Deterministic Heavy Hitters with Sublinear Query Time

Yi Li
Nanyang Technological University, Singapore
yili@ntu.edu.sg

Vasileios Nakos[1]
Harvard University, USA
vasileiosnakos@g.harvard.edu

Abstract

We study the classic problem of finding ℓ_1 heavy hitters in the streaming model. In the general turnstile model, we give the first deterministic sublinear-time sketching algorithm which takes a linear sketch of length $\mathcal{O}(\epsilon^{-2}\log n \cdot \log^*(\epsilon^{-1}))$, which is only a factor of $\log^*(\epsilon^{-1})$ more than the best existing polynomial-time sketching algorithm (Nelson et al., RANDOM '12). Our approach is based on an iterative procedure, where most unrecovered heavy hitters are identified in each iteration. Although this technique has been extensively employed in the related problem of sparse recovery, this is the first time, to the best of our knowledge, that it has been used in the context of heavy hitters. Along the way we also obtain a sublinear time algorithm for the closely related problem of the ℓ_1/ℓ_1 compressed sensing, matching the space usage of previous (super-)linear time algorithms. In the strict turnstile model, we show that the runtime can be improved and the sketching matrix can be made strongly explicit with $O(\epsilon^{-2}\log^3 n/\log^3(1/\epsilon))$ rows.

2012 ACM Subject Classification Theory of computation → Streaming, sublinear and near linear time algorithms

Keywords and phrases heavy hitters, turnstile model, sketching algorithm, strongly explicit

Digital Object Identifier 10.4230/LIPIcs.APPROX-RANDOM.2018.18

Related Version Full version available at `https://arxiv.org/abs/1712.01971`.

1 Introduction

The problem of detecting *heavy hitters*, also frequently referred to as *elephants* or *hot items*, is one of the most well-studied problems in databases and data streams, from both theoretical and practical perspectives. In this problem, we are given a long data stream of elements coming from a large universe, and we are asked to report all the elements that appear at least a large number of times (called *heavy hitters*), using space that is much smaller than the size of the universe and the length of the stream.

Finding popular terms in search queries, identifying destination adresses of packets, detecting anomalies in network traffic streams such as denial-of-service (DoS) attacks, or performing traffic engineering, are only some of the important practical appearances of the heavy hitters problem. For example, the central task of managing large-scale networks lies in accurately measuring and monitoring network traffic [28, 26]. Interestingly, empirical studies [6, 21, 23, 27] indicate that flow-statistics in large networks follow an elephant/mice

[1] Supported in part by NSF grant IIS-1447471

Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques (APPROX/RANDOM 2018).
Editors: Eric Blais, Klaus Jansen, José D. P. Rolim, and David Steurer; Article No. 18; pp. 18:1–18:18

Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

phenomenon, i.e., the vast majority of the bytes are concentrated on only a small fraction of the flows.

On the theoretical side, heavy hitters appear very often, both in streaming algorithms and sparse recovery tasks. For the problems such as streaming entropy estimation [11], ℓ_p sampling [1, 20], finding duplicates [14], block heavy hitters [13], sparse recovery tasks [8, 9, 10], many algorithmic solutions use heavy hitters algorithms as subroutines.

Streaming Models. In this paper we consider the most general streaming model, called the *(general) turnstile model*, defined as follows. There is an underlying vector $x \in \mathbb{R}^n$, which is initialized to zero and is maintained throughout the input stream. Each element in the input stream describes an update $x_i \leftarrow x_i + \delta$ for some index i and increment δ, where δ can be either positive or negative.

We also consider a restricted version of the general turnstile model, called the *strict turnstile model*, under which it is guaranteed that $x_i \geq 0$ for all i throughout the input stream. This restricted model captures the practical scenario where item deletions are allowed but an item cannot be deleted more than it is inserted.

Sketching Algorithms. An important class of streaming algorithms is called *sketching algorithms*. A sketching algorithm maintains a short linear sketch $v = \Phi x$ (where $\Phi \in \mathbb{R}^{m \times n}$) throughout the input stream and then runs a recovery algorithm $\mathcal{D}$, which has access to only v and Φ, to output a desired $\hat{x}$. The space usage is proportional to m (the length of the sketch v) and to the memory needed to store Φ. Therefore we wish to minimize m and design a structured Φ such that storing Φ takes little space. Surprisingly, all existing streaming algorithms under the general turnstile model are sketching algorithms, and it has been shown [17] that all streaming algorithms under the general turnstile model can be converted to sketching algorithms with a mild increase in the space usage.

1.1 ℓ_∞/ℓ_1 Sparse Recovery

The heavy hitter problem has been studied under various streaming models and various recovery guarantees. Depending on the heaviness we are interested in, we distinguish between ℓ_1 and ℓ_2 heavy hitters. We are interested in finding, in the first case, the coordinates which are at least $\epsilon\|x\|_1$ in magnitude, and in the second case, the coordinates which are at least $\epsilon\|x\|_2$. Although finding ℓ_2 heavy hitters is strictly stronger than finding ℓ_1 heavy hitters, we consider only ℓ_1 heavy hitters in this paper, for it is impossible to find ℓ_2 heavy hitters using a deterministic space-saving sketching algorithm (see details below).

Specifically, we consider a classical recovery guarantee, called the ℓ_∞/ℓ_1 error guarantee in the literature, that is, the algorithm outputs an $\mathcal{O}(1/\epsilon)$-sparse vector $\hat{x}$ such that

$$\|\hat{x} - x\|_\infty \leq \epsilon\|x_{-r}\|_1, \tag{1}$$

for some parameters ϵ and r, where x_{-r} is the vector obtained by zeroing out the largest r coordinates of x in magnitude (absolute value). This type of guarantee requires not only finding the heavy hitters, but also giving 'good enough' estimates of them, where the estimates are measured with respect to $\|x_{-r}\|_1$ instead of the larger $\|x\|_1$; this type of guarantee is called the *tail guarantee*. It should be noted that the ℓ_∞/ℓ_1 guarantee has been extensively studied and is provided by several classical algorithms, e.g. Count-Min [5], LossyCounting [18], SpaceSaving [19], although not all of them work under the general turnstile model.

In this paper we focus on deterministic sketching algorithms, which means that both the matrix Φ and the recovery algorithm $\mathcal{D}$ allow uniform reconstruction of every $x \in \mathbb{R}^n$ up to $\epsilon\|x_{-r}\|_1$ error, providing the best applicability. This is also referred to as "for-all" guarantee in the sparse recovery literature, in contrast to the "for-each" guarantee, which allows reconstruction of a fixed vector with some target success probability. Most previous sketching algorithms for the heavy hitter problems concern the "for-each" model and resort to randomization (e.g. [5, 3]), by drawing a random Φ from some distribution and guaranteeing $\mathcal{D}$ to output an acceptable $\hat{x}$ with a good probability. Other sketching algorithms are deterministic, however, they run in time at least linear in the universe size n. The goal of fast query time, say, logarithmic in n, is crucial to streaming applications. For instance, in traffic monitoring n equals the number of all possible packets, namely 2^{32}; a linear runtime would be prohibitive in any reasonable real-world scenario. A natural goal is to design a sketching algorithm with sublinear query time, preferably $\mathcal{O}(\text{poly}(1/\epsilon, r, \log n))$, with as little space usage as possible.

Apart from practical importance, deterministic algorithms for heavy hitters is an interesting theoretical subfield of streaming algorithms, connected to dimensionality reduction and incoherent matrices [22]. Moreover, gaining insight into such questions may give insight for many other data stream problems where heavy hitter algorithms are used as subroutines. We note that any deterministic sketching algorithm that finds ℓ_2 heavy hitters requires $\Omega(n)$ space [4] (which implies that the trivial algorithm storing the entire input vector x is asymptotically optimal), while the best lower bound for ℓ_1 heavy hitters is $\Omega(r\log(n/r)/\log r + \epsilon^{-2} + \epsilon^{-1}\log n)$ [22, 7].

The state-of-the-art deterministic sketching algorithms for ℓ_1 heavy hitters are found in [22], where two algorithms are given. The first algorithm uses $m = \mathcal{O}(\epsilon^{-2}\log n \cdot \min\{1, \log n/(\log\log n + \log(1/\epsilon))^2\})$ rows and achieves $\epsilon\|x_{-\lceil 1/\epsilon\rceil}\|_1$ tail guarantee (setting $r = \lceil 1/\epsilon\rceil$ in (1)). The second algorithm uses $m = \mathcal{O}(\epsilon^{-2}\log n)$ rows and achieves a stronger $\epsilon\|x_{-\lceil 1/\epsilon^2\rceil}\|_1$ tail guarantee. We can see that when $\epsilon < 2^{-\Omega(\sqrt{\log n})}$, the first algorithm uses less space, whereas when $\epsilon \geq 2^{-\Omega(\sqrt{\log n})}$ the second algorithm is better. The number of rows used by both algorithms is at most suboptimal by a $\log n$ factor while the runtimes are both superlinear $\Omega(\epsilon^{-1}n\log n)$. In this paper our goal is to obtain an algorithm which runs in sublinear time in n while attaining the stronger $\epsilon\|x_{-\lceil 1/\epsilon^2\rceil}\|_1$ tail guarantee with near-optimal number of rows. Our main theorem is formally stated below.

