BPTree: an ℓ_2 heavy hitters algorithm using constant memory

Vladimir Braverman* Stephen R. Chestnut† Nikita Ivkin‡ Jelani Nelson§
Zhengyu Wang¶ David P. Woodruff‖

March 8, 2016

Abstract

The task of finding heavy hitters is one of the best known and well studied problems in the area of data streams. In sub-polynomial space, the strongest guarantee available is the ℓ_2 guarantee, which requires finding all items that occur at least $\varepsilon\|f\|_2$ times in the stream, where the ith coordinate of the vector f is the number of occurrences of i in the stream. The first algorithm to achieve the ℓ_2 guarantee was the CountSketch of [CCF04], which for constant ε requires $O(\log n)$ words of memory and $O(\log n)$ update time, and is known to be space-optimal if the stream allows for deletions. The recent work of [BCIW16] gave an improved algorithm for insertion-only streams, using only $O(\log\log n)$ words of memory.

In this work, we give an algorithm BPTree for ℓ_2 heavy hitters in insertion-only streams that achieves $O(1)$ words of memory and $O(1)$ update time for constant ε, which is optimal. In addition, we describe an algorithm for tracking $\|f\|_2$ at all times with $O(1)$ memory and update time. Our analyses rely on bounding the expected supremum of a Bernoulli process involving Rademachers with limited independence, which we accomplish via a Dudley-like chaining argument that may have applications elsewhere.

1 Introduction

The *streaming model* of computation is well-established as one important model for processing massive datasets. In this model, a massive sequence of data is seen, and the system must be able to answer some pre-defined types of queries, such as distinct element counts, quantiles, and frequent items, to name a few. It is typically assumed that the stream being processed is so large that explicitly storing it is either impossible or undesirable. Ideally streaming algorithms should use space sublinear, or even exponentially smaller than, the size of the data, to allow the algorithm's memory footprint to fit in cache for fast stream processing. The reader is encouraged to read [BBD+02, Mut05] for further background on the streaming model of computation.

Within the study of streaming algorithms, the problem of finding frequent items is one of the most well-studied and core problems, with work on the problem beginning in 1981 [BM81, BM91]. Stated simply, given a stream of items with IDs (such as destination IP addresses of packets, or terms in search queries), the goal is to report the list of items that appear as a constant fraction of the stream. Aside from being an interesting problem in its own right, algorithms for finding frequent items are used as subroutines to solve many other streaming problems, such as moment estimation [IW05], entropy estimation [CCM10, HNO08], ℓ_p-sampling [MW10], finding duplicates [GR09], and several others.

*vova@cs.jhu.edu. Supported in part by NSF grant 1447639, by a Google Faculty Award.
†stephenc@ethz.ch.
‡nivkin1@jhu.edu. Supported in part by NSF grant 1447639 and by DARPA grant N660001-1-2-4014.
§minilek@seas.harvard.edu. Supported by NSF grant IIS-1447471 and CAREER CCF-1350670, ONR Young Investigator award N00014-15-1-2388, and a Google Faculty Research Award.
¶zhengyuwang@g.harvard.edu. Supported in part by NSF grant CCF-1350670.
‖dpwoodru@us.ibm.com.

1.1 Previous work

Work on the heavy hitters problem began in 1981 with the MJRTY algorithm of [BM81, BM91], which gave an algortihm using only two machine words of memory that could identify an item whose frequency was strictly more than half the stream. This result was generalized by the MisraGries algorithm in [MG82], which uses $2(\lceil 1/\varepsilon \rceil - 1)$ counters to identify all items that occur strictly more than an ε-fraction of the time in the stream, for any $0 < \varepsilon \leq 1/2$. This data structure was rediscovered at least two times afterward [DLM02, KSP03] and became also known as the Frequent algorithm, with implementations that use $O(1/\varepsilon)$ words of memory, $O(1)$ expected update time to process a stream item (using hashing), and $O(1/\varepsilon)$ query time to report all the frequent items. Similar space requirements and running times for finding ε-frequent items were later achieved by the SpaceSaving [MAE05] and LossyCounting [MM12] algorithms. A later analysis of these algorithms in [BCIS09] showed that they not only identify the heavy hitters, but when using $O(k/\varepsilon)$ counters, for each heavy hitter $i \in \{1, \ldots, n\}$ they provide an estimate $\tilde{f}_i$ of the frequency f_i such that $|\tilde{f}_i - f_i| \leq (\varepsilon/k) \cdot \|f_{tail(k)}\|_1$ (we call this the "$((\varepsilon/k), k)$-tail guarantee"). Here $f_{tail(k)}$ is the vector f but in which the top k entries have been replaced by zeros (and thus the norm of $f_{tail(k)}$ can never be larger than that of f). A recent work of [BDW16] shows that for $0 < \alpha < \varepsilon \leq 1/2$, all ε-heavy hitters can be found together with approximate for them $\tilde{f}_i$ such that $|\tilde{f}_i - f_i| \leq \alpha \|f\|_1$, and the space complexity is $O(\alpha^{-1} \log(1/\varepsilon) + \varepsilon^{-1} \log n + \log\log \|f\|_1)$ bits.

All the works in the previous paragraph work in the *insertion-only* model, also known as the *cash-register* model [Mut05], where deletions from the stream are not allowed. Subsequently, many works provided algorithms that work in more general models such as the strict turnstile and general turnstile models. In the turnstile model, a vector $f \in \mathbb{R}^n$ receives updates of the form $\texttt{update}(i, \Delta)$, which triggers the change $f_i \leftarrow f_i + \Delta$. The value Δ is assumed to be some bounded precision integer fitting in a machine word[1], which can be either positive or negative. In the *strict* turnstile model, we are given the promise that $f_i \geq 0$ at all times in the stream. That is, items cannot be deleted if they were never inserted in the first place. In the general turnstile model, no such restriction is promised (i.e. entries in f are allowed to be negative). This can be useful when tracking differences or changes across streams. For example, if f^1 is the query stream vector with $(f^1)_i$ being the number of times word i was queried to a search engine yesterday, and f^2 is the similar vector corresponding to today, then finding heavy coordinates in the vector $f = f^1 - f^2$ can be used to track big changes across time. Note then f receives a sequence of updates with $\Delta = +1$ (from yesterday) followed by updates with $\Delta = -1$ (from today).

In the general turnstile model, an ε-heavy hitter in the ℓ_p norm is defined as an index $i \in [n]$ such that $|f_i| \geq \varepsilon \|f\|_p$. Recall $\|f\|_p$ is defined as $(\sum_{i=1}^n |f_i|^p)^{1/p}$. The CountMin sketch treats the case of $p = 1$ and uses $O(\varepsilon^{-1} \log n)$ memory to find all ε-heavy hitters and achieve the $(\varepsilon, 1/\varepsilon)$-tail guarantee for its estimates $\tilde{f}_i$ [CM05]. The CountSketch treats the case of $p = 2$ and uses $O(\varepsilon^{-2} \log n)$ memory, achieving the $(\varepsilon, 1/\varepsilon^2)$-tail guarantee. It was later showed in [JST11] that the CountSketch actually solves ℓ_p-heavy hitters for all $0 < p \leq 2$ using $O(\varepsilon^{-p} \log n)$ memory and achieving the $(\varepsilon, 1/\varepsilon^p)$-tail guarantee. In fact they showed something stronger: that *any* ℓ_2 heavy hitters algorithm with error parameter $\varepsilon^{p/2}$ achieving the tail guarantee automatically solves the ℓ_p heavy hitters problem with error parameter ε for any $p \in (0, 2]$. In this sense, solving the heavy hitters for $p = 2$ with tail error provides the strongest guarantee amongst all $p \in (0, 2]$. It is worth pointing out that both the CountMin sketch and CountSketch are randomized algorithms, and with small probability $1/n^c$ for an arbitrarily large constant $c > 0$, they can fail to achieve their stated guarantees. The work [JST11] also showed that the CountSketch algorithm is optimal: they showed that *any* algorithm, even in the strict turnstile model, solving ℓ_p heavy hitters even with $1/3$ failure probability must use $\Omega(\varepsilon^{-p} \log n)$ memory.

