2016 IEEE 57th Annual Symposium on Foundations of Computer Science

Heavy hitters via cluster-preserving clustering

Kasper Green Larsen
Aarhus University
larsen@cs.au.dk

Jelani Nelson
Harvard University
minilek@seas.harvard.edu

Huy L. Nguyễn
Northeastern University
hlnguyen@cs.princeton.edu

Mikkel Thorup
University of Copenhagen
mikkel2thorup@gmail.com

***Abstract*—In the turnstile ℓ_p heavy hitters problem with parameter ε, one must maintain a high-dimensional vector $x \in \mathbb{R}^n$ subject to updates of the form $\texttt{update}(i, \Delta)$ causing the change $x_i \leftarrow x_i + \Delta$, where $i \in [n]$, $\Delta \in \mathbb{R}$. Upon receiving a query, the goal is to report every "heavy hitter" $i \in [n]$ with $|x_i| \geq \varepsilon \|x\|_p$ as part of a list $L \subseteq [n]$ of size $O(1/\varepsilon^p)$, i.e. proportional to the maximum possible number of heavy hitters.**

For any $p \in (0, 2]$ the CountSketch of [CCFC04] solves ℓ_p heavy hitters using $O(\varepsilon^{-p} \lg n)$ words of space with $O(\lg n)$ update time, $O(n \lg n)$ query time to output L, and whose output after any query is correct with high probability (whp) $1 - 1/\operatorname{poly}(n)$ [JST11, Section 4.4]. This space bound is optimal even in the strict turnstile model [JST11] in which it is promised that $x_i \geq 0$ for all $i \in [n]$ at all points in the stream, but unfortunately the query time is very slow. To remedy this, the work [CM05] proposed the "dyadic trick" for the CountMin sketch for $p = 1$ in the strict turnstile model, which to maintain whp correctness achieves suboptimal space $O(\varepsilon^{-1} \lg^2 n)$, worse update time $O(\lg^2 n)$, but much better query time $O(\varepsilon^{-1} \operatorname{poly}(\lg n))$. An extension to all $p \in (0, 2]$ appears in [KNPW11, Theorem 1], and can be obtained from [Pag13].

We show that this tradeoff between space and update time versus query time is unnecessary. We provide a new algorithm, ExpanderSketch, which in the most general turnstile model achieves optimal $O(\varepsilon^{-p} \lg n)$ space, $O(\lg n)$ update time, and fast $O(\varepsilon^{-p} \operatorname{poly}(\lg n))$ query time, providing correctness whp. In fact, a simpler version of our algorithm for $p = 1$ in the strict turnstile model answers queries even *faster* than the "dyadic trick" by roughly a $\lg n$ factor, dominating it in all regards. Our main innovation is an efficient reduction from the heavy hitters to a clustering problem in which each heavy hitter is encoded as some form of noisy spectral cluster in a much bigger graph, and the goal is to identify every cluster. Since every heavy hitter must be found, correctness requires that every cluster be found. We thus need a "cluster-preserving clustering" algorithm, that partitions the graph into clusters with the promise of not destroying any original cluster. To do this we first apply standard spectral graph partitioning, and then we use some novel combinatorial techniques to modify the cuts obtained so as to make sure that the original clusters are sufficiently preserved. Our cluster-preserving clustering may be of broader interest much beyond heavy hitters.

KGL is supported by Center for Massive Data Algorithmics, a Center of the Danish National Research Foundation, grant DNRF84, a Villum Young Investigator Grant and an AUFF Starting Grant. JN is supported by NSF grant IIS-1447471 and CAREER award CCF-1350670, ONR Young Investigator award N00014-15-1-2388, and a Google Faculty Research Award. MT is supported in part by Advanced Grant DFF-0602-02499B from the Danish Council for Independent Research under the Sapere Aude research career programme.

I. Introduction

Finding *heavy hitters* in a data stream, also known as *elephants* or *frequent items*, is one of the most practically important and core problems in the study of streaming algorithms. The most basic form of the problem is simple: given a long stream of elements coming from some large universe, the goal is to report all frequent items in the stream using an algorithm with very low memory consumption, much smaller than both the stream length and universe size.

In practice, heavy hitters algorithms have been used to find popular destination addresses and heavy bandwidth users by AT&T [CJK$^+$04], to find popular search query terms at Google [PDGQ05], and to answer so-called "iceberg queries" in databases [FSG$^+$98] (to name a few of many applications). A related problem is finding superspreaders [LCG13], the "heavy hitters" in terms of number of distinct connections, not total bandwidth.

In the theoretical study of streaming algorithms, finding heavy hitters has played a key role as a subroutine in solving many other problems. For example, the first algorithm providing near-optimal space complexity for estimating ℓ_p norms in a stream for $p > 2$ needed to find heavy hitters as a subroutine [IW05], as did several follow-up works on the same problem [GB09], [AKO11], [BO13], [BKSV14], [Gan15]. Heavy hitters algorithms are also used as a subroutine in streaming entropy estimation [CCM10], [HNO08], ℓ_p-sampling [MW10], cascaded norm estimation and finding block heavy hitters [JW09], finding duplicates [GR09], [JST11], fast ℓ_p estimation for $0 < p < 2$ [NW10], [KNPW11], and for estimating more general classes of functions of vectors updated in a data stream [BO10], [BC15].

In this work we develop a new heavy hitters algorithm, ExpanderSketch, which reduces the heavy hitters problem to a new problem we define concerning graph clustering, which is an area of interest in its own right. Our special goal is to identify *all* clusters of a particular type, even if they are much smaller than the graph we are asked to find them in. While our algorithm as presented is rather theoretical with large constants, the ideas in it are simple with potential practical impact.

The particular formulation we focus on is finding clusters with low external conductance in the graph, and with good connectivity properties internally (a precise definition of these clusters, which we call *ϵ-spectral clusters*, will be given soon). Similar models have been used for finding a community within a large network, and algorithms for identifying them have applications in community detection and network analysis. In computer vision, many segmentation algorithms such as [SM00], [AMFM11] model images as graphs and use graph partitioning to segment the images. In theory, there has been extensive study on the problem with many approximation algorithms [LR88], [ARV09]. Our focus in this work is on theoretical results, with constants that may be unacceptable in practice.

II. Previous work

In this work we consider the ℓ_p heavy hitters problem for $0 < p \leq 2$. A vector $x \in \mathbb{R}^n$ is maintained, initialized to the all zero vector. A parameter $\varepsilon \in (0, 1/2)$ is also given. This is a data structural problem with updates and one allowed query that are defined as follows.

- `update`(i, Δ) : Update $x_i \leftarrow x_i + \Delta$, where $i \in [n]$ and $\Delta \in \mathbb{R}$ has finite precision.
- `query()` : Return a set $L \subseteq [n]$ of size $|L| = O(\varepsilon^{-p})$ containing all ε-heavy hitters $i \in [n]$ under ℓ_p. Here we say i is an ε-heavy hitter under ℓ_p if $|x_i| \geq \varepsilon \|x_{\overline{[1/\varepsilon^p]}}\|_p$, where $x_{\overline{[k]}}$ denotes the vector x with the largest k entries (in absolute value) set to zero. Note the number of heavy hitters never exceeds $2/\varepsilon^p$.

