Distrib Parallel Databases (2014) 32:315–335
DOI 10.1007/s10619-013-7131-9

A unifying framework for ℓ_0-sampling algorithms

Graham Cormode · Donatella Firmani

Published online: 25 July 2013

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.

Keywords Data summarization · Sampling · Distinct elements · Graph sketches

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 [17] allow the Euclidean distance between high dimensional vectors to be approximated accurately via much lower-dimensional projections [1, 10, 16]. Many constructions in the new area of compressed sensing can also be expressed as

Communicated by: Feifei Li and Suman Nath.

This paper is an extended version of [5].

G. Cormode (✉)
University of Warwick, Coventry, UK
e-mail: G.Cormode@warwick.ac.uk

D. Firmani
Sapienza University of Rome, Rome, Italy
e-mail: firmani@dis.uniroma1.it

sketches [14]. 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 [7], log analysis [25] and approximate query processing [9].

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 [6, 20], and for tracking the cardinality of sets of items [4, 21].

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 [8]. 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 [12]. 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. Simple solutions, such as sampling a subset of items from the input stream, are not sufficient, as the sampled items may end up with zero weight. Instead, more complex algorithms have been employed to ensure that information is retained on items with non-zero weight. Early constructions made use of universal hash functions [8, 12]. Stronger results were shown using higher-independence hash functions [22], and most recently assuming fully-independent hash functions [18]. 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 influence 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 Sect. 1.1. In Sect. 2 we give a canonical 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 prior work [12, 18, 22]. We present our experimental comparison of methods in Sect. 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 $\|a\|_p = (\sum_{i=1}^n |a_i|^p)^{1/p}$ 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

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

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

[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.

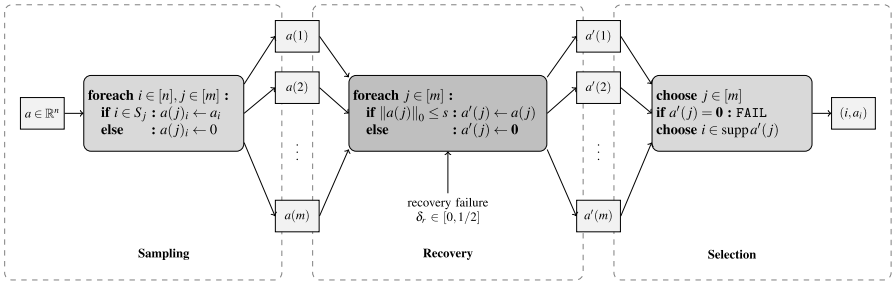


Fig. 1 Overall ℓ_0-sampling process, $m = O(\log n)$

schematically in Fig. 1. We outline these steps below, and progressively fill in more details of how they can be implemented through the course of the paper.

1. SAMPLING. The purpose of the sampling step is to define a set of subvectors of the input a, so that one of them is "sparse enough" to be able to recover accurately from the stored sketch. 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 goal of the recovery step is to try to recover each of the sampled vectors: one of these should succeed. 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.

Putting these pieces together, the process succeeds if the selection step chooses a level for recovery at which the sampled vector is sufficiently sparse (not too dense, and not zero). If the vector from this level is recovered correctly, then an item can be sampled from it. The intuition for this process is that the (space) cost of sparse recovery algorithms grows with the sparsity s of the vector that is to be recovered, hence we want to use this only for a small value of the parameter s.

As mentioned above, existing ℓ_0-samplers [12, 18, 22] 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 Sect. 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.

Note that although this process is described over the whole vector a, it also applies when the vector is described incrementally as a sequence of updates, since we can apply the process to each update in turn, and propagate the changes to the data structures. Moreover, the process meets our definition of a sketch since, given the data structures for two input vectors a and b based on the same hash function h, we can compute the summary for $(a + b)$ by merging each of the corresponding recovery data structures.

To aid in the analysis, 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 first part of the analysis, let $\mathcal{P}_s$ be a perfect s-sparse recovery algorithm: that is, an algorithm which can recover any vector x with $\|x\|_0 \leq s$, and otherwise outputs FAIL. Let $\mathcal{S}$ denote the above selection step algorithm using $\mathcal{P}_s$.

Lemma 1 (Probability of successful recovery) *Given a k-wise independent family $\mathcal{F}_k$, with $k \geq s/2$ 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 $1 \leq \|a(j)\|_0 \leq s$: (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 $1 \leq \|a(j)\|_0 \leq s$ 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 where we expect to see a number of items mapped that is a constant fraction of s, so that

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

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

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

We invoke [27, 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 $r \geq 1$ and $k = \lceil r\mathbf{E}[X]\rceil$:

$$\mathbf{Pr}\big[\big|X - \mathbf{E}[X]\big| \geq r\mathbf{E}[X]\big] \leq \exp\big(-\mathbf{E}[X]r/3\big)$$

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

$$\mathbf{Pr}\big[\big|N_j - \mathbf{E}[N_j]\big| \geq \mathbf{E}[N_j]\big] \leq \exp(-s/12) \leq \delta_t \quad \text{if } s \geq 12\ln\frac{1}{\delta_t}$$

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:

$$\mathbf{Pr}[\text{output}_{\mathcal{S}} \in \operatorname{supp} a] \geq \mathbf{Pr}[1 \leq N_j \leq s] \geq 1 - \delta_t \qquad \square$$

Lemma 2 (Output distribution of ℓ_0-sampler) *Let $1 - \delta$ be the success probability of the ℓ_0-sampler* (*from Lemma 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 [15]. 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 $(1 \pm \exp(-s))\frac{1}{N} \pm \delta$. $\square$

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 [19]. This is well-principled, but adds an overhead to the process. We refer to this as "fixed-level recovery". An alternative is to aggressively attempt to recover vectors, and sample from the first level that succeeds. We refer to this as "greedy-level recovery". We compare these alternatives empirically in Sect. 3.

