Probabilistic Counting with Randomized Storage*

Benjamin Van Durme
University of Rochester
Rochester, NY 14627, USA

Ashwin Lall
Georgia Institute of Technology
Atlanta, GA 30332, USA

Abstract

Previous work by Talbot and Osborne [2007a] explored the use of randomized storage mechanisms in language modeling. These structures trade a small amount of error for significant space savings, enabling the use of larger language models on relatively modest hardware.

Going beyond space efficient *count storage*, here we present the *Talbot Osborne Morris Bloom (TOMB) Counter*, an extended model for performing space efficient *counting* over streams of finite length. Theoretical and experimental results are given, showing the promise of approximate counting over large vocabularies in the context of limited space.

1 Introduction

The relative frequency of a given word, or word sequence, is an important component of most empirical methods in computational linguistics. Traditionally, these frequencies are derived by naively counting occurrences of all interesting word or word sequences that appear within some document collection; either by using something such as an in-memory table mapping labels to positive integers, or by writing such observed sequences to disk, to be later sorted and merged into a final result.

As ever larger document collections become available, such as archived snapshots of Wikipedia [Wikimedia Foundation, 2004], or the results of running a spider bot over significant portions of the internet, the space of interesting word sequences (n-grams) becomes so massive as to overwhelm reasonably available resources. Indeed, even just storing the end result of collecting counts from such large collections has become an active topic of interest (e.g., [Talbot and Osborne, 2007a; Talbot and Branst, 2008]).

As such collections are inherently variable in the empirical counts that they give rise to, we adopt the intuition behind the seminal work of Talbot and Osborne [2007a]: since a fixed corpus is only a sample from the underlying distribution we truly care about, it is not essential that counts stored (or, in our case, collected) be perfectly maintained; trading a small amount of precision for generous space savings can be a worthwhile proposition.

Extending existing work in space efficient *count storage*, where frequencies are known *a priori*, this paper is concerned with the task of online, approximate, space efficient *counting* over streams of fixed, finite length. In our experiments we are concerned with streams over document collections, which can be viewed as being generated by a roughly stationary process: the combination of multiple human language models (authors). However, the techniques described in what follows are applicable in any case where the number of unique items to be counted is great, and the size[1] of each identity is nontrivial.

In what follows, we provide a brief background on the topics of randomized storage and probabilistic counting, moving on to propose a novel data structure for maintaining the frequencies of n-grams either when the frequencies are known in advance or when presented as updates. This data structure, the *Talbot Osborne Morris Bloom (TOMB) Counter*, makes use of both lossy storage, where n-gram identities are discarded and their frequencies stored probabilistically, as well as probabilistic updates in order to decrease the memory footprint of storing n-gram frequencies. These approximations introduce false positives and estimation errors. We provide initial experimental results that demonstrate how the parameters of our data structure can be tuned to trade off between these sources of error.

2 Background

2.1 Randomized Storage of Counters

Cohen and Matias [2003] introduced the Spectral Bloom Filter, a data structure for lossily maintaining frequencies of many different elements without explicitly storing their identities. For a set $U = \{u_1, u_2, u_3, \ldots, u_n\}$ with associated frequencies $f(u_1), f(u_2), \ldots, f(u_n)$, a spectral Bloom filter maintains these frequencies in an array of counters, the length of which is proportional to n but independent of

*This work was supported by a 2008 Provost's Multidisciplinary Award from the University of Rochester, and NSF grants IIS-0328849, IIS-0535105, CNS-0519745, CNS-0626979, and CNS-0716423.

[1] In the sense of space required for representation, e.g., the character string *"global economic outlook"* requires 23 bytes under ASCII encoding.

the description lengths of the elements of U. For example, the space required to store the identity/frequency tuple $\langle$ *"antidisestablishmentarianism"*, $5\,\rangle$ precisely could be replaced by a few bytes' worth of counters. The price that must be paid for such savings is that, with small probability, a spectral Bloom filter may over-estimate the frequency of an element in such a way that it is impossible to detect.

