Loglog Counting of Large Cardinalities
(Extended Abstract)

Marianne Durand and Philippe Flajolet

Algorithms Project, INRIA–Rocquencourt, F78153 Le Chesnay (France)

Abstract. Using an auxiliary memory smaller than the size of this abstract, the LogLog algorithm makes it possible to estimate in a single pass and within a few percents the number of different words in the whole of Shakespeare's works. In general the LogLog algorithm makes use of m "small bytes" of auxiliary memory in order to estimate in a single pass the number of distinct elements (the "*cardinality*") in a file, and it does so with an accuracy that is of the order of $1/\sqrt{m}$. The "small bytes" to be used in order to count cardinalities till $N_{\max}$ comprise about $\log\log N_{\max}$ bits, so that cardinalities well in the range of billions can be determined using one or two kilobytes of memory only. The basic version of the LogLog algorithm is validated by a complete analysis. An optimized version, super–LogLog, is also engineered and tested on real-life data. The algorithm parallelizes optimally.

1 Introduction

The problem addressed in this note is that of determining the number of *distinct* elements, also called the *cardinality*, of a large file. This problem arises in several areas of data-mining, database query optimization, and the analysis of traffic in routers. In such contexts, the data may be either too large to fit at once in core memory or even too massive to be stored, being a huge continuous flow of data packets. For instance, Estan *et al.* [3] report traces of packet headers, produced at a rate of 0.5GB per hour of compressed data (!), which were collected while trying to trace a "worm" (Code Red, August 1 to 12, 2001), and on which it was necessary to count the number of *distinct* sources passing through the link. We propose here the LogLog algorithm that estimates cardinalities using only a very small amount of auxiliary memory, namely m memory units, where a memory unit, a "*small byte*", comprises close to $\log\log N_{\max}$ bits, with $N_{\max}$ an *a priori* upperbound on cardinalities. The estimate is (in the sense of mean values) asymptotically *unbiased*; the relative accuracy of the estimate (measured by a standard deviation) is close to $1.05/\sqrt{m}$ for our best version of the algorithm, Super–LogLog. For instance, estimating cardinalities till $N_{\max} = 2^{27}$ (a hundred million different records) can be achieved with $m = 2048$ memory units of 5 bits each, which corresponds to 1.28 kilobytes of auxiliary storage in total, the error observed being typically less than 2.5%. Since the algorithm operates incrementally and in a single pass it can be applied to data flows for which it provides on-line estimates available at any given time. Advantage can be taken

G. Di Battista and U. Zwick (Eds.): ESA 2003, LNCS 2832, pp. 605–617, 2003.

of the low memory consumption in order to gather *simultaneously* a *very large number of statistics* on huge heterogeneous data sets. The LogLog algorithm can also be *fully distributed* or *parallelized*, with *optimum* speed-up and *minimal* interprocess communication. Finally, an embedded hardware design would involve strictly minimal resources.

Motivations. A traditional application of cardinality estimates is database query optimization. There, a complex query typically involves a variety of set-theoretic operations as well as projections, joints, and so on. In this context, knowing "for free" cardinalities of associated sets provides a valuable guide for selecting an efficient processing strategy best suited to the data at hand. Even a problem as simple as merging two large files with duplicates can be treated by various combinations of sorting, straight merging, and filtering out duplicates (in one or both of the files); the cost function of each possible strategy is then determined by the number of records as well as by the cardinality of each file. Probabilistic estimation algorithms also find a use in large data recording and warehousing environments. There, the goal is to provide an approximate response in time that is orders-of-magnitude less than what computing an exact answer would require: see the description of the Aqua Project by Gibbons *et al.* in [8].

The analysis of traffic in routers, as already mentioned, benefits greatly of cardinality estimators—this is lucidly exposed by Estan *et al.* in [2,3]. Certain types of attacks ("denial of service" and "port scans") are betrayed by alarmingly high counts of certain characteristic events in routers. In such situations, there is usually not enough resource available to store and search on-line the very large number of events that take place even in a relatively small time window.

Probabilistic counting algorithms can also be used within other algorithms whenever the final answer is the cardinality of a large set and a small tolerance on the quality of the answer is acceptable. Palmer *et al.* [13] describe the use of such algorithms in an extensive connectivity analysis of the internet topology. For instance, one of the tasks needed there is to determine, for each distance h, the number of pairs of nodes that are at distance at most h in the internet graph. Since the graph studied by [13] has close to 300,000 nodes, the number of pairs to be considered is well over 10^{10}, upon which costly list operations must be performed by exact algorithms. In contrast an algorithm that would be, in the abstract, suboptimal can be coupled with adapted probabilistic counting techniques and still provide reliable estimates. In this way, the authors of [13] were able to extract extensive metric information on the internet graph by keeping a reduced collection of data that reside in core memory. They report a reduction in run-time by a factor of more than 400.

