New Heavy Hitters Algorithms with Tail Bounds

Anonymous Author(s)

ABSTRACT

In a stream of m items belonging to $\{1, \ldots, n\}$, the (ε, φ)-*Heavy Hitters* problem is to output a set S of items containing all items of frequency $\geq \varphi m$ and no item of frequency $< (\varphi - \varepsilon)m$. It is one of the most heavily-studied problems in data streams, with a wide variety of applications.

In this work, we propose new randomized counter-based algorithms for the heavy-hitters problem. We show that not only do they have nearly optimal space complexity with respect to ε, φ, and n, they also exhibit strong error bounds in terms of the frequency tail of the input and perform favorably on realistic data sets.

CCS CONCEPTS

• **Information systems** → **Data mining**; • **Theory of computation** → *Design and analysis of algorithms*; • **Networks** → Network monitoring.

KEYWORDS

data streaming, frequent items, randomized algorithms

ACM Reference Format:

Anonymous Author(s). 2019. New Heavy Hitters Algorithms with Tail Bounds. In *KDD '19: SIGKDD Conference on Knowledge Discovery and Data Mining, Aug 04–06, 2019, Anchorage, AL.* ACM, New York, NY, USA, 7 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 INTRODUCTION

1.1 Background

The *data streaming* model is an important tool for analyzing algorithms on massive data sets [16]. Streaming algorithms take a long stream of items as input and produce a compact summary of the data as output. This summary can be used to answer queries about the properties of the input data stream. The algorithms are often constrained to take a single pass over the data.

A concrete application is the problem of monitoring IP network traffic. Here, data stream management systems monitor IP packets sent on communication links and perform detailed statistical analyses. These analyses are crucial for fault diagnoses and for verifying network performance and security. Examples of such systems are Gigascope at AT&T [10] and CMON at Sprint [17]. The main challenge in these systems is to quickly and accurately perform the needed analyses in the face of a very high rate of updates.

In this work, we consider the *heavy-hitters* problem. Informally, the goal is to find the frequent elements in the stream; the same problem is also studied under the names of top-k, popular items, frequent items, elephants, or iceberg queries. It is very heavily studied and commonly used, e.g. in network analyses to find addresses that account for a large fraction of a network link utilization in a given time window. It is also often solved as a subroutine within more advanced data stream computations, e.g. approximating the entropy of a stream. Cormode and Hadjieleftheriou in [8] give an excellent survey of the work on this problem in a unified framework.

We formulate the heavy-hitters problem rigorously as follows:

Definition 1.1. In the (ε, φ)-*Heavy Hitters Problem*, we are given parameters $0 < \varepsilon < \varphi \leqslant 1$ and a stream $a_1, \ldots., a_m$ of items a_j in $\{1, 2, \ldots..n\}$. Let f_i denote the number of occurrences of item i, i.e., its frequency[1]. An algorithm for the problem should make one pass over the stream and at the end of the stream output a set $S \subseteq \{1, 2, \ldots..n\}$ which contains any i such that $f_i \geq \varphi m$ and does not contain any i such that $f_i < (\varphi - \varepsilon)m$.

Furthermore, for each item $i \in S$, the algorithm should output an estimate $\tilde{f}_i$ of the frequency f_i which satisfies $|f_i - \tilde{f}_i| \leqslant \varepsilon m$.

A large number of algorithms have been proposed for finding the heavy hitters and their variants. Following [8], they can be broadly classified into *Counter-based* algorithms, *Quantile* Algorithms, and *Sketch* algorithms.

One of the first algorithms for the problem, as well as perhaps the most widely used, is the deterministic counter-based Frequent algorithm due to Misra and Gries [15] (independently re-discovered 20 years later with some refinements to the implementation by Karp et al. [12] and Demaine et al. [11]). There are several variants of the algorithm, notably LossyCounting [13] and SpaceSaving [14], that sometimes perform better in terms of space usage or update time than the original version. All these algorithms use $O(\varepsilon^{-1}(\log n + \log m))$ bits of space; note the bound does *not* depend on φ. Sketching algorithms, such as the CountSketch [7] or the Count-Min Sketch [9], require more storage (but they can handle more general streams with both insertions and deletions that we do not study here).

Recently, in [6], Bhattacharyya et al. introduced two new algorithms for the (ε, φ)-heavy hitters problem that improve on the space usage of Frequent, particularly when $\varphi \gg \varepsilon$. One of their algorithms uses $O(\varepsilon^{-1} \log \varphi^{-1} + \varphi^{-1} \log n + \log \log m)$ bits of storage, which is optimal upto constant factors. Moreover, the algorithms take a constant amount of time per update. These algorithms are fundamentally counter-based and use the Misra-Gries algorithm as a subroutine, but they also use random sampling and hashing in crucial ways.