Based on the Bloom Filter of Bloom [1970], the spectral Bloom filter uses hash functions $h_1, \ldots, h_k$ to update an array of counters. When an element is inserted into the filter, the hash functions are used to index the element into the array and increment the counters at those k locations. Querying an array similarly looks up the k locations indexed by the element and returns the minimum of these counts. Owing to the possibility of hash collisions (which result in distinct elements incrementing a counter at the same location), each of the k counters serve as the upper bound on the true count. This data structure hence never underestimates the frequency of any element.

The overestimation probability of the spectral Bloom filter is identical to the false positive probability of a Bloom filter: $(0.6185)^{m/n}$, where m is the number of counters in the array (see Cohen and Matias [2003] for details). What is important to note is that this is the probability with which an element has collisions at every counter. In practice, with long-tailed data sets, we expect these collisions to add only a small amount to the estimate of the frequency of each element.

2.2 Probabilistic Counting of Large Values

Robert Morris once gave the following problem description:

> "An n-bit register can ordinarily only be used to count up to $2^n - 1$. I ran into a programming situation that required using a large number of counters to keep track of the number of occurrences of many different events. The counters were 8-bit bytes and because of the limited amount of storage available on the machine being used, it was not possible to use 16-bit counters. [...] The resulting limitation of the maximum count to 255 led to inaccuracies in the results, since the most common events were recorded as having occurred 255 times when in fact some of them were much more frequent. [...] On the other hand, precise counts of the events were not necessary since the processing was statistical in nature and a reasonable margin of error was tolerable." – Morris [1978]

The proposed solution was simple: by maintaining the value of the logarithm of the frequency, rather than the frequency itself, n bits allow us to count up to 2^{2^n}. Since this necessitates lossy counting, the solution would have to endure some amount of error. What Morris was able to show was that by using a probabilistic counting process he could guarantee a constant relative error estimate for each count with high probability.

Morris counting begins by setting the counter to zero. In order to perform an update (i.e., "to count"), the current stored frequency, $\hat{f}$, is read, and the counter is incremented by 1 with probability $2^{-\hat{f}}$.

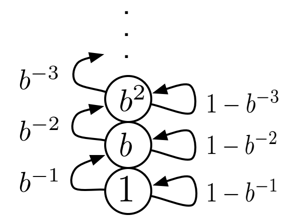


Figure 1: A Morris Counter with base b.

It can be shown [Flajolet, 1985] that after n increments, if the value of the counter is $\hat{f}$, then $\mathrm{E}[2^{\hat{f}} - 1] = n$, which gives us an unbiased estimate of the true frequency. Additionally, the variance of this estimate is $n(n+1)/2$, which results in nearly constant expected relative error.

The disadvantage of keeping counts so succinctly is that estimates of the true frequencies are inaccurate up to a binary order of magnitude. However, Morris gave a way of ameliorating this effect by modifying the *base* of the probabilistic process. Rather than using a base of 2, a smaller base, $b > 1$, may be used in order to obtain similar savings with smaller relative error. The algorithm is modified to update counters with probability $b^{-\hat{f}}$ and estimate the frequency using the unbiased estimate $(b^{\hat{f}} - 1)/(b - 1)$ with variance $(b-1)n(n+1)/2$. Using a smaller base allows for obtaining more accurate estimates at the cost of a smaller maximum frequency, using the same number of bits.

2.3 Approximate Counts in Language Modeling

Talbot and Osborne [2007a] gave a scheme for storing frequencies of n-grams in a succinct manner when all the identity-frequency pairs are known *a priori*. Their method involves first constructing a quantization codebook from the frequencies provided. For each n-gram, it is the resultant quantization point that is stored, rather than the original, true count. To save the space of retaining identities, e.g., the bytes required for storing *"the dog ran"*, Bloom filters were used to record the set of items within each quantization point. In order to minimize error, each n-gram identity was stored in each quantization point up to and including that given by the codebook for the given element. This guarantees that the reported frequency of an item is never under-estimated, and is over-estimated only with low probability. For example, given a binary quantization, and the $\langle$identity, frequency$\rangle$ pair: $\langle$*"the dog ran"*, $32\rangle$, then *"the dog ran"* would be hashed into Bloom filters corresponding to $2^0, 2^1, 2^2, ..., 2^5$. Later, the value of this element can be retrieved by a linear sequence of queries, starting with the filter corresponding to 2^0, up until a result of *false* is reported. The last filter to report positive membership is taken as the true quantization point of the original frequency for the given element. This structure is illustrated in Figure 2.