Algorithms. The LogLog algorithm is probabilistic. Like in many similar algorithms, the first idea is to appeal to a hashing function in order to randomize data and bring them to a form that resembles random (uniform, independent) binary data. It is this hashed data set that is distilled into cardinality estimates by the algorithm. Various algorithms perform various tests on the hashed data set, then compare "*observables*" to what probabilistic analysis predicts, and finally "deduce" a plausible value of the parameter of interest. In the case of

```
ghfffghfghgghgggggghghheehfhfhhgghghghhfgffffhhhiigfhhffgfiihfhhh
igigighfgihfffghigihghigfhhgeegeghgghhhgghhfhidiigihighihehhhfgg
hfgighigffghdieghhhggghhfghhfiiheffghghihifgggffihgihfggighgiiif
fjgfgjhhjiifhjgehgghfhhfhjhiggghghihigghhihihgiighgfhlgjfgjjjmfl
```

The LogLog Algorithm with $m = 256$ condenses the whole of Shakespeare's works to a table of 256 "small bytes" of 4 bits each. The estimate of the number of distinct words is here $n^\circ = 30897$ (true answer: $n = 28239$), i.e., a relative error of +9.4%.

LogLog counting, the observable should *only* be linked to cardinality, and hence be totally independent of the nature of replications and the ordering of data present in the file, on which no information at all is available. (Depending on context, collisions due to hashing can either be neglected or their effect can be estimated and corrected.)

Whang, Zanden, and Taylor [16] have developed *Linear Counting*, which distributes (hashed) values into buckets and only keeps a bitmap indicating which buckets are hit. Then observing the number of hits in the table leads to an estimate of cardinality. Since the number of buckets should not be much smaller than the cardinalities to be estimated (say, $\geq N_{\max}/10$), the algorithm has space complexity that is $O(N_{\max})$ (typically, $N_{\max}/10$ bits of storage). The linear space is a drawback whenever large cardinalities, multiple counts, or limited hardware are the rule. Estan, Varghese, and Fisk [3] have devised a multiscale version of this principle, where a hierarchical collection of small windows on the bitmap is kept. From simulation data, their *Multiresolution Bitmap* algorithm appears to be about 20% more accurate than Probabilistic Counting (discussed below) when the same amount of memory is used. The best algorithm of [3] for flows in routers, *Adaptive Bitmap*, is reported to be about 3 times more efficient than either Probabilistic Counting or Multiresolution Bitmap, but it has the disadvantage of not being *universal*, as it makes definite statistical assumptions ("stationarity") regarding the data input to the algorithm. (We recommend the thorough engineering discussion of [3].)

Closer to us is the *Probabilistic Counting* algorithm of Flajolet and Martin [7]. This uses a certain observable that has excellent statistical properties but is relatively costly to maintain in terms of storage. Indeed, Probabilistic Counting estimates cardinalities with an error close to $0.78/\sqrt{m}$ given a table of m "words", each of size about $\log_2 N_{\max}$.

Yet another possible idea is sampling. One may use any filter on hashed values with selectivity $p \ll 1$, store exactly and without duplicates the data items filtered and return as estimate $1/p$ times the corresponding cardinality. Wegner's *Adaptive Sampling* (described and analyzed in [5]) is an elegant way to maintain dynamically varying values of p. For m "words" of memory (where here "word" refers to the space needed by a data item), the accuracy is about $1.20/\sqrt{m}$, which is about 50% less efficient than Probabilistic Counting.

An insightful complexity-theoretic discussion of approximate counting is provided by Alon, Matias, and Szegedy in [1]. The authors discuss a class of "frequency–moments" statistics which includes ours (as their F_0 statistics). Our

LogLog Algorithm has principles that evoke some of those found in the intersection of [1] and the earlier [7], but contrary to [1], we develop here a complete eminently *practical* algorithmic solution and provide a very precise analysis, including bias correction, error and risk evaluation, as well as complete dimensioning rules.

We estimate that our LogLog algorithm outperforms the earlier Probabilistic Counting algorithm and the similarly performing Multiresolution Bitmap of [3] by a factor of 3 at least as it replaces "words" (of 16 to 32 bits) by "*small bytes*" of typically 5 bits each, while being based on an observable that has only slightly higher dispersion than the other two algorithms—this is expressed by our two formulæ $1.30/\sqrt{m}$ (LogLog) and $1.05/\sqrt{m}$ (super–LogLog). This places our algorithm in the same category as Adaptive Bitmap of [3]. However, compared to Adaptive Bitmap, the LogLog algorithm has the great advantage of being *universal* as it makes no assumptions on the statistical regularity of data. We thus believe LogLog and its improved version Super–LogLog to be the best general-purpose algorithmic solution currently known to the problem of estimating large cardinalities.

Note. The following related references were kindly suggested by a referee: Cormode *et al.*, in *VLDB*–2002 (a new counting method based on stable laws) and Bar-Yossef *et al.*, *SODA*–2002 (a new application to counting triangles in graphs).

