Practical aspects of ℓ_0-sampling algorithms

Juan P. A. Lopes[1], **Fabiano S. Oliveira**[2], **Valmir C. Barbosa**[1]

[1]PESC/COPPE – Universidade Federal do Rio de Janeiro – Rio de Janeiro – Brasil

[2]IME – Universidade do Estado do Rio de Janeiro – Rio de Janeiro – Brasil

jlopes@cos.ufrj.br, fabiano.oliveira@ime.uerj.br, valmir@cos.ufrj.br

Abstract. *The ℓ_0-sampling problem plays an important role in streaming graph algorithms. In this paper, we revisit a near-optimal ℓ_0-sampling algorithm, proposing a variant that allows proving a tighter upper bound for the probability of failure. We compare experimental results of both variants, providing empirical evidence of their applicability in real-case scenarios.*

1. Introduction

The ℓ_0-sampling problem consists in sampling a nonzero coordinate from a dynamic vector $\boldsymbol{a} = (a_1, \ldots, a_n)$ with uniform probability. This vector is defined in a turnstile model, which consists of a stream of updates $S = \langle s_1, s_2, \ldots, s_t \rangle$ on $\boldsymbol{a}$ (initially $\mathbf{0}$), where $s_i = (u_i, \Delta_i) \in \{1, \ldots, n\} \times \mathbb{R}$ for all $1 \leq i \leq t$, meaning an increment of Δ_i units to a_{u_i}. It is desirable that such sample be produced in a single pass through the stream with sublinear space complexity. The challenge arises from the fact that, since Δ_i can be negative and hence some updates in the stream may cancel others, directly sampling the stream may lead to incorrect results.

The research on ℓ_0-sampling algorithms has recently gained some traction, in part due to results showing that these algorithms can be used as building blocks in many other algorithms [Cormode and Firmani 2014]. For example, the sampling of nonzero coordinates from rows of the incidence matrix of a graph can be used to compute connected components, k-connectivity, bipartiteness, and approximate minimum spanning trees in dynamic graphs using $O(n \log^c n)$ bits, for some constant c [Ahn et al. 2012, McGregor 2014].

In order to achieve sublinear space complexity in a single pass, an ℓ_0-sampling algorithm must represent $\boldsymbol{a}$ through a lower-dimensional projection. This representation is known as a *sketch*. Sketch-based algorithms are common in streaming scenarios, by virtue of allowing compact representations of the original data, whilst retaining some useful information about them.

In [Cormode et al. 2005], a seminal sketch-based algorithm for the ℓ_0-sampling problem is introduced. The algorithm uses a universal family of hash functions to partition the vector $\boldsymbol{a}$ into $O(\log n)$ subvectors with exponentially decreasing probabilities of representing each element of $\boldsymbol{a}$. It is proven that there is a constant lower bound on the probability that at least one of those subvectors has exactly one nonzero coordinate. Through a procedure called 1-*sparse recovery* (Section 2), which requires $O(\log n)$ bits for each subvector, it is possible to recover such coordinate. Considering that the probability of failure has a constant upper bound, running $O(\log(1/\delta))$ independent instances of the algorithm

can ensure a success probability of at least $1 - \delta$. The total space complexity of this algorithm is $O(\log^2 n \log(1/\delta))$. Further studies show stronger results by relaxing assumptions on the hash functions used [Monemizadeh and Woodruff 2010, Jowhari et al. 2011]. Nevertheless, they keep the same worst-case space complexity. In fact, any algorithm that performs ℓ_0-sampling in a single pass should require $\Omega(\log^2 n)$ bits in the worst case [Jowhari et al. 2011]. This holds even if the algorithm allows a relative error of ϵ and a failure probability of δ, for constants ϵ and δ.

