Less Hashing, Same Performance: Building a Better Bloom Filter

Adam Kirsch* and Michael Mitzenmacher**

Division of Engineering and Applied Sciences
Harvard University, Cambridge, MA 02138
{kirsch, michaelm}@eecs.harvard.edu

Abstract. A standard technique from the hashing literature is to use two hash functions $h_1(x)$ and $h_2(x)$ to simulate additional hash functions of the form $g_i(x) = h_1(x) + ih_2(x)$. We demonstrate that this technique can be usefully applied to Bloom filters and related data structures. Specifically, only two hash functions are necessary to effectively implement a Bloom filter without any loss in the asymptotic false positive probability. This leads to less computation and potentially less need for randomness in practice.

1 Introduction

A Bloom filter is a simple space-efficient randomized data structure for representing a set in order to support membership queries. Although Bloom filters allow false positives, the space savings often outweigh this drawback. The Bloom filter and its many variations have proven increasingly important for many applications (see, for example, the survey [3]). Although potential alternatives have been proposed [15], the Bloom filter's simplicity, ease of use, and excellent performance make it a standard data structure that is and will continue to be of great use in many applications. For space reasons, we do not review the standard Bloom filter results; for more background, see [3].

In this paper, we show that applying a standard technique from the hashing literature can simplify the implementation of Bloom filters significantly. The idea is the following: two hash functions $h_1(x)$ and $h_2(x)$ can simulate more than two hash functions of the form $g_i(x) = h_1(x) + ih_2(x)$. (See, for example, Knuth's discussion of open addressing with double hashing [11].) In our context i will range from 0 up to some number $k - 1$ to give k hash functions, and the hash values are taken modulo the size of the relevant hash table. We demonstrate that this technique can be usefully applied to Bloom filters and related data structures. Specifically, only two hash functions are necessary to effectively implement a Bloom filter without any increase in the asymptotic false positive probability. This leads to less computation and potentially less need

* Supported in part by an NSF Graduate Research Fellowship, NSF grants CCR-9983832 and CCR-0121154, and a grant from Cisco Systems.

** Supported in part by NSF grants CCR-9983832 and CCR-0121154 and a grant from Cisco Systems.

Y. Azar and T. Erlebach (Eds.): ESA 2006, LNCS 4168, pp. 456–467, 2006.

for randomness in practice. Specifically, in query-intensive applications where computationally non-trivial hash functions are used (such as in [5, 6]), hashing can be a potential bottleneck in using Bloom filters, and reducing the number of required hashes can yield an effective speedup. This improvement was found empirically in the work of Dillinger and Manolios [5, 6], who suggested using the hash functions $g_i(x) = h_1(x) + ih_2(x) + i^2 \bmod m$, where m is the size of the hash table.

Here we provide a full theoretical analysis that holds for a wide class of variations of this technique, justifies and gives insight into the previous empirical observations, and is interesting in its own right. In particular, our methodology generalizes the standard asymptotic analysis of a Bloom filter, exposing a new convergence result that provides a common unifying intuition for the asymptotic false positive probabilities of the standard Bloom filter and the generalized class of Bloom filter variants that we analyze in this paper. We obtain this result by a surprisingly simple approach; rather than attempt to directly analyze the asymptotic false positive probability, we formulate the initialization of the Bloom filter as a balls-and-bins experiment, prove a convergence result for that experiment, and then obtain the asymptotic false positive probability as a corollary.

We start by analyzing a specific, somewhat idealized Bloom filter variation that provides the main insights and intuition for deeper results. We then move to a more general setting that covers several issues that might arise in practice, such as when the size of the hash table is a power of two as opposed to a prime.

Because of space limitations, we leave some results in the full version of this paper [10]. For example, rate of convergence results appear in the full version [10], although in Section 6 we provide some experimental results showing that the asymptotics kick in quickly enough for this technique to be effective in practice. Also, in the full version we demonstrate the utility of this approach beyond the simple Bloom filter by showing how it can be used to reduce the number of hash functions required for Count-Min sketches [4], a variation of the Bloom filter idea used for keeping approximate counts of frequent items in data streams.

Before beginning, we note that Luecker and Molodowitch [12] and Schmidt and Siegel [17] have shown that in the setting of open addressed hash tables, the double hashing technique gives the same performance as uniform hashing. These results are similar in spirit to ours, but the Bloom filter setting is sufficiently different from that of an open addressed hash table that we do not see a direct connection. We also note that our use of hash functions of the form $g_i(x) = h_1(x) + ih_2(x)$ may appear similar to the use of pairwise independent hash functions, and that one might wonder whether there is any formal connection between the two techniques in the Bloom filter setting. Unfortunately, this is not the case; a straightforward modification of the standard Bloom filter analysis yields that if pairwise independent hash functions are used instead of fully random hash functions, then the space required to retain the same bound on the false positive probability increases by a constant factor. In contrast, we show that using the g_i's causes *no* increase in the false positive probability, so they can truly be used as a replacement for fully random hash functions.