The reader may also recall the Pick-and-Drop algorithm of [BO13] for finding ℓ_p heavy hitters, $p \geq 3$, in insertion-only streams. Pick-and-Drop uses $O(n^{1-2/p})$ words, so it's natural to wonder whether the same approach would work for ℓ_2 heavy hitters in $O(1)$ memory. However, Pick-and-Drop breaks down in multiple, fundamental ways that seem to prevent any attempt to repair it, as we describe in Appendix A. In particular,

[1] Henceforth, unless explicitly stated otherwise, space is always measured in machine words. It is assumed a machine word has at least $\log n$ bits, to store any ID in the stream, and also at least $\log(m\Delta)$ bits for m the length of the stream, to be able to store the total mass of items seen so far.

for certain streams it has only polynomially small probability to correctly identify an ℓ_2 heavy hitter.

Of note is that the MisraGries and other algorithms in the insertion-only model solve ℓ_1 heavy hitters using (optimal) $O(1/\varepsilon)$ memory, whereas the CountMin and CountSketch algorithms use a larger $\Theta(\varepsilon^{-1}\log n)$ memory in the strict turnstile model, which is optimal in that model. Thus there is a gap of $\log n$ between the space complexities of ℓ_1 heavy hitters in the insertion-only and strict turnstile models. [BCIW16] recently studied whether this gap also exists for ℓ_2 heavy hitters. They showed that it does: whereas the CountSketch uses $\Theta(\log n)$ memory for ℓ_2 heavy hitters for constant ε, they gave an algorithm CountSieve which uses only $\Theta(\log\log n)$ memory: an exponential improvement!

1.2 Our contributions

We provide a new algorithm, BPTree, which in the insertion-only model solves ℓ_2 heavy hitters and achieves the $(\varepsilon, 1/\varepsilon^2)$-tail guarantee. For any constant ε our algorithm only uses a constant $O(1)$ words of memory, which is optimal. This is the first optimal-space algorithm for ℓ_2 heavy hitters in the insertion-only model for constant ε. The algorithm is described in Corollary 7.

En route to describing BPTree and proving its correctness we describe another result that may be of independent interest. Theorem 4 gives a more advanced analysis of the AMS algorithm, showing that for constant ε one can achieve the same (additive) error as the AMS algorithm *at all points in the stream* with only a constant increase in storage over AMS and a change from 4-wise independence to 6-wise independence. Note that [BCIW16] describes an algorithm using $O(\log\log n)$ words that does F_2 tracking in an insertion only stream with a multiplicative error $(1\pm\varepsilon)$. The multiplicative guarantee is stronger, albeit with more space for the algorithm, but the result can be recovered as a corollary to our additive F_2 tracking theorem, which has a much simplified algorithm and analysis compared to [BCIW16].

After some preliminaries, Section 3 presents both algorithms and their analyses. The description of BPTree is split into three parts. Section 4 states and proves the chaining inequality. Appendix A describes a counter example where Pick-and-Drop fails to find an ℓ_2 heavy hitter with probability $1 - 1/\operatorname{poly}(n)$ and explains why simple modifications of the algorithm also fail for ℓ_2 heavy hitters.

1.3 Overview of approach

Here we describe the intuition for our heavy hitters algorithm in the case of a single heavy hitter $H \in [n]$ such that $f_H^2 \geq \frac{9}{10}\|f\|_2^2$. The reduction from multiple heavy hitters to this case is standard. Suppose also for this discussion we knew a constant factor approximation to $F_2 = \|f\|_2^2$. Our algorithm and its analysis use several of the techniques developed in [BCIW16]. We briefly review that algorithm for comparison.

Both CountSieve and BPTree share the same basic building block, which is a subroutine that tries to identify one bit of information about the identity of H. The subroutine hashes the elements of the stream into two buckets, computes one Bernoulli process in each bucket, and then compares the two values. The Bernoulli process is just the inner product of the frequency vector with a vector of Rademacher (i.e. uniform ± 1) random variables. The hope is that the Bernoulli process in the bucket with H grows faster than the other one, so the larger of the two processes reveals which bucket contains H. In order to prove that the process with H grows faster, [BCIW16] introduce a chaining inequality for insertion-only streams that bounds the supremum of the Bernoulli processes over all times. The subroutine essentially gives us a test that H will pass with probability, say, at least 9/10 and that every other items passes with probability at most 6/10. The high-level strategy of both algorithms is to repeat this test sequentially over the stream.

CountSieve uses the subroutine in a two part strategy to identify ℓ_2 heavy hitters with $O(\log\log n)$ memory. The two parts are (1) amplify the heavy hitter so $f_H \geq (1 - \frac{1}{\operatorname{poly}(\log n)})\|f\|_2$ and (2) identify H with independent repetitions of the subroutine. First it winnows the stream from, potentially, n distinct elements to at most $n/\operatorname{poly}(\log n)$ elements. The heavy hitter remains and, furthermore, it becomes $\operatorname{poly}(\log n)$-heavy because many of the other elements are removed. CountSieve accomplishes this by running $\Theta(\log\log n)$ independent copies of the subroutine in parallel, and discarding elements that do not pass a super-majority of the tests. A standard Chernoff bound implies that only $n/2^{O(\log\log n)} = n/\operatorname{poly}(\log n)$ items survive. The second part of the strategy identifies $\Theta(\log n)$ 'break-points' where $\|f\|_2$ of the winnowed stream increases by approximately a $(1 + 1/\log n)$ factor from one break-point to the next. Because H already accounts for nearly all of the value of $\|f\|_2$ it is still a heavy hitter within each of the $\Theta(\log n)$ intervals. CountSieve learns one bit of the identity of H on each interval by running the subroutine.

BPTree merges the two parts of the above strategy. As above, the algorithm runs a series of $\Theta(\log n)$ rounds where the goal of each round is to learn one bit of the identity of H. The difference from CountSieve is that BPTree discards more items after every round, then recurses on learning the remaining bits. As the algorithm proceeds, it discards more and more items and H becomes heavier and heavier in the stream. This is reminiscent of work on adaptive compressed sensing [IPW11], but here we are able to do everything in a single pass given the insertion-only property of the stream. Given that the heavy hitter is even heavier, it allows us to weaken our requirement on the two counters at the next level in the recursion tree: we now allow their suprema to deviate even further from their expectation, and this is precisely what saves us from having to worry that one of the $O(\log n)$ Bernoulli processes that we encounter while walking down the tree will have a supremum which is too large and cause us to follow the wrong path. The fact that the heavy hitter is even heavier also allows us to "use up" even fewer updates to the heavy hitter in the next level of the tree, so that overall we have enough updates to the heavy hitter to walk to the bottom of the tree.

2 Preliminaries

An insertion only stream is a list of items $p_1, p_2, \ldots, p_m \in [n]$. The frequency of j at time t is $f_j^{(t)} := \#\{i \le t \mid p_i = j\}$, $f^{(t)} \in \mathbb{Z}^n_{\ge 0}$ is called the *frequency vector*, we denote $f := f^{(m)}$, $F_2^{(t)} = \sum_{i=1}^n (f_i^{(t)})^2$, $F_2 = \sum_{j=1}^n f_j^2$, and $F_0 = \#\{j \in [n] : f_j > 0\}$. An item $H \in [n]$ is a α-heavy hitter[2] if $f_H^2 \ge \alpha^2 \sum_{j \ne H} f_j^2 = \alpha^2(F_2 - f_H^2)$. In a case where the stream is semi-infinite (it has no defined end) m should be taken to refer to the time of a query of interest. When no time is specified, quantities like F_2 and f refer to the same query time m.

Our algorithms make use of 2-universal (pairwise independent) and 6-wise independent hash functions. We will commonly denote such a function $h : [n] \to [p]$ where p is a prime larger than n, or we may use $h : [n] \to \{0,1\}^R$, which may be taken to mean a function of the first type for some prime $p \in [2^{R-1}, 2^R)$. We use $h(x)_i$ to denote the ith bit, with the first bit most significant (big-endian). A crucial step in our algorithm involves comparing the bits of two values $a, b \in [p]$. Notice that, for any $0 \le r \le \lceil \log_2 p \rceil$, we have $a_i = b_i$, for all $1 \le i \le r$, if and only if $|a - b| < 2^{\lceil \log_2 p \rceil - r}$. Therefore, the test $a_i = b_i$, for all $1 \le i \le r$, can be performed with a constant number of operations.