[1] We assume throughout this paper that there are only insertions (no deletion) in the stream and that each insertion of an item increases its frequency by 1.

KDD '19, August 04–06, 2019, Anchorage, AL

ACM ISBN 978-x-xxxx-xxxx-x/YY/MM…$15.00
https://doi.org/10.1145/nnnnnnn.nnnnnnn

1.2 Our Work

Given the strong theoretical guarantees of the algorithms proposed in [6], it is natural to wonder how well they fare in practice.

Berinde et al. [5] observe that one of the main reasons for the real-world success of FREQUENT is that it displays "better-than-advertised" performance on streams which have a *skewed* frequency distribution. In particular, they showed that the error made by FREQUENT in frequency estimation satisfies a "tail bound", so that it is very small when heavy hitters constitute most of the stream.

The starting point of this work is the observation that the algorithms proposed in [6] do *not* exhibit a tail bound, so that the estimation error can be large even when the frequency distribution is very skewed. We design new randomized algorithms inspired by [6] that improve FREQUENT in terms of bits of storage while still achieving a tail bound. We show two different algorithms with this feature. Although they are novel to the best of our knowledge, they are both quite simple to implement and analyze. The first algorithm combines a FREQUENT data structure with a Count-Min sketch [9]. The second algorithm maintains two FREQUENT data structures, one with a smaller number of counters containing items from the original universe and another with a larger number of counters recording hashes of the original item id's. Armed with the tail guarantee and following the analysis of Berinde et al. [5], we then show that our proposed algorithms enjoy several other appealing properties, such as guarantees for skewed Zipfian streams, for the sparse recovery problem and for merging multiple streams.

We experimentally evaluate . . . .

2 PRELIMINARIES

2.1 General

For a stream consisting of m insertions from the universe $[n] := \{1, \ldots, n\}$, we let the n-dimensional vector $f = (f_1, \ldots, f_n)$ denote the frequencies of the n items. So, $\sum_{i=1}^{n} f_i = m$. Without loss of generality, we will assume that $f_1 \geq f_2 \geq \cdots \geq f_n$.

For any integer $k \geq 0$, the *residual k-tail* of the stream is defined as[2]:

$$F_1^{\text{res}(k)} = \sum_{i=k+1}^{n} f_i.$$

That is, the residual k-tail is the sum of the frequencies except the k largest ones.

Recall from Definition 1.1 that an algorithm for the (ε, φ)-heavy hitters problem is supposed to output estimates $\tilde{f}_i$ for each i in the output set such that:

$$|\tilde{f}_i - f_i| \leq \varepsilon \cdot m = \varepsilon \cdot F_1^{\text{res}(0)}.$$

Definition 2.1 (Tail Bound). An algorithm for the (ε, φ)-heavy hitters problem is said to satisfy the *K-tail bound* if for every $0 \leq k \leq K$:

$$|\tilde{f}_i - f_i| \leq \varepsilon \cdot F_1^{\text{res}(k)}$$

for all i in the output set of the algorithm.

The larger the K we can get for the tail bound, the stronger is the obtained guarantee.

[2] The 1 in the subscript refers to the fact that this is the 1st moment of the residual stream.

Our proposed algorithms use the notion of a *universal hash family*, which we define next.

Definition 2.2. A family $\mathcal{H}$ consisting of functions $h : U \to [m]$ is said to be a *universal hash family* if for all distinct $x, y \in U$:

$$\Pr_{h \leftarrow \mathcal{H}}[h(x) = h(y)] \leq \frac{1}{m}$$

where the probability is over h drawn uniformly at random from $\mathcal{H}$.

It is folklore that if $m \leq |U|$, there exists a universal hash family whose members can be represented uniquely using $O(\log |U|)$ bits.

2.2 The FREQUENT algorithm

Algorithm 1: FREQUENT, [15]

```
Data: Number of counters t
Initialization: begin
    Empty set T
    Empty array c of length n
end
Function Insert(i)
    if i ∈ T then
        c_i ← c_i + 1;
    end
    else if |T| < t then
        T ← T ∪ {i};
        c_i ← 1;
    end
    else
        foreach j ∈ T do
            c_j ← c_j − 1;
            if c_j = 0 then T ← T \ {j};
        end
    end
Function Report()
    foreach i ∈ T do Output (i, c_i);
```

FREQUENT, shown in Algorithm 1, maintains t (item, counter) pairs. Its reported frequencies are accurate to within $m/(t+1)$ where m is the length of the stream. In particular, it reports all items with frequency at least $m/(t+1)$.