2 A Simple Construction Using Two Hash Functions

As an instructive example case, we consider a specific application of the general technique described in the introduction. We devise a Bloom filter that uses k fully random hash functions on some universe U of items, each with range $\{0, 1, 2, \ldots, p-1\}$ for a prime p. Our hash table consists of $m = kp$ bits; each hash function is assigned a disjoint subarray of p bits in the filter, that we treat as numbered $\{0, 1, 2, \ldots, p-1\}$. Our k hash functions will be of the form $g_i(x) = h_1(x) + ih_2(x) \bmod p$, where $h_1(x)$ and $h_2(x)$ are two independent, uniform random hash functions on the universe with range $\{0, 1, 2, \ldots, p-1\}$, and throughout we assume that i ranges from 0 to $k-1$.

As with a standard partitioned Bloom filter, we fix some set $S \subseteq U$ and initialize the filter with S by first setting all of the bits to 0 and then, for each $x \in S$ and i, setting the $g_i(x)$-th bit of the i-th subarray to 1. For any $y \in U$, we answer a query of the form "Is $y \in S$?" with "Yes" if and only if the $g_i(y)$-th bit of the i-th subarray is 1 for every i. Thus, an item $z \notin S$ generates a false positive if and only if each of its hash locations in the array is also a hash location for some $x \in S$.

The advantage of our simplified setting is that for any two elements $x, y \in U$, exactly one of the following three cases occurs: $g_i(x) \neq g_i(y)$ for all i, or $g_i(x) = g_i(y)$ for exactly one i, or $g_i(x) = g_i(y)$ for all i. That is, because we have partitioned the bit array into disjoint hash tables, each hash function can be considered separately. Moreover, by working modulo p, we have arranged that if $g_i(x) = g_i(y)$ for at least two values of i, then we must have $h_1(x) = h_1(y)$ and $h_2(x) = h_2(y)$, so all hash values are the same. This codifies the intuition behind our result: the most likely way for a false positive to occur is when each element in the Bloom filter set S collides with at most one array bit corresponding to the element generating the false positive; other events that cause an element to generate a false positive occur with vanishing probability. It is this intuition that motivates our analysis; in Section 3, we consider more general cases where other non-trivial collisions can occur.

Proceeding formally, we fix a set $S = \{x_1, x_2, \ldots, x_n\}$ of n elements from U and another element $z \notin S$, and compute the probability that z yields a false positive. A false positive corresponds to the event $\mathcal{F}$ that for each i there is (at least) one j such that $g_i(z) = g_i(x_j)$. Obviously, one way this can occur is if $h_1(x_j) = h_1(z)$ and $h_2(x_j) = h_2(z)$ for some j. The probability of this event $\mathcal{E}$ is

$$\mathbf{Pr}(\mathcal{E}) = 1 - \left(1 - 1/p^2\right)^n = 1 - \left(1 - k^2/m^2\right)^n.$$

Notice that when $m/n = c$ is a constant and k is a constant, as is standard for a Bloom filter, we have $\mathbf{Pr}(\mathcal{E}) = o(1)$. Now since

$$\begin{aligned}\mathbf{Pr}(\mathcal{F}) &= \mathbf{Pr}(\mathcal{F} \mid \mathcal{E})\,\mathbf{Pr}(\mathcal{E}) + \mathbf{Pr}(\mathcal{F} \mid \neg\mathcal{E})\,\mathbf{Pr}(\neg\mathcal{E}) \\ &= o(1) + \mathbf{Pr}(\mathcal{F} \mid \neg\mathcal{E})(1 - o(1)),\end{aligned}$$

it suffices to consider $\mathbf{Pr}(\mathcal{F} \mid \neg\mathcal{E})$ to obtain the (constant) asymptotic false positive probability.

