Counting by Coin Tossings

Philippe Flajolet

Algorithms Project, INRIA-Rocquencourt, 78153 Le Chesnay (France)
Philippe.Flajolet@inria.fr

Abstract. This text is an informal review of several randomized algorithms that have appeared over the past two decades and have proved instrumental in extracting efficiently quantitative characteristics of very large data sets. The algorithms are by nature probabilistic and based on hashing. They exploit properties of simple discrete probabilistic models and their design is tightly coupled with their analysis, itself often founded on methods from analytic combinatorics. Singularly efficient solutions have been found that defy information theoretic lower bounds applicable to deterministic algorithms. Characteristics like the total number of elements, cardinality (the number of distinct elements), frequency moments, as well as unbiased samples can be gathered with little loss of information and only a small probability of failure. The algorithms are applicable to traffic monitoring in networks, to data base query optimization, and to some of the basic tasks of data mining. They apply to massive data streams and in many cases require strictly minimal auxiliary storage.

1 Approximate Counting.

Assume a blind man (a computer?) wants to locate a single black sheep amongst a flock of $N-1$ white sheep. The man can only ask an assistant questions with a Yes/No answer. Like for instance: "*Tell me whether the black sheep is amongst sheep ranked between 37 and 53 from the left*". This is a variation on the theme of "Twenty Questions". Clearly, about $\log_2 N \equiv \lg N$ operations are both necessary and sufficient. (The proof relies on the fact that with ℓ bits of information, you can only distinguish between 2^ℓ possibilities, so that one must have $2^\ell \geq N$, hence $\ell \geq \lg N$.) This simple argument is of an information-theoretic nature. It implies the fact that one cannot keep a counter (the "black sheep") capable of recording counts between 1 and N with less than $\lg N$ bits.

Assume that we want to run a counter known *a priori* to be in the range $1\,..\,N$. Can one beat the information-theoretic lower bound? Yes and No! Not if we require an exact count as this would contradict the information theoretic argument. But, ... Say we relax the constraints and tolerate an uncertainty on the counts of at most 10% (say) in relative terms. The situation changes dramatically. We now just need to locate our count amongst the terms of a geometric scale, $1, A, A^2, A^3 \ldots$ (till N), where $A = 1.1$. The problem then becomes that of finding an interval in a collection of about $\log_A N$ intervals. Information theory then tells us that this cannot be effected in fewer than

$$\lg \log_A N \approx \lg \lg N + 2.86245$$

bits, but it also tells us that the "amount of information" contained in an approximate answer is of that very same order. For instance, it is conceivable that an algorithm could exist with which counts till $N = 2^{16} \equiv 65536$ can be maintained using $8 + 3 = 11$ bits instead of 16.

This is the situation which Robert Morris encountered at Bell Labs in 1977. He needed to maintain the logs of a very large number of events in small registers, since the space available at the time was too small to allow exact counts to be kept. A gain by a factor of 2 in memory against a degradation in accuracy by some 30% was perfectly acceptable. How to proceed? Information theory provides a theoretical possibility, not a solution.

Morris's solution [23], known as the Approximate Counting Algorithm, goes at follows. In its crudest (binary) version, you maintain a counter K that initially receives the value $K := 0$. Counter K is meant to be a logarithmic counter, in the sense that when the exact count is n, the value K of the counter at that time should be close to $\lg n$. Note that the single value that gets stored is K, which itself only requires $\lg K \approx \lg \lg n$ bits. Morris' ingenious idea consists in updating the counter K to its new value $K^\star$ according to the following procedure:

$$K^\star = K + 1 \quad \text{with probability } 2^{-K}; \qquad K^\star = K \quad \text{with probability } 1 - 2^{-K}.$$

As time goes, the counter increases, but at a smaller and smaller pace. Implementation is especially easy given a (pseudo)random number generator of sorts [21].

