Distinct Counting with a Self-Learning Bitmap

Aiyou Chen and Jin Cao
Bell Labs, Alcatel-Lucent
{aychen,cao}@research.bell-labs.com

***Abstract*— Estimating the number of distinct values is a fundamental problem in database that has attracted extensive research over the past two decades, due to its wide applications (especially in the Internet). Many algorithms have been proposed via sampling or sketching for obtaining statistical estimates that only require limited computing and memory resources. However, their performance in terms of relative estimation accuracy usually depends on the unknown cardinalities. In this paper, we address the following question: can a distinct counting algorithm have uniformly reliable performance, i.e. constant relative estimation errors for unknown cardinalities in a wide range, say from tens to millions? We propose a self-learning bitmap algorithm (S-bitmap) to answer this question. The S-bitmap is a bitmap obtained via a novel adaptive sampling process, where the bits corresponding to the sampled items are set to 1, and the sampling rates are learned from the number of distinct items already passed and reduced sequentially as more bits are set to 1. A unique property of S-bitmap is that its relative estimation error is truly stabilized, i.e. invariant to unknown cardinalities in a prescribed range. We demonstrate through both theoretical and empirical studies that with a given memory requirement, S-bitmap is not only uniformly reliable but more accurate than state-of-the-art algorithms such as the multiresolution bitmap [6] and Hyper LogLog algorithms [8] under common practice settings.**

I. Introduction

Estimating the number of distinct values (cardinality) in a large dataset or streaming data is a fundamental problem that has wide applications, including data integration, query optimization, and especially, network traffic monitoring. Over the past two decades, many algorithms have been developed in the literature that assume only one-scan of the data and derive statistical estimates with limited memory and computational resources ([9], [15], [7], [1], [4], [11], [2], [5], [6], [10], [12], [3], [8]). Almost all of them make use of one of the following basic ideas plus appropriate estimation methods: [i] sampling or Wegman's adaptive sampling [7] to obtain a portion of distinct values; [ii] bitmap that first randomly assigns each distinct value to an entry of a bit vector and then uses the number of empty entries to infer the count; [iii] order statistics (e.g. minimal) obtained by hashing each distinct value to a random number with a common distribution; and [iv] combination of the above [i], [ii] or [iii]. See [13] for a comprehensive review and extensive simulation comparisons of the algorithms appearing before 2006. Most notable algorithms include linear counting [15] falling into [ii], Flajolet-Martin's logarithmic counting algorithm [9] based on minimal statistics of geometric random numbers (i.e. [iii]), the multiresolution bitmap algorithm (mr-bitmap) [6] using a combination of sampling and bitmap (i.e. [iv]), and the LogLog counting algorithm (LLog) [5] ([iii]) that improves the logarithmic counting algorithm from space requirement $O(\log(N_{max}))$ to $O(\log\log(N_{max}))$. Here N_{max} is an upper bound of cardinalities to be estimated. The Hyper-Loglog (HLLog) algorithm proposed recently by Flajolet et al [8] that further improves LLog also belongs to [iii].

Although many algorithms for distinct counting exist, their performance in terms of relative estimation accuracy usually depend on the unknown cardinalities. For example, with limited memory, linear counting [15] works best with small cardinalities while LLog works best with large cardinalities. However, in many network applications, the cardinalities involved not only may have large scales but also may fall in a wide range due to the non-stationarity of network traffic in the time domain or inhomogeneity of link speeds in the spatial domain. For example, network traffic is typically non-stationary and bursty, especially under unusual events. The property of non-stationarity imposes the challenge that algorithms relying on sampling design adapted from historical data will not be reliable. Further, network links belonging to a large service provider may have different link speeds, and therefore real-time flow counts on these links can be very heterogeneous, for which, it is critical but challenging to guarantee reliable estimation performance among all links using a uniform design of memory and computation resources. The lack of homogeneity in these problems, imposes a central question: *Can a distinct counting algorithm provide uniformly reliable estimates, in the sense of constant relative estimation errors, no matter what the cardinalities are, say from tens to millions?* Here we use the error metric defined by the square root of mean square relative errors, i.e. $\sqrt{E(\hat{n}n^{-1}-1)^2}$ where $\hat{n}$ is an estimate of the cardinality, say n.