Conditioned on $\neg\mathcal{E}$ and $(h_1(z), h_2(z))$, the pair $(h_1(x_j), h_2(x_j))$ is uniformly distributed over the $p^2 - 1$ values in $V = \{0, \ldots, p-1\}^2 - \{(h_1(z), h_2(z))\}$. Of these, for each $i^* \in \{0, \ldots, k-1\}$, the $p-1$ pairs in

$$V_{i^*} = \{(a, b) \in V \,:\, a \equiv i^*(h_2(z) - b) + h_1(z) \bmod p,\ b \not\equiv h_2(z) \bmod p\}$$

are the ones such that if $(h_1(x_j), h_2(x_j)) \in V_{i^*}$, then i^* is the unique value of i such that $g_i(x_j) = g_i(z)$. We can therefore view the conditional probability as a variant of a balls-and-bins problem. There are n balls (each corresponding to some $x_j \in S$), and k bins (each corresponding to some $i^* \in \{0, \ldots, k-1\}$). With probability $k(p-1)/(p^2-1) = k/(p+1)$ a ball lands in a bin, and with the remaining probability it is discarded; when a ball lands in a bin, the bin it lands in is chosen uniformly at random. What is the probability that all of the bins have at least one ball?

This question is surprisingly easy to answer. By the Poisson approximation, the total number of balls that are not discarded has distribution $\mathrm{Bin}(n, k/(p+1)) \approx \mathrm{Po}(k^2/c)$, where $\mathrm{Bin}(\cdot,\cdot)$ and $\mathrm{Po}(\cdot)$ denote the binomial and Poisson distributions, respectively. Since each ball that is not discarded lands in a bin chosen at random, the joint distribution of the number of balls in the bins is asymptotically the same as the joint distribution of k independent $\mathrm{Po}(k/c)$ random variables, by a standard property of Poisson random variables. The probability that each bin has a least one ball now clearly converges to

$$\mathbf{Pr}(\mathrm{Po}(k/c) > 0)^k = (1 - \exp[-k/c])^k\,,$$

which is the asymptotic false positive probability for a standard Bloom filter, completing the analysis.

We make the above argument much more general and rigorous in Section 3, but for now we emphasize that we have actually characterized much more than just the false positive probability of our Bloom filter variant. In fact, we have characterized the asymptotic joint distribution of the number of items in S hashing to the locations used by some $z \notin S$ as being independent $\mathrm{Po}(k/c)$ random variables. Furthermore, from a technical perspective, this approach appears fairly robust. In particular, the above analysis uses only the facts that the probability that some $x \in S$ shares more than one of z's hash locations is $o(1)$, and that if some $x \in S$ shares exactly one of z's hash locations, then that hash location is nearly uniformly distributed over z's hash locations. These observations suggest that the techniques used in this section can be generalized to handle a much wider class of Bloom filter variants, and form the intuitive basis for the arguments in Section 3.

3 A General Framework

In this section, we introduce a general framework for analyzing Bloom filter variants, such as the one examined in Section 2. We start with some new notation. For any integer ℓ, we define the set $[\ell] = \{0, 1, \ldots, \ell - 1\}$ (note that this definition is slightly non-standard). We denote the support of a random variable X

by $\mathrm{Supp}(X)$. For a multi-set M, we use $|M|$ to denote the number of distinct elements of M, and $\|M\|$ to denote the number of elements of M with multiplicity. For two multi-sets M and M', we define $M \cap M'$ and $M \cup M'$ to be, respectively, the intersection and union of M' as *multi-sets*. Furthermore, in an abuse of standard notation, we define the statement $i, i \in M$ as meaning that i is an element of M of multiplicity at least 2.

We are now ready to define the framework. As before, U denotes the universe of items and $S \subseteq U$ denotes the set of n items for which the Bloom filter will answer membership queries. We define a *scheme* to be a method of assigning hash locations to every element of U. Formally, a scheme is specified by a joint distribution of discrete random variables $\{H(u) : u \in U\}$ (implicitly parameterized by n), where for $u \in U$, $H(u)$ represents the multi-set of hash-locations assigned to u by the scheme. We do not require a scheme to be defined for every value of n, but we do insist that it be defined for infinitely many values of n, so that we may take limits as $n \to \infty$. For example, for the class of schemes discussed in Section 2, we think of the constants k and c as being fixed to give a particular scheme that is defined for those values of n such that $p \stackrel{\text{def}}{=} m/k$ is a prime, where $m \stackrel{\text{def}}{=} cn$. Since there are infinitely many primes, the asymptotic behavior of this scheme as $n \to \infty$ is well-defined and is the same as in Section 2, where we let m be a free parameter and analyzed the behavior as $n, m \to \infty$ subject to m/n and k being fixed constants, and m/k being prime.

Having defined the notion of a scheme, we may now formalize some important concepts with new notation (all of which is implicitly parameterized by n). We define H to be the set of all hash locations that can be assigned by the scheme (formally, H is the set of elements that appear in some multi-set in the support of $H(u)$, for some $u \in U$). For $x \in S$ and $z \in U - S$, define $C(x, z) = H(x) \cap H(z)$ to be the multi-set of hash collisions of x with z. We let $\mathcal{F}(z)$ denote the *false positive event* for $z \in U - S$, which occurs when each of z's hash locations is also a hash location for some $x \in S$.