2. 1-sparse recovery procedure

A vector is 1-*sparse* when it has a single nonzero coordinate. A 1-sparse recovery procedure allows deciding whether a vector $\boldsymbol{a}$ is 1-sparse, and possibly recover the only nonzero coordinate from it. Note that while $\boldsymbol{a}$ is expected to be 1-sparse at the time of a successful recovery, it may have any number of nonzero coordinates before that. This procedure is a building block for many ℓ_0-sampling algorithms. Here we present a false-biased randomized variant that handles cases where $\boldsymbol{a}$ has negative values [Cormode and Firmani 2014]. It begins by choosing a sufficiently large prime $p \in \Theta(n^c)$, with $c > 1$, and random integer $z \in \mathbb{Z}_p$. Then, iterating through all $s_i = (u_i, \Delta_i) \in S$, three sums are computed:

$$b_0 = \sum_{i=1}^{t} \Delta_i, \qquad b_1 = \sum_{i=1}^{t} \Delta_i u_i, \qquad b_2 = \sum_{i=1}^{t} \Delta_i z^{u_i} \mod p.$$

If $\boldsymbol{a}$ is 1-sparse, it is easy to see that the nonzero coordinate can be recovered as $i = b_1/b_0$, with $a_i = b_0$. However, verifying that $\boldsymbol{a}$ is 1-sparse requires more effort.

Theorem 1. *If $\boldsymbol{a}$ is 1-sparse, then $b_2 \equiv b_0 z^{b_1/b_0} \mod p$. Otherwise, $b_2 \not\equiv b_0 z^{b_1/b_0} \mod p$ with probability at least $1 - n/p$.*

Proof (sketch). If $\boldsymbol{a}$ is 1-sparse, with a nonzero coordinate i, it is trivial to see that $b_2 \equiv a_i z^i \mod p$. Otherwise, $b_2 \equiv b_0 z^{b_1/b_0} \mod p$ may still hold if z is a root in $\mathbb{Z}_p$ of the polynomial $p(z) = b_0 z^{b_1/b_0} - \sum \Delta_i z^{u_i}$. As $p(z)$ is an degree-n polynomial, it has at most n roots in $\mathbb{Z}_p$. Therefore, given that z is chosen at random, the probability of a false recovery is at most n/p. ■

This 1-sparse recovery procedure stores z, and the sums b_0, b_1, and b_2. Assuming that every a_i is limited by a polynomial in n, the total space required is $O(\log n)$ bits.

3. ℓ_0-sampling algorithm

In this work, two variants of the same ℓ_0-sampling algorithm are presented. Both variants define $\boldsymbol{a^{(1)}}, \boldsymbol{a^{(2)}}, \ldots, \boldsymbol{a^{(m)}}$ subvectors of $\boldsymbol{a}$. For all $1 \leq j \leq m$, each $a_i \neq 0$ has a $1/2^j$ probability of being *present* at $\boldsymbol{a^{(j)}}$, that is, $a_i^{(j)} = a_i$ with probability $1/2^j$, otherwise $a_i^{(j)} = 0$. To decide whether $a_i^{(j)}$ is present, we draw a hash function $h_j : \{1, \ldots, n\} \to \{0, \ldots, 2^m - 1\}$ from a universal family, and observe whether $m - \lfloor \log_2 h_j(i) \rfloor = j$, which happens with probability $1/2^j$. An independent 1-sparse recovery is then computed for each $\boldsymbol{a^{(j)}}$. The variants differ only in the number of functions used. Variant (a) uses a single hash function for every $\boldsymbol{a^{(j)}}$ (Algorithm 1), while Variant (b) uses a different function for each subvector (Algorithm 2). While (a) is more useful in practice, the error analysis in (b) is more straightforward. We provide empirical evidence in Section 4 that the error in either variant converges quickly as a function of n.

