# Succinct Approximate Counting of Skewed Data*

**David Talbot**
Google Inc., Mountain View, CA, USA
talbot@google.com

## Abstract

Practical data analysis relies on the ability to count observations of objects *succinctly* and *efficiently*. Unfortunately the space usage of an exact estimator grows with the size of the *a priori* set from which objects are drawn while the time required to maintain such an estimator grows with the size of the data set. We present static and on-line approximation schemes that avoid these limitations when approximate frequency estimates are acceptable. Our *Log-Frequency Sketch* extends the *approximate counting* algorithm of Morris [1978] to estimate frequencies with bounded relative error via a single pass over a data set. It uses *constant space per object* when the frequencies follow a power law and can be maintained in *constant time per observation*. We give an $(\epsilon, \delta)$-approximation scheme which we verify empirically on a large natural language data set where, for instance, 95 percent of frequencies are estimated with relative error less than $0.25$ using fewer than 11 bits per object in the static case and 15 bits per object on-line.

## 1 Introduction

Frequency estimation lies at the heart of many applications of statistics. As data sets grow in size, estimating the frequency of distinct objects becomes challenging in many settings. Recently there has been significant interest in constant space data structures for the analysis of *unbounded data streams* [Cormode and Muthukrishnan, 2005]. Unfortunately, such constructions are of limited practical use for frequency estimation from skewed data as their expected error grows with the size of the stream. For large but finite data sets, however, we can construct frequency estimators that use constant space *per object* and constant time *per observation* with bounded relative error.

Fundamentally the problem we face is that any lossless frequency estimation scheme will use more space per object and take more time per observation as our data sets grow in size: not only do we need to maintain more counters as we observe more distinct objects, we must also use more space to represent each counter if we are to distinguish between them without error. Once the counters no longer fit in main memory, we must either resort to slow secondary storage or to sorting the data set; either approach implies that the time taken to process each observation will grow with the amount of data. When data follows a *power law*, the situation is exacerbated by the need to maintain counters for a large number of low-frequency objects. Here we focus on this important class of distribution observed in many practical settings.

Our main contribution is a simple frequency approximation scheme, the *Log-Frequency Sketch*, that can circumvent these constraints when approximate answers are acceptable. We expect it to be of practical interest in applications that involve large-scale data analysis. The Log-Frequency Sketch estimates the frequencies of objects $x \in U$ via a single pass over a data set $D$ with the following guarantees

(i) (*Bounded relative error*) Estimates are within a constant factor of their true frequencies with high probability.

(ii) (*Constant space*) Estimates are maintained using constant space per object independent of $|U|$.

(iii) (*Constant update complexity*) Estimates are maintained in constant time per observation independent of $|D|$.

In Section 2 we state the problem formally and consider lower-bounds; in Section 3 we review related work; in Section 4 we present a static approximation scheme which we extend to the on-line setting in Section 5. Section 6 demonstrates empirically the space savings that can be achieved with these data structures and in Section 7 we conclude.

## 2 Problem Statement

We wish to count the frequency of distinct objects observed in a data set $D$ and drawn from an *a priori* universe $U$. We refer to the set of objects $x \in U$ that have non-zero frequency $F(x)$ in $D$ as the *support* $S$. We seek 'scalable' solutions whereby the space used per object $x \in S$ is independent of $|U|$ and the time required to process each observation is independent of $|D|$. We assume that the frequencies $F(x), x \in S$ follow a power law whereby $\mathbf{Pr}[F(x) \geq f] \propto f^{-\alpha}$ for some parameter $\alpha$ (the skew) and that the support is sparse (i.e., $|U| \gg |S|$). These assumptions are commonly met in practice, particularly when $U$ is combinatorial in nature.

*Work completed at the University of Edinburgh, U.K.

We denote the frequency of an object $x \in U$ after $t$ observations $x \in D$ have been processed by $F_t(x)$. A frequency estimator supports the following two operations:

- (*Estimate*) Given an observation $x \in U$ return $F_t(x)$;
- (*Update*) Given an observation $x \in U$ update the estimator such that $F_t(x) = F_{t-1}(x) + 1$.

### 2.1 Lower bounds on exact estimators

When $U$ and $S$ are small, exact frequency estimation is trivial: maintain $|S|$ counters and update the appropriate one after each observation. Unfortunately a simple counting argument [Carter *et al.*, 1978] implies that any data structure supporting exact set membership queries for a set $S \subset U$ must use at least $\log \binom{|U|}{|S|}$ bits per object on average and hence must grow with the size of the *a priori* universe $U$.