In the schemes that we consider, $\{H(u) : u \in U\}$ will always be independent and identically distributed. In this case, $\mathbf{Pr}(\mathcal{F}(z))$ is the same for all $z \in U - S$, as is the joint distribution of $\{C(x, z) : x \in S\}$. Thus, to simplify the notation, we may fix an arbitrary $z \in U - S$ and simply use $\mathbf{Pr}(\mathcal{F})$ instead of $\mathbf{Pr}(\mathcal{F}(z))$ to denote the false positive probability, and we may use $\{C(x) : x \in S\}$ instead of $\{C(x, z) : x \in S\}$ to denote the joint probability distribution of the multi-sets of hash collisions of elements of S with z.

The main technical result of this section is the following key theorem, which is a formalization and generalization of the analysis in Section 2.

Theorem 1. *Fix a scheme. Suppose that there are constants λ and k such that:*

1. *$\{H(u) : u \in U\}$ are independent and identically distributed.*
2. *For $u \in U$, $\|H(u)\| = k$.*

3. *For* $x \in S$, $\mathbf{Pr}(\|C(x)\| = i) = \begin{cases} 1 - \frac{\lambda}{n} + o(1/n) & i = 0 \\ \frac{\lambda}{n} + o(1/n) & i = 1 \\ o(1/n) & i > 1 \end{cases}$.

4. *For* $x \in S$, $\max_{i \in H} \left| \mathbf{Pr}(i \in C(x) \mid \|C(x)\| = 1,\ i \in H(z)) - \frac{1}{k} \right| = o(1)$.

Then $\lim_{n \to \infty} \mathbf{Pr}(\mathcal{F}) = \left(1 - \mathrm{e}^{-\lambda/k}\right)^k$.

Proof. For ease of exposition, we assign every element of $H(z)$ a unique number in $[k]$ (treating multiple instances of the same hash location as distinct elements). More formally, we define an arbitrary bijection f_M from M to $[k]$ for every multiset $M \subseteq H$ with $\|M\| = k$ (where f_M treats multiple instances of the same hash location in M as distinct elements), and label the elements of $H(z)$ according to $f_{H(z)}$. This convention allows us to identify the elements of $H(z)$ by numbers $i \in [k]$, rather than hash locations $i \in H$.

For $i \in [k]$ and $x \in S$, define $X_i(x) = 1$ if $i \in C(x)$ and 0 otherwise, and define $X_i \stackrel{\text{def}}{=} \sum_{x \in S} X_i(x)$. Note that $i \in C(x)$ is an abuse of notation; what we really mean is $f^{-1}_{H(z)}(i) \in C(x)$, although we will continue using the former since it is much less cumbersome. We show that $X^n \stackrel{\text{def}}{=} (X_0, \ldots, X_{k-1})$ converges in distribution to a vector $P \stackrel{\text{def}}{=} (P_0, \ldots, P_{k-1})$ of k independent $\mathrm{Po}(\lambda/k)$ random variables as $n \to \infty$. To do this, we make use of moment generating functions. For a random variable R, the moment generating function of R is defined by $M_R(t) \stackrel{\text{def}}{=} \mathbf{E}[\exp(tR)]$. We show that for any $t_0, \ldots, t_k$, $\lim_{n \to \infty} M_{\sum_{i=0}^{k-1} t_i X_i}(t_k) = M_{\sum_{i=0}^{k-1} t_i P_i}(t_k)$, which is sufficient by [1, Theorem 29.4 and p. 390], since $M_{\sum_{i=0}^{k-1} t_i P_i}(t_k) = \exp\left[\frac{\lambda}{k}\left(\sum_{i \in k} \mathrm{e}^{t_k t_i} - 1\right)\right] < \infty$, by an easy calculation. Proceeding, we write

$$M_{\sum_{i \in [k]} t_i X_i}(t_k) = M_{\sum_{i \in [k]} t_i \sum_{x \in S} X_i(x)}(t_k) = M_{\sum_{x \in S} \sum_{i \in [k]} t_i X_i(x)}(t_k)$$
$$= \left(M_{\sum_{i \in [k]} t_i X_i(x)}(t_k)\right)^n,$$

where the first two steps are obvious, and the third step follows from the fact that the $H(x)$'s are independent and identically distributed (for $x \in S$) conditioned on $H(z)$, so the $\sum_{i \in [k]} t_i X_i(x)$'s are too, since each is a function of the corresponding $H(x)$. Continuing, we have (as $n \to \infty$)

