Heavy hitters via cluster-preserving clustering

Kasper Green Larsen* Jelani Nelson† Huy L. Nguyễn‡ Mikkel Thorup§

April 5, 2016

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 $|x_i| \ge \varepsilon \|x\|_p$ 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. In fact we solve the stronger tail version, in which L should include every i such that $|x_i| \ge \varepsilon \|x_{\overline{[1/\varepsilon^p]}}\|_p$, where $x_{\overline{[k]}}$ denotes the vector obtained by zeroing out the largest k entries of x in magnitude.

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 $x_i \ge 0$ 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.

*Aarhus University. `larsen@cs.au.dk`. 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.

†Harvard University. `minilek@seas.harvard.edu`. 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.

‡Toyota Technological Institute at Chicago. `hlnguyen@cs.princeton.edu`.

§University of Copenhagen. `mikkel2thorup@gmail.com`. Supported in part by Advanced Grant DFF-0602-02499B from the Danish Council for Independent Research under the Sapere Aude research career programme.

1 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.

2 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 $|x_i| \ge \varepsilon \|x\|_p$, 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 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 $\texttt{update}(i, \Delta)$ have $\Delta = 1$.
- **Strict turnstile:** Each update Δ may be an arbitrary positive or negative number, but we are promised that $x_i \ge 0$ 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 $x_i \ge 0$ 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 $|x_i| \ge \varepsilon \|x\|_1$. 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 $O(\varepsilon^{-2} \lg(1/\varepsilon))$

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 $1/\operatorname{poly}(n)$. 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 $(\varepsilon^{-p} \lg n)((\lg n)/(\varepsilon\delta))^{\gamma}$.

[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 $0 < p \le 2$ 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 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 for a non-promise problem is non-adaptive.

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 $0 < p \leq 2$ 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 (see Section D).

2.1 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 $\operatorname{poly}(k)$ 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 high 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 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 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.

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, $|\partial(W)| \leq \epsilon\, vol(W)$, 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*

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

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 $V_1, \ldots V_\ell$ so that *every* ϵ-spectral cluster W matches some subset V_i up to an $O(\epsilon\, vol(W))$ symmetric difference. The algorithm is described in Section 6, with guarantee given by Theorem 1 below.

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

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

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.

3 Preliminaries

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

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

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

4 Overview of approach

We now provide an overview of our algorithm, ExpanderSketch. In Section A we show 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 $|x'_i| \geq (1/\sqrt{C \lg n})\|x'_{\overline{[C \lg n]}}\|_2$. 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 Section B.2 and Corollary 1). 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 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 Section B.2). 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 Section B for more details.

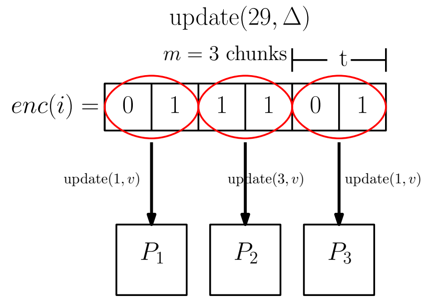

Figure 2: Simplified version of final data structure. The update is $x_{29} \leftarrow x_{29} + \Delta$ with $m = 3$, $t = 2$ in this example. Each P_j is a b-tree operating on a partition of size 2^t.

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 $\mathrm{enc}(i)$ 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 $\mathrm{enc}(i)$ into m contiguous bitstrings each of length $t = T/m = \Theta(\lg \lg n)$. We let $\mathrm{enc}(i)_j$ for $j \in [m]$ denote the jth bitstring of length t when partitioning $\mathrm{enc}(i)$ in this way. For our data structure, we instantiate m separate partition heavy hitter data structures from Section B.2 (a partition heavy hitter variant of the b-tree), $P_1, \ldots, P_m$, each with failure probability $1/\mathrm{poly}(\lg n)$. What remains is to describe the m partitions $\mathcal{P}_j$. We now describe these partitions, where each will have $N = 2^{O(t)} = \mathrm{poly}(\lg n)$ 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 $\mathcal{O}_j(i) = \mathrm{enc}(i)_j$. 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: 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 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 Section C), but since no explicit codes are

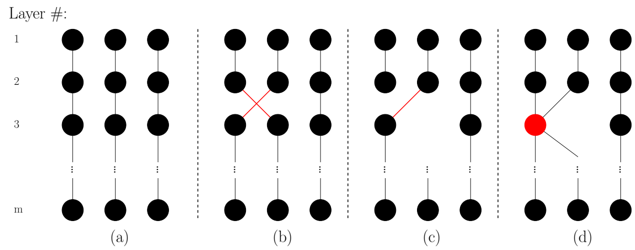

Figure 3: Each vertex in row j corresponds to an element of L_j, i.e. the heavy hitter chunks output by P_j. When indices in $\mathcal{P}_j$ are partitioned by $h_j(i) \circ \mathrm{enc}(i)_j \circ h_{j+1}(i)$, we connect chunks along paths. Case (a) is the ideal case, when all j are good. In (b) P_2 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.

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 $o((1/\sqrt{\lg n})\|x_{\overline{[1/\varepsilon^2]}}\|_2$. 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 as defined in Section 6. 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 as per Lemma 14 with partition $\mathcal{P}_j$, failure probability $1/\mathrm{poly}(\lg n)$, 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 (for the exact form of approximation, see Eq. (4) from Lemma 14). 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

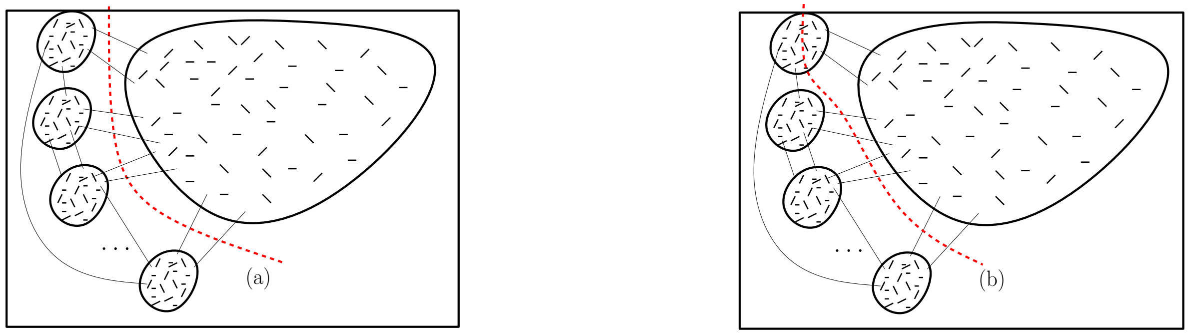

Figure 4: Each small oval is a spectral cluster. They are well-connected internally, with sparse cuts to the outside. The large oval is the rest of the graph, which can look like anything. Cut (a) represents a good low-conductance cut, which makes much progress (cutting the graph in roughly half) while 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.

finding all ϵ-spectral clusters in a given graph. We show in Section 6 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. 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; see Section D for details. For this special case we also give in Section E 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).

4.1 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 5/9ths 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 2/3rds 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 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. 1/6th, 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 2/3rds of W (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 Section 6. 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 $vol(W) \le (1/2)\, vol(G)$ and $\phi(W) \le O(\gamma)$ 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 $O(\varepsilon^{-2} \operatorname{poly}(\lg n))$.

5 General turnstile updates

In this section we analyze our algorithm ExpanderSketch described in Section 4. Recall that our final algorithm, including the reduction in Section A, is as follows. First we pick a hash function $h : [n] \to [q]$ from a $\Theta(\lg n)$-wise independent hash family for $q = \lceil 1/(\varepsilon^2 \lg n) \rceil$. Then we initialize q data structures $D^1, \ldots, D^q$, where each D^k is an ε'-heavy hitters data structure as described in Section 4, for $\varepsilon' = \max\{\varepsilon, 1/\sqrt{C \lg n}\}$. We also during intialization construct a d-regular λ_0-spectral expander F on $m = \Theta(\lg n / \lg\lg n)$ vertices for some $d = O(1)$, where $\lambda_0 = \epsilon d$ for some (small) constant $\epsilon > 0$ to be specified later. Such an F can be constructed in time $\operatorname{poly}(\lg n)$ with $d = \operatorname{poly}(1/\epsilon)$ deterministically [RVW02], then stored in adjacency list representation consuming $O(\lg n / \lg\lg n)$ words of memory.

The D^k are independent, and for each D^k we pick hash functions $h^k_1, \ldots, h^k_m : [n] \to [\operatorname{poly}(\lg n)]$ independently from a pairwise independent family and instantiate partition ε'-heavy hitter data structures $P^k_1, \ldots, P^k_m$ (as b-trees; see Section B.2) with $\mathcal{O}^k_j : [n] \to [2^{O(t)}]$ for $j \in [m]$ defined by

$\mathcal{O}_j^k(i) = h_j^k(i) \circ \mathrm{enc}(i)_j \circ h^k_{\Gamma(j)_1} \circ \cdots \circ h^k_{\Gamma(j)_d}$. Each P_j^k has failure probability $1/\operatorname{poly}(\lg n)$. We also instantiate partition $c\varepsilon$-point query data structures $Q_1^k, \ldots, Q_m^k$ with small constant c and failure probability $1/\operatorname{poly}(\lg n)$ as per Lemma 14, with the same $\mathcal{O}_j^k$. Here enc is an encoding as in Section 4 mapping into $T = O(\lg n)$ bits, and $t = T/m = \Theta(\lg\lg n)$. $\Gamma(j)_k$ is the kth neighbor of j in F. To answer a query to our overall data structure, we query each D^k separately then output the union of their results. To answer a query to D^k, we form a chunk graph G^k from the outputs of the P_j^k as in Section 4. We then find all ϵ_0-spectral clusters using Theorem 1 for a sufficiently small constant ϵ_0, throw away all clusters of size less than $m/2$, then from each remaining cluster W' we: (1) remove all vertices of degree $\leq d/2$, (2) remove $v \in W'$ coming from the same layer $j \in [m]$ as some other $v' \in W'$, then (3) form a (partially corrupted) codeword using the bits associated with vertices left in W'. We then decode to obtain a set B^k containing all heavy hitter indices $i \in h^{-1}(k)$ with high probability. We then output $L = \cup_{k=1}^q B^k$ as our final result.

Henceforth we condition on the event $\mathcal{E}$ that every heavy hitter $i \in [n]$ is ε'-heavy in $D^{h(i)}$, which happens whp by Theorem 5. Our final analysis makes use of the following.

Lemma 1. *Suppose $\mathcal{E}$ occurs. Focus on a particular $k \in [q]$. Let $x' = x_{h^{-1}(k)}$ be the projection of x onto vertices hashing to k. As in Section 4, we say an index $j \in [m]$ is* good *if (a) P_j^k succeeded, (b) h_j^k perfectly hashes all ε'-heavy hitters in x', (c) for each ε'-heavy hitter i, the total ℓ_2 weight from non-heavy hitters in $h^{-1}(k)$ hashing to $h_j(i)$ is under h_j is $o((1/\sqrt{\lg n}))\|x_{\overline{[1/\varepsilon'^2]}}\|_2$, and (d) Q_j^k succeeded on every $z \in L_j^k$. Call j* bad *otherwise. Then if the failure probability of each P_j^k, Q_j^k is $1/\lg^{C+1} n$, the range of each h_j^k is $[\lg^{C+3} n]$, and $m = C\lg n/\lg\lg n$ for some constant $C > 0$, then with probability $1 - 1/n^c$ the number of bad levels is at most βm, where c can be made an arbitrarily large constant and β an arbitrarily small constant by increasing C.*

Proof. The probability of (b), that h_j^k perfectly hashes all $O(\lg n)$ ε'-heavy hitters, is $1 - O(1/\lg^{C+1} n)$. For (c), focus on a particular heavy hitter i. The expected squared ℓ_2 mass from non-ε' heavy hitters colliding i under h_j is $(1/\lg^{C+3} n)\|x_{\overline{[1/\varepsilon'^2]}}\|_2^2$ by pairwise independence of h_j. Thus by Markov, the probability that more than $(1/\lg^2 n)\|x_{\overline{[1/\varepsilon'^2]}}\|_2^2$ squared ℓ_2 mass collided with i from light items is at most $1/\lg^{C+1} n$. Since there are only $O(\lg n)$ ε'-heavy hitters, by a union bound we have that (c) holds, i.e. that no heavy hitter collides with more than $(1/\lg n)\|x_{\overline{[1/\varepsilon'^2]}}\|_2 = o(1/\sqrt{\lg n})\|x_{\overline{[1/\varepsilon'^2]}}\|_2$ ℓ_2 mass from light items under ℓ_j, with probability $1 - O(1/\lg^C n)$. For (d), we perform $O(1/\varepsilon'^2) = O(\lg n)$ point queries on Q_j^k, and each point query has failure probability $1/\lg^{C+1} n$, so the probability that any of the point query fails is at most $1/\lg^C n$. Thus j is bad with probability $O(1/\lg^C n)$. Now also choose $m = C\lg n/\lg\lg n$. Then the probability that there are more than βm bad levels is at most, for some constant C' independent of C,

$$\binom{m}{\beta m}\left(\frac{C'}{\lg^C n}\right)^{\beta m} \leq \left(\frac{C'e}{\beta \lg^C n}\right)^{\beta C\lg n/\lg\lg n} = \left(\frac{C'e}{\beta}\right)^{\beta C\lg n/\lg\lg n} \cdot \frac{1}{n^{\beta C^2}} = \frac{1}{n^{\beta C^2 - o_n(1)}},$$

using $\binom{a}{b} \leq (ea/b)^b$. Thus the lemma holds for $\beta = 1/C$, and $c = C/2$ for sufficiently large n. ■

Our correctness analysis also makes use of the following lemma [AC88, Lemma 2.3] to show that, with high probability, each heavy hitter is represented by an ϵ_0-spectral cluster. The lemma states that for spectral expanders, sets expand with small sets S satisfying even better expansion properties. Some version of this lemma is known in the literature as the expander mixing lemma.