Recently, Goyal *et al.* [2009] adapted the lossy counting algorithm designed by Manku and Motwani [2002] to contruct high-order approximate n-gram frequency counts. This data structure has tight accuracy guarantees on all n-grams that

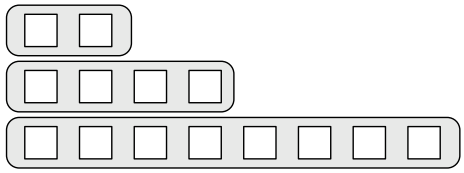


Figure 2: A small, three layer structure of Talbot and Osborne, with widths eight, four and two (written as $(8, 4, 2)$ when using the syntax introduced in Section 3).

have sufficiently large frequencies. In comparison, the parametric structure described herein allows for the allocation of space in order to prioritize the accuracy of estimates over the long tail of low-frequency n-grams as well.

Finally, we note that simple sampling techniques may also be employed for this task (with weaker guarantees than in [Goyal *et al.*, 2009]), but this has the same drawback that we will miss many of the low-frequency events.

3 TOMB Counters

Building off of intuitions from previous work summarized in the last section, as well as the thesis work of Talbot [2009],[2] the following contains a description of this paper's primary contribution: the Talbot Osborne Morris Bloom (TOMB) Counter. We first describe two simpler variants of this structure, which will then both be shown as limit cases within a larger space of potential parametrizations.

3.1 Morris Bloom

We call the combination of a spectral Bloom filter that uses Morris style counting a *Morris Bloom Counter*. This data structure behaves precisely in the same manner as a Bloom filter, with the exception that the counter within each cell of the Bloom filter operates as according to Morris [1978]. For a given Morris Bloom counter, M, we refer to the number of individual counters as M's *width*, while the number of bits per counter is M's *height*; together, these are written as $\langle\, width, height\,\rangle$. An $\langle 8, 3\rangle$ Morris Bloom counter is illustrated in Figure 3.

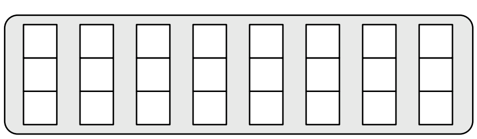


Figure 3: A Morris Bloom Counter of width eight and height three.

The benefit of Morris Bloom counters is that they give randomized storage combined with lossy counting. We are able to do away with the need to store the identity of the elements being inserted, while at the same time count up to, e.g., $2^{2^3} = 256$ with a height of only 3 bits. However, when the Bloom filter errs (with small probability) the amount of error that can be introduced in the frequency can be exceedingly large.

[2]Through personal communication.

3.2 Talbot Osborne Bloom

An alternative method for counting with Bloom filters is to extend the structure of Talbot and Osborne [2007a]. Rather than embedding multi-bit counters within each cell of a single Bloom filter, we can employ multiple layers of traditional filters (i.e., they maintain sets rather than frequencies) and make use of some counting mechanism to *transition* from layer to layer. Such a counter with *depth* 3 and layer widths 8, 4, and 2 is illustrated in Figure 4.

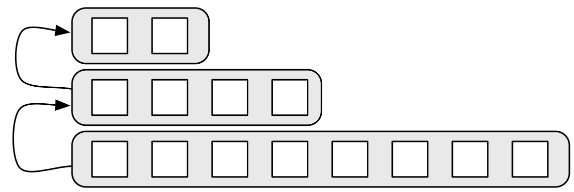


Figure 4: An $(8, 4, 2)$ Talbot Osborne Bloom Counter.

This data structure extends the one of Talbot and Osborne in that, rather than simply storing quantized versions of statically provided frequencies, we now have access to an insertion operation that allows updating in an online manner. Various insertion mechanisms could be used dependent on the context. For example, an exponential quantization similar to Talbot and Osborne's static counter could be replicated in the streaming case by choosing insertion probabilities that decrease exponentially with each successive layer.[3]

There are two advantages to counting with this structure. First, it limits the over-estimate of false positives: since it is unlikely that there will be false positives in several consecutive layers, over-estimates of frequency will often be at worst a single order of magnitude (with respect to some base). Second, since we are able to control the width of individual layers, less space may be allocated to higher layers if it is known that there are relatively few high-frequency elements.

