On Unifying the Space of ℓ_0-Sampling Algorithms

Graham Cormode[*] Donatella Firmani[†]

Abstract

The problem of building an ℓ_0-sampler is to sample near-uniformly from the support set of a dynamic multiset. This problem has a variety of applications within data analysis, computational geometry and graph algorithms. In this paper, we abstract a set of steps for building an ℓ_0-sampler, based on sampling, recovery and selection. We analyze the implementation of an ℓ_0-sampler within this framework, and show how prior constructions of ℓ_0-samplers can all be expressed in terms of these steps. Our experimental contribution is to provide a first detailed study of the accuracy and computational cost of ℓ_0-samplers.

1 Introduction

In recent years, there has been an explosion of interest in sketch algorithms: compact data structures that compress large amounts of data to constant size while capturing key properties of the data. For example, sketches realizing the Johnson-Lindenstrauss lemma [15] allow the Euclidean distance between high dimensional vectors to be approximated accurately via much lower-dimensional projections [13, 1, 8]. Many constructions in the new area of compressed sensing can also be expressed as sketches [12]. Since most sketches can be updated incrementally and merged together, they can be used in streaming and distributed settings. Due to this flexibility, sketches have found use in a wide range of applications, such as network monitoring [4], log analysis [21] and approximate query processing [6].

From these practical motivations, and since there are often several competing sketch constructions for the same problem, it is important to unify and compare the efficacy of different solutions. Prior work has evaluated the performance of sketches for recovering frequent items [18, 7], and for tracking the cardinality of sets of items [19, 3]. In this work, we focus on sketches for a fundamental sampling problem, known as ℓ_0-sampling. Over a large data set that assigns weights to items, the goal of an ℓ_0-sampler is to draw (approximately) uniformly from the set of items with non-zero weight. This is challenging, since while an item may appear many times within the raw data, it may have an aggregate weight of zero; meanwhile, another item may appear only once with a non-zero weight. The sketch must be designed so that only the aggregate weight influences the sampling process, not the number of occurrences of the item.

This sampling distribution turns out to have a number of applications. Drawing such a sample allows one to characterize many properties of the underlying data, such as the distribution of occurrence frequencies, and other natural functions of these frequencies. Such queries over the “inverse distribution” (which gives the fraction of items whose count is i) are important within a variety of network and database applications [5]. ℓ_0-sampling is also used over geometric data, to generate ϵ-nets and ϵ-approximations to approximate the occupancy of ranges; and to approximate the weight of (geometric) minimum spanning trees [10]. Most recently, it has been shown that ℓ_0-sampling allows the construction of graph sketches, which in turn provide the first sketch algorithms for computing connected components, k-connectivity, bipartiteness and minimum spanning trees over graphs [2].

In response to these motivations, several different constructions of ℓ_0-samplers have been proposed. Early constructions made use of universal hash functions [5, 10]. Stronger results were shown using higher-independence hash functions [20], and most recently assuming fully-independent hash functions [16]. Comparing these approaches, we observe that there is a common outline to them all. A hashing procedure assigns items to levels with a geometric distribution, so that each item is consistently assigned to the same level(s) whenever it appears in the data. Then at each level, a “sparse recovery” data structure summarizes the items and weights. If the number of items with non-zero weight at a level is small enough, then the full set can be recovered. A sample is drawn by choosing an appropriate level, attempting to recover the set of items at that level, and selecting one as the sampled item.

Although similar in outline, the constructions differ in the details of the process and in their description. In this work, we provide a single unified framework for ℓ_0-sampling and its analysis, and demonstrate how the prior constructions fit into this framework based on a small number of choices: primarily, the strength of hash functions used, and the nature of the recovery data structures adopted. This characterization allows us to better understand the choices in the prior constructions. It also allows us to present a detailed empirical comparison of different parameter settings, and their in-

[*] graham@research.att.com

[†] firmani@dis.uniroma1.it. This work was done while the author was visiting Shannon Laboratories and DIMACS.

fluence on the performance of the sampling procedure, in terms of speed and uniformity. Despite their many applications, there has been no prior experimental comparison of ℓ_0-sampling algorithms and their costs. Our experiments show that these algorithms can be implemented effectively, and sample accurately from the desired distribution with low costs.

Outline. First, we present the formal definition of ℓ_0-sampling in Section 1.1. In Section 2 we give a ℓ_0-sampler algorithm, and analyze the performances that can be achieved assuming a perfect s-sparse recovery algorithm. We then describe how to construct a randomized exact s-sparse recovery algorithm, and hence realize an ℓ_0-sampler. We finally discuss how this framework incorporates the results in [10, 16, 20]. We present our experimental comparison of methods in Section 3.

1.1 The ℓ_0-sampling problem

We give a formal definition of ℓ_p-samplers over data defining a vector.

DEFINITION 1. (ℓ_p-DISTRIBUTION) *Let $a \in \mathbb{R}^n$ be a non-zero vector. For $p > 0$ we call the ℓ_p-*distribution *corresponding to vector a the distribution on $[n]$ that takes i with probability $\frac{|a_i|^p}{\|a\|_p^p}$ where $\|a\|_p = \left(\sum_{i=1}^n |a_i|^p\right)^{1/p}$ is the ℓ_p-norm of a. For $p = 0$, the ℓ_0-*distribution *corresponding to a is the uniform distribution over the non-zero coordinates of a, which are denoted as* $\operatorname{supp} a$.

A sketch algorithm is an ℓ_0-sampler if it can take as input a stream of updates to the coordinates of a non-zero vector a, and output a non-zero coordinate (i, a_i) of a[1]. The algorithm may fail with small probability δ and, conditioned on no failure, outputs the item $i \in \operatorname{supp} a$ (and corresponding weight a_i) with probability

$$(1.1) \qquad (1 \pm \epsilon)\frac{1}{\|a\|_0} \pm \delta$$

for a parameter ϵ. The quantity $\|a\|_0 := |\operatorname{supp} a|$ is often called the ℓ_0-norm (although it is not strictly a norm) and represents the number of non-zero coordinates of a.

2 The ℓ_0-sampling process

We observe that existing ℓ_0-sampling algorithms can be described in terms of a three-step process, namely SAMPLING, RECOVERY and SELECTION. This is illustrated schematically in Figure 1.