For example when counting observations of $n$-grams in a natural language corpus, not only must we use more counters as more distinct $n$-grams are observed, we must also use more space to identify each counter as the vocabulary grows in size: the more distinct $n$-grams that may be observed *a priori*, the more space we must use to identify each observed type.

As soon as lossless counters can no longer be maintained in main memory, we must either resort to slow secondary storage or estimate the frequencies by sorting and then scanning the observations in $D$. This implies that we must perform $\Theta(\ln |D|)$ comparisons on average per observation in $D$ and hence will spend more time per observation as $D$ grows in size. (Note that this is also true in a distributed setting.)

### 2.2 Approximation schemes

Statistical analysis often introduces approximations such as modeling assumptions. It is therefore natural to consider how approximate data structures may allow us to leverage more data, more efficiently. An approximate data structure is one that is allowed (occasionally) to fail to distinguish between two or more $x \in U$. Allowing such errors fundamentally alters the space complexity of the data structure problem.

An approximate key/value map in which a value $v \in V$ is associated with each key $x \in S$ requires at least $\log 1/\epsilon + \mathbf{H}(\vec{p})$ bits per key on average where $\mathbf{H}(\vec{p})$ is the entropy of the distribution over values in the map and $\epsilon$ is an upper bound on the probability of a false positive, i.e. that a value $v \in V$ is returned for a key $x \in U \setminus S$ [Talbot and Talbot, 2008]. Hence we may store object/frequency pairs in space that depends only on the skew of the frequencies $\alpha$ and the error rate $\epsilon$ independent of the a priori universe $U$, if errors are allowed.

A frequency approximation scheme $\hat{F}(x), x \in U$ provides an estimate of the true frequency $F(x)$. To be useful, an approximation scheme must provide guarantees on its accuracy. To quantify errors here we use the relative error for $x \in S$, i.e. $\ell_{rel} = |\hat{F}(x) - F(x)|/|F(x)|$. This captures the intuition that small absolute errors are less significant when they occur for high-frequency objects. For $x \in U \setminus S$, we use the absolute loss since $F(x) = 0$.for such objects.

We consider approximation schemes with bounded *expected relative error* whereby for some $\epsilon > 0$

$$\sum_{x \in U} \mathbf{Pr}[x] \sum_{f'} \mathbf{Pr}[\hat{F}(x) = f'|x] \frac{|\hat{F}(x) - F(x)|}{|F(x)|} \leq \epsilon$$

and $(\epsilon, \delta)$-approximation schemes that guarantee that with probability at least $(1 - \delta)$ the loss will not exceed $\epsilon$ on *any single* use of the data structure, i.e. for the relative loss

$$\mathbf{Pr}[|\hat{F}(x) - F(x)| \leq \epsilon F(x)] \geq (1 - \delta).$$

Here the frequency estimator $\hat{F}(x), x \in U$ will be a random variable due to the random selection of hash functions for any given instantiation of the data structure. We restrict ourselves to data structures offering guarantees that hold without *a priori* assumptions regarding the distribution of queries $\mathbf{Pr}[x]$, $x \in U$ but note that such assumptions might be used to further optimize space-usage in specific applications.

## 3 Related Work

A number of succinct approximate frequency estimators have been proposed. The Spectral Bloom filter [Cohen and Matias, 2003] has bounded 0-1-loss, but comes with no guarantees on the size of errors. It is also unclear what constant terms are hidden in the analysis of its space usage [Cohen and Matias, 2003]. The Space-Code Bloom filter [Kumar *et al.*, 2004] uses a version of the *coupon collector problem*, e.g. [Motwani and Raghavan, 1995], to estimate frequencies with bounded relative error. Unfortunately, the high variance of the coupon collector estimator when either a small or large number of 'coupons' have been sampled implies that this construction will waste space over large ranges of frequencies. Moreover, its configuration requires significant *a priori* assumptions regarding the input distribution and no explicit analysis of the space/error trade-off is given in [Kumar *et al.*, 2004].

The Log-Frequency Bloom filter [Talbot and Osborne, 2007] uses the low-entropy of a power law to store such frequencies succinctly off-line with bounded relative error. Our work extends this construction to the on-line setting by adapting the approximate counting algorithm of Morris [1978].