We will use, as a subroutine, and also compare our algorithm against CountSketch [CCF04]. To understand our results, one needs to know that CountSketch has two parameters, which determine the number of "buckets" and "repetitions" or "rows" in the table it stores. The authors of [CCF04] denote these parameters b and r, respectively. The algorithm selects, independently, r functions $h_1, \ldots, h_r$ from a 2-universal family with domain $[n]$ and range $[b]$ and r functions $\sigma_1, \ldots, \sigma_r$ from a 2-universal family with domain $[n]$ and range $\{-1, 1\}$. CountSketch stores the value $\sum_{j : h_t(j) = i} \sigma(j) f_j$, in cell $(t, i) \in [r] \times [b]$ of the table.

In our algorithm we use the notation $\mathbf{1}(A)$ denote the indicator function of the event A. Namely, $\mathbf{1}(A) = 1$ if A is true and 0 otherwise.

3 Algorithm and analysis

We first describe HH1, formally Algorithm 1, that finds a single $O(1)$-heavy hitter supposing we have an estimate $\sqrt{F_2} \le \sigma \le 2\sqrt{F_2}$ of the second moment of the stream. HH2, Algorithm 2, finds a single $O(1)$-heavy hitter without assuming an estimate of F_2. It repeatedly "guesses" values for σ and restarts HH1 as more items arrive. Finally, Corollary 7 describes BPTree, an algorithm that employs HH2 and a standard hashing technique to find ε-heavy hitters with the same guarantee as CountSketch [CCF04], but requiring only

[2] This definition is in a slightly different form from the one given in the introduction, but this form is more convenient when f_H^2 is very close to F_2.

Algorithm 1 Identify a heavy hitter given $\sigma \in [\sqrt{F_2}, 2\sqrt{F_2}]$.

```
procedure HH1(σ, p_1, p_2, ..., p_m)
    R ← 3⌊log_2(min{n, σ^2} + 1)⌋
    Sample h : [n] → {0,1}^R ∼ 2-wise independent family
    Initialize b = b_1 b_2 ··· b_R = 0 ∈ [2^R], r ← 1,
    Sample Z ∈ {−1,1}^n 6-wise independent, X_0, X_1 ← 0
    β ← 3/4, c ← 1/32, H ← −1
    for t = 1, 2, ..., m and r < R do
        if h(p_t)_i = b_i, for all i ≤ r − 1 then
            H ← p_t
            X_{h(p_t)_r} ← X_{h(p_t)_r} + Z_{p_t}
            if |X_0 + X_1| ≥ cσβ^r then
                Record one bit b_r ← 1(|X_1| > |X_0|)
                Refresh (Z_i)_{i=1}^n, X_0, X_1 ← 0
                r ← r + 1
            end if
        end if
    end for
    return H
end procedure
```

$O(\varepsilon^{-2}\log\varepsilon^{-1})$ words of space and $O(\log\varepsilon^{-1})$ update time. In comparison, CountSketch uses $O(\varepsilon^{-2}\log n)$ words and has $O(\log n)$ update time.[3] Note that the authors of [CCF04] call the parameter k (as in Top-k) whereas we use ε^2 for the same value. The present algorithm is also more efficient than CountSieve of [BCIW16], which uses $O(\varepsilon^{-2}(\log\varepsilon^{-1})(\log\log n))$ words and $O(e^{O((\log\log n)^2)})$ update time (owing to their use of the JL generator from [KMN11]) for the same guarantee.

The algorithm begins with randomizing the item labels by replacing them with pairwise independent values on $R = \Theta(\log\min\{n,\sigma^2\})$ bits, via the hash function h. Since n and $\sigma^2 \geq F_2$ are both upper bounds for the number of distinct items in the stream, R can be chosen so that every item recieves a distinct hash value. We recommend choosing a prime $p \approx \min\{n, F_2\}^2$ and assigning the labels $h(i) = a_0 + a_1 i \bmod p$, for a_0 and a_1 randomly chosen in $\{0, 1, \ldots, p-1\}$ and $a_1 \neq 0$. We can always achieve this with $R = 3\log_2(\min\{n, F_2\}+1)$, which is convenient for the upcoming analysis. This distribution on h is known to be a 2-wise independent family [CW79]. Note computing $h(i)$ for any i takes $O(1)$ time. It is also simple to invert: namely $x = a_1^{-1}(h(x) - a_0) \bmod p$, so x can be computed quickly from $h(x)$ when $p > n$. Inverting requires computing the inverse of a_1 modulo p, which takes $O(\log\min\{n, F_2\})$ time via repeated squaring, however this computation can be done once, for example during initialization of the algorithm, and the result stored for all subsequent queries. Thus, the time to compute $a_1^{-1} \bmod p$ is negligible in comparison to reading the stream.

Once the labels are randomized, HH1 proceeds in rounds wherein one bit of the randomized label of the heavy hitter is determined during each round. As the rounds proceed, items are discarded from the stream. The remaining items are called *active*. When the algorithm discards an item it will never reconsider it (unless the algorithm is restarted). In each round, it creates two Bernoulli processes X_0 and X_1. In the rth round, X_0 will be determined by the active items whose randomized labels have their rth bit equal to 0, and X_1 determined by those with rth bit 1. Let $f_0^{(t)}, f_1^{(t)} \in \mathbb{Z}_{\geq 0}^n$ be the frequency vectors of the active items in each category, respectively, initialized to 0 at the beginning of the round. Then the Bernoulli processes are $X_0^{(t)} = \langle Z, f_0^{(t)}\rangle$ and $X_1^{(t)} = \langle Z, f_1^{(t)}\rangle$, where Z is a vector of 6-wise independent Rademacher random variables (i.e. the Z_i are marginally uniformly random in $\{-1, 1\}$).

[3] This assumes $O(1)$ time for dictionary look-up, insert, and delete, which can be achieved in expectation using e.g. hashing with chaining [CLRS09, Section 11.2].

The rth round ends when $|X_0 + X_1| > c\sigma\beta^{r-1}$, for specified[4] constants c and β. At this point, the algorithm compares the values $|X_0|$ and $|X_1|$ and records the identity of the larger one as the rth bit of the candidate heavy hitter. All those items with rth bit corresponding to the smaller counter are discarded (made inactive), and the next round is started.

After R rounds are completed, if there is a heavy hitter then its randomized label will be known with good probability. The identity of the item can be determined selecting an item in the stream that passes all of the R bit-wise tests, or by inverting the hash function used for the label, as described above. If it is a $O(1)$-heavy hitter then the algorithm will find it with probability at least $2/3$. The algorithm is formally presented in Algorithm 1.

The key idea behind the algorithm is that as we learn bits of the heavy hitter and discard other items, it becomes easier to learn additional bits of the heavy hitter's identity. With fewer items in the stream as the algorithm proceeds, the heavy hitter accounts for a larger and larger fraction of the remaining stream as time goes on. As the heavy hitter gets heavier the discovery of the bits of its identity can be sped up. When the stream does not contain a heavy hitter this acceleration of the rounds might not happen, though that is not a problem because when there is no heavy hitter the algorithm is not required to return any output. Early rounds will each use a constant fraction of the updates to the heavy hitter, but the algorithm will be able to finish all $R = \Theta(\log n)$ rounds because of the speed-up. The parameter β controls the speed-up of the rounds. Any value of $\beta \in (\frac{1}{2}, 1)$ can be made to work (possibly with an adjustment to c), but the precise value affects the heaviness requirement and the failure probability.