1. SAMPLING. Given vector a, the sampling process defines m vectors $a(1), \ldots, a(m) \in \mathbb{R}^n$ from a. For each $j \in [m]$, the vector $a(j)$ contains a subset S_j of the coordinates of the input vector a while the others are set to zero—that is, $\operatorname{supp} a(j) \subseteq \operatorname{supp} a$. These vectors are not materialized, but are summarized implicitly by the next step.
2. RECOVERY. The recovery piece creates m data structures based on a parameter s. For each $j \in [m]$, if $a(j)$ is s-sparse then this structure allows us to recover $a(j)$ with probability $1 - \delta_r$. We call this "exact s-sparse recovery".
3. SELECTION. When the ℓ_0-sampler is used to draw a sample, a *level* $j \in [m]$ is chosen so that the vector $a(j)$ should be s-sparse (but non-empty). If a non-zero vector $a'(j)$ at this level is successfully recovered, then an entry of this vector $(i, a'(j)_i)$ is selected and returned as the sampled item.

As mentioned above, existing ℓ_0-samplers [10, 16, 20] fit this pattern, but vary in details. Specifically, they differ in how the subsets S_j are chosen in the SAMPLING step, and in the specification of the s-sparse recovery data structure.

2.1 ℓ_0-sampling with k-wise independent hashing

In this section, we describe and analyze an instantiation of the above framework which synthesizes the prior results in this area. We then show how this captures existing algorithms for this problem as special cases.

Let $\mathcal{F}_k$ be a k-wise independent family of hash functions, with $k = O(s)$, and let $h : [n] \to [n^3]$ be randomly selected from $\mathcal{F}_k$. The ℓ_0-sampling algorithm is defined by:

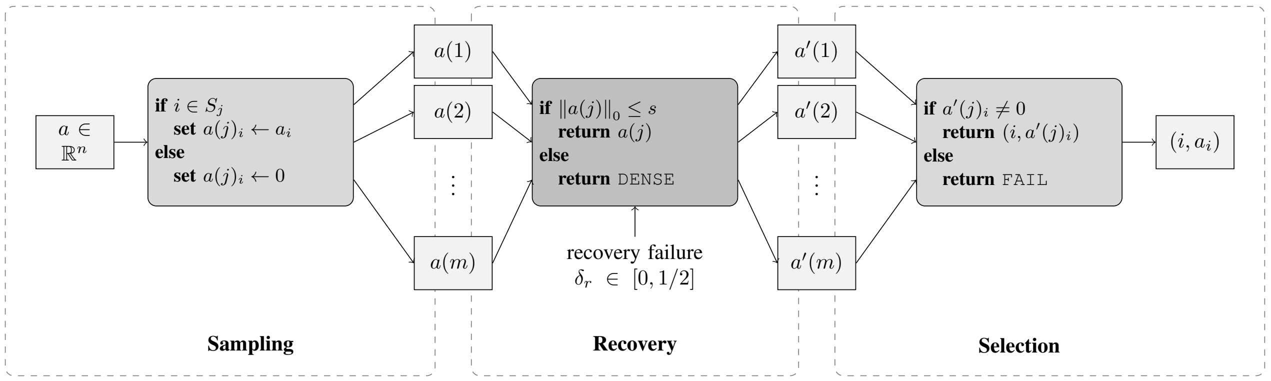
1. SAMPLING. If $n^3 2^{-j} \geq h(i)$, then set $a(j)_i = a_i$, else set $a(j)_i = 0$. We get $a(j)_i = a_i$ with uniform probability $p_j = 2^{-j}$, and $m = O(\log n)$.
2. RECOVERY. We describe and analyze how to perform the s-sparse recovery in Section 2.3.
3. SELECTION. The selection process identifies a level j that has a non-zero vector $a(j)$ and attempts to recover a vector from this level, as $a'(j)$. If successful, the non-zero coordinate $(i, a'(j)_i)$ obtaining the smallest value of $h(i)$ is returned as the sampled item.

We use the notation $N_j = \|a(j)\|_0$ and $N = \|a\|_0$. The random variable N_j can be thought of as the sum of N Bernoulli random variables $x_i \in \{0, 1\}$, where x_i represents the event $a(j)_i = a_i$, $a_i \in \operatorname{supp} a$. The expectation of N_j is $\mathbf{E}[N_j]$.

2.2 Analysis of the ℓ_0-sampler

We show that this ℓ_0-sampler achieves high success probability and small error on the ℓ_0-distribution, without requiring full randomness of $\mathcal{F}_k$. For the purpose of this analysis, let $\mathcal{P}_s$ be a perfect s-sparse recovery algorithm: that is, an algorithm which can recover

[1]More generally, we also seek solutions so that, given sketches of vectors a and b, we can form a sketch of $(a+b)$ and sample from the ℓ_0-distribution on $(a+b)$. All the algorithms that we discuss have this property.



$a \in \mathbb{R}^n$
if $i \in S_j$ set $a(j)_i \leftarrow a_i$ else set $a(j)_i \leftarrow 0$
$a(1)$
$a(2)$
$a(m)$
if $\|a(j)\|_0 \le s$ return $a(j)$ else return DENSE
recovery failure $\delta_r \in [0, 1/2]$
$a'(1)$
$a'(2)$
$a'(m)$
if $a'(j)_i \neq 0$ return $(i, a'(j)_i)$ else return FAIL
(i, a_i)
Sampling
Recovery
Selection

Figure 1: Overall ℓ_0-sampling process, $m = O(\log n)$.

any vector x with $\|x\|_0 \le s$, and otherwise outputs FAIL. Let $\mathcal{S}$ denote the above selection step algorithm using $\mathcal{P}_s$.

LEMMA 2.1. (PROBABILITY OF SUCCESSFUL RECOVERY) *Given a k-wise independent family $\mathcal{F}_k$, with $k \ge s/2$ and $s = O(\log 1/\delta_t)$, the ℓ_0-sampler successfully recovers an item with probability at least $1 - \delta_t$.*

Proof. Let $a(j)$ be the vector extracted by the sampling step and submitted to the recovery step. If $1 \le \|a(j)\|_0 \le s$: (i) $\mathcal{P}_s$ recovers the vector $a'(j) = a(j)$ with probability 1, because $a(j)$ is s-sparse and so $\mathcal{P}_s$ will succeed; (ii) $\mathcal{S}$ outputs a non-zero coordinate (i, a_i), because $a'(j)$ is non-zero and $\mathcal{S}$ can choose $a'(j)$.