Unless stated otherwise we consider randomized data structures in which any individual query has some failure probability δ. Note our definition of ε-heavy hitters includes all coordinates satisfying the usual definition requiring $|x_i| \geq \varepsilon \|x\|_p$, and potentially more. One can thus recover all the usual heavy hitters from our result by, in post-processing, filtering L to only contain indices that are deemed large by a separate CountSketch run in parallel. A nice feature of the more stringent version we solve is that, by a reduction in [JST11, Section 4.4], solving the ℓ_2 version of the problem with error $\varepsilon' = \varepsilon^{p/2}$ implies a solution for the ℓ_p ε-heavy hitters for any $p \in (0, 2]$. Hence for the remainder of the paper, assume we discuss $p = 2$ unless stated otherwise. We also note that any solution to ℓ_p heavy hitters for $p > 2$, even for constant ε, δ, must use polynomial space $\Omega(n^{1-2/p})$ bits [BYJKS04].

Before describing previous work, we first describe, in the terminology of [Mut05], three different streaming models that are frequently considered.

- **Cash-register:** This model is also known as the *insertion-only* model, and it is characterized by the fact that all updates `update`(i, Δ) have $\Delta = 1$.
- **Strict turnstile:** Each update Δ may be an arbitrary positive or negative number, but we are promised that $x_i \geq 0$ for all $i \in [n]$ at all points in the stream.
- **General turnstile:** Each update Δ may be an arbitrary positive or negative number, and there is no promise that $x_i \geq 0$ always. Entries in x may be negative.

It should be clear from the definitions that algorithms that work correctly in the general turnstile model are the most general, followed by strict turnstile, then followed by the cash-register model. Similarly, lower bounds proven in the cash-register model are the strongest, and those proven in the general turnstile model are the weakest. The strict turnstile model makes sense in situations when deletions are allowed, but items are never deleted more than they are inserted. The general turnstile model is useful when computing distances or similarity measures between two vectors z, z', e.g. treating updates (i, Δ) to z as $+\Delta$ and to z' as $-\Delta$, so that x represents $z - z'$.

For the heavy hitters problem, the first algorithm with provable guarantees was a deterministic algorithm for finding ℓ_1 heavy hitters in the cash-register model [MG82], for solving the non-tail version of the problem in which heavy hitters satisfy $|x_i| \geq \varepsilon \|x\|_1$. The space complexity and query time of this algorithm are both $O(1/\varepsilon)$, and the algorithm can be implemented so that the update time is that of insertion into a dynamic dictionary on $O(1/\varepsilon)$ items [DLM02] (and thus expected $O(1)$ time if one allows randomness, via hashing). We measure running time in the word RAM model and space in machine words, where a single word is assumed large enough to hold the maximum $\|x\|_\infty$ over all time as well as any $\lg n$ bit integer. The bounds achieved by [MG82] are all optimal, and hence ℓ_1 heavy hitters in the cash-register model is completely resolved. An improved analysis of [BICS10] shows that this algorithm also solves the tail version of heavy hitters. In another version of the problem in which one simultaneously wants the list L of ε heavy hitters together with additive $\varepsilon' \|x\|_1$ estimates for every $i \in L$, [BDW16] gives space upper and lower bounds which are simultaneously optimal in terms of ε, ε', n, and the stream length.

For ℓ_2 heavy hitters in the cash-register model, the CountSketch achieves $O(\varepsilon^{-2} \lg n)$ space. Recent works gave new algorithms using space $O(\varepsilon^{-2} \lg(1/\varepsilon) \lg \lg n)$ [BCIW16] and $O(\varepsilon^{-2} \lg(1/\varepsilon))$ [BCI$^+$16]. A folklore lower bound in the cash-register model is $\Omega(1/\varepsilon^2)$ machine words, which follows since simply encoding the names of the up to $1/\varepsilon^2$ heavy hitters requires $\Omega(\lg(\binom{n}{1/\varepsilon^2}))$ bits.

Despite the ℓ_1 and ℓ_2 heavy hitters algorithms above in the cash-register model requiring $o(\varepsilon^{-p} \lg n)$ machine words, it is known finding ε-heavy hitters under ℓ_p for any $0 < p \leq 2$ in the strict turnstile model requires $\Omega(\varepsilon^{-p} \lg n)$ words of memory, even for constant probability of success [JST11]. This optimal space complexity is achieved even in the more general turnstile model by the CountSketch of [CCFC04] for all $p \in (0, 2]$ (the original work [CCFC04] only analyzed the CountSketch for $p = 2$, but a a very short argument of [JST11, Section 4.4] shows that finding

reference	space	update time	query time	randomized?	norm
[CM05]	$\varepsilon^{-1}\lg n$	$\lg n$	$n\lg n$	Y	ℓ_1
[CM05]*	$\varepsilon^{-1}\lg^2 n$	$\lg^2 n$	$\varepsilon^{-1}\lg^2 n$	Y	ℓ_1
[NNW14]	$\varepsilon^{-2}\lg n$	$\varepsilon^{-1}\lg n$	$n\lg n$	N	ℓ_1
[CCFC04]	$\varepsilon^{-p}\lg n$	$\lg n$	$n\lg n$	Y	ℓ_p, $p \in (0,2]$
[KNPW11], [Pag13]*	$\varepsilon^{-p}\lg^2 n$	$\lg^2 n$	$\varepsilon^{-p}\lg^2 n$	Y	ℓ_p, $p \in (0,2]$
[CH08]	$\varepsilon^{-p}\lg n$	$\lg n$	$\varepsilon^{-p}\cdot n^{\gamma}$	Y	ℓ_p, $p \in (0,2]$
This work	$\varepsilon^{-p}\lg n$	$\lg n$	$\varepsilon^{-p}\operatorname{poly}(\lg n)$	Y	ℓ_p, $p \in (0,2]$

Figure 1. Comparison of turnstile results from previous work and this work for ℓ_p heavy hitters when desired failure probability is $1/\text{poly}(n)$. Space is measured in machine words. For [CH08], $\gamma > 0$ can be an arbitrarily small constant. If a larger failure probability $\delta \gg 1/\operatorname{poly}(n)$ is tolerable, one $\lg n$ factor in space, update time, and query time in each row with an asterisk can be replaced with $\lg((\lg n)/(\varepsilon\delta))$. For [Pag13], one $\lg n$ in each of those bounds can be replaced with $\lg(1/(\varepsilon\delta))$. For [CH08], the query time can be made $(\varepsilon^{-p}\lg n)((\lg n)/(\varepsilon\delta))^{\gamma}$.

ℓ_p heavy hitters for $p \in (0,2)$ reduces to finding ℓ_2 heavy hitters by appropriately altering ε).