While this arrangement can promise small over-estimation error, a significant drawback compared to the previously described Morris Bloom counter is the limit to expressibility. Since this counter counts in unary, even if a quantization scheme is used to express frequencies succinctly, such a data structure can only express d distinct frequencies when constructed with depth d. In contrast, a Morris Bloom counter with height h can express 2^h different frequencies since it counts in binary.

3.3 Talbot Osborne Morris Bloom

Motivated by the advantages and disadvantages of the two data structures described above, we designed a hybrid arrangement that trades off the expressibility of the Morris

[3]We note that the recently released RandLM package (http://sourceforge.net/projects/randlm) from Talbot and Osborne supports an initial implementation of a structure similar to this.

Bloom counter with the low over-estimation error of the Talbot Osborne Bloom counter. We call this data structure the *Talbot Osborne Morris Bloom (TOMB) Counter*.

The TOMB counter has similar workings to the Talbot Osborne Bloom counter described above, except that we have full spectral Bloom filters at each layer. When an item is inserted into this data structure, the item is iteratively searched for in each layer, starting from the bottom layer. It is probabilistically (Morris) inserted into the first layer in which the spectral Bloom filter has not yet reached its maximum value. Querying is done similarly by searching layers for the first one in which the spectral Bloom filter is not at its maximum value, with the true count then estimated as done by Morris.

Notationally, we denote a depth d TOMB counter as $(\langle w_1, h_1\rangle, \ldots, \langle w_d, h_d\rangle)$, where the w_is are the widths of the counter arrays and the h_is are the heights of the counter at each layer. An $(\langle 8, 2\rangle, \langle 4, 2\rangle)$ TOMB counter is illustrated in Figure 5.

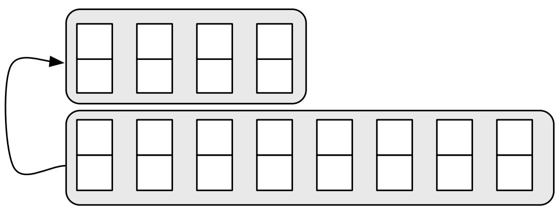


Figure 5: An $(\langle 8, 2\rangle, \langle 4, 2\rangle)$ TOMB Counter.

It can be seen that this structure is a generalization of those given earlier in this section. A TOMB counter of depth 1 (i.e., only a single layer) is identical in function to a Morris Bloom counter. On the other hand, a TOMB counter with multiple layers, each of height one, is identical to a Talbot Osborne Bloom counter. Hence, the limiting cases for a TOMB counter are the structures previously described, and by varying the depth and height parameters we are able to trade off their respective advantages.

Finally, we add to our definition the optional ability for a TOMB counter to *self-loop*, where in the case of counter overflow, transitions occur from cells within the final layer back into this same address space using alternate hash functions. Whether or not to allow self-loops in a transition counter is a practical decision based on the nature of what is being counted and how those counts are expected to be used. Without self-loops, the count for any particular element may overflow. On the other hand, overflow of a handful of elements may be preferable to using the additional storage space that self-loops may absorb.

Note that, in one extreme, a single layer structure of height one, with self-loops, may be used as a TOMB counter.

3.4 Analysis

We give here an analysis of various properties of the transition counter. Two simplifications are made: we assume uniform height h per layer, and that space is allocated between layers in such a manner that the false positive probability is the same for all layers. In the context of text-based streams, this can

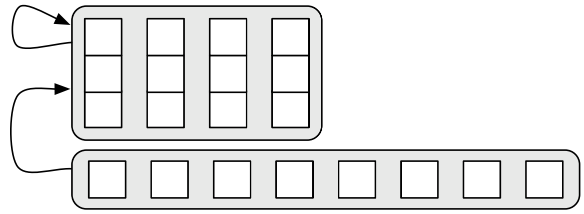


Figure 6: An $(\langle 8, 1\rangle, \langle 4, 3\rangle)$ TOMB Counter, with a self-loop on the final layer.

in practice be approximated by caching some fixed number of initial elements seen in order to know these frequencies exactly and then using the best-fit power-law distribution to decide how much space should be allocated to each layer.[4] We also assume that the final layer has self-loops, but that its false positive probability is still only p.