2.3 Sparse recovery

The problem of 'sparse recovery' arises in a number of contexts, including compressed sensing. The central problem is to design schemes that can operate on an input vector, so that from a small amount of stored information a good, sparse representation of the input vector can be extracted as output. The general s-sparse recovery problem is to design a sketch, such that for any vector a we can efficiently recover an s-sparse vector a' satisfying $\|a - a'\|_p \leq C \min_{s\text{-sparse } a'} \|a - a'\|_q$ for some norm parameters p and q and an approximation factor $C = C(s)$. Essentially, a' provides a good approximation of a. We consider the *exact* version of this problem, defined for truly s-sparse vectors (i.e., having at most s non-zero components). Here, the goal is to output $a' \in \mathbb{R}^n$ or `FAIL`, such that for any s-sparse vector a the output is $a' = a$ with high probability, otherwise the output is `FAIL`. For more information and background, see the work of Price [26].

In this section, we discuss how to implement an efficient exact s-sparse recovery algorithm. Many approaches have been made to this question, due to its connection to problems in coding theory and compressed sensing. [13] 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. [22] describe a sketch-based solution 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. The approach echoes similar techniques used for sparse recovery [13, 26] and "straggler identification" [11], and is presented here for completeness.

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:

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

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 [13]

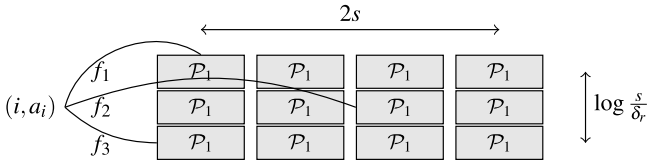


Fig. 2 Exact s-sparse recovery with perfect 1-sparse recovery

is to additionally compute $\tau = \sum_{i \in \text{supp}\, a} a_i \cdot i^2$, 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 $\tau = \sum_{i \in \text{supp}\, a} a_i \cdot z^i \mod p$, and test if $\tau = \phi \cdot z^{\iota/\phi} \mod p$.

Lemma 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 $p(z) = \sum_{i \in \text{supp}\, a} a_i \cdot z^i - \phi \cdot z^{\iota/\phi}$.

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 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 Sect. 2.3.1, while $\mathcal{R}_s$ is our exact s-sparse recovery algorithm. Similar to [13], 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 Fig. 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 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 $\mathbf{Pr}[\text{rec}_i]$ 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 $C_i = \sum_{l \neq i, l \in \text{supp}\, a} c_l$, 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 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]

This sparse recovery structure meets our definition of a sketch: we can combine the structures of two vectors to obtain a recovery structure for their sum (assuming that they are built using the same parameters and hash functions). This is because the innermost $\mathcal{P}_1$ structures are linear functions of their input: we can sum up each of the variables in $\mathcal{P}_1$ to obtain the summary for the sum of the inputs.

2.4 Main result

Replacing the perfect s-sparse recovery procedure assumed in Sect. 2.2 with the above procedure affects the output distribution by at most an additive δ_r. Hence, combining Lemmas 1, 2, and 4, we obtain:

Theorem 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 $i \in \text{supp}\, a$ 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:

[2] We note that tighter bounds are possible via a similar construction and a more involved analysis: adapting the approach of [11] improves the log term from $\log(s/\delta_r)$ to $\log 1/\delta_r$, and the analysis of [26] further improves it to $\log_s 1/\delta_r$.

Corollary 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.*

From the lower bounds perspective, Jowhari et al. [18] have shown that $\Omega(\log^2 n)$ bits of space are needed to solve this problem with constant probability of failure and constant distortion.

2.5 Comparison with previous results

We now compare the ℓ_0-sampler analyzed thus far to those described in prior work, and show that this construction captures existing ℓ_0-samplers as special cases. To this end, we first describe the prior algorithms in terms of the three-step process of SAMPLING, RECOVERY and SELECTION presented in Sect. 2. In fact, these algorithms can be thought of as ℓ_0-samplers with k-wise independent hashing, differing in the strength of the hash functions used and the nature of the recovery data structures adopted. We then outline how our analyses apply to prior results assuming full and limited independence of $\mathcal{F}_k$ and can be extended to include pairwise independence.

2.5.1 Full independence

The analysis of the ℓ_0-sampler due to [18] assumes full independence of $\mathcal{F}_k$. In fact, the sampling process defines $m = \lfloor \log n \rfloor$ vectors each containing a subset S_j of the coordinates of the input vector, which is chosen uniformly at random.[3] The recovery piece is implemented with a perfect s-sparse recovery data structure, and the selection of the level for recovery is greedy: the algorithm returns a uniform random non-zero coordinate from the first successful recovery that gives a non-zero s-sparse vector, if it exists, otherwise the algorithm fails.

Jowhari et al. [18] prove that their ℓ_0-sampler 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.