A simple grouping argument can be used to verify the correctness of the algorithm. With $t = \lceil 1/\varepsilon \rceil$ counters, FREQUENT solves the (ε, φ)-Heavy hitters problem using $O(\varepsilon^{-1}(\log n + \log m))$ bits of space. Berinde et al. [5] showed a stronger tail bound:

THEOREM 2.3. *([5]) For any $t \geq \frac{1}{\varepsilon}$, the FREQUENT algorithm with t counters satisfies the $\left(t - \frac{1}{\varepsilon}\right)$-tail bound and uses $O(t(\log n + \log m))$ bits of storage.*

2.3 The work of [6]

In the recent work [6], Bhattacharyya, Dey and Woodruff introduce a couple of randomized counter-based algorithms that improve upon FREQUENT in terms of the number of bits needed for storage.

Their first algorithm solves the (ε, φ)-Heavy hitters problem with space $O(1/\varepsilon \log 1/\varepsilon + 1/\varphi \log n + \log\log m)$ bits, while their second algorithm uses $O(1/\varepsilon \log 1/\varphi + 1/\varphi \log n + \log\log m)$ bits. When $\varepsilon \ll \varphi$, the space requirement for these algorithms is asymptotically smaller than the worst-case space requirement for Frequent. In fact, [6] show that the second algorithm achieves the optimal (up to constant factors) number of bits for solving the (ε, φ)-Heavy hitters problem.

The first algorithm of [6], which we dub CompressedFrequent, is shown below as Algorithm 2. CompressedFrequent runs Misra and Gries' Frequent algorithm on a sampled stream of length $O(1/\varepsilon^2)$ on a hashed universe of size $O(1/\varepsilon^4)$. By the choice of parameters, there are no hash collisions among the elements sampled from the stream with high probability. Also, by standard concentration results, the relative frequency of any element in the original stream and the sampled stream differ by at most $\varepsilon/2$. Some additional bookkeeping needs to be done to ensure that we have the original (unhashed) id's of the heavy hitters, but otherwise, the analysis is complete.

Algorithm 2: CompressedFrequent, [6]

Data: Parameters ε and φ with $0 < \varepsilon \leq \varphi < 1$.
Initialization: begin
  $\ell \leftarrow 6\log(6/\delta)/\varepsilon^2$.
  Hash function h drawn uniformly from a universal family $H \subseteq \left\{[n] \to [4\ell^2/\delta]\right\}$.
  An empty Frequent data structure $\mathcal{T}_1$ with ε^{-1} counters.
  An empty set $\mathcal{T}_2$.
end
Function *Insert(x)*
  With probability $6\ell/m$, **continue**. Else, **return**;
  Call Insert($h(x)$) on $\mathcal{T}_1$;
  if *$h(x)$ is among the top $1/\varphi$ valued items in $\mathcal{T}_1$* **then**
    if $x \notin \mathcal{T}_2$ **then**
      if $|\mathcal{T}_2| = 1/\varphi$ **then**
        For some y in $\mathcal{T}_2$ such that $h(y)$ is not among the top $1/\varphi$ valued items in $\mathcal{T}_1$, replace y with x
      else
        Insert x in $\mathcal{T}_2$;
      end
    end
  end
Function *Report()*
  foreach x *in* $\mathcal{T}_2$ **do**
    if $\mathcal{T}_1[h(x)] \geq (\varphi - \varepsilon/2)\ell$ **then** Output $(x, \mathcal{T}_1[h(x)])$;
  end

The second algorithm of [6] is more intricate. It also operates on $O(1/\varepsilon^2)$ samples from the original stream and then maintains a compressed histogram for hashes of the original items. It is somewhat reminiscent of the Count-Min sketch algorithm of Cormode and Muthukrishnan [9], but it also subsamples items from the sample with subsampling rate tuned to a mutliplicative approximation of the frequency. We omit a more detailed description of this algorithm because we do not study it in this work.

3 NEW ALGORITHMS

We start with the observation that the algorithms proposed in [6] cannot satisfy a non-trivial tail guarantee. Consider the CompressedFrequent algorithm shown in Figure 2, and suppose the input stream only consists of two items with each inserted $\Omega(m)$ times. Although $F_1^{\mathrm{res}(2)} = 0$, the frequency estimates output by the algorithm will still have nonzero error because due to random sampling, the relative frequency of the two items in the sampled stream will (with high probability) differ from that in the original stream. The same argument holds for the second algorithm of [6].