Lemma 2. *[AC88] Let A be the adjacency matrix of a d-regular graph with vertex set V. Suppose the second largest eigenvalue of A in magnitude is $\lambda > 0$. Then for any $S \subseteq V$, writing $|S| = r|V|$, $|\partial S| \geq (d - \lambda)(1 - r)|S|$.*

Theorem 2. *Assume n is larger than some constant. Let $\mathcal{A}$ be the algorithm from Theorem 1 which finds all ϵ_0-spectral clusters in a graph on $O(\varepsilon'^{-2} \lg n / \lg\lg n)$ vertices and edges whp in time $\mathcal{T}$ and space $\mathcal{S}$. For any $0 < \varepsilon < 1/2$, there is an algorithm solving the ε-heavy hitters problem in general turnstile streams on vectors of length n whp with space and update time $O(\varepsilon^{-2} \lg n)$ and $O(\lg n)$, respectively. Answering a query uses an additional $O(\mathcal{S}) = o(\varepsilon^{-2} \lg n)$ space and runs in time $O(\varepsilon^{-2} \lg^{1+\gamma} n + q\mathcal{T}) = O(\varepsilon^{-2} \operatorname{poly}(\lg n))$, by q successive calls to $\mathcal{A}$. Here $\gamma > 0$ can be chosen as an arbitrarily small constant.*

Proof. We first analyze space and running times. We store F, taking $O(\lg n/\lg\lg n)$ space, which can be ignored as it is dominated by other parts of the algorithm. Storing h requires $O(\lg n)$ words of memory. Let us now focus on a specific D^k. It stores m hash functions drawn independently from a pairwise family mapping $[n]$ to $[\operatorname{poly}(\lg n)]$, consuming $O(m) = O(\lg n/\lg\lg n)$ space. The m b-trees combined P_j^k consume space $m \cdot O(\varepsilon'^{-2} t) = O(\varepsilon'^{-2} \lg n)$, and the same for the Q_j^k. Thus the total space per D^k is $O(\varepsilon'^{-2} \lg n)$, and thus the total space across the entire algorithm is $O(q\varepsilon'^{-2} \lg n) = O(\varepsilon^{-2} \lg n)$. We also need an additional $O(\mathcal{S})$ space to run $\mathcal{A}$ repeatedly during a query. For the update time, we first hash using h, taking $O(\lg n)$ time. Then within a D^k we perform m hash evaluations h_j^k, taking total time $O(m)$. We then compute the encoding $\operatorname{enc}(i)$ of i, taking $O(\lg n)$ time, then perform m b-tree updates taking total time $m \cdot O(t) = O(\lg n)$. Thus the total update time is $O(\lg n)$. For the query time, for a given D^k we have to query m b-trees, taking time $m \cdot O(\varepsilon'^{-2} t \lg^\gamma n) = O(\varepsilon'^{-2} \lg^{1+\gamma} n)$. We also run $\mathcal{A}$ on G^k, taking time $\mathcal{T}$. Thus the total query time is $O(q(\varepsilon'^{-2} \lg^{1+\gamma} n + \mathcal{T})) = O(\varepsilon^{-2} \lg^{1+\gamma} n + q\mathcal{T})$.

Now it only remains to argue correctness. Each L^k has size $O(1/\varepsilon'^2)$ with probability 1, which is a guarantee of the b-tree. Then when we form the chunk graph G^k, it has $O(\varepsilon'^{-2} \lg n/\lg\lg n)$ vertices. We only insert a decoded spectral cluster into L if its size is at least $m/2$, so there can be at most $O(1/\varepsilon'^2)$ such clusters found. Thus $|L| \leq O(q/\varepsilon'^2) = O(1/\varepsilon^2)$ with probability 1.

It now only remains to show that each heavy hitter $i \in [n]$ is represented by some spectral cluster of size at least $m/2$ in $G^{h(i)}$. We assume event $\mathcal{E}$, which happens whp. Now let us focus on a particular G^k. Define $x' = x_{h^{-1}(k)}$. We will show that each ε'-heavy hitter in x' is represented by such a cluster in G^k. Focus on a particular ε'-heavy hitter i. Let W be the set of m vertices $\{(j, h_j^k(i))\}_{j=1}^m$. Suppose that all levels $j \in [m]$ were good. We claim that in this case, W would be an isolated connected component in G^k, and furthermore the induced graph on W would be F. To see this, observe that before filtering by Q_j^k, $z^k(i)_j$ appears in L_j^k (see Section 4 for the definitions of L_j and $z(i)_j$; L_j^k, $z^k(i)_j$ are then the natural modifications corresponding to $k \in [q]$). Once we group the $z \in L_j^k$ by name and filter according to Q_j^k, let $z \neq z^k(i)_j$ be such that its name also equals $h_j^k(i)$. Since level j is good, none of the mass of z is from a heavy hitter. Thus by condition (c) in the definition of goodness, the ℓ_2 weight y_z of partition z is at most $o(1/\sqrt{\lg n})\|x_{\overline{[1/\varepsilon'^2]}}\|_2$. Thus by the condition of Eq. (4), since Q_j^k succeeds we will will have $\tilde{y}_{z^k(i)_j} > \tilde{y}_z$ and not remove $z^k(i)_j$ from L_j^k. Thus $z^k(i)_j$ remains in L_j^k even after filtering. Since this is true across all $j \in [m]$, we will add all edges of F to W. Furthermore no vertices outside W will connect to W, since edge insertion requires mutual suggestion of the edge by both endpoints. Thus, in the case of all good levels, by Lemma 2 for any subset A of W with $|A| = r|W| = rm$,

$$|E(A, W\backslash A)| \geq (d - \lambda_0)r(1-r)|W| = (r(1-r) - r(1-r)\frac{\lambda_0}{d})d|W| \geq (r(1-r) - \frac{\lambda_0}{4d})dm.$$

By Lemma 1, the number of bad levels is at most βm with high probability, for β an arbitrarily small constant. Let us now understand the possible effects of a bad level on W. Suppose j is bad. Then the L_j^k obtained after filtering by Q_j^k may not include $z^k(i)_j$. Thus W may lose at most d edges from F (those corresponding to edges incident upon vertex j in F). Second, L_j^k after filtering may instead contain some z whose name is $h_j^k(i)$, but whose edge suggestions differ from $z^k(i)_j$, possibly causing edges to be inserted within W which do not agree with F, or even inserting edges that cross the cut $(W, G\backslash W)$. At most d such edges are inserted in this way. Thus, across all bad levels, the total number of F-edges deleted within W is at most βdm, and the total number of edges crossing the cut $(W, G\backslash W)$ is also at most βdm. Also, the volume $vol(W)$ is always at most dm and at least $(1-\beta)dm$. Thus after considering bad levels, for any subset A of W as above,

$$|E(A, W\backslash A)| \geq (r(1-r) - \frac{\lambda_0}{4d} - \beta)dm \geq (r(1-r) - \epsilon_0)\, vol(W).$$

for $\epsilon_0 \geq \beta + \lambda_0/(4d)$. Furthermore, the number of edges leaving W to $G\backslash W$ is

$$|\partial(W)| \leq \beta dm \leq \frac{\beta}{1-\beta}\, vol(W) \leq \epsilon_0\, vol(W)$$

for $\epsilon_0 \geq \beta/(1-\beta)$. Thus W is an ϵ_0-spectral cluster in G^k representing i for $\epsilon_0 = \max\{\beta + \lambda_0/(4d), \beta/(1-\beta)\} = \max\{\beta + \epsilon/4, \beta/(1-\beta)\}$. Thus by Theorem 1, $\mathcal{A}$ recovers a W' missing at most $3\epsilon_0\, vol(W)$ volume from W, and containing at most $2250000\epsilon_0$ volume from $G\backslash W$. Note we remove any vertex from W' of degree $\leq d/2$, so W' contains at most $5500000\epsilon_0 m$ vertices from outside W. Furthermore, since there are at most βdm edges lost from W due to bad levels, at most $2\beta m$ vertices in W had their degrees reduced to $\leq d/2$, and thus removing low-degree vertices removed at most $2\beta m$ additional vertices from W. Also, since the max degree in G^k is d and at most a 2β fraction of vertices in W have $\leq d/2$ degree, since W' is missing at most $3\epsilon_0\, vol(W)$ vertices from W, this corresponds to at most $2\beta m + 6\epsilon_0 m$ vertices missing from W. We then form a (corrupted) codeword $C' \in \{0,1\}^{mt}$ by concatenating encoding chunks specified by the vertices in W'. Since then at most a $(5500006\epsilon_0 + 2\beta)$-fraction of entries in $\mathrm{enc}(i)$ and C' differ, for ϵ, β sufficiently small constants we successfully decode and obtain the binary encoding of i, which is included in L. ■

Remark 1. It is possible to slightly modify our algorithm so that the $\varepsilon^{-2}\lg^{1+\gamma} n$ summand in the query time becomes $\varepsilon^{-2}\lg^{1+o(1)} n$. We state the modification here without providing the full details. Specifically, rather than implement the P_j^k as b-trees, one could instead implement them as recursive versions of the ExpanderSketch on a smaller universe. Then, after a single level of recursion down, one switches back to using b-trees. Then the b-trees at the second level of recursion have query time which includes an arbitrarily small power of $\lg\lg n$, which is $\lg^{o(1)} n$. One may be tempted to avoid b-trees altogether and simply continue the recursion to the leaves, but due to the blowup in bitlength of the partition sizes stemming from concatenating h_j values, it seems such an approach would result in the space complexity being multiplied by an additional $2^{O(\lg^* n)}$ factor.

6 Cluster-preserving clustering

In this section we present our cluster-preserving clustering, partitioning the graph into clusters with the promise to preserve and identify all ϵ-spectral clusters as per Definition 1.

The main goal of this section is to prove Theorem 1. To prove Theorem 1, we shall use cuts based on Cheeger's inequality. Given an undirected graph $G = (V, E)$, let its Laplacian matrix be $\mathcal{L}_G$ (normalized so that $(\mathcal{L}_G)_{i,i} = 1$ for all i). It is known that that $\mathcal{L}_G$ is positive semidefinite with eigenvalues $0 = \lambda_1 \le \lambda_2 \le \ldots \le \lambda_{|V|}$ with $\lambda_k = 0$ iff G has at least k connected components. A fundamental result in spectral graph theory is the following, which is a robust form of this fact when $k = 2$.

Theorem 3 (Cheeger's inequality for graphs [AM85, Alo86, SJ89])**.** *For $G = (V, E)$ an undirected graph, let λ_2 be the second smallest eigenvalue of $\mathcal{L}_G$. Then a cut $(S, \overline{S})$ satisfying*

$$\frac{\lambda_2}{2} \le \phi(G) \le \phi(G, S) \le \sqrt{2\lambda_2} \le 2\sqrt{\phi(G)} \tag{1}$$

can be found in time $\mathrm{poly}(|V|)$ *and space* $O(|V| + |E|)$.

Algorithm 1 Top level of cluster preserving clustering of G_0 as stated in Theorem 1.

---

function $\textsc{Main}(G_0)$
    $\{U_1 \ldots, U_\ell\} \leftarrow \textsc{CutGrabClose}(G_0)$
    For each U_i, recursively remove $v \in U_i$ with $\ge 5/9$ths neigbors outside U_i. $\triangleright$ Cleaning
    return $\{U_1 \ldots, U_\ell\}$
end function

Algorithm 2 Cluster preserving clustering of $G = G_0|V$, approximately isolating clusters W of G_0.

---

function $\textsc{CutGrabClose}(G = G_0|V)$
    Use Theorem 3 to find a cut $(S, \overline{S})$ satisfying Eq. (1); say $|S| \le |\overline{S}|$.
    if $\phi(G, S) \ge 1/500$ **then return** $\{V\}$ $\triangleright$ We've identified a potential cluster.
    else
        $\textsc{LocalImprovements}(G, S, V)$ $\triangleright$ Local improvements of both $S, \overline{S}$.
        $\textsc{Grab}(G, S)$
        $\textsc{LocalImprovements}(G, S, \overline{S})$ $\triangleright$ Local improvements of S.
        $\textsc{Grab}(G, S)$
        $\textsc{LocalImprovements}(G, S, \overline{S})$ $\triangleright$ Local improvements of S.
        $\textsc{LocalImprovements}(G, \overline{S}, S)$ $\triangleright$ Local improvements of $\overline{S}$.
        $\mathcal{C} \leftarrow \textsc{CutGrabClose}(G|S) \cup \textsc{CutGrabClose}(G|\overline{S})$
        return $\mathcal{C}$
    end if
end function

We are now ready to describe our algorithm, presented in pseudo-code in Algorithms 2–4.