In this paper we address the above central question and develop a self-learning bitmap algorithm that meets the requirement of uniform reliability. Our idea is to build an adaptive sampling process into a bitmap such that the relative errors are stabilized constantly. To be more precise, the sampling rates are learned from the number of distinct items already passed and decreased sequentially as more and more bits in the bitmap are filled. The abbreviation of "S-bitmap" therefore comes from the features, self-learning and adaptive sampling bitmap. Our adaptive sampling idea can be traced back to Morris in 1978 [14] who proposed it for the simple item counting problem with no replicate items. We note that Morris' adaptive sampling is different from Wegman's adaptive sampling since Wegman's sampling scheme was designed for distinct counting; Furthermore, Wegman's sampling does not consider uniform reliability defined in the above and is yet computationally more expensive. We show that the S-bitmap algorithm is not only uniformly reliable but more accurate than state-of-the-art algorithms for common practice cardinality scales. For example, with 2,520 bits (or 315 bytes), the S-bitmap can estimate arbitrary cardinalities from 1 to a million with relative errors uniformly about 4%. As a comparison, state-of-the-art algorithms such as mr-bitmap, LogLog counting require requires at least twice the memory, and even Hyper-LogLog requires 30% more memory, in order to achieve the similar performance.

Organization. Section II describes the the S-bitmap algorithm, dimensioning rules and some analysis; Section III reports performance comparison with state-of-the-art algorithms and experimental studies on monitoring worm traffic.

II. Self-learning Bitmap

In this section we begin with a description of the S-bitmap algorithm. The key idea is to build a novel adaptive sampling process into a bitmap. Since typically $N_{max} \gg m$, mapping from $\{0, \cdots, m\}$ to $\{0, \cdots, N_{max}\}$ cannot be one-to-one, but one-to-multiple. In order to efficiently make use of these m bits, the sampling rates are learned from the number of distinct values already passed by and are updated adaptively for dynamic arrival of new distinct values. The sampling process forms a non-stationary Markov chain, from which an efficient statistical estimate is derived. Below we describe the algorithm, its dimensioning rules of the sampling rates and some analysis.

Self-learning bitmap. A bitmap is a vector $V \in \{0,1\}^m$ with length m and is initialized with 0s. A counter L is initialized by 0 for counting the number of buckets filled with 1s, then $0 \le L \le m$. Assume that items come sequentially as a data stream. Upon the arrival of an item (the same item may appear multiple times), it is mapped by a universal hash function using the item ID as the key to the kth bucket in the bitmap for some $k \in \{1, \cdots, m\}$. If $V[k] = 1$, then skip to the next item; Otherwise, with a sampling probability p_L, $V[k]$ is changed from 0 into 1, in which case we update L by $L+1$. Note that the sampling is also applied with a universal hash using the item IDs as keys. Here, $L \in \{0, 1, \cdots, m\}$ tells how many 1-bits by the end of the stream update. The bigger L is, the larger the cardinality is expected to be. We require that $p_1, p_2, \cdots$ be decreasing such that for a large cardinality, items are sampled with smaller and smaller probability sequentially, while for a small cardinality, items are sampled using the first a few of $p_1, p_2, \cdots$, i.e. higher sampling rates. The rationale is to make the relative estimation error constant for an arbitrary cardinality in the whole range $[0, \cdots, N_{max}]$. The method can also be applied for the range $[N_{min}, \cdots, N_{max}]$ when $N_{min} > 1$ is a known lower bound of cardinalities, see our full paper at http://cm.bell-labs.com/who/aychen/sbitmap.pdf.

The S-bitmap update algorithm is described in Algorithm 1, where the configuration parameters, i.e. sampling rates, $\{p_1, p_2, \cdots, p_m\}$ are described later. Since both sampling and bitmap hashing use the item as the key, we can use one universal hash function to serve both purposes. Following the setting of [5], each item is assumed to be a binary vector (say after a universal hash) with length $c+d$, where c is determined by bitmap size i.e. $m = 2^c$, and d is pre-determined by N_{max} (i.e. $c + d = O(\log(N_{max}))$) and the sampling rate precision. Since the sequential sample rates p_L only depend on L which allow us to learn the number of distinct items already passed, the algorithm is called self-learning bitmap (S-bitmap).

