sketch guided sampling - using on-line estimates of flow size for adaptive data collection. In *INFOCOM 2006. 25th IEEE International Conference on Computer Communications, Joint Conference of the IEEE Computer and Communications Societies, 23-29 April 2006, Barcelona, Catalunya, Spain*, 2006.
- [34] Kasper Green Larsen, Jelani Nelson, Huy Le Nguyen, and Mikkel Thorup. Optimal space heavy hitters with fast update and query time. 2016.
- [35] Gurmeet Singh Manku and Rajeev Motwani. Approximate frequency counts over data streams. In *Proceedings of the 28th international conference on Very Large Data Bases*, pages 346–357. VLDB Endowment, 2002.

- [36] Xiangrui Meng and Michael W. Mahoney. Low-distortion subspace embeddings in input-sparsity time and applications to robust linear regression. In *Symposium on Theory of Computing Conference, STOC'13, Palo Alto, CA, USA, June 1-4, 2013*, pages 91–100, 2013.
- [37] Ahmed Metwally, Divyakant Agrawal, and Amr El Abbadi. Efficient computation of frequent and top-k elements in data streams. In *Proceedings of the 10th International Conference on Database Theory, ICDT'05*, pages 398–412, Berlin, Heidelberg, 2005. Springer-Verlag.
- [38] Gregory T. Minton and Eric Price. Improved concentration bounds for count-sketch. In *Proceedings of the Twenty-Fifth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2014, Portland, Oregon, USA, January 5-7, 2014*, pages 669–686, 2014.
- [39] Jayadev Misra and David Gries. Finding repeated elements. *Sci. Comput. Program.*, 2(2):143–152, 1982.
- [40] Morteza Monemizadeh and David P. Woodruff. 1-pass relative-error ℓ_p -sampling with applications. In *Proceedings of the Twenty-First Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2010, Austin, Texas, USA, January 17-19, 2010*, pages 1143–1160, 2010.
- [41] Jelani Nelson and Huy L. Nguyen. OSNAP: faster numerical linear algebra algorithms via sparser subspace embeddings. In *54th Annual IEEE Symposium on Foundations of Computer Science, FOCS 2013, 26-29 October, 2013, Berkeley, CA, USA*, pages 117–126, 2013.
- [42] Noam Nisan. Pseudorandom generators for space-bounded computation. *Combinatorica*, 12(4):449–461, 1992.
- [43] Eric Price. Efficient sketches for the set query problem. In *Proceedings of the Twenty-Second Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2011, San Francisco, California, USA, January 23-25, 2011*, pages 41–56, 2011.
- [44] Ashok Savasere, Edward Omiecinski, and Shamkant B. Navathe. An efficient algorithm for mining association rules in large databases. In *VLDB'95, Proceedings of 21th International Conference on Very Large Data Bases, September 11-15, 1995, Zurich, Switzerland.*, pages 432–444, 1995.
- [45] Michel Talagrand. Majorizing measures: The generic chaining. *The Annals of Probability*, 24(3), 1996.
- [46] Mikkel Thorup and Yin Zhang. Tabulation-based 5-independent hashing with applications to linear probing and second moment estimation. *SIAM J. Comput.*, 41(2):293–331, 2012.
- [47] Hannu Toivonen. Sampling large databases for association rules. In *VLDB'96, Proceedings of 22th International Conference on Very Large Data Bases, September 3-6, 1996, Mumbai (Bombay), India*, pages 134–145, 1996.