Over the past decade there has been significant work on the analysis of unbounded data streams [Alon *et al.*, 1996]. Frequency estimators that use constant space independent of both $|S|$ and $|U|$ have been proposed, e.g. [Cormode and Muthukrishnan, 2005], with bounded errors that depend on $|D|$. Unfortunately, these estimators are of little practical interest for estimating the frequency of individual objects other than those that make up a significant fraction of $D$ (so-called *heavy-hitters*) since when $|D|$ is large the error bound $\epsilon|D|$ will be far greater than the average frequency for $x \in U$.

## 4 Static Approximation Schemes

We now describe two static approximation schemes for representing frequencies succinctly when these follow a power law. These schemes are based on the Log-Frequency Bloom filter [Talbot and Osborne, 2007] which itself is a simple extension of the Bloom filter data structure [Bloom, 1970].

### 4.1 An approximate set membership tester

A Bloom filter consists of a bit array $B$ of size $m$ initialized to 0 and $k$ hash functions $h_i, i \in [k], h_i : U \rightarrow [m]$. Given a set of $n$ objects $S \subset U$, a Bloom filter can be used to implement an approximate set membership test by setting each of the bits $B[h_i(x)], i \in [k]$ to 1 for $x \in S$. Once $S$ has been stored in this way, we can check whether a query object $x \in U$ is an element of $S$ by hashing $x$ under the $k$ hash functions and examining the bits in $B$. No false negatives will occur for $x \in S$ and if some of the bits in $B$ remain 0 after all of $S$ has been stored, there is a non-zero probability that one of the $k$ hash functions will index a 0 in $B$ when evaluated for an object $x \in U \setminus S$. When this occurs we know that $x \in U \setminus S$. A false positive occurs when we fail to find a zero among the $k$ bits examined for an object $x \in U \setminus S$. Given $m$, $n$ and $k$, the probability that a random object $x \in U \setminus S$ results in a false positive can be computed since the number of bits that remain 0 in $B$ will be very close to its expected value w.h.p. For fixed $n$ and $m$ the expected false positive rate is minimized by choosing $k$ such that the expected proportion of bits that remain 0 is 0.5 [Bloom, 1970].

### 4.2 An approximate frequency table

The Log-Frequency Bloom filter [Talbot and Osborne, 2007] stores a static set of object/frequency pairs $\{(x, F(x))|x \in S\}$ using a bit array $B$ and an indexed set of hash functions, $H = \{h_i : U \rightarrow [m]|i \in [\infty]\}$. A frequency $F(x)$ is associated with an object $x \in S$ by storing a unary code that approximates $\log(F(x))$. Logarithmic quantization converts a power law into an approximately geometric distribution, hence unary codes resemble Huffman codes for the quantized frequencies. **Algorithm A** uses this data structure to implement an $(\epsilon, \delta)$-approximation scheme which guarantees that the probability of an error exceeding $\epsilon$ on any single use is less than $\delta$ while **Algorithm B** constructs an estimator with bounded relative error. **Algorithms C** and **D** extend these estimators to function in an on-line setting.

All our data structures use space $m = |B|$ that is linear in the number of objects stored $n = |S|$ and the average natural logarithm of their associated frequencies $\gamma$; space usage is therefore stated in terms of $n\gamma$.

**Static $(\epsilon, \delta)$-approximation scheme (Algorithm A)**
Given $n$, $\gamma$, $\epsilon > 0$ and $\delta > 0$, choose $m$ such that

$$m/n\gamma = \log(e)1/\epsilon \log 1/\delta \text{ bits},$$

and let $b = 1 + \epsilon$ and $k = \log 1/\delta$.

To store $F(x)$ for $x \in S$ evaluate the first $k\lceil \log_b F(x) \rceil$ hash functions in $H$ and set the corresponding bits in $B$ to 1.

To retrieve an estimate $\hat{F}(x)$ for $x \in S$, evaluate hash functions $h \in H$ in index order until a 0 is found in $B$. If $r$ is the number of hash functions evaluated prior to finding the first 0 then return $\hat{F}(x) = b^{\lfloor \frac{r}{k} \rfloor}$.