▶ **Theorem 1** (ℓ_∞/ℓ_1). *There exists a linear sketch $\Phi \in \mathbb{R}^{m\times n}$ such that for every $x \in \mathbb{R}^n$, we can, given Φx, find an $\mathcal{O}(1/\epsilon)$-sparse vector $\hat{x}$ such that*

$$\|x - \hat{x}\|_\infty \leq \epsilon\|x_{-\lceil 1/\epsilon^2\rceil}\|_1,$$

in $\mathcal{O}((1/\epsilon)^6\,\text{poly}(\log n))$ time. The number of rows of Φ equals $m = \mathcal{O}(\epsilon^{-2}\log n\log^*(\epsilon^{-1}))$.

The number of rows in Φ is more than that in [22] by merely a factor of $\mathcal{O}(\log^*(1/\epsilon))$, while the query time is sublinear for all small $\epsilon \geq n^{-1/7}$, a significant improvement upon the previous $\mathcal{O}(\epsilon^{-1}n\log n)$ runtime in [22].

Difference Between Deterministic and Explicit Schemes. To avoid confusion, we note the difference between 'deterministic' and 'explicit'. In the compressed sensing/sparse recovery literature 'deterministic' is also called 'for-all' or 'uniform', which means that a single matrix Φ suffices for the reconstruction of all vectors. 'Explicit' means that the matrix Φ can be constructed in time $\text{poly}(1/\epsilon, r, n)$. 'Strongly explicit' means that any entry of the matrix can be computed in time $\text{poly}(1/\epsilon, r, \log n)$. Hence, in this paper we show the existence of a

single matrix that allows reconstrction of all vectors, which we argue via the probabilistic method. The same holds for some of the schemes in [22].

In the strict turnstile model, we show that for the tail guarantee of $r = \lceil 1/\epsilon \rceil$, the matrix Φ can be made strongly explicit, with a mild increase in the number of rows, and the runtime can be improved polynomially. Note, however, the tail guarantee is with respect to $r = \lceil 1/\epsilon \rceil$ instead of $r = \lceil 1/\epsilon^2 \rceil$. We state our theorem below and omit the proof owing to space limitations.

▶ **Theorem 2** (ℓ_∞/ℓ_1, strict turnstile)**.** *There exists a strongly explicit matrix M of $O((1/\epsilon)^2 \log^3 n/\log^3(1/\epsilon))$ rows, which, given Mx in the strict turnstile model, allows us to find an $O(1/\epsilon)$-sparse vector $\hat{x}$ such that*

$$\|x - \hat{x}\|_\infty \leq \epsilon \|x_{-\lceil 1/\epsilon \rceil}\|_1,$$

in time $O((1/\epsilon)^3 \log^3 n/\log^3(1/\epsilon))$.

1.2 ℓ_1/ℓ_1 Sparse Recovery

In the ℓ_1/ℓ_1 sparse recovery problem, instead of the guarantee (1), the algorithm should output $\hat{x}$ such that

$$\|\hat{x} - x\|_1 \leq (1+\epsilon)\|x_{-k}\|_1. \tag{2}$$

It is known that any deterministic sketching algorithm requires $m = \Omega(\epsilon^{-2} + \epsilon^{-1} k \log(\epsilon n/k))$ rows of Φ [22], and the best known upper bound is $m = O(\epsilon^{-2} k \log(n/k))$ rows [12, 2, 10], suboptimal from the lower bound by only a logarithmic factor. However all these algorithms suffer from various defects: the algorithms in [12, 2] run in polynomial time in n, and that in [10] imposes a constraint on ϵ that precludes it from being a small constant when k is small. In this paper, we show that one can achieve sublinear runtime with the same number of rows for small k.

▶ **Theorem 3** (ℓ_1/ℓ_1)**.** *There exists a matrix $\Phi \in \mathbb{R}^{m \times n}$ such that, given Φx with $x \in \mathbb{R}^n$, we can find an $\mathcal{O}(k)$-sparse vector $\hat{x}$ satisfying (2) in $\mathcal{O}(k^3 \operatorname{poly}(1/\epsilon, \log n))$ time. The number of rows of Φ is $m = \mathcal{O}(\epsilon^{-2} k \log n)$.*

This result is the first sublinear time algorithm for all small $k \leq n^{0.3}$ with constant ϵ, while the algorithm in [10], using the same number of measurements, works for constant ϵ only when $n^{\delta'} \leq k \leq n^{1-\delta}$ (where $\delta, \delta' > 0$ are arbitrarily small constants), owing to its use of the list-decodable code. Combining the two results, we have solved the ℓ_1/ℓ_1 problem in sublinear time for all $k \leq n^{1-\delta}$ and constant ϵ.

▶ Remark. Our results heavily involve random hash functions, for which $\mathcal{O}(1/\epsilon)$- or $\mathcal{O}(k)$-wise independence would be sufficient. The space complexity of our algorithms is the same as the number of rows, unless stated otherwise.

2 Overview of Techniques

The main result on ℓ_∞/ℓ_1 combines different ideas from sparse recovery and heavy hitters literature. We first prove a result with the $\epsilon\|x_{-1/\epsilon}\|_1$ tail guarantee. We need a different, more careful construction of the weak system, akin to that in [10], which does not detect only a constant fraction of the heavy hitters, but a much larger fraction, as much as $(1 - \epsilon \log\log(1/\epsilon))$. One of our technical contributions and tools is the design of a more

general form of the weak system, which we then apply iteratively with carefully chosen parameters to recover all heavy hitters. We then iterate by subtracting the found heavy hitters, and try to find the remaining ones using a new matrix of the same number of resources (rows). Similar iteration techniques have been adopted in most combinatorial sparse recovery tasks, where the algorithms are allowed to miss even all heavy hitters if they are not large enough! Our ℓ_∞/ℓ_1 error guarantee, however, makes it prohibitive to miss small heavy hitters in later iterations. To that end, we heavily exploit the more abundant resource of $1/\epsilon^2$ rows in each iteration throughout, for which we pay a mild extra factor in the total number of rows. While in previous sparse recovery tasks the number of rows decreases across different iterations, this does not happen in our case. As aforementioned, we pay an additional $\log^*(1/\epsilon)$ factor as we shall have $\mathcal{O}(\log^*(1/\epsilon))$ iterations until all heavy hitters are recovered.

To obtain the stronger tail guarantee of $\epsilon\|x_{-1/\epsilon^2}\|_1$, we invoke additionally our ℓ_1/ℓ_1 algorithm and the point-query algorithm of [22]. We note that any sub-optimality in the number of rows of the ℓ_1/ℓ_1 linear sketch would yield a worse result for our main scheme, which forces us to obtain also an improved result for the ℓ_1/ℓ_1 problem. Our new weak system and the novel idea of using the iterative loop to satisfy the ℓ_∞/ℓ_1 guarantee may indicate new approaches to tackle heavy hitters tasks, and might be of interest beyond the scope of this paper. Our side-result on ℓ_1/ℓ_1 sparse recovery, is a combination of [10] and [16]. Specifically, one can avoid the Parvaresh-Vardy list-recoverable code that [10] employed, and use instead the clustering technique in [16], upon the two-layer hashing schemes and linking technique in [10]. This makes possible an improved result for ℓ_1/ℓ_1 that removes the restriction on ϵ the previous work of [10] was suffering from.

We remark that any improvement in the running time of the clustering algorithm of [16] immediately translates to improvement to ℓ_∞/ℓ_1 and ℓ_1/ℓ_1 schemes. More specifically, a near-quadratic or near-linear algorithm for that clustering would imply a near-quadratic or near-linear (in the number of rows of the sketching matrix) time algorithm for all three of our tasks. The current state of the art for that algorithm is $\tilde{\mathcal{O}}(N^3)$ runtime on a graph of N vertices, since the algorithm performs N calls to a routine that finds a Cheeger cut. We also remark that we could obtain our ℓ_∞/ℓ_1 result using explicit list-recoverable codes, such as Parvaresh-Vardy code, but this would lead to a slightly worse result than what we currently have.

In the strict turnstile model, we obtain a family of fully explicit matrices that solves the point query problem in sublinear time using the family of matrices from [22]. First, we show how to stengthen the guarantee obtained by the matrix in [22], achieving a stronger tail guarantee, and then we show how to recursively combine those explicit matrices to obtain a sketching algorithm that gives the ℓ_∞/ℓ_1 guarantee in sublinear time. Our result in the strict turnstile model, not only is fully explicit, but also has a better running time than the general turnstile model and gets also improved space for some regime of ϵ, namely $\epsilon \leq 2^{-\sqrt{\log n}}$. We note though that we obtain a weaker tail guarantee than in the general case, namely $r = 1/\epsilon$, instead of $1/\epsilon^2$. This weaker guarantee stems from the lack of fully explicit ℓ_1/ℓ_1 schemes with nearly optimal measurements, that can also answer queries in sublinear time.

3 Preliminaries

For a vector $x \in \mathbb{R}^n$, we define x_{-k} to be the vector obtained by zeroing out the largest k coordinates in magnitude, and $\text{supp}(x)$ to be the set of the non-zero coordinates of x. We also define $H(x, k)$ to be the index set of the largest k coordinates of x in magnitude. Thus,

$\mathrm{supp}(x_{-k}) \cap H(x,k) = \emptyset$. We also define $H(x,k,\epsilon) = \{i \in [n] : |x_i| \geq \epsilon/k \|x_{-k}\|_1\}$. We assume that the word size is $w = \Theta(\log n)$.