For any $p \in (0,2]$, the CountSketch achieves optimal $O(\varepsilon^{-p}\lg n)$ space, the update time is $O(\lg n)$, the success probability of any query is with high probability (whp) $1 - 1/\operatorname{poly}(n)$, but the query time is a very slow $\Theta(n\lg n)$. In [CM05], for $p = 1$ in the strict turnstile model the authors described a modification of the CountMin sketch they dubbed the "dyadic trick" which maintains whp correctness for queries and significantly improves the query time to $O(\varepsilon^{-1}\lg^2 n)$, but at the cost of worsening the update time to $O(\lg^2 n)$ and space to a suboptimal $O(\varepsilon^{-1}\lg^2 n)$. An easy modification of the dyadic trick extends the same bounds to the general turnstile model and also to ε-heavy hitters under ℓ_p for any $0 < p \le 2$ with the same bounds [KNPW11, Theorem 1] (but with ε^{-1} replaced by ε^{-p}). A different scheme using error-correcting codes in [Pag13, Theorem 4.1] achieves the same exact bounds for ℓ_p heavy hitters when whp success is desired. This tradeoff in sacrificing space and update time for better query time has been the best known for over a decade for any $0 < p \le 2$.

From the perspective of *time* lower bounds, [LNN15] showed that any "non-adaptive" turnstile algorithm for constant ε and using $\operatorname{poly}(\lg n)$ space must have update time $\Omega(\sqrt{\lg n/\lg\lg n})$. A non-adaptive algorithm is one in which, once the randomness used by the algorithm is fixed (e.g. to specify hash functions), the cells of memory probed when processing `update`(i,Δ) depend only on i and not on the history of the algorithm. Note every known turnstile algorithm is non-adaptive (except one that only works for a promise problem).

In summary, there are several axes on which to measure the quality of a heavy hitters algorithm: space, update time, query time, and failure probability. The ideal algorithm should achieve optimal space, fast update time ($O(\lg n)$ is the best we know), nearly linear query time e.g. $O(\varepsilon^{-p}\operatorname{poly}(\lg n))$, and $1/\operatorname{poly}(n)$ failure probability. Various previous works were able to achieve various subsets of at most three out of four of these desiderata, but none could achieve all four simultaneously.

Our contribution I: We show the tradeoffs between space, update time, query time, and failure probability in previous works are unnecessary (see Figure 1). Specifically, in the most general turnstile model we provide a new streaming algorithm, ExpanderSketch, which for any $0 < p \le 2$ provides whp correctness for queries with $O(\lg n)$ update time, $O(\varepsilon^{-p}\operatorname{poly}(\lg n))$ query time, and optimal $O(\varepsilon^{-p}\lg n)$ space. In fact in the strict turnstile model and for $p = 1$, we are able to provide a simpler variant of our algorithm with query time $O(\varepsilon^{-1}\lg^{1+\gamma} n)$ for any constant $\gamma > 0$, answering queries even *faster* than the fastest previous known algorithms achieving suboptimal update time and space, by nearly a $\lg n$ factor, thus maintaining whp correctness while dominating it in all of the three axes: space, update time, and query time. Due to space constraints, this algorithm can be found in the full version of this paper [LNNT16].

A. Cluster-preserving clustering

Our algorithm ExpanderSketch operates by reducing the heavy hitters problem to a new clustering problem we formulate of finding clusters in a graph, and we then devise a new algorithm CutGrabClose which solves that problem. Specifically, the ExpanderSketch first outputs a graph in which each heavy hitter is encoded by a well-connected cluster (which may be much smaller than the size of the total graph), then CutGrabClose recovers the clusters, i.e. the heavy hitters, from the graph. There have been many works on finding clusters S in a graph such that the conductance of the cut $(S,\bar{S})$ is small. We will only mention some of them with features similar to our setting. Roughly speaking, our goal is to find *all* clusters that are low conductance sets in the graph, and furthermore induce subgraphs that satisfy something weaker than having good spectral expansion. It is necessary for us to (1) be able to identify *all* clusters in the graph; and (2) have a quality guarantee that does not degrade with the number of clusters nor the relative size of the clusters compared with the size of the whole graph. As a bonus, our algorithm is also (3) able to work without knowing the number of clusters.

In the context of heavy hitters, requirement (1) arises since all heavy hitters need to be returned. The number of heavy hitters is not known and can be large, as is the ratio between the size of the graph and the size of a cluster (both can be roughly the square root of the size of our graph), leading to the requirement (2) above. One line of previous works [ST04], [AP09], [GT12] gives excellent algorithms for finding small clusters in a much larger graph with runtime proportional to the size of the clusters, but their approximation guarantees depend on the size of the whole graph, violating our requirement (2). In fact, requirement (2) is related to finding non-expanding small sets in a graph, an issue at the forefront of hardness of approximation [RS10]. As resolving this issue in general is beyond current techniques, many other works including ours attempt to solve the problem in structured special cases. There are many other excellent works for graph clustering, but whose performance guarantees deterioriate as k increases, also violating (2). The work [GT14] shows results in the same spirit as ours: if there are k clusters and a multiplicative $\mathrm{poly}(k)$ gap between the kth and $(k+1)$st smallest eigenvalues of the Laplacian of a graph, then it is possible to partition the graph into at most k low conductance induced subgraphs. Unfortunately, such a guarantee is unacceptable for our application due to more stringent requirements on the clusters being required as k increases, i.e. the $\mathrm{poly}(k)$ gap in eigenvalues — our application provides no such promise. Another work with a goal similar in spirit to ours is [PSZ15], which given the conductances in our graphs deriving from heavy hitters would promise to find k clusters W up to error $\mathrm{poly}(k)\cdot|W|$ symmetric difference each. Unfortunately such a result is also not applicable to our problem, since in our application we could have $k \gg |W|$ so that the guarantee becomes meaningless. The work [OZ14] presents a clustering algorithm based on flow computations. If every cluster S has good internal expansion, and only $O(1/\lg n)$ conductance in the cut $(S, \bar{S})$, then the algorithm finds all the clusters. We could use this as part of our heavy hitters algorithm, but it would cost a $\lg\lg n$ factor in space and time, since we would have to use sub data structures with better error guarantees to obtain conductance $O(1/\lg n)$.

A different line of works [MMV12], [MMV14] give algorithms with constant approximation guarantees but without being able to identify the planted clusters, which is not possible in their models in general. In their setting, edges inside each cluster are adversarially chosen but there is randomness in the edges between clusters. Our setting is the opposite: the edges inside each cluster have nice structure while the edges between clusters are adversarial as long as there are few of them.

From the practical point of view, a drawback with several previous approaches is the required knowledge of the number of clusters. While perhaps not the most pressing issue in theory, it is known to cause problems in practice. For instance, in computer vision, if the number of clusters is not chosen correctly, algorithms like k-means tend to break up uniform regions in image segmentation [AMFM11]. Instead, a preferred approach is hierarchical clustering [AMFM11], which is in the same spirit as our solution.

Definition 1. *An* ϵ-spectral cluster *is a vertex set $W \subseteq V$ of any size satisfying the following two conditions: First, only an ϵ-fraction of the edges incident to W leave W, that is, $|\partial(W)| \le \epsilon\, vol(W)$, where $vol(W)$ is the sum of edge degrees of vertices inside W. Second, given any subset A of W, let $r = vol(A)/\,vol(W)$ and $B = W \setminus A$. Then*

$$|E(A,B)| \ge (r(1-r) - \epsilon)\, vol(W).$$

Note $r(1-r)\, vol(W)$ is the number of edges one would expect to see between A and B had W been a random graph with a prescribed degree distribution.