For a fixed base b, we first study how large a frequency can be stored in a transition counter with heights h and depth d. Since each layer has h bits for each counter, it can count up to $2^h - 1$. As there are d such layers, the largest count that can be stored is $d(2^h - 1)$. This count corresponds to a Morris estimate of:

$$\frac{b^{d(2^h-1)} - 1}{b - 1}.$$

In particular, for the Morris Bloom counter (which has $d = 1$), this works out to $\frac{b^{2^h-1}-1}{b-1}$. On the other hand, the transition Bloom counter (which has $h = 1$) can only count up to $\frac{b^d-1}{b-1}$.

Let us denote the false positive probability of the counting Bloom filters by p. The probability that an element that has not been inserted into this data structure will erroneously be reported is hence the Bloom error of the lowest layer of the counter, which is bounded by p. This suggests that we might wish to allocate a large amount of space to the lowest layer to prevent false positives.

As the probabilistic counting process of Morris is unbiased, we study the bias introduced by hash collisions from the Bloom counters. We leave the problem of analyzing the interaction of the Morris counting process with the transitioning process for future work and here focus specifically on the bias from Bloom error.

Since the height of each counting Bloom filter is h, the first i layers allow us to count up to $\frac{b^{i(2^h-1)}-1}{b-1}$. Hence, the expected bias due to Bloom error can be bounded as:

[4] This assumes the remainder of the observed stream is roughly stationary with respect to the exact values collected; dynamic adjustment of counter structure in the face of streams generated by highly non-stationary distributions is a topic for future work.

$$
\begin{aligned}
\text{E[bias]} &\leq p(1-p)\frac{b^{2^h-1}-1}{b-1} + p^2(1-p)\frac{b^{2(2^h-1)}-1}{b-1} \\
&+ p^3(1-p)\frac{b^{3(2^h-1)}-1}{b-1} + \ldots \\
&\leq p(1-p)\frac{b^{2^h-1}}{b-1} + p^2(1-p)\frac{b^{2(2^h-1)}}{b-1} + \ldots \\
&= \left(\frac{1-p}{b-1}\right)\frac{pb^{2^h-1}}{1-pb^{2^h-1}},
\end{aligned}
$$

where we assume that $pb^{2^h-1} < 1$ for the series to converge.

We can similarly estimate the expected relative bias for for an arbitrary item. Let us assume that the item we are interested in is supposed to be in layer l. Then, similar to above, the expected overestimate of its frequency O due to Bloom error can be bounded as

$$
\begin{aligned}
O &\leq \frac{(1-p)pb^{l(2^h-1)}}{b-1} + \frac{(1-p)p^2b^{(l+1)(2^h-1)}}{b-1} + \ldots \\
&= \left(\frac{1-p}{b-1}\right)\frac{pb^{l(2^h-1)}}{1-pb^{2^h-1}}.
\end{aligned}
$$

On the other hand, since by assumption the item of interest is supposed to be in the lth level, its frequency must be at least $\frac{b^{(l-1)(2^h-1)}}{(b-1)}$. Hence, the expected relative bias is bounded by