The probability of the event $1 \le \|a(j)\|_0 \le s$ can therefore lower-bound the probability of success of the ℓ_0-sampler, $\mathbf{Pr}[\text{output}_{\mathcal{S}} \in \operatorname{supp} a]$. Consider in particular the level j such that

$$\frac{s}{4} \le \mathbf{E}[N_j] \le \frac{s}{2}$$

For this vector $a(j)$, we can compute the probability of the event $1 \le \|a(j)\|_0 \le s$, from the probability that N_j is close to its expectation $\mathbf{E}[N_j] = Np_j$:

$$\begin{aligned}\mathbf{Pr}[|N_j - \mathbf{E}[N_j]| < \mathbf{E}[N_j]]& \\ &\le \mathbf{Pr}[1 \le N_j \le 2\mathbf{E}[N_j]] \\ &\le \mathbf{Pr}[1 \le N_j \le s]\end{aligned}$$

We invoke [23, Theorem 2.5], which gives a Chernoff bound-like result under limited independence. If X is the sum of k-wise independent random variables, each of which is confined to the interval $[0, 1]$, then for $r \ge 1$ and $k = \lceil r\mathbf{E}[X] \rceil$:

$$\mathbf{Pr}[|X - \mathbf{E}[X]| \ge r\mathbf{E}[X]] \le \exp(-\mathbf{E}[X]r/3)$$

Thus, since we have $\mathbf{E}[N_j] \ge \frac{s}{4}$, we obtain

$$\begin{aligned}\mathbf{Pr}[|N_j - \mathbf{E}[N_j]| \ge \mathbf{E}[N_j]] \le \exp(-s/12) \le \delta_t \\ \text{if } s \ge 12 \ln \tfrac{1}{\delta_t}\end{aligned}$$

Setting $s = 12 \log 1/\delta_t$ we ensure that we can recover at this level j with high probability (and possibly also recover at higher levels j also). Hence, we obtain the claimed result on the success probability of the ℓ_0-sampler:

$$\mathbf{Pr}[\text{output}_{\mathcal{S}} \in \operatorname{supp} a] \ge \mathbf{Pr}[1 \le N_j \le s] \ge 1 - \delta_t$$

□

LEMMA 2.2. (OUTPUT DISTRIBUTION OF ℓ_0-SAMPLER) *Let $1 - \delta$ be the the success probability of the ℓ_0-sampler (from Lemma 2.1). Then the sampler outputs the item $i \in \operatorname{supp} a$ with probability $(1 \pm \exp(-s))\frac{1}{N} \pm \delta$*

Proof. We make use of the fact if h is chosen to be $O(\log 1/\epsilon)$-wise independent and has large enough range, then the i which obtains the smallest value of $h(i)$ is chosen with probability $(1 \pm \epsilon)/N$. This follows since h is approximately min-wise independent [14]. Since h is $O(s)$-wise independent, each i should be chosen with probability $(1 \pm \exp(-s))/N$.

However, we have to account for the fact that some instances fail, due to having too many items chosen to level j. By the above argument, this happens with probability at most $1 - \delta$. Consequently, the probability of picking i is affected by at most an additive δ amount. Thus, we obtain that i is output with probability $(1 \pm \exp(-s))\frac{1}{N} \pm \delta$. □

Note that in our setting, the single parameter s controls both the relative error term and the additive error term, so to obtain a guarantee of the form $(1 \pm \epsilon)\frac{1}{N} \pm \delta$, we set $s = O(\max(\log 1/\epsilon, \log 1/\delta))$.

Recovery level selection. The above analysis indicates that there is likely to be a level j at which recovery can succeed, and that sampling from this level approximates the desired distribution. In an implementation, there are two approaches to choosing the level for recovery. The first is to run an approximate ℓ_0 estimation algorithm in parallel with the ℓ_0-sampler, and use the estimate of N to choose the level [17]. This is well-principled, but adds an overhead to the process. The alternative is to aggressively attempt to recover vectors, and sample from the first level that succeeds. We compare these alternatives empirically in Section 3

2.3 Sparse Recovery In this section, we discuss how to implement an efficient s-sparse recovery algorithm. Many approaches have been made to this question, due to its connection to problems in coding theory and compressed sensing. In [11], Ganguly provides a solution to the exact s-sparse recovery problem for non-negative vectors $a \in (\mathbb{Z}^+)^n$. The space required is close to linear in s. In [20] a sketch-based solution is described which provides small failure probability for $a \in \mathbb{R}^n$, but requires substantial space (polynomial in s). Here, we give an exact s-sparse recovery algorithm for $a \in \mathbb{Z}^n$, built using multiple instances of a 1-sparse recovery structure.

2.3.1 Perfect 1-sparse recovery A natural approach to building a 1-sparse recovery algorithm is to keep track of the sum of weights ϕ, and a weighted sum of item identifiers ι, as:

$$\iota = \textstyle\sum_{i \in \mathrm{supp}\, a} a_i \cdot i \qquad \text{and} \qquad \phi = \textstyle\sum_{i \in \mathrm{supp}\, a} a_i$$

Given an update $(i, \Delta a_i)$ to the coordinates of the input vector a, the counters are updated accordingly: $\iota = \iota + \Delta a_i \cdot i$ and $\phi = \phi + \Delta a_i$. It is easy to verify that, if the input vector a is indeed 1-sparse, $i = \iota/\phi$ and $a_i = \phi$. However, additional tests are required to determine if a is truly 1-sparse. A simple test proposed by Ganguly [11] is to additionally compute $\tau = \sum_{i \in \mathrm{supp}\, a} a_i \cdot i^2$, and check that $\iota^2 = \phi\tau$. The test will always pass when a is 1-sparse, and it is straightforward to show that it will not pass when a is not 1-sparse, provided all entries of a are non-negative. However, when a may contain negative entries, the test may give a false positive.

We now propose a variant test which works over arbitrary integer vectors. Let p be a suitably large prime, and choose a random $z \in \mathbb{Z}_p$. We compute the fingerprint $\tau = \sum_{i \in \mathrm{supp}\, a} a_i \cdot z^i \mod p$, and test if $\tau = \phi \cdot z^{\iota/\phi} \mod p$.

LEMMA 2.3. *If a is 1-sparse, then the fingerprint test always gives a positive answer. If a is s-sparse, $s > 1$, then the fingerprint test gives a negative answer with probability at least $1 - n/p$.*

Proof. If a is 1-sparse, the input vector contains a single non-zero coordinate (i, a_i). Therefore $\iota = a_i \cdot i$ and $\phi = a_i$. We get $\phi \cdot z^{\iota/\phi} = a_i \cdot z^{\frac{a_i \cdot i}{a_i}} = a_i \cdot z^i$, therefore $\tau = \phi \cdot z^{\iota/\phi} \mod p$, as required.

For the other case, it is easy to verify that the fingerprint test gives a positive result in two cases: (i) a is 1-sparse; (ii) z is a root in $\mathbb{Z}_p$ of the polynomial $p(z) = \sum_{i \in \mathrm{supp}\, a} a_i \cdot z^i - \phi \cdot z^{\iota/\phi}$.