The notable fact here is the appeal to a *probabilistic* idea in order to increment the counter. A plausible argument for the fact that K at time n should be close to $\lg n$ is the fact that it takes 1 impulse for the counter to go from value 0 to value 1, then on average 2 more impulses to from 1 to 2, then on average 4 more impulses from 2 to 3, and so on. In other words, a value $K = \kappa$ should be reached after about

$$1 + 2 + 4 + \cdots + 2^{\kappa-1} = 2^\kappa - 1$$

steps. Thus, provided this informal reasoning is correct, one should have the numeric and probabilistic approximation $2^\kappa \approx n$. Then, the algorithm should return at each instant $n^\circ = 2^K$ as an estimate of the current value of n.

We have just exposed the binary version of the algorithm, which can at best provide an estimate within a factor of 2 since the values it returns are by design restricted to powers of 2. However, it is easy to change the *base* of the counter: it suffices to replace 2 by a smaller number q typically chosen of the form $q = 2^{1/r}$. Then, the new counter gets updated at basically r times the rate of the binary counter. Its granularity is improved, as is, we may hope, the accuracy of the result it provides.

So far, we have offered hand-waving arguments to justify the plausible effectiveness of the algorithm. It can be proved indeed for the base q algorithm that $n^\circ := (q^K - 1)/(q - 1) + 1$ is strictly an *unbiased estimator* of the unknown quantity n [the expected value of n° is n exactly], and that the chances

of a *large deviation* of the counter are small. (For instance, with a fair probability, its deviation will not exceed twice the accuracy for which it was designed.) The mathematical problem is not totally obvious since probabilistic decisions pile one upon another. A complete analysis of the states of Morris' Approximate Counting Algorithm was first published by the author in 1985; see [11]. The analysis combines regular languages from theoretical computer science, pure birth processes, basic calculus, Euler's partition identities, some identities due to the German mathematician Heine in the nineteenth century (as later noted by Prodinger [24]), as well as a use of the Mellin transform [13] otherwise familiar from pure and applied mathematics.

The abstract process underlying Morris' algorithm has a neat formulation. It can be described as a stochastic progression in an exponentially hardening medium. It appears to be closely related to recent probabilistic studies by Bertoin-Biane-Yor [4] and by Guillemin-Robert-Zwart [17], the latter relative to the transaction control protocol TCP—the additive-increase multiplicative-decrease process known as AIMD. Other connections include the space efficient simulation of non-deterministic computations by probabilistic devices, as was observed by Freivalds already in the late 1970's. Interestingly enough, the pure-birth process underlying Approximate Counting also surfaces in the analysis of a transitive closure algorithm for acyclic graphs; see Andrews-Crippa-Simon [2]. Such unexpected connections between various areas of the mathematical sciences are fascinating.

2 Probabilistic and Log-Log Counting

Admittedly, nowadays, memory is not such a major concern, so that one might be tempted to regard Morris' algorithm as a somewhat old-fashioned curiosity. This is not so. Morris' ideas have had an important rôle by demonstrating that complexity barriers can be bypassed by means of probabilistic techniques, as long as a small tolerance on the quality of results is granted.

Consider now information flowing at a very high rate in a network. It is impossible to store it and we do not have time to carry much computation on the fly as we see messages or packets passing by. Can one still extract some *global information* out of the flux of data? An example, which has motivated much of the recent work of Estan, Varghese, and others is as follows. Imagine you have a large volume of data (say, packet headers to fix ideas). Is it possible to *estimate* the number of *distinct* elements? This is an important question in the context of network security [7, 9] since several attacks may be detected at router's level by the fact that they generate an unusual number of *distinct* open connections (i.e., source-destination pairs also known as "flows").

Clearly, the solution that stores everything and sorts data afterwards is far too costly in terms of both processing time and storage consumption. An elegant solution is provided by a technique combining *hashing* and *signatures*. Say we have a *multiset*[1] $\mathfrak{M}$ of elements and call *cardinality* the number of distinct ele-

[1] A multiset is like a set but with repetitions allowed.

ments $\mathfrak{M}$ contains. Suppose we have a "good" hash function h, which is assumed to provide long enough strings of pseudorandom bits from actual data and neglect the effect of collisions. Then the collection $h(\mathfrak{M})$ has repetitions that are of the *very same* structure as those of $\mathfrak{M}$ itself, safe for the important remark that one may now assume the elements to be random real numbers uniformly drawn on the interval $[0, 1]$.