Our contribution II: We give a hierarchical graph partitioning algorithm combining traditional spectral methods and novel local search moves that can identify *all* ϵ-spectral clusters in a potentially much larger graph for ϵ below some universal constant, independent of the number of clusters or the ratio between the graph size and cluster sizes. More precisely, consider a graph $G = (V,E)$. In polynomial time, our algorithm can partition the vertices in a graph $G = (V,E)$ into subsets $V_1, \dots V_\ell$ so that *every* ϵ-spectral cluster W matches some subset V_i up to an $O(\epsilon\, vol(W))$ symmetric difference. Due to space constraints, the algorithm can be found in the full version of the paper [LNNT16]. A description of the main ideas can be found in Section IV-A. The algorithm has the guarantee given by Theorem 1 below.

Theorem 1. *For any given $\epsilon \le 1/2000000$ and graph $G = (V,E)$, in polynomial time and linear space, we can find a family of disjoint vertex sets $U_1, \dots U_\ell$ so that every ϵ-*spectral cluster *W of G matches some set U_i in the sense that*

- $vol(W \setminus U_i) \le 3\,\epsilon\, vol(W)$.
- $vol(U_i \setminus W) \le 2250000\,\epsilon\, vol(W)$

Moreover, if $v \in U_i$, then $> 4/9$ of its neighbors are in U_i.

An interesting feature of our solution is that in optimization in practice, it is usually expected to perform local search to finish off any optimization algorithm but it is rarely the case that one can prove the benefit of the local search step. In contrast, for our algorithm, it is exactly our local search moves which guarantee that every cluster is found.

III. Preliminaries

The letter C denotes a positive constant that may change from line to line, and $[n] = \{1, \dots, n\}$. We use "with high probability" (whp) to denote probability $1 - 1/\mathrm{poly}(n)$.

All graphs $G = (V, E)$ discussed are simple and undirected. We say that a graph is a λ-*spectral expander* if the second largest eigenvalue, in magnitude, of its unnormalized adjacency matrix is at most λ. Given two vertex sets $S, T \subseteq V$, we define $E(S, T) = E \cap (S \times T)$. For S, we define the *volume* $vol(S) = \sum_{v \in S} \deg(v)$, and *boundary* $\partial S = E(S, \overline{S})$ where $\overline{S} = V \setminus S$. We specify vol_G and ∂_G when the graph may not be clear from context. We define the *conductance* of the cut $(S, \overline{S})$ as

$$\phi(G, S) = \frac{|\partial S|}{\min\{vol(S), vol(\overline{S})\}}.$$

If S or $\overline{S}$ is empty, the conductance is ∞. We define the conductance of the entire graph G by

$$\phi(G) = \min_{\emptyset \subsetneq S \subsetneq V} \phi(G, S).$$

IV. Overview of approach

Due to space constraints, this paper gives only an overview of the ideas in our work. See the full version [LNNT16] for the details.

We now provide an overview of our algorithm, ExpanderSketch. In the full version [LNNT16], we show how turnstile heavy hitters with arbitrary ε reduces to several heavy hitters problems for which ε is "large". Specifically, after the reduction whp we are guaranteed each of the ε-heavy hitters i in the original stream now appears in some substream updating a vector $x' \in \mathbb{R}^n$, and $|x'_i| \ge (1/\sqrt{C \lg n}) \|x'_{\overline{[C \lg n]}}\|_2$. That is, i is an $\Omega(1/\sqrt{\lg n})$-heavy hitter in x'. One then finds all the $\Omega(1/\sqrt{\lg n})$ heavy hitters in each substream then outputs their union as the final query result. For the remainder of this overview we thus focus on solving ε-heavy hitters for $\varepsilon > 1/\sqrt{C \lg n}$, so there are at most $2/\varepsilon^2 = O(\lg n)$ heavy hitters. Our goal is to achieve failure probability $1/\operatorname{poly}(n)$ with space $O(\varepsilon^{-2} \lg n)$, update time $O(\lg n)$, and query time $\varepsilon^{-2} \operatorname{poly}(\lg n) = \operatorname{poly}(\lg n)$.

There is a solution, which we call the "b-tree" (based on [CH08]), which is almost ideal (see the full version [LNNT16]). It achieves $O(\varepsilon^{-2} \lg n)$ space and $O(\lg n)$ update time, but query time inverse polynomial in the failure probability δ, which for us is $\delta = 1/\operatorname{poly}(n)$. Our main idea is to reduce to the case of much larger failure probability (specifically $1/\operatorname{poly}(\lg n)$) so that using the b-tree then becomes viable for fast query. We accomplish this reduction while maintaining $O(\lg n)$ update time and optimal space by reducing our heavy hitter problem into $m = \Theta(\lg n / \lg \lg n)$ separate *partition heavy hitters problems*, which we now describe.

In the partition ε-heavy hitters problem, there is some parameter $\varepsilon \in (0, 1/2)$. There is also some partition $\mathcal{P} = \{S_1, \ldots, S_N\}$ of $[n]$, and it is presented in the form of an oracle $\mathcal{O} : [n] \to [N]$ such that for any $i \in [n]$, $\mathcal{O}(i)$ gives the $j \in [N]$ such that $i \in S_j$. In what follows

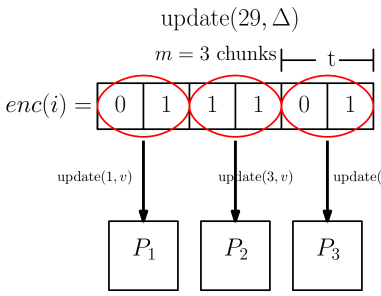

Figure 2. Simplified version of final data structure. The update is $x_{29} \leftarrow x_{29} + \Delta$ with $m = 3$, $t = 2$ in this example. Each P_j is a b-tree operating on a partition of size 2^t.

the partitions will be random, with $\mathcal{O}_j$ depending on some random hash functions (in the case of adapting the b-tree to partition heavy hitters the $\mathcal{O}_j$ are deterministic; see the full version [LNNT16]). Define a vector $y \in \mathbb{R}^N$ such that for each $j \in [N]$, $y_j = \|x_{S_j}\|_2$, where x_S is the projection of x onto a subset S of coordinates. The goal then is to solve the ε-heavy hitters problem on y subject to streaming updates to x: we should output a list $L \subset [N]$, $|L| = O(1/\varepsilon^2)$, containing all the ε-heavy hitters of y. See the full version for more details [LNNT16].