2.5.2 Limited independence

The analysis of the ℓ_0-sampler due to [22] assumes $c \log n/\epsilon$-wise independence of $\mathcal{F}_k$, with $c > 1$, to get ϵ-min-wise independence over N elements [15]. The recovery step is implemented with an exact s-sparse recovery algorithm, with $\delta_r = n^{-c}$. Monemizadeh and Woodruff [22] prove that their ℓ_0-sampler 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 $\operatorname{poly}(1/\epsilon \log n)$ bits. In fact, the dependence on ϵ appears to be at most $\operatorname{poly}(\log 1/\epsilon)$. Essentially the same

[3] Jowhari et al. [18] first present their algorithm assuming a random oracle, and then they remove this assumption through the use of the pseudo-random generator of Nisan [23].

result is shown in Theorem 1, where the dependency on $\log 1/\epsilon$ and $\log n$ is made explicit.

A very recent approach that uses limited independence is described by [3]. This also fits within the framework we provide. The authors use $\Theta(\log 1/\delta)$-wise independence of $\mathcal{F}_k$, and the recovery step is implemented with an exact s-sparse recovery algorithm, with $\delta_r = \delta$ and $s = \Omega(\log 1/\delta)$. Barkay et al. [3] use this sampler to return a larger sample of size s with probability at least $1 - \delta$. They show that each subset of k items in supp a has equal probability to be in the sample. Barkay et al. [3] also discuss reducing the space of the s-sparse recovery step, by tolerating approximate recovery which fails to recover a fraction of the items. This reduces the number of rows in the sparse recovery structure to a constant.

Sampling multiple items Equivalently, the approach by [3] can be seen as saying that a *good* sample of size $O(s)$ can be drawn if we return the full set of items recovered, instead of just one. This can be more space and time efficient than repeating the sampling process independently to draw one item at a time. The analysis of Lemma 2 does not apply for larger samples of size s, however it is natural to ask whether these samples can be drawn out from a uniform distribution. This is confirmed by the analysis provided by [3], which shows that recovering a k-wise independent set of s items, with $k = \Omega(\log 1/\delta)$ and $s = \Omega(1/\epsilon^2 \log 1/\delta)$, suffices to achieve an additive ϵ-approximation to the inverse distribution with probability at least $1 - \delta$.

2.5.3 Pairwise independence

Frahling et al. [12] describe an ℓ_0-sampler using pairwise independence of $\mathcal{F}_k$. As above, items are mapped to levels with geometrically decreasing probabilities 2^{-j}. Then at each level, items are further hashed to $2/\epsilon$ buckets, and a 1-sparse recovery structure is maintained for only the first of these buckets. This secondary hashing is equivalent to mapping to a higher level $j + \log 1/\epsilon$. Recovery is attempted at a high level where the expected number of items is approximately ϵ. The process succeeds if a unique item is recovered, with at least constant probability. A similar approach is described by [8], where items are mapped to levels with geometrically decreasing probabilities, determined by a strong universal hash function.

The analyses of Lemmas 1 and 2 do not apply directly for fixed $k = 2$, so we outline the differences.

This ℓ_0-sampler achieves success probability proportional to ϵ, and conditioned on this event, the probability of choosing any item is $(1 \pm \epsilon)/N$.

Consider the level $j = \lceil \log(N/\epsilon) \rceil$. The probability that the sampler captures any fixed item in the recovery structure at level j is $2^{-(j+1)}$, and the probability that no other item is also mapped there is at least $\epsilon/2$, by the Markov inequality, as this is the expected number of items at this level. The probability of success of the ℓ_0 sampler, i.e. the probability that any item is returned by the sampler, is then at least $\epsilon/4$, by summing over the $\epsilon 2^j \geq N > \epsilon 2^{j-1}$ different items with non-zero frequency. Note that this success probability is quite low: to ensure δ_t probability of failure, we need to take $O(1/\epsilon \log 1/\delta_t)$ repetitions of this sampler.

To show the uniformity of the sampling process, we cannot rely on min-wise hashing, since we use only pairwise independent hash functions. However, Frahling et

al. [12] show that the probability of sampling each item is approximately the same, up to a $(1+\epsilon)$ factor. This is sufficient to argue that i is output with probability $(1 \pm \epsilon)\frac{1}{N} \pm \delta$.

Combining this with the $\mathcal{P}_1$ procedure for 1-sparse recovery, the space complexity of the resulting ℓ_0-sampler is $O(1/\epsilon \log^2 n \log 1/\delta)$ bits. For high accuracy (small ϵ), the space is dominated by the $O(1/\epsilon)$ repetitions required to obtain any sample with constant probability.

3 Experimental evaluation

In this section we present an experimental analysis of the ℓ_0-sampling process described so far. The experimental objectives are several: (1) to analyze the output distribution, (2) to study the probability of successful recovery of an ℓ_0-sampling algorithm, (3) to quantify the running time and space requirements of the used data structures, and (4) to tune the parameters in practical scenarios. First, we give some details on our implementation, benchmarks, and methodology.

3.1 Framework implementation

We implemented the ℓ_0-sampling process in a C framework that allows us to compare different parameter settings and implementation choices. We adopt a standard random number generator that can be seeded, invoked whenever random values are needed.