The "failure" probability of the test is given by the probability of (ii). Since $p(z)$ has degree n, $p(z)$ has at most n roots in $\mathbb{Z}_p$. As z is chosen independently of of i, ι, ϕ, the probability that z is one of these roots is at most n/p, and the claimed result follows. □

The space required by this 1-sparse recovery algorithm is $O(\log n + \log u + \log p)$ bits, where $[-u, +u]$ is the range of the frequencies of a. In the following we assume $O(\log n + \log u + \log p) = O(\log n)$.

2.3.2 Exact s-sparse recovery algorithm We now describe how to build an s-sparse recovery algorithm using 1-sparse recovery as a primitive. Let $\mathcal{G}_2$ be a family of pairwise independent hash functions, and let $f_r : [n] \to [2s]$, $r \in [\log s/\delta_r]$, be randomly selected from $\mathcal{G}_2$. We denote by $\mathcal{P}_1$ the 1-sparse recovery algorithm shown above in Section 2.3.1, while $\mathcal{R}_s$ is our exact s-sparse recovery algorithm. Similar to [11], we use a two-dimensional array, with $\log s/\delta_r$ rows and $2s$ columns, where each cell contains an instance of $\mathcal{P}_1$, as illustrated in Figure 2.

Given an update $(i, \Delta a_i)$ to the coordinates of input vector a, $(i, \Delta a_i)$ is submitted to $\log s/\delta_r$ independent instances of $\mathcal{P}_1$, each of them having position $\langle \text{row}, \text{column} \rangle = \langle r, f_r(i) \rangle$. To perform the recovery, the algorithm interrogates each of the instances of $\mathcal{P}_1$, and extracts the unique item stored there, if there is one. The total collection of recovered items and weights are returned as the recovered vector a'.

LEMMA 2.4. *The exact s-sparse recovery algorithm recovers an s-sparse vector a, with probability at least $1 - \delta_r$.*

Proof. We start with the analysis of the probability $\mathbf{Pr}[\mathrm{rec}_i]$ that $\mathcal{R}_s$ recovers a *particular* coordinate (i, a_i) of a, then we extend the result to the (s-sparse) vector a. To this end, let $C_{r,i}$ be the sum of (at most) $s - 1$ random variables $c_l \in \{0, 1\}$, each of them representing the event $f_r(i) = f_r(l)$. We have $\mathbf{Pr}[c_l = 1] = 1/(2s)$. Writing $C_i = \sum_{l \neq i, l \in \mathrm{supp}\, a} c_l$, we have that $\mathbf{Pr}[C_{r,i} \geq 1] \leq \mathbf{E}[C_{r,i}] < \frac{1}{2}$. The probability that we do not recover i in any row is therefore $\frac{1}{2}^{\log s/\delta_r} = \delta_r/s$. Summed over the s non-zero coordinates, we recover them all with probability at least $1 - \delta_r$. □

We comment that there is the possibility of a false positive if one of the $\mathcal{P}_1$ structures erroneously reports a

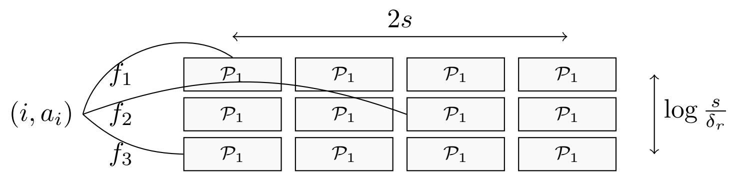


Figure 2: Exact s-sparse recovery with perfect 1-sparse recovery.

singleton item. This probability is polynomially small in n based on the choice of the prime $p = \text{poly}(n)$, so we discount it. We also remark that in the case that a has more than s entries, the procedure may recover a subset of these. We can either accept this outcome, or detect it by keeping an additional fingerprint of a in its entirety, and comparing this to the fingerprint of the recovered vector. The size of this data structure $\mathcal{R}_s$ is $O(s \log(s/\delta_r))$ instances of $\mathcal{P}_1$, i.e. a total of $O(s \log n \log(s/\delta_r))$ bits[2].

2.4 Main Result and Comparison with previous results

Replacing the perfect s-sparse recovery procedure assumed in Section 2.2 with the above procedure affects the output distribution by at most an additive δ_r. Hence, combining Lemmas 2.1, 2.2, and 2.4, we obtain:

THEOREM 2.1. *Given a k-wise independent family $\mathcal{F}_k$, with $k \geq s/2$ and $s = O(\log 1/\epsilon + \log 1/\delta)$, the ℓ_0-sampler succeeds with probability at least $1 - \delta$ and, conditioned on successful recovery, outputs the item $i \in \operatorname{supp} a$ with probability $(1 \pm \epsilon)\frac{1}{N} \pm \delta$.*

The space complexity of the ℓ_0-sampler is $O(s \log^2 n \log s/\delta)$ bits, if we set $\delta_r = \delta_t = \delta/2$: the space is dominated by the $O(\log n)$ instances of the s-sparse recovery algorithms. If we set $\epsilon = \delta = \text{poly}(1/n)$, then the additive error term can absorb the relative error term, and we obtain:

COROLLARY 2.1. *There is an ℓ_0-sampler using space $O(\log^4 n)$ bits that succeeds with probability at least $1 - n^{-c}$ and, conditioned on successful recovery, outputs item i with probability $\frac{1}{N} \pm n^{-c}$ for constant c.*

We now compare this construction to those described in prior work.

Full independence. The analysis of the ℓ_0-sampler in [16] assumes full independence of $\mathcal{F}_k$, and the recovery step is implemented with a perfect s-sparse recovery algorithm. Jowhari, Saglam and Tardos prove that the ℓ_0-sampler in [16] succeeds with probability at least $1 - \delta$ and, conditioned on no failure, outputs the item $i \in \operatorname{supp} a$ with almost uniform probability. The total space cost is $O(\log n)$ levels, each of which uses an s-sparse recovery algorithm for $s = O(\log 1/\delta)$. Assuming full independence means that ϵ can be assumed to be 0, and so the error arises from the failure probability.

ϵ-min-wise independence. The analysis of the ℓ_0-sampler in [20] assumes $c \log n/\epsilon$-wise independence of $\mathcal{F}_k$, with $c > 1$, to get ϵ-min-wise independence over N elements [14]. The recovery step is implemented with an exact s-sparse recovery algorithm, with $\delta_r = n^{-c}$.