Algorithm 1 Variant (a)

$M[1..m]$: 1-sparse recoveries
for each $(u_i, \Delta_i) \in S$ **do**
  $k \leftarrow m - \lfloor \log_2 h(u_i) \rfloor$
  $M[k].b_0 \mathrel{+}= \Delta_i$
  $M[k].b_1 \mathrel{+}= \Delta_i u_i$
  $M[k].b_2 \mathrel{+}= \Delta_i M[k].z^{u_i} \mod p$
for $j \in [1..m]$ **do**
  $v \leftarrow M[j].b_0 M[j].z^{M[j].b_1/M[j].b_0} \mod p$
  if $M[j].b_2 = v$ **then**
    return $M[j].b_1/M[j].b_0$
report FAILURE

Algorithm 2 Variant (b)

$M[1..m]$: 1-sparse recoveries
for each $(u_i, \Delta_i) \in S$ **do**
  for $j \in [1..m]$ **do**
    $k \leftarrow m - \lfloor \log_2 h_j(u_i) \rfloor$
    if $k = j$ **then**
      $M[k].b_0 \mathrel{+}= \Delta_i$
      $M[k].b_1 \mathrel{+}= \Delta_i u_i$
      $M[k].b_2 \mathrel{+}= \Delta_i M[k].z^{u_i} \mod p$
for $j \in [1..m]$ **do**
  $v \leftarrow M[j].b_0 M[j].z^{M[j].b_1/M[j].b_0} \mod p$
  if $M[j].b_2 = v$ **then**
    return $M[j].b_1/M[j].b_0$
report FAILURE

Every variant either succeeds in returning a single nonzero coordinate of $\boldsymbol{a}$, or reports a failure. The probability of failure is given by the joint probability of failure of all m 1-sparse recoveries. In Variant (b), those are independent events. Moreover, the probability that a single recovery $M[j]$ fails is the complement of the probability that $\boldsymbol{a}^{(j)}$ is 1-sparse, that is, assuming $\boldsymbol{a}$ has $r \gg 1$ nonzero coordinates:

$$\Pr[\text{FAILURE}] = \prod_{j=1}^{m} \left(1 - r2^{-j}(1-2^{-j})^{r-1}\right) \approx \prod_{j=1}^{m} \left(1 - r2^{-j}e^{-r2^{-j}}\right).$$

Theorem 2. *If* $5 \leq \log_2 r \leq m - 5$*, then* $\Pr[\text{FAILURE}] \leq 0.31$*, for Variant (b).*

Proof (sketch). It is easy to see that the lowest probabilities of failure concentrate around j such that $2^j \leq r < 2^{j+1}$. Letting $q = r/2^{\lfloor \log_2 r \rfloor}$, it holds that

$$\Pr[\text{FAILURE}] \leq \prod_{k=-5}^{5} \left(1 - q2^k e^{-q2^k}\right).$$

Note that $1 \leq q < 2$. In this interval, all factors $1 - q2^k e^{-q2^k}$ are either monotonically increasing or decreasing. Analyzing their global maxima, we arrive at a maximum product of approximately 0.3071, therefore $\Pr[\text{FAILURE}] \leq 0.31$. ■

This result shows that, as n grows, choosing $m = 5 + \lceil \log_2 n \rceil$ is enough to ensure a constant upper bound on the probability of failure. Furthermore, to ensure a success probability of at least $1 - \delta$, it is sufficient to run $\lceil \log_{0.31} \delta \rceil$ instances of the algorithm.

4. Empirical evaluation and outlook

In order to assess the algorithms behavior in a real implementation, an experiment was set up. Both variants were implemented and tested with a vector of size $n = 4\,096$ and increasing values of r. We tested both a correctly sized (i.e., for $m = 17$) and an undersized instance of the ℓ_0-sampling algorithm. The empirical cumulative distribution was also recorded. The experiment was run $100\,000$ times and the mean value for each data point is reported in Figure 1.

