Distance-Sensitive Bloom Filters

Adam Kirsch [*] Michael Mitzenmacher [†]

Abstract

A Bloom filter is a space-efficient data structure that answers set membership queries with some chance of a false positive. We introduce the problem of designing generalizations of Bloom filters designed to answer queries of the form, "Is x close to an element of S?" where closeness is measured under a suitable metric. Such a data structure would have several natural applications in networking and database applications.

We demonstrate how appropriate data structures can be designed using locality-sensitive hash functions as a building block, and we specifically analyze the performance of a natural scheme under the Hamming metric.

1 Introduction

A Bloom filter is a simple, space-efficient, randomized data structure that allows one to answer set membership queries with a small but constant probability of a false positive.[1] Bloom filters have found numerous uses, particularly in distributed databases and networking (see, e.g. [2, 9, 10]). Here we initiate a new direction in the study of Bloom filters by considering *distance-sensitive Bloom filters* that answer approximate set membership queries in the following sense: given a metric space (U, d), a finite set $S \subset U$, and parameters $0 \leq \varepsilon < \delta$, the filter aims to effectively distinguish between inputs $u \in U$ such that $d(u, x) \leq \varepsilon$ for some $x \in S$ and inputs $u \in U$ such that $d(u, x) \geq \delta$ for every $x \in S$. Our constructions allow false positives and false negatives. By comparison, the standard Bloom filter corresponds to the case where $\varepsilon = 0$ and δ is any positive constant, and it only gives false positives.

We establish a framework for constructing distance-sensitive Bloom filters when the metric d admits a locality-sensitive hash family (see, e.g., [4, 5, 7]). The potential benefits of this type of data structure are its speed and space; it can provide a quick answer without performing comparisons against the entire set, or even without performing a full nearest-neighbor query, and it should require less space than the original data. For example, in a distributed setting, client processes might use such a filter to determine whether sending a nearest neighbor query to a server would be worthwhile without actually querying the server; if the filter indicates that there is no sufficiently close neighbor to the query, then the request can be skipped. Of course, in all applications the consequences of and tradeoffs between false positives and false negatives need to be considered carefully.

As an example of a possible application of a distance-sensitive Bloom filter, consider a large database that identifies characteristics of people using a large number of fields. Given a vector of characteristic values, one may want to know if there is a person in the database that matches on a large fraction of the characteristics. One can imagine this being useful for employment databases, where one is seeking a candidate matching a list of attributes. A company may wish to provide a distance-sensitive Bloom filter as a way of advertising the utility of its database while providing very little information about its actual content. Similarly, the data structure may also be useful for criminal identification; it could provide a quick spot-test for whether a suspect in custody matches characteristics of suspects for other unsolved crimes.

As a networking example, the SPIE system for IP traceback represents packet headers seen over a small interval of time by a Bloom filter [16]. The authors of that work found that packet headers usually remain consistent as packets travel among routers. A distance-sensitive Bloom filter might allow small changes in the packet header bits while still allowing the Bloom filter to answer queries appropriately.

As a very general example, suppose that we have a collection of sets, with each set being represented by a Bloom filter. For example, a Bloom filter might represent packet traces as above, or a sketch of a document as in [3]. The relative Hamming distance between two Bloom filters (of the same size, and created with the same hash functions) can be used as a measure

[*] Division of Engineering and Applied Sciences, Harvard University, Cambridge, MA 02138. Supported in part by an NSF Graduate Research Fellowship and NSF grants CCR-9983832 and CCR-0121154. Email: kirsch@eecs.harvard.edu

[†] Division of Engineering and Applied Sciences, Harvard University, Cambridge, MA 02138. Supported in part by NSF grants CCR-9983832 and CCR-0121154. Email: michaelm@eecs.harvard.edu

[1] We assume some familiarity with Bloom filters throughout the paper; see [2] for background.

Copyright © by SIAM. Unauthorized reproduction of this article is prohibited

of the similarity of the underlying sets (see, e.g., [2]). Hence, one can construct a distance-sensitive Bloom filter on top of such a collection of Bloom filters to attempt to quickly and easily answer questions of the form, "Are there any sets in the collection very close to this query set?" Such a general construction may prove useful for other distributed applications where Bloom filters are used. For this reason, we pay special attention to the case where $U = \{0,1\}^\ell$ and d is the relative Hamming metric on U.

A more distant but potentially very exciting application of distance-sensitive Bloom filters is in the context of DNA sequences. One might hope that such a filter could effectively handle queries when d is chosen to be the edit distance metric, in order to answer questions of the form, "Is there a DNA sequence close to this one in your database?" Unfortunately, edit distance currently appears to be too difficult to adequately handle in our framework, as there is no known good locality-sensitive hash function for edit distance, although there is recent work attempting to connect edit distance and Hamming distance via various reductions [7, 11]. Similar problems arise in computer virus detection, and ad hoc variations of Bloom filters have recently been used in this setting [15].

Although there are many potential applications, this problem does not appear to have been the subject of much study. Manber and Wu [8] considered the problem of handling a single character error using Bloom filters in the context of password security. Work on nearest-neighbor problems (including [4, 5, 7]) is clearly related, but the problem setting is not equivalent. Our work also seems similar in spirit to work on property testing [14, 6], and specifically to the recently introduced notion of tolerant property testing [13], although here the task is to design a *structure* based on an input set that will allow quick subsequent testing of closeness to that set, instead of an algorithm to test closeness of an object to some property.

Our main contributions in this paper are therefore

1. introducing the formal problem of developing Bloom filter variants that effectively determine whether queries are *close* to an item in a particular set,
2. developing the connection between this problem and locality-sensitive hashing, and
3. examining in detail the case where $U = \Sigma^\ell$ and d is the relative Hamming metric.