Monemizadeh and Woodruff prove that the ℓ_0-sampler in [20] succeeds with probability at least $1 - n^{-c}$ and, conditioned on no failure, outputs the item $i \in \operatorname{supp} a$ with probability $(1 \pm \epsilon)\frac{1}{N} \pm n^{-c}$. The space complexity is stated as $\text{poly}(1/\epsilon \log n)$ bits (in fact, the dependence on ϵ appears to be at most $\text{poly}(\log 1/\epsilon)$). Essentially the same result is shown in Theorem 2.1, where the dependency on $\log 1/\epsilon$ and $\log n$ is made explicit.

Pair-wise Independence. In [10], Frahling, Indyk and Sohler describe an ℓ_0-sampler using pairwise independence of $\mathcal{F}$. As above, items are mapped to levels with geometrically decreasing probabilities 2^j. Then at each level, items are further hashed to $1/\epsilon$ buckets, and information about one of these buckets is retained. However, this secondary hashing can be thought of as equivalent to mapping to a higher level $j + \log 1/\epsilon$, and attempting recovery at that level (a similar approach is described in [5]). The process succeeds if a unique item is recovered. The analysis of Lemma 2.1 does not apply directly for $k = 2$, but a modified analysis suffices to show that the probability of recovering a unique item at level $\frac{1}{\epsilon} \log N$ is ϵ, and conditioned on this event, the probability of choosing any item is $(1 \pm \epsilon)/N$. It appears that for high accuracy (small ϵ), the $O(1/\epsilon)$ repetitions required to obtain any sample with constant probability may drive the space cost of this approach higher than other approaches. We refer to this variant approach as "FIS", after the initials of the authors.

3 Experimental Evaluation

In this section we present an experimental analysis of the ℓ_0-sampling process described so far. The experimental objec-

[2]We note that tighter bounds are possible via a similar construction and a more involved analysis: adapting the approach of Eppstein and Goodrich [9] improves the log term from $\log(s/\delta_r)$ to $\log 1/\delta_r$, and the analysis of Price [22] further improves it to $\log_s 1/\delta_r$.

tives are threefold: (1) to study the accuracy of ℓ_0-sampling with k-wise independent hashing (2) to quantify the running time and space requirements of the data structures, and (3) to tune the parameter k in practical scenarios. First, we give some details on our implementation, benchmarks, and methodology.

Framework implementation. We implemented the ℓ_0-sampler in a C framework that allows us to compare different parameter settings and implementation choices. We set $s = 2k$, because our tests showed that setting s more than $2k$ does not help in practice.

We conducted two sets of experiments based on our assumptions about the s-sparse recovery data structure, used to recover items at a particular level j:

- For the purpose of testing different settings of k and the error due to the sampling step, we kept exact information on all items mapped to each level. This lets us simulate an "ideal" s-sparse recovery process which guarantees $\delta_r = 0$. However, we still consider a level to have "failed" if more than s items are mapped to this level by the sampling process. Here, we compare the two versions of the recovery level selection step: one assuming (exact) knowledge of N so we probe level $\lceil \log N/k \rceil$; and the other which tries each level in turn until an item can be sampled.
- Our final experiments implement the efficient s-sparse recovery mechanism described in Section 2.3. With this representation, the space required is linear in k, and we can update the sketch in constant time with a constant number of direct memory access. Although the analysis indicates that $\log s/\delta_r$ rows are sufficient to guarantee recovery with high probability, we found in our experiments that using 4 or 5 rows worked well enough in practice.

We measure the memory cost assuming that no space is used for "empty" levels (which have no items mapped to them), and so only account for the occupied levels. We return to this issue in Section 3.

Benchmarks. Tests were performed on sets of (fixed) vectors with 32 bit item identifiers (that is, $n = 2^{32}$) containing respectively 10^3, 10^4 and 10^5 non-zero items. The vectors are drawn randomly, and represented as streams containing N updates, one for each non-zero entry. We adopt a standard random number generator that can be seeded, invoked whenever random values are needed.

We measure the accuracy of the samplers based on how close the sample distribution is to uniform, based on multiple repetitions of the sampling process (over different random choices of the hash functions). Note that there will be some variation in this distribution simply due to the bounded number of repetitions: for example, to analyze the distribution of samples drawn from a vector containing N non-zero items, we need the number of samples to be at least of size N to have expectation of seeing any item once. In our experiments, we used $5 \cdot 10^5$ independent repetitions of the ℓ_0 samplers to test each setting. This is enough to see the overall trend in accuracy. To understand how well the different samplers are doing, we also compare to an "ideal" sampler, which samples items via a strong random number generator based on the true ℓ_0-distribution. We call this process "Balls-in-Bins" (BiB).

Platform. Running times were measured on a 2.8 GHz Intel Core i7 with 3 GB of main memory, running Debian 6.0.4, Linux Kernel 2.6.32, 32 bit.

3.1 Accuracy of ℓ_0-sampler

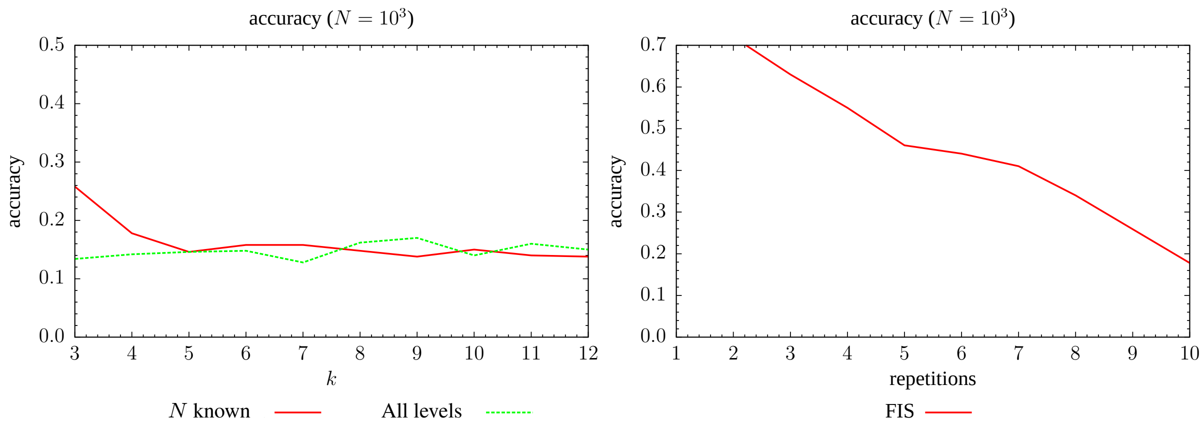
The results described in Section 2.2 show that the ℓ_0-sampler can achieve high success probability and small error on the ℓ_0-distribution, without requiring full randomness of $\mathcal{F}_k$. In many of the motivating scenarios in Section 1, the goal is to obtain an output distribution that is as close to uniform as possible[3].