$$\left(M_{\sum_{i \in [k]} t_i X_i(x)}(t_k)\right)^n$$
$$= \Bigg(\mathbf{Pr}(\|C(x)\| = 0) + \sum_{j=1}^{k} \mathbf{Pr}(\|C(x)\| = j)$$
$$\times \sum_{T \subseteq [k] : |T| = j} \mathbf{Pr}(C(x) = f^{-1}_{H(z)}(T) \mid \|C(x)\| = j) \mathrm{e}^{t_k \sum_{i \in T} t_i} \Bigg)^n$$
$$= \left(1 - \frac{\lambda}{n} + \frac{\lambda \sum_{i \in [k]} \mathrm{e}^{t_k t_i}}{kn} + o(1/n)\right)^n$$
$$\to \mathrm{e}^{-\lambda + \frac{\lambda}{k} \sum_{i \in [k]} \mathrm{e}^{t_k t_i}} = \mathrm{e}^{\frac{\lambda}{k}\left(\sum_{i \in [k]} \left(\mathrm{e}^{t_k t_i} - 1\right)\right)} = M_{\sum_{i \in [k]} t_i \mathrm{Po}_i(\lambda_k)}(t_k).$$

The first step follows from the definition of the moment generating function. The second step follows from the assumptions on the distribution of $C(x)$ (the conditioning on $i \in H(z)$ is implicit in our convention that associates integers in $[k]$ with the elements of $H(z)$). The next two steps are obvious, and the last step follows from a previous computation.

We have now established that X^n converges to P in distribution as $n \to \infty$. Standard facts from probability theory [1] now imply that as $n \to \infty$,

$$\mathbf{Pr}(\mathcal{F}) = \mathbf{Pr}(\forall i \in [k], X_i > 0) \to \mathbf{Pr}(\forall i \in [k], P_i > 0) = \left(1 - \mathrm{e}^{-\lambda/k}\right)^k.$$

□

It turns out that the conditions of Theorem 1 can be verified very easily in many cases.

Lemma 1. *Fix a scheme. Suppose that there are constants λ and k such that:*

1. *$\{H(u) : u \in U\}$ are independent and identically distributed.*
2. *For $u \in U$, $\|H(u)\| = k$.*
3. *For $u \in U$, $\max_{i \in H} \left|\mathbf{Pr}(i \in H(u)) - \frac{\lambda}{kn}\right| = o(1/n)$.*
4. *For $u \in U$, $\max_{i_1, i_2 \in H} \mathbf{Pr}(i_1, i_2 \in H(u)) = o(1/n)$.*
5. *The set of all possible hash locations H satisfies $|H| = O(n)$.*

Then the conditions of Theorem 1 hold (with the same values for λ and k), and so the conclusion does as well.

Remark 1. Recall that, under our notation, the statement $i, i \in H(u)$ is true if and only if i is an element of $H(u)$ of multiplicity at least 2.

Proof. The proof is essentially just a number of applications of the first two Boole-Bonferroni inequalities. For details, see [10].

4 Some Specific Schemes

We are now ready to analyze some specific schemes. In particular, we examine a natural generalization of the scheme described in Section 2, as well as the double hashing and extended double hashing schemes introduced in [5, 6]. In both of these cases, we consider a Bloom filter consisting of an array of $m = cn$ bits and k hash functions, where $c > 0$ and $k \geq 1$ are fixed constants. The nature of the hash functions depends on the particular scheme under consideration.

4.1 Partition Schemes

First, we consider the class of *partition schemes*, where the Bloom filter is defined by an array of m bits that is partitioned into k disjoint arrays of $m' = m/k$ bits (we require that m be divisible by k), and an item $u \in U$ is hashed to location

$$h_1(u) + ih_2(u) \bmod m'$$

of array i, for $i \in [k]$, where h_1 and h_2 are independent fully random hash functions with codomain $[m']$. Note that the scheme analyzed in Section 2 is a partition scheme where m' is prime (and so is denoted by p in Section 2).

Unless otherwise stated, henceforth we do all arithmetic involving h_1 and h_2 modulo m'. We prove the following theorem concerning partition schemes.

Theorem 2. *For a partition scheme,* $\lim_{n\to\infty} \mathbf{Pr}(\mathcal{F}) = \left(1 - \mathrm{e}^{-k/c}\right)^k$.

Proof. We show that the $H(u)$'s satisfy the conditions of Lemma 1 with $\lambda = k^2/c$. For $i \in [k]$ and $u \in U$, define $g_i(u) = (i, h_1(u) + ih_2(u))$ and $H(u) = (g_i(u) : i \in [k])$. That is, $g_i(u)$ is u's ith hash location, and $H(u)$ is the multi-set of u's hash locations. This notation is obviously consistent with the definitions required by Lemma 1.