An error correcting code is a subset $\mathcal{C} \subseteq \Sigma^n$, where Σ is a finite set called alphabet and $|\mathcal{C}| = |\Sigma|^k$ for some $k \geq n$, together with an injective encoding map $\mathrm{enc} : \Sigma^k \to \mathcal{C}$ and a decoding map $\mathrm{dec} : \mathcal{C} \to \Sigma^k$. The parameter k is called the message length and n is called codeword length or block length. We say that an error correcting code can correct up to θ-fraction of errors if for any message $m \in \Sigma^k$ and any $x \in \Sigma^n$ such that the Hamming distance $d(\mathrm{enc}(m), x) \leq \theta n$, it holds that $\mathrm{dec}(x) = m$.

3.1 Two-layer Hashing Schemes

In this subsection we review the two-layer hashing scheme and the linking techniques used in [10], which will be the skeleton of construction for all our sparse recovery results.

First we recall some definitions, taken from [10], regarding bipartite expander, two-layer hashing and isolation of heavy hitters.

▶ **Definition 4** (bipartite expander)**.** An $(n, m, d, \ell, \epsilon)$-bipartite expander is a d-left-regular bipartite graph $G(L \cup R, E)$ where $|L| = n$ and $|R| = m$ such that for any $S \subseteq L$ with $|S| \leq \ell$ it holds that $|\Gamma(S)| \geq (1-\epsilon)d|S|$, where $\Gamma(S)$ is the neighbour of S (in R). When n and m are clear from the context, we abbreviate the expander as (ℓ, d, ϵ)-bipartite expander.

▶ **Definition 5** (one-layer hashing scheme)**.** The (N, B, d) (one layer) hashing scheme is the uniform distribution on the set of all functions $f : [N] \to [B]^d$. We write $f(x) = (f_1(x), \ldots, f_d(x))$, where f_i's are independent (N, B) hashing schemes.

Each instance of such a hashing scheme induces a d-left-regular bipartite graph with Bd right nodes. When N is clear from the context, we simply write (B, d) hashing scheme.

▶ **Definition 6** (two-layer hashing scheme)**.** An (N, B_1, d_1, B_2, d_2) (two-layer) hashing scheme is a distribution μ on the set of all functions $f : [N] \to [B_2]^{d_1 d_2}$ defined as follows. Let g be a random function subject to the (N, B_1, d_1) hashing scheme and $\{h_{i,j}\}_{i \in [d_1], j \in [d_2]}$ be a family of independent functions subject to the (B_1, B_2, d_2) hashing scheme which are also independent of g. Then μ is defined to be the distribution induced by the mapping

$$x \mapsto (h_{1,1}(g_1(x)), \ldots, h_{1,d_2}(g_1(x)), h_{2,1}(g_2(x)), \ldots, h_{2,d_2}(g_2(x)), \ldots, h_{d_1,1}(g_{d_1}(x)), \ldots, h_{d_1,d_2}(g_{d_1}(x))) .$$

Each instance of such a hashing scheme gives a $d_1 d_2$-left-regular bipartite graph of $B_2 d_1 d_2$ right nodes. When N is clear from the context, we simply write (B_1, d_1, B_2, d_2) hashing scheme. Conceptually we hash N elements into B_1 buckets and repeat d_1 times; these buckets will be referred to as first-layer buckets. In each of the d_1 repetitions, we hash B_1 elements into B_2 buckets and repeat d_2 times, those buckets will be referred to as second-layer buckets.

Bipartite expander graphs can be used as hashing schemes because of their isolation property.

▶ **Definition 7** (isolation property)**.** An $(n, m, d, \ell, \epsilon)$-bipartite expander G is said to satisfy the (ℓ, η, ζ)-isolation property if for any set $S \subset L(G)$ with $|S| \leq \ell$, there exists $S' \subset S$ with $|S'| \geq (1-\eta)|S|$ such that for all $x \in S'$ it holds that $|\Gamma(\{x\}) \setminus \Gamma(S \setminus \{x\})| \geq (1-\zeta)d$.

The following lemma shows that a random two-layer hashing satisfies a good isolation property with high probability. Previous works [25, 10] build sparse recovery systems upon this lemma.

▶ **Lemma 8** ([10]). *Let $\epsilon > 0$, $\alpha > 1$ and (N, B_1, d_1, B_2, d_2) be a two-layer hashing scheme with $B_1 = \Omega(\frac{k}{\zeta^\alpha \epsilon^{2\alpha}})$, $d_1 = \Omega(\frac{\alpha}{\alpha-1} \cdot \frac{1}{\zeta\epsilon} \frac{\log N}{\log(B_1/k)})$, $B_2 = \Omega(\frac{k}{\zeta\epsilon})$ and $d_2 = \Omega(\frac{1}{\zeta} \log \frac{B_1}{k})$. Then with probability $\geq 1 - 1/N^c$, the two-layer hashing scheme with parameters prescribed above gives an $(N, B_2 d_1 d_2, d_1 d_2, 4k, \epsilon)$ bipartite graph with the (L, ϵ, ζ)-isolation property, where $L = O(k/\epsilon)$.*

4 A Sublinear Time ℓ_1/ℓ_1 Algorithm

The result is almost immediate by replacing the list-recoverable code in [10] with the clustering algorithm in [16], which we now give a brief review. The overall algorithm is an iterative algorithm of $\Theta(\log k)$ iterations. Each iteration is called a *weak system*, which (i) recovers at least a constant fraction of the remaining heavy hitters (and hence all heavy hitters can be recovered in $\Theta(\log k)$ iterations) and (ii) introduces only a small amount of error (see, e.g. [25, 10]). The introduced error comes from two sources: the estimation error of those recovered heavy hitters and some small coordinates that can be safely ignored.

The following lemma is crucial in bounding the number of missed heavy hitters in iteration, modified from [10]. For completeness we include the proof in Appendix A.

▶ **Lemma 9.** *Let $\theta, \epsilon \in (0,1)$, $\delta \in (0, \frac{1}{2}]$ and $\beta, \zeta > 0$ such that $0 < \zeta < \delta - \frac{64\beta}{\theta}$. Suppose that G is a $(4s, d, \beta\epsilon)$-bipartite expander which satisfies the $(\frac{6}{\gamma\epsilon}, \frac{\epsilon\theta}{12}, \zeta)$-isolation property, where $\gamma \in [\frac{\theta}{s}, 1]$. Let $x \in \mathbb{R}^n$ be a vector which can be written as $x = y + z$, where y and z have disjoint supports, $|\operatorname{supp}(y)| \leq s$ and $\|z\|_1 \leq 3/2$. For each $i \in [n]$ define the multiset E_i as*

$$E_i = \left\{ \sum_{(u,v)\in E} x_u \right\}_{v \in \Gamma(\{i\})}.$$

Note that $|E_i| = d$ since it is a multiset. Then, for every $D \subset [n], |D| \leq 2s$, we have that

$$\left| \left\{ i \in D : |x_i - w| \geq \frac{\epsilon\gamma}{4} \text{ for at least } (1-\delta)d \text{ values } w \text{ in } E_i \right\} \right| \leq \frac{\theta}{\gamma}.$$

We remark that the construction of the weak system is similar to the partition setup in [16], where the linking information and the message block are 'absorbed' into the fashion coordinates are split into buckets so that recovering a heavy hitter in a bucket will automatically recover that information correctly instead of recovering the information from second-layer buckets with an error correcting code. In this paper, however, we opt for the two-layer construction for the 'for-all' guarantee, for the presentation would be simpler for our sparse recovery results as some auxiliary lemmata are already proved in [10].

We now present the precise statement of our weak system , which is central to our ℓ_1/ℓ_1 result. The proof is almost identical to that in [10] and is postponed to Appendix B.

▶ **Lemma 10** (Weak system). *Suppose that $s \leq \sqrt{n}$ and $\epsilon \in (0,1)$. There exist a linear sketch $\Phi \in \mathbb{R}^{m \times n}$ and an algorithm $\textsc{WeakSystem}(x, s, \epsilon)$ satisfying the following:*

- *For any vector $x \in \mathbb{R}^n$ that can be written as $x = y + z$, where y and z have disjoint supports, $|\operatorname{supp}(y)| \leq s$, $\|y\|_\infty \geq \epsilon/(2s)$ and $\|z\|_1 \leq 3/2$, given the measurements Φx, the decoding algorithm $\mathcal{D}$ returns $\hat{x}$ such that x admits the decomposition of*

  $$x = \hat{x} + \hat{y} + \hat{z},$$

  where $|\operatorname{supp}(\hat{x})| = s$, $|\operatorname{supp}(\hat{y})| \leq s/8$ and $\|\hat{z}\|_1 \leq \|z\|_1 + \epsilon/4$. Intuitively, $\hat{y}$ and $\hat{z}$ will be the head and the tail of the residual $x - \hat{x}$, respectively;
- *$m = \mathcal{O}(\epsilon^{-2} s \log n)$;*
- *$\mathcal{D}$ runs in $\mathcal{O}(s^3 \operatorname{poly}(1/\epsilon, \log n))$ time.*

5 A Sublinear Time ℓ_∞/ℓ_1 Algorithm

For ease of exposition and connection with the previous algorithms, we set $k = \lceil 1/\epsilon \rceil$ in this section.

First, we prove the following weaker theorem, and shall show how to bootstrap this theorem in order to obtain Theorem 1 in Section 5.1.