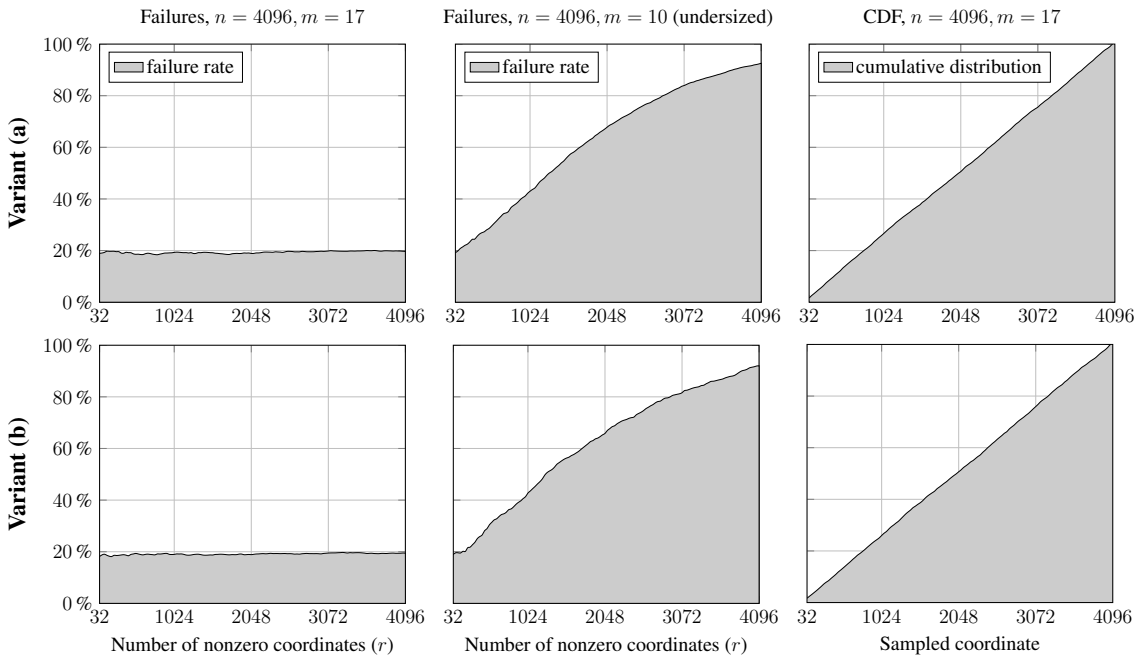

Figure 1. Failure rate and cumulative distribution of successes.

This experiment suggests that in a correctly sized ℓ_0-sampling, the failure probability stays almost constant under 20%. There is little difference between Variants (a) and (b). Furthermore, in an undersized setup, the failure rate rapidly reaches critical levels.

In conclusion, we have introduced a variant of the ℓ_0-sampling algorithm and proved its failure probability to be bounded by a constant value, provided a certain structure-size condition is met. Research is ongoing on the proof of exact probabilities of failure for both algorithm variants. Future research may also include novel graph algorithms that use ℓ_0-sampling as a primitive.

References

Ahn, K. J., Guha, S., and McGregor, A. (2012). Analyzing graph structure via linear measurements. In *Proceedings of SODA'12*, pages 459–467.

Cormode, G. and Firmani, D. (2014). A unifying framework for ℓ_0-sampling algorithms. *Distributed and Parallel Databases*, 32(3):315–335.

Cormode, G., Muthukrishnan, S., and Rozenbaum, I. (2005). Summarizing and mining inverse distributions on data streams via dynamic inverse sampling. In *Proceedings of VLDB'05*, pages 25–36.

Jowhari, H., Sağlam, M., and Tardos, G. (2011). Tight bounds for l_p samplers, finding duplicates in streams, and related problems. In *Proceedings of PODS'11*, pages 49–58.

McGregor, A. (2014). Graph stream algorithms: a survey. *ACM SIGMOD Record*, 43(1):9–20.

Monemizadeh, M. and Woodruff, D. P. (2010). 1-pass relative-error l_p-sampling with applications. In *Proceedings of SODA'10*, pages 1143–1160.