Algorithm 1 S-bitmap (UPDATE)

```
Input: a stream of items $x$ (a binary vector with size $c + d$)
       $V$ (a bitmap vector of zeros with size $m = 2^c$)
Output: $B$ (number of buckets with 1s in $V$)
Configuration: $m, C$ and $(p_1, \cdots, p_m)$
1: Initialize $L = 0$
2: for $x = b_1 \cdots b_{c+d} \in \mathcal{U}$ do
3:     set $j := [b_1 \cdots b_c]_2$ (value of first $c$ bits in base 2)
4:     if $V[j] = 0$ then
5:         $u = [b_{c+1} \cdots b_{c+d}]_2$
6:         if $u2^{-d} < p_{L+1}$ then
7:             $V[j] = 1$
8:             $L = L + 1$
9: Return $B = L$.
```

Note that in the update process, for each item, only one hash is needed. The bucket location is computed using the first c bits of hashed binary values. For bucket update, only if the localized bucket is empty, the next d-bits block is used which determines whether the bucket should be filled with 1 or not. Note that the sampling rate changes only when an empty bucket is filled with 1. For example, if K buckets become filled by the end of the stream, the sample rates only need to be updated K times. Therefore the computational cost is low compared to to most existing algorithms.

Estimation. From the update process, it is clear that each of the n distinct items is assigned into one of the m bit locations independently with equal probability. Replicate items will be hashed into the same bit location and be sampled either all or none. For simplicity, let the distinct items be index sequentially by $1, 2, \cdots, n$. Let L_t be the number of 1 bits in the bitmap when the t-th distinct item arrives. It can be shown that $\{L_t : t = 1, \cdots, n\}$ follows a non-stationary Markov chain model as below:

$$\begin{aligned} L_t &= L_{t-1} + 1, \text{ with probability } q_{L_{t-1}+1} \\ &= L_{t-1}, \text{ with probability } 1 - q_{L_{t-1}+1}, \end{aligned}$$

where $q_l = m^{-1}(m - l + 1)p_l$.

Let T_l be the index for the distinct item that fills an empty bucket with 1 such that there are l buckets filled with 1 by that time. By default, $T_0 \equiv 0$. Now given the output B from the update process, obviously, we have $T_B \le n < T_{B+1}$. Let $t_b = ET_b$ be the expectation of T_b, $b = 1, \cdots, m$. Then a natural estimate of n could be any value in $[t_b, t_{b+1})$ if $B = b$. In order to stabilize the estimation errors at boundaries, our S-bitmap estimator is defined as below:

$$\hat{n} = \begin{cases} \frac{2t_B t_{B+1}}{t_B + t_{B+1}} & \text{if } B < m \text{ and } t_{B+1} - t_B > 1.5 \\ t_B & \text{otherwise.} \end{cases} \quad (1)$$

Since $t_{B+1} - t_B \geq 1$, the threshold 1.5 is used to avoid interpolation for cardinalities close to 1.

Lemma 1: $\{T_l - T_{l-1} : 1 \leq l \leq m\}$ are independent geometric random variables with corresponding means q_l^{-1} and variances $(1-q_l)q_l^{-2}$. Therefore $t_l = \sum_{j=1}^{l} q_j^{-1}$.

Dimensioning rule. Notice that the stopping times T_b are unbiased estimates of $t_b = ET_b$ if the stopping times are observable but the parameters are unknown, where $t_1 < t_2 < \cdots < t_m$. Notice that the unknown cardinality n must fall into some two consecutive t_b. Let

$$Re(T_b) = \sqrt{E(T_b t_b^{-1} - 1)^2} = t_b^{-1}\sqrt{var(T_b)}$$

be the relative error for T_b estimating t_b. In order to make the S-bitmap estimation error invariant to an arbitrary n, the idea is to choose sample rates that stabilize the relative estimation errors at these breaking points, i.e. $Re(T_b) \equiv \epsilon$ for $b \in \{1, \cdots, m\}$, where ϵ is the desired relative estimation error. We will show later that although the stopping times T_b are unobservable, choosing parameters that stabilizes the relative errors of T_b is sufficient for stabilizing the relative estimation errors of our S-bitmap estimator for arbitrary $n \leq N_{max}$. The theorem below summarizes the dimensioning rule.