3.1 SketchFrequent

Algorithm 3: SketchFrequent

Data: Width w, Input parameters $\varphi, \delta \in (0, 1)$.
Initialization: **begin**
  $\ell \leftarrow \log(4/\delta\varphi)$
  Hash functions $h_1, \ldots, h_\ell$ drawn uniformly from a universal family $H \subseteq \{[n] \to [w]\}$.
  Empty Frequent data structure $\mathcal{T}$ with $2/\varphi$ counters.
  Empty matrix C with ℓ rows and w columns.
end
Function *Insert(x)*
  Insert x into $\mathcal{T}$;
  for $i = 1$ *to* ℓ **do**
    Increment $C[i][h_i(x)]$;
  end
Function *Report()*
  foreach $x \in \mathcal{T}$ **do**
    $\hat{f}_x \leftarrow \min_{i\in[\ell]} C[i][h_i(x)]$;
    if $\hat{f}_x \geq \varphi m$ **then**
      Output $(x, \hat{f}_x)$;
    end
  end

Our first algorithm, shown above, runs the Count-Min sketch on the output of a Frequent counter solving the $(\varphi, \varphi/2)$-Heavy hitters problem.

Theorem 3.1. *Let $\ell = \log(4/\delta\varphi)$ as in the algorithm above. Suppose that $\varepsilon < \frac{\varphi^2\delta}{2\ell}$.*

For any $w \geq \frac{2}{\varepsilon}$ and $\delta \in [0, 1]$, SketchFrequent satisfies the $\sqrt{\frac{\delta w}{\ell}}$-tail bound with probability $1 - \delta$ and uses $O(\varphi^{-1}\log n + w\log(\delta\varphi)^{-1}\log m)$ bits of storage.

In particular, for $w = \frac{2}{\varepsilon}$ and $\delta = \frac{1}{5}$, SketchFrequent solves the (ε, φ)-heavy hitters problem with probability at least $\frac{4}{5}$ using $O(\varphi^{-1}\log n + \varepsilon^{-1}\log\varphi^{-1}\log m)$ bits[3].

Proof. The space complexity is immediate. We will focus on the tail bound. We argue first that on the top $k = \sqrt{\delta w/\ell}$ elements of the stream, $h_1, \ldots, h_\ell$ do not collide with probability at least $1 - \delta/2$.

[3]The assumption that $\varepsilon < \varphi^2\delta/2\ell$ is not needed for the last part of the theorem.

Lemma 3.2. *Let $k = \sqrt{\delta w/\ell}$. For any fixed $T \subseteq [n]$ of size k,*

$$\Pr[\exists i \in [\ell], \exists x \neq y \in T, h_j(x) = h_j(y)] < \delta/2.$$

Proof. By definition of a universal hash family, for any fixed $i \in [\ell]$ and $x \neq y \in T$, $\Pr[h_i(x) = h_i(y)] < 1/w$. Using the union bound:

$$\Pr[\exists i \in [\ell], \exists x \neq y \in T, h_i(x) = h_i(y)] \leq \ell \cdot \binom{t}{2} \cdot \frac{1}{w} < \frac{\delta}{2}$$

□

Condition on the above event. Note that by our assumption, $k > 2/\varphi$. We can now run the usual analysis of the count-min sketch, which we reproduce here for completeness.

Consider an $x \in [n]$ that is stored in one of $\mathcal{T}$'s counters. Because of the above, and using again universality of the hash family, for any $i \in [\ell]$:

$$\mathbb{E}[C[i][h_i(x)]] = f_x + \frac{1}{w}\sum_{y>k} f_y = f_x + \frac{F_1^{\mathrm{res}(k)}}{w} \leq f_x + \frac{\varepsilon}{2}F_1^{\mathrm{res}(k)}.$$

Then, by Markov's inequality: $\Pr[C[i][h_i(x)] - f_x > \varepsilon F_1^{\mathrm{res}(k)}] \leq \frac{1}{2}$. Therefore:

$$\Pr[\hat{f}_x - f_x > \varepsilon F_1^{\mathrm{res}(k)}] \leq \frac{1}{2^\ell} = \frac{\varphi\delta}{4}.$$

Doing a union bound over $\frac{2}{\varphi}$ elements shows that with probability at least $1 - \delta/2$, for each $x \in \mathcal{T}$, $\hat{f}_x - f_x \leq \varepsilon F_1^{\mathrm{res}(k)}$. Moreover, observe that every x with $f_x \geq \varphi m$ is output by the algorithm. □