2 The Basic LogLog Algorithm

In computing practice, one deals with a *multiset* of data items, each belonging to a discrete universe $\mathcal{U}$. For instance, in the case of natural text, $\mathcal{U}$ may be the set of all alphabetic strings of length ≤ 28 ('antidisestablishmentarianism'), double floats represented on 64 bits, and so on. A multiset $\mathfrak{M}$ of elements of $\mathcal{U}$ is given and the problem is to estimate its cardinality, that is, the number of *distinct* elements it comprises. Here is the principle of the basic LogLog algorithm.

Algorithm LogLog($\mathfrak{M}$: Multiset of hashed values; $m \equiv 2^k$)
Initialize $M^{(1)}, \ldots, M^{(m)}$ to 0;
let $\rho(y)$ be the rank of first 1-bit from the left in y;
  for $x = b_1 b_2 \cdots \in \mathfrak{M}$ do
    set $j := \langle b_1 \cdots b_k \rangle_2$ (value of first k bits in base 2)
    set $M^{(j)} := \max(M^{(j)}, \rho(b_{k+1} b_{k+2} \cdots)$;
return $E := \alpha_m m 2^{\frac{1}{m}\sum_j M^{(j)}}$ as cardinality estimate.

We assume throughout that a *hash function*, h, is available that transforms elements of $\mathcal{U}$ into sufficiently long binary strings, in such a way that bits composing the hashed value closely resemble random uniform independent bits. This pragmatic attitude[1] is justified by Knuth who writes in [10]: "*It is theoretically*

[1] The more theoretically inclined reader may prefer to draw h at random from a family of universal hash functions; see, e.g., the general discussion in [12] and the specific [1].

impossible to define a hash function that creates random data from non-random data in actual files. But in practice it is not difficult to produce a pretty good imitation of random data." Given this, we formalize our basic problem as follows.

> Take $\mathcal{U} = \{0,1\}^\infty$ as the universe of data endowed with the uniform (product) probability distribution. An *ideal multiset* $\mathfrak{M}$ of cardinality n is a random object that is produced by first drawing an n-sequence independently at random from $\mathcal{U}$, then replicating elements in an arbitrary way, and finally, applying an arbitrary permutation.
> The user is provided with the (extremely large) ideal multiset $\mathfrak{M}$ and its goal is to estimate the (unknown to him) value of n at a small computational cost. No information is available, hence no statistical assumption can be made, regarding the behaviour of the replicator-shuffler daemon.

(The fact that we consider infinite data is a convenient abstraction at this stage; we discuss its effect, together with needed adjustments, in Section 5 below.)

The basic idea consists in scanning $\mathfrak{M}$ and observing the patterns of the form $0^\star 1$ that occur at the beginning of (hashed) records. For a string $x \in \{0,1\}^\infty$, let $\rho(x)$ denote the position of its first 1-bit. Thus $\rho(1\cdots) = 1$, $\rho(001\cdots) = 3$, etc. Clearly, we expect about $n/2^k$ amongst the distinct elements of $\mathfrak{M}$ to have a ρ-value equal to k. In other words, the quantity,

$$R(\mathfrak{M}) := \max_{x \in \mathfrak{M}} \rho(x),$$

can reasonably be hoped to provide a rough indication on the value of $\log_2 n$. It is an "observable" in the sense above since it is totally independent of the order and the replication structure of the multiset $\mathfrak{M}$. In fact, in probabilistic terms, the quantity R is precisely distributed in the same way as 1 plus the maximum of n independent geometric variables of parameter $\frac{1}{2}$. This is an extensively researched subject; see, e.g., [14]. It turns out that R estimates $\log_2 n$ with an additive bias of 1.33 and a standard deviation of 1.87. Thus, in a sense, the observed value of R estimates "logarithmically" n within ± 1.87 binary orders of magnitude. Notice however that the expectation of 2^R is infinite so that 2^R cannot in fact be used to estimate n.

The next idea consists in separating elements into m groups also called "buckets", where m is a design parameter. With $m = 2^k$, this is easily done by using the first k bits of x as representing in binary the index of a bucket. One can then compute the parameter R on each bucket, after discarding the first k bits. If $M^{(j)}$ is the (random) value of parameter R on bucket number j, then the arithmetic mean $\frac{1}{m}\sum_{j=1}^{m} M^{(j)}$, can legitimately be expected to approximate $\log_2(n/m)$ plus an additive bias. The estimate of n returned by the LogLog algorithm is accordingly

$$E := \alpha_m m 2^{\frac{1}{m}\sum M^{(j)}}. \tag{1}$$

The constant α_m comes out of our later analysis as $\alpha_m := \left(\Gamma(-1/m)\frac{1-2^{1/m}}{\log 2}\right)^{-m}$, where $\Gamma(s) := \frac{1}{s}\int_0^\infty e^{-t} t^s \, dt$. It precisely corrects the systematic bias of the raw arithmetic mean in the asymptotic limit. One may also hope for a greater concentration of the estimates, hence better accuracy, to result from averaging over $m \gg 1$ values. The main characteristics

of the algorithm are summarized below in Theorem 1. The letters $\mathbb{E}, \mathbb{V}$ denote expectation and variance, and the subscript n indicates the cardinality of the underlying random multiset.