Let us now examine the elements once hashed. Given the uniformity assumption, we expect the following patterns in values to be observed, with the corresponding frequencies as described in this table:

$.1\cdots$	$.01\cdots$	$.001\cdots$	$.0001\cdots$
$\frac{1}{2}$	$\frac{1}{4}$	$\frac{1}{8}$	$\frac{1}{16}$

Let $\rho(x)$ be the rank of the first bit 1 in the representation of x (its hashed value, rather). If hashed values are uniform and independent, the event $\rho(x) = k$ will occur with probability $\frac{1}{2^k}$. The quantity,

$$R := \max_{k \in \mathbb{N}} \Big[\rho = 1,\, \rho = 2, \ldots,\, \rho = k \text{ are } \textit{all} \text{ observed} \Big], \tag{1}$$

is then expected to provide a good indication of the *logarithm* of the unknown cardinality of $\mathfrak{M}$ and 2^R should, roughly speaking, estimate the cardinality $n = \|\mathfrak{M}\|$. Note that the algorithm needs to maintain a bitmap table, which records on the fly the values of ρ that are observed. (A *word* of 32 bits is typically adequate.) By construction, the algorithm depends only on the underlying *set* and it is in no way affected by replications of records: whether an element occurs once or a million times still results in the same operation of setting a flag in the bitmap to 1.

The complete algorithm was published in the period 1983–1985 by Flajolet and Martin [14, 15]. The motivation at the time was query optimization in database systems. Indeed, such data tend to have a lot of replications (think of the collection of towns in which employees of an organization reside). In this context, computing an intersection of two multisets $\mathfrak{A} \cap \mathfrak{B}$ (towns of your own company's employees *and* of employees of your partner's company) benefits greatly of knowing how many *distinct* elements $\mathfrak{A}$ and $\mathfrak{B}$ may have since then the suitable nearly optimal strategy may be set up. (Options include sorting, merging, hashing, looking up elements of $\mathfrak{A}$ in $\mathfrak{B}$ after an index has been set up, and so on).

With respect to the crude binary version outlined above, the actual algorithm named Probabilistic Counting differs in two respects:

— A general technique known as *stochastic averaging* emulates the effect of m simultaneous observations at the sole expense of a hash function calculation and a simple "switch". The purpose is to increase the accuracy from about one binary order of magnitude for a single bitmap to $O(1/\sqrt{m})$ for stochastic averaging. The idea consists in computing the basic observable (1) in each of the $m = 2^\ell$ groups determined by the first ℓ bits of hashed values, then averaging, and finally scaling the estimate by m.

— The algorithm as described so far would be *biased*. A very precise probabilistic analysis (using inclusion-exclusion, generating functions, and Mellin transforms) makes it possible to effect the proper corrections and devise an algorithm that is *asymptotically unbiased*.

Without entering into the arcanes of the analysis, the reader can at least get a feeling of what goes on by contemplating the magical constant [10, p. 437],

$$\varphi := \frac{e^\gamma}{\sqrt{2}} \prod_{m=1}^{\infty} \left(\frac{2m+1}{2m} \right)^{\varepsilon(m)} \doteq 0.7735162909,$$

where γ is Euler's constant and $\varepsilon(m) = \pm 1$ indicates the parity of the number of 1–bits in the binary representation of m. This constant, provided by a Mellin analysis, enters the design of the algorithm as it corrects a multiplicative bias inherent in the raw parameter (1). In this way, the following proves to be possible: *To estimate the cardinality of large multisets (up to over a billion distinct elements) using* m words *of auxiliary memory, with a relative accuracy close to*