3.2 DoubleFrequent

Algorithm 4: DoubleFrequent

```
Data: Number of counters t, Input parameters ε, φ, δ ∈ (0, 1)
      with ε ≤ φ/2.
Initialization: begin
    Hash function h drawn uniformly from a universal
      family H ⊆ {[n] → [t²/δ]} .
    An empty Frequent data structure T1 with 2/φ
      counters.
    An empty Frequent data structure T2 with t counters.
end
Function Insert(x)
    Insert x into T1;
    Insert h(x) into T2;
Function Report()
    foreach x ∈ T1 do
        if T1(x) ≥ φm/2 ∧ T2[h(x)] ≥ (φ − ε)m then
            Output (x, T2[h(x)]);
        end
    end
```

Our second algorithm runs two Frequent algorithms in parallel. The first is a gross estimation of the frequencies with relative error up to $\varphi/2$. This is done to filter out a list of $O(1/\varphi)$ elements on which we would like the estimate to be more accurate. For this, we use more counters but we also hash the universe size down to $\mathrm{poly}(1/\varepsilon)$ to save space. The resulting algorithm inherits the desirable tail behavior of Frequent. The analysis which follows is an interesting mix of the analyses for Frequent and the Count-Min sketch.

Theorem 3.3. *Assume $\varepsilon < \varphi/2$. For any $t \geq \frac{1}{\varepsilon}$ and $\delta \in [0, 1]$, DoubleFrequent satisfies the $\left(t - \frac{1}{\varepsilon}\right)$-tail bound with probability $1 - \delta$ and uses $O(\varphi^{-1}\log n + t\log m + t\log(t/\delta))$ bits of storage.*

In particular, setting $t = 1/\varepsilon$ and $\delta = 1/5$, DoubleFrequent solves the (ε, φ)-heavy hitters problem with probability at least $\frac{4}{5}$ using space $O(\varphi^{-1}\log n + \varepsilon^{-1}\log m + \varepsilon^{-1}\log\varepsilon^{-1})$ bits of storage.

Proof. First, observe that there are likely going to be no hash collisions among the top $1/\varepsilon$ elements in the stream.

Lemma 3.4. *For any fixed $S \subseteq [n]$ of size t:*

$$\Pr[\exists x \neq y \in S, h(x) = h(y)] \leq \frac{\delta}{2}.$$

Proof. By the union bound and definition of universal hash family,

$$\Pr[\exists x \neq y \in S, h(x) = h(y)] \leq \binom{t}{2} \cdot \frac{1}{2t^2/\delta} \leq \frac{\delta}{2}$$

□

Henceforth, condition on Lemma 3.4 holding true for the top t elements. Define:

$$\overline{f}_x = \sum_{y:h(y)=h(x)} f_y.$$

Clearly, $\overline{f}_x \geq f_x$ for any x. Also, without loss of generality, suppose that $f_1 \geq f_2 \geq \cdots \geq f_n$ so that:

$$F_1^{\mathrm{res}(k)} = \sum_{i=k+1}^{n} f_i.$$

Lemma 3.5. *For any $0 \leq k \leq t$:*

$$\Pr[\exists x, \mathcal{T}_1[x] \geq \varphi m/2 \wedge \overline{f}_x > f_x + \varepsilon \cdot F_1^{res(k)}] < \frac{\delta}{2}$$

Proof. We observe that for x is among the top $2/\varphi \leq 1/\varepsilon \leq t$ elements. For any such x:

$$\mathbb{E}\overline{f}_x = f_x + \mathbb{E}\sum_{y\neq x:h(y)=h(x)} f_y = f_x + \mathbb{E}\sum_{y>t:h(y)=h(x)} f_y$$

where the last equality is from conditioning on no collisions in the top t elements. Again, using the definition of universal hash family, we get that:

$$\mathbb{E}\overline{f}_x \leq f_x + \frac{\delta}{2t^2}F_1^{\mathrm{res}(k)}$$

for any $k \leq t$. By Markov's inequality and the union bound:

$$\Pr[\exists x, \mathcal{T}_1[x] \geq \varphi m/2 \wedge \overline{f}_x > f_x + \varepsilon \cdot F_1^{\mathrm{res}(k)}] < \frac{2}{\varphi}\frac{1}{\varepsilon}\frac{\delta}{2t^2} \leq \frac{\delta}{2}$$

□

We now bound the error from the use of the Frequent algorithm in $\mathcal{T}_2$.