Theorem 1. *Consider the basic* LogLog *algorithm applied to an ideal multiset of (unknown) cardinality n and let E be the estimated value of cardinality returned by the algorithm.*

(*i*) *The estimate E is asymptotically unbiased in the sense that, as $n \to \infty$,*

$$\frac{1}{n}\mathbb{E}_n(E) = 1 + \theta_{1,n} + o(1), \qquad \text{where} \quad |\theta_{1,n}| < 10^{-6}.$$

(*ii*) *The standard error defined as $\frac{1}{n}\sqrt{\mathbb{V}_n(E)}$ satisfies as $n \to \infty$,*

$$\frac{1}{n}\sqrt{\mathbb{V}_n(E)} = \frac{\beta_m}{\sqrt{m}} + \theta_{2,n} + o(1), \qquad \text{where} \quad |\theta_{2,n}| < 10^{-6}.$$

One has: $\beta_{128} \doteq 1.30540$, $\quad \beta_\infty = \sqrt{\frac{1}{12}\log^2 2 + \frac{1}{6}\pi^2} \doteq 1.29806$.

In summary, apart from completely negligible fluctuations whose amplitude is less than 10^{-6}, the algorithm provides asymptotically a valid estimator of n. The standard error, which measures in a mean-quadratic sense and in proportion to n the deviations to be expected, is closely approximated by the formula[2]

$$\text{Standard error} \approx \frac{1.30}{\sqrt{m}}.$$

For instance, $m = 256$ and $m = 1024$ give a standard error of 8% and 4% respectively. (These figures are compatible with what was obtained on the Shakespeare data.) Observe also that $\alpha_m \sim \alpha_\infty - (2\pi^2 + \log^2 2)/(48m)$, where $\alpha_\infty = e^{-\gamma}\sqrt{2}/2 \doteq 0.39701$ (γ is Euler's constant), so that, in practical implementations, α_m can be replaced by α_∞ without much detectable bias as soon as $m \geq 64$.

The proof of Theorem 1 will occupy the whole of the next section.

3 The Basic Analysis

Throughout this note, the *unknown* number of distinct values in the data set is denoted by n. The LogLog algorithm provides an estimator, E, of n. We first provide formulæ for the expectation and variance of E. Asymptotic analysis is performed next: The Poissonization paragraph introduces the Poisson model where n is allowed to vary according to a Poisson law, while the Depoissonization paragraph shows the Poisson model to be asymptotically equivalent to the "fixed–n" model that we need. The expected value of the estimator is found to be asymptotically n, up to minute fluctuations. This establishes the asymptotically unbiased character of the algorithm as asserted in (*i*) of Theorem 1. The standard deviation of the estimator is also proved to be of the order of n with the proportionality coefficient providing the value of the standard error, hence the accuracy of the algorithm, as asserted in (*ii*) of Theorem 1.

[2] We use '$\sim$' to denote asymptotic expansions in the usual mathematical sense and reserve the informal '$\approx$' for "approximately equal".

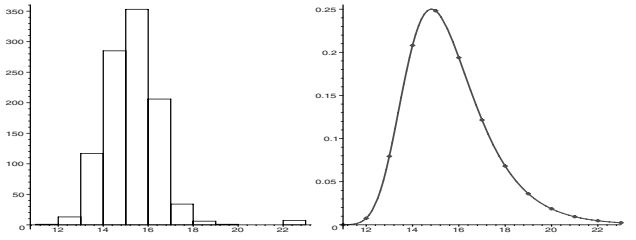

Fig. 1. The distribution of observed register values for the Pi file, $n \approx 2 \cdot 10^7$ with $m = 1024$ [left]; the distribution $\mathbb{P}_\nu(M = k)$ of a register M, for $\nu = 2 \cdot 10^4$ [right].