▶ **Theorem 11.** *There exists a linear sketch $\Phi \in \mathbb{R}^{m\times n}$ such that for every $x \in \mathbb{R}^n$, we can, given Φx, find an $\mathcal{O}(k)$-sparse vector $\hat{x}$ such that $\|x - \hat{x}\|_\infty \le \frac{1}{k}\|x_{-k}\|_1$ in $\mathcal{O}(k^6 \operatorname{poly}(\log n))$ time. The number of rows of Φ equals $m = \mathcal{O}(k^2 \log n \log^* k)$. The space needed to store (y, Φ) is $\mathcal{O}(k^2 \log n \cdot \log^* k \cdot \log\log k)$ words.*

We remark that this theorem is slightly weaker than our main result, namely Theorem 1, because the error is measured with respect to $\|x_{-k}\|_1$, and not with respect to $\|x_{-k^2}\|_1$. We are going to bootstrap Theorem 11 later.

Weak-Level System. Lemma 9 is a central argument for deterministic sparse recovery tasks. Previous works [25, 10] used the lemma with $\gamma = 1/s$ and constant $\theta \in (0, 1)$ to show that if we estimate every coordinate x_i to be the median of E_i and take the biggest $\Theta(s)$ estimates in magnitude, we shall miss at most $2s$ heavy hitters, upon which weak systems that miss a θ-fraction of heavy hitters were constructed. The overall algorithm makes sequential calls to weak systems with geometrically decreasing number of remaining heavy hitters. In our case, since we want the stronger ℓ_∞/ℓ_1 guarantee, we are not allowed to decrease geometrically the number of rows for the weak systems. But, with more allotted number of rows, we can recover much more than a constant fraction of heavy hitters by exploiting the power of θ. The full proof is postponed to Appendix C.

▶ **Lemma 12** (Weak system)**.** *Suppose that $w \le s \le k \le \sqrt{n}$ and $\eta \in (0, 1)$ is an arbitrarily small constant. There exist a linear sketch $\Phi \in \mathbb{R}^{m\times n}$ and an algorithm* WeakSystem *(x, k, s, w) satisfying the following:*

- *For any vector $x \in \mathbb{R}^n$ that can be written as $x = y + z$, where y and z have disjoint supports, $|\operatorname{supp}(y)| \le s$, $\|y\|_\infty \ge 1/(2k)$ and $\|z\|_1 \le 2 - s/k$, given the sketch Φx, the decoding algorithm $\mathcal{D}$ returns $\hat{x}$ such that x admits the decomposition of*

  $$x = \hat{x} + \hat{y} + \hat{z},$$

  where $|\operatorname{supp}(\hat{x})| = s$, $\|(x - \hat{x})_{\operatorname{supp}(\hat{x})}\|_\infty \le \frac{1}{2k}$, $|\operatorname{supp}(\hat{y})| \le \sqrt{sw}$ and $\|\hat{z}\|_1 \le 2 - \frac{s}{2k}$. Intuitively, $\hat{y}$ and $\hat{z}$ will be the head and the tail of the residual $x - \hat{x}$, respectively.
- *$m = \mathcal{O}(\frac{k^2}{w} \log n)$,*
- *$\mathcal{D}$ runs in $\mathcal{O}(k^6 \operatorname{poly}(\log n))$ time.*

Proof Sketch. We only specify the parameters of the two-layer hashing and the application of Lemma 9 below. We instantiate the two-layer hashing and the encoding scheme as in Section 3.1, where $\alpha \in (1, 2)$, $B_1 = \Theta(k^{2\alpha})$, $d_1 = \Theta(\frac{k}{\sqrt{sw}} \frac{\log n}{\log(B_1/s)})$, $B_2 = \Theta(k\sqrt{s/w})$ and $d_2 = \Theta(\log(B_1/s))$. By Lemma 8 (to see the conditions hold, replace k with s and ϵ with $k/\sqrt{sw}$), we can find a two-layer hashing with these prescribed parameters which satisfies $(\Theta(k), d_1 d_2, \frac{\sqrt{sw}}{k})$-expansion property and $(\Theta(k), \frac{\sqrt{sw}}{k}, \Theta(1))$-isolation property. Invoking Lemma 9 with $\delta = \Theta(\sqrt{sw}/k)$, $\theta = \Theta(\sqrt{sw}/k)$, $\epsilon = 1$ and $\gamma = 1/k$, and following the argument in [10, Section 4.1], we have good estimates for all but at most $\Theta(\theta/\gamma) = \sqrt{sw}$ heavy hitters (elements in $\operatorname{supp}(y)$). We are able to recover those heavy hitters as argued in [10] with the clustering algorithm from [16]. ◀

Algorithm 1 Overall algorithm for ℓ_∞/ℓ_1 sparse recovery. In the pseudocode below, $v^{(i,r)}$ as an argument of WeakSystem is understood to be restricted to the corresponding coordinates.

Input: sketching matrix Φ, sketch $v = \Phi x$, sparsity parameter k
Output: $\hat{x}$ that approximates x with ℓ_∞/ℓ_1 guarantee

$v^{(0,1)} \leftarrow v$
$k_1 \leftarrow k$
while $k_r > 4$ **do** ▷ $\mathcal{O}(\log^* k)$ rounds
  $i \leftarrow 0$
  while $k_r^{2^{-i}} \geq \max\{(i+1)^2, 4(1+\frac{1}{i+1})^4\}$ **do** ▷ $\mathcal{O}(\log\log k_r)$ steps
    $s_{i,r} \leftarrow (i+1)^2 k_r^{2^{-i}}$
    $w_{i,r} \leftarrow (i+1)^2$
    $\hat{x}^{(i,r)} \leftarrow \text{WeakSystem}(\Phi^{(i,r)}, v^{(i,r)}, k, s_{i,r}, w_{i,r})$
    $v^{(i+1,r)} \leftarrow v^{(i,r)} - \Phi\hat{x}^{(i,r)}$
    $i \leftarrow i+1$
  end while
  $i_r^* \leftarrow i-1$
  $s_{i,r} \leftarrow (s_{i_r^*,r})^{2^{-(i-i_r^*-1)}}$
  while $s_{i,r} \geq \max\{4, \log\log k_r\}$ **do** ▷ $\mathcal{O}(1)$ steps
    $w_{i,r} \leftarrow 1$
    $\hat{x}^{(i,r)} \leftarrow \text{WeakSystem}(\Phi^{(i,r)}, v^{(i,r)}, k, s_{i,r}, w_{i,r})$
    $v^{(i+1,r)} \leftarrow v^{(i,r)} - \Phi\hat{x}^{(i,r)}$
    $i \leftarrow i+1$
    $s_{i,r} \leftarrow (s_{i_r^*,r})^{2^{-(i-i_r^*-1)}}$
  end while
  $i_r^+ \leftarrow i - i_r^* - 1$
  $v^{(0,r+1)} \leftarrow v^{(i,r)}$
  $k_{r+1} \leftarrow s_{i,r}$
  $r \leftarrow r+1$
end while
$\hat{x}_{\text{final}} \leftarrow \text{WeakSystem}(\Phi^{(0,r)}, v^{(0,r)}, k, 4, 1/5)$ ▷ last round
return $\hat{x} \leftarrow \hat{x}_{\text{final}} + \sum_{r,i} \hat{x}^{(i,r)}$

Construction of Measurement Matrix. Now we construct the sketch for Theorem 11. The main idea is to apply the weak system (Lemma 12) repeatedly. We form our linear sketch Φ as illustrated below and present our recovery algorithm in Algorithm 1.

$$\Phi = \begin{bmatrix} \Phi_1 \\ \Phi_2 \\ \vdots \\ \Phi_R \\ \Phi_{\text{final}} \end{bmatrix}, \qquad \text{where } \Phi_r = \begin{bmatrix} \Phi^{(1,r)} \\ \vdots \\ \Phi^{(i_r^*,r)} \\ \Phi^{(i_r^*+1,r)} \\ \vdots \\ \Phi^{(i_r^*+i_r^+,r)} \end{bmatrix}, r = 1, \ldots, R.$$

Here

- the overall Φ is the vertical concatenation of $R+1$ matrices and every layer, except the last one, is further a concatenation of $i_r^* + i_r^+$ matrices, where $R = \Theta(\log^* k)$ and i_r^*, i_r^+

are computed as in Algorithm 1;
- the i-th layer in Φ_r, namely $\Phi_{i,r}$, is the sketching matrix for the weak system (Lemma 12) with parameters $s = s_{i,r}$ and $w = w_{i,r}$, the values of which are as assigned in Algorithm 1;
- the last layer of Φ, namely Φ_{final}, is the sketching matrix for the weak system with parameters $s = 4$ and $w = 1/5$.

Overall there are $i_r^* + i_r^+ + 1$ iterations in the algorithm and each iteration corresponds to one block of Φ. There are k heavy hitters at the beginning. Each iteration reduces the number of remaining heavy hitters to almost its square root and hence in $O(\log\log k)$ iterations the number of remaining heavy hitters will be reduced to a constant, and those heavy hitters will be all recovered in the last iteration. In order to minimize the number of measurements, in each of the first i_r^* iterations, the number of remaining heavy hitters is reduced to slightly bigger than its square root, and in each of the next i_r^+ iterations, to exactly the square root.

The parameters $s_{i,r}, w_{i,r}, i_r^*, i_r^+$ may seem adaptive at the first glance, but they in fact do not depend on the input x and depend only on the sparsity parameter k and can thus be pre-computed. The whole algorithm is non-adaptive.

Proof of Theorem 11. We only provide a sketch of the proof below and leave the full proof to Appendix D.