At the top level, from $\textsc{Main}$, the function $\textsc{CutGrabClose}$ is fed as input the graph $G = (V, E)$ from Theorem 1. We denote this top level graph $G_0 = (V_0, E_0)$. $\textsc{CutGrabClose}$ is a recursive

Algorithm 3 Local improvements across cut $(S, \overline{S})$ only moving vertices from T.

function LocalImprovements(G, S, T) ▷ $(S, \overline{S})$ is a cut in $G = (V, E)$
  while $\exists v \in T$ with at least 5/9ths of its edges crossing the cut **do** ▷ Local improvement.
    Move v to the other side of the cut.
  end while
end function

Algorithm 4 Expanding S by grabbing all vertices with 1/6th neighbors in S.

function Grab(G, S)
  Let T be set of vertices from $G \setminus V$ with at least 1/6th of its neighbors in S.
  $S \leftarrow S \cup T$.
end function

algorithm, working on induced subgraphs $G = G_0|V$ of G_0, producing a family of disjoint subsets of the vertices. Suppose G_0 has an ϵ-spectral cluster W. Our goal is to show that one set in the final partition produced by CutGrabClose matches W in a slightly weaker sense than Theorem 1 (namely $vol_{U_i}(U_i \setminus W) = O(\epsilon\, vol(W))$ instead of the second matching condition). At the end of the section, we describe how the last cleaning step in Main removes extraneous mass from candidate clusters and achieves the stronger conditions in Theorem 1.

On input $G = (V, E)$, CutGrabClose works as follows: if there is no low-conductance cut, it returns just the single-element partition $\{V\}$. Otherwise, it finds a low-conductance cut $(S, \overline{S})$. It then performs a sequence of recursive *local improvements* and two *grabs* to maintain certain invariants in our analysis. Namely, we want to maintain the invariant that throughout all levels of recursion, most of W stays in the same recursive branch. Here, by most of W, we mean relative to the volume of W in the original G_0. This will ensure that the final output $\mathcal{C}$ of the topmost level of recursion contains a single set S matching W.

We now proceed with a formal description. Since we will be talking both about the original graph G_0 and the recursive induced subgraphs $G = G_0|V$, we will sometimes use a subscript to indicate which graph we are working in, e.g., if $A \subseteq V$, then $vol_G(A)$ and $vol_{G_0}(A)$ are the volumes of A in G and G_0 respectively. Since the subgraphs are all induced, the edge sets are always determined from the vertices included.

Closure We say set A of vertices from G is *closed* in G if there is no vertex $v \in G \backslash A$ with at least 5/9 of its neighbors in A. The closure property is transitive, which is very useful for our recursion:

Lemma 3. *If $A \subseteq V \subseteq V(G_0)$, A is closed in $G_0|V$, and V is closed in G_0, then A is closed in G_0.*

Proof. Suppose for a contradiction that we have a vertex $v \in G_0 \backslash A$ with at least 5/9ths of its neighbors in A. Then v also has 5/9ths of its neighbors in V, but V is closed in G_0, so v should be in V. However, in $G_0|V$, the vertex v can only have lost neighbors from outside A, so in $G_0|V$, v also has at least 5/9ths of its neighbors in A. This contradicts that A is closed in $G_0|V$. ■

We say that a vertex set $A \subseteq V_0$ *dominates* the ϵ-spectral cluster W if $vol_{G_0}(W \cap V) > (1 - 3\,\epsilon)\, vol_{G_0}(W)$. We will show that if A is closed and has more than 2/3 of the volume of W, then it dominates W:

Lemma 4. *If A is closed in G_0 and $vol_{G_0}(W \cap A) \geq \frac{2}{3}\, vol_{G_0}(W)$, then A dominates W.*

Proof. By assumption $r = vol_{G_0}(W \setminus A)/\, vol_{G_0}(W) \leq 1/3$. By the spectral expansion of W, we have

$$E(W \setminus A, W \cap A) \geq (r(1-r) - \epsilon) \underset{G_0}{vol}(W) \geq (1 - r - \epsilon/r) \underset{G_0}{vol}(W \setminus A).$$

The average degree from $W \setminus A$ to A is thus at least $(1 - r - \epsilon/r)$, which is bigger than $5/9$ if $3\epsilon \leq r \leq 1/3$ and $\epsilon \leq 1/27$ (we have $\epsilon < 1/2000000$). This would contradict that A is closed, so we conclude that $r < 3\epsilon$, and hence that $vol_{G_0}(W \cap A)/\, vol_{G_0}(W) = 1 - r > 1 - 3\epsilon$. ■

We shall frequently use the fact that if V dominates W, then within $V \cap W$, there is not much difference between volume in G and G_0. More precisely,

Lemma 5. *Consider an arbitrary set $S \subset V$. If V dominates W then $vol_{G_0}(W \cap V) \leq vol_G(W \cap S) + 4\,\epsilon\, vol_{G_0}(W)$.*

Proof. The edges from $W \cap S$ in G_0 that are not in G, are either edges leaving W, of which, by isolation of W, there are at most $\epsilon\, vol_{G_0}(W)$, or edges to $W \setminus V$, of which there at at most $vol_{G_0}(W \setminus V) = vol_{G_0}(W) - vol_{G_0}(V \cap W) \leq 3\epsilon\, vol_{G_0}(W)$. ■

We now continue describing our algorithm to isolate W (plus any other ϵ-spectral cluster). As our invariant, we are given some induced subgraph $G = (V, E)$ of G_0 such that

(1) V is closed in G_0.

(2) V dominates W.

To maintain this invariant, we want to show that CutGrabClose finds a cut $(S, \overline{S})$ where both sides are closed and one side dominates W. We also want to make sure that both sides are non-empty. By definition, this follows if the conductance is below 1, and we will, in fact, always keep the conductance below 1/9.

The first step of CutGrabClose is to use Theorem 3 to find a cut $(S, \overline{S})$ of G satisfying Eq. (1); say $|S| \leq |\overline{S}|$ (line 2 of Algorithm 2). If $\phi(G, S) \geq 1/500$, we will just return $\{V\}$. The following lemma states that V does not have too much volume in G outside W.

Lemma 6. *If $\phi(G, S) \geq 1/500$, then $vol_G(V \setminus W) \leq 1000000\,\epsilon\, vol_{G_0}(W)$.*

Proof. By isolation of the ϵ-cluster W in G_0, we have $|\partial_{G_0}(W)| \leq \epsilon\, vol_{G_0}(W)$. However, $\partial_G(V \setminus W) = \partial_G(V \cap W) \subseteq \partial_{G_0}(W)$, so $|\partial_G(V \setminus W)| \leq \epsilon\, vol_{G_0}(W)$.

Suppose $vol_G(V \setminus W) > 1000000\,\epsilon\, vol_{G_0}(W)$. We want to conclude that $\phi(G, V \setminus W) < 1/1000000$. To do that, we also need to argue that $vol_G(V \cap W) > 1000000\,\epsilon\, vol_{G_0}(W)$. Since V dominates W, we have $vol_{G_0}(V \cap W) > (1 - 3\,\epsilon)\, vol_{G_0}(W)$ and by Lemma 5, we have

$$\underset{G}{vol}(V \cap W) \geq \underset{G_0}{vol}(V \cap W) - 4\,\epsilon \underset{G_0}{vol}(W) \leq (1 - 7\,\epsilon) \underset{G_0}{vol}(W) > 1000000\,\epsilon \underset{G_0}{vol}(W).$$

The last equality follows because $\epsilon \leq 1/2000000$. We have now proved that $vol_G(V \setminus W) > 1000000\,\epsilon\, vol_{G_0}(W)$ implies that $\phi(G) \leq \phi(G, V \setminus W) < 1/1000000$. By Eq. (1) this contradicts that we did not find a cut of size less than $2\sqrt{\phi(G)} < 1/500$. ■

The last cleaning in Main will reduce the volume of $V \setminus W$ in G_0 as required for Theorem 1. We shall return to that later. Below we assume that we found a low conductance cut $(S, \overline{S})$ of G with $\phi(G, S) \leq 1/500$. We are going to move vertices between S and $\overline{S}$, and will always maintain a conductance below 1/9.

Local improvements towards closure We are now going to move vertices between S and $\overline{S}$ to make sure that W is mostly contained in one side. As a first step, we will make sure that both sides are closed (line 5 of Algorithm 2). This is done iteratively. If one vertex has at least 5/9ths fraction of its neighbors on the other side of the cut, we move it to the other side, calling it a *local improvement* (c.f. Algorithm 3). When no more local improvements are possible, both sides S and $\overline{S}$ must be closed in G. We call these moves improving because they always improve both cut size and conductance, as described more formally below.

Lemma 7. *Consider a local improvement moving a vertex v. It reduces the cut size by at least $d_G(v)/9$. It also improves the conductance if it was below $1/9$ before the move.*

Proof. When we move v, we replace at least $\frac{5}{9}d_G(v)$ cut edges by at most $\frac{4}{9}d_G(v)$ cut edges, reducing the cut size by at least $d_G(v)/9$. The volume moved is $d_G(v)$, so if $\phi(G,S) < 1/9$ and the new cut is $(S', \overline{S}')$, we get

$$\phi(G,S') = \frac{|\partial_G(S')|}{\min\{vol_G(S'), vol_G(\overline{S}')\}} \le \frac{|\partial_G(S)| - d_G(v)/9}{\min\{vol_G(S), vol_G(\overline{S})\} - d_G(v)} < \phi(G,S).$$

■

When both sides are closed under local improvements, we have a situation where either W is almost completely dominated by one side, or it roughly balanced between the two sides. More precisely,

Lemma 8. *When both S and $\overline{S}$ are closed in G (and hence in G_0), then either $vol_{G_0}(S \cap W) < 3\epsilon\, vol_{G_0}(W)$, or $(1/3 - 3\epsilon)\, vol_{G_0}(W) < vol_{G_0}(S \cap W) < (2/3)\, vol_{G_0}(W)$, or $vol_{G_0}(S \cap W) > (1 - 3\epsilon)\, vol_{G_0}(W)$.*

Proof. This follows almost directly from Lemma 4. We just note that since $vol_{G_0}(S \cap W) + vol_{G_0}(\overline{S} \cap W) = vol_{G_0}(V \cap W) \ge (1 - 3\epsilon)\, vol_{G_0}(W)$, if $vol_{G_0}(S) \le (1/3 - 3\epsilon)\, vol_{G_0}(W)$, then $vol_{G_0}(\overline{S} \cap W) \ge 2/3\, vol_{G_0}(W)$, and then, by Lemma 4, we have $vol_{G_0}(\overline{S} \cap W) > (1 - 3\epsilon)\, vol_{G_0}(W)$, and hence $vol_{G_0}(S \cap W) < 3\epsilon\, vol_{G_0}(W)$. ■

Grabbing for dominance Having made sure that both sides are closed, as described in Lemma 8, we now have W either almost completely dominated by one side, or roughly balanced between both sides. We will now introduce a *grab* operation (c.f. Algorithm 4) that in the balanced case will help S get dominance. The grab operation itself is simple. In one round, it moves every vertex to S that before the grab has more than 1/6th of its neighbors in S. An important point here is that contrary to the local improvements, the grabbing is not recursive. We are actually going to do this grabbing twice, interspersed with local improvements (c.f., Algorithm 2): First we grab from S, then we do local improvements of S, only moving vertices to S, thus only closing S. We do this grabbing followed by local improvements from S twice, calling it the *big expansion* of S (Algorithm 2 lines 6–9). Finally we do local improvements of $\overline{S}$. We will prove that when all this is completed, then one of the two sides dominate W, but first we have to argue that the conductance stays low.

Lemma 9. *With the above grabs and local improvements, the conductance always stays below* $1/9$.

Proof. Our starting point is a Cheeger cut $(S, \overline{S})$ with conductance $\phi(G,S) \le 1/500$. From Lemma 7 will never increase the conductance if it is below 1/9. Thus, it is only the grabs that can increase the conductance. Let S' be the result of the grab from S. We know that all the vertices grabbed from $\overline{S}$ have at least 1/6th of their neighbors in S. This means that $|\partial_G(S')| \leq 5|\partial_G(S)|$ and $vol_G(S') \leq vol_G(S) + 6\partial_G(S)$. It follows that

$$\begin{aligned}\phi(G, S') \leq & \frac{|\partial_G(S')|}{\min\{vol_G(S'), vol_G(\overline{S'})\}} \leq \frac{5|\partial_G(S)|}{\min\{vol_G(S), vol_G(\overline{S})\} - 6|\partial_G(S)|} \\ & \leq \frac{5|\partial_G(S)|}{(1 - 6\phi(G, S))\min\{vol_G(S), vol_G(\overline{S})\}} = \frac{5}{(1 - 6\phi(G, S))}\phi(G, S).\end{aligned}$$

Before the first grab, we have conductance below 1/500, which the grab now increases to at most $6/((1-6/500)500) < 1/80$. The second grab can then increase the conductance to at most $6/((1-6/80)80) < 1/12$, so, in fact, the conductance will always stay below 1/12. ■

Below we first argue that if we started with balance, then after the big expansion of S, we have $vol_{G_0}(S \cap W) \geq (1 - 3\,\epsilon)\, vol_{G_0}(W)$.

Lemma 10. *If S is closed and $vol_{G_0}(W \setminus S) \leq (2/3)\, vol_{G_0}(W)$, then S dominates W after the big expansion from S.*

Proof. Let $r = vol_{G_0}(W \setminus S)/\, vol_{G_0}(W) \leq 2/3$. If we can show that the big expansion of S brings $r \leq 1/3 - 3\,\epsilon$, then we are done by Lemma 4 since S is closed and $vol_{G_0}(S \cap W) \geq (1 - r - 3\,\epsilon)\, vol_{G_0}(W \setminus S)$.

We study the effect of a single grab from a closed set S. By spectral expansion, we have $|E(W \setminus S, S \cap W)| \geq (r(1-r) - \epsilon)\, vol_{G_0}(W)$.