The final algorithm, which removes the assumption of knowing a value $\sigma \in [\sqrt{F_2}, 2\sqrt{F_2}]$, is described in Algorithm 2. It proceeds by sequentially guessing σ, and it requires about a factor 2 increase in the heaviness because it will lose some items before arriving at good value of σ. We also describe an algorithm that hashes the items into $O(1/\varepsilon^2)$ buckets, like a CountSketch does, for $\log(1/\varepsilon)$ repetitions. This brings the heaviness requirement down to $f_H^2 \geq \varepsilon^2 F_2$ and leads to the space bound given in Corollary 7.

3.1 Identifying a single heavy hitter given an approximation to F_2

In this section we use $H \in [n]$ to stand for the identity of the most frequent item in the stream. It is not assumed to be a heavy hitter unless explicitly stated. For each $r \geq 0$, let

$$\mathcal{H}_r \coloneqq \{i \in [n] \setminus \{H\} \mid h(i)_k = h(H)_k \text{ for all } 1 \leq k \leq r\},$$

and let $\bar{\mathcal{H}}_r \coloneqq \mathcal{H}_{r-1} \setminus \mathcal{H}_r$, with $\bar{\mathcal{H}}_0 = \emptyset$ for convenience. By definition, $\mathcal{H}_R \subseteq \mathcal{H}_{R-1} \subseteq \cdots \subseteq \mathcal{H}_0 = [n] \setminus \{H\}$, and, in round $r \in [R]$, our hope is that the active items are those in $\mathcal{H}_{r-1}$ so that one of the variables X_0, X_1 depends on the frequencies of items in $\mathcal{H}_r$, while the other depends on $\bar{\mathcal{H}}_r$.

For $W \subseteq [n]$, denote by $f^{(t)}(W) \in \mathbb{Z}_{\geq 0}^n$ the frequency vector at time t of the stream restricted to the items in W, that is, a copy of $f^{(t)}$ with the ith coordinate replaced by 0 for every $i \notin W$. We also define $f^{(s:t)}(W) \coloneqq f^{(t)}(W) - f^{(s)}(W)$ and $F_2(W) = \sum_{j \in W} f_j^2$. In our notation $F_2 - f_H^2 = F_2(\mathcal{H}_0)$.

Lemma 1. *For any $r \in \{0, 1, \ldots, R\}$ and $K > 0$, the events*

$$E_{2r-1} \coloneqq \left\{ \max_{s,t \leq m} |\left\langle Z, f^{(s:t)}(\bar{\mathcal{H}}_r) \right\rangle| \leq K F_2(\mathcal{H}_0)^{1/2} \right\}$$
$$E_{2r} \coloneqq \left\{ \max_{s,t \leq m} |\left\langle Z, f^{(s:t)}(\mathcal{H}_r) \right\rangle| \leq K F_2(\mathcal{H}_0)^{1/2} \right\}.$$

have respective probabilities at least $1 - \frac{4C_*}{K2^r}$ of occurring, where $C_* < 34$ is the constant from Theorem 10.

[4] $c = 1/32$ and $\beta = 3/4$ would suffice.

Proof. By the Law of Total Probability and Theorem 10 with Markov's Inequality we have

$$\begin{aligned}
&\Pr\left(\max_{t\le m}|\langle Z, f^{(t)}(\mathcal{H}_r)\rangle| \ge \frac{1}{2}KF_2(\mathcal{H}_0)^{1/2}\right)\\
&\quad = \mathbb{E}\left\{\Pr\left(\max_{t\le m}|\langle Z, f^{(t)}(\mathcal{H}_r)\rangle| \ge \frac{1}{2}KF_2(\mathcal{H}_0)^{1/2}\middle|\mathcal{H}_r\right)\right\}\\
&\quad \le \mathbb{E}\left\{\frac{2C_*F_2(\mathcal{H}_r)^{1/2}}{KF_2(\mathcal{H}_0)^{1/2}}\right\}\\
&\quad \le \frac{2C_*F_2(\mathcal{H}_0)^{1/2}}{KF_2(\mathcal{H}_0)^{1/2}2^r},
\end{aligned}$$

where the last inequality is Jensen's. The same holds if $\mathcal{H}_r$ is replaced by $\bar{\mathcal{H}}_r$.

Applying the triangle inequality to get $|\langle Z, f^{(s:t)}(\mathcal{H}_r)\rangle| \le |\langle Z, f^{(s)}(\mathcal{H}_r)\rangle| + |\langle Z, f^{(t)}(\mathcal{H}_r)\rangle|$ we then find $P(E_{2r}) \ge 1 - \frac{4C_*}{K2^r}$. A similar argument proves $P(E_{2r-1}) \ge 1 - \frac{4C_*}{K2^r}$. □

Let U be the event $\{h(j) \ne h(H)$ for all $j \ne H, f_j > 0\}$ which has, by pairwise independence, probability $\Pr(U) \ge 1 - F_0 2^{-R} \ge 1 - \frac{1}{\min\{n,F_2\}^2}$, recalling that $F_0 \le \min\{n, F_2\}$ is the number of distinct items appearing before time m.

Lemma 2. *Let $K' \ge 100$. If $F_2^{1/2} \le \sigma \le 2F_2^{1/2}$ and $f_H > 2K'C_*\sqrt{F_2(\mathcal{H}_0)}$ then, with probability at least $1 - \frac{1}{\min\{F_2,n\}^2} - \frac{8}{K'c(2\beta-1)}$ the algorithm HH1 returns H.*

Proof. Recall that H is *active* during round r if it happens that $h(H)_i = b_i$, for all $1 \le i \le r-1$, which implies that updates from H are not discarded by the algorithm during round r. Let $K = K(r) = K'cC_*\beta^r$ in Lemma 1, and let E be the event that U and $\cap_{r=1}^{2R} E_r$ both occur. We prove by induction on r that if E occurs then either $b_r = h(H)_r$, for all $r \in [R]$ or H is the only item appearing in the stream. In either case, the algorithm correctly outputs H, where in the former case it follows because $E \subseteq U$.

Let $r \ge 1$, be such that H is still active in round r, i.e., $b_i = h(H)_i$ for all $1 \le i \le r-1$. Note that all items are active in round 1. Since H is active, the remaining active items are exactly $\mathcal{H}_{r-1} = \mathcal{H}_r \cup \bar{\mathcal{H}}_r$. Let t_r denote the time of the last update received during the rth round, and define $t_0 = 0$. At time $t_{r-1} \le t < t_r$ we have

$$\begin{aligned}
c\sigma\beta^r &> |X_0 + X_1|\\
&= |\langle Z, f^{(t_{r-1}:t)}(\mathcal{H}_r \cup \bar{\mathcal{H}}_r)\rangle + Z_H f_H^{(t_{r-1}:t)}|\\
&\ge f_H^{(t_{r-1}:t)} - K(r-1)F_2(\mathcal{H}_0)^{1/2},
\end{aligned}$$

where the last inequality follows from the definition of $E_{2(r-1)}$. Rearranging and using the heaviness assumption on f_H, we get $2\max C_*\sqrt{F_2(\mathcal{H}_0)} < \sqrt{F_2} \le \sigma$ which implies the bound

$$K(r-1)F_2(\mathcal{H}_0)^{1/2} < \frac{K(r-1)}{2K'C_*}\sigma = \frac{1}{2}c\sigma\beta^{r-1}. \tag{1}$$

Therefore, by rearranging we see $f_H^{(t_{r-1}:t_r)} \le 1 + f_H^{(t_{r-1}:t)} < 1 + \frac{3}{2}c\sigma\beta^{r-1}$. That implies

$$f_H^{(t_r)} = \sum_{k=1}^{r} f_H^{(t_{k-1}:t_k)} < r + \frac{3}{2}c\sigma\sum_{k=1}^{r}\beta^{k-1} \le r + \frac{3c}{1-\beta}\sqrt{F_2}.$$

Thus, if $f_H > R + \frac{3c}{1-\beta}\sqrt{F_2}$ then round $r \le R$ is guaranteed to be completed and a further update to H appears after the round. Suppose, that is not the case, and rather $R \ge f_H - \frac{3c}{1-\beta}\sqrt{F_2} \ge \frac{1}{2}\sqrt{F_2}$, where the last inequality follows from our choices $K' \ge 100$, $\beta = 3/4$, and $c = 1/32$. Then, by the definition of R, $9(1 + \log_2 4F_2)^2 \ge R^2 \ge \frac{1}{4}F_2$. One can check that this inequality implies that $F_2 \le 10^4$, hence $f_H < 100$.