**Theorem 1. Algorithm A** *implements an $(\epsilon, \delta)$-approximation scheme for $F(x), x \in S$ under the relative loss using $\log(e)1/\epsilon \log 1/\delta$ bits on average.*

*Proof.* The number of hashes performed when quantizing base $b$ will be proportional to $1/\ln(b) \log 1/\delta$ which, using $\ln(1 + x) \approx x$ for small $x$, is approximately $1/\epsilon \log 1/\delta$. By an argument that is identical to that for the original Bloom filter, setting $m$ to be a factor $\log(e)$ greater than the total number of hashes performed when storing $S$ ensures that w.h.p. at least half the bits in $B$ remain 0 once the whole of $S$ has been stored (likewise this configuration minimizes the space usage). By storing the ceiling of the logarithm base $b = 1 + \epsilon$ we incur a relative quantization error of at most $\epsilon$ for any $F(x), x \in S$. Since each increment on the logarithmic scale is encoded using $k = \log 1/\delta$ hash functions, we must find an additional $k$ bits set in $B$ at random in order to incur any additional error when retrieving a frequency. As half the bits in $B$ remain 0, the probability of this event and hence of a relative error greater than $\epsilon$ is at most $2^{-\log(1/\delta)} = \delta$. □

**Static bounded relative error estimator (Algorithm B)**
Given $n$, $\gamma$ and $\epsilon > 0$ choose $m$ such that

$$\frac{m}{n\gamma} = \left( \ln \frac{1 + \epsilon}{1 + \epsilon(1 - 1/e)} \right)^{-1}$$

and let $b = e^{n\gamma/m}$.

To store $F(x), x \in S$, evaluate the first $\lceil \log_b F(x) \rceil$ hash functions in $H$ and set the corresponding bits in $B$ to 1.

To retrieve an estimate $\hat{F}(x)$ for $x \in S$, evaluate hash functions $h \in H$ in index order until a 0 is found in $B$. Let $r$ be the number of hash functions evaluated prior to finding the first 0 then return $\hat{F}(x) = b^r$.

**Theorem 2. Algorithm B** *has expected relative error of at most $\epsilon$ independent of $F(x)$. For small $\epsilon$ it uses approximately $e/\epsilon$ bits on average.*

*Proof.* Let $\rho$ be the proportion of bits that remain 0 in $B$ once $S$ has been stored. By our choice of quantization base $b$, we perform $m$ hashes to store $S$ hence w.h.p. $\rho \approx (1-1/m)^m \approx 1/e$. The expected relative error for any $F(x), x \in S$ is then

$$E[\ell_{rel}|F(x) = f] \leq \sum_{i=0}^{\infty} \rho(1 - \rho)^i \frac{|fb^{i+1} - f|}{|f|} \quad (1)$$

$$= \frac{b - 1}{1 - b(1 - \rho)} = \epsilon \quad (2)$$

where in Eq. (1) we use the fact that the quantization error cannot be greater than a factor of $b$ and $i$ counts the number of ones found in the bit array at random after the first $\lceil \log_b F(x) \rceil$ hash functions have been evaluated: each additional increment occurs independently with probability $(1 - \rho)$ and multiplies the estimate by a factor of $b$. Eq. (2) uses the fact that the error converges for our choice of $b$ and $\rho$ and follows by substituting these values and rearranging. Using $\ln(1 + x) \approx x$ for small $x$, the space usage $m/n\gamma$ is seen to be approximately $e/\epsilon$. □

**Algorithms A** and **B** show that logarithmic quantization and unary coding result in a natural spacing between codewords in a Bloom filter such that the expected relative error of $\hat{F}(x)$ is constant and the probability that an estimate deviates from $F(x)$ decays geometrically. Unlike a standard Bloom filter, **Algorithm B** results in a fraction $1/e$ of the bits in

$B$ remaining 0. By differentiating the expected relative error we can see that for small $\epsilon$ this is the optimal configuration. Using more than 1 hash function per increment (as in **Algorithm A**) is suboptimal when bounding the expected relative error, but necessary when we want stronger guarantees on the probability that the estimator deviates on any single use.

Neither **Algorithm A** or **B** give any guarantees on frequency estimates $\hat{F}(x)$ for $x \in U \setminus S$ for which $F(x) = 0$. Assuming that $\epsilon \leq 1$ we can add guarantees for these under the absolute loss by additionally storing each $x \in S$ in a separate Bloom filter using $k = \log 1/\delta$ hash functions for **Algorithm A** and $k = \epsilon + \log 1/\epsilon$ for **Algorithm B**. Prior to estimating the frequency for a key $x \in U$, we first query this separate Bloom filter and return 0 if it is not present.

## 5 On-line Approximation Schemes

An on-line frequency estimator constructed via a single pass over a data set avoids the need to precompute $F(x), x \in S$. On-line estimation poses two additional problems:

(i) Errors may be compounded over multiple updates;

(ii) A succinct counter may be slow to update.