After enough repetitions, we expect that the set of samples S contains approximately the same number of occurrences of each item and that, if an item i has been drawn more (or less) often, still the number $f(i)$ of its occurrences is close to the expected number according to the ℓ_0-distribution. We measure the maximum relative error of the obtained distribution as follows. Letting $f^* = \frac{|S|}{N}$, we define $\max_i \frac{|f(i)-f^*|}{f^*}$ to be the *accuracy* of the ℓ_0-sampler. We plot the evolution of this statistic as N increases, as this should converge to the desired distribution.

The theory predicts that choosing k proportional to (at least) $\log 1/\epsilon$ is needed in order to achieve ϵ-relative error. However, the constants of proportionality do not emerge clearly from the analysis. Moreover, we hope that in practice a moderate (constant) k will suffice, since the cost of evaluating the hash functions scales linearly with k, which determines the main overhead of the process. Hence we study the accuracy of the ℓ_0-sampler as k varies (thus effectively capturing the algorithms described by [20] and [16]). To this end, for each test vector, we compare the accuracy of the ℓ_0-sampler, as the number of samples increases against the corresponding accuracy of the uniform balls-in-bins process. We also compare to the variant ℓ_0-sampler proposed by Frahling, Indyk and Sohler [10] ("FIS") as discussed in Section 2.4 where we use 10 repetitions. This is implemented using the same codebase as the other samplers, but called with different parameter settings.

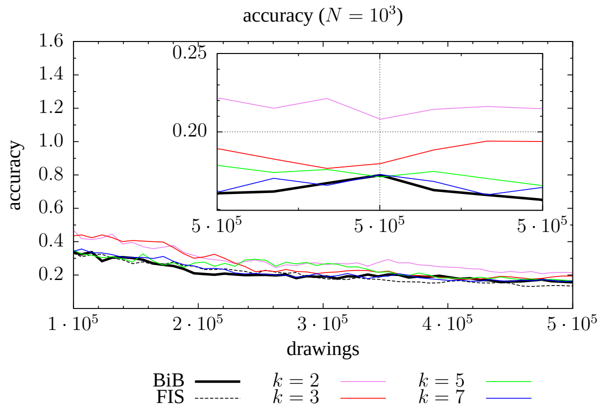
Sparse vectors. In Figure 3 we report the outcome of the experiment on 10^3 non-zero items, for different settings.

[3]However, in some applications, such as graph sketching [2] it is only important to draw some item from the support set $\operatorname{supp} a$: the exact distribution does not matter.



(a) Accuracy as k increases.

(b) Accuracy of FIS.



(c) Accuracy as number of repetitions increases.

Figure 3: Accuracy of ℓ_0-sampler with $N = 10^3$.

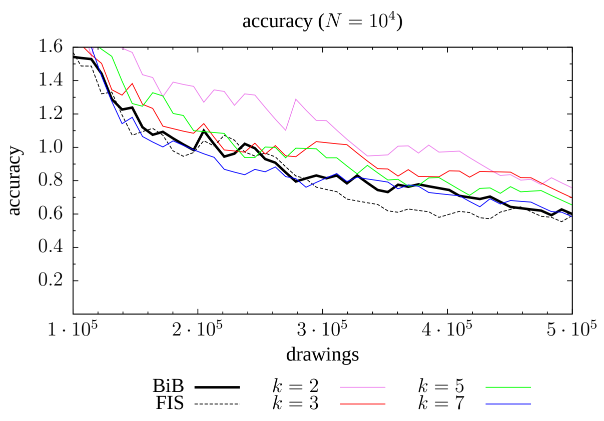
In Figure 3(a) we show how the accuracy for changes for different values of k of the k-wise independent hash functions for both versions of the level selection step (where N is known exactly, and where we try recovery at all levels until successful). Figure 3(b) shows the accuracy for the FIS variant with increasing number of repetitions. Figure 3(c) shows how the accuracy changes as the number of drawings increases. In particular, in Figure 3(a) we

According to this experiment, we get better accuracy as k increases, and the FIS variant achieves good accuracy with a small number of repetitions. Finally, as shown in the zoomed pane of in Figure 3(c), the experiment suggests that rather small values of k, for instance 7, are sufficient on average to achieve comparable accuracy to the fully random sampling process ("BiB", or *balls in bins* process).

Dense vectors. In Figure 4 we report the outcome of the same experiment of Figure 3(c), executed on denser vectors, containing respectively 10^4 and 10^5 non-zero items. Again, as predicted by the analysis, the accuracy increases as we increase the value of k. On the other hand, setting $k = 7$ is enough to observe that the error is close to 0.5 when sampling from a set of 10^4 non-zero items, and slightly bigger than 3 on the denser vector. On this data, the FIS variant also performs well, although it is the slowest of the methods compared on the plot, due to the number of parallel repetitions.

For the larger vectors ($N = 10^5$), the overall accuracy performance seems poorer: the relative error is around 3. However, this is mostly a function of the number of repetitions being insufficient to induce the true sampling distribution. We see that the accuracy obtained by the ideal "BiB" process is quite similar to the best of the ℓ_0-samplers, so we conclude that this variation is dominated by the random variation in the number of selected items.

3.2 Success probability of ℓ_0-sampler The main result in Section 2.4 shows that the single parameter s can control both the error on the output distribution and the failure probability. According to the experiments discussed in the previous section, high accuracy can be achieved with rather small values of the independence parameter k. However, when the goal is to draw any item from $\operatorname{supp} a$, it is important that the



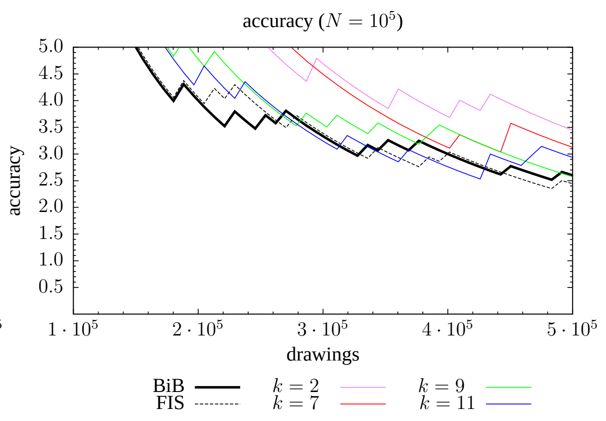


Figure 4: Accuracy as the number of drawings increases, with N up to 10^5.