The implementation of ℓ_0-sampling algorithm with k-wise independence is parametrized by the value of k (thus effectively capturing the algorithms described by [22] and [18]), while its variant with pairwise independence is parametrized by the number r of independent repetitions executed to draw a sample (capturing the algorithms described by [12] and [8]). We now give some details on our implementation of the three steps of the ℓ_0-sampling process.

1. SAMPLING. We used k 128-bit integers to represent $h \in \mathcal{F}_k$ as a polynomial hash function over a large prime field. With this representation, both the space required and the computation time are linear in k.
2. RECOVERY. We implement the sparse recovery mechanism described in Sect. 2.3. We found in our experiments that using 3 or 4 rows was sufficient to guarantee recovery with high probability. 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 accesses.
3. SELECTION. For each setting of the ℓ_0-sampling process, we compare two different implementations of the recovery level selection step, as mentioned in Sect. 2.2. Fixed-level recovery uses (exact) knowledge of N, where we probe the level at which recovery is most likely to succeed.[4] Greedy-level recovery attempts to recover a vector at each level in turn, until the procedure is successful (or all levels fail to return a vector).

[4] This level is $\lceil \log(2N/k) \rceil$ for the ℓ_0-sampler with k-wise independence, and $\lceil \log N/\epsilon \rceil$ for the variant with pairwise independence.

From the above discussion, it follows that the space required by the ℓ_0-sampling algorithm with k-wise independence consists of the space used for a k-sparse recovery data structure at each level, plus the k integers to describe the hash function. Similarly, the space required by a single repetition of the ℓ_0-sampler with pairwise independence is given by the space used for the 1-sparse recovery data structures, plus the constant space required to represent the hash function.

3.2 Benchmarks

Tests were performed on sets of (fixed) vectors with 32 bit item identifiers (that is, $n = 2^{32}$). We restrict our focus to two instances, both containing $N = 10^3$ non-zero items. In the first, the items were allocated uniformly across the whole domain (the *uniform input*), while in the second they were distributed according to an exponential distribution (the *exponential input*). Additional experiments in a prior presentation of this work showed that similar results are achieved with denser vectors [5]. The vectors are represented as streams containing N updates, one for each non-zero entry: the format or ordering of the input does not affect the accuracy of the recovery, due to the linear nature of the ℓ_0-sampling algorithms.

We study both the error on the output distribution and failure rate of the ℓ_0-sampler, as the parameters k or the number of repetitions r (for the sampler using pairwise independence) vary. Our tests consist of: (1) 10 different instances of the ℓ_0-sampler with k-wise independence, corresponding to values of k in the range $[3, 12]$; and (2) 10 different instances of the ℓ_0-sampler with pairwise independence, corresponding to 10 different values of r in the range $[5, 50]$. If we fix the space usage of the two approaches, then we find that there is roughly a factor of 5 between the two: $k = 3$ corresponds to the space used by $r = 15$ (which is approximately 7.5 KB), $k = 4$ corresponds to $r = 20$ ($\approx$10 KB), and so on.

We performed 10^6 independent executions of the ℓ_0-samplers, over different random choices of the hash functions, to test each setting. This is enough to see the overall trend in both error on the output distribution and failure rate. The error on the output distribution depends on how many repetitions are made, and there will be some variation in this distribution simply due to the bounded number of drawings. In particular, to analyze the distribution of samples drawn from a vector containing N non-zero items, we need the number of drawings to be at least of size N to expect to see any item even once. To understand how well the different samplers are doing, we 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.

3.3 Accuracy of ℓ_0-sampler

The results described in Sect. 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 Sect. 1, the goal is to obtain an output distribution that is as close to uniform as possible.

After a large number of repetitions, we expect that the set of samples S contains approximately the same number of occurrences of each item. Of course, there will be some variation from the mean, but the number $f(i)$ of occurrences of item i should be close to the expected number $f^*(i)$ according to the uniform ℓ_0-distribution. We want to measure the *accuracy* of ℓ_0-sampling, by looking at the proximity of the obtained distribution to the uniform distribution. To this end, we use the metrics below:

- **Standard deviation.** We measure the standard deviation of the observed sampling distribution, normalized by the target probability $f^* = \frac{|S|}{N}$.
- **Maximum deviation.** Analogously, we define the maximum deviation as $\max_i \frac{|f(i)-f^*|}{f^*}$.

We plot the evolution of these statistics as functions of the space used or the number of drawings. We also compare the accuracy of each setting of the ℓ_0-sampler to the corresponding accuracy of the uniform balls-in-bins process.

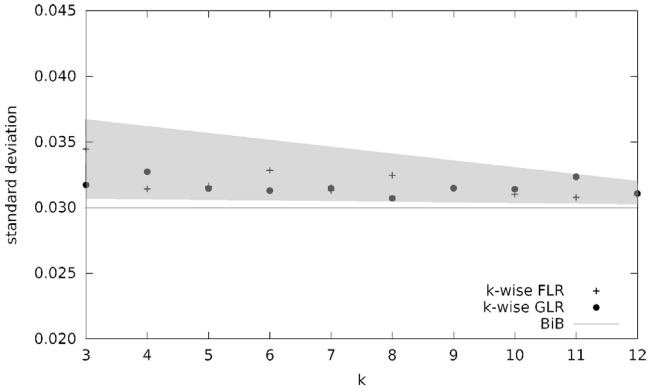
Accuracy as the space increases In Figs. 3 and 4 we show how the accuracy changes after one million drawings for different settings. In particular, we plot the standard deviation in Fig. 3 and the maximum deviation of the obtained distribution in Fig. 4.