Now $K' \geq 100$ and the heaviness requirement of H implies that $F_2(\mathcal{H}_0) = 0$. Therefore, H is the only item in the stream, and, in that case the algorithm will always correctly output H.

Furthermore, at the end of round r, $|X_0 + X_1| \geq c\sigma\beta^r$, so we have must have either $|X_0| \geq c\sigma\beta^r/2$ or $|X_1| \geq c\sigma\beta^r/2$. Both cannot occur for the following reason. The events E_{2r-1} and E_{2r} occur, recall these govern the non-heavy items contributions to X_0 and X_1, and these events, with the inequality (1), imply

$$|\langle Z, f^{(t_{r-1}:t_r)}(\mathcal{H}_r)\rangle| \leq K(r)F_2(\mathcal{H}_0)^{1/2} < \frac{1}{2}c\sigma\beta^r$$

and the same holds for $\bar{\mathcal{H}_r}$. Therefore, the Bernoulli process not including H has its value smaller than $c\sigma\beta^r/2$, and the other, larger process identifies the bit $h(H)_r$. By induction, the algorithm completes every round $r = 1, 2, \ldots, R$ and there is at least one update to H after round R. This proves the correctness of the algorithm assuming the event E occurs.

It remains to compute the probability of E. Lemma 1 provides the bound

$$\begin{aligned}
\Pr(U \text{ and } \cap_{i=1}^{2R} E_i) &\geq 1 - \frac{1}{\min\{n, F_2\}^2} - \sum_{r=0}^{R} \frac{8C_*}{K(r)2^r} \\
&= 1 - \frac{1}{\min\{n, F_2\}^2} - \sum_{r=0}^{R} \frac{8}{K'c\beta^r 2^r} \\
&> 1 - \frac{1}{\min\{n, F_2\}^2} - \frac{8}{K'c(2\beta - 1)}.
\end{aligned}$$

□

Theorem 3 (HH1 Correctness)**.** *There is a constant K such that if H is a K-heavy hitter and $\sqrt{F_2} \leq \sigma \leq 2\sqrt{F_2}$, then with probability at least $2/3$ algorithm HH1 returns H. HH1 uses $O(1)$ words of storage.*

Proof. The Theorem follows immediately from Lemma 2 by setting $c = 1/32$, $\beta = 3/4$, and $K' = 2^{11}$, which allows $K = 2^{12}C_* \leq 140{,}000$. □

3.2 F_2 Tracking

The step in HH2 that "guesses" an approximation σ for $\sqrt{F_2}$ works as follows. We create an estimator $\hat{F}_2$ to (approximately) track F_2. We set a series of increasing thresholds and restart HH1 each time $\hat{F}_2$ crosses a threshold with the value σ depending on the threshold. At least one of the thresholds will be the "right" one, in the sense that HH1 gets initialized with σ in the interval $[\sqrt{F_2}, 2\sqrt{F_2}]$, so we expect the corresponding instance of HH1 to identify the heavy hitter, if one exists.

Algorithm HH1 could fail if $\hat{F}_2$ is wildly inaccurate at the time it crosses the right threshold, because it might be initialized . It is not hard to guarantee that $\hat{F}_2 \gtrsim F_2$ when the threshold is crossed—that can be achieved using the AMS F_2 estimator [AMS99]—but the reverse inequality is nontrivial. We will use a modified version of the AMS estimator given below.

Let $N \in \mathbb{N}$, Z^j be a vector of 6-wise independent Rademacher random variables for $j \in N$, and define $X_{j,t} = \langle Z^j, f^{(t)}\rangle$. Let $Y_t = \frac{1}{N}\sum_{j=1}^{N} X_{j,t}^2$, obviously Y_t can be computed by a streaming algorithm.

Theorem 4. *Let $0 < \varepsilon < 1$. There is a streaming algorithm that outputs at each time t a value $\hat{F}_2^{(t)}$ such that*

$$\Pr(|\hat{F}_2^{(t)} - F_2^{(t)}| \leq \varepsilon F_2, \textit{ for all } 0 \leq t \leq m) \geq 1 - \delta.$$

The algorithm use $O(\frac{1}{\varepsilon^2}\log n \log \frac{1}{\delta\varepsilon})$ bits of storage and has $O(\frac{1}{\varepsilon^2}\log \frac{1}{\delta\varepsilon})$ update time.

The proof of Theorem 4 uses the following technical lemma that bounds the divergence the estimate over an entire interval of updates. It follows along the lines of Lemma 22 from the full version of [BCIW16].

Lemma 5. *Let $\varepsilon < 1$, $N \geq 12/\varepsilon^2$, and $\Delta > 0$. If $F_2^{(v)} - F_2^{(u)} \leq (\frac{\varepsilon}{20NC_*})^2 F_2$, then*

$$\Pr\left(Y_t - F_2^{(t)} \leq 2\varepsilon F_2,\, \forall t \in [u,v]\right) \geq \frac{2}{3}.$$

Proof. We denote $\Delta = (F_2^{(v)} - F_2^{(u)})/F_2$ and split the expression above as follows

$$Y_t - F_2^{(t)} = (Y_t - Y_u) + (Y_u - F_2^{(u)}) + (F_2^{(t)} - F_2^{(u)}).$$

Let $b_1 > 0$. For the second term, and it is shown in [AMS99] that

$$\Pr(Y_u - F_2^{(u)} \geq b_1) \leq \frac{2(F_2^{(u)})^2}{Nb_1^2}.$$

Let $X^{(t)} = \frac{1}{\sqrt{N}}(X_{1,t}, X_{2,t}, \ldots, X_{N,t})$ and $X^{(u:t)} = X_t - X_u$, in particular $Y_t = \|X^{(t)}\|_2^2$. For the first term, we have

$$Y_t = \|X^{(u)} + X^{(u:t)}\|_2^2 \leq \left(\|X^{(u)}\|_2 + \|X^{(u:t)}\|_2\right)^2,$$

so

$$Y_t - Y_u \leq 2\sqrt{Y_u}\|X^{(u:t)}\|_2 + \|X^{(u:t)}\|_2^2.$$

Now from Theorem 10 and a union bound it follows that for $b_2 > 0$

$$P\left(\sup_{j\in[N], u\leq t\leq v} |X_{j,t} - X_{j,u}| \geq b_2\right) \leq \frac{NC_*\|f^{(u:v)}\|_2}{b_2},$$

so $P(\sup_{t\geq u}\|X^{(u:t)}\|_2 \geq b_2) \leq NC_*\|f^{(u:v)}\|_2/b_2$.

With probability at least $1 - \frac{2(F_2^{(u)})^2}{Nb_1^2} - \frac{NC_*\|f^{(u:v)}\|_2}{b_2}$ we get, for all $t \geq u$

$$Y_t - F_2^{(t)} \leq 2(F_2^{(u)} + b_1)^{\frac{1}{2}} b_2 + b_2^2 + b_1 + \Delta F_2.$$

Now we set $b_1 = \varepsilon F_2$ and $b_2 = 6NC_*\sqrt{\Delta F_2}$ and the above expression is bounded by

$$\begin{aligned} Y_t - F_2^{(t)} &\leq 12\sqrt{2}NC_* F_2\sqrt{\Delta} + \varepsilon F_2 + 2\Delta \\ &\leq 20NC_* F_2\sqrt{\Delta} + \varepsilon F_2, \\ &\leq 2\varepsilon F_2 \end{aligned}$$

since $\Delta \leq F_2$. The probability of success is at least

$$1 - \frac{2(F_2^{(u)})^2}{N\varepsilon^2 F_2^2} - \frac{\|f^{(u:m)}\|_2}{6\sqrt{\Delta}} \leq 1 - \frac{1}{3}.$$

□

We now proceed to describe the F_2 estimation algorithm and prove Theorem 4.