Proof Sketch. Without loss of generality, assume that $\|x_{-k}\|_1 = 1$. We shall apply Lemma 12 repeatedly to obtain a sequence of vectors $\hat{x}^{(i,r)}$, which admit decompositions $x^{(i,r)} = x - \hat{y}^{(i,r)} - \hat{z}^{(i,r)}$. We can show inductively that the loops invariants (I) below, parametrized by (i, r), are satisfied at the beginning on each while loop from Line 5 to 11 in Algorithm 1 and the loop invariants (II) below are satisfied at the beginning on each while loop from Line 14 to 20.

$$\text{(I)} \begin{cases} |\operatorname{supp}(\hat{y}^{(i,r)})| \le s_{i,r} := (i+1)^2 k_r^{2^{-i}}, \\ \|\hat{z}^{(i,r)}\|_1 \le 2 - s_{i,r}/k; \end{cases} \qquad \text{(II)} \begin{cases} |\operatorname{supp}(\hat{y}^{(i,r)})| \le s_{i,r} := (s_{i_r^*,r})^{2^{-(i-i_r^*-1)}}, \\ \|\hat{z}^{(i,r)}\|_1 \le 2 - s_{i,r}/k. \end{cases}$$

When the algorithm runs into Line 25, that is, when there are at most 4 heavy hitter left, we shall recover all of them in one call to the weak system.

The total number of rows and runtime, etc., follow from direct calculations. ◀

5.1 Getting the Final Result

We now show how to combine the ℓ_1/ℓ_1 scheme with the ℓ_∞/ℓ_1 scheme to obtain the main result of the paper. For completeness, we restate the main theorem with the substitution of $k = \lceil 1/\epsilon \rceil$.

▶ **Theorem 1** (rephrased). *There exists a linear sketch $\Phi \in \mathbb{R}^{m\times n}$ such that for every $x \in \mathbb{R}^n$, we can, given Φx, find an $\mathcal{O}(k)$-sparse vector $\hat{x}$ such that $\|x - \hat{x}\|_\infty \le (1/k)\|x_{-k^2}\|_1$ in $\mathcal{O}(k^6 \operatorname{poly}(\log n))$ time. The sketch length is $m = \mathcal{O}(k^2 \log n \log^* k)$ and the space needed to store (y, Φ) is $\mathcal{O}(k^2 \log n \cdot \log^* k \cdot \log\log k)$ words.*

We shall need the following lemma from [22].

▶ **Lemma 13** (Point Query [22]). *There exists a matrix $C \in \mathbb{R}^{m\times n}$ with $m = \mathcal{O}(k^2 \log n)$ rows, such that given $y = Cx$ and $i \in [n]$, it is possible to find in $\mathcal{O}(k \log n)$ time a value $\hat{x}_i$ such that $|x_i - \hat{x}_i| \le (1/k)\|x_{[n]\setminus\{i\}}\|_1$.*

The construction of C given in [22] includes taking C to be a Johnson-Lindenstrauss Transform matrix for the set of points $\{0, e_1, \ldots, e_n\}$, where $e_1, \ldots, e_n$ is the canonical basis of $\mathbb{R}^n$.

We now give a sketch of the proof of Theorem 1 and leave the full proof to Appendix E.

Proof of Theorem 1 (Sketch). We first pick a matrix A using Lemma 3, setting the sparsity parameter to k^2 and $\epsilon = 1$. We also pick a matrix B satisfying the guarantees of Theorem 11, with sparsity $6k$, and a matrix C using Lemma 13 with sparsity parameter $6k$. Our sketching matrix Φ is the vertical concatenation of A, B and C.

We first run the algorithm on Ax to obtain an $\mathcal{O}(k^2)$-sparse vector z such that $\|x - z\|_1 \leq 2\|x_{-k^2}\|_1$. Then we form $B(x - z)$ and using the query algorithm for B, we find an $\mathcal{O}(k)$-sparse vector w such that $\|(x - z) - w\|_\infty \leq \frac{1}{2k}\|x - z\|_1 \leq \frac{1}{k}\|x_{-k^2}\|_1$. It then follows that $\left\{i \in [n] : |x_i| > \frac{1}{k}\|x_{-k^2}\|_1\right\} \subseteq \operatorname{supp}(z) \cup \operatorname{supp}(w)$ and so it suffices to estimate x_i, for every $i \in \operatorname{supp}(z) \cup \operatorname{supp}(w)$, up to $(1/k)\|x_{-k^2}\|_1$ error, which we can do exactly as in [22].

The total number of rows and runtime, etc., follow from direct calculations. ◀

6 Conclusion and Open Problems

In this work, we present the first algorithm for finding ℓ_1 heavy hitters (ℓ_∞/ℓ_1 guarantee) deterministically in sublinear time, up to an $\mathcal{O}(\log^*(1/\epsilon))$ factor in the number of measurement from the best superlinear-time algorithm. It still remains to improve the dependence on ϵ in the running time, ideally to $\mathcal{O}(\epsilon^{-2}\operatorname{poly}(\log n))$. The problem could first be approached in the strict turnstile model, where it is possible to avoid the heavy machinery of list-recoverable codes or the clustering algorithm of [16]. Another open problem is to find (fully) explicit constructions that allows a sublinear-time decoding with the number of rows near $\mathcal{O}(\epsilon^{-2}\log n)$ in the strict turnstile model. In the general turnstile model, our current understanding and techniques suggest that an explicit scheme would require an explicit construction of expanders or lossless condensers, together with list-recoverable codes with nearly optimal encoding and decoding time, constructions that are currently out of reach. In conclusion, we hope that our work will ignite further work in the field, and towards the resolution of some of these questions.

References

1. Alexandr Andoni, Robert Krauthgamer, and Krzysztof Onak. Streaming algorithms via precision sampling. In *Proceedings of the 2011 IEEE 52Nd Annual Symposium on Foundations of Computer Science*, FOCS '11, pages 363–372, Washington, DC, USA, 2011. IEEE Computer Society.
2. R. Berinde, A. C. Gilbert, P. Indyk, H. Karloff, and M. J. Strauss. Combining geometry and combinatorics: A unified approach to sparse signal recovery. In *2008 46th Annual Allerton Conference on Communication, Control, and Computing*, pages 798–805, Sept 2008. doi:10.1109/ALLERTON.2008.4797639.
3. Radu Berinde, Piotr Indyk, Graham Cormode, and Martin J. Strauss. Space-optimal heavy hitters with strong error bounds. *ACM Trans. Database Syst.*, 35(4):26:1–26:28, 2010. doi:10.1145/1862919.1862923.
4. Albert Cohen, Wolfgang Dahmen, and Ronald DeVore. Compressed sensing and best k-term approximation. *Journal of the American mathematical society*, 22(1):211–231, 2009.
5. Graham Cormode and Shan Muthukrishnan. An improved data stream summary: the count-min sketch and its applications. *Journal of Algorithms*, 55(1):58–75, 2005.

6 S Ben Fred, Thomas Bonald, Alexandre Proutiere, Gwénaël Régnié, and James W Roberts. Statistical bandwidth sharing: a study of congestion at flow level. In *ACM SIGCOMM Computer Communication Review*, volume 31, pages 111–122. ACM, 2001.

7 Sumit Ganguly. Lower bounds on frequency estimation of data streams. *Lecture Notes in Computer Science*, 5010:204–215, 2008.

8 A. C. Gilbert, M. J. Strauss, J. A. Tropp, and R. Vershynin. One sketch for all: Fast algorithms for compressed sensing. In *Proceedings of the Thirty-ninth Annual ACM Symposium on Theory of Computing*, STOC '07, pages 237–246, New York, NY, USA, 2007. ACM.

9 Anna C Gilbert, Yi Li, Ely Porat, and Martin J Strauss. Approximate sparse recovery: optimizing time and measurements. *SIAM Journal on Computing*, 41(2):436–453, 2012.

10 Anna C. Gilbert, Yi Li, Ely Porat, and Martin J. Strauss. For-all sparse recovery in near-optimal time. *ACM Trans. Algorithms*, 13(3):32:1–32:26, 2017.

11 Nicholas J. A. Harvey, Jelani Nelson, and Krzysztof Onak. Sketching and streaming entropy via approximation theory. In *Proceedings of the 2008 49th Annual IEEE Symposium on Foundations of Computer Science*, FOCS '08, pages 489–498, Washington, DC, USA, 2008. IEEE Computer Society.

12 P. Indyk and M. Ruzic. Near-optimal sparse recovery in the l1 norm. In *Proceedings of the 49th Annual IEEE Symposium on Foundations of Computer Science*, pages 199–207, Oct 2008. doi:10.1109/FOCS.2008.82.

13 T. S. Jayram and D. P. Woodruff. The data stream space complexity of cascaded norms. In *2009 50th Annual IEEE Symposium on Foundations of Computer Science*, pages 765–774, Oct 2009.

14 Hossein Jowhari, Mert Sağlam, and Gábor Tardos. Tight bounds for lp samplers, finding duplicates in streams, and related problems. In *Proceedings of the Thirtieth ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems*, PODS '11, pages 49–58, New York, NY, USA, 2011. ACM.

15 Daniel M. Kane and Jelani Nelson. Sparser johnson-lindenstrauss transforms. *J. ACM*, 61(1):4:1–4:23, 2014. doi:10.1145/2559902.

16 Kasper Green Larsen, Jelani Nelson, Huy L Nguyen, and Mikkel Thorup. Heavy hitters via cluster-preserving clustering. In *Foundations of Computer Science (FOCS), 2016 IEEE 57th Annual Symposium on*, pages 61–70. IEEE, 2016.