If $S$ and $U$ are too large to allow exact counters to be held in memory, we must rely on approximate estimators on-line. Each time we observe an object $x \in U$, we will therefore retrieve an estimate $\hat{F}_t(x)$ rather than the true frequency $F_t(x)$. An approximate estimator may also not have sufficient resolution to reflect an increment of 1 exactly (cf. the logarithmic quantization scheme used in Section 4).

A succinct approximate encoding of a set of object/frequency pairs should assign short codes to the most common frequencies leaving longer codes for the rarer frequencies [Talbot and Talbot, 2008]. For a power law distribution, most *objects* will have low frequencies and hence these should be assigned short codes. A large proportion of *observations*, on the other hand, will be drawn from a small set of high-frequency objects which must then be assigned longer codes. If the time needed to update our data structure depends on the size of the code representing the current observation's frequency, on-line counting may be inefficient.

### 5.1 Approximate Counting

Morris [1978] presented an elegant algorithm for estimating frequencies approximately on-line. His algorithm allows a binary counter of size approximately $\log \log N$ to be used to count up to a maximum frequency $N$. Morris' *approximate counting* algorithm results in an unbiased estimator that has approximately constant relative error independent of $F(x)$.

Approximate counting is an adaptive Monte Carlo sampling algorithm. For each observation $x \in D$, we generate a uniform random variate $u \in (0, 1)$ and update a counter $R(x)$ iff $u \leq \Delta_r$ where $r$ is the current value of the counter,

$$\Delta_r = \frac{1}{G(r+1) - G(r)}$$

and $G : \mathbb{Z}_+ \rightarrow \mathbb{R}$ is any non-decreasing function. If $R(x), x \in S$ are initialized to 0, then $G_t(x)$ will be an unbiased estimator of $F_t(x)$ since the update rule implies that

$$\mathbb{E}[G_{t+1}(x)|R(x)] \quad = \quad E[G_t(x)|R(x)] + 1. \qquad (3)$$

If $G$ is the exponential function $b^{R(x)}$, then the variance of $G(x)$ is approximately quadratic in $F(x)$. Using Chebyshev's inequality, the probability $\delta$ that this estimator deviates from its expected value by more than a factor of $\epsilon$ can be bounded and an $(\epsilon, \delta)$-approximation for $F(x)$ constructed by choosing $b = 1 + 2\epsilon^2\delta$ [Morris, 1978].

Morris' algorithm reduces the space requirements of a counter exponentially but suffers from the following drawbacks when $F$ follows a power law and $S$ and $U$ are large:

(i) Fixed-width binary counters use $O(\log \log N)$ space where $N$ depends on $|D|$ but most counts will be small;

(ii) The space required to identify counters $R(x)$ exactly for each object $x \in U$ will depend on $|U|$.

We now present algorithms that circumvent these drawbacks by approximating the approximate counters $R(x)$ using the Log-Frequency Bloom filter scheme presented in Section 4.

**Biased *approximate*, approximate counting (Algorithm C)** Given a bit array $B$ and set of indexed hash functions $H$ as in Section 4, process each observation $x \in D$ as follows:

(i) Set $r = 0$ and evaluate hash functions $h_{r+1} \in H$ in index order incrementing $r$ until we find a 0 in $B$;

(ii) Generate a random variate $u \in (0, 1)$ and iff $u \leq 1/(b^{r+1} - b^r)$ set $B[h_{r+1}(x)] = 1$.

Retrieve estimates $\hat{F}(x), x \in U$ as in **Algorithm B**.

**Algorithm C** is a natural application of Morris' algorithm to obtain an on-line version of **Algorithm B**; unfortunately, it is biased and inefficient. The expected bias incurred on each update can be computed from the change in the conditional expectation of the estimator as a single observation is processed. Here we must average over three random variables: the uniform variate $u$ generated in step (ii), the number of bits $i$ found erroneously to be 1 when retrieving $r$ in step (i) and (if we perform an update in step (ii)) the number of bits $j$ found erroneously to be 1 when estimating $R_{t+1}(x)$ afterwards. Here $\rho$ is the proportion of zeros in $B$ and assuming that $h \in H$ are independent, the probability of retrieving $i$ bits erroneously when estimating $\hat{R}_t(x)$ does not depend on when these errors occur but only on their number