of Theorem 4. The algorithm returns at each time t the value

$$\hat{F}_2(t) = \max_{s\leq t} \operatorname{median}(Y_{1,t}, Y_{2,t}, \ldots, Y_{M,t}),$$

which are $M = \Theta(\log\frac{1}{\delta\varepsilon})$ independent copies of Y_t with $N = 12(\frac{3}{\varepsilon})^2$. Let $\Delta = (\frac{\varepsilon/3}{20NC_*})^2$ and define $t_0 = 0$ and $t_k = \max\{t \leq m \mid F_2^{(t_k)} \leq F_2^{(t_{k-1})} + \Delta F_2\}$, for $k \geq 1$. These times separate the stream into no more than

Algorithm 2 Identify a heavy hitter by guessing σ.

```
procedure HH2(p_1, p_2, ..., p_m)
    Run F̂_2 from Theorem 4 with ε = 1/100 and δ = 1/20
    Start HH1 (1, p_1, ..., p_m)
    Let t_0 = 1 and t_k = min{t | F̂_2^(t) ≥ 2^k}, for k ≥ 1.
    for each time t_k do
        Start HH1((F̂_2^(t_k))^(1/2), p_{t_k}, p_{t_k+1}, ... p_m)
        Let H_k denote its ouput if it completes
        Discard H_{k-2} and the copy of HH1 started at t_{k-2}
    end for
    return H_{k-1}
end procedure
```

$2/\Delta$ intervals during which the second moment increases no more than ΔF_2. Lemma 5 and an application of Chernoff's Inequality imply that for each interval k

$$\Pr\left(\text{median}_{j\in[M]}(Y_{j,t}) - F_2^{(t)} \leq \frac{2}{3}\varepsilon F_2,\ \forall t \in [t_{k-1}, t_k)\right)$$
$$\geq 1 - \text{poly}(\delta\varepsilon).$$

The original description of the AMS algorithm [AMS99] implies that, for all k,

$$\Pr\left(\text{median}_{j\in[M]}(Y_{j,t_k}) \geq (1-\varepsilon/3)F_2^{(t)}\right) \geq 1 - \text{poly}(\delta\varepsilon).$$

By choosing the constants appropriately and applying a union bound over all $O(\varepsilon^{-2})$ intervals and endpoints we achieve all of these events events occur with probability at least $1-\delta$. One easily checks that this gives the desired guarantee. □

Let us remark that using $O(\varepsilon^{-2}(\log\log n + \log\varepsilon^{-1}))$ words one can achieve a $(1 \pm \varepsilon)$ *multiplicative* approximation to F_2 at all points in the stream, in contrast to the additive εF_2 approximation of Theorem 4. The only change needed is to increase M to $\Theta(\log\log(n) + \log\frac{1}{\varepsilon})$ independent copies of Y_t. The proof that this works runs along the lines of the proof of Theorem 4, but instead of breaking the stream into $O(1/\varepsilon^2)$ intervals with equal change in F_2, break the stream into $O(\frac{1}{\varepsilon^2}\log n)$ intervals of where the change in F_2 is geometrically increasing size. That is also the approach taken by [BCIW16], which also achieves the same space, up to constant factors, for a $(1\pm\varepsilon)$-approximation at all times, but the details of the Bernoulli processes presented here, i.e., with 6-wise independence, are much simpler to implement.

3.3 The complete heavy hitters algorithm

This section describes BPTree, our main heavy hitters algorithm. We first prove the correctness of HH2, formally given in Algorithm 2, which accomplishes the guessing step. The algorithm sets thresholds at $1, 2, \ldots, 2^k, \ldots$ and starts a new instance of HH1 each time the estimate $\hat{F}_2$ described in the previous section crosses one of the thresholds. Each new instance is initialized with the current value of $\sqrt{F_2}$ as the value for σ. It maintains only the two most recent copies of HH1, so even though overall it may instantiate $\Omega(\log n)$ copies of HH1 at most two will running concurrently and the total storage remains $O(1)$ words.

Theorem 6. *There exists a constant $K > 0$ and a 1-pass streaming algorithm HH2, Algorithm 2, such that if the stream contains a K-heavy hitter then with probability at least 0.6 HH2 returns the identity of the heavy hitter. The algorithm uses $O(1)$ words of memory and $O(1)$ update time.*

Proof. The space and update time bounds are immediate from the description of the algorithm. The success probability follows by a union bound over the failure probabilities in Theorems 3 and 4, which are 1/3 and

$\delta = 0.05$ respectively. It remains to prove that there is a constant K such that conditionally given the success of the F_2 estimator, the hypotheses of Theorem 3 are satisfied by the penultimate instance of HH1 by HH2.

Let K' denote the constant from Theorem 3 and set $K = 12K'$, so if H is a K-heavy hitter then for any $\alpha > 0$ such that $\alpha\sqrt{F_2} \geq 2$ and in any interval $(t, t']$ where $(F_2^{(t')})^{1/2} - (F_2^{(t)})^{1/2} \geq \alpha\sqrt{F_2}$ we will have

$$f_H^{(t:t')} + F_2(\mathcal{H}_0)^{1/2} > \|f^{(t:t')}\|_2 > \|f^{(t')}\|_2 - \|f^{(t)}\|_2 \geq \alpha\sqrt{F_2}.$$

If follows with in the stream $p_t, p_{t+1}, \ldots, p_{t'}$ the heaviness of H is at least

$$(\alpha\sqrt{F_2} - \sqrt{F_2(\mathcal{H}_0)})/\sqrt{F_2(\mathcal{H}_0)} \geq 6K'\alpha. \tag{2}$$

Let k be the last iteration of HH2. By the definition of t_k, we have $(\hat{F}_2^{(t_{k-1})})^{1/2} \geq \frac{1}{4}(\hat{F}_2)^{1/2} \geq \frac{1}{4}\sqrt{(1-\varepsilon)F_2}$. Similar calculations show that there exists a time $t_* > t_{k-1}$ such that $(F_2^{(t_*)})^{1/2} - (F_2^{(t_{k-1})})^{1/2} \geq F_2/6$ and $\|f^{t_{k-1}:t_*}\|_2 \leq (\hat{F}_2^{(t_{k-1})})^{1/2} \leq 2\|f^{t_{k-1}:t_*}\|_2$. Furthermore, H is a K' heavy hitter on the substream $p_{t_{k-1}}, p_{t_{k-1}+1}, \ldots, p_{t_*}$ by (2). This proves that the hypotheses of Theorem 3 are satisfied. It follows that from Theorem 3 that HH1 correctly identifies $H_{k-1} = H$ on that substream and the remaining updates in the interval $(t_*, t_m]$ do not affect the outcome. □

Corollary 7. *For any $\varepsilon > 0$ there is 1-pass streaming algorithm BPTree that, with probability at least $(1-\delta)$, returns a set of $\frac{\varepsilon}{2}$-heavy hitters containing every ε-heavy hitter and an approximate frequency for every item returned satisfying the $(\varepsilon, 1/\varepsilon^2)$-tail guarantee. The algorithm uses $O(\frac{1}{\varepsilon^2}(\log\frac{1}{\varepsilon\delta})(\log n + \log m))$ bits of space and has $O(\log\frac{1}{\varepsilon\delta})$ update and $O(\varepsilon^{-2}\log\frac{1}{\varepsilon\delta})$ retrieval time.*

Proof. The algorithm BPTree constructs a hash table in the same manner as CountSketch where the items are hashed into $b = O(1/\varepsilon^2)$ buckets for $r = O(\log 1/\varepsilon\delta)$ repetitions. On the stream fed into each bucket we run an independent copy of HH2. A standard $r \times b$ CountSketch is also constructed. The constants are chosen so that when an ε-heavy hitter in the stream is hashed into a bucket it becomes a K-heavy hitter with probability at least 0.95. Thus, in any bucket with a the ε-heavy hitter, the heavy hitter is identified with probability at least 0.55 by Theorem 6 and the aforementioned hashing success probability.