Let T be the set of vertices from W moved by the grab, that is, T is the set of vertices from $W \setminus S$ with at least 1/6th of their neigbors in S. Also, let B be the set of vertices from $W \setminus S$ that are not moved, that is, $B = W \setminus (S \cup T)$. Then

$$|E(S \cap W, B)| \leq |E(S, B)| < \underset{G}{vol}(B)/6 \leq \underset{G_0}{vol}(W \setminus S)/6 = r\, \underset{G_0}{vol}(W)/6.$$

The number of edges from $S \cap W$ to vertices from $W \setminus S$ that will be grabbed is therefore at least

$$|E(S \cap W, T)| = |E(S \cap W, W \setminus S)| - E(S \cap W, B) \geq (r(1-r) - \epsilon - r/6)\, \underset{G_0}{vol}(W) = (5r/6 - r^2 - \epsilon)\, \underset{G_0}{vol}(W).$$

The function $f(r) = 5r/6 - r^2$ is convex, having its minimum over any interval in its end points. We are only concerned with $r \in [(1/3 - 3\,\epsilon), 2/3]$, where the smallest value is $f(2/3) = 1/9$. The number of edges to grabbed vertices is thus $|E(S \cap W, T)| \geq (1/9 - \,\epsilon)\, vol_{G_0}(W)$.

We also know that before a grab, the set S is closed. Therefore, every vertex v outside S has less than $\frac{5}{9}d_G(v)$ edges going to S, and no more going to $S \cap W$. We conclude that

$$\underset{G_0}{vol}(T) \geq \underset{G}{vol}(T) \geq \frac{9}{5}|E(S \cap W, T)| \geq \frac{9}{5}(1/9 - \,\epsilon)\, \underset{G_0}{vol}(W) \geq (1/5 - 2\,\epsilon)\, \underset{G_0}{vol}(W).$$

Each grab thus decreases $r = vol_{G_0}(W \setminus S)/\, vol_{G_0}(W)$ by $(1/5 - 2\,\epsilon)$. Starting from $r \leq 2/3$, with two grabs followed by local improvements of S, we thus end up with $r \leq 2/3 - 2(1/5 - 2\,\epsilon) \leq 4/15 + 4\,\epsilon$. This is less than the desired $1/3 - 3\,\epsilon$ if $\epsilon < 1/105$, which is indeed the case. ■

Next, we will show that $\overline{S}$ dominated W before the big expansion of S, then $\overline{S}$ will also dominate W at the end.

Lemma 11. *Suppose that S and $\overline{S}$ are closed and that $\overline{S}$ dominate W. Then $\overline{S}$ will also dominate W after a big expansion from S followed by local improvements of $\overline{S}$.*

Proof. We want to show that after the big expansion of S, we still have $vol_{G_0}(W\cap\overline{S}) \geq \frac{2}{3}\, vol_{G_0}(W)$. This can only increase when we subsequently do local improvements of $\overline{S}$, closing $\overline{S}$, and then the result follows from Lemma 4.

Stepping back to S before the big expansion, we are trying to bound how much volume from $vol_G(W \setminus S)$ that can be moved to S. Both for local improvements and for grabbing, the chance of moving v to S increases the more neighbors v has in S. Thus, we maximize the potential of moving vertices from W to S if we assume that we start with all of $V \setminus W$ in S. The only cut edges are now those between S and $W \setminus S$.

When a local improvement moves a vertex v to S, we replace at least $\frac{5}{9}d_G(v)$ cut edges with at most $\frac{4}{9}d_G(v)$ cut edges while increasing $vol_G(W\cap S)$ by $d_G(v)$. It follows that if local improvements increase $vol_G(S)$ by x, then they decrease $|\partial_G(S)|$ by at least $x/9$.

When we grab a set T of vertices to S, we know that each vertex $v \in T$ had at least $\frac{1}{6}d_G(v)$ of its edges to S. When the grab is done, setting $S' = S \cup T$, we have at most $\frac{5}{6}d_G(v)$ edges from v to $W \setminus S'$. We conclude that $|\partial_G(S')| \leq 5|\partial_G(S)|$ and that $vol_G(S') \leq vol_G(S) + 6|\partial_G(S)|$. Subsequent local improvements can further increase the volume by at most $9|\partial_G(S')| \leq 45|\partial_G(S)|$. Thus, when we from S do one grab followed by local improvements, we increase the volume $vol_G(S)$ by at most $51|\partial_G(S)|$.

We now note that if we did local improvements of S before the grab, then we would only get a smaller bound. More precisely, if the the local improvements moved volume x to $vol_G(S)$, then $|\partial_G(S)|$ would be reduced by $x/9$ before the grab, and then the total volume increase would be at most $x + 51(|\partial_G(S)| - x/9) \leq 51|\partial_G(S)|$.

We now study the big expansion from S, starting with a grab followed by any number of local improvements followed by another grab followed any number of local improvements. From the above analysis, it follows that the total increase in $vol_G(S)$ is at most $6|\partial_G(S)| + 51 \cdot 5|\partial_G(S)| \leq 261|\partial_G(S)|$. This bounds the total volume moved from $vol_G(W \setminus S)$ to $vol_G(S)$.

We started with $vol_G(W \cap S) \leq 3\,\epsilon\, vol_{G_0}(W)$. Also, originally, before the big expansion, the set $\overline{S}$ was closed, so vertices from S had less than a fraction $5/9$ of their neighbors in $\overline{S}$. This fraction could only be reduced when we artificially added all of $V \setminus W$ to S. After this artificial change, which only improved movement of vertices from $W \setminus S$ to S, we had $|\partial S| \leq \frac{5}{9}\, vol_G(W \cap S) + |\partial_G(W)| \leq (5/3 + 1)\epsilon\, vol_{G_0}(W)$. Thus, after the big expansion, we end up with

$$\underset{G}{vol}(W \cap S) \leq (3 + 261(5/3 + 1)\epsilon \underset{G_0}{vol}(W) < 699\ \epsilon \underset{G_0}{vol}(W).$$

Finally, by Lemma 5, we have $vol_{G_0}(W \cap S) \leq vol_G(W \cap S) + 4\,\epsilon\, vol_{G_0}(W) \leq 703\ \epsilon\, vol_{G_0}(W)$, which with $\epsilon \leq 1/1000000$, is much less than the required $(1/3 - 3\,\epsilon)\, vol_{G_0}(W)$. ■

Finally we need

Lemma 12. *Suppose that S is closed and dominates W. Then both of these properties are preserved if we do local improvements of $\overline{S}$.*

Proof. It sufficies to consider a local improvement moving a single vertex v to $\overline{S}$.

First we show that S being closed is preserved when we do a local improvement from $\overline{S}$ moving a vertex v to $\overline{S}$. Suppose for contradiction that we afterwards had some vertex $u \in \overline{S}$ with 5/9ths of its neighbors in S. Then we cannot have $u = v$; for we moved v because it had 5/9ths of its neighbors in $\overline{S}$. However, any other $u \neq v$ can only have fewer neighbors in S after the move.

Since S starts dominating, we start with $vol_{G_0}(W \setminus S) \leq 3\,\epsilon\, vol_{G_0}(W)$. The dominance can only be affected if $v \in W$, and we only move v to $\overline{S}$ because it has at least 5/9ths of its neighbors in $\overline{S}$. Trivially v can have at most $vol_{G_0}(W \cap \overline{S}) + |\partial_{G_0}(W)| \leq 4\,\epsilon\, vol_{G_0}(W)$ neighbors in $\overline{S}$, so we conclude that $d_G(v) \leq \frac{9}{5} \cdot 4\,\epsilon\, vol_{G_0}(W) < 8\,\epsilon\, vol_{G_0}(W)$, and by Lemma 5, we have $d_{G_0}(v) \leq d_G(v) + 4\,\epsilon\, vol_{G_0}(W) < 12\,\epsilon\, vol_{G_0}(W)$. Thus we end up with $vol_{G_0}(W \setminus S) < 15\,\epsilon\, vol_{G_0}(W)$. This is much less that $(1/3 - 3\,\epsilon)\, vol_{G_0}(W)$. Since S remains closed, we conclude from Lemma 4 that S still dominates W, hence that $vol_{G_0}(W \setminus S) \leq 3\,\epsilon\, vol_{G_0}(W)$. ■

Let us now sum up what we have proved. Suppose we found an initial cut $(S, \overline{S})$ of conductance below 1/500. This is when we start modifying the cut, and we need to show that our invariants are preserved for the recursive calls.

By Lemma 12, the local improvements of S followed by the final local improvements of $\overline{S}$ imply that both sets end up closed, so invariant (1) is satisfied. The hard part was ensure that one side dominates W. We had Lemma 9 showing that we keep the conductance below 1/9. First we did local improvements of both sides. By Lemma 8, we end up either with one side dominating W, or with W roughly balanced between the sides. If W is balanced or dominated by S, then after by big expansion from S, by Lemma 10, S dominates W. By Lemma 12, this dominance is preserved when we do the final local improvements of $\overline{S}$. If instead $\overline{S}$ dominated W, then, by Lemma 11 this is dominance is preserved. Thus we always end up with one side dominating W, so invariant (2) is satisfied. We conclude that W is always dominated by one branch of the recursion.

The recursion finishes when no small cut is found. We have V dominating S, and by Lemma 6, we have $vol_G(V \setminus W) \leq 1000000\,\epsilon\, vol_{G_0}(W)$. Thus we have proved

Theorem 4. *For any given $\epsilon \leq 1/2000000$ and graph G_0, in polynomial time,* CutGrabClose *(Algorithm 2) finds a partitioning $\{U_1, \ldots U_\ell\}$ of the vertex set so that every ϵ-*spectral cluster *W of G_0 matches some set U_i in the sense that*

- *$vol(W \setminus U_i) \leq 3\,\epsilon\, vol(W)$.*
- *$vol_{G_0|U_i}(U_i \setminus W) \leq 1000000\,\epsilon\, vol(W)$*

The only detail missing in proving Theorem 1 is that we want a bound on $vol_{G_0}(U_i \setminus W)$ rather than $vol_{G_0|U_i}(U_i \setminus W)$. This is where the last cleaning of Main (Algorithm 1) comes in. It removes all vertices from U_i with more than 5/9ths of their neighbors outside U_i, so now, for all $v \in U_i$, we have $d_{G_0}(v) \leq \frac{9}{4} d_{G_0|U_i}(v)$.

Lemma 13. *The cleaning of U_i preserves that U_i dominates W, and now $vol_{G_0}(U_i \setminus W) \leq \frac{9}{4}\, vol_{G_0|U_i}(U_i \setminus W) \leq 225000000\,\epsilon\, vol(W)$.*

Proof. Recall that U_i before the cleaning is closed and dominates W. The cleaning has exactly the same effect as if we did local improvements of $V(G_0) \setminus U_i$, so by Lemma 12, we preserve both that U_i is closed, and that it dominates W. ■

Theorem 4 and Lemma 13 immediately imply the statement of Theorem 1.

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.

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 *52nd Annual Symposium on Foundations of Computer Science (FOCS)*, pages 363–372, 2011.

[Alo86] Noga Alon. Eigenvalues and expanders. *Combinatorica*, 6:83–96, 1986.

[AM85] Noga Alon and Vitali Milman. Isoperimetric inequalities for graphs, and superconcentrators. *Journal of Combinatorial Theory, Series B*, 38(1):73–88, 1985.

[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 *Proceedings of the 41st annual ACM Symposium on Theory of Computing (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.

[ASS08] Noga Alon, Oded Schwartz, and Asaf Shapira. An elementary construction of constant-degree expanders. *Combinatorics, Probability and Computing*, 17(3):319–327, May 2008.

[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 *Proceedings of the 48th Annual ACM Symposium on Theory of Computing (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 *Proceedings of the 42nd ACM Symposium on Theory of Computing (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.

[Che09] Mahdi Cheraghchi. Noise-resilient group testing: Limitations and constructions. In *17th International Symposium on Fundamentals of Computation Theory (FCT)*, pages 62–73, 2009.

[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.

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

[DKN10] Ilias Diakonikolas, Daniel M. Kane, and Jelani Nelson. Bounded independence fools degree-2 threshold functions. In *Proceedings of the 51th Annual IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 11–20, 2010.

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

[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 *42nd International Colloquium on Automata, Languages, and Programming (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 *41st International Colloquium on Automata, Languages, and Programming (ICALP)*, pages 538–550, 2014.

[GNP+13] Anna C. Gilbert, Hung Q. Ngo, Ely Porat, Atri Rudra, and Martin J. Strauss. ℓ_2/ℓ_2-foreach sparse recovery with low risk. In *Proceedings of the 40th International Colloquium on Automata, Languages, and Programming (ICALP)*, pages 461–472, 2013.

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

[GT12] Shayan Oveis Gharan and Luca Trevisan. Approximating the expansion profile and almost optimal local graph clustering. In *Foundations of Computer Science (FOCS), 2012 IEEE 53rd Annual Symposium on*, pages 187–196. IEEE, 2012.

[GT14] Shayan Oveis Gharan and Luca Trevisan. Partitioning into expanders. In *Proceedings of the 25th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 1256–1266, 2014.

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

[HW15] Brett Hemenway and Mary Wootters. Linear-time list recovery of high-rate expander codes. In *42nd International Colloquium on Automata, Languages, and Programming (ICALP)*, pages 701–712, 2015.

[INR10] Piotr Indyk, Hung Q. Ngo, and Atri Rudra. Efficiently decodable non-adaptive group testing. In *Proceedings of the Twenty-First Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 1126–1142, 2010.

[IW05] Piotr Indyk and David P. Woodruff. Optimal approximations of the frequency moments of data streams. In *Proceedings of the 37th Annual ACM Symposium on Theory of Computing (STOC)*, pages 202–208, 2005.

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

[JW09] T. S. Jayram and David P. Woodruff. The data stream space complexity of cascaded norms. In *50th Annual IEEE Symposium on Foundations of Computer Science (FOCS)*, 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 *Proceedings of the 43rd ACM Symposium on Theory of Computing (STOC)*, pages 745–754, 2011.