17 Yi Li, Huy L. Nguyen, and David P. Woodruff. Turnstile streaming algorithms might as well be linear sketches. In *Proceedings of the Forty-sixth Annual ACM Symposium on Theory of Computing*, STOC '14, pages 174–183, New York, NY, USA, 2014. ACM. doi:10.1145/2591796.2591812.

18 Gurmeet Singh Manku and Rajeev Motwani. Approximate frequency counts over data streams. In *Proceedings of the 28th International Conference on Very Large Data Bases*, VLDB '02, pages 346–357. VLDB Endowment, 2002. URL: http://dl.acm.org/citation.cfm?id=1287369.1287400.

19 Ahmed Metwally, Divyakant Agrawal, and Amr El Abbadi. *Efficient Computation of Frequent and Top-k Elements in Data Streams*, pages 398–412. Springer Berlin Heidelberg, Berlin, Heidelberg, 2005.

20 Morteza Monemizadeh and David P. Woodruff. 1passs relative-error lp-sampling with applications. In *Proceedings of the Twenty-first Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '10, pages 1143–1160, Philadelphia, PA, USA, 2010. Society for Industrial and Applied Mathematics.

21 Tatsuya Mori, Ryoichi Kawahara, Shozo Naito, and Shigeki Goto. On the characteristics of internet traffic variability: Spikes and elephants. *IEICE TRANSACTIONS on Information and Systems*, 87:2644–2653, 2004.

22 Jelani Nelson, Huy L. Nguyên, and David P. Woodruff. On deterministic sketching and streaming for sparse recovery and norm estimation. In *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques - 15th International Workshop, APPROX 2012, and 16th International Workshop, RANDOM 2012, Cambridge, MA, USA, August 15-17, 2012. Proceedings*, pages 627–638, 2012.

23 Konstantina Papagiannaki, Nina Taft, S Bhattacharya, Patrick Thiran, Kave Salamatian, and Christophe Diot. On the feasibility of identifying elephants in internet backbone traffic. *Sprint Labs, Sprint ATL, Tech. Rep. TR01-ATL-110918*, 2001.

24 Mark S. Pinsker. On the complexity of a concentrator. In *Proceedings of 7th International Teletraffic Conference*, 1973.

25 Ely Porat and Martin J. Strauss. Sublinear time, measurement-optimal, sparse recovery for all. In *Proceedings of the Twenty-third Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '12, pages 1215–1227, 2012.

26 Liang Yang, Bryan Ng, and Winston KG Seah. Heavy hitter detection and identification in software defined networking. In *Computer Communication and Networks (ICCCN), 2016 25th International Conference on*, pages 1–10. IEEE, 2016.

27 Yin Zhang, Lee Breslau, Vern Paxson, and Scott Shenker. On the characteristics and origins of internet flow rates. In *ACM SIGCOMM Computer Communication Review*, volume 32, pages 309–322. ACM, 2002.

28 Yu Zhang, BinXing Fang, and YongZheng Zhang. Identifying heavy hitters in high-speed network monitoring. *Science China Information Sciences*, 53(3):659–676, 2010.

A Proof of Lemma 9

Proof. Suppose that $|D| > \theta s$, otherwise the result holds automatically. Assume that $|x_1| \geq |x_2| \geq \cdots \geq |x_n|$. Let $T = D \cup \{i : |x_i| \geq \epsilon\gamma/4\}$, then $t := |T| \leq \|z\|_1/(\epsilon\gamma/4) + |D| \leq 6/(\epsilon\gamma)$.

Note that $|x_{t+1}| \leq \epsilon\gamma/4$. Taking $\alpha = 2$ in [10, Lemma 3.3], we know that

$$\|(\Phi(x - x_{[t]}))_{\Gamma(D)}\|_1 \leq 4 \cdot \beta\epsilon d \left(\frac{3}{2} + 2s \cdot \frac{\epsilon\gamma}{4}\right) \leq 8\beta\epsilon d.$$

By the isolation property, there are at most $\frac{6}{\epsilon\gamma} \cdot \frac{\epsilon\theta}{12} = \frac{\theta}{2\gamma}$ elements in T which are not isolated in at least $(1-\zeta)d$ nodes from other elements in T. This implies that at least $\theta/(2\gamma)$ elements in D are isolated in at least $(1-\zeta)d$ nodes from other elements in T.

A decoy at position i receives at least $\epsilon\gamma/4$ noise in at least $(\beta-\zeta)d$ isolated nodes of $\Gamma(\{i\})$, hence in total, a decoy element receives at least $\epsilon\gamma(\beta-\zeta)d/4$ noise. Therefore at least $\theta/(2\gamma)$ decoys overall should receive noise at least

$$\frac{\epsilon\gamma(\beta-\zeta)d}{4} \cdot \frac{\theta}{2\gamma} > 8\beta\epsilon d \geq \|(\Phi(x - x_{[t]}))_{\Gamma(D)}\|_1,$$

which is a contradiction. Therefore there are at most θs decoys. ◀

B Proof of Lemma 10

Before presenting the proof, we sketch our setup of message encoding. Let $\mathrm{enc} : \{0,1\}^{\log n} \to \{0,1\}^{\mathcal{O}(\log n)}$ be an error-correcting code that corrects a constant fraction of errors in linear time. For notational convenience, let $m_i = \mathrm{enc}(i)$, the codeword for the binary representation of i. Furthermore, we break m_i into d_1 blocks of length $\Theta((\log n)/d_1)$ each, say, $m_i = (m_{i,1}, \ldots, m_{i,d_1})$.

Let G be a Δ-regular edge-expander graph on d_1 vertices, where Δ is an absolute constant (we may assume that d_1 is even and such edge expander exists by [24]). Let j be a node in G and denote its neighbours by $\Gamma_1(j), \ldots, \Gamma_\Delta(j)$. Let $idx(r, i)$ ($r \in [d_1]$ and $i \in [n]$) denote the index of the bucket where i is hashed in the r-th first-layer repetition. Construct the message

$$\bar{m}_{i,r} = m_{i,r} \circ idx(\Gamma_1(r), i) \circ \cdots \circ idx(\Gamma_\Delta(r), i), \qquad i \in [n], r \in [d_1],$$

where $\circ$ denotes concatenation of strings and $idx(\cdot, \cdot)$ is understood as its binary representation of $\log(B_1)$ bits.

Now for each index i we have d_1 blocks of message $\bar{m}_{i,1}, \ldots, \bar{m}_{i,d_1}$. We can protect each block using a constant-rate error correcting code which tolerates a constant fraction of error and decodes in polynomial time, so that if we can recover a fraction of $\bar{m}_{i,r}$ we can recover the entire message $\bar{m}_{i,r}$ efficiently. The high-level idea is then to recover a good fraction of $\{\bar{m}_{i,r}\}_{r\in[d]}$ for a good fraction of heavy hitters i, so that we can recover m_i using the linking information embedded in $\bar{m}_{i,r}$ and the clustering algorithm in [16]. Finally we decode m_i to obtain the corresponding index i.

Proof of Lemma 10. We follow the construction and the argument as in [10]. We instantiate the two-layer hashing and the encoding scheme as in Section 3.1, where $\alpha \in (1, 2)$, $B_1 = \Theta(s^\alpha/\epsilon^{2\alpha})$, $d_1 = \Theta(\epsilon^{-1}\frac{\log n}{\log(B_1/s)})$, $B_2 = \Theta(s/\epsilon)$ and $d_2 = \Theta(\log(B_1/s))$. By Lemma 8 we can find a two-layer hashing with these prescribed parameters which satisfies $(4s, d_1d_2, O(\epsilon))$-expansion property and $(O(s/\epsilon), O(\epsilon), \Theta(1))$-isolation property. It is also easy to verify that the length of each message block $\bar{m}_{i,r}$ is $L = \Theta(\log(B_1/s)) + \Delta\log(B_1) \le d_2/2$ if we choose d_2 large enough. We can use two second-layer measurements to encode 1 bit of message by replacing a single entry a in the measurement matrix with a 2×1 block of $\binom{a}{0}$ or $\binom{0}{a}$, depending on the bit to encode. To decode the bit, suppose that the two corresponding measurements are $\binom{a}{b}$ and we convert them back to 0 if $|a| < |b|$ and 1 if $|a| \ge |b|$. When the heavy hitter is isolated and the noise is small in a bucket, the bit is expected to be the corresponding bit in the message for that heavy hitter.

Next we shall show that we can recover most bits of the message for at least a large constant fraction of heavy hitters. Invoking Lemma 9 with $\delta = O(1)$, $\theta = O(1)$, $\beta = O(1)$ and $\gamma = 1/s$, and following the argument in [10, Section 4.1], we have good estimates for all but at most $s/8$ heavy hitters (elements in $\mathrm{supp}(y)$). Call those heavy hitters *well-estimated*. The two-layer hashing eventually hashes n coordinates into B_2 buckets and repeat d_1d_2 times, and we know that each well-estimated heavy hitter i receives small noise in at least $(1-\delta)d_1d_2$ repetitions. This implies that there exist δ_1 and δ_2 such that for each well-estimated heavy hitter i, there exist $(1-\delta_1)d_1$ first-layer repetitions such that in each such first-layer repetition r the heavy hitter i receives small noise in at least $(1-\delta_2)d_2$ second-layer repetitions. For each such pair (i, r), we can recover at least $(1-\delta_2)$ fraction of the message $\bar{m}_{i,r}$, and if we protect $\bar{m}_{i,r}$ using a constant-rate error-correcting code (e.g. Reed-Solomon code) that can tolerate up to δ_2 fraction of error, we shall recover $\bar{m}_{i,r}$ in entirety. To summarize, for each well-estimated heavy hitter i, we can recover $\bar{m}_{i,r}$ for at least $(1-\delta_1)d_1$ values of $r \in [d_1]$. We note that δ_1 can made arbitrarily small by adjusting the constants in the two-layer construction and making δ arbitrarily small.