At the end of the stream, all of the items returned by instances of HH2 are collected and their frequencies checked using the CountSketch. Any items that cannot be ε-heavy hitters are discarded. The correctness of this algorithm, the bound on its success probability, and the $(\varepsilon, 1/\varepsilon^2)$-tail guarantee follow directly from the correctness of CountSketch and the fact that no more than $O(\varepsilon^{-2}\log(1/\delta\varepsilon))$ items are identified as potential heavy hitters. □

Remark 8. *We can amplify the success probability of HH2 to any $1-\delta$ by running $O(\log(1/\delta))$ copies in parallel and taking a majority vote for the heavy hitter. This allows one to track $O(1)$-heavy hitters at all points in the stream with an additional $O(\log\log m)$ factor in space and update time. The reason is because there can be a succession of at most $O(\log m)$ 2-heavy hitters in the stream, since their frequencies must increase geometrically, so setting $\delta = \Theta(1/\log m)$ is sufficient. The same scheme works for BPTree tree, as well, and if one replaces each of the counters in the attached CountSketch with an F_2-at-all-times estimator of [BCIW16] then one can track the frequencies of all ε-heavy hitters at all times as well. The total storage becomes $O(\frac{1}{\varepsilon^2}(\log\log n + \log\frac{1}{\varepsilon}))$ words and the update time is $O(\log\log n + \log\frac{1}{\varepsilon})$.*

4 Chaining

Let $Z \in \{-1,1\}^n$ be random. We are interested in upper-bounding $\mathbb{E}\sup_t |\langle f^{(t)}, Z\rangle|$. It was shown in [BCIW16] that if each entry in Z is drawn independently and uniformly from $\{-1,1\}$, then $\mathbb{E}\sup_t |\langle f^{(t)}, Z\rangle| \lesssim \|f^{(m)}\|_2$. We show that this inequality still holds if the entries of Z are drawn from a 6-wise independent family, which is used both in our analyses of HH1 and our F_2 tracking algorithm.

The following is implied by [Haa82].

Lemma 9 (Khintchine's inequality)**.** *Let $Z \in \{-1,1\}^n$ be chosen uniformly at random, and $x \in \mathbb{R}^n$ a fixed vector. Then for any even integer p, $\mathbb{E}\,\langle Z, x\rangle^p \le \sqrt{p}^p \cdot \|x\|_2^p$.*

We now prove the main theorem of this section.

Theorem 10. *If $Z \in \{-1,1\}^n$ is drawn from a 6-wise independent family, $\mathbb{E}\sup_t |\langle f^{(t)}, Z\rangle| < 34 \cdot \|f^{(m)}\|_2$.*

Proof. To simplify notation, we first normalize the vectors in $\{f^{(0)} = 0, f^{(1)}, \dots, f^{(m)}\}$ (i.e., divide by $\|f^{(m)}\|_2$). Denote the set of these normalized vectors by $T = \{v_0, \dots, v_m\}$, where $\|v_m\|_2 = 1$. For every $k \in \mathbb{N}$, we can find a $1/2^k$-net of T in ℓ_2 with size $|S_k| \le 2^{2k}$ by a greedy construction as follows. (Recall: an ε-net of some set of points T under some metric d is a set of point T' such that for each $t \in T$, there exists some $t' \in T'$ such that $d(t,t') \le \varepsilon$.)

To construct an ε-net for T, we first take v_0, then choose the smallest i such that $\|v_i - v_0\|_2 > \varepsilon$, and so on. To prove the number of elements selected is upper bounded by $1/\varepsilon^2$, let $u_0, u_1, u_2, \dots, u_t$ denote the vectors we selected accordingly, and note that the second moments of $u_1 - u_0, u_2 - u_1, \dots, u_t - u_{t-1}$ are greater than ε^2. Because the vectors $u_i - u_{i-1}$ have non-negative coordinates, $\|u_t\|_2^2$ is lower bounded by the summation of these moments, while on the other hand $\|u_t\|_2^2 \le 1$. Hence the net is of size at most $1/\varepsilon^2$.

Let S be a set of vectors. Let $Z \in \{-1,1\}^n$ be drawn from a p-wise independent family, where p is an even integer. By Markov and Khintchine's inequality,

$$\Pr(|\langle x, Z\rangle| > \lambda \cdot |S|^{1/p} \cdot \|x\|_2) < \frac{\mathbb{E}|\langle x, Z\rangle|^p}{\lambda^p \cdot |S| \cdot \|x\|_2^p} < \frac{1}{|S|} \cdot \left(\frac{\sqrt{p}}{\lambda}\right)^p.$$

Hence,

$$\begin{aligned}
\mathbb{E}\sup_{x\in S} |\langle x, Z\rangle| &= \int_0^\infty \Pr(\sup_{x\in S} |\langle x, Z\rangle| > u) du \\
&= |S|^{1/p} \cdot \sup_{x\in S} \|x\|_2 \cdot \\
&\quad \int_0^\infty \Pr(\sup_{x\in S} |\langle x, Z\rangle| > \lambda \cdot |S|^{1/p} \cdot \sup_{x\in S} \|x\|_2) d\lambda \\
&< |S|^{1/p} \cdot \sup_{x\in S} \|x\|_2 \cdot \left(\sqrt{p} + \int_{\sqrt{p}}^\infty \left(\frac{\sqrt{p}}{\lambda}\right)^p d\lambda\right) \\
&\quad \text{(union bound)} \\
&= |S|^{1/p} \cdot \sup_{x\in S} \|x\|_2 \cdot \sqrt{p} \cdot \left(1 + \frac{1}{p-1}\right)
\end{aligned}$$

Now we apply a similar chaining argument as in the proof of Dudley's inequality (cf. [Dud67]). For $x \in T$, let x^k denote the closest point to x in S_k. Then $\|x^k - x^{k-1}\|_2 \le \|x^k - x\|_2 + \|x - x^{k-1}\|_2 \le (1/2^k) + (1/2^{k-1})$. Note that the size of $\{x^k - x^{k-1} | x \in T\}$ is upper bounded by $|S_k| \cdot |S_{k-1}| \le 2^{4k}$. Therefore for $p = 6$,

$$\begin{aligned}
\mathbb{E}\sup_{x\in T} |\langle x, Z\rangle| &\le \sum_{k=1}^\infty \mathbb{E}\sup |\langle x^k - x^{k-1}, Z\rangle| \\
&< 3\sqrt{p}\left(1 + \frac{1}{p-1}\right) \sum_{k=1}^\infty (2^{4k})^{1/p} \cdot (1/2^k) \\
&< 34.
\end{aligned}$$

□

References

[AMS99] Noga Alon, Yossi Matias, and Mario Szegedy. The space complexity of approximating the frequency moments. *J. Comput. Syst. Sci.*, 58(1):137–147, 1999.

[BBD$^+$02] Brian Babcock, Shivnath Babu, Mayur Datar, Rajeev Motwani, and Jennifer Widom. Models and issues in data stream systems. In *Proceedings of the Twenty-first ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems (PODS)*, pages 1–16, 2002.

[BCIS09] Radu Berinde, Graham Cormode, Piotr Indyk, and Martin J. Strauss. Space-optimal heavy hitters with strong error bounds. In *Proceedings of the Twenty-Eigth ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems (PODS)*, pages 157–166, 2009.

[BCIW16] Vladimir Braverman, Stephen R. Chestnut, Nikita Ivkin, and David P. Woodruff. Beating CountSketch for Heavy Hitters in Insertion Streams. In *Proceedings of the 48th Annual ACM Symposium on Theory of Computing (STOC), to appear*, 2016. Full version at arXiv abs/1511.00661.

[BDW16] Arnab Bhattacharyya, Palash Dey, and David P. Woodruff. An optimal algorithm for ℓ_1-heavy hitters in insertion streams and related problems. *CoRR*, abs/1511.00661, 2016.

[BM81] Robert S. Boyer and J. Strother Moore. A fast majority vote algorithm. Technical Report Technical Report ICSCA-CMP-32, Institute for Computer Science, University of Texas, 1981.

[BM91] Robert S. Boyer and J. Strother Moore. MJRTY: A fast majority vote algorithm. In *Automated Reasoning: Essays in Honor of Woody Bledsoe*, pages 105–118, 1991.