[KNW10] Daniel M. Kane, Jelani Nelson, and David P. Woodruff. On the exact space complexity of sketching and streaming small norms. In *Proceedings of the Twenty-First Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 1161–1178, 2010.

[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 *Proceedings of the Forty-Seventh Annual ACM on Symposium on Theory of Computing (STOC)*, pages 803–812, 2015.

[LR88] Tom Leighton and Satish Rao. An approximate max-flow min-cut theorem for uniform multicommodity flow problems with applications to approximation algorithms. In *Proceedings of the 29th Annual IEEE Symposium on Foundations of Computer Science (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 *Proceedings of the 44th Annual ACM Symposium on Theory of Computing Conference (STOC)*, pages 367–384, 2012.

[MMV14] Konstantin Makarychev, Yury Makarychev, and Aravindan Vijayaraghavan. Constant factor approximation for balanced cut in the PIE model. In *Proceedings of the 46th Annual ACM Symposium on Theory of Computing (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 *Proceedings of the Twenty-First Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 1143–1160, 2010.

[NNW14] Jelani Nelson, Huy L. Nguy~ên, 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.

[NPR12] Hung Q. Ngo, Ely Porat, and Atri Rudra. Efficiently decodable compressed sensing by list-recoverable codes and recursion. In *29th International Symposium on Theoretical Aspects of Computer Science (STACS)*, pages 230–241, 2012.

[NW10] Jelani Nelson and David P. Woodruff. Fast manhattan sketches in data streams. In *Proceedings of the Twenty-Ninth ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems (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 *Proceedings of the 44th Symposium on Theory of Computing Conference, (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 *Proceedings of the Twenty-Second Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 532–545, 2011.

[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 *Proceedings of the 42nd ACM Symposium on Theory of Computing (STOC)*, pages 755–764, 2010.

[RVW02] Omer Reingold, Salil Vadhan, and Avi Wigderson. Entropy waves, the zig-zag graph product, and new constant-degree expanders. *Annals of Mathematics*, 155(1):157–187, 2002.

[SJ89] Alistair J. Sinclair and Mark R. Jerrum. Approximative counting, uniform generation and rapidly mixing Markov chains. *Information and Computation*, 82(1):93–133, 1989.

[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 *Proceedings of the 36th Annual ACM Symposium on Theory of Computing (STOC)*, pages 81–90, 2004.

Appendix

A Heavy hitters reduction from small to large ϵ

The main theorem of this section is the following.

Theorem 5. *Let $0 < \epsilon, \delta < 1/2$ be such that $1/\epsilon^2 > \lg(1/\delta)$. Then there is a reduction from turnstile ℓ_2 heavy hitters with parameter ϵ and failure probability δ to $z = \lceil 1/(\epsilon^2 \lg(1/\delta)) \rceil$ separate $(1/\sqrt{t})$-heavy hitters problems for $t = C \lg(1/\delta)$ for some constant $C > 0$, and where each such problem must be solved with failure probability at most $\delta' = \delta/(3z)$. Specifically, given an algorithm for the $(1/\sqrt{t})$-heavy hitters problem with these parameters using space S, update time t_u and query time t_q, the resulting algorithm for ϵ-heavy hitters from this reduction uses space $O(z \cdot S + \lg(1/\delta))$, and has update time $t_u + O(\lg(1/\delta))$ and query time $O(z \cdot t_q)$.*

Proof. We pick a hash function $h : [n] \to [z]$ at random from a $\Theta(\lg(1/\delta))$-wise independent family as in [CW79]. Such h requires $O(\lg(1/\delta))$ space to store and can be evaluated in $O(\lg(1/\delta))$ time. We also instantiate z independent $(1/\sqrt{t})$-heavy hitter data structures $P_1, \ldots, P_z$ each with failure probability δ'. Upon receiving an update (i, v) in the stream, we feed the update (i, v) to $P_{h(i)}$. To answer a query, we first query each P_j to obtain a list L_j, then we output $L = \cup_{j=1}^{z} L_j$.

This concludes the description of the data structures stored, and the implementations of update and query. The space bound and running times thus follow.

We now argue correctness. Let H be the set of ϵ-heavy hitters of x, so that $|H| \le 2/\epsilon^2$. It follows by the Chernoff bound and union bound over all $j \in [z]$ that, for some constant $C > 0$,

$$\mathop{\mathbb{P}}_{h}(\exists j \in [z],\ |H \cap h^{-1}(j)| > C \lg(1/\delta)) < \delta/3. \tag{2}$$

We next invoke Bernstein's inequality, which says there exists constant $c > 0$ such that for independent $X_1, \ldots, X_r$ each bounded by K in magnitude with $\sigma^2 = \sum_i \mathbb{E}(X_i - \mathbb{E} X_i)^2$,

$$\forall \lambda > 0,\ \mathbb{P}(|\sum_i X_i - \mathbb{E} \sum_i X_i| > \lambda) \lesssim e^{-c\lambda^2/\sigma^2} + e^{-c\lambda/K}.$$

We will consider the collection of random variables X_i indexed by $i \in [n] \backslash H$ defined as follows (so $r = n - |H|$). Fix $j \in [z]$. Define $X_i = \mathbf{1}_{h(i)=j} \cdot x_i^2$. It then follows that $K < \epsilon^2 \cdot \|x_{\overline{[1/\epsilon^2]}}\|_2^2$ and

$$\sigma^2 = \sum_{i \notin H} x_i^4 (1/z - 1/z^2) \le \frac{\epsilon^2}{z} \cdot \|x_{\overline{[1/\epsilon^2]}}\|_2^2 \cdot \sum_{i \notin H} x_i^2 = \frac{\epsilon^2}{z} \cdot \|x_{\overline{[1/\epsilon^2]}}\|_2^4$$

It then follows by Bernstein's inequality and a union bound over all $j \in [z]$ that

$$\mathop{\mathbb{P}}_{h}(\exists j \in [z],\ \sum_{\substack{i \notin H \\ h(i)=j}} x_i^2 > C\epsilon^2 \lg(1/\delta) \|x_{\overline{[1/\epsilon^2]}}\|_2^2) < \delta/3. \tag{3}$$

Now, the hash function h divides the input stream into z separate substreams, with $x(j)$ denoting the vector updated by the substream containing i with $h(i) = j$ (i.e. $(x(j))_i = \mathbf{1}_{h(i)=j} \cdot x_i$). Conditioned on the events of (2), (3) occurring, we have that any $i \in H$ satisfies $x_i^2 \ge \epsilon^2 \|x_{\overline{[1/\epsilon^2]}}\|_2^2 \ge 1/(C \lg(1/\delta)) \cdot \|x(h(i))_{\overline{[C \lg(1/\delta)]}}\|_2^2$. Thus if we also condition on every P_j succeeding, which happens with probability at least $1 - z\delta' = 1 - \delta/3$ by a union bound, then our output L is correct. Thus our overall failure probability is at most δ. ■

B Partition heavy hitters and the b-tree

We first define a generalization of the heavy hitters problem that we call *partition heavy hitters*. The standard heavy hitters problem is simply the special case when $\mathcal{P} = \{\{1\}, \ldots, \{n\}\}$, and the oracle $\mathcal{O}$ is the identity map.

Definition 2. *In the ℓ_2* partition heavy hitters problem *there is some error parameter $\epsilon \in (0, 1/2)$. 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 index $j \in [N]$ such that $i \in S_j$. Define a vector $y \in \mathbb{R}^N$ such that for each $j \in [N]$,*

$$y_j = \sqrt{\sum_{i \in S_j} x_i^2}.$$

The goal then is to solve the ℓ_2 ϵ-heavy hitters problem on y subject to streaming updates to x. That is, x is being updated in a stream, and we should at the end output a list $L \subset [N]$, $|L| = O(1/\epsilon^2)$, such that L contains all the ϵ-heavy hitters of y.

A related problem is the ℓ_2 partition point query problem*, in which queries takes as input some $j \in [N]$ and must output a value $\tilde{y}_j$ such that $\alpha_1 y_j - \epsilon \|y_{\overline{[1/\epsilon^2]}}\|_2 \le \tilde{y}_j \le \alpha_2 y_j + \epsilon \|y_{\overline{[1/\epsilon^2]}}\|_2$ for some approximation parameters $\alpha_1, \alpha_2, \epsilon$. We call such a solution an $(\alpha_1, \alpha_2, \epsilon)$-partition point query algorithm.*

We now describe the b-tree, introduced in [CH08] (and also called the Hierarchical CountSketch there and in other parts of the literature) and analyze its performance for solving the partition heavy hitters problem. The b-tree was only suggested in [CH08] for the strict turnstile model, and a detailed analysis was not given. Here we give a fully detailed analysis showing that, in fact, that structure gives good guarantees even in the general turnstile model, and even for the partition heavy hitters problem. Before delving into the b-tree, we will analyze the performance of the PartitionCountSketch, a very slight modification of the CountSketch, for the partition point query problem.

B.1 PartitionCountSketch

In the idealized PartitionCountSketch one chooses independent hash functions $h_1, \ldots, h_L : [N] \to [k]$ and $\sigma_1, \ldots, \sigma_L : [N] \to \{-1, 1\}$ from pairwise independent families, and $\beta_1, \ldots, \beta_L : [n] \to \mathbb{R}$ independently from a C_α-wise independent family for some small constant $\alpha > 0$. The marginal distribution of $\beta_t(i)$ for any $i \in [n]$ and $t \in [L]$ is the standard gaussian $\mathcal{N}(0, 1)$, and C_α is chosen so that for any $z \in \mathbb{R}^n$, $\sum_i z_i \beta_t(i)$ has Kolmogorov distance at most α from $\mathcal{N}(0, 1)$ (see Theorem 6). One also initializes counters $C_{a,b}$ to 0 for all $(a, b) \in [L] \times [k]$, each consuming one machine word. The space is then clearly $O(kL)$ machine words.

The update algorithm for $x_i \leftarrow x_i + \Delta$ performs the change $C_{t,h_t(\mathcal{O}(i))} \leftarrow C_{t,h_t(\mathcal{O}(i))} + \Delta \cdot \sigma_t(\mathcal{O}(i)) \cdot \beta_t(i)$ for each $t \in [L]$. To answer an ℓ_2 point query to index $i \in [N]$, one outputs $\tilde{y}_i$ being the median value of $|C_{t,h_t(i)}|$ across t. In the actual, non-idealized version of PartitionCountSketch, the marginal distribution of each $\beta_t(i)$ is a discretized gaussian with precision fitting in a single machine word; see [KNW10, Section A.6] for details. We henceforth just discuss the idealized version. In fact, for the types of instances of partition heavy hitters that our heavy hitters algorithm needs to solve, the β_t can be replaced with pairwise independent hash functions mapping to $\{-1, 1\}$ (see

Remark 2), which would make an implementation less complicated. We instead have chosen to analyze the version with gaussians since doing so then does not require us to limit the types of partition heavy hitter instances we can handle, thus providing a more general guarantee that may prove useful as a subroutine in future works.

Before analyzing PartitionCountSketch, we state a useful theorem from [KNW10].

Theorem 6. *[KNW10] Let $z \in \mathbb{R}^n$ and $0 < \alpha < 1$ be arbitrary. There exists a constant C_α depending on α such that if $\beta_1, \ldots, \beta_n$ are C_α-wise independent standard gaussians and g is standard gaussian, then*

$$\forall A < B \in \mathbb{R},\ |\mathbb{P}(\sum_i \beta_i z_i \in [A, B]) - \mathbb{P}(g \in [A, B])| < \alpha.$$

In fact C_α can be taken as some value in $O(1/\alpha^2)$ by combining the FT-mollification construction in [DKN10] with Lemma 2.2 and the argument in the proof of Theorem 2.1 in [KNW10]. Using the FT-mollification construction in [KNW10] would imply the weaker statement that $C_\alpha = \Omega(\lg^6(1/\alpha)/\alpha^2)$ suffices. In our current setting either is acceptable since α is some fixed constant.

The following lemma shows the correctness of PartitionCountSketch, following an argument similar to that of [CCFC04, Lemma 3].