$$
(1-p)\frac{pb^{2^h-1}}{1-pb^{2^h-1}}.
$$

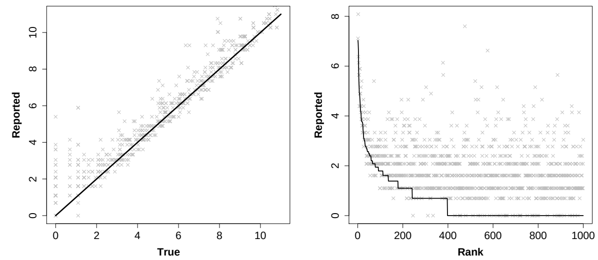
4 Experiments

4.1 Counting Trigrams in Text

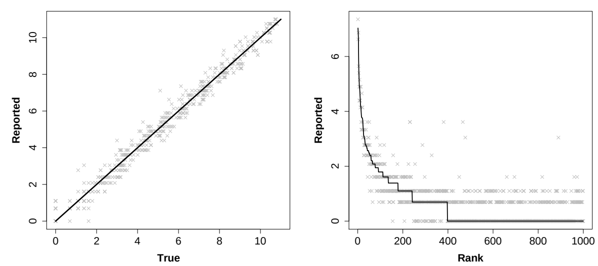
Counters of various size, measured in megabytes (MB), were used to count trigram frequencies in the Gigaword Corpus [Graff, 2003], a large collection of newswire text from a variety of reporting agencies. Trigrams were counted over each sentence in isolation, with the standard inclusion of begin and end sentence tokens, and numeral characters mapped to a single representative digit. The relevant (approximate) statistics for this collection, computed offline, are: 130 million sentences, 3 billion tokens, a vocabulary of 2.4 million distinct elements, 400 million unique trigrams (the set over which we are counting), and a maximum individual trigram count of 9 million. A text file containing just the strings for each such unique Gigaword trigram requires approximately 7GB of memory.

All counters used were of depth 12, with three layers of height 1, followed by 9 layers of height 3. Amongst these layers, half of memory was allocated for the first three layers, half for the later nine. An important point of future work is to develop methods for determining (near) optimal values for these parameters automatically, given at most a constant initial portion of the incoming stream (a technique we refer to as *buffered inspection*).

Figure 7 gives results for querying values from two of the counters constructed. This figure shows that as we move from 100 to 500 megabytes, the resulting counting structure is significantly less saturated, and is thus able to report more reliable estimates.



(a) 100 MB



(b) 500 MB

Figure 7: Results based on querying different size counters built over trigrams in Gigaword. On the left, log-log plots of frequency for elements equally sampled from each range $(\log(i-1)..\log(i)$ for i from 1 to $\lceil\log(\text{max count})\rceil)$ with the x-axis being True values, y-axis being Reported. On the right, values for elements sampled uniformly at random from the collection, presented as Rank versus Reported frequency (log-scale). Solid line refers to true frequency.

4.2 False Positive Rate

The false positive rate, p, of a TOMB counter can be empirically estimated by querying values for a large number of elements known to have been unseen. Let Y be such a set (i.e., $Y \cap U$ is empty).[5] With indicator function $I_{>0}(z)$ equaling 1 when $z > 0$ and 0 otherwise, an estimate for p is then:

$$
\hat{p} = \frac{1}{|Y|}\sum_{y \in Y} I_{>0}(\hat{f}(y)).
$$

Table 1 contains estimated false positive rates for three counters, showing that as more memory is used, the expected reported frequency of an unseen item approaches the true value of 0. Note that these rates could be lowered by allocating a larger percentage of the available memory to the bottom most layers. This would come at the cost of greater saturation in the higher layers, leading to increased relative error for items more frequently seen.

[5]We generate Y by enumerating the strings *"1"* through *"1000"*.

SIZE (MB)	$I_{>0}$	$I_{>1}$	$I_{>2}$	$I_{>3}$
100	86.5%	74.2%	66.1%	43.5%
500	26.9%	6.7%	1.8%	0.3%
2,000	10.9%	0.9%	0.1%	0.0%

Table 1: False positive rates when using indicator functions $I_{>0}$, ..., $I_{>3}$. A perfect counter has a rate of 0.0% using $I_{>0}$.

TRUE	260MB	100MB	50MB	25MB	NO LM
22.75	22.93	22.27	21.59	19.06	17.35
-	22.88	21.92	20.52	18.91	-
-	22.34	21.82	20.37	18.69	-

Table 2: BLEU scores using language models based on true counts, compared to approximations using various size TOMB counters. Three trials for each counter are reported (recall Morris counting is probabilistic, and thus results may vary between similar trials).

4.3 Language Models for Machine Translation

As an example of approximate counts in practice, we follow Talbot and Osborne [2007a] in constructing a n-gram language models for Machine Translation (MT). Experiments compared the use of unigram, bigram and trigram counts stored explicitly in hashtables, to those collected using TOMB counters allowed varying amounts of space. Counters had five layers of height one, followed by five layers of height three, with 75% of available space allocated to the first five layers. Smoothing was performed using Absolute Discounting [Ney *et al.*, 1994] with an *ad hoc* value of $\alpha = 0.75$.