We start by examining what happens in a bucket that receives ν elements (Figure 1). The random variable M is, we recall, the maximum of ν random variables that are independent and geometrically distributed according to $\mathbb{P}(Y \geq k) = \frac{1}{2^{k-1}}$. Consequently, the probability distribution of M is characterized by $\mathbb{P}_\nu(M \leq k) = \left(1 - \frac{1}{2^k}\right)^\nu$, so that $\mathbb{P}_\nu(M = k) = \left(1 - \frac{1}{2^k}\right)^\nu - \left(1 - \frac{1}{2^{k-1}}\right)^\nu$. The bivariate (exponential) generating function of this family of probability distributions as ν varies is then

$$G(z, u) := \sum_{\nu,k} \mathbb{P}_\nu(M = k) u^k \frac{z^\nu}{\nu!}. = \sum_k u^k \left(e^{z(1-1/2^k)} - e^{z(1-1/2^{k-1})} \right), \qquad (2)$$

as shown by a simple calculation. The starting point of the analysis is an expression in terms of G of the mean and variance of $Z := E/\alpha_m \equiv m 2^{\frac{1}{m}\sum_j M^{(j)}}$, which is the unnormalized version of the estimator E. With the expression $[z^n]f(z)$ representing the coefficient of z^n in the power series $f(z)$, we state:

Lemma 1. *The expected value and variance of the unnormalized estimator Z are $\mathbb{E}_n(Z) = mn![z^n]G\left(\frac{z}{m}, 2^{1/m}\right)^m$, and*

$$\mathbb{V}_n(Z) = m^2 n![z^n]\left(G\left(\tfrac{z}{m}, 2^{2/m}\right)\right)^m - \left(mn![z^n]G\left(\tfrac{z}{m}, 2^{1/m}\right)^m\right)^2$$

Proof. The multinomial convolution relations corresponding to mth powers of generating functions imply that $n![z^n]G(z/m, u)^m$ is the probability generating function of $\sum_j M^{(j)}$. (The multinomials enumerate all ways of distributing elements amongst buckets.) The expressions for the first and second moment of Z are obtained from there by substituting $u \mapsto 2^{1/m}$ and $u \mapsto 2^{2/m}$.

Proving Theorem 1 is reduced to estimating asymptotically these quantities.

Poissonization. We "poissonize" the problem of computing the expected value and the variance. In this way, calculations take advantage of powerful properties of the Mellin transform. The Poisson law of rate λ is the law of a random variable X such that $\mathbb{P}(X = \ell) = e^{-\lambda}\frac{\lambda^\ell}{\ell!}$. Given a class $\mathcal{M}_s$ of probabilistic models indexed by integers s, poissonizing means considering the "supermodel" where model $\mathcal{M}_s$ is chosen according to a Poisson law of rate λ. Since the poisson model of a large parameter λ is predominantly a mixture of models $\mathcal{M}_s$ with s near λ (the Poisson law is "concentrated" near its mean), one can expect

properties of the fixed-n model $\mathcal{M}_n$ to be reflected by corresponding properties of the Poisson model taken with rate $\lambda = n$.

A useful feature is that expressions of moments and probabilities under the Poisson model are closely related to exponential generating functions of the fixed-n models. This owes to the fact that if $f(z) = \sum_n f_n z^n/n!$ is the exponential generating function of expectations of a parameter, then the quantity $e^{-\lambda} f(\lambda) = \sum_n f_n e^{-\lambda} \frac{\lambda^n}{n!}$ gives the corresponding expectation under the Poisson model. In this way, one sees that the quantities $\mathcal{E}_n = mG\left(\frac{n}{m}, 2^{1/m}\right)^m e^{-n}$ and $\mathcal{V}_n = m^2 G\left(\frac{n}{m}, 2^{2/m}\right)^m e^{-n} - \left(mG\left(\frac{n}{m}, 2^{1/m}\right)^m e^{-n}\right)^2$ are respectively the mean and variance of Z when the cardinality of the underlying multiset obeys a Poisson law of rate $\lambda = n$.

Lemma 2. *The Poisson mean and variance $\mathcal{E}_n$ and $\mathcal{V}_n$ satisfy as $n \to \infty$:*

$$\mathcal{E}_n \sim \left[\left(\Gamma(-1/m)\frac{1-2^{1/m}}{\log 2}\right)^m + \epsilon_n\right] \cdot n$$

$$\mathcal{V}_n \sim \left[\left(\Gamma(-2/m)\frac{1-2^{2/m}}{\log 2}\right)^m - \left(\Gamma(-1/m)\frac{1-2^{-1/m}}{\log 2}\right)^{2m} + \eta_n\right] \cdot n^2.$$

where $|\epsilon_n|$ and $|\eta_n|$ are bounded by 10^{-6}.

The proof crucially relies on the Mellin transform [6].

Depoissonization. Finally, the asymptotic forms of the first two moments of the LogLog estimator can be transferred back from the Poisson model to the fixed-n model that underlies Theorem 1. The process involved is known as "depoissonization". Various options are discussed in Chapter 10 of Szpankowski's book [15]. We choose the method called "analytic depoissonization" by Jacquet and Szpankowski, whose underlying engine is the saddle point method applied to Cauchy integrals; see [9,15]. In essence, the *values* of an exponential generating function at large arguments are closely related to the asymptotic form of its *coefficients* provided the generating function decays fast enough away from the positive real axis in the complex plane. The complete proof is omitted.