Lemma 14. *For $L \gtrsim \lg(1/\delta)$ and $k \gtrsim 1/\epsilon^2$,*

$$\forall j \in [N],\ \mathbb{P}(\tilde{y}_j \notin [(1/20)y_j - \epsilon\|y_{\overline{[1/\epsilon^2]}}\|_2, 3y_j + \epsilon\|y_{\overline{[1/\epsilon^2]}}\|_2]) < \delta. \tag{4}$$

That is, the PartitionCountSketch *with these parameters is an $(\alpha_1, \alpha_2, \epsilon)$-partition point query structure with failure probability δ, for $\alpha_1 = 1/20, \alpha_2 = 3$.*

Proof. For $t \in [L]$, define $w_t \in \mathbb{R}^N$ with $(w_t)_j = \sum_{i \in S_j} \beta_t(i) x_i$. Then by Theorem 6, $\mathbb{P}((w_t)_j \in [-\alpha_1, \alpha_2]) \leq \mathbb{P}(y_j g \in [-\alpha_1, \alpha_1]) + \alpha_1$ for g a standard gaussian. Since $\mathbb{P}(|g| \leq \epsilon) \leq \epsilon\sqrt{2/\pi}$ for any $\epsilon > 0$,

$$\mathbb{P}(|(w_t)_j| > \alpha_1 y_j) \geq 1 - \alpha_1(1 + \sqrt{2/\pi}) > 8/9. \tag{5}$$

Also by pairwise independence of β_t, $\mathbb{E}(w_t)_j^2 = y_j^2$, so

$$\mathbb{P}(|(w_t)_j| > \alpha_2 y_j) < 1/\alpha_2^2 = 1/9. \tag{6}$$

Next, let $H \subset [N]$ be the set of indices of top $1/(4\epsilon^2)$ entries of y in magnitude. Note

$$\mathop{\mathbb{E}}_{h_t, \beta_t} |\sum_{\substack{h_t(\mathcal{O}(i)) = h_t(j) \\ \mathcal{O}(i) \notin H \cup \{j\}}} \beta_t(i) x_i| \leq \frac{1}{\sqrt{k}} \cdot \|y_{\overline{[1/\epsilon^2]}}\|_2$$

by pairwise independence of β_t, h_t. Thus

$$\mathop{\mathbb{P}}_{h_t, \beta_t}(|\sum_{\substack{h_t(\mathcal{O}(i)) = h_t(j) \\ \mathcal{O}(i) \notin H \cup \{j\}}} \beta_t(i) x_i| > \epsilon\|y_{\overline{[1/\epsilon^2]}}\|_2) < 1/9 \tag{7}$$

by Markov's inequality. Also $\mathbb{E}_{h_t} |h_t^{-1}(h_t(j)) \cap (H \backslash \{j\})| \leq |H|/k < 1/9$, so

$$\mathbb{P}(\exists j' \in (H \backslash \{j\}) : h_t(j') = h_t(j)) < 1/9 \tag{8}$$

by Markov's inequality. Thus by a union bound, the conditions of (5), (6), (7), and (8) all happen simultaneously with probability at least 5/9. Now note when all these events occur,

$$|C_{t,h_t(j)}| \in [\alpha_1 y_j - \epsilon\|y_{\overline{[1/\epsilon^2]}}\|_2, \alpha_2 y_j + \epsilon\|y_{\overline{[1/\epsilon^2]}}\|_2] \tag{9}$$

Thus by a chernoff bound, the probability that the median over $t \in [L]$ of $|C_{t,h_t(j)}|$ does not satisfy (9) is $\exp(-\Omega(l)) < \delta$. ■

Remark 2. It is possible to obtain the main result of this paper, i.e. our bounds for (non-partition) heavy hitters, without ever making use of Theorem 6. This might be useful for pedagogical reasons, since the proof of Theorem 6 is somewhat complicated. We sketch how to do so here. We chose to present the approach above since it divides our final algorithm into a more abstracted set of subproblems, which we felt made it easier to describe. Furthermore, the "partition heavy hitters" problem as formulated above seems quite clean and may find uses as a subroutine elsewhere.

The main idea to avoid Theorem 6 is as follows. First, in our application of partition heavy hitters to solve the vanilla heavy hitters problem (see the proof of Theorem 2), we are only interested in the case where each partition S_j in $\mathcal{P}$ contains at most one ϵ-heavy hitter $i \in [n]$ with respect to x. Furthermore, the other items in the same partition will, with good probability over the choice of the hash function h_j used in the ExpanderSketch, have much smaller total ℓ_2 mass than $|x_i|$. These observations can be used to show that, for applying partition heavy hitters to heavy hitters as in Theorem 2, the β_t above can be replaced with pairwise independent hash functions mapping to $\{-1, 1\}$. This ensures that the "noise", i.e. the non-heavy indices in the same partition as i, with say 8/9 probability do not subtract much away from $|x_i|$ so that $|(w_t)_j|$ will be $\Omega(|x_i|)$. Note that in the general partition heavy hitters problem the range of the β_t cannot just be $\{-1, 1\}$. The reason is that we want (5), (6), (7), and (8) to hold simultaneously with some probability strictly larger than 1/2, to allow invoking Chernoff. However, (5) alone can fail with probability at least 1/2 even when using fully independent $\beta_t : [n] \to \{-1, 1\}$. For example, consider when a partition $j \in [N]$ contains exactly two elements with equal weight. Then with probability 1/2 their signs are opposite, in which case $(w_t)_j$ equals zero. Note this is not an issue when S_j contains a single heavy hitter that is much heavier than the total ℓ_2 mass in the rest of the partition.

The second issue is that PartitionCountSketch is again used in the b-tree (see Section B.2 below), which we use in our final heavy hitters algorithm. Unfortunately there, we do *not* have the guarantee that each partition contains one item much heavier than the combined mass of the rest. One way around this is to simply not use the b-tree at all, but to just use the PartitionCountSketch itself. Doing so, though, would make our final query time blow up to $O(\epsilon^{-2} \lg^C n)$ for a large constant C. This can be fixed, and one can achieve our same current query time, by instead implementing the data structures P_j in Section 5 as themselves recursive instantiations of the ExpanderSketch data structure! Then after a constant number of levels of recursion (even one level down), one then implements the P_j using the PartitionCountSketch (see Remark 1). All in all, [KNW10] is thus avoided. We do not delve further into the details since the bounds do not improve, and we feel that a fully detailed exposition is not worth the effort.

We also record a lemma here which will be useful later.

Lemma 15. *Consider an instance of partition heavy hitters with partition $\mathcal{P} = \{S_1, \ldots, S_N\}$ and oracle $\mathcal{O}$. Suppose $A \subset [N]$, and D is an $(\alpha_1, \alpha_2, \epsilon')$-partition point query structure for some $0 < \alpha_1 \le 1 \le \alpha_2$ which succeeds on point querying every $j \in A$, where $\epsilon' = (\alpha_1/3)\epsilon$. Then if $L \subseteq A$*

is defined to be the $2/\epsilon''^2$ indices of A with the largest point query results from D for $\epsilon'' = \epsilon'/\alpha_2$, then L contains all partition ϵ-heavy hitters contained in A.

Proof. Let $\tilde{y}_j$ be the result of a point query of $j \in A$ using D. If $j \in A$ is a partition ϵ-heavy hitter then

$$\tilde{y}_j \geq \alpha_1\epsilon\|y_{\overline{[1/\epsilon^2]}}\|_2 - (\alpha_1/3)\epsilon\|y_{\overline{[1/\epsilon'^2]}}\|_2 \geq (2/3)\alpha_1\epsilon\|y_{\overline{[1/\epsilon^2]}}\|_2.$$

by Eq. (4). Meanwhile, if some j' is not even an ϵ''-partition heavy hitter, then

$$\tilde{y}_{j'} < \alpha_2\epsilon''\|y_{\overline{[1/\epsilon'^2]}}\|_2 + \epsilon'\|y_{\overline{[1/\epsilon'^2]}}\|_2 \leq (2/3)\alpha_1\epsilon\|y_{\overline{[1/\epsilon'^2]}}\|_2.$$

Thus every set which is not even a partition ϵ''-heavy hitter has a point query value strictly less than every partition ϵ-heavy hitter in A. Since there are at most $2/\epsilon''^2$ sets which are partition ϵ''-heavy hitters, correctness follows, since L by definition contains the $2/\epsilon''^2$ sets with largest point query values. ∎

B.2 b-tree

Here we describe the b-tree for partition heavy hitters, which is a slight modification of the Hierarchical CountSketch from [CH08] for non-partition strict turnstile heavy hitters. We assume $2 \leq b \leq N$ is an integer power of 2.

For what follows, we assume vector entries in both $x \in \mathbb{R}^n$ and $y \in \mathbb{R}^N$ are 0-based indexed, e.g. $x = (x_0, \ldots, x_{n-1})^T$ and $y = (y_0, \ldots, y_{N-1})^T$. Conceptually we create virtual streams $J_0, \ldots, J_{\lg_b N}$. For the stream J_r we solve an instance of partition point query with b^r sets in the partition defined as follows. At the level $r = \lg_b N$, the partition is $\mathcal{P}_{\lg_b N} = \mathcal{P} = \{S_0, \ldots, S_{N-1}\}$, with oracle $\mathcal{O}_{\lg_b N} = \mathcal{O}$. For $0 \leq r < \lg_b N$, the partitions are defined inductively: we imagine a complete b-ary tree on $\{0, \ldots, N-1\}$ where the jth leaf node denotes S_j, and for $r < \lg_b N$ each internal node represents a set equal to the union of the sets at its children. Said succinctly, for $r < \lg_b N$, the oracle evaluation $\mathcal{O}_r(i)$ equals $\mathcal{O}(i)/b^{\lg_b N - r}$. Note that since $b = 2^\ell$ is a power of 2, $\mathcal{O}_r$ can be implemented in constant time by bitshifting right $\ell(\lg_b N - r)$ positions ($\lg_b N$ is a fixed value that can be computed once and stored during initialization).

When an update (i, v) is seen in the actual stream, it is fed into a PartitionCountSketch P_r at level r for each $0 \leq r \leq \lg_b N$, with partition $\mathcal{P}_r$ and oracle $\mathcal{O}_r$. Each P_r is chosen to be a $(1/20, 3, \epsilon/60)$-partition point query structure, i.e. the error parameter is $\epsilon' = \epsilon/60$. Set $Q = 2b(\lg_b N)/\epsilon''^2$, which is an upper bound on the total number of point queries we make to all P_r combined. We set the failure probability for each P_r to be $\eta = \delta/Q = \Theta(\epsilon^2\delta/(b\lg_b N))$.

To answer a heavy hitters query, we walk down the tree as follows (our goal is to have, at all times $0 \leq r \leq \lg_b N$, a list L_r of size $O(1/\epsilon^2)$ which contains all the ϵ-heavy hitter partitions at level r). First we set $L_0 = \{0\}$. Then for $r = 1$ to $\lg_b N$, we point query every child of an element of L_{r-1}. We then set L_r to be the nodes at level r whose point query result were amongst the $2/\epsilon''^2$ highest for $\epsilon'' = \epsilon'/3 = \epsilon/180$. We finally return $L = L_{\lg_b N}$ as our set of partition heavy hitters.

Theorem 7. *Suppose $1/N^c \leq \epsilon < 1/2$ and $\delta \in (0, 1/2)$. Then the b-tree as described above produces a correct output L for partition ϵ-heavy hitters with probability at least $1 - \delta$ for any $2 \leq b \leq N$. Furthermore, for any constant $c' \geq 1$ and any constant $0 < \gamma < \gamma_0$ where γ_0 depends on c, c', for any failure probability $1/N^{c'} \leq \delta < 1/2$, there is a choice of b so that the b-tree uses space $O(\epsilon^{-2}\lg N)$, has update time $O(\lg N)$, and query time $O((\epsilon^{-2}\lg N)((\lg N)/(\epsilon\delta))^\gamma)$. The constants in the big-Oh depend on $1/\gamma$.*

Proof. We have $|L| \le 2/\epsilon''^2 = O(1/\epsilon^2)$ by definition of the algorithm. Note that we make at most Q point queries since each L_r has size at most $2/\epsilon''^2$, and each element of L_r has at most b children. Also, there are $\lg_b N$ values of r for which we query children (since the leaves have no children). Thus the total number of queries is at most $2b(\lg_b N)/\epsilon''^2 = Q$. Now we show that, conditioned on the event that all Q point queries succeed, correctness holds by induction on r. This would complete the correctness analysis, since the probability that any point query fails at all is at most $Q\eta = \delta$, by a union bound. For the induction, our inductive hypothesis is that L_r contains every ϵ-partition heavy hitter at level r. This is true for $r = 0$ since $L_0 = \{0\}$, and the root of the tree contains only a single partition. Now, assume L_{r-1} satisfies the inductive hypothesis. Since any ancestor of a partition ϵ-heavy hitter is itself an ϵ-partition heavy hitter, it follows that every partition heavy hitter at level r is the child of some set in L_{r-1}. Thus if we let A be the collection of children of sets in L_{r-1}, Lemma 15 implies that L_r also contains all the partition ϵ-heavy hitters

As for space and time bounds, we choose b to be $((\lg N)/(\epsilon\delta))^\gamma$, rounded up to the nearest integer power of 2. This is at most N for γ sufficiently small since $\epsilon, \delta > 1/\operatorname{poly}(N)$, and at least 2 for N larger than some constant depending on γ (which we can assume without loss of generality, by padding the partition with empty sets). By Lemma 14 and choice of b, the space is asymptotically

$$\frac{1}{\epsilon^2} \cdot \frac{\lg N}{\lg b} \lg(1/\eta) \simeq \frac{1}{\epsilon^2} \cdot \frac{\lg N}{\lg b} \cdot (\lg(1/\delta) + \lg b + \lg\lg N + \lg(1/\epsilon)) \simeq \epsilon^{-2} \lg N.$$

The update time is asymptotically

$$\frac{\lg N}{\lg b} \lg(1/\eta) \simeq \frac{\lg N}{\lg b} \cdot (\lg(1/\delta) + \lg b + \lg\lg N + \lg(1/\epsilon)) \simeq \lg N.$$

The query time is asymptotically

$$\frac{b}{\epsilon^2} \cdot \frac{\lg N}{\lg b} \lg(1/\eta) \simeq \frac{b}{\epsilon^2} \cdot \frac{\lg N}{\lg b} \cdot (\lg(1/\delta) + \lg b + \lg\lg N + \lg(1/\epsilon)) \simeq \epsilon^{-2} \lg N \cdot (\frac{\lg N}{\epsilon\delta})^\gamma.$$

■

Corollary 1. *For any constant* $0 < \gamma < 1/2$, *(non-partition) heavy hitters with error parameter* $\epsilon \in (0, 1/2)$ *and failure probability* $1/\operatorname{poly}(n) < \delta < 1/2$ *can be solved with space* $O(\epsilon^{-2} \lg n)$*, update time* $O(\lg n)$*, query time* $O(\epsilon^{-2} \lg n((\lg n)/(\epsilon\delta))^\gamma)$*, and failure probability* δ*.*