Lemma 3.6. *For all x and all $0 \leq k \leq t - \frac{1}{\varepsilon}$:*

$$\overline{f}_x - \mathcal{T}_2[h(x)] \leq \varepsilon F_1^{res(k)}.$$

Proof. This directly follows from Theorem 2.3 shown in [5] and the fact that the top-k residual of the $\overline{f}_x$ is at most the top-k residual of f_x. □

Putting the lemmas together, we get that with probability at least $1 - \delta$, for all $0 \leq k \leq t - \frac{1}{\varepsilon}$, if x is such that $\mathcal{T}_1[x] \geq \varphi m/2$:

$$f_x \leq \overline{f}_x \leq \mathcal{T}_2[h(x)] + \varepsilon F_1^{\mathrm{res}(k)}$$

from Lemma 3.6 and

$$f_x \geq \overline{f}_x - \varepsilon F_1^{\mathrm{res}(k)} \geq \mathcal{T}_2[h(x)] - \varepsilon F_1^{\mathrm{res}(k)}$$

from Lemma 3.5.

The space bound for DoubleFrequent is immediate from the space bound for Frequent. □

3.3 Consequences

Berinde et al. [5] show that the tail guarantee implies desirable behavior for a few other settings. We state the results below for the DoubleFrequent algorithm, though similar results should also hold for the SketchFrequent algorithm.

3.3.1 Zipfian distribution. A stream of length m on a universe of size n is said be *Zipfian* if there exists $\alpha \geq 1$ such that for every $i \in [n]$,

$$f_i = \frac{m}{i^\alpha \zeta(\alpha)} \quad \text{where} \quad \zeta(\alpha) = \sum_{i=1}^{n} \frac{1}{i^\alpha}.$$

In fact, for the results here, it is only required that the tail of the distribution be upper-bounded by the above Zipfian distribution. Of course, the order of arrivals in the stream is arbitrary.

Theorem 3.7. *Given a Zipfian stream with parameter $\alpha \geq 1$, the error in the frequency for any item reported by the DoubleFrequent algorithm with parameters δ, ε, and φ is at most $O(\varepsilon^\alpha m)$ with probability at least $1 - \delta$.*

Proof. Corollary of Theorem 8 in [5]. □

3.3.2 k-sparse recovery. In the k-sparse recovery problem, given a frequency vector f, we want to find a vector f' such that f' is k-sparse (has only k nonzero entries) and the p-norm error $\|f - f'\|_p = (\sum_i |f_i - f_i'|^p)^{1/p}$ is minimized. The next theorem shows that if $k \leq 1/\varphi$, taking the top k of the output of DoubleFrequent is a good approximation.

Theorem 3.8. *If $k \leq \frac{1}{\varphi}$ and f' is the k-sparse vector obtained by taking the top k elements of the output of DoubleFrequent (run with parameters δ, ε, and φ), with probability at least $1 - \delta$, for any $p \geq 1$:*

$$\|f - f'\|_p \leq (F_p^{res(k)})^{1/p} + O\left(\frac{\varepsilon F_1^{res(k)}}{k^{2-1/p}}\right)$$

where $(F_p^{res(k)})^{1/p}$ is the smallest L_p error of any k-sparse recovery of f.

Proof. Corollary of Theorem 5 in [5]. □

The following result also gives a guarantee if we would like to output exactly the top k elements.

Theorem 3.9. *Given a Zipfian stream with parameter $\alpha > 1$, if $k = O((\alpha/\varphi^\alpha)^{1/(\alpha+1)})$, then with probability at least $1 - \delta$, DoubleFrequent can retrieve the top-k elements of the stream in correct order.*

Proof. Corollary of Theorem 9 of [5]. □

4 EXPERIMENTAL EVALUATIONS

4.1 Setup

Existing algorithms for the heavy hitters problem have been carefully implemented in a few different places: the MassDAL Public Code Bank [3], its extension by Cormode [1], and the Yahoo Sketches Library [4]. We chose to build on the second option, Cormode's C++ extension of the MassDAL codebase, to include our implementations. We ran our experiments on a standard laptop [machine info]. Our code is available at [2].

Among previous algorithms, we considered Frequent and LCDelta (a version of the Lossy Counting algorithm of Manku and Motwani [13]; see [8]). For new algorithms, in addition to the above described SketchFrequent and DoubleFrequent, we also considered SketchLC and DoubleLC which invoke the LCDelta algorithm in place of Frequent. The analyses of SketchLC and DoubleLC mimic that of SketchFrequent and DoubleFrequent respectively, and we do not reproduce them.

REFERENCES

[1] [n. d.]. Finding Frequent Items. http://hadjieleftheriou.com/frequent-items. Accessed: 2019-02-03.

[2] [n. d.]. Heavy-HittersIISc. https://github.com/tejas1812/Heavy-HittersIISc. Accessed: 2019-02-03.

[3] [n. d.]. MassDAL Public Code Bank. https://www.cs.rutgers.edu/~muthu/massdal-code-index.html. Accessed: 2019-02-03.