Figure 3(a) shows the standard deviation of the sampler using k-wise independence for different settings of k, while Fig. 3(b) shows the results for the pairwise independence sampler as the number of repetitions r is varied. The corresponding results for maximum deviation are shown in Figs. 4(a) and 4(b). We compare these accuracy measures for performing fixed-level recovery (FLR) and greedy-level recovery (GLR), and contrast to the corresponding measure for the BiB process. We use gray shading on the plot to indicate the region occupied by the experimental results.

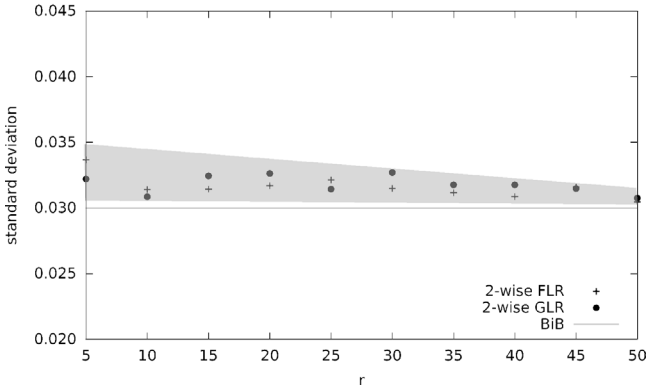
As predicted by the analysis, the results show that we get better accuracy as either k or r increases. The theory predicts that choosing k proportional to (at least) $\log 1/\epsilon$ is needed in order to achieve ϵ-relative error. However, the constants of proportionality do not emerge clearly from 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. We observe that, indeed, moderate values of k are sufficient on average to achieve comparable accuracy to the fully random sampling process, BiB.

For the shown values of k and r, the resulting standard deviation indicates that the occurrences in the set of samples S tend to be very close to those given by BiB. In addition, over one million repetitions, each setting achieves approximately the same maximum deviation as the ideal sampling process, BiB. That suggests that the sampling process is almost indistinguishable from a uniform sampling. The experiment on exponential input (not shown) has a similar outcome: increasing space yields better accuracy.

Accuracy as the number of drawings increases In Figs. 5(a) and 5(b), we analyze the accuracy as the number of drawings increases for the ℓ_0-samplers with least space usage.



(a) Standard deviation for k-wise independence



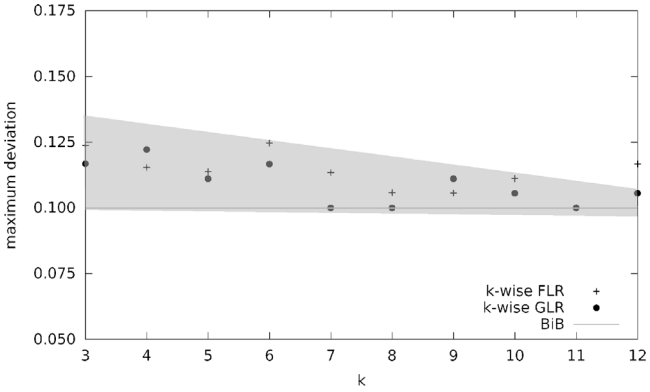
(b) Standard deviation for pairwise independence

Fig. 3 Standard deviation of ℓ_0-sampler as space increases on uniform input

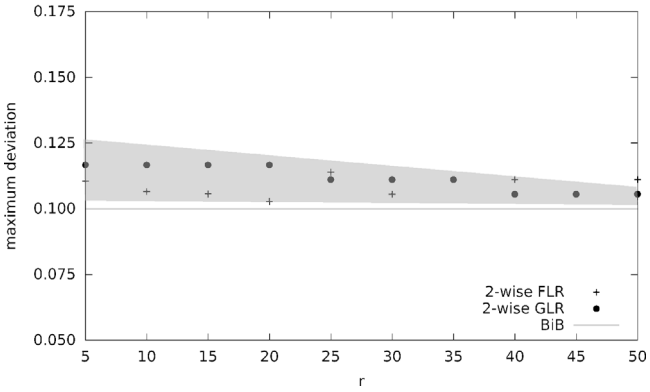
Figure 5(a) shows the standard deviation for the different samplers on the uniform input, as the number of independent drawings increases. Figure 5(b) shows the corresponding results for the exponential vector. In both cases, we show the results for the different level selection steps, FLR and GLR, and contrast to the BiB process. The results show that in fact greedy-level recovery provides greater accuracy than its fixed-level recovery counterpart—comparable to that of the BiB process. This indicates that greedy-level recovery is a good implementation choice, particularly since it does not require estimation of the number of distinct elements. Over uniform input, pairwise independence and 3-wise do not differ greatly (under GLR), but pairwise independence appears to slightly outperform the 3-wise independent sampler.