$$\alpha = \frac{0.78}{\sqrt{m}}. \tag{2}$$

Marianne Durand and I recently realized that one could even do a bit better, namely: *To estimate the cardinality of large multisets (up to several billion distinct elements) using* m short bytes *(*< 8 *bits) of auxiliary memory, with a relative accuracy close to*

$$\alpha = \frac{1.30}{\sqrt{m}}. \tag{3}$$

Rather than maintaining a bitmap table (of one word) out of which the new observable (previously R) is computed, we choose as "observable" an integer parameter of the file:

$$S := \max_{x \in \mathfrak{M}} \rho(x).$$

This quantity is stored in binary, so that only $\lg S$ bits are needed. Since S is itself a logarithmic estimator, its storage only requires $\lg \lg N$ bits. Now, with a *full byte* (8 bits), one could estimate cardinalities till about $2^{2^8} \approx 10^{77}$, which is of the order of the number of particles in the universe. Taking each memory unit to be of 5 bits then suffices for counts till $N = 2^{2^5} \approx 4 \cdot 10^9$, i.e., four billions. When experiments are repeated, or rather "emulated" by means of stochastic averaging, precision increases (in proportion to $1/\sqrt{m}$ if m is the number of memory units). This gives rise to the LogLog Counting algorithm. There is a slight theoretical loss in accuracy as the constant 0.78 of (2) relative to Probabilistic Counting is replaced by a slightly larger value, the formulæ being

$$\alpha = \frac{1.30}{\sqrt{m}} \qquad \text{and} \qquad \alpha = \frac{1.05}{\sqrt{m}}, \tag{4}$$

depending on implementation choices. (The first formula repeats (3); the second one is relative to the variant known as SuperLogLog). However this effect is

```
ghfffghfghgghgggggghghheehfhfhhgghghghhfgffffhhhiigfhhffgfiihfhhh
igigighfgihfffghigihghigfhhgeegeghgghhhgghhfhidiigihighihehhhfgg
hfgighigffghdieghhhgggghhfghhfiiheffghghihifgggffihgihfggighgiiif
fjgfgjhhjiifhjgehgghfhhfhjhiggghghihigghhihihgiighgfhlgjfgjjjmfl
```

Fig. 1. 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 in this run is $n^\circ = 30897$ (the true answer is $n = 28239$), which represents a relative error of +9.4%.

completely offset by the fact that LogLog's memory units (from words to small bytes) are smaller by a factor of about 4 than the words that Probabilistic Counting requires. Globally, *Probabilistic Counting is about 3 to 5 times more accurate than Probabilistic Counting, for a fixed amount of global storage.*

In summary, LogLog counting creates a signature from a multiset of data and then deduces a cardinality estimator. That signature only depends on the set of distinct values underlying the multiset of data input to the algorithm. It consists of m small-byte registers, resulting in an accuracy of $\approx 1/\sqrt{m}$. For instance, $m = 1024$ corresponds to a typical accuracy of 3% and its maintenance necessitates under a kilobyte of auxiliary memory. In Figure 1 (taken from [6]), we give explicitly the four line signature by which the number of distinct words in all of Shakespeare's writings is predicted to be 30,987 (the true answer is 28,239). The algorithm offers at present the best accuracy/memory trade-off known.

Yet other observables are conceivable. For instance, Giroire (2004, unpublished) has recently succeeded in analysing a class of *algorithms based on minima*, where the observable is now a collection of some of the smallest hashed values.

The algorithms discussed above involve a fascinating interplay between theory and practice. Analysis of algorithms intervenes at several crucial stages. First in order to correct the bias of raw observables suggested by probabilistic intuition and back of an envelope calculations; next to estimate variance and limit probability distributions, thereby quantifying the risk of "large deviations", which means abnormally inaccurate estimates. The LogLog algorithm for instance relies on maxima of geometric random variables, exponential generating functions, analytic depoissonization (Jacquet-Szpankowski [20, 26]), and once again Mellin transforms.