[4] [n. d.]. Sketches Library from YAHOO! https://datasketches.github.io. Accessed: 2019-02-03.

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

[6] Arnab Bhattacharyya, Palash Dey, and David P Woodruff. 2018. An Optimal Algorithm for ℓ_1-Heavy Hitters in Insertion Streams and Related Problems. *ACM Transactions on Algorithms (TALG)* 15, 1 (2018), 2.

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

[8] Graham Cormode and Marios Hadjieleftheriou. 2009. Finding the frequent items in streams of data. *Commun. ACM* 52, 10 (2009), 97–105.

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

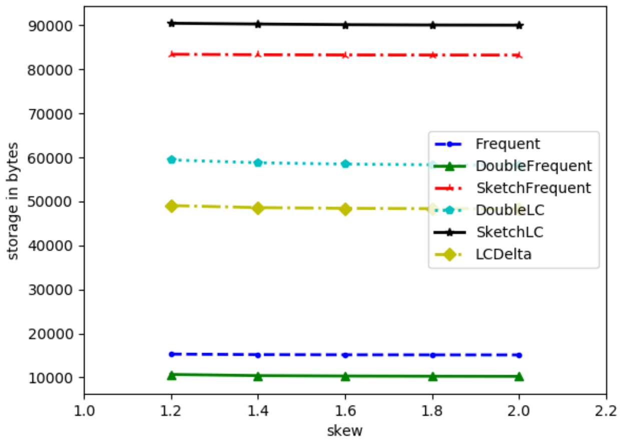
[10] Chuck Cranor, Theodore Johnson, Oliver Spataschek, and Vladislav Shkapenyuk. 2003. Gigascope: a stream database for network applications. In *Proceedings of the 2003 ACM SIGMOD international conference on Management of data*. ACM, 647–651.

[11] Erik D Demaine, Alejandro López-Ortiz, and J Ian Munro. 2002. Frequency estimation of internet packet streams with limited space. In *European Symposium on Algorithms*. Springer, 348–360.

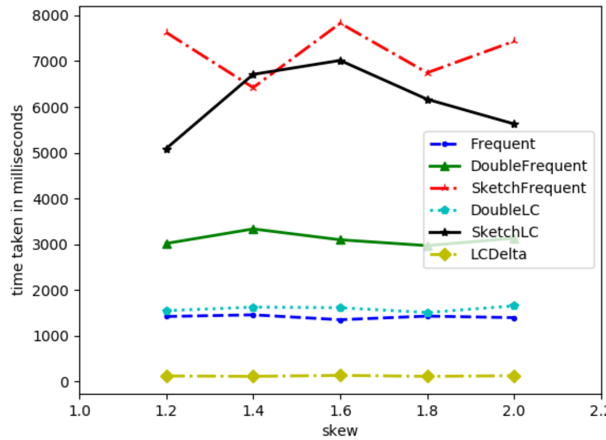
[12] Richard M Karp, Scott Shenker, and Christos H Papadimitriou. 2003. A simple algorithm for finding frequent elements in streams and bags. *ACM Transactions on Database Systems (TODS)* 28, 1 (2003), 51–55.

[13] Gurmeet Singh Manku and Rajeev Motwani. 2002. Approximate frequency counts over data streams. In *VLDB'02: Proceedings of the 28th International Conference on Very Large Databases*. Elsevier, 346–357.

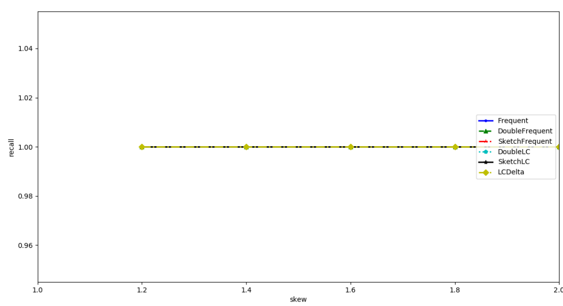
[14] Ahmed Metwally, Divyakant Agrawal, and Amr El Abbadi. 2005. Efficient computation of frequent and top-k elements in data streams. In *International Conference on Database Theory*. Springer, 398–412.