Now we explain how we make use of partition heavy hitters. For each index $i \in [n]$ we can view i as a length $\lg n$ bitstring. Let $\operatorname{enc}(i)$ be an encoding of i into $T = O(\lg n)$ bits by an error-correcting code with constant rate that can correct an $\Omega(1)$-fraction of errors. Such codes exist with linear time encoding and decoding [Spi96]. We partition $\operatorname{enc}(i)$ into m contiguous bitstrings each of length $t = T/m = \Theta(\lg \lg n)$. We let $\operatorname{enc}(i)_j$ for $j \in [m]$ denote the jth bitstring of length t when partitioning $\operatorname{enc}(i)$ in this way. For our data structure, we instantiate m separate partition heavy hitter data structures based on the b-tree solution, $P_1, \ldots, P_m$, each with failure probability $1/\operatorname{poly}(\lg n)$. What remains is to describe the m partitions $\mathcal{P}_j$. We now describe these partitions, where each will have $N = 2^{O(t)} = \operatorname{poly}(\lg n)$ sets.

Now, how we would *like* to define the partition is as follows, which unfortunately does not quite work. For each $j \in [m]$, we let the jth partition $\mathcal{P}_j$ have oracle $\mathcal{O}_j(i) = \operatorname{enc}(i)_j$. That is, we partition the universe according to the jth chunks of their codewords. See Figure 2 for an illustration, where upon an update (i, Δ) we write for each P_j the name of the partition that is being updated. The key insight is that, by definition of the partition heavy hitters problem, any partition containing a heavy hitter $i \in [n]$ is itself a heavy partition in $\mathcal{P}_j$. Then, during a query, we would query each P_j separately to obtain lists $L_j \subseteq [2^t]$. Thus the P_j which succeed produce L_j that contain the jth chunks of encodings of heavy hitters (plus potentially other t-bit chunks). Now, let us perform some wishful thinking:

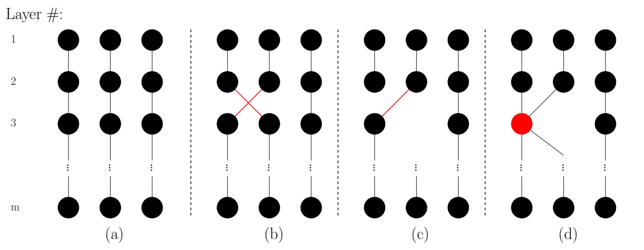

Figure 3. Each vertex in row j corresponds to an element of L_j, i.e. the heavy hitter chunks output by P_j. When indices in $\mathcal{P}_j$ are partitioned by $h_j(i) \circ \mathrm{enc}(i)_j \circ h_{j+1}(i)$, we connect chunks along paths. Case (a) is the ideal case, when all j are good. In (b) P_2 failed, producing a wrong output that triggered incorrect edge insertions. In (c) both P_2 and P_3 failed, triggering an incorrect edge and a missing vertex, respectively. In (d) two heavy hitters collided under h_3, causing their vertices to have the same name thereby giving the appearance of a merged vertex. Alternatively, light items masking as a heavy hitter might have appeared in L_3 with the same h_3 evaluation as a heavy hitter but different h_4 evaluation, causing the red vertex to have two outgoing edges to level 4.

suppose there was only one heavy hitter i^*, and furthermore each L_j contained exactly $\mathrm{enc}(i^*)_j$ and nothing else. Then we could obtain i^* simply by concatenating the elements of each L_j and then decoding. In fact one can show that whp no more than an arbitrarily small fraction (say 1%) of the P_j fail, implying we would still be able to obtain 99% of the chunks of $\mathrm{enc}(i^*)$, which is sufficient to decode. The main complication is that, in general, there may be more than one heavy hitter (there may be up to $\Theta(\lg n)$ of them). Thus, even if we performed wishful thinking and pretended that every P_j succeeded, and furthermore that every L_j contained exactly the jth chunks of the encodings of heavy hitters and nothing else, it is not clear how to perform the concatenation. For example, if there are two heavy hitters with encoded indices 1100 and 0110 with $m = t = 2$, suppose the P_j return $L_1 = \{11, 01\}$ and $L_2 = \{00, 10\}$ (i.e. the first and second chunks of the encodings of the heavy hitters). How would we then know which chunks matched with which for concatenation? That is, are the heavy hitter encodings $1100, 0110$, or are they $1110, 0100$? Brute force trying all possibilities is too slow, since $m = \Theta(\lg n / \lg\lg n)$ and each $|L_j|$ could be as big as $\Theta(\lg n)$, yielding $(C \lg n)^m = \mathrm{poly}(n)$ possibilities. In fact this question is quite related to the problem of *list-recoverable codes* (see the full version [LNNT16]), but since no explicit codes are known to exist with the efficiency guarantees we desire, we proceed in a different direction.

To aid us in knowing which chunks to concatenate with which across the L_j, a first attempt (which also does not quite work) is as follows. Define m pairwise independent hash functions $h_1, \ldots, h_m : [n] \to [\mathrm{poly}(\lg n)]$. Since there are $O(\lg n)$ heavy hitters, any given h_j perfectly hashes them with decent probability. Now rather than partitioning according to $\mathcal{O}_j(i) = \mathrm{enc}(i)_j$, we imagine setting $\mathcal{O}_j(i) = h_j(i) \circ \mathrm{enc}(i)_j \circ h_{j+1}(i)$ where $\circ$ denotes concatenation of bitstrings. Define an index $j \in [m]$ to be *good* if (a) P_j succeeds, (b) h_j perfectly hashes all heavy hitters $i \in [n]$, and (c) for each heavy hitter i, the total ℓ_2 weight from non-heavy hitters hashing to $h_j(i)$ is $o((1/\sqrt{\lg n})\|x_{\overline{[1/\epsilon^2]}}\|_2$. A simple argument shows that whp a $1 - \epsilon$ fraction of the $j \in [m]$ are good, where ϵ can be made an arbitrarily small positive constant. Now let us perform some wishful thinking: if *all* $j \in [m]$ are good, and furthermore no non-heavy elements appear in L_j with the same h_j but different h_{j+1} evaluation as an actual heavy hitter, then the indices in L_j tell us which chunks to concatenate within L_{j+1}, so we can concatenate, decode, then be done. Unfortunately a small constant fraction of the $j \in [m]$ are not good, which prevents this scheme from working (see Figure 3). Indeed, in order to succeed in a query, for each heavy hitter we must correctly identify a large connected component of the vertices corresponding to that heavy hitter's path — that would correspond to containing a large fraction of the chunks, which would allow for decoding. Unfortunately, paths are not robust in having large connected component subgraphs remaining even for $O(1)$ bad levels.

The above consideration motivates our final scheme, which uses an expander-based idea first proposed in [GLPS14] in the context of "for all" ℓ_1/ℓ_1 sparse recovery, a problem in compressed sensing. Although our precise motivation for the next step is slightly different than in [GLPS14], and our query algorithm and our definition of "robustness" for a graph will be completely different, the idea of connecting chunks using expander graphs is very similar to an ingredient in that work. The idea is to replace the path in the last paragraph by a graph which is robust to a small fraction of edge insertions and deletions, still allowing the identification of a large connected component in such scenarios. Expander graphs will allow us to accomplish this. For us, "robust" will mean that over the randomness in our algorithm, whp each corrupted expander is still a spectral cluster (see Definition 1). For [GLPS14], robustness meant that each corrupted expander still contains an induced small-diameter subgraph (in fact an expander) on a small but constant fraction of the vertices, which allowed them a recovery procedure based on a shallow breadth-first search. They then feed the output of this breadth-first search into a recovery algorithm for an existing list-recoverable code (namely Parvaresh-Vardy codes). Due to suboptimality of known list-recoverable codes, such an approach would not allow us to obtain our optimal results.