3 Sampling

Yet another algorithm originally designed for cardinality estimation in data bases and due to Wegman around 1980 turns out to regain importance in the modern context of approximate queries and sketches in very large data bases [3].

Once more, a *multiset* $\mathfrak{M}$ is given. Say we would like to extract a sample of s *distinct* value. In other words, we are sampling the domain of values, the *set*, that underlies $\mathfrak{M}$. Such a sample may be used to design adaptive data structures

or to gather useful statistics on the "profile" of data, like approximate quantiles: think of the problem of estimating the median salary in a population, given aggregated heterogeneous files presenting unpredictably repeated entries.

First, as usual, there is a simple-minded algorithm that proceeds by keeping at each stage, as data flows, the exact set (without repetitions) of distinct elements encountered so far and finally extracting the desired sample of size s. This algorithm suffers from a storage complexity that is at best proportional to the cardinality of $\mathfrak{M}$ and of a time complexity that is even nonlinear if sorting or index trees (B–trees) are used. The method may be improved a bit by fixing *a priori* a sampling factor p, say $p = \frac{1}{1024} = 2^{-10}$. This is STRAIGHT SAMPLING: Elements are hashed and only elements whose hashed value starts with a sequence of 10 zeros are filtered in and kept as a distinct set in a temporary file; at the end, resample the p-sample and obtain the needed collection of size s. The algorithm has become probabilistic. It will be reasonably behaved provided the cardinality of $\mathfrak{M}$ is well in excess of $1/p$ (precisely, we need $p\|\mathfrak{M}\| \gg s$), and the storage complexity will decrease by a factor of about 1/1000 (not bad!). However, if the nature of the data is *a priori* completely unknown and p has not been chosen in the right interval, the algorithm may fail dramatically by oversampling or undersampling. For in instance, in the case of only $n = 500$ different records and $p = 1/1000$, it is more than likely that *no* element at all is selected so that no sample of whatever size is obtained.

Wegman's remedy is known as ADAPTIVE SAMPLING. This elegant algorithm is described in the article that analyses it [12]. Assume again a sample of s distinct values is wanted. Fix a parameter b, called the bucket size, commensurate with s, e.g, $b = \frac{5}{2}s$, and prepare to manage a running list of distinct elements whose length is $\leq b$. The algorithm maintains a parameter called the sampling depth, δ whose value will get incremented as the algorithm proceeds; the sampling probability p is also a variable quantity bound to δ by the condition $p = 2^{-\delta}$. The algorithm runs then as follows:

> Start with $\delta = 0$ and accordingly $p = 1$. All the distinct elements seen are stored into the bucket until the bucket overflows ($b+1$ distinct elements have been found). At this stage, increase the sampling depth by 1, setting $\delta := \delta+1$ and $p := \frac{1}{2}p$. Then eliminate from the bucket all elements whose hashed value does not start with *one* initial zero: this has the effect of stochastically splitting the bucket and dividing its content by a factor close to 2. Resume and only consider from this point on, elements whose hashed value is of the form $0\cdots$, discarding the others. Repeat the following cycle: "increase-depth, decrease-sampling-ratio, and split-bucket" after each overflow occurs.

This algorithm is closely related to an important data structure known as the *digital tree* or "*trie*" (pronounce like"try") and its analysis benefits from techniques known for about three decades in analysis of algorithms [18, 22, 25, 26]. Let M be the (random) number of elements contained in the bucket once all elements have been scanned. It turns out that the bucket (conditioned upon the particular value of M) contains an unbiased sample. If $b/2$ suitably exceeds the desired sample size s, then, with high probability, one has $M > s$, so that

a subsample of size s can be extracted from the M elements that are available. This subsample is unbiased. Et voila!

The Adaptive Sampling algorithm also serves as a cardinality estimator, and this appears to have been the primary motivation for its design. Indeed, as is intuitively plausible, the quantity $M \cdot 2^{\delta} = M/p$ (with p, δ given their final values) turns out to be an unbiased estimator of the cardinality of $\mathfrak{M}$. Analysis based on generating functions and Mellin transforms shows that the accuracy is now

$$\alpha = \frac{1.20}{\sqrt{b}},$$

which is slightly less favorable than previous solutions. The algorithm however has the advantage of being totally unbiased, including in its nonasymptotic regime.