3.4 Failure rate of ℓ_0-sampler

The results in Sect. 2 show that the single parameter s (and hence k) for the ℓ_0-sampler with k-wise independence, and r for its variant with pairwise independence, can control both the error on the output distribution and the failure probabil-



(a) Maximum deviation for k-wise independence



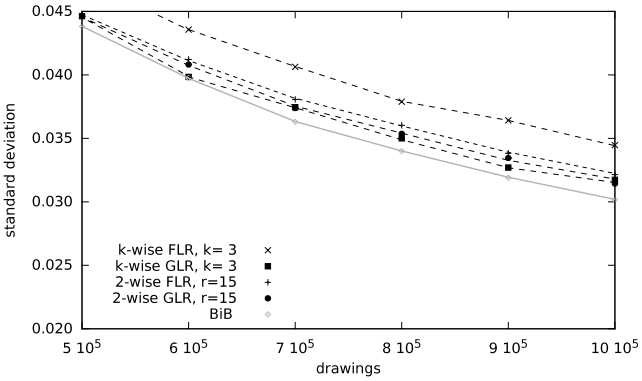
(b) Maximum deviation for pairwise independence

Fig. 4 Maximum deviation of ℓ_0-sampler as space increases, on uniform input

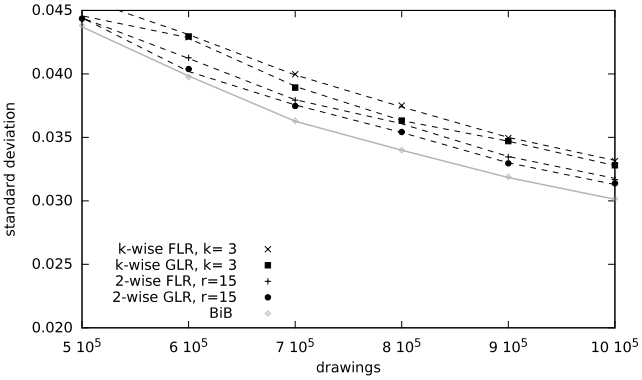
ity. According to the experiments shown in the previous section, high accuracy can be achieved with moderate values of k and r. However, when the goal is to draw any item from $\operatorname{supp} a$, it is important that the number of failures of the ℓ_0-sampler is also small.

We recall that if the sampling process maps either too few (0) or too many (more than s) non-zero items to the level j chosen by the selection step, then the ℓ_0-sampler will fail. The second source of failure 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. We observed that the failure rate is mainly due to the first source, as noted previously [5]. Nevertheless, the sparse recovery does have some impact on this rate.

We computed the percentage of failed drawings as both the space used and the number of drawings increase. Our general finding was that most failures are due to too many or too few items mapped to level j, rather than failure of the recovery step. We next show in more detail the percentage of failed drawings of each setting of the ℓ_0-sampler as the space increases, for different versions of the level selection step.



(a) Standard deviation for uniform input

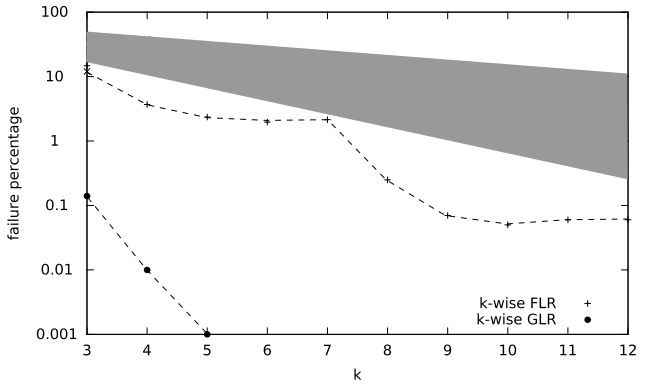


(b) Standard deviation for exponential input

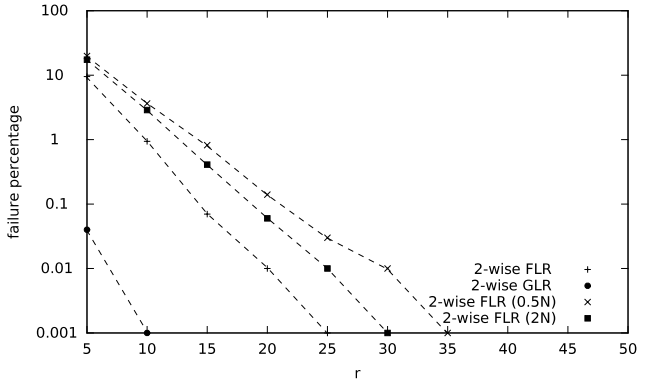
Fig. 5 Accuracy of ℓ_0-sampler, for uniform and exponential inputs

Failure rate as the space increases In Fig. 6 we show how the percentage of failed drawings changes, over one million repetitions, for different settings. In Fig. 6(a) we show how the percentage of failed drawings of the ℓ_0-sampler with k-wise independence changes for different values of k, for both versions of the recovery selection step. We observe that the error decays quickly as k is increased, as theory predicts, except for some random variations. When we recover at a fixed-level based on N, it suffices to use small values of k, for instance $k = 8$, to ensure that the failure percentage is smaller than 1 %, independent of the number of drawings. We indicate with a gray band the range of failures observed when we attempt fixed-level recovery with an approximation of the level $j = \lceil \log(2N/k) \rceil$, up to a factor of two. This shows that an accurate estimate of N is important: the failure rate increases quite notably when recovering at the "wrong" fixed-level. Meanwhile, under greedy-level recovery, this method rarely fails to draw a sample.