Now we construct the chunk graph as in [16]. The chunk graph has B_1d_1 nodes, indexed by pairs (b, r) for $b \in [B_1]$ and $r \in [d_1]$. For each bucket b in the first-repetition r, we recover a message of length L, break it up into blocks of the same structure as in $\hat{m}$ and extract the linking information $q_1(b, r), \ldots, q_\Delta(b, r)$. We say in $\tilde{G}$ the node (b, r) makes suggestion to connect to $(q_\ell(b, r), \Gamma_\ell(r))$, and we add an edge if both endpoints suggest each other. By the argument in [16, Lemma 2], a well-estimated heavy hitter i corresponds to an ϵ_0-spectral

cluster of $\tilde{G}$ for some small $\epsilon_0 > 0$. The spectral clustering algorithm ([16, Theorem 1]) will find all those spectral clusters, recovering a constant fraction of message m_i and enabling us to identify the index i of the heavy hitter. In each first-layer we retain the bucket of magnitude at least $\epsilon/(4s)$ so there are $O(s/\epsilon)$ buckets, and we therefore have a candidate list of size $\mathcal{O}(s/\epsilon)$ which misses at most $s/8$ heavy hitters. Finally we evaluate every candidate and retain the biggest s ones (in magnitude).

Each recovered coordinate is estimated to within $\epsilon/(4s)$, thus

$$\|\hat{z}\|_1 \leq \|z\|_1 + |\operatorname{supp}(\hat{x})| \cdot \frac{\epsilon}{4s} \leq \|z\|_1 + \frac{\epsilon}{4}.$$

The total number of measurements is $B_2 d_1 d_2 = \mathcal{O}(\epsilon^{-2} s \log n)$. For each first-layer repetition, we enumerate all coordinates in the bucket of size B_1, and decode the associated message (which is of length d_2), which takes time $\mathcal{O}(B_1 \operatorname{poly}(d_2)) = \mathcal{O}(s^\alpha \operatorname{poly}(1/\epsilon, \log n))$. We then run the spectral clustering algorithm on a graph of size $O(s/\epsilon \cdot d_1)$ in time $\tilde{\mathcal{O}}((s/\epsilon \cdot d_1)^3) = \mathcal{O}(s^3 \operatorname{poly}(1/\epsilon, \log n))$. To obtain the indices of the candidates, We decode $\mathcal{O}(s/\epsilon \cdot d_1) = \mathcal{O}(\epsilon^{-2} s \log n)$ messages, and the decoding algorithm on each $\Theta(\log n)$-bit-long m_i with a constant fraction corruption runs in time $O(\operatorname{poly}(\log n))$. Lastly we estimate each of the candidates and retain the biggest s ones, which takes time $\mathcal{O}((s/\epsilon) d_1 \cdot B_2 d_1 d_2) = \mathcal{O}(s^2 \operatorname{poly}(1/\epsilon, \log n))$. The overall runtime is dominated by the clustering algorithm and is therefore $\mathcal{O}(s^3 \operatorname{poly}(1/\epsilon, \log n))$. ◀

C Proof of Lemma 12

We follow the argument in [10]. We instantiate the two-layer hashing and the encoding scheme as in Section 3.1, where $\alpha \in (1, 2)$, $B_1 = \Theta(k^{2\alpha})$, $d_1 = \Theta(\frac{k}{\sqrt{sw}} \frac{\log n}{\log(B_1/s)})$, $B_2 = \Theta(k\sqrt{s/w})$ and $d_2 = \Theta(\log(B_1/s))$. By Lemma 8 (to see the conditions hold, replace k with s and ϵ with $k/\sqrt{sw}$), we can find a two-layer hashing with these prescribed parameters which satisfies $(\Theta(k), d_1 d_2, \frac{\sqrt{sw}}{k})$-expansion property and $(\Theta(k), \frac{\sqrt{sw}}{k}, \Theta(1))$-isolation property. The constants in the Θ-notations above all depend on η. It is also easy to verify that the length of each message block $\bar{m}_{i,r}$ is $L = \Theta(\log(B_1/s)) + \Delta \log(B_1) \leq d_2$ if we choose d_2 large enough.

Invoking Lemma 9 with $\delta = \Theta(\sqrt{sw}/k)$, $\theta = \Theta(\sqrt{sw}/k)$, $\epsilon = 1$ and $\gamma = 1/k$, and following the argument in [10, Section 4.1], we have good estimates for all but at most $\Theta(\theta/\gamma) = \sqrt{sw}$ heavy hitters (elements in $\operatorname{supp}(y)$). Call those heavy hitters *well-estimated*. Following the same argument as in the proof of Lemma 12, we can, for each well-estimated heavy hitter i, recover $\bar{m}_{i,r}$ for at least $(1 - \delta_1)d_1$ values of $r \in [d_1]$. We note that δ_1 can made arbitrarily small by adjusting the constants in the two-layer construction and making δ arbitrarily small.

We construct the chunk graph $\tilde{G}$ as in [16]. By the argument in [16, Lemma 2], a well-estimated heavy hitter i corresponds to an ϵ_0-spectral cluster of $\tilde{G}$ for some small $\epsilon_0 > 0$. The spectral clustering algorithm ([16, Theorem 1]) will find all those spectral clusters, recovering a constant fraction of message m_i and enabling us to identify the index i of the heavy hitter. In each first-layer we retain the bucket of magnitude at least $1/(4k)$ so there are $O(k)$ buckets, and we therefore have a candidate list of size $\mathcal{O}(k)$ which misses at most $\sqrt{sw}$ heavy hitters. Finally we evaluate every candidate and retain the biggest s ones (in magnitude).

Each recovered coordinate is estimated to within $\epsilon\gamma/4 \leq 1/(4k)$, thus

$$\|\hat{z}\|_1 \leq \|z\|_1 + \frac{|\operatorname{supp}(\hat{x})|}{4k} \leq 2 - \frac{s}{k} + \frac{s}{2k} \leq 2 - \frac{s}{2k}.$$

The total number of rows is $B_2 d_1 d_2 = \mathcal{O}(\frac{k^2}{w} \log n)$. For each first-layer repetition, we enumerate all coordinates in the bucket of size B_1, and decode the associated message (which

is of length d_2), which takes time $\mathcal{O}(B_1 \operatorname{poly}(d_2)) = \mathcal{O}(k^{2\alpha} \operatorname{poly}(\log k))$. We then run the spectral clustering algorithm on a graph of size $\mathcal{O}(kd_1)$ in time $\mathcal{O}((kd_1)^3) = \mathcal{O}(k^6 \operatorname{poly}(\log n))$. To obtain the indices of the candidates, We decode $\mathcal{O}(kd_1) = \mathcal{O}(k^2 \log n)$ messages, and the decoding algorithm on each $\Theta(\log n)$-bit-long m_i with a constant fraction corruption runs in time $O(\operatorname{poly}(\log n))$. Lastly we estimate each of the candidates and retain the biggest s ones, which takes time $\mathcal{O}(kd_1 \cdot B_2 d_1 d_2) = \mathcal{O}(k^4 \log^2 n)$. The overall runtime is dominated by the clustering algorithm and is therefore $\mathcal{O}(s^6 \operatorname{poly}(\log n)) = \mathcal{O}(k^6 \operatorname{poly}(\log n))$.

D Proof of Theorem 11

Without loss of generality, assume that $\|x_{-k}\|_1 = 1$. We shall apply Lemma 12 repeatedly to obtain a sequence of vectors $\hat{x}^{(i,r)}$, which admit decompositions $x^{(i,r)} = x - \hat{y}^{(i,r)} - \hat{z}^{(i,r)}$. Consider the following loop invariants, parametrized by (i, r), at the beginning of the i-th step in the r-th round:

$$\begin{aligned} |\operatorname{supp}(\hat{y}^{(i,r)})| &\le s_{i,r} := (i+1)^2 k_r^{2^{-i}} \\ \|\hat{z}^{(i,r)}\|_1 &\le 2 - s_{i,r}/k \end{aligned} \tag{3}$$

We claim that the loop invariants above hold for $(0, r)$ for $r = O(\log^* k)$. The base case is $(0, 0)$ and the loop invariants holds trivially. Suppose that the loop invariants hold for (i, r), we shall show that it holds for $(i, r+1)$ whenever $r \le r_0$ for some $r_0 = O(\log^* k)$.

To prove the inductive step w.r.t. r, we consider an inductive proof w.r.t. i for a fixed r. For the inductive step, if $i < i_r^*$ we apply Lemma 12 with $w_{i,r} = (i+1)^2$ and $s_{i,r} = (i+1)^2 k_r^{2^{-i}}$. We then get that

$$|\operatorname{supp}(\hat{y}^{(i,r)})| \le \sqrt{s_{i,r} w_{i,r}} = (i+1)^2 k_r^{2^{-(i+1)}} \le s_{i+1,r},$$

and

$$\|\hat{z}^{(i,r)}\|_1 \le 2 - s_{i,r}/(2k) \le 2 - s_{i+1,r}/k$$

when $s_{i+1,r} \le s_{i,r}/2$, that is, when $4(1 + \frac{1}{i+1})^4 \le k_r^{2^{-i}}$.