$$\begin{aligned}
&\mathbb{E}[G_{t+1}(x)|R_t(x) = r] = \\
&\sum_{i=0}^{\infty} \mathbf{Pr}[\hat{R}_t(x) = r + i] \Bigg( (1 - \mathbf{Pr}[u \leq \Delta_{r+i}]) b^{(r+i)} + \\
&\mathbf{Pr}[u \leq \Delta_{r+i}] \sum_{j=0}^{\infty} \mathbf{Pr}[\hat{R}_t(x) = r + i + 1 + j] b^{(r+i+j+1)} \Bigg) \\
&\approx \mathbb{E}[G_t(x)|R_t(x) = r] + \sum_{i=0}^{\infty} \rho(1-\rho)^i \left( \frac{b \sum_{j=0}^{\infty} \rho(1-\rho)^j b^j - 1}{b - 1} \right) \\
&= \quad \mathbb{E}[G_t(x)|R_t(x) = r] + 1 + \frac{b(1-\rho)}{1 - b(1-\rho)}. \qquad (4)
\end{aligned}$$

The third term in Eq. (4) is the bias accumulated over a single update. This term is 0 in Morris' original algorithm, cf. Eq. (3), and will affect high-frequency objects disproportionately in **Algorithm C** resulting in non-uniform errors.

The need to retrieve $\hat{R}_t(x)$ on each observation of $x$ also makes **Algorithm C** inefficient. When $F(x)$ follows a power law, a small number of high-frequency objects for which $F(x) = O(|D|)$ will use $O(|D| \log_b |D|)$ probes. We now propose an algorithm that avoids both these flaws.

**The Log-Frequency Sketch (Algorithm D)**
Let $\beta = \sum_{j=0}^{\infty} \rho(1-\rho)^j b^j$. Given a bit array $B$ and set of indexed hash functions $H$ as in Section 4, process each observation $x \in D$ as follows:

(i) Generate a random variate $u \in (0, 1)$;

(ii) Set $r = 0$ and evaluate hash functions $h_{r+1} \in H$ incrementing $r$ while $u \leq 1/(\beta b^{r+1} - b^r)$ and $B[h_{r+1}(x)] = 1$;

(iii) If (ii) terminates due to finding a 0 in $B$ then set $B[h_{r+1}(x)] = 1$; otherwise do nothing.

Retrieve estimates $\hat{F}(x), x \in U$ as in **Algorithm B**.

**Theorem 3. Algorithm D** *results in an estimator* $\hat{F}(x)$ *with constant bounded relative error independent of* $F(x)$.

*Proof.* (Sketch) The revised update probability used in step (ii) offsets the expected bias incurred when processing an observation. This can be seen by re-computing the conditional expectation Eq. (4) and noting that the expected increment is now 1 as in Eq. (3). The variance of the revised estimator is approximately $O(F(x)^2)$ as in Morris' original algorithm hence we may use Chebyshev's inequality to bound the probability that it deviates from $F(x)$ by more than a factor of $\epsilon$ for any $F(x)$ uniformly. □

The following theorem shows that **Algorithm D** has constant expected update complexity per observation.

**Theorem 4. Algorithm D** *performs at most* $1 + 1/(b-1)^2$ *probes per observation independent of* $|D|$ *and* $F(x)$.

*Proof.* An upper bound on the number of probes $r_{max}$ performed on average in step (ii) can be computed by assuming that $B$ contains no zeroes and ignoring $\beta$. Since probes in step (ii) are now conditioned on $\Delta_r < u$ we have

$$\mathbb{E}[r_{max}] \leq 1 + \sum_{i=0}^{\infty} \frac{i}{b^i} = 1 + \left(\frac{1}{b-1}\right)^2$$

which depends only on the error rate. □

# 6 Experiments

We evaluated **Algorithms A** to **D** by using them to estimate the frequencies of $n$-gram sequences observed in natural language text. Applications such as speech recognition and machine translation rely on such frequencies to estimate language model scores used to prune a typically exponential hypothesis space. Estimating and storing such frequencies is known to be challenging, e.g. [Brants *et al.*, 2007].

The $n$-gram data consisted of all 1 to 5-grams observed in 10 million sentences drawn from the Gigaword Corpus [Graff *et al.*, 2005]. **Algorithms A** and **B** took the pre-computed $n$-gram frequencies as input while **Algorithms C** and **D** used the corpus directly. We queried each data structure with $n$-grams drawn from a held-out set of 1 million sentences and computed the empirical relative error for $n$-grams with non-zero frequency; we also measured the proportion of estimates that incurred a relative error greater than 0.25 and 0.5. The probability of returning a non-zero estimate for $n$-grams not observed in the training data was set to 0.015 in all cases using an additional 6 hash functions to encode that $x \in S$.