Since h_1 and h_2 are independent and fully random, the first two conditions are trivial. The last condition is also trivial, since there are $m = cn$ possible hash locations. For the remaining two conditions, fix $u \in U$. Observe that for $(i, r) \in [k] \times [m']$,

$$\mathbf{Pr}((i, r) \in H(u)) = \mathbf{Pr}(h_1(u) = r - ih_2(u)) = 1/m' = (k^2/c)/kn,$$

and that for distinct $(i_1, r_1), (i_2, r_2) \in [k] \times [m']$, we have

$$\begin{aligned}
&\mathbf{Pr}((i_1, r_1), (i_2, r_2) \in H(u)) \\
&= \mathbf{Pr}(i_1 \in H(u))\, \mathbf{Pr}(i_2 \in H(u) \mid i_1 \in H(u)) \\
&= \frac{1}{m'}\, \mathbf{Pr}(h_1(u) = r_2 - i_2 h_2(u) \mid h_1(u) = r_1 - i_1 h_2(u)) \\
&= \frac{1}{m'}\, \mathbf{Pr}((i_1 - i_2)h_2(u) = r_1 - r_2) \\
&\leq \frac{1}{m'} \cdot \frac{\gcd(|i_2 - i_1|, m')}{m'} \leq \frac{k}{(m')^2} = o(1/n),
\end{aligned}$$

where the fourth step is the only nontrivial step, and it follows from the standard fact that for any $r, s \in [m]$, there are at most $\gcd(r, m)$ values $t \in [m]$ such that $rt \equiv s \bmod m$ (see, for example, [9, Proposition 3.3.1]). Finally, since it is clear that from the definition of the scheme that $|H(u)| = k$ for all $u \in U$, we have that for any $(i, r) \in [k] \times [m']$, $\mathbf{Pr}((i, r), (i, r) \in H(u)) = 0$. □

4.2 (Extended) Double Hashing Schemes

Next, we consider the class of double hashing and extended double hashing schemes, which are analyzed empirically in [5, 6]. In these schemes, an item $u \in U$ is hashed to location

$$h_1(u) + ih_2(u) + f(i) \bmod m$$

of the array of m bits, for $i \in [k]$, where h_1 and h_2 are independent fully random hash functions with codomain $[m]$, and $f : [k] \to [m]$ is an arbitrary function.

When $f(i) \equiv 0$, the scheme is called a *double hashing scheme*. Otherwise, it is called an *extended double hashing scheme (with f)*. We show that the asymptotic false positive probability for an (extended) double hashing scheme is the same as for a standard Bloom filter. The proof is analogous to the proof of Theorem 2. For details, see the technical report version of this paper [10].

Theorem 3. *For any (extended) double hashing scheme,*

$$\lim_{n\to\infty} \mathbf{Pr}(\mathcal{F}) = \left(1 - \mathrm{e}^{-k/c}\right)^k.$$

5 Multiple Queries

In the previous sections, we analyzed the behavior of $\mathbf{Pr}(\mathcal{F}(z))$ for some fixed z and moderately sized n. Unfortunately, this quantity is not directly of interest in most applications. Instead, one is usually concerned with certain characteristics of the distribution of the number of elements in a sequence (of distinct elements) $z_1, \ldots, z_\ell \in U - S$ for which $\mathcal{F}(z)$ occurs. In other words, rather than being interested in the probability that a particular false positive occurs, we are concerned with, for example, the fraction of distinct queries on elements of $U - S$ posed to the filter for which it returns false positives. Since $\{\mathcal{F}(z) : z \in U - S\}$ are not independent, the behavior of $\mathbf{Pr}(\mathcal{F})$ alone does not directly imply results of this form. This section is devoted to overcoming this difficulty.

We start with a definition.

Definition 1. *Consider any scheme where $\{H(u) : u \in U\}$ are independent and identically distributed. Write $S = \{x_1, \ldots, x_n\}$. The* false positive rate *is defined to be the random variable $R = \mathbf{Pr}(\mathcal{F} \mid H(x_1), \ldots, H(x_n))$.*

The false positive rate gets its name from the fact that, conditioned on R, the events $\{\mathcal{F}(z) : z \in U - S\}$ are independent with common probability R. Thus, the fraction of a large number of queries on elements of $U - S$ posed to the filter for which it returns false positives is very likely to be close to R. In this sense, R, while a random variable, acts like a rate for $\{\mathcal{F}(z) : z \in U - S\}$.