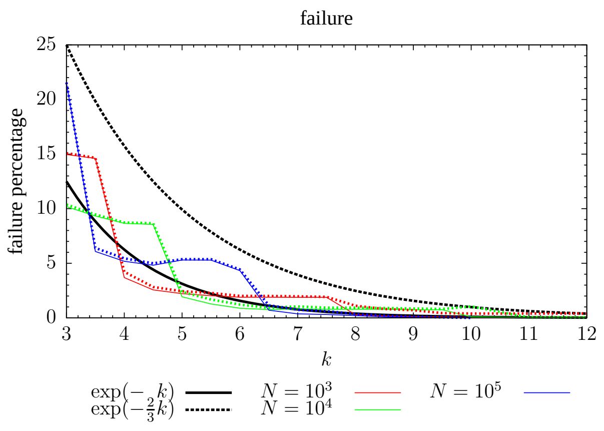


Figure 5: Failures as k increases, after 10^4 drawings.

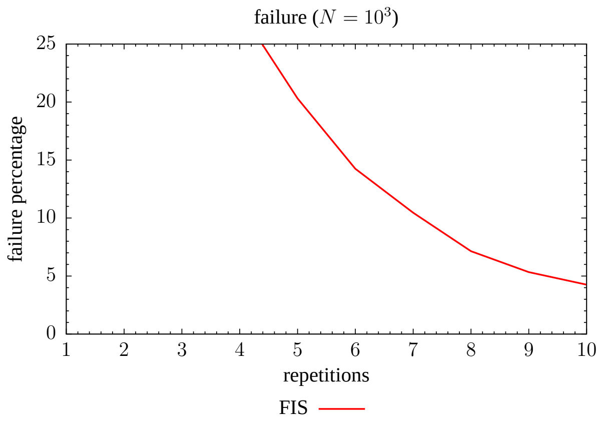


Figure 6: Failures of FIS, after 10^4 drawings.

number of failures of the ℓ_0-sampler is small. We recall that if the sampling process maps either 0 or more than s non-zero items to the level j chosen by the selection step, then the ℓ_0-sampler fails with probability 1. The second source of error is if the s-sparse recovery algorithm is unable to recover the vector, which can be made smaller by adjusting the space allocated to this algorithm. To this end, for each test vector, we compare the percentage of failed drawings in the "ideal" s-sparse recovery experimental setup, to what can be done using the s-sparse recovery mechanism described in Section 2.3.

Failure due to sampling step. In Figure 5 we show how the percentage of failures after 10^4 drawings, changes for different values of k. We also plot guidelines proportional to exponentially decaying probabilities, to compare to the analytical bound on the error (which is exponential, but with weaker constants). We observe that the error does decay quickly as k is increased. Moreover, we observed that it suffices to use small values of k, for instance $k = 7$, to ensure that the failure rate holds steady, independent of the number of repetitions made.

Figure 6 shows how accuracy changes for the FIS variant as the number of repetitions increases. On the vector containing 10^3 non-zero items, the error of the FIS variant becomes rapidly smaller than 5%, as we increase accordingly the number of repetitions. On the dense vectors, the FIS variant with 10 repetitions rarely fails to draw a sample, displaying approximately the same behavior as for $k = 4$, i.e. failure rate of below 5% for $N = 10^4$, and around 10% for $N = 10^5$.

Failure due to recovery step. Figure 5 also shows with a dashed line how the outcome of the experiment changes when our s-sparse recovery mechanism is used. We observe that the percentage of failures increases a little due to this, and that the gap decreases as we increase s. The theory predicts that with a sketch of size $2s \times \log s/\delta_r$, the failure probability of the s-sparse recovery algorithm is smaller than δ_r. From our experiments, we found that in fact using a constant number of rows ($\sim$ 5) sufficed to have a good probability of recovery. Since the cost of updating the

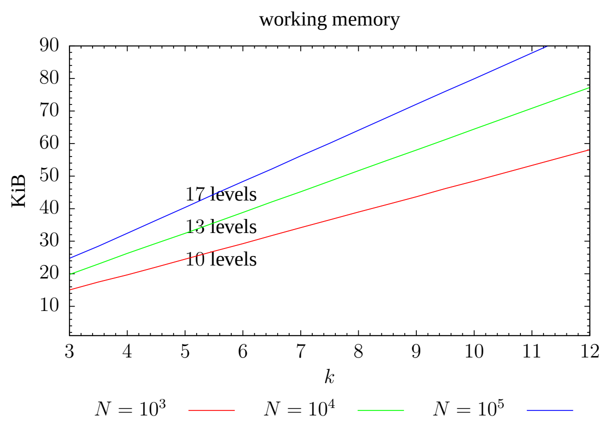


Figure 7: Working memory as k increases.

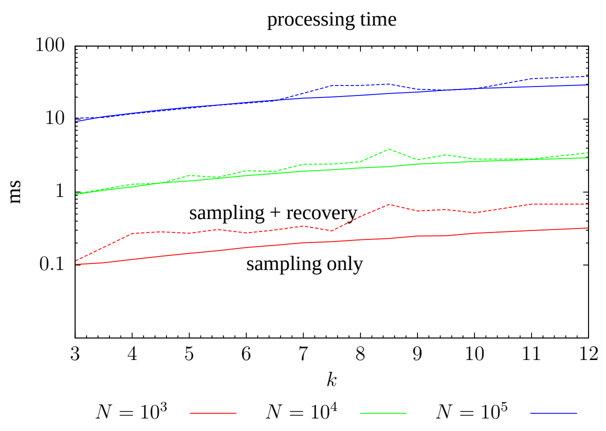


Figure 8: Processing time as k increases.

sketch scales linearly with the number of rows, using a small constant allows the execution of an update operation in constant time.

3.3 Efficiency of ℓ_0-sampler In the following we analyze the space requirements and running time of the ℓ_0-sampler.

Space usage. We evaluate the space required by the ℓ_0-sampler as a function of k, for each test vector. Figure 7 plots the space used in KiB (1KiB= 2^{10} bytes), to run a single instance of the ℓ_0-sampler. We used k 32-bit integers to represent $h \in \mathcal{F}_k$, 10 32-bit integers for $f_{1,\dots,5} \in \mathcal{G}_2$ and 2 32-bit integers plus 2 64-bit integers to implement each instance of $\mathcal{P}_1$. The size of a single level scales linearly with s independently from the input stream, while in the FIS variant each of the 10 repetitions is implemented with 2 32-bit integers to represent $h \in \mathcal{F}_2$ and a single instance of $\mathcal{P}_1$ per level. The space required for any of the samplers depends on the number of "non-empty" levels, which varies proportional to $\log N$, the ultimate number of distinct elements. If we know in advance good bounds on the final ℓ_0 norm, this suggests we can maintain a smaller number of levels.