Theorem 1: Given m and ϵ, let $r = 1 - 2\epsilon^2(1+\epsilon^2)^{-1}$. Let the sequential sampling rates be $p_k = m(m+1-k)^{-1}(1+\epsilon^2)r^k$. Then for $k = 1, \cdots, m$,

$$Re(T_k) \quad \equiv \quad \epsilon.$$

That is, the relative errors $Re(T_b)$ for stopping time expectations are constant.

For the upper bound N_{max}, we let $t_m = N_{max}$, i.e., the bitmap is expected to be filled up when the maximum number of distinct items are reached. So, $N_{max} = .5\epsilon^{-2}(r^{-m} - 1)$. From this, we can express m as

$$m \quad = \quad \frac{\log(1 + 2N_{max}\epsilon^2)}{\log(1 + 2\epsilon^2(1-\epsilon^2)^{-1})}. \qquad (2)$$

This tells how to choose the bitmap size with given N_{max} and estimation precision ϵ.

Analysis. We conjecture that the following is true. If given N_{max}, the sequential sampling rates are designed by Theorem 1 with parameters (m, ϵ), where m is given by (2), then the relative error for the S-bitmap algorithm is $\epsilon(1 + o(1))$ for any $n \in \{1, \cdots, N_{max}\}$. This is supported by two sets of empirical studies. First, we fix $m = 4,000$ bits and $\epsilon = 0.033$ and design the sequential sampling rates according to Theorem 1, then for n the power of 2 in the range [32, 1.2×10^6], we simulate n distinct items and obtain S-bitmap estimate. For each n, we replicate the simulation 1000 times and obtain the empirical mean square relative error. The second set uses $m = 1,800$ bits and $\epsilon = 0.05$, and similarly empirical estimation errors are computed. These empirical errors are compared with theoretical error ϵ, shown in Figure 1. In both studies, the empirical errors and theoretical errors match very well. We leave theoretical justification of the empirical results to future studies. A less rigorous result is given below.

Theorem 2: Under the S-bitmap algorithm setting, the relative error is of order ϵ.

III. Comparison and Empirical Evaluation

In this section we compare S-bitmap both theoretically and empirically with state-of-the-art algorithms including mr-bitmap, LLog and HLLog which are known to be best performers in terms of space efficiency for the same accuracy. The studies show that S-bitmap is uniformly reliable and also performs better than the alternative algorithms in common practice setting with $N_{max} = O(10^7)$.

Theoretical comparison. Notice that from the asymptotic analysis in [5] and [8], the space requirements for LLog and HLLog are asymptotically most efficient, about $1.30^2\epsilon^{-2}\log(\log(N_{max}))$ and $1.04^2\epsilon^{-2}\log(\log(N_{max}))$ in bits respectively, where ϵ is the desired relative error. We provide a comparison in Table I below between S-bitmap, LLog and HLLog for the memory requirement in bits. Table I tells that to achieve the accuracy of 1% relative errors, S-bitmap is more efficient than LLog and HLLog when N_{max} is of order 10^8 or less. S-bitmap is more advantageous for smaller $N_{max}\epsilon^2$ while HLLog is most advantageous for extremely large N_{max}.

N_{max}	S-bitmap (Kb)	Hyper LogLog (Kb)	LogLog (Kb)
10000	5.5	44.1	67.6
1e+05	15.2	54.1	84.5
1e+06	26.5	54.1	84.5
1e+07	38.0	54.1	84.5
1e+08	49.5	54.1	84.5
1e+09	61.0	54.1	84.5

TABLE I
MEMORY REQUIREMENT IN KILO-BITS (KB) FOR S-BITMAP, LOGLOG AND HYPER-LOGLOG TO ACHIEVE ACCURACY OF 1% RELATIVE ERROR