It is important to note that in much of literature concerning standard Bloom filters, the false positive rate is not defined as above. Instead the term is often used as a synonym for the false positive probability. Indeed, for a standard Bloom filter, the distinction between the two concepts as we have defined them is unimportant in practice, since one can easily show that R is very close to $\mathbf{Pr}(\mathcal{F})$ with extremely high probability (see, for example, [13]). It turns out that this result generalizes very naturally to the framework presented in this paper, and so the practical difference between the two concepts is largely unimportant even in our very general setting. However, the proof is more complicated than in the case of a standard Bloom filter, and so we must be careful to use the terms as we have defined them.

We give only an outline of our results here, deferring the details to [10]. First, we use a standard Doob martingale argument to apply the Azuma-Hoeffding

inequality to R, which tells us that R is concentrated around $\mathbf{E}[R] = \mathbf{Pr}(\mathcal{F})$. We then use that result to prove versions of the strong law of large numbers, the weak law of large numbers, Hoeffding's inequality, and the central limit theorem.

6 Experiments

In this section, we evaluate the theoretical results of the previous sections empirically for small values of n. We are interested in the following specific schemes: the standard Bloom filter scheme, the partition scheme, the double hashing scheme, and the extended double hashing schemes where $f(i) = i^2$ and $f(i) = i^3$.

For $c \in \{4, 8, 12, 16\}$, we do the following. First, compute the value of $k \in \{\lfloor c \ln 2 \rfloor, \lceil c \ln 2 \rceil\}$ that minimizes $p = (1 - \exp[-k/c])^k$. Next, for each of the schemes under consideration, repeat the following procedure 10,000 times: instantiate the filter with the specified values of n, c, and k, populate the filter with a set S of n items, and then query $\lceil 10/p \rceil$ elements not in S, recording the number Q of those queries for which the filter returns a false positive. We then approximate the false positive probability of the scheme by averaging the results over all 10,000 trials. We use the standard Java pseudorandom number generator to simulate independent hash values.

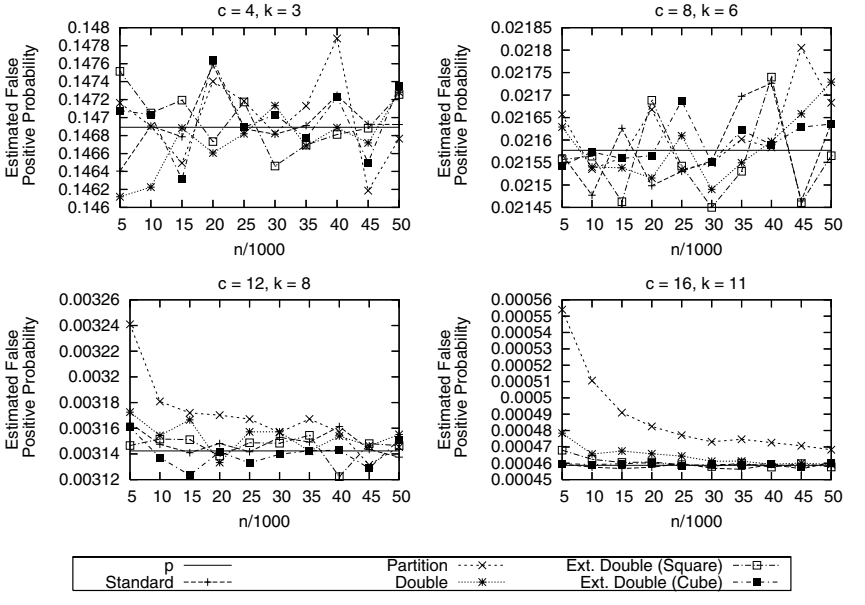

Fig. 1. Estimates of the false positive probability for various schemes and parameters

The results are shown in Figure 1. In Figure 1, we see that for small values of c, the different schemes are essentially indistinguishable from each other, and simultaneously have a false positive probability/rate close to p. This result is

particularly significant since the filters that we are experimenting with are fairly small, supporting our claim that these schemes are useful even in settings with very limited space. However, we also see that for the slightly larger values of $c \in \{12, 16\}$, the partition scheme is no longer particularly useful for small values of n, while the other schemes are. This result is not particularly surprising, since we know from [10, Section 6] that all of these schemes are unsuitable for small values of n and large values of c. Furthermore, we expect that the partition scheme is the least suited to these conditions, given the standard fact that the partitioned version of a standard Bloom filter never performs better than the original version. Nevertheless, the partition scheme might still be useful in certain settings, since it gives a substantial reduction in the range of the hash functions.

7 Conclusion

Bloom filters are simple randomized data structures that are extremely useful in practice. In fact, they are so useful that any significant reduction in the time required to perform a Bloom filter operation immediately translates to a substantial speedup for many practical applications. Unfortunately, Bloom filters are so simple that they do not leave much room for optimization.