Running time. Finally, we analyzed the impact of k on the performance of both sampling and recovery steps. We measure the total time to process the whole input stream, given as the average over 10^4 repetitions. As might be expected, the time is dominated by the sampling step: Figure 8 shows that the time required by the ℓ_0-sampler is barely affected by the s-sparse recovery algorithm. The time required by the FIS variant is constant, and is determined by the number of repetitions used to draw each sample. With 10 repetitions, the time cost is approximately equivalent to that for $k = 11$ in Figure 8.

The main effort using k-wise independence is in the evaluation of the selected hash function $h \in \mathcal{F}_k$. The overall processing time grows linearly as k (which is proportional to s) increases, proving the usefulness of an implementation based on the limited independence of $\mathcal{F}_k$. Figure 8, together with the analysis of accuracy and failure, suggests that there is a tradeoff between the reliability of the ℓ_0-sampler and its efficiency. When time is important, using $s \leq 12$ and $k \leq 6$ ensures fast computation. On the other hand, by selecting bigger values for both s and k, the process becomes slower than the FIS variant, while achieving similar failure rates and accuracy over our data. Hence in this regime, the usage of the FIS variant can result in better performance.

4 Concluding Remarks

The problem of drawing a sample from the ℓ_0-distribution has multiple applications over stream processing, computational geometry and graph processing. We have shown how existing algorithms fit into the framework of sampling, recovery and selection. Our experimental study shows that this framework can be instantiated effectively. Based on low-independence hash functions, we are able to draw samples close to uniform, and process millions of items per second. The space overhead is moderate, indicating that the algorithms are practical when a small number of samples are needed from a large amount of data. However, it is natural to ask whether the samplers can be made more space efficient, either by further engineering the subroutines and parameters, or by a fundamentally new approach to ℓ_0-sampling.

Acknowledgments. The work of Donatella Firmani was supported by a visitor grant from DIMACS, funded by NSF/CCF award number 1144502.

References

[1] D. Achlioptas, *Database-friendly random projections*, ACM Principles of Database Systems, 2001, pp. 274–281.

[2] Kook Jin Ahn, Sudipto Guha, and Andrew McGregor, *Analyzing graph structure via linear measurements*, ACM-SIAM Symposium on Discrete Algorithms, 2012.

[3] K. Beyer, R. Gemulla, P. J. Haas, B. Reinwald, and Y. Sismanis, *Distinct-value synopses for multiset operations*, Communications of the ACM **52** (2009), no. 10, 87–95.

[4] G. Cormode, F. Korn, S. Muthukrishnan, T. Johnson, O. Spatscheck, and D. Srivastava, *Holistic UDAFs at streaming speeds*, ACM SIGMOD International Conference on Management of Data, 2004, pp. 35–46.

[5] G. Cormode, S. Muthukrishnan, and I. Rozenbaum, *Summarizing and mining inverse distributions on data streams via dynamic inverse sampling*, International Conference on Very Large Data Bases, 2005.

[6] Graham Cormode, Minos Garofalakis, Peter Haas, and Chris Jermaine, *Synposes for massive data: Samples, histograms, wavelets and sketches*, now publishers, 2012.

[7] Graham Cormode and Mario Hadjieleftheriou, *Finding frequent items in data streams*, International Conference on Very Large Data Bases, 2008.

[8] S. Dasgupta and A. Gupta, *An elementary proof of the Johnson-Lindenstrauss lemma*, Tech. Report TR-99-006, International Computer Science Institute, Berkeley, 1999.

[9] D. Eppstein and M. T. Goodrich, *Space-efficient straggler identification in round-trip data streams via newton's identitities and invertible bloom filters*, WADS, 2007.

[10] Gereon Frahling, Piotr Indyk, and Christian Sohler, *Sampling in dynamic data streams and applications*, Proceedings of the twenty-first annual symposium on Computational geometry (New York, NY, USA), SCG '05, ACM, 2005, pp. 142–149.

[11] Sumit Ganguly, *Counting distinct items over update streams*, Theor. Comput. Sci. **378** (2007), no. 3, 211–222.

[12] Anna C. Gilbert, Martin J. Strauss, Joel A. Tropp, and Roman Vershynin, *One sketch for all: fast algorithms for compressed sensing*, ACM Symposium on Theory of Computing, 2007, pp. 237–246.

[13] P. Indyk and R. Motwani, *Approximate nearest neighbors: Towards removing the curse of dimensionality*, ACM Symposium on Theory of Computing, 1998, pp. 604–613.

[14] Piotr Indyk, *A small approximately min-wise independent family of hash functions*, Proceedings of the tenth annual ACM-SIAM symposium on Discrete algorithms (Philadelphia, PA, USA), SODA '99, Society for Industrial and Applied Mathematics, 1999, pp. 454–456.

[15] W.B. Johnson and J. Lindenstrauss, *Extensions of Lipshitz mapping into Hilbert space*, Contemporary Mathematics **26** (1984), 189–206.

[16] Hossein Jowhari, Mert Sağlam, and Gábor Tardos, *Tight bounds for lp samplers, finding duplicates in streams, and related problems*, Proceedings of the thirtieth ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems (New York, NY, USA), PODS '11, ACM, 2011, pp. 49–58.

[17] Daniel M. Kane, Jelani Nelson, and David P. Woodruff, *An optimal algorithm for the distinct elements problem*, ACM Principles of Database Systems, 2010, pp. 41–52.

[18] Nishad Manerikar and Themis Palpanas, *Frequent items in streaming data: An experimental evaluation of the state-of-the-art*, Data Knowl. Eng. **68** (2009), no. 4, 415–430.

[19] Ahmed Metwally, Divyakant Agrawal, and Amr El Abbadi, *Why go logarithmic if we can go linear?: Towards effective distinct counting of search traffic*, EDBT, 2008, pp. 618–629.

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

[21] Rob Pike, Sean Dorward, Robert Griesemer, and Sean Quinlan, *Interpreting the data: Parallel analysis with sawzall*, Dynamic Grids and Worldwide Computing **13** (2005), no. 4, 277–298.

[22] Eric Price, *Efficient sketches for the set query problem*, ACM-SIAM Symposium on Discrete Algorithms, 2011, pp. 41–56.

[23] Jeanette P. Schmidt, Alan Siegel, and Aravind Srinivasan, *Chernoff-hoeffding bounds for applications with limited independence*, Proceedings of the fourth annual ACM-SIAM Symposium on Discrete algorithms (Philadelphia, PA, USA), SODA '93, Society for Industrial and Applied Mathematics, 1993, pp. 331–340.