Proof. We can assume $\epsilon > 1/\sqrt{n}$ without loss of generality, since otherwise the trivial solution of keeping x in memory explicitly suffices. Otherwise, the claim follows from Theorem 7 by considering the partition $\mathcal{P} = \{\{1\}, \ldots, \{n\}\}$ with $N = n$ and $\mathcal{O}(i) = i$. ■

C Connection between heavy hitters and list-recoverable codes

Our approach is somewhat related to list-recoverable codes, which were first used in group testing in [Che09, INR10] and in compressed sensing in [NPR12] (and also in subsequent compressing works, e.g. [GNP+13, GLPS14]). We say a code $\mathcal{C} \subset [q]^m$ is (ϵ, ℓ, L)-*list recoverable* if for any sequence of lists $L_1, \ldots, L_m \subset [q]$ with $|L_j| \le \ell$ for all $j \in [m]$, there are at most L codewords in $\mathcal{C}$ whose jth symbol appears in at least a $(1-\epsilon)$-fraction of the lists L_j. A decoding algorithm is then given these m lists and must find these at most L codewords. To see the connection to our current heavy hitters problem, it is known (via the probabilistic method) that such codes exist with $|\mathcal{C}| \geq n$, $q = \mathrm{poly}(\lg n)$, $m = O(\lg n/\lg\lg n)$, and $\ell, L = O(\lg n)$, where ϵ can be made an arbitrarily small constant (see for example the first row of [HW15, Figure 1]). Suppose we had such a code $\mathcal{C}$ with encoding function $\mathrm{enc} : [n] \to \mathcal{C}$. Then our heavy hitters algorithm could follow the scheme of Figure 2. That is, we would instantiate m b-trees $P_1, \ldots, P_m$ for partition heavy hitters with $\mathcal{O}_j(i) = \mathrm{enc}(i)_j$. By picking constants appropriately in the parameter settings, one can ensure that whp at most an ϵ-fraction of the P_j fail. Thus, whp every heavy hitter appears in a $(1-\epsilon)$-fraction of the lists L_j, and we would then perform list-recovery decoding to find all the heavy hitters. The trouble with this approach is that there currently are no *explicit* codes achieving these parameter settings, let alone with linear time encoding and fast decoding. Indeed, this is the source of slight suboptimality in the ℓ_1/ℓ_1 "for all" compressed sensing scheme of [GLPS14].

The key to our progress is to sidestep the lack of explicit optimal list-recoverable codes with linear-time encoding and fast decoding by realizing that list-recoverability is stronger than what we actually need. First of all, list-recovery says that *for all* choices of lists $L_1, \ldots, L_m$, some condition holds. In our case the lists are random (symbols in it contain concatenations with random hash evaluations h_j), since our heavy hitters algorithm is allowed to be a randomized algorithm. Secondly, our decoding algorithm does not need to handle arbitrary lists, but rather lists in which each symbol has a distinct name (recall from Section 4 that the "name" of $z(i)_j$ is $h_j(i)$). This is because, in our problem, with good probability any list L_j has the property that for each heavy hitter i with $z(i)_j \in L_j$, $z(i)_j$ will be reported by a partition point query data structure to have much heavier weight than any other $z \in L_j$ with the same name. Thus our post-processing step using Q_j allows us to only have to perform decoding from lists with a certain structure which does not exist in the general list-recovery problem, namely that no two symbols in L_j agree on the first few (namely $O(\lg\lg n)$) bits.

D A simpler query algorithm for strict turnstile ℓ_1 heavy hitters

In this section, we describe our simpler algorithm for strict turnstile ℓ_1 heavy hitters. The high-level idea is the following: Recall the general turnstile algorithm from Section 5. There we end up having to find ϵ-spectral clusters. This subproblem arose because several data structures P_j^k may err and thereby insert spurious edges into the graph G. Now in the strict turnstile case, we can roughly ensure that we can recognize when a P_j^k errs. By simply deleting all edges returned by such a P_j^k, the remaining graph G has each heavy hitter corresponding to a number of connected components that are completely disjoint from the subgraphs corresponding to other heavy hitters and noise from light elements. Furthermore, for each heavy hitter, one of these connected components has at least 90% of the corresponding codeword $\mathrm{enc}(i)$. We can thus replace the cluster finding algorithm by a simple connected components algorithm and thereby obtain a much simpler algorithm. This section gives all the details of the above.

Our solution needs the more standard definition of heavy hitters: On a query, we must return a set L containing all indices i where $x_i \geq \epsilon\|x\|_1$ and no indices j such that $x_j < (\epsilon/2)\|x\|_1$.

The first thing our solution in the strict turnstile case needs, is the fact that one can maintain $\|x\|_1$ exactly in $O(1)$ space and with $O(1)$ update time. This follows trivially by maintaining a counter C that is initialized as 0, and upon every update (i, Δ), we update $C \leftarrow C + \Delta$. Since we are in the strict turnstile model, we will always have $C = \|x\|_1$.

The second thing is a variant of partition point queries that have been adapted to the ℓ_1 strict turnstile case.

Definition 3. *In the* ℓ_1 strict turnstile partition point query problem *there is some error parameter* $\epsilon \in (0, 1/2)$. *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 index* $j \in [N]$ *such that* $i \in S_j$. *Define a vector* $y \in \mathbb{R}^N$ *such that for each* $j \in [N]$,

$$y_j = \sum_{i \in S_j} x_i.$$

On a query for index $j \in [N]$, *we must output a value* $\tilde{y}_j$ *such that* $y_j \leq \tilde{y}_j \leq y_j + \epsilon \|y_{\overline{[1/\epsilon]}}\|_1$.

Lemma 16. *For any failure probability* $1/\operatorname{poly}(n) < \delta < 1/2$, *there is a solution to the strict turnstile partition point query problem with space* $O(\epsilon^{-1} \lg(1/\delta))$ *words, update time* $O(\lg(1/\delta))$ *and query time* $O(\lg(1/\delta))$. *Furthermore, the solution guarantees that even when it errs, the returned estimate* $\tilde{y}_j$ *is at least* y_j.

Proof. Simply implement the CountMin sketch of [CM05] on the vector y. Thus on an update (i, Δ), feed the update $(\mathcal{O}(i), \Delta)$ to the CountMin sketch and on a query for y_j, return the estimate for j in the CountMin sketch. ■

Using Lemma 16, we can also modify the b-tree from Section B.2 to obtain a no *false negatives* guarantee. More specifically, define a new version of heavy hitters called *threshold heavy hitters with no false negatives*. We say that an algorithm solves threshold ℓ_1 heavy hitters with no false negatives and failure probability δ, if it supports taking a query value $\phi > 0$. If the query value ϕ is less than $\epsilon\|x_{\overline{[1/\epsilon]}}\|_1$, then the algorithm may return an arbitrary answer. If $\phi \geq \epsilon\|x_{\overline{[1/\epsilon]}}\|_1$, then with probability at least $1 - \delta$, it must return a set L that has size $O(\phi^{-1}\|x\|_1)$ and contains all indices i with $x_i \geq \phi$ and no indices j with $x_j \leq \phi - (\epsilon/2)\|x_{\overline{[1/\epsilon]}}\|_1$. With the remaining probability at most δ, it returns the empty set.

Such algorithms can thus recognize when they err (indicated by an empty return set) on queries for sufficiently large ϕ. The following lemma essentially follows from the b-tree solution in Theorem 7 combined with Lemma 16:

Lemma 17. *Suppose* $1/n^c \leq \delta < 1/2$ *where* $c > 0$. *Let* γ *be any constant satisfying* $0 < \gamma < \gamma_0$ *for some fixed constant* γ_0. *Then there is a modification of the b-tree described in Section B.2 and a choice of b, which in the strict turnstile* ℓ_1 *setting solves threshold heavy hitters with no false negatives and failure probability at most* δ. *Furthermore, the b-tree uses space* $O(\epsilon^{-1} \lg n)$, *has update time* $O(\lg n)$, *and query time* $O((\epsilon^{-1} \lg n)((\lg n)/(\epsilon\delta))^{\gamma})$. *The constants in the big-Oh depend on* $1/\gamma$.

Proof. Take the b-tree solution as described in Section B.2. Replace each PartitionCountSketch for P_r with the algorithm from Lemma 16 with approximation factor $\epsilon' = \epsilon/2$. This means that at each level of the b-tree, we can do a point query for the total mass of a subtree. By Lemma 16 these estimates are never underestimates, even when the algorithm errs. And when the algorithm does not err, the returned estimate is at most $\epsilon'\|y_{\overline{[1/\epsilon]}}\|_1 \leq \epsilon'\|x_{\overline{[1/\epsilon]}}\|_1 = (\epsilon/2)\|x_{\overline{[1/\epsilon]}}\|_1$ too high. We set the failure probability for each data structure from Lemma 16 to $O(\epsilon\delta/(b \lg_b n))$.

On a query for all indices i with $x_i \geq \phi$, we traverse the tree roughly as described in Section B.2. The main difference is that when choosing the set L_r after having point queried every child of nodes in L_{r-1}, we choose L_r to be all nodes where the point query returned an estimate of at least ϕ. In case $|L_r| > 3\epsilon^{-1}$ at any level, we abort and return $L = \emptyset$.

Now observe that if no query to the children of nodes in L_{r-1} err, then for $\phi \geq \epsilon \|x_{\overline{[1/\epsilon]}}\|_1$, there can be no more than $3\epsilon^{-1}$ nodes for which the estimate is at least ϕ. This follows since if the point queries did not err, then the true mass of those subtrees must be at least $\phi - (\epsilon/2)\|x_{\overline{[1/\epsilon]}}\|_1 \geq (\epsilon/2)\|x_{\overline{[1/\epsilon]}}\|_1$. There can be at most $3\epsilon^{-1}$ such subtrees. Thus conditioned on not making an error in any point query on the first $r-1$ levels of the b-tree, we will visit at most $3\epsilon^{-1}$ nodes on level r. It follows that we only abort and return an empty list in case some query errs. And if no query errs, we will ask only $O(\epsilon^{-1} b \lg_b n)$ queries. Since we set the failure probability in each algorithm from Lemma 16 to $O(\epsilon\delta/(b \lg_b n))$, the lemma follows. ∎

We only need one more ingredient in our solution, namely a property of edge-expander graphs. We have not been able to find a reference for this result, but suspect it is known.

Lemma 18. *If $G = (V, E)$ is a d-regular δ-edge-expander, then for any constant $0 < \mu < 1$, if we remove any set of at most $(\delta\mu/2)|V|$ vertices in G and their incident edges, then there remains a connected component of size at least $(1-\mu)|V|$.*

Proof. Let $S \subseteq V$ be any set of at most $(\delta\mu/2)|V|$ vertices in G. The number of edges with an endpoint in S is at most $d|S|$ since G is d-regular. Let $C_1, \ldots, C_m \subseteq V$ be the connected components in $G_{V \setminus S}$, where $G_{V \setminus S}$ is the graph obtained from G by removing the vertices S and all edges incident to S. Assume for contradiction that $|C_i| < (1-\mu)|V|$ for all C_i. For each C_i, let C_i^* denote the smaller of C_i and $V \setminus C_i$. It follows from $|C_i|$ being less than $(1-\mu)|V|$ that

$$\frac{\mu}{1-\mu} \cdot |C_i| < |C_i^*| \leq \min\left\{\frac{|V|}{2}, |C_i|\right\}.$$

Since G is a δ-edge-expander and $|C_i^*| \leq |V|/2$, there are at least

$$\delta d|C_i^*| \geq \delta \min\left\{1, \frac{\mu}{1-\mu}\right\} \cdot d|C_i| \geq \delta \min\{1, \mu\} d|C_i| = \delta\mu d|C_i|$$

edges across the cut $(C_i^*, V \setminus C_i^*)$ in G. This cut is just $(C_i, V \setminus C_i)$. But C_i is a connected component in $G_{V \setminus S}$ and thus all edges leaving C_i in G must go to a node in S. Therefore, the number of edges incident to a node in S is at least $\sum_i \delta\mu d|C_i| = \delta\mu d(|V| - |S|)$. But $|S| \leq (\delta\mu/2)|V|$ and hence

$$|V| - |S| \geq (1 - \delta\mu/2)|V| \geq \frac{1-\delta\mu/2}{\delta\mu/2} \cdot |S|.$$

We thus have at least

$$\delta\mu \frac{1-\delta\mu/2}{\delta\mu/2} \cdot d|S| = 2(1-\delta\mu/2)d|S| > d|S|$$

edges incident to S. But this contradicts that there are only $d|S|$ edges incident to S. ∎

The following result due to Alon et al. [ASS08] shows that there exists efficient explicit edge-expanders:

Theorem 8 (Alon et al. [ASS08])**.** *There exists a constant $\delta > 0$, such that for any positive integer n, there exists a 12-regular δ-edge-expander $G = (V, E)$ with $n/2 \leq |V| \leq n$. Furthermore, G is constructable in time $O(\mathrm{poly}(n))$.*

We have set the stage for the description of our simpler ℓ_1 strict turnstile solution. Our new algorithm is described in the following:

Strict Turnstile Algorithm. If $\epsilon = o(1/\lg n)$, we re-execute the reduction in Section A, modified to the ℓ_1 setting. More specifically, we hash to $z = \Theta(1/(\epsilon \lg(1/n)))$ subproblems. Each subproblem is solved with a threshold heavy hitters structure with $\epsilon = O(1/\lg n)$ and failure probability $1/\operatorname{poly}(n)$. Thus on a query to report all heavy hitters, we use the fact that we know $\|x\|_1$ exactly in order to query each threshold heavy hitters structure with $\phi = \epsilon\|x\|_1$. By an analysis similar to Section A, with probability at least $1 - 1/\operatorname{poly}(n)$, this will ensure $\phi \geq \epsilon\|y_{\overline{[1/\epsilon]}}\|_1$ where y denotes the vector corresponding to any of these subproblem. If ϵ is $\Omega(1/\lg n)$, we can also reduce to threshold heavy hitters with the same ϵ and failure probability $1/\operatorname{poly}(n)$ simply by maintaining one structure and querying it with $\phi = \epsilon\|x\|_1$. Thus we have now reduced the problem to solving threshold heavy hitters with $\epsilon = \Omega(1/\lg n)$ in the ℓ_1 strict turnstile case, where we are guaranteed that the query value ϕ is at least $\epsilon\|x_{\overline{[1/\epsilon]}}\|_1$.