This paper focuses on modifying Bloom filters to use less of the only resource that they traditionally use liberally: (pseudo)randomness. Since the only nontrivial computations performed by a Bloom filter are the constructions and evaluations of pseudorandom hash functions, any reduction in the required number of pseudorandom hash functions yields a nearly equivalent reduction in the time required to perform a Bloom filter operation (assuming, of course, that the Bloom filter is stored entirely in memory, so that random accesses can be performed very quickly).

We have shown that a Bloom filter can be implemented with only two pseudorandom hash functions without any increase in the asymptotic false positive probability. We have also shown that the asymptotic false positive probability acts, for all practical purposes and reasonable settings of a Bloom filter's parameters, like a false positive rate. This result has enormous practical significance, since the analogous result for standard Bloom filters is essentially the theoretical justification for their extensive use.

More generally, we have given a framework for analyzing modified Bloom filters, which we expect will be used in the future to refine the specific schemes that we analyzed in this paper. We also expect that the techniques used in this paper will be usefully applied to other data structures, as demonstrated by our modification to the Count-Min sketch (in [10]).

Acknowledgements

We are very grateful to Peter Dillinger and Panagiotis Manolios for introducing us to this problem, providing us with advance copies of their work, and also for many useful discussions.

References

1. P. Billingsley. Probability and Measure, Third Edition. John Wiley & Sons, 1995.
2. P. Bose, H. Guo, E. Kranakis, A. Maheshwari, P. Morin, J. Morrison, M. Smid, and Y. Tang. On the false-positive rate of Bloom filters. Submitted. `http://cg.scs.carleton.ca/~morin/publications/ds/bloom-submitted.pdf`
3. A. Broder and M. Mitzenmacher. Network Applications of Bloom Filters: A Survey. *Internet Mathematics*, 1(4):485-509, 2004.
4. G. Cormode and S. Muthukrishnan. Improved Data Stream Summaries: The Count-Min Sketch and its Applications. DIMACS Technical Report 2003-20, 2003.
5. P. C. Dillinger and P. Manolios. Bloom Filters in Probabilistic Verification. In *Proceedings of the 5th International Conference on Formal Methods in Computer-Aided Design (FMCAD 2004)*, pp. 367-381, 2004.
6. P. C. Dillinger and P. Manolios. Fast and Accurate Bitstate Verification for SPIN. In *Proceedings of the 11th International SPIN Workshop on Model Checking of Software (SPIN 2004)*, pp. 57-75, 2004.
7. D. P. Dubhashi and D. Ranjan. Balls and Bins: A Case Study in Negative Dependence. *Random Structures and Algorithms*, 13(2):99-124, 1998.
8. L. Fan, P. Cao, J. Almeida, and A. Z. Broder. Summary cache: a scalable wide-area Web cache sharing protocol. *IEEE/ACM Transactions on Networking*, 8(3):281-293, 2000.
9. K. Ireland and M. Rosen. A Classical Introduction to Modern Number Theory, Second Edition. Springer-Verlag, 1990.
10. A. Kirsch and M. Mitzenmacher. Building a Better Bloom Filter. Harvard University Computer Science Technical Report TR-02-05, 2005. `ftp://ftp.deas.harvard.edu/techreports/tr-02-05.pdf`.
11. D. Knuth. The Art of Computer Programming, Volume 3: Sorting and Searching. Addison-Wesley, 1973.
12. G. Lueker and M. Molodowitch. More analysis of double hashing. In *Proceedings of the Twentieth Annual ACM Symposium on Theory of Computing (STOC '88)*, pp. 354-359, 1988.
13. M. Mitzenmacher. Compressed Bloom Filters. *IEEE/ACM Transactions on Networking*, 10(5):613-620, 2002.
14. M. Mitzenmacher and E. Upfal. Probability and Computing: Randomized Algorithms and Probabilistic Analysis. Cambridge University Press, 2005.
15. A. Pagh, R. Pagh, and S. Srinivas Rao. An Optimal Bloom Filter Replacement. In *Proceedings of the Sixteenth Annual ACM-SIAM Symposium on Discrete Algorithms (SODA '05)*, pp. 823-829, 2005.
16. M. V. Ramakrishna. Practical performance of Bloom filters and parallel free-text searching. *Communications of the ACM*, 32(10):1237-1239, 1989.
17. J. P. Schmidt and A. Siegel. The analysis of closed hashing under limited randomness. In *Proceedings of the Twenty-Second Annual ACM Symposium on Theory of Computing (STOC '90)*, pp. 224-234, 1990.