Empirical comparison. We now use empirical studies to compare mr-bitmap, LLog and HLLog with S-bitmap. In the first experiment, with $N_{max} = 1.5 \times 10^6$ and $\epsilon = 1\%$, the design of multiresolution bitmap algorithm requires $m = 55,500$ bits of memory. We therefore apply this m to all algorithms. Let the true cardinality n vary from 10 to 10^6 and run the algorithms to obtain corresponding estimates $\hat{n}$. The relative estimation error, measured by $\sqrt{E(\hat{n}n^{-1} - 1)^2}$, is computed based on 1000 replicates of this procedure. This is reported in the first panel of Figure 3. Note that the LLog algorithm has a flaw in handling small cardinalities, which is remedied in HLLog (see [8] for details). The result also tells that mr-bitmap does not produce reliable results as its performance is better for small cardinalities but worse for large cardinalities, while S-bitmap always produces reliable estimates with relative error about 0.7%. In the second and third experiments, we use $\epsilon = 4\%, 10\%$ for the mr-bitmap design and obtain $m = 3242$ and $m = 941$ respectively. The simulation setup is similar to the above and the results are reported in the second and third panels of Figure 3 respectively. As shown, besides uniform reliability, S-bitmap is most accurate in most cases.

Worm traffic monitoring. We use a 9-hour traffic traces collected at a large university network during the period of first

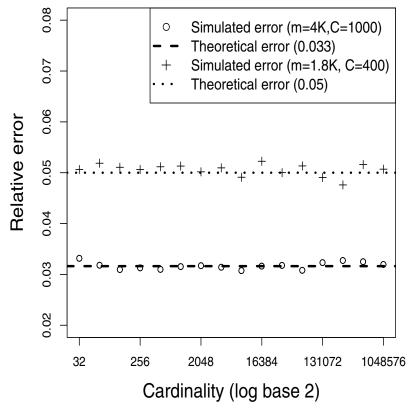


Fig. 1. Empirical and theoretical estimation errors with (m, ϵ) taking values (4000,0.033) and (1800, 0.05) (Note: $C = \epsilon^{-2}$).

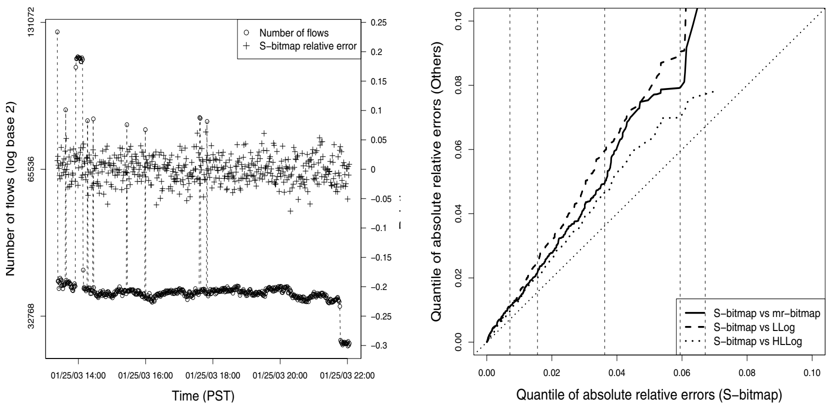
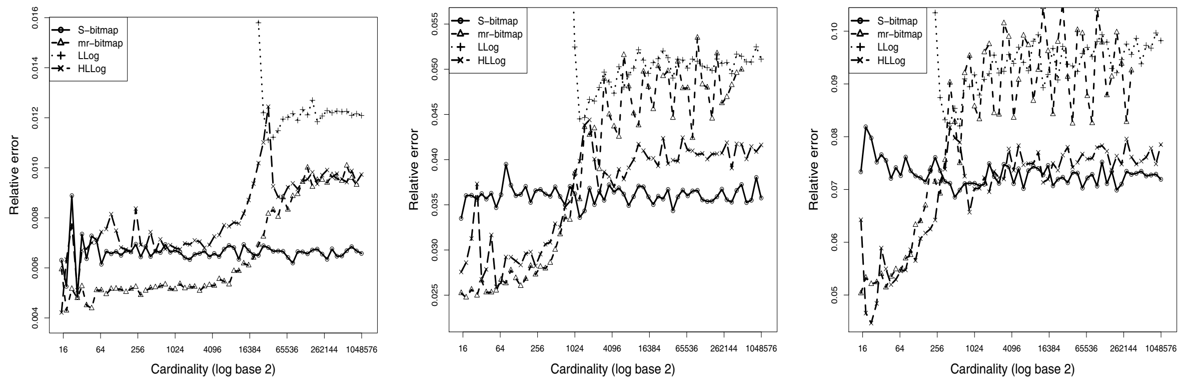