This proves the loop invariants (3) when $k_r^{2^{-i}} \ge \max\{(i+1)^2, 4(1 + \frac{1}{i+1})^4\}$, and that is $i \le i_r^*$ for some $i_r^* = \mathcal{O}(\log \log k_r)$. At this stage, the residual admits the decomposition $y^{(i_r^*,r)} + z^{(i_r^*,r)}$ with $|\operatorname{supp}(y^{(i_r^*,r)})| \le \mathcal{O}(\log^4 \log k)$ and $\|z^{(i,r)}\|_1 \le 2 - s_{i_r^*,r}/k$.

Now we change our choice of parameters and the loop invariants. In the i-th step ($i \ge i_r^* + 1$), we claim the following invariants hold at the beginning of the i-th step by changing $w_{i,r}$ to $w_{i,r} = 1$:

$$\begin{aligned} |\operatorname{supp}(\hat{y}^{(i,r)})| &\le s_{i,r} := (s_{i_r^*,r})^{2^{-(i-i_r^*-1)}} \\ \|\hat{z}^{(i,r)}\|_1 &\le 2 - s_{i,r}/k \end{aligned} \tag{4}$$

By our choice of i_r^* and the argument above the invariants hold when $i = i_r^* + 1$. Applying Lemma 12 with $w_{i,r} = 1$, we see that

$$|\operatorname{supp}(\hat{y}^{(i,r)})| \le \sqrt{s_{i,r}} \le s_{i+1,r}$$

and

$$\|\hat{z}^{(i,r)}\|_1 \le 2 - s_{i,r}/(2k_r) \le 2 - s_{i+1,r}/k,$$

whenever $s_{i,r} \geq 4$. This proves the loop invariants (4) when $s_{i,r} \geq \max\{4, \log\log k_r\}$, which holds when $i \leq i_r^* + i_r^+$ for some $i_r^+ = O(1)$ (recall that $s_{i_r^*} = O(\log^4 \log k_r)$). These steps leaves us a decomposition of the residual as $y^{(i,r)} + z^{(i,r)}$, where $|\operatorname{supp}(y^{(i,r)})| = s_{i,r} \leq \max\{4, \log\log k_r\}$ and $\|z^{(i,r)}\|_1 \leq 2 - s_{i,r}/k$.

Next, we start a new round by setting $k_{r+1} = s_{0,r+1} = s_{i_r^* + i_r^+ + 1, r}$. The loop invariants in (3) continue to hold in the base case $i = 0$. This completes proof of the claim that the loop invariants hold for $(0, r+1)$, provided that $k_{r+1} > 4$. Since $k_{r+1} \leq \log\log k_r$ and $k_0 = k$, the loop invariants in (3) hold for all $r \leq r_0$ for some $r_0 = O(\log^* k)$.

When $k_{r+1} \leq 4$, that is, there are at most 4 heavy hitter left, we shall recover all of them in one call to the weak system. Setting $w < 1/4$ in Lemma 12 yields that $|\operatorname{supp}(\hat{y})| < 1$; it thus must hold that $|\operatorname{supp}(\hat{y})| = 0$, or $\hat{y} = 0$, which means that all heavy hitters have recovered. This last call recovers $\hat{x}_{\text{final}}$, which has support size $O(1)$.

Support size of output. The support size of the output $\hat{x}$ is upper bounded by

$$|\operatorname{supp}(\hat{x}_{\text{final}})| + \sum_{r,i} |\operatorname{supp}(\hat{x}^{(i,r)})| \leq \mathcal{O}(1) + \sum_r \mathcal{O}(k_r) = \mathcal{O}(k).$$

Number of rows. In all rounds except the last round, the number of rows is bounded by

$$\mathcal{O}\left(\sum_{i=0}^{i_r^*} \frac{k^2}{(i+1)^2} \log n\right) + \mathcal{O}(i_r^+ \cdot k^2 \log n) = \mathcal{O}(k^2 \log n)$$

and the last round needs $\mathcal{O}(k^2 \log n)$ rows. The overall number of rows is therefore $m = \mathcal{O}(k^2 \log n \log^* k)$ as there are $\mathcal{O}(\log^* k)$ rounds.

Runtime. Each call to the weak system runs in $\mathcal{O}(k^6 \operatorname{poly}(\log n))$ time and there are $(\sum_r (i_r^* + i_r^+)) + 1 = \mathcal{O}(\log\log k \cdot \log^* k)$ calls. Each update of $y^{(i+1,r)} \leftarrow y^{(i,r)} - \Phi\hat{x}^{(i,r)}$ takes $\mathcal{O}(mk)$ since Φ has m rows and $|\operatorname{supp}(\hat{x})| = \mathcal{O}(k)$; there are $O(\log\log k \cdot \log^* k)$ such updates. The overall runtime is therefore $\mathcal{O}(k^6 \operatorname{poly}(\log n))$.

Storage of the sketching matrix. Each weak system uses $\mathcal{O}(k \log n)$ random $\mathcal{O}(k)$-wise independent hash function and needs space $\mathcal{O}(k^2 \log n)$ words. We have $\mathcal{O}(\log\log k \cdot \log^* k)$ such hash functions and thus the total storage for sketching matrix is $\mathcal{O}(k^2 \log n \cdot \log\log k \log^* k)$ words.

E Proof of Theorem 1

We first pick a matrix A using Theorem 3, setting the sparsity parameter to k^2 and $\epsilon = 1$. We also pick a matrix B satisfying the guarantees of Theorem 11, with sparsity $6k$, and a matrix C using Lemma 13 with sparsity parameter $6k$. Our sketching matrix Φ is the vertical concatenation of A, B and C. The total number of rows is $\mathcal{O}(k^2 \log n)$ for A and C, and $\mathcal{O}(k^2 \log n \log^* k)$ for B, for a total of $\mathcal{O}(k^2 \log n \log^* k)$ rows.

We first run the algorithm on Ax to obtain an $\mathcal{O}(k^2)$-sparse vector z such that $\|x - z\|_1 \leq 2\|x_{-k^2}\|_1$. Then we form $B(x-z)$ and using the query algorithm for B, we find an $\mathcal{O}(k)$-sparse vector w such that

$$\|(x-z) - w\|_\infty \leq \frac{1}{2k}\|x - z\|_1 \leq \frac{1}{k}\|x_{-k^2}\|_1. \tag{5}$$

Let $\mathcal{H}$ be the set of coordinates $i \in [n]$ such that $|x_i| > \frac{1}{k}\|x_{-k^2}\|_1$. We claim that $\mathcal{H} \subseteq \text{supp}(z) \cup \text{supp}(w)$; otherwise, it holds for $i \in \mathcal{H}$ that

$$|((x-z)-w)_i| = |x_i| > \frac{1}{k}\|x_{-k^2}\|_1,$$

which contradicts (5). The next step is to estimate x_i, for every $i \in \text{supp}(z) \cup \text{supp}(w)$, up to $(1/k)\|x_{-k^2}\|_1$ error. This argument is almost identical to [22], but we include it here for completeness. For every such i, define vector z' to be equal to z but with the i-th coordinate zeroed out. Then we run the point query algorithm of Lemma 13 on sparsity parameter $6k$ with sketch $C(x-z')$ to obtain a value $\hat{x}_i$ such that

$$|\hat{x}_i - x_i| = |\hat{x}_i - (x-z')_i| \leq \frac{1}{6k}\|(x-z')_{[n]\setminus\{i\}}\|_1 \leq \frac{1}{6k}\|x-z\|_1 \leq \frac{1}{3k}\|x_{-k^2}\|_1.$$

We note that $|\mathcal{H}| \leq k$ and, hence, by keeping the top $4k$ coordinates in magnitude, we shall include all elements in $\mathcal{H}$. Otherwise, there are at least $3k$ estimates of value at least $\frac{2}{3k}\|x_{-k^2}\|_1$ and so there are at least $3k$ coordinates of $x_{\text{supp}(z)\cup\text{supp}(w)}$ of magnitude at least $\frac{1}{3k}\|x_{-k^2}\|_1$, which is impossible. This concludes the proof of correctness.

Running time. The first step of obtaining z takes time $\mathcal{O}(k^3 \operatorname{poly}(\log n))$ by Theorem 3. The second step of obtaining w takes time $\mathcal{O}(k^6 \operatorname{poly}(\log n))$ by Theorem 11. The third step makes $\mathcal{O}(k^2)$ point queries. For each point query, it computes $C(x-z') = Cx - Cz'$, where Cx is part of the overall sketch and Cz' can be efficiently computed in $\mathcal{O}(k^4 \log n)$ time since C has $\mathcal{O}(k^2 \log n)$ rows and z' is $\mathcal{O}(k^2)$-sparse. Then the point query procedure itself runs in time $\mathcal{O}(k \log n)$ by Lemma 13. The total runtime of the third step is thus $\mathcal{O}(k^6 \log n)$. The overall runtime is dominated by that of the second step.

Storage space. The space to store A is $\mathcal{O}(k^2 \log n)$ words by Theorem 3. The space to store B is $\mathcal{O}(k^2 \log n \cdot \log^* k \cdot \log\log k)$ words by Theorem 11. The space to store C is $\mathcal{O}(k^2 \log n)$ words by taking C to be a fast Johnson-Lindenstrauss Transform matrix [15]. The overall storage space is dominated by that of B.