All results show averages over 25 instantiations of each data structure with different random seeds to the hash functions. The space usage given on all plots is bits per distinct $n$-gram type in the training data. For **Algorithm D** we also recorded the number of probes performed per observation when constructing the data structure.

For **Algorithms C** and **D** an interesting open question is how to determine the size of the bit array $B$ *a priori* for a given data set. We could use a sketch-based data structure [Cormode and Muthukrishnan, 2005] to estimate the number of distinct objects and the skew of the data via an initial pass and set the expected memory requirements accordingly. Here we side-step this issue by allocating the same amount of space to **Algorithm D** as is used by **Algorithm B** off-line.

## 6.1 Optimizing for integer counts

The analysis of Sections 4 and 5 holds for any positive frequencies; here, however, we are interested only in counting observations and can restrict ourselves to integer counts. This allows us to optimize the space-usage of the data structures.

Firstly we can replace the ceiling operator used for static quantization by a floor operator; this complicates the analysis but is preferable since our static data structures can only overestimate the quantized count that is stored. Secondly when $b < 2$ the logarithmic quantization scheme will have more codes than are necessary since for small $i$ we will have $|b^{i+1} - b^i| < 1$. We can save space by removing these codes from the codebook. This results in deterministic updates for the online algorithms in this range since $\Delta_r = 1$. Results obtained using this optimization are labelled *optimized*.

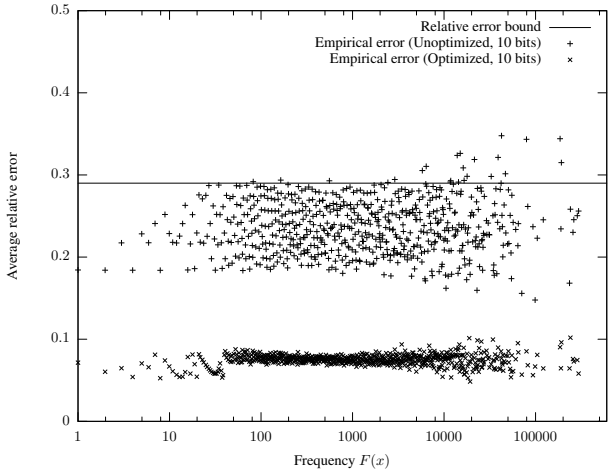


Figure 1: Relative Error for **Algorithm B**.

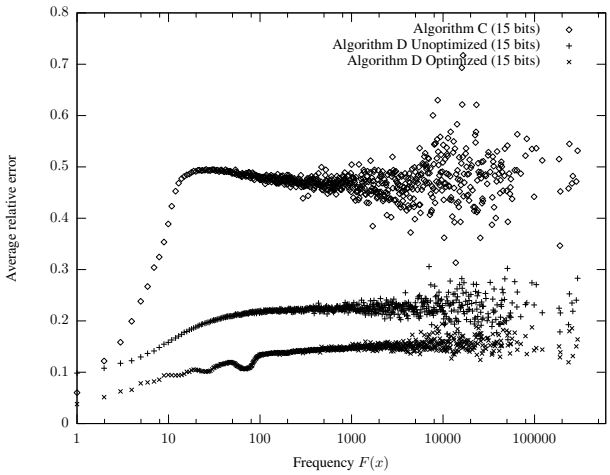


Figure 2: Relative Error for **Algorithms C** and **D**.

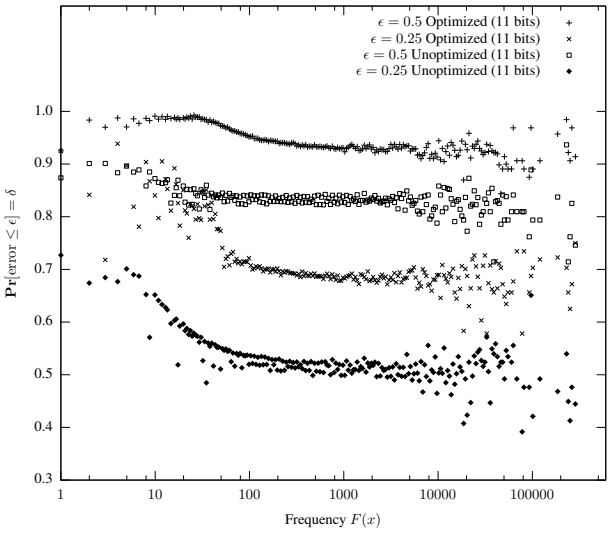


Figure 3: Distribution of Errors **Algorithm D** .