Fig. 2. Left: Time series of flow counts (in circles) and S-bitmap estimation relative errors (in '+') per minute; Right: Quantile-Quantile plots of absolute relative estimation errors between S-bitmap and others (mr-bitmap, LLog, HLLog).



(a) Memory: m=55,500 bits (b) Memory: m=3,242 bits (c) Memory: m=941 bits

Fig. 3. Comparison between S-bitmap and other algorithms: mr-bitmap, loglog and loglog-bitmap

outbreak of the 'Slammer' worm in 2003. Distinct counting algorithms are used to estimate the number of network flows per minute. The left panel of Figure 2 shows the true flow counts in triangles and the relative errors of S-bitmap estimates in '+' with standard deviation 2.2% and median absolute error about 3.5%. The right panel of Figure 2 compares the quantiles of absolute relative errors of the four algorithms, where the vertical lines are the 25, 50, 90, 99 and 99.9 percent quantiles. The result shows that the absolute error quantiles of S-bitmap is uniformly better than all others, while HLLog is better than mr-bitmap and LLog.

REFERENCES

[1] N. Alon, Y. Matias, and M. Szegedy. The space complexity of approximating the frequency moments. In *Proc. ACM Symposium on Theory of Computing*, 1996.

[2] Z. Bar-Yossef, T. S. Jayram, R. Kumar, D. Sivakumar, and L. Trevisan. Counting distinct elements in a data stream. In *RANDOM*, pages 1–10, 2002.

[3] K. Beyer, P. Haas, B. Reinwald, Y. Sismanis, and R. Gemulla. On synopses for distinct-value estimation under multiset operations. In *SIGMOD*, 2007.

[4] E. Cohen. Size-estimation framework with applications to transitive closure and reachability. *Journal of Computer and System Sciences*, 55(3):441–453, 1997.

[5] M. Durand and P. Flajolet. Loglog counting of large cardinalities. In *European Symposium on Algorithms*, pages 605–617, 2003.

[6] C. Estan, G. Varghese, and M. Fisk. Bitmap algorithms for counting active flows on high speed links. *IEEE/ACM Trans. on Networking*, 14(5):925–937, 2006.

[7] P. Flajolet. On adaptive sampling. *Computing*, 34:391–400, 1990.

[8] P. Flajolet, E. Fusy, O. Gandouet, and F. Meunier. Hyperloglog: the analysis of a near-optimal cardinality estimation algorithm. *Analysis of Algorithms*, 2007.

[9] P. Flajolet and G. N. Martin. Probabilistic counting algorithms for data base applications. *Journal of Computer and System Sciences*, 31(2):182–209, 1985.

[10] S. Ganguly. Counting distinct items over update streams. In *Proceedings of the 16th ISAAC International Symposium on ALgorithms and Computation*, pages 505–514, 2005.

[11] P. Gibbons. Distinct sampling for highly-accurate answers to distinct values queries and event reports. *The VLDB Journal*, pages 541–550, 2001.

[12] F. Giroire. Order statistics and estimating cardinalities of massive datasets. In *Proceedings of the 6th DMTCS Discrete Mathematics and Theoretical Computer Science*, pages 157–166, 2005.

[13] A. Metwally, D. Agrawal, and A. E. Abbadi. Why go logarithmic if we can go linear? towards effective distinct counting of search traffic. In *Proc. ACM EDBT*, March 2008.

[14] R. Morris. Counting large numbers of events in small registers. *Commun. ACM*, 21(10):840–842, 1978.

[15] K. Whang, B. Vander-Zanden, and H. Taylor. A linear-time probabilistic counting algorithm for database applications. *ACM Transactions on Database Systems*, 1990.