An expander-based approach: Let F be an arbitrary d-regular connected graph on the vertex set $[m]$ for some $d = O(1)$. For $j \in [m]$, let $\Gamma(j) \subset [m]$ be the set of neighbors of vertex j. We partition $[n]$ according to $\mathcal{O}_j(i) = z(i)_j = h_j(i) \circ \mathrm{enc}(i)_j \circ h_{\Gamma(j)_1} \circ \cdots \circ h_{\Gamma(j)_d}$ where $\Gamma(j)_k$ is the kth neighbor of j in F. Given some such z, we say its *name* is the first $s = O(\lg\lg n)$ bits comprising the h_j portion of the concatenation. Now, we can imagine a graph G on the layered vertex set $V = [m] \times [2^s]$ with m layers. If L_j is the output of a heavy hitter query on P_j, we can view each element z of L_j as suggesting d edges to add to G, where each such z connects d vertices in various layers of V to the vertex in layer j corresponding to the name of z. The way we actually insert the edges is as follows. First, for each $j \in [m]$ we instantiate a partition point query structure Q_j with partition $\mathcal{P}_j$, failure probability $1/\operatorname{poly}(\lg n)$, and error parameter $c\varepsilon$ for a small constant c. We modify the definition of a level $j \in [m]$ being "good" earlier to say that Q_j must also succeed on queries to every $z \in L_j$. We point query every partition $z \in L_j$ to obtain an estimate $\tilde{y}_z$ approximating y_z. We then group all $z \in L_j$ by name, and within each group we remove all z from L_j except for the one with the largest $\tilde{y}_z$, breaking ties arbitrarily. This filtering step guarantees that the vertices in layer j have unique names, and furthermore, when j is good all vertices corresponding to heavy hitters appear in L_j and none of them are thrown out by this filtering. We then let G be the graph created by including the at most $(d/2) \cdot \sum_j |L_j|$ edges suggested by the z's across all L_j (we only include an edge if both endpoints suggest it). Note G will have many isolated vertices since only $m \cdot \max_j |L_j| = O(\lg^2 n/\lg\lg n)$ edges are added, but the number of vertices in each layer is 2^s, which may be a large power of $\lg n$. We let G be its restriction to the union of non-isolated vertices and vertices whose names match the hash value of a heavy hitter at the m different levels. This ensures G has $O(\lg^2 n/\lg\lg n)$ vertices and edges. We call this G the *chunk graph*.

Now, the intuitive picture is that G *should* be the vertex-disjoint union of several copies of the expander F, one for each heavy hitter, plus other junk edges and vertices coming from other non-heavy hitters in the L_j. Due to certain bad levels j however, some expanders might be missing a small constant ϵ-fraction of their edges, and also the ϵm bad levels may cause spurious edges to connect these expanders to the rest of the graph. The key insight is as follows. Let W be the vertices of G corresponding to some particular heavy hitter, so that in the ideal case W would be a single connected component whose induced graph is F. What one can say, even with ϵm bad levels, is that every heavy hitter's such vertices W forms an $O(\epsilon)$-*spectral cluster* as per Definition 1. Roughly this means that (a) the cut separating W from the rest of G has $O(\epsilon)$ conductance (i.e. it is a very sparse cut), and (b) for any cut $(A, W\backslash A)$ within W, the number of edges crossing the cut is what is guaranteed from a spectral expander, minus $O(\epsilon) \cdot vol(W)$. Our task then reduces to finding all ϵ-spectral clusters in a given graph. We show in the full version of the paper [LNNT16] that for each such cluster W, we are able to find a $(1 - O(\epsilon))$-fraction of its volume with at most $O(\epsilon) \cdot vol(W)$ erroneous volume from outside W, see Theorem 1. This suffices for decoding for ϵ a sufficiently small constant, since this means we find most vertices, i.e. chunks of the encoding, of the heavy hitter.

For the special case of ℓ_1 heavy hitters in the strict turnstile model, we are able to devise a much simpler query algorithm that works. For this special case, we also give a space-optimal algorithm with $O(\lg n)$ update time, whp success, and *expected* query time $O(\varepsilon^{-1} \lg n)$ (though unfortunately the variance of the query time may be quite high). See the full version [LNNT16] for both results.

A. Cluster-preserving clustering

An ϵ-spectral cluster is a subset W of the vertices in a graph $G = (V, E)$ such that (1) $|\partial W| \le \epsilon\, vol(W)$, and (2) for any $A \subsetneq W$ with $vol(A)/vol(W) = r$, $|E(A, W\backslash A)| \ge (r(1-r) - \epsilon)\, vol(W)$. Item (2) means the number of edges crossing a cut within W is what you would expect from a random graph, up to $\epsilon\, vol(W)$. Our goal is to, given G, find a partition of V such that every ϵ-spectral cluster W in G matches some partition up to $\epsilon\, vol(W)$ symmetric difference.

Our algorithm CUTGRABCLOSE is somewhat similar to the spectral clustering algorithm of [KVV04, Section 4], but with local search. That algorithm is quite simple: find a low-conductance cut (e.g. a Fiedler cut) to split G into two pieces, then recurse on both pieces. Details aside, Fiedler cuts are guaranteed by Cheeger's inequality to find a cut of conductance $O(\sqrt{\gamma})$ as long as a cut of conductance at most γ exists in the graph. The problem with this basic recursive approach is shown in Figure 4 (in particular cut (b)). Note that a cluster can be completely segmented after a few levels of recursion, so that a large portion of the cluster is never found.

Our approach is as follows. Like the above, we find a low-conductance cut then recurse on both sides. However, before recursing on both sides we make certain "improvements" to the cut. We say $A \subset V$ is *closed* in G if there is no vertex $v \in G\backslash A$ with at least $5/9$ths of its neighbors in A. Our algorithm maintains that all recursive subcalls are to closed subsets in G as follows. Suppose we are calling CUTGRABCLOSE on some set A. We first try to find a low-conductance cut within A. If we do not find one, we terminate and let A be one of the sets in the partition. Otherwise, if we cut A into $(S, \bar{S})$, then we close both $S, \bar{S}$ by finding vertices violating closure and simply moving them. It can be shown that if the $(S, \bar{S})$ cut had sufficiently low conductance, then these local moves can only improve conductance further. Now both S and $\bar{S}$ are closed in A (which by a transitivity lemma we show, implies they are closed in G as well). We then show that if (1) some set S is closed, and (2) S has much more than half the volume of some spectral cluster W (e.g. a $2/3$rds fraction), then in fact S contains a $(1 - O(\epsilon))$-fraction of W. Thus after closing both $S, \bar{S}$, we have that S either: (a) has almost none