### 6.2 Results

Figure 1 shows that **Algorithm B** has bounded relative error that is approximately constant; the larger fluctuations at higher frequencies are due to data sparsity in this range (very few objects have such large frequencies). Unoptimized, **Algorithm B** performs close to the bound of Theorem 2 (Section 4) while optimized its error rate is significantly lower. While not shown here, **Algorithm A** also performed close to the bounds given by Theorem 1 and significantly better when using the integer optimization.

Figure 2 shows that the error for **Algorithm C** diverges as predicted by our analysis in Section 5. The bias-corrected **Algorithm D**, however, maintains an approximately constant relative error and our integer optimization further improves its performance. Estimates are particularly good for low frequencies due to lower variance in the probabilistic updates in this range.

Figure 3 shows the proportion of frequency estimates with relative error less than $0.5$ and $0.25$ respectively for **Algorithm D** both optimized and unoptimized. These plots suggest that the errors made by this estimators are relatively concentrated around their expected value; this may be attributed to the geometrically decaying probability of error in the Bloom filter encoding (see Section 4).

Extending this empirical analysis, we found that **Algorithms B** and **D** could estimate $95$ percent of $n$-gram frequencies with relative error less than $0.25$ using $11$ bits and $15$ bits respectively. This compares very favourably with $gzip$ which used $41$ bits per $n$-gram to store the static frequency data and does not support random access. We note also that the update complexity per observation for **Algorithm D** tended towards the bound given in Theorem 4 but was strictly lower in practice.

## 7 Conclusions

We have presented a range of succinct approximate counting algorithms for skewed data with robust guarantees on their expected error. We believe that algorithms such as these may help with the exploitation of larger data sets in a range of applications. In future work we intend to adapt these data structures for a distributed setting.

## References

[Alon *et al.*, 1996] N. Alon, Y. Matias, and M. Szegedy. The space complexity of approximating the frequency moments. In *Proceedings of STOC '96*, pages 20–29, 1996.

[Bloom, 1970] B. Bloom. Space/time tradeoffs in hash coding with allowable errors. *Comm. of ACM*, 13:422–426, 1970.

[Brants *et al.*, 2007] T. Brants, A.C. Popat, P. Xu, F.J. Och, and J. Dean. Large language models in machine translation. In *Proceedings of EMNLP-CoNLL 2007*, Prague, 2007.

[Carter *et al.*, 1978] L. Carter, R.W. Floyd, J. Gill, G. Markowsky, and M.N. Wegman. Exact and approximate membership testers. In *Proceedings of STOC '78*, pages 59–65, 1978.

[Cohen and Matias, 2003] S. Cohen and Y. Matias. Spectral Bloom filters. In *Proceedings of the 2003 ACM SIGMOD, International Conference on Management of Data*, pages 241–252, 2003.

[Cormode and Muthukrishnan, 2005] G. Cormode and S. Muthukrishnan. An improved data stream summary: The Count-Min Sketch. *Journal of Algorithms*, 55:58–75, 2005.

[Graff *et al.*, 2005] D. Graff, J. Kong, K. Chen, and K. Maeda. English Gigaword Second Edition, 2005. LDC2005T12.

[Kumar *et al.*, 2004] A. Kumar, J. Xu, J. Wang, O. Spatscheck, and L. Li. Space-code Bloom filter for efficient per-flow traffic measurement. In *Proceedings of INFOCOM '04*, 2004.

[Morris, 1978] Robert Morris. Counting large numbers of events in small registers. *Comm. of the ACM*, 21:840–842, 1978.

[Motwani and Raghavan, 1995] R. Motwani and P. Raghavan. *Randomized Algorithms*. Cambridge University Press, 1995.

[Talbot and Osborne, 2007] D. Talbot and M. Osborne. Randomised language modelling for statistical machine translation. In *Proc. of 45th Annual Meeting of the ACL*, 2007.

[Talbot and Talbot, 2008] D. Talbot and J. Talbot. Bloom maps. In *Proc. of 4th Workshop on Analytic Algorithmics and Combinatorics 2008 (ANALCO'08)*, pages 203–212, San Francisco, 2008.