[BO13] Vladimir Braverman and Rafail Ostrovsky. Approximating large frequency moments with pick-and-drop sampling. In *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques*, pages 42–57. Springer, 2013.

[CCF04] Moses Charikar, Kevin C. Chen, and Martin Farach-Colton. Finding frequent items in data streams. *Theor. Comput. Sci.*, 312(1):3–15, 2004.

[CCM10] Amit Chakrabarti, Graham Cormode, and Andrew McGregor. A near-optimal algorithm for estimating the entropy of a stream. *ACM Transactions on Algorithms*, 6(3), 2010.

[CLRS09] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. *Introduction to Algorithms*. MIT Press, 3rd edition, 2009.

[CM05] Graham Cormode and S. Muthukrishnan. An improved data stream summary: the count-min sketch and its applications. *J. Algorithms*, 55(1):58–75, 2005.

[CW79] Larry Carter and Mark N. Wegman. Universal classes of hash functions. *J. Comput. Syst. Sci.*, 18(2):143–154, 1979.

[DLM02] Erik D. Demaine, Alejandro López-Ortiz, and J. Ian Munro. Frequency estimation of Internet packet streams with limited space. In *Proceedings of the 10th Annual European Symposium on Algorithms (ESA)*, pages 348–360, 2002.

[Dud67] Richard M. Dudley. The sizes of compact subsets of Hilbert space and continuity of Gaussian processes. *J. Functional Analysis*, 1:290–330, 1967.

[GR09] Parikshit Gopalan and Jaikumar Radhakrishnan. Finding duplicates in a data stream. In *Proceedings of the Twentieth Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 402–411, 2009.

[Haa82] Uffe Haagerup. The best constants in the Khintchine inequality. *Studia Math.*, 70(3):231–283, 1982.

[HNO08] Nicholas J. A. Harvey, Jelani Nelson, and Krzysztof Onak. Sketching and streaming entropy via approximation theory. In *49th Annual IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 489–498, 2008.

[IPW11] Piotr Indyk, Eric Price, and David P. Woodruff. On the power of adaptivity in sparse recovery. In *IEEE 52nd Annual Symposium on Foundations of Computer Science (FOCS)*, pages 285–294, 2011.

[IW05] 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 (STOC)*, pages 202–208, 2005.

[JST11] Hossein Jowhari, Mert Sağlam, and Gábor Tardos. Tight bounds for Lp samplers, finding duplicates in streams, and related problems. In *Proceedings of the 30th ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems (PODS)*, pages 49–58, 2011.

[KMN11] Daniel Kane, Raghu Meka, and Jelani Nelson. Almost optimal explicit Johnson-Lindenstrauss families. In *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques*, pages 628–639. Springer, 2011.

[KSP03] Richard M. Karp, Scott Shenker, and Christos H. Papadimitriou. A simple algorithm for finding frequent elements in streams and bags. *ACM Trans. Database Syst.*, 28:51–55, 2003.

[MAE05] 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)*, pages 398–412, 2005.

[MG82] Jayadev Misra and David Gries. Finding repeated elements. *Sci. Comput. Program.*, 2(2):143–152, 1982.

[MM12] Gurmeet Singh Manku and Rajeev Motwani. Approximate frequency counts over data streams. *PVLDB*, 5(12):1699, 2012.

[Mut05] S. Muthukrishnan. Data streams: Algorithms and applications. *Foundations and Trends in Theoretical Computer Science*, 1(2), 2005.

[MW10] 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)*, pages 1143–1160, 2010.

A Pick-and-Drop counter example

We begin with a brief description of the Pick-and-Drop algorithm and refer the reader to [BO13] for the full details. Afterwards we will describe a stream where the algorithm fails, with probability at least $1 - n^{-1/8}$, to find a ℓ_2 heavy hitter and give some intuition about why the algorithm has this behavior. That is, the probability the algorithm succeeds is inverse polynomially small.

There is a parameter to the algorithm t and the algorithm begins by partitioning the stream into m/t consecutive intervals of t updates each. The value of t is chosen such that m/t is roughly the smallest frequency of a heavy hitter that we wish to find. Hence, the average number of heavy hitter updates per interval is $\Omega(1)$. In each interval, independently, a position $T \in \{1, 2, \ldots, t\}$ is chosen at random and the item in the Tth position within the interval is sampled. We also count how many times the sampled item it appears within $\{T, T+1, \ldots, t\}$. Next, the following "competition" is performed. We traverse the intervals sequentially from the first to the last and maintain a global sample and counter. Initially, the global sample is the sample from the first interval and the global counter is the counter from the first interval. For each future interval i, two options are possible: (1) the global sample is replaced with the sample from i and the global counter is replaced with the counter from interval i, or (2) the global sample remains unchanged and the global counter is increased by the number of times the global sample item appears in interval i. Also, the algorithm maintains X which is the current number of intervals for which the current global counter has not been replaced. If the maximum between X and the counter from the ith interval is greater than the global counter then (1) is executed, otherwise (2) is executed.

Consider the following counter example that shows Pick-and-Drop cannot find ℓ_2 heavy hitters in $O(1)$ words. $f \in \mathbb{R}^{2n}$ is a frequency vector where one coordinate H has frequency $\sqrt{n}$, n elements have frequency 1, and $\sqrt{n}$ elements have frequency $n^{1/4}$, call the latter "pseudo-heavy". The remaining coordinates of f are 0. Consider the stream that is split into t intervals $B_1, \ldots, B_t$ where $t = \sqrt{n}$ and each interval has size $\Theta(t)$. The items are distributed as follows.

- Each interval w where $w = qn^{1/4}$ for $q = 1, 2, \ldots, n^{1/4}$, is filled with $n^{1/4}$ pseudo-heavy elements each appearing $n^{1/4}$ times and appearing in no other interval.
- Each interval $w + h$, for $h = 1, 2, \ldots, n^{1/8}$ contains $n^{1/8}$ appearances of H and remaining items that appear only once in the stream.
- Each interval $w + h$, for $h = n^{1/8} + 1, \ldots, n^{1/4} - 1$ contains items that appear only once in the stream.

Obviously, a pseudo-heavy element will be picked in every "w interval". In order for it to be beaten by H, its count must be smaller than $n^{1/8}$ and H must be picked from one of the $n^{1/8}$ intervals immediately following. The intersection of these events happens with probability no more than $n^{-1/8}\left(n^{1/8} \cdot \frac{n^{1/8}}{n^{1/2}}\right) = n^{-3/8}$. As there are only $n^{1/4}$ "w intervals", the probability that the algorithm outputs H is smaller than $n^{-1/8}$.

Note that the algorithm cannot be fixed by choosing other values of t in the above example. If $t \gg \sqrt{n}$ then H might be sampled with higher probability but the pseudo-heavy elements will also have higher probabilities and the above argument can be repeated with a different distribution in the intervals. If $t \ll \sqrt{n}$ then the probability to sample H in any of the rounds becomes too small.

This counterexample is not contrived—it shows why the whole Pick-and-Drop sampling algorithm fails to shed any light on the ℓ_2 heavy hitters problem. Let us explain why the algorithm will not work in polylogarithmic space for $k = 2$. Consider the case when the global sample is $h \neq H$ and the global counter is f_h. In this case, the global sample can "kill" f_h appearances of H in the next f_h intervals, by the description of the algorithm. The probability to sample h is f_h/t, so the expected number of appearances of H that will be killed is upper bounded by $\sum_h f_h^3/t = F_3/t$. In the algorithm, we typically choose $t = \sqrt{F_1}$. Consider the case when $F_1 = \Theta(n)$, $F_2 = \Theta(n)$ but $F_2 = o(F_3)$. In this case the algorithm needs f_H to be at least $CF_3/\sqrt{F_2}$ for a constant C. This is impossible if $f_H^2 = \Theta(F_2)$. For $t = o(\sqrt{n})$ the probability that H is sampled becomes $o(1)$. For $t = \omega(\sqrt{n})$ we need a smaller decay for H to survive until the end in which case the above analysis can be repeated with the new decay factor for pseudo-heavy elements.