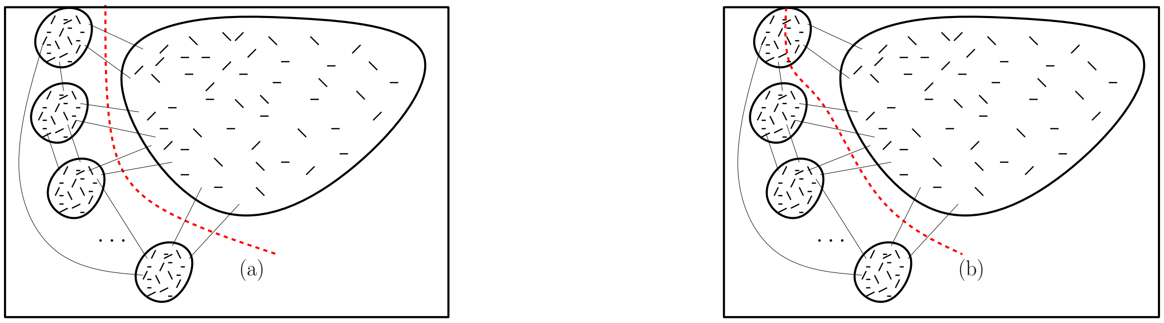

Figure 4. Each small oval is a spectral cluster. They are well-connected internally, with sparse cuts to the outside. The large oval is the rest of the graph, which can look like anything. Cut (a) represents a good low-conductance cut, which makes much progress (cutting the graph in roughly half) while not losing any mass from any cluster. Cut (b) is also a low-conductance cut as long as the number of small ovals is large, since then cutting one cluster in half has negligible effect on the cut's conductance. However, (b) is problematic since recursing on both sides loses half of one cluster forever.

of W, (b) has almost all of W, or (c) has roughly half of W (between $1/3$ and $2/3$, say). To fix the latter case, we then "grab" all vertices in $\bar{S}$ with some $\Omega(1)$-fraction, e.g. $1/6$th, of their neighbors in S and simply move them all to S. Doing this some constant number of times implies S has much more than $2/3$rds of W by expansion (and if S was in case (a), then we show it still has almost none of W). Then by doing another round of closure moves, one can ensure that both $S, \bar{S}$ are closed, and each of them has either an $O(\epsilon)$-fraction of W or a $(1 - O(\epsilon))$-fraction. The details are in the full version [LNNT16]. It is worth noting that our algorithm can make use of *any* spectral cutting algorithm as a black box and not just Fiedler cuts, followed by our grab and closure steps. For example, algorithms from [OV11], [OSV12] run in nearly linear time and either (1) report that no γ-conductance cut exists (in which case we could terminate), (2) find a *balanced* cut of conductance $O(\sqrt{\gamma})$ (where both sides have nearly equal volume), or (3) find an $O(\sqrt{\gamma})$-conductance cut in which every $W \subset G$ with $vol(W) \leq (1/2)\, vol(G)$ and $\phi(W) \leq O(\gamma)$ has more than half its volume on the smaller side of the cut. Item (2), if it always occurred, would give a divide-and-conquer recurrence to yield nearly linear time for finding all clusters. It turns out item (3) though is even better! If the small side of the cut has half of every cluster W, then by grabs and closure moves we could ensure it is still small and has almost all of W, so we could recurse just on the smaller side. It appears an $O(|E| \lg |V|)$-space implementation of such a cutting algorithm achieving this guarantee would lead to $O(\varepsilon^{-2} \lg^{2+o(1)} n)$ query time for whp heavy hitters (with $O(\varepsilon^{-2} \lg^{1+o(1)} n)$ expected query time), but in this version of our work we simply focus on achieving $O(\varepsilon^{-2} \operatorname{poly}(\lg n))$.

Acknowledgments

We thank Noga Alon for pointing us to [AC88, Lemma 2.3], Jonathan Kelner for the reference [OSV12], Lorenzo Orecchia and Sushant Sachdeva for answering several questions about [OV11], [OSV12], Piotr Indyk for the reference [GLPS14], Graham Cormode for the reference [Pag13], Yi Li for answering several questions about [GLPS14], Mary Wootters for making us aware of the formulation of the list-recovery problem in coding theory and its appearance in prior work in compressed sensing and group testing, Atri Rudra for useful discussions on list-recovery algorithms for Parvaresh-Vardy and Folded Reed-Solomon codes, and Fan Chung Graham and Olivia Simpson for useful conversations about graph partitioning algorithms. Part of the work was done while H.N. was at the Toyota Technological Institute at Chicago.

References

[AC88] Noga Alon and Fan R. K. Chung. Explicit construction of linear sized tolerant networks. *Discrete Mathematics*, 72:15–19, 1988.

[AKO11] Alexandr Andoni, Robert Krauthgamer, and Krzysztof Onak. Streaming algorithms via precision sampling. In *Proc. 52nd FOCS*, pages 363–372, 2011.

[AMFM11] Pablo Arbelaez, Michael Maire, Charless Fowlkes, and Jitendra Malik. Contour detection and hierarchical image segmentation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 33(5):898–916, 2011.

[AP09] Reid Andersen and Yuval Peres. Finding sparse cuts locally using evolving sets. In *Proc. 41st STOC*, pages 235–244, 2009.

[ARV09] Sanjeev Arora, Satish Rao, and Umesh Vazirani. Expander flows, geometric embeddings and graph partitioning. *Journal of the ACM (JACM)*, 56(2):5, 2009.

[BC15] Vladimir Braverman and Stephen R. Chestnut. Universal sketches for the frequency negative moments and other decreasing streaming sums. In *RANDOM*, pages 591–605, 2015.

[BCI$^+$16] Vladimir Braverman, Stephen R Chestnut, Nikita Ivkin, Jelani Nelson, David P Woodruff, and Zhengyu Wang. BPTree: an ℓ_2 heavy hitters algorithm using constant memory. *abs/1603.00759*, 2016.

[BCIW16] Vladimir Braverman, Stephen R. Chestnut, Nikita Ivkin, and David P. Woodruff. Beating CountSketch for heavy hitters in insertion streams. In *Proc. 48th STOC, to appear*, 2016.

[BDW16] Arnab Bhattacharyya, Palash Dey, and David P. Woodruff. An optimal algorithm for ℓ_1-heavy hitters in insertion streams and related problems. In *Proceedings of the 35th ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems (PODS)*, 2016.

[BICS10] Radu Berinde, Piotr Indyk, Graham Cormode, and Martin J. Strauss. Space-optimal heavy hitters with strong error bounds. *ACM Trans. Database Syst.*, 35(4):26, 2010.

[BKSV14] Vladimir Braverman, Jonathan Katzman, Charles Seidell, and Gregory Vorsanger. An optimal algorithm for large frequency moments using $O(n^{1-2/k})$ bits. In *RANDOM*, pages 531–544, 2014.

[BO10] Vladimir Braverman and Rafail Ostrovsky. Zero-one frequency laws. In *Proc. 42th STOC*, pages 281–290, 2010.

[BO13] Vladimir Braverman and Rafail Ostrovsky. Approximating large frequency moments with pick-and-drop sampling. In *APPROX*, pages 42–57, 2013.

[BYJKS04] Ziv Bar-Yossef, T. S. Jayram, Ravi Kumar, and D. Sivakumar. An information statistics approach to data stream and communication complexity. *J. Comput. Syst. Sci.*, 68(4):702–732, 2004.