At this stage, we should mention yet another algorithm for cardinality estimation. It is due to K-Y. Whang *et al.* [28] and is sometimes called Hit Counting. Its ideas are somewhat related to sampling. Say again we want to estimate the number of distinct elements in a file and this number is known in advance not to exceed a certain bound ν. Hash elements of $\mathfrak{M}$ and keep the skeletton of a hash table of size (say) $m := \frac{\nu}{5}$ in the following manner: a bit is set to 1 for each cell that is hit at least once during the hashing process. (This technique is akin to the famous Bloom filters.) The observable that is chosen is the proportion E of empty (not hit) cells at the end of the procedure. Let n be the unknown cardinality of $\mathfrak{M}$; a Poisson approximation shows that the mean proportion of empty cells (0-bits in the skeletton) at the end is about $e^{-n/m}$. Thus, based on the approximate equality $E \approx e^{-n/m}$, propose $n^{\circ} = m \log E$ as an estimator of the unknown cardinality n. The algorithm works fine in theory, it is extremely accurate for small values of n, but its memory cost, which is of $O(n)$ bits, becomes exorbitant for massive data sets. At least, Hit Counting can be recycled within the LogLog algorithm [5]: this permits to correct nonlinearities present in the case of very low cardinalities when the full asymptotic regime of LogLog counting is not yet attained.

Returning to sampling, we note that the problem of sampling with multiplicities (i.e., one samples k positions out of n) looks simpler, but sophisticated algorithms can be developed; see Vitter's reference work [27].

4 Frequency moments

The amount of literature on the topic of estimating characteristics of very large data sets has exploded over the past few years. In addition to databases and networking, data mining has come into the picture. Hashing remains a prime randomization technique in this context and statistical observations made on hashed values can again provide very accurate information of various sorts.

An influential paper by Alon, Matias, and Szegedy [1] has proposed an important conceptual framework that includes many highly interesting quantitative characteristics of multisets, while encompassing the maintenance of approximate

counters and the estimation of cardinalities. Let $\mathfrak{M}$ be a multiset and let V be the underlying set, that is, the set of distinct values that elements of $\mathfrak{M}$ assume. Let f_v be the frequency (number of occurrences) of value v in $\mathfrak{M}$. The rth *frequency moment* is defined as

$$F_r(\mathfrak{M}) := \sum_{v \in V} f_v^r.$$

Thus maintaining a counter of the number of nondistinct elements of $\mathfrak{M}$ (with multiplicity counted) is equivalent to determining F_1, for which Approximate Counting is applicable. Similarly, the problem of estimating cardinalities is expressed in this framework as that of obtaining a good approximation to F_0, a problem addressed by algorithms Probabilistic Counting, LogLog, Adaptive Sampling, and Hit Counting discussed above.

The quantity F_2, a sort of variance, when suitably normalized, provides an indication of the amount by which the empirical distribution of elements of $\mathfrak{M}$ differs from the flat (uniform) distribution. Its estimation can be approached from

$$\phi_2 := \left(\sum_{x \in \mathfrak{M}} \epsilon(x) \right)^2, \tag{5}$$

where $\epsilon(x)$ is ± 1 and can be determined by translating the first bit (say) of the hashed value $h(x)$. This estimator is however subject to great stochastic fluctuations when several experiments are performed using various bits of hashed values, so that the estimated values are much less accurate than in the cases of F_0 and F_1.

The quantities F_r for $r > 2$ are intrinsically hard to estimate in the absence of any *a priori* assumption on the frequency profile of $\mathfrak{M}$. Of great interest is F_∞, which is taken to mean

$$F_\infty := \lim_{r \to \infty} F_r^{1/r},$$

as this is the frequency of the most frequent element in the multiset $\mathfrak{M}$. The determination of this quantity is crucial in many networking applications [8], as it is part of the general problem known as *Mice and Elephants*: in a large flux of data, how to recognize the ones that occupy most of the bandwidth?