Lemma 3. *The first two moments of the* LogLog *estimator are asymptotically equivalent under the Poisson and fixed–n model: $\mathbb{E}_n(Z) \sim \mathcal{E}_n$, and $\mathbb{V}_n(Z) \sim \mathcal{V}_n$.*

Lemmas 2 and 3 together prove Theorem 1. Easy numerical calculations and straight asymptotic analysis of β_m conclude the evaluations stated there.

4 Space Requirements

Now that the correctness—the absence of bias as well as accuracy—of the basic LogLog algorithm has been established, there remains to see that it performs as promised and only consumes $O(\log\log n)$ bits of storage if counts till n are needed[3].

[3] A counting algorithm exhibiting a log-log feature in a different context is Morris's *Approximate Counting* [11] analyzed in [4].

In its abstract form of Section 1, the LogLog algorithm operates with potentially unbounded integer registers and it consumes m of these. What we call an ℓ–restricted algorithm is one in which each of the $M^{(j)}$ registers is made of ℓ bits, that is, it can store any integer between 0 and $2^\ell - 1$. We state a shallow result only meant to phrase mathematically the log-log property of the basic space complexity:

Theorem 2. *Let $\omega(n)$ be a function that tends to infinity arbitrarily slowly and consider the function $\ell(n) = \log_2 \log_2 \left(\frac{n}{m}\right) + \omega(n)$. Then, the $\ell(n)$–restricted algorithm and the* LogLog *algorithm provide the same output with probability tending to 1 as n tends to infinity.*

The auxiliary tables maintained by the algorithm then comprise m "small bytes", each of size $\ell(n)$. In other words, the total space required by the algorithm in order to count till n is $m \log_2 \log_2 \left(\frac{n}{m}\right) (1 + o(1))$. The hashing function needs to hash values from the original data universe onto exactly $2^{\ell(n)} + \log_2 m$ bits. Observe also that, whenever no discrepancy is present at the value n itself, the restricted algorithm automatically provides the right answer for all values $n' \leq n$.

The proof of this theorem results from tail properties of the multinomial distributions and of maxima of geometric random variables.

Assume for instance that we wish to count cardinalities till 2^{27}, that is, over a hundred million, with an accuracy of about 4%. By Theorem 1, one should adopt $m = 1024 = 2^{10}$. Then, each bucket is visited roughly $n/m = 2^{17}$ times. One has $\log_2 \log_2 2^{17} \doteq 4.09$. Adopt $\omega = 0.91$, so that each register has a size of $\ell = 5$ bits, i.e., a value less than 32. Applying the upperbound of the overall probability failure shows that an ℓ–restriction will have little incidence on the result: the probability of a discrepancy[4] is lower than 12%. In summary: *The basic* LogLog *counting algorithm makes it possible to estimate cardinalities till 10^8 with a standard error of 4% using 1024 registers of 5 bits each, that is, a table of 640 bytes in total.*

5 Algorithmic Engineering

In this section, we describe a concrete implementation of the LogLog algorithm that incorporates the probabilistic principles seen in previous sections. At the same time, we propose an optimization that has several beneficial effects: (*i*) it increases at no extra cost the accuracy of the results, i.e., it decreases the dispersion of the estimates around the mean value; (*ii*) it allows for the use of smaller register values, thereby improving the storage utilization of the algorithm and nullifying the effect of length restriction discussed in Section 4.

The fundamental probability distribution is that of the value of the M–register in a bucket that receives ν elements (where $\nu \approx n/m$). This is the

[4] In addition, a correction factor, calculated according to the principles of Section 3, could easily be built into the algorithm, in order to compensate the small bias induced by restriction

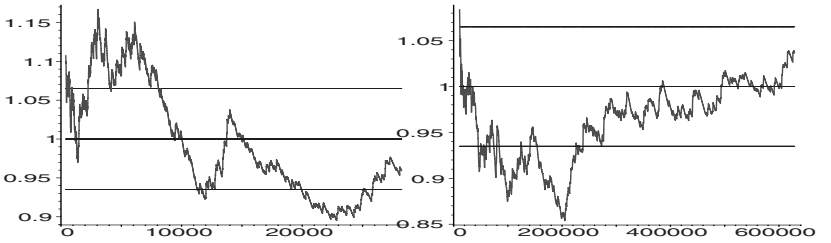

Fig. 2. The evolution of the estimate (divided by the current value of n) provided by super–LogLog on all of Shakespeare's works: (left) words; (right) pairs of consecutive words. Here $m = 256$ (standard error=6.5%).

maximum of ν geometric random variables with mean close to $\log_2 n$. The tails of this distribution, though exponential, are still relatively "soft", as there holds $\mathbb{P}_\nu(M > \log_2 \nu + k) \approx 2^{-k}$. Since the estimate returned involves an exponential of the arithmetic mean of bucket registers, a few exceptional values may still distort the estimate produced by the algorithm, while more tame data will not induce this effect. Altogether, this phenomenon lies at the origin of a natural dispersion of estimates produced by the algorithm, hence it places a limit on the accuracy of cardinality estimates. A simple remedy to the situation consists in using *truncation*:

> **Truncation Rule**. When collecting register values in order to produce the final estimate, retain only the $m_0 := \lfloor \theta_0 m \rfloor$ smallest values and discard the rest. There θ_0 is a real number between 0 and 1, with $\theta_0 = 0.7$ producing near-optimal results. The mean of these registers is computed and the estimate returned is $m_0 \widetilde{\alpha}_m 2^{\frac{1}{m_0} \Sigma^\star M^{(j)}}$, where $\Sigma^\star$ indicates the truncated sum. The modified constant $\widetilde{\alpha}_m$ ensures that the algorithm remains unbiased.

When the truncation rule is applied, accuracy does increase. An empirically determined formula for the standard error is $\frac{1.05}{\sqrt{m}}$, when the Truncation Rule with $\theta_0 = 0.7$ is employed.

Empirical studies justify the fact that register values may be ceiled at the value $\lceil \log_2 \left(\frac{n}{m}\right) \rceil + \delta$, without detectable effect for $\delta = 3$. In other words, one may freely combine the algorithm with restriction as follows:

> **Restriction Rule**. Use register values that are in the interval $[0\,..B]$, where $\left\lceil \log_2 \left(\frac{N_{\max}}{m}\right) + 3 \right\rceil \leq B$.

For instance for the data at the end of Section 4, with $n = 2^{27}$, $m = 1024$, the value $B = 20$ (encoded on 5 bits) is sufficient. But now, the probability that length-restriction affects the estimate of the algorithm drops tremendously.

> **Fact 1**. *Combining the basic* LogLog *counting algorithm, the Truncation Rule and the Restriction Rule yields the super-*LogLog *algorithm that estimates cardinalities with a standard error of* $\approx \frac{1.05}{\sqrt{m}}$ *when* m *"small bytes" are used. Here a small byte has size* $\left\lceil \log_2 \left\lceil \log_2 \left(\frac{N_{\max}}{m}\right) + 3 \right\rceil \right\rceil$*, that is, 5 bits for maximum cardinalities* $N_{\max}$ *well over* 10^8.

Length of the hash function and collisions. The length H of the hash function—how many bits should it produce?— is guided by previous considerations. There must be $\log_2 m$ bits reserved for bucketing and the bound on register values should be at least as large as the quantity B above. Accordingly this value H must satisfy: $H \geq H_0$, where $H_0 := \log_2 m + \lceil \log_2 \left(\frac{N_{\max}}{m}\right) + 3 \rceil$. In case a value too close to H_0 is adopted (say $0 \leq H - H_0 \leq 3$), then the effect of hashing collisions must be compensated for. This is achieved by inverting the function that gives the expected value of the number of collisions in a hash table (see [3,16] for an analogous discussion). The estimator is then to be changed into $-2^H \log\left(1 - \frac{\widetilde{\alpha}_m m}{2^H} 2^{\frac{1}{m}\sum^\star M^{(j)}}\right)$. (No detectable degradation of performance results from the last modification of the estimator function, and it can safely be used in all cases.)

Risk analysis. For the pure LogLog algorithm, the estimate is an empirical mean of random variables that are approximately identically distributed (up to statistical fluctuations in bucket sizes). From there, it can be proved that the quantity $\frac{1}{m}\sum_j M^{(j)}$ is numerically closely approximated by a Gaussian. Consequently, the estimate returned is very roughly Gaussian: at any rate, it has exponentially decaying tails. (In principle, a full analysis would be feasible.) A similar property is expected for the super-LogLog algorithm since it is based on the same principles. As a consequence, we obtain the following pragmatic conclusion:

Fact 2. *Let $\sigma := \frac{1.05}{\sqrt{m}}$. The estimate is within σ, 2σ, and 3σ of the exact value of the cardinality n in respectively 65%, 95%, and 99% of the cases.*

6 Conclusions

That super–LogLog performs quite well in practice is confirmed by the following data from simulations:

$k = \log_2 m$	4	5	6	7	8	9	10	11	12
$\sigma^\star$	29.5	19.8	13.8	9.4	6.5	4.5	3.1	2.2	1.5
$1.05/\sqrt{m}$	26.3	18.6	13.1	9.3	6.5	4.6	3.3	2.3	1.6
Random	22	16	11	8	6	4	3	2.3	2
KingLear	8.2	1.6	2.1	3.9	2.9	1.2	0.3	1.7	—
ShAll	2.9	13.9	4.4	0.9	9.4	4.1	3.0	0.8	0.6
Pi	67	28	9.7	8.6	2.8	5.1	1.9	1.2	0.7

Note. $\sigma^\star$ refers to standard error as estimated from extensive simulations, to be compared to the empirical formula $1.05/\sqrt{m}$. The next lines display the *absolute value* of the relative error measured. Random refers to averages over 10,000 runs with $n = 20,000$; the other data are *single* runs: Pi is formed of $2 \cdot 10^7$ records that are consecutive 10–digit slices of the first 200 million decimals of π; ShAll is the whole of Shakespeare's works. KingLear is what its name says. (Naturally, inherent stochastic fluctuations prevent the estimates from always depending

monotonically on memory size (m) in the case of single runs on a given piece of data.)