[CCFC04] Moses Charikar, Kevin 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.

[CH08] Graham Cormode and Marios Hadjieleftheriou. Finding frequent items in data streams. *PVLDB*, 1(2):1530–1541, 2008.

[CJK$^+$04] Graham Cormode, Theodore Johnson, Flip Korn, S. Muthukrishnan, Oliver Spatscheck, and Divesh Srivastava. Holistic UDAFs at streaming speeds. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, pages 35–46, 2004.

[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.

[DLM02] Erik D. Demaine, Alejandro López-Ortiz, and J. Ian Munro. Frequency estimation of internet packet streams with limited space. In *Proc. 10th ESA*, pages 348–360, 2002.

[FSG$^+$98] Min Fang, Narayanan Shivakumar, Hector Garcia-Molina, Rajeev Motwani, and Jeffrey D. Ullman. Computing iceberg queries efficiently. In *Proceedings of 24rd International Conference on Very Large Data Bases (VLDB)*, pages 299–310, 1998.

[Gan15] Sumit Ganguly. Taylor polynomial estimator for estimating frequency moments. In *Proc. 42th ICALP*, pages 542–553, 2015.

[GB09] Sumit Ganguly and Lakshminath Bhuvanagiri. Hierarchical sampling from sketches: Estimating functions over data streams. *Algorithmica*, 53(4):549–582, 2009.

[GLPS14] Anna C. Gilbert, Yi Li, Ely Porat, and Martin J. Strauss. For-all sparse recovery in near-optimal time. In *Proc. 41st ICALP*, pages 538–550, 2014.

[GR09] Parikshit Gopalan and Jaikumar Radhakrishnan. Finding duplicates in a data stream. In *Proc. 20th SODA*, pages 402–411, 2009.

[GT12] Shayan Oveis Gharan and Luca Trevisan. Approximating the expansion profile and almost optimal local graph clustering. In *Proc. 53rd FOCS*, pages 187–196. IEEE, 2012.

[GT14] Shayan Oveis Gharan and Luca Trevisan. Partitioning into expanders. In *Proc. 25th SODA*, pages 1256–1266, 2014.

[HNO08] Nicholas J. A. Harvey, Jelani Nelson, and Krzysztof Onak. Sketching and streaming entropy via approximation theory. In *Proc. 49th FOCS*, pages 489–498, 2008.

[IW05] Piotr Indyk and David P. Woodruff. Optimal approximations of the frequency moments of data streams. In *Proc. 37th STOC*, pages 202–208, 2005.

[JST11] Hossein Jowhari, Mert Saglam, and Gábor Tardos. Tight bounds for Lp samplers, finding duplicates in streams, and related problems. In *Proc. 30th PODS*, pages 49–58, 2011.

[JW09] T. S. Jayram and David P. Woodruff. The data stream space complexity of cascaded norms. In *Proc. 50th FOCS*, pages 765–774, 2009.

[KNPW11] Daniel M. Kane, Jelani Nelson, Ely Porat, and David P. Woodruff. Fast moment estimation in data streams in optimal space. In *Proc. 43rd STOC*, pages 745–754, 2011.

[KVV04] Ravi Kannan, Santosh Vempala, and Adrian Vetta. On clusterings: Good, bad and spectral. *J. ACM*, 51(3):497–515, 2004.

[LCG13] Yang Liu, Wenji Chen, and Yong Guan. Identifying high-cardinality hosts from network-wide traffic measurements. In *IEEE Conference on Communications and Network Security, (CNS)*, pages 287–295, 2013.

[LNN15] Kasper Green Larsen, Jelani Nelson, and Huy L. Nguyễn. Time lower bounds for nonadaptive turnstile streaming algorithms. In *Proc. 47th STOC*, pages 803–812, 2015.

[LNNT16] Kasper Green Larsen, Jelani Nelson, Huy L. Nguyễn, and Mikkel Thorup. Heavy hitters via cluster-preserving clustering. 2016. http://arxiv.org/abs/1604.01357.

[LR88] Tom Leighton and Satish Rao. An approximate max-flow min-cut theorem for uniform multicommodity flow problems with applications to approximation algorithms. In *Proc. 29th FOCS*, pages 422–431, 1988.

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

[MMV12] Konstantin Makarychev, Yury Makarychev, and Aravindan Vijayaraghavan. Approximation algorithms for semi-random partitioning problems. In *Proc. 44th STOC*, pages 367–384, 2012.

[MMV14] Konstantin Makarychev, Yury Makarychev, and Aravindan Vijayaraghavan. Constant factor approximation for balanced cut in the PIE model. In *Proc. 46th STOC*, pages 41–49, 2014.

[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 *Proc. 21st SODA*, pages 1143–1160, 2010.

[NNW14] Jelani Nelson, Huy L. Nguyễn, and David P. Woodruff. On deterministic sketching and streaming for sparse recovery and norm estimation. *Lin. Alg. Appl.*, 441:152–167, January 2014. Preliminary version in RANDOM 2012.

[NW10] Jelani Nelson and David P. Woodruff. Fast manhattan sketches in data streams. In *Proc. 29th PODS*, pages 99–110, 2010.

[OSV12] Lorenzo Orecchia, Sushant Sachdeva, and Nisheeth K. Vishnoi. Approximating the exponential, the lanczos method and an $\tilde{O}(m)$-time spectral algorithm for balanced separator. In *Proc. 44th STOC*, pages 1141–1160, 2012.

[OV11] Lorenzo Orecchia and Nisheeth K. Vishnoi. Towards an SDP-based approach to spectral methods: A nearly-linear-time algorithm for graph partitioning and decomposition. In *Proc. 22nd SODA*, pages 532–545, 2011.

[OZ14] Lorenzo Orecchia and Zeyuan Allen Zhu. Flow-based algorithms for local graph clustering. In *Proc. 25th SODA*, pages 1267–1286, 2014.

[Pag13] Rasmus Pagh. Compressed matrix multiplication. *TOCT*, 5(3):9, 2013.

[PDGQ05] Rob Pike, Sean Dorward, Robert Griesemer, and Sean Quinlan. Interpreting the data: Parallel analysis with Sawzall. *Scientific Programming*, 13(4):277–298, 2005.

[PSZ15] Richard Peng, He Sun, and Luca Zanetti. Partitioning well-clustered graphs: Spectral clustering works! In *Proceedings of The 28th Conference on Learning Theory (COLT)*, pages 1423–1455, 2015.

[RS10] Prasad Raghavendra and David Steurer. Graph expansion and the unique games conjecture. In *Proc. 42nd STOC*, pages 755–764, 2010.

[SM00] Jianbo Shi and Jitendra Malik. Normalized cuts and image segmentation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 22(8):888–905, 2000.

[Spi96] Daniel A. Spielman. Linear-time encodable and decodable error-correcting codes. *IEEE Trans. Information Theory*, 42(6):1723–1731, 1996.

[ST04] Daniel A Spielman and Shang-Hua Teng. Nearly-linear time algorithms for graph partitioning, graph sparsification, and solving linear systems. In *Proc. 36th STOC*, pages 81–90, 2004.