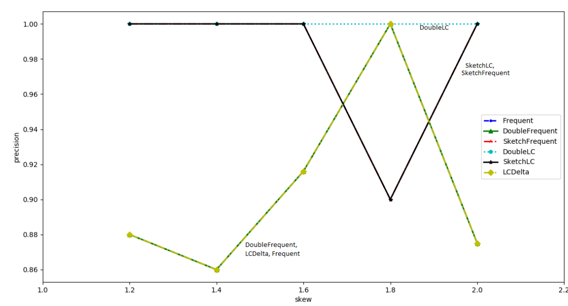
(a) Zipf: Space vs. skew



(b) Zipf: Update time vs. skew

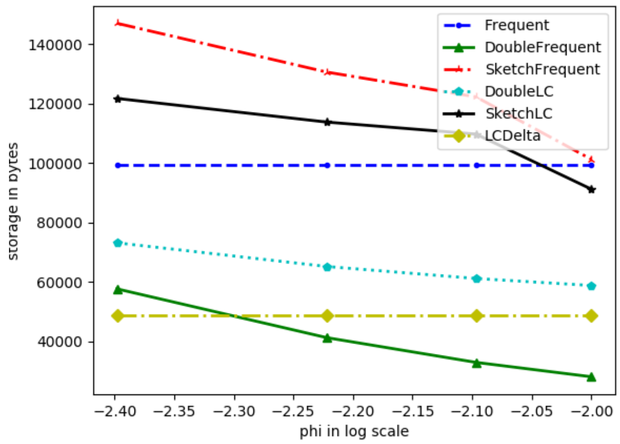


(c) Zipf: Recall vs. skew

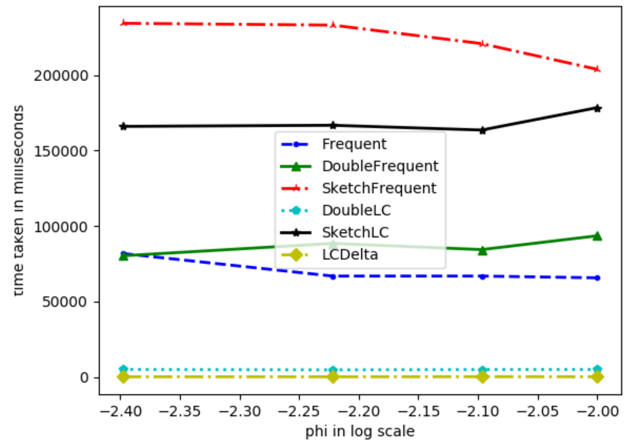


(d) Zipf: Precision vs. skew

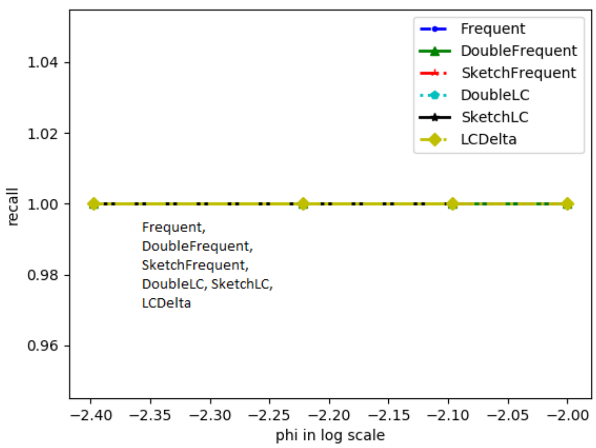
[15] J MISRA. 1982. FINDING REPEATED ELEMENTS. *Science of Computer Programming* 2 (1982), 143–152.
[16] Shanmugavelayutham Muthukrishnan et al. 2005. Data streams: Algorithms and applications. *Foundations and Trends® in Theoretical Computer Science* 1, 2 (2005), 117–236.
[17] K To, Tao Ye, and S Bhattacharyya. 2004. CMON: A general purpose continuous ip backbone traffic analysis platform. *Research Report RR04-ATL-110309, Sprint ATL* (2004).



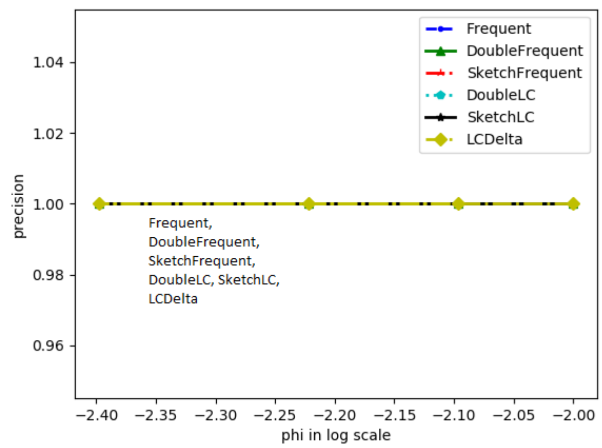
(a) Generated: Space vs. φ



(b) Generated: Update time vs. φ



(c) Generated: Recall vs. φ



(d) Generated: Precision vs. φ