The quantities F_r when r lies in the interval $(0, 2)$ can be estimated by resorting to stable laws of probability theory. This is a brilliant idea pioneered by Piotr Indyk [19] that relates to active research in the area of dimension reduction in statistics and computational geometry. For our purposes, we may take a *stable law of index* α to be the law of a random variable X whose characteristic function (Fourier transform) is

$$\mathbb{E}\left(e^{itX}\right) = e^{-|t|^\alpha}.$$

Such laws exist for $\alpha \in (0, 2)$. Given a multiset $\mathfrak{M}$ compute the quantity (compare with (5))

$$\phi_r := \sum_{x \in \mathfrak{M}} \epsilon(x), \tag{6}$$

where each $\epsilon(x)$ now obeys a stable law of parameter r and is determined via hashing from x itself. We have also, with previous notations,

$$\phi_r := \sum_{v \in V} f_v \epsilon(v).$$

Now a fundamental property of stable laws is that if the X_j are r–stable and independent, then

$$\sum_j \lambda_j X_j \overset{\text{(distribution)}}{=} \xi Z, \qquad \text{with} \quad \xi = \left(\sum_j |\lambda_j|^r \right)^{1/r},$$

and Z being itself r–stable. There results from the last two displayed equations the possibility of constructing an estimator for F_r when $r \in (0, 2)$. It suffices to devise an estimator of the multiplier of an r–stable law: in the literature, this is usually based on medians of experiments; it has not been investigated yet whether logarithmic techniques might be competitive.

The fact that F_0 can be approached as $\lim_{\eta \to 0} F_\eta$ attaches the problem of cardinality estimation (F_0) to this circle of ideas, but a precise assessment of practical complexity issues seems to be still lacking and the corresponding algorithm does not seem to outperform our earlier solutions.

Let us mention finally that the F_r can be used for parametric statistics purposes. If the data in the multiset $\mathfrak{M}$ are known to be drawn according to a family of distributions $(\mathcal{D}_\theta)$ (say a class of generalized Zipf laws), then the F_r which are computationally easily tractable (in linear time and small storage) may be used to infer a plausible value of θ best suited to a multiset $\mathfrak{M}$.

5 Conclusion

Simple probabilistic ideas combined with suitable analysis of the intervening probabilistic and analytic phenomena leads to a class of algorithms that perform counting by coin flippings. The gains attained by probabilistic design are in a number of cases quite spectacular. This is also an area where practice merges agreeably with theory (discrete models, combinatorics and probability, generating functions, complex analysis and integral transforms); see the manuscript of the forthcoming book by Flajolet & Sedgewick [16] for a systematic exposition. The resulting algorithms are not only useful but also much used in databases, network management, and data mining.

Acknowledgements. I am grateful to the organizers of *ASIAN'04* (the Ninth Asian Computing Science Conference held at Chiang Mai in December 2004) for their kind invitation. Many thanks in particular to Ms Kanchanasut, "Kanchana", for pushing me to write this text, for her patience regarding the manuscript, her numerous initiatives, and her constant support of French–Thai cooperation in this corner of science.