The resultant language models were substituted for the trigram model used in the experiments of Post and Gildea [2008], with counts collected over the same approximately 833 thousand sentences described therein. Explicit, non-compressed storage of these counts required 260 MB. Case-insensitive BLEU-4 scores were computed for those authors' DEV/10 development set, a collection of 371 Chinese sentences comprised of twenty words or less. While more advanced language modeling methods exist (see, e.g., [Yuret, 2008]), our concern here is specifically on the impact of approximate counting with respect to a given framework, relative to the use of actual values.[6]

As shown in Table 2, performance declines as a function of counter size, verifying that the tradeoff between space and accuracy in applications explored by Talbot and Osborne extends to approximate counts collected *online*.

5 Conclusions

Building on existing work in randomized count storage, we have presented a general model for probabilistic counting over large numbers of elements in the context of limited space. We have defined a parametrizable structure, the *Talbot Osborne Morris Bloom (TOMB) counter*, and presented analysis along with experimental results displaying its ability to trade space for loss in reported count accuracy.

Future work includes looking at optimal classes of counters for particular tasks and element distributions. While motivated by needs within the Computational Linguistics community, there are a variety of fields that could benefit from methods for space efficient counting. For example, we've recently begun experimenting with visual n-grams using vocabularies built from SIFT features, based on images from the Caltech-256 Object Category Dataset [**?**].

Finally, developing clever methods for buffered inspection will allow for online parameter estimation, a required ability if TOMB counters are to be best used successfully with no knowledge of the target stream distribution *a priori*.

[6]Post and Gildea report a trigram-based BLEU score of 26.18, using more sophisticated smoothing and backoff techniques.

Acknowledgements The first author benefited from conversations with David Talbot concerning the work of Morris and Bloom, as well as with Miles Osborne on the emerging need for randomized storage. Daniel Gildea and Matt Post provided general feedback and assistance in experimentation.

References

[Bloom, 1970] Burton H. Bloom. Space/time trade-offs in hash coding with allowable errors. *Communications of the ACM*, 13:422–426, 1970.

[Cohen and Matias, 2003] Saar Cohen and Yossi Matias. Spectral Bloom Filters. In *Proceedings of SIGMOD*, 2003.

[Flajolet, 1985] Philippe Flajolet. Approximate counting: a detailed analysis. *BIT*, 25(1):113–134, 1985.

[Goyal *et al.*, 2009] Amit Goyal, Hal Daume III, and Suresh Venkatasubramanian. Streaming for large scale NLP: Language Modeling. In *Proceedings of NAACL*, 2009.

[Graff, 2003] David Graff. English Gigaword. Linguistic Data Consortium, Philadelphia, 2003.

[Manku and Motwani, 2002] Gurmeet Singh Manku and Rajeev Motwani. Approximate frequency counts over data streams. In *Proceedings of VLDB*, 2002.

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

[Ney *et al.*, 1994] Hermann Ney, Ute Essen, and Reinhard Kneser. On structuring probabilistic dependences in stochastic language modeling. *Computer, Speech, and Language*, 8:1–38, 1994.

[Post and Gildea, 2008] Matt Post and Daniel Gildea. Parsers as language models for statistical machine translation. In *Proceedings of AMTA*, 2008.

[Talbot and Branst, 2008] David Talbot and Thorsten Branst. Randomized language models via perfect hash functions. In *Proceedings of ACL*, 2008.

[Talbot and Osborne, 2007a] David Talbot and Miles Osborne. Randomised Language Modelling for Statistical Machine Translation. In *Proceedings of ACL*, Prague, Czech Republic, 2007.

[Talbot and Osborne, 2007b] David Talbot and Miles Osborne. Smoothed Bloom filter language models: Tera-Scale LMs on the Cheap. In *Proceedings of EMNLP*, 2007.

[Talbot, 2009] David Talbot. *Bloom Maps for Big Data*. PhD thesis, University of Edinburgh, 2009.

[Wikimedia Foundation, 2004] Wikimedia Foundation. Wikipedia: The free encyclopedia. http://en.wikipedia.org, 2004.

[Yuret, 2008] Deniz Yuret. Smoothing a tera-word language model. In *Proceedings of ACL*, 2008.