In Fig. 6(b) we show the percentage of failed drawings for the pairwise independence case with increasing number of repetitions. Here, we also show the impact of performing recovery at the "wrong" fixed level. We recover based on an estimate N' of N, such as $N' = N/2$ and $N' = 2N$, corresponding to levels $j - 1$ and $j + 1$. The



(a) Failure rate for k-wise independence



(b) Failure rate for pairwise independence

Fig. 6 Failure percentage of ℓ_0-sampler as space increases, for the uniform vector

outcome of the experiment suggests that the percentage of failed drawings becomes rapidly smaller than 5 % as we increase the number of repetitions, displaying approximately the same behavior as for $k = 4$. An adequate success rate can be achieved even if N is not known exactly, but is estimated up to a factor 2, although this is clearly weaker than when N is known accurately.

Experiments on the exponential input showed the same trend, that is, increasing space yields lower failure rate.

3.5 Efficiency of ℓ_0-sampling

In the following we consider the impact of the choice of hash function on the update and recovery time. We discuss alternate choices of hash functions, and measure the total time to update our data structure upon the read of a single item, given as the average over the whole stream; we also measure the time to recover an item, given as the average over one million drawings. Motivated by some applications, such as graph sketching [2], where the only goal is to draw some item from the support set $\operatorname{supp} a$.

We also discuss which of the settings discussed so far works best when accuracy does not matter, only recovery probability.

Alternate Hash functions Throughout, we have been using k-wise independent hash functions within the ℓ_0-samplers, based on randomly chosen polynomials of degree k. We discuss some other choices that are possible. Recently, the model of *tabulation hashing* has been advocated [24]. Here, the key to be hashed is broken into a small number of characters, so that each character is hashed independently via a table of randomly chosen values, and these values combined to give the final hash value. Although this approach has only limited independence, it can be shown to nevertheless provide approximate min-wise independence, as needed by our samplers.

We performed some experiments using this hash function, based on splitting keys into four characters. Experiments confirm that putting in tabulation hashing into our samplers gives similar accuracy to $k = 5$. Tabulation hashing can be used for the hash function in the pairwise algorithm, or the one expecting a k-wise independent hash function, without yielding a significant change in accuracy.

We show the running time cost of using this hashing scheme in the subsequent plots. We denote with "$\mathcal{A}$ TH" the variant of the ℓ_0-sampler algorithm $\mathcal{A}$ using this hashing scheme.

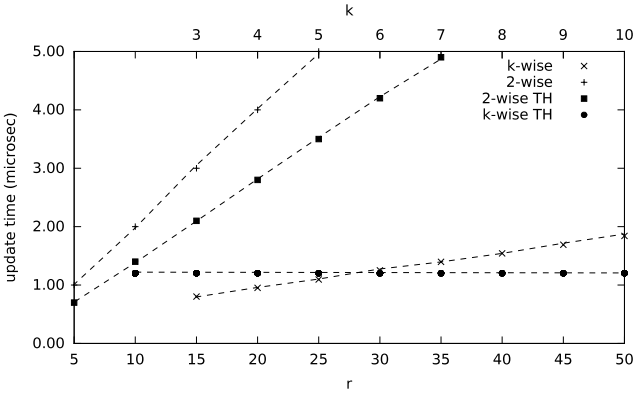
Running time as the space increases As shown in Fig. 7(a), in general, the time to perform each update is very small. Here, we match up k and r values for corresponding amounts of space used: $k = 3$ corresponds to $r = 15$, and so on. The main effort using k-wise independence is in the evaluation of the selected hash function $h \in \mathcal{F}_k$. The update time grows linearly as k increases, proving the importance of an implementation based on the limited independence of $\mathcal{F}_k$. Figure 7(a), together with the analysis of accuracy and failure, suggests that there is a trade-off between the reliability of the ℓ_0-sampler and its efficiency. The update time of the tabulation hashing (TH) variant is constant as a function of k, and is faster beyond $k = 6$. In the pairwise case, using a simple pairwise hash function is faster than tabulation hashing.

Figure 7(b) shows the corresponding time to perform recovery. In the pairwise independence case it is very fast to recover, since only a single $\mathcal{P}_1$ structure has to be probed in each repetition. The greedy-level recovery is naturally slower than fixed-level recovery, since more levels have to be inspected until a vector is recovered. Note that the recovery time does not depend on the hash function used.

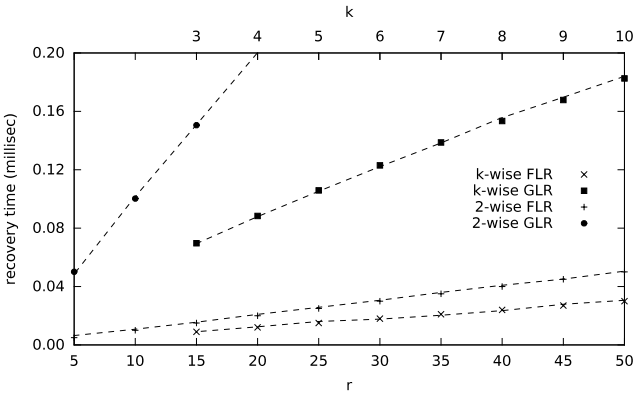
Arbitrary sampling If only the failure rate of the ℓ_0-sampler is important, we want to minimize the space required by the ℓ_0-sampler while drawing *some* sampled item from the input.

As shown in Sect. 3.4, for a given failure rate, the ℓ_0-sampler with pairwise independence requires less space than its variant with k-wise independence, and ensures quicker recovery, especially when greedy-level recovery is used.