As we have strived to demonstrate, the LogLog algorithm in its optimized version performs quite well. The following table (grossly) summarizes the accuracy (measured by standard error σ) in relation to the storage used for the major methods known. Note that different algorithms operate with different memory units.

Algorithm	*Std. Err.* (σ)	*Memory units*	$n = 10^8$, $\sigma = 0.02$
Adaptive Sampling	$1.20/\sqrt{m}$	Records ($\geq$24–bit words)	10.8 kbytes
Prob. Counting	$0.78/\sqrt{m}$	Words (24–32 bits)	6.0 kbytes
Multires. Bitmap	$\approx 4.4/\sqrt{m}$	Bits	4.8 kbytes
LogLog	$1.30/\sqrt{m}$	"Small bytes" (5 bits)	2.1 kbytes
Super-LogLog	$1.05/\sqrt{m}$	"Small bytes" (5 bits)	1.7 kbytes

The last column is a rough indication of the storage requirement for an accuracy of 2% and a file of cardinality 10^8. (The formula for Multiresolution Bitmap is a crude extrapolation based on data of [3].)

Distributing or parallelizing the algorithm is trivial: it suffices to have different processors (sharing the same hash function) operate on different slices of the data and then "max–merge" their tables of registers. Optimal speed-up is clearly attained and interprocess communication is limited to just a few kilobytes. Requirements for an embedded hardware design are absolutely minimal as only addressing, register comparisons, and integer addition are needed.

Acknowledgements. This work has been partly supported by the European Union under the Future and Emerging Technologies programme of the Fifth Framework, Alcom-ft Project IST-1999-14186. The authors are grateful to Cristian Estan and George Varghese for very liberally sharing ideas and preliminary versions of their works, and to Keith Briggs for his suggestions regarding implementation.

References

1. Alon, N., Matias, Y., and Szegedy, M. The space complexity of approximating the frequency moments. *Journal of Computer and System Sciences 58* (1999), 137–147.
2. Estan, C., and Varghese, G. New directions in traffic measurement and accounting. In *Proceedings of SIGCOMM 2002* (2002), ACM Press. (Also: UCSD technical report CS2002-0699, February, 2002; available electronically.).
3. Estan, C., Varghese, G., and Fisk, M. Bitmap algorithms for counting active flows on high speed links. Technical Report CS2003-0738, UCSD, Mar. 2003.
4. Flajolet, P. Approximate counting: A detailed analysis. *BIT 25* (1985), 113–134.
5. Flajolet, P. On adaptive sampling. *Computing 34* (1990), 391–400.
6. Flajolet, P., Gourdon, X., and Dumas, P. Mellin transforms and asymptotics: Harmonic sums. *Theoretical Computer Science 144*, 1-2 (1995), 3–58.

7. Flajolet, P., and Martin, G. N. Probabilistic counting algorithms for data base applications. *Journal of Computer and System Sciences 31*, 2 (1985), 182–209.
8. Gibbons, P. B., Poosala, V., Acharya, S., Bartal, Y., Matias, Y., Muthukrishnan, S., Ramaswamy, S., and Suel, T. AQUA: System and techniques for approximate query answering. Tech. report, Bell Laboratories, Murray Hill, New Jersey, Feb. 1998.
9. Jacquet, P., and Szpankowski, W. Analytical depoissonization and its applications. *Theoretical Computer Science 201*, 1–2 (1998).
10. Knuth, D. E. *The Art of Computer Programming*, 2nd ed., vol. 3: Sorting and Searching. Addison-Wesley, 1998.
11. Morris, R. Counting large numbers of events in small registers. *Communications of the ACM 21* (1978), 840–842.
12. Motwani, R., and Raghavan, P. *Randomized Algorithms*. Cambridge University Press, 1995.
13. C. R. Palmer, G. Siganos, M. Faloutsos, C. Faloutsos, and P. Gibbons. The connectivity and fault-tolerance of the Internet topology. In *Workshop on Network-Related Data Management* (NRDM-2001).
14. Prodinger, H. Combinatorics of geometrically distributed random variables: Left-to-right maxima. *Discrete Mathematics 153* (1996), 253–270.
15. Szpankowski, W. *Average-Case Analysis of Algorithms on Sequences*. John Wiley, New York, 2001.
16. Whang, K.-Y., Zanden, B. T. V., and Taylor, H. M. A linear-time probabilistic counting algorithm for database applications. *TODS 15*, 2 (1990), 208–229.