We now focus on solving threshold heavy hitters with $\epsilon = \Omega(1/\lg n)$ and failure probability $1/\operatorname{poly}(n)$ for a query value $\phi \geq \epsilon\|x_{\overline{[1/\epsilon]}}\|_1$. Following Section 5, we construct the 12-regular δ-edge-expander from Theorem 8 with $m = \Theta(\lg n/\lg\lg n)$ vertices. Call it F. This takes $\operatorname{poly}\lg n$ time. We store it in adjacency list representation, consuming $O(\lg n/\lg\lg n)$ words of memory.

We then pick hash functions $h_1, \ldots, h_m : [n] \to [\operatorname{poly}(\lg n)]$ independently from a pairwise independent family and instantiate data structures $D_1, \ldots, D_m$. Each D_i is the threshold heavy hitters with no false negatives from Lemma 17. The failure probability of each structure is $1/\operatorname{poly}(\lg n)$ and the approximation factor is ϵ.

On an update (i, Δ), we compute in $O(\lg n)$ time an encoding $\operatorname{enc}(i)$ of i into a string of length $T = O(\lg n)$ bits by an error-correcting code with constant rate that can correct an $\Omega(1)$-fraction of errors (as in Section 5). We again partition $\operatorname{enc}(i)$ into m contiguous bitstrings each of length $t = T/m = \Theta(\lg\lg n)$. We let $\operatorname{enc}(i)_j$ denote the jth bitstring. For each $j = 1, \ldots, m$, we feed the update $(h_j(i) \circ \operatorname{enc}(i)_j \circ h_{\Gamma(j)_1} \circ \cdots \circ h_{\Gamma(j)_{12}}, \Delta)$ to D_j (similarly to Section 5).

On a query with threshold $\phi > 0$, we query each D_j with ϕ. Let L_j be the output of D_j. If L_j is non-empty, we go through all returned values and check if there are two with the same name (same $h_j(i)$). If so, we overwrite $L_j \leftarrow \emptyset$ and say that D_j failed. Now in all remaining L_j, all names are unique. From these lists, we construct the same layered graph G as described in Section 4. Recall that we only add edges if both endpoints suggest it. Now the crucial observation which makes the ℓ_1 strict turnstile easier to solve, is that the D_j's guarantee no false negatives. Thus if some D_j did not err and did not fail, then no node in G corresponding to a name $h_j(i)$ returned by D_j, can be connected by an edge to a node $h_{j'}(i')$ returned by a $D_{j'}$ for any $i' \neq i$. To see this, notice that the actual index $h_{j'}(i) \circ \operatorname{enc}(i)_{j'} \circ h_{\Gamma(j')_1} \circ \cdots \circ h_{\Gamma(j')_{12}}$ corresponding to $h_{j'}(i)$ must not be in $L_{j'}$ for this to happen. But we are guaranteed no false negatives, so if $D_{j'}$ returns anything, it must have $h_{j'}(i) \circ \operatorname{enc}(i)_{j'} \circ h_{\Gamma(j')_1} \circ \cdots \circ h_{\Gamma(j')_{12}}$ in its output. To summarize, for every index i with $x_i > \phi$, all D_j that did not err and did not fail produce nodes corresponding to names $h_j(i)$, and these nodes are not connected to nodes of name $h_{j'}(i')$ for an $i' \neq i$. Furthermore, for two D_j and $D_{j'}$ that both did not err and fail, the edges between the returned nodes are in the graph G. But the hash function h_j have collisions with probability $1/\operatorname{poly}(\lg n)$ and the data structures D_j have failure probability $1/\operatorname{poly}(\lg n)$. Thus with probability $1 - 1/\operatorname{poly}(n)$, there are at most a γ-fraction of the D_j that either fail or err. Here we can choose γ as an arbitrarily small constant by increasing the polylogs in $1/\operatorname{poly}(\lg n)$ failure probability. By Lemma 18, we get that with probability $1 - 1/\operatorname{poly}(n)$, each index i with $x_i \geq \phi$ has a connected component in G of size at least $0.9m$. We now finish off by computing all connected components in G. For each component of size at least $0.9m$, we stitch together the pieces $\operatorname{enc}(i)_j$ to obtain 90% of the codeword $\operatorname{enc}(i)$. We decode i in linear time in its bit length, which is $O(\lg n)$ time. Finally, we verify all these returned indices by doing a point query against a separate CountMin structure with failure probability $1/\operatorname{poly}(n)$ and the same ϵ. This also costs $O(\lg n)$ time per index.

Since we are guaranteed that each returned list L_j has size at most $\phi^{-1}\|x\|_1$, the query time is $O(\phi^{-1}\|x\|_1 \lg n)$ for decoding codewords and for verifying the recovered indices (the norm of the vector represented by each D_j is precisely the same as the norm of x). Now ϕ is $\epsilon\|x\|_1$, so the time spent is $O(\epsilon^{-1} \lg n)$. For querying the structures D_j, Lemma 17 gives us a query time of $O((\epsilon^{-1} \lg\lg n)((\lg\lg n \operatorname{poly}(\lg n))/\epsilon))^{\gamma})$ per D_j. Since $\epsilon = \Omega(1/\lg n)$, we can choose γ sufficiently small to obtain a query time of $\epsilon^{-1} \lg^{\gamma'} n$ for any constant $\gamma' > 0$. Since we have $m = O(\lg n/\lg\lg n)$, the total time for querying all D_j is $O(\epsilon^{-1} \lg^{1+\gamma} n)$ for any constant $\gamma > 0$.

For the update time, observe that each D_j has update time $O(\lg\lg n)$ and there are $O(\lg n/\lg\lg n)$ such structures to be updated, for a total of $O(\lg n)$ time. Computing $\operatorname{enc}(i)$ also takes $O(\lg n)$ time. The space is optimal $O(\epsilon^{-1} \lg n)$ words.

Theorem 9. *For any constant $0 < \gamma < 1/2$, heavy hitters with error parameter $\epsilon \in (0, 1/2)$ and failure probability $1/\operatorname{poly}(n) < \delta < 1/2$ can be solved in the strict ℓ_1 turnstile setting with optimal $O(\epsilon^{-1} \lg n)$ space, update time $O(\lg n)$, query time $O(\epsilon^{-1} \lg^{1+\gamma} n)$, and failure probability δ.*

E Expected Time for Strict Turnstile ℓ_1

If we are satisfied with expected query time, then there is an even simpler and very practical algorithm for the strict turnstile ℓ_1 case. The algorithm is based on the b-tree, but with $b = 2$, i.e. a binary tree. As in Section D, we solve the more standard variant of heavy hitters in which we must return a set L containing all indices i where $x_i \geq \epsilon\|x\|_1$ and no indices j such that $x_j < (\epsilon/2)\|x\|_1$.

For what follows, we assume vector entries in $x \in \mathbb{R}^n$ are 0-based indexed, e.g. $x = (x_0, \ldots, x_{n-1})^T$. Conceptually we create virtual streams $J_0, \ldots, J_{\lg_2 n}$. For the stream J_r we solve an instance of partition point query with 2^r sets in the partition defined as follows. At the level $r = \lg_2 n$, the partition is $\mathcal{P}_{\lg_2 n} = \mathcal{P} = \{S_0, \ldots, S_{n-1}\}$, with oracle $\mathcal{O}_{\lg_2 n} = \mathcal{O}$. For $0 \leq r < \lg_2 n$, the partitions are defined inductively: we imagine a complete binary tree on $\{0, \ldots, n-1\}$ where the jth leaf node denotes S_j, and for $r < \lg_2 n$ each internal node represents a set equal to the union of the sets at its children. Said succinctly, for $r < \lg_2 n$, the oracle evaluation $\mathcal{O}_r(i)$ equals $\mathcal{O}(i)/2^{\lg_2 n - r}$. Note that $\mathcal{O}_r$ can be implemented in constant time by bitshifting right $\ell(\lg_2 n - r)$ positions ($\lg_2 n$ is a fixed value that can be computed once and stored during initialization).

When an update (i, v) is seen in the actual stream, it is fed into a PartitionCountSketch P_r at level r for each $0 \leq r \leq \lg_2 n$, with partition $\mathcal{P}_r$ and oracle $\mathcal{O}_r$. Each P_r is chosen to be the partition point query structure from Lemma 16 with error parameter $\epsilon' = \epsilon/2$ and failure probability $\delta = 1/4$. This has each P_r using $O(\epsilon^{-1})$ words of space, supporting updates and queries in $O(1)$ time. We also maintain one CountMin sketch on x with error parameter $\epsilon' = \epsilon/2$ and failure probability $1/\operatorname{poly}(n)$.

The space is $O(\epsilon^{-1} \lg n)$ and the update time is $O(\lg n)$.

To answer a heavy hitters query, we walk down the tree as follows (our goal is to have, at all times $0 \leq r \leq \lg_2 n$, a list L_r of size $O(1/\epsilon^{-1})$ which contains all the ϵ-heavy hitter partitions at level r). First we set $L_0 = \{0\}$. Then for $r = 1$ to $\lg_2 n$, we point query every child of an element of L_{r-1}. We then set L_r to be the nodes at level r whose point query result was at least $\epsilon\|x\|_1$. Recall from Section D that in the strict turnstile ℓ_1 setting, we can maintain $\|x\|_1$ exactly. Once we have computed $L_{\lg_2 n}$, we point query each element in the list against the CountMin sketch and filter out those indices where the returned value is less than $\epsilon\|x\|_1$.

The key insight is that the data structure from Lemma 16 never returns an underestimate. Thus the list $L_{\lg_2 n}$ will always contain all indices i with $x_i \geq \epsilon\|x\|_1$. The only issue is that we may consider more candidate heavy hitters than really exist, which might increase the query time t_q. We show that in fact t_q remains small *in expectation*.

Let the $2n-1$ nodes in the conceptual binary tree be called $u_1, \ldots, u_{2n-1}$. During query, we traverse this tree starting at the root. While visiting a node u at level r, we query it in P_r and only recurse to its children if the subtree corresponding to u is declared to have mass at least $\epsilon\|x\|_1$ at level r. For $j \in [2n-1]$ let Y_j be an indicator random variable for whether P_r declares u_j as having mass at least $\epsilon\|x\|_1$. Note the runtime of a query is $\sum_j Y_j$. Thus the expected runtime satisfies

$$t_q = \mathbb{E} \sum_{j=1}^{2n-1} Y_j = \sum_j \mathbb{E}\, Y_j.$$

Now, what is $\mathbb{E}\, Y_j$? Call a node u_j which *actually* corresponds to a subtree of mass at least $(\epsilon/2)\|x\|_1$ *heavy*. Then for heavy u_j, we use the simple upper bound $\mathbb{E}[Y_j] \leq 1$; there are at most $2\epsilon^{-1}\lg n$ such heavy j summed over the entire tree. Otherwise, if d is the distance to the lowest ancestor v of u_j which is heavy, then for Y_j to be 1, it must be that every point query to an ancestor of u_j below v must fail. This happens with probability $1/4^{d-1}$ since the data structures are independent across the levels of the tree. However, if we fix d and ask how many nodes in the entire tree have shortest distance to a heavy ancestor equal to d, then that is at most $(2\epsilon^{-1}\lg n) \cdot 2^{d-1}$.

Therefore the expected query time t_q for traversing the tree is:

$$\sum_j \mathbb{E}\, Y_j \leq (\epsilon^{-1}\lg n) \cdot \sum_{r=0}^{\infty} \left(\frac{2}{4}\right)^r = O(\epsilon^{-1}\lg n).$$

Finally, we also have $|L_{\lg_2 n}|$ queries to a CountMin sketch with failure probability $1/\operatorname{poly}(n)$. These queries cost $O(\lg n)$ time each. Thus we need to bound $O(\lg n) \cdot \mathbb{E}\,|L_{\lg_2 n}|$. By an argument similar to the above, consider a leaf u_j corresponding to an index j with $x_j \leq (\epsilon/2)\|x\|_1$. Let d denote the distance from u_j to its nearest heavy ancestor. For u_j to be in $L_{\lg_2 n}$, every point query on this path must fail. This happens with probability $1/4^{d-1}$. Now how many *leaves* can have shortest distance to a heavy ancestor equal to d? Since we are only considering leaves, ancestors of distance d are at the exact same level for all leaves. But a level has at most $2\epsilon^{-1}$ heavy nodes, and thus there are at most $(2\epsilon^{-1}) \cdot 2^{d-1}$ leaves with distance exactly d to their nearest heavy ancestor. By an argument similar to the above, we get that $\mathbb{E}\,|L_{\lg_2 n}| = O(\epsilon^{-1})$. We conclude:

Theorem 10. *For any constant $0 < \gamma < 1/2$, heavy hitters with error parameter $\epsilon \in (0, 1/2)$ and failure probability $1/\operatorname{poly}(n) < \delta < 1/2$ can be solved in the strict ℓ_1 turnstile setting with optimal $O(\epsilon^{-1}\lg n)$ space, update time $O(\lg n)$, expected query time $O(\epsilon^{-1}\lg n)$, and failure probability $1/\operatorname{poly}(n)$.*