References

1. Alon, N., Matias, Y., and Szegedy, M. The space complexity of approximating the frequency moments. *Journal of Computer and System Sciences 58*, 1 (1999), 137–147.
2. Andrews, G. E., Crippa, D., and Simon, K. q-series arising from the study of random graphs. *SIAM Journal on Discrete Mathematics 10*, 1 (1997), 41–56.
3. Babcock, B., Babu, S., Datar, M., Motwani, R., and Widom, J. Models and issues in data stream systems. In *Proceedings of Symposium on Principles of Database Systems (PODS)* (2002), pp. 1–16.
4. Bertoin, J., Biane, P., and Yor, M. Poissonian exponential functionals, q-series, q-integrals, and the moment problem for log-normal distributions. Tech. Rep. PMA-705, Laboratoire de Probabilitś et Modèles Aléatoires, Université Paris VI, 2002.
5. Durand, M. *Combinatoire analytique et algorithmique des ensembles de données*. PhD thesis, École Polytechnique, France, 2004.
6. Durand, M., and Flajolet, P. LogLog counting of large cardinalities. In *Annual European Symposium on Algorithms (ESA03)* (2003), G. Di Battista and U. Zwick, Eds., vol. 2832 of *Lecture Notes in Computer Science*, pp. 605–617.
7. 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.).
8. Estan, C., and Varghese, G. New directions in traffic measurement and accounting: Focusing on the elephants, ignoring the mice. *ACM Transactions on Computer Systems 21*, 3 (2003), 270–313.
9. Estan, C., Varghese, G., and Fisk, M. Bitmap algorithms for counting active flows on high speed links. Technical Report CS2003-0738, UCSD, Mar. 2003. Available electronically. Summary in *ACM SIGCOMM Computer Communication Review* Volume 32 , Issue 3 (July 2002), p. 10.
10. Finch, S. *Mathematical Constants*. Cambridge University Press, New-York, 2003.
11. Flajolet, P. Approximate counting: A detailed analysis. *BIT 25* (1985), 113–134.
12. Flajolet, P. On adaptive sampling. *Computing 34* (1990), 391–400.
13. Flajolet, P., Gourdon, X., and Dumas, P. Mellin transforms and asymptotics: Harmonic sums. *Theoretical Computer Science 144*, 1–2 (June 1995), 3–58.
14. Flajolet, P., and Martin, G. N. Probabilistic counting. In *Proceedings of the 24th Annual Symposium on Foundations of Computer Science* (1983), IEEE Computer Society Press, pp. 76–82.
15. Flajolet, P., and Martin, G. N. Probabilistic counting algorithms for data base applications. *Journal of Computer and System Sciences 31*, 2 (Oct. 1985), 182–209.
16. Flajolet, P., and Sedgewick, R. *Analytic Combinatorics*. 2004. Book in preparation; Individual chapters are available electronically.
17. Guillemin, F., Robert, P., and Zwart, B. AIMD algorithms and exponential functionals. *Annals of Applied Probability 14*, 1 (2004), 90–117.
18. Hofri, M. *Analysis of Algorithms: Computational Methods and Mathematical Tools*. Oxford University Press, 1995.
19. Indyk, P. Stable distributions, pseudorandom generators, embeddings and data stream computation. In *Proceedings of the 41st Annual IEEE Symposium on Foundations of Computer Science (FOCS)* (2000), pp. 189–197.

20. Jacquet, P., and Szpankowski, W. Analytical de-Poissonization and its applications. *Theoretical Computer Science 201*, 1-2 (1998), 1–62.
21. Knuth, D. E. *The Art of Computer Programming*, 3rd ed., vol. 2: Seminumerical Algorithms. Addison-Wesley, 1998.
22. Knuth, D. E. *The Art of Computer Programming*, 2nd ed., vol. 3: Sorting and Searching. Addison-Wesley, 1998.
23. Morris, R. Counting large numbers of events in small registers. *Communications of the ACM 21*, 10 (1977), 840–842.
24. Prodinger, H. Approximate counting via Euler transform. *Mathematica Slovaka 44* (1994), 569–574.
25. Sedgewick, R., and Flajolet, P. *An Introduction to the Analysis of Algorithms*. Addison-Wesley Publishing Company, 1996.
26. Szpankowski, W. *Average-Case Analysis of Algorithms on Sequences*. John Wiley, New York, 2001.
27. Vitter, J. Random sampling with a reservoir. *ACM Transactions on Mathematical Software 11*, 1 (1985).
28. Whang, K.-Y., Vander-Zanden, B., and Taylor, H. A linear-time probabilistic counting algorithm for database applications. *ACM Transactions on Database Systems 15*, 2 (1990), 208–229.