However, when update time is important, using the variant with k-wise independence may yield faster computation. If the desired failure rate is small, for example smaller than 1 %, the ℓ_0-sampler with pair-wise independence may become too



(a) Update time



(b) Recovery time

Fig. 7 Running time of ℓ_0-sampler as space increases

slow with respect to the k-wise variant that achieves similar failure rate over our data ($k = 8$). Hence in this regime, the usage of the ℓ_0-sampler with k-wise independence 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.

References

1. Achlioptas, D.: Database-friendly random projections. In: ACM Principles of Database Systems, pp. 274–281 (2001)
2. Ahn, K.J., Guha, S., McGregor, A.: Analyzing graph structure via linear measurements. In: ACM-SIAM Symposium on Discrete Algorithms, pp. 459–467 (2012)
3. Barkay, N., Porat, E., Shalem, B.: Feasible Sampling of Non-strict Turnstile Data Streams (2012). arXiv:1209.5566
4. Beyer, K., Gemulla, R., Haas, P.J., Reinwald, B., Sismanis, Y.: Distinct-value synopses for multiset operations. Commun. ACM **52**(10), 87–95 (2009)
5. Cormode, G., Firmani, D.: On unifying the space of ℓ_0 sampling algorithms. In: Meeting on Algorithm Engineering & Experiments, pp. 163–172 (2013)
6. Cormode, G., Hadjieleftheriou, M.: Finding frequent items in data streams. In: International Conference on Very Large Data Bases, pp. 3–20 (2008)
7. Cormode, G., Korn, F., Muthukrishnan, S., Johnson, T., Spatscheck, O., Srivastava, D.: Holistic UDAFs at streaming speeds. In: ACM SIGMOD International Conference on Management of Data, pp. 35–46 (2004)
8. Cormode, G., Muthukrishnan, S., Rozenbaum, I.: Summarizing and mining inverse distributions on data streams via dynamic inverse sampling. In: International Conference on Very Large Data Bases, pp. 25–36 (2005)
9. Cormode, G., Garofalakis, M., Haas, P., Jermaine, C.: Synposes for Massive Data: Samples, Histograms, Wavelets and Sketches. Now Publishers, Hanover (2012)
10. Dasgupta, S., Gupta, A.: An Elementary Proof of the Johnson–Lindenstrauss Lemma. International Computer Science Institute, Berkeley (1999). Tech. Rep. TR-99-006
11. Eppstein, D., Goodrich, M.T.: Space-efficient straggler identification in round-trip data streams via Newton's identitities and invertible Bloom filters. In: Workshop on Algorithms and Data Structures, pp. 637–648 (2007)
12. Frahling, G., Indyk, P., Sohler, C.: Sampling in dynamic data streams and applications. In: Symposium on Computational Geometry, pp. 142–149 (2005)
13. Ganguly, S.: Counting distinct items over update streams. Theor. Comput. Sci. **378**(3), 211–222 (2007)
14. Gilbert, A.C., Strauss, M.J., Tropp, J.A., Vershynin, R.: One sketch for all: fast algorithms for compressed sensing. In: ACM Symposium on Theory of Computing, pp. 237–246 (2007)
15. Indyk, P.: A small approximately min-wise independent family of hash functions. In: ACM-SIAM Symposium on Discrete Algorithms, pp. 454–456 (1999)
16. Indyk, P., Motwani, R.: Approximate nearest neighbors: towards removing the curse of dimensionality. In: ACM Symposium on Theory of Computing, pp. 604–613 (1998)
17. Johnson, W., Lindenstrauss, J.: Extensions of Lipshitz mapping into Hilbert space. Contemp. Math. **26**, 189–206 (1984)
18. Jowhari, H., Sağlam, M., Tardos, G.: Tight bounds for l_p samplers, finding duplicates in streams, and related problems. In: ACM Principles of Database Systems, pp. 49–58 (2011)
19. Kane, D.M., Nelson, J., Woodruff, D.P.: An optimal algorithm for the distinct elements problem. In: ACM Principles of Database Systems, pp. 41–52 (2010)
20. Manerikar, N., Palpanas, T.: Frequent items in streaming data: an experimental evaluation of the state-of-the-art. Data Knowl. Eng. **68**(4), 415–430 (2009)
21. Metwally, A., Agrawal, D., El Abbadi, A.: Why go logarithmic if we can go linear?: Towards effective distinct counting of search traffic. In: EDBT, pp. 618–629 (2008)
22. Monemizadeh, M., Woodruff, D.P.: 1-pass relative-error l_p-sampling with applications. In: ACM-SIAM Symposium on Discrete Algorithms, pp. 1143–1160 (2010)
23. Nisan, N.: Pseudorandom generators for space-bounded computations. In: ACM Symposium on Theory of Computing, pp. 204–212 (1990)
24. Patrascu, M., Thorup, M.: The power of simple tabulation hashing. In: ACM Symposium on Theory of Computing, pp. 1–10 (2011)
25. Pike, R., Dorward, S., Griesemer, R., Quinlan, S.: Interpreting the data: parallel analysis with sawzall. Sci. Program. **13**(4), 277–298 (2005)
26. Price, E.: Efficient sketches for the set query problem. In: ACM-SIAM Symposium on Discrete Algorithms, pp. 41–56 (2011)
27. Schmidt, J.P., Siegel, A., Srinivasan, A.: Chernoff–Hoeffding bounds for applications with limited independence. In: ACM-SIAM Symposium on Discrete Algorithms, pp. 331–340 (1993)