Our initial results are not as strong as one might hope. For example, when $U = \{0,1\}^\ell$, d is the relative Hamming metric on U, and $|S| = n$, our distance-sensitive Bloom filter is only efficient and effective for constant δ when $\varepsilon = O(1/\log n)$. That is, we can only differentiate between query strings that differ from all strings in S on a (constant) δ-fraction of bits and query strings that share a $1 - \varepsilon = 1 - O(1/\log n)$-fraction of bits with some string in S. We would prefer ε to be constant. Nevertheless, our experiments suggest that even with this limitation distance-sensitive Bloom filters work sufficiently well to be useful in practice.

Our work leaves many additional open problems. Indeed, one obvious question is whether Bloom filters provide the appropriate paradigm for this problem; alternatives to standard Bloom filters have recently received study [12]. Our major reasons for initiating our study with Bloom filters are because they provide a natural theoretical framework and because Bloom filters are already widely accepted, understood, and used by practitioners.

2 A General Approach

This section gives a general approach to designing distance-sensitive Bloom filters for metric spaces (U, d) that admit a locality-sensitive hash family [5].

Definition 2.1. *A family $\mathcal{H} = \{h : U \to V\}$ is* (r_1, r_2, p_1, p_2)-sensitive *with respect to a metric space (U, d) if $r_1 < r_2$, $p_1 > p_2$, and for any $x, y \in U$,*

- *if $d(x,y) \leq r_1$ then $\Pr_{h \leftarrow \mathcal{H}}(h(x) = h(y)) \geq p_1$, and*
- *if $d(x,y) > r_2$ then $\Pr_{h \leftarrow \mathcal{H}}(h(x) = h(y)) \leq p_2$.*

We say that any such family is a (U, d)-locality-sensitive hash (LSH) family*, omitting (U, d) when the meaning is clear.*

It turns out that our approach is more effectively expressed when we generalize Definition 2.1.

Definition 2.2. *Let (U, d) be a metric space, and let $p_L : \mathbb{R}_{\geq 0} \to [0,1]$ and $p_H : \mathbb{R}_{\geq 0} \to [0,1]$ be non-increasing. A hash family $\mathcal{H} : U \to V$ is called* (p_L, p_H)-distance sensitive *(with respect to (U, d)) if for all $x, y \in U$*

$$p_L(d(x,y)) \leq \Pr_{h \leftarrow \mathcal{H}}(h(x) = h(y)) \leq p_H(d(x,y)).$$

We note that Definition 2.2 really does generalize Definition 2.1, since for any (r_1, r_2, p_1, p_2)-locality-sensitive hash family $\mathcal{H}$, we may set

$$p_L(r) = \begin{cases} p_1 & \text{if } r \leq r_1 \\ 0 & \text{otherwise} \end{cases}$$

Copyright © by SIAM. Unauthorized reproduction of this article is prohibited

and

$$p_H(r) = \begin{cases} p_2 & \text{if } r > r_2 \\ 1 & \text{otherwise} \end{cases}$$

and get that $\mathcal{H}$ is a (p_L, p_H)-distance-sensitive hash family.

We are now ready to present our first and most general distance-sensitive Bloom filter construction, which is essentially a standard partitioned Bloom filter where the random hash functions are replaced by distance-sensitive hash functions. Let $\mathcal{H} : U \to V$ be a (p_L, p_H)-distance sensitive hash function, fix some $S \subset U$ with n elements, and let A be an array consisting of k disjoint m'-bit arrays, $A[1, 1], \ldots, A[k, m']$ (for a total of $m = km'$ bits), where k and m' are parameters. If $V \neq [m']$, then let $\mathcal{H}' : V \to [m']$ be a weakly pairwise-independent hash family.

To initialize the filter, we first choose $h_1, \ldots, h_k \leftarrow \mathcal{H}$ independently. If $V \neq [m']$, then we also choose $h'_1, \ldots, h'_k \leftarrow \mathcal{H}'$ independently, and define $g_i = h'_i \circ h_i$ for $i \in [k]$. If $V = [m']$, then we define $g_i = h_i$ for $i \in [k]$. Next, we set all of the bits in A to zero. Finally, for $x \in S$ and $i \in [k]$, we set $A[i, g_i(x)] = 1$.

It remains to specify how to use the filter to effectively determine whether a query $u \in U$ is close to some $x \in S$. In a standard Bloom filter, to answer queries of the form, "Is $u \in S$?" we simply check if all of u's hash locations in A (that is, the set of bits $\{A[i, h_i(u)] : i \in [k]\}$) are set to 1; we return "$u \in S$" if this is the case and we return "$u \notin S$" otherwise. In our setting, we must consider that u might not be in S but still be close to some element of S, in which case it might be that not all of u's hash locations are set to 1. We consider instead the set of u's hash locations that are set to 1; specifically, by symmetry, it suffices to consider the number $B(u)$ of u's hash locations that are set to 1.

Now, it is easy to see that $B(u) = \sum_{i \in [k]} A[i, g_i(u)]$. As the $A[i, g_i(u)]$'s are independent and identically distributed bits, $B(u) \sim \text{Bin}(k, q_u)$ for some $q_u \in [0, 1]$ (where $\text{Bin}(t, r)$ denotes the binomial distribution with t trials and common success probability r). We derive upper and lower bounds on q_u.

Proposition 2.1. *In an abuse of notation, we define $d(u, S) = \min_{x \in S} d(u, x)$ for $u \in U$. Let $\mathbf{1}(\cdot)$ denote the indicator function. For any $u \in U$,*

$$p_L(d(u, S)) \leq q_u \leq \sum_{x \in S} p_H(d(x, u)) + \frac{nk}{m} \cdot \mathbf{1}(V \neq [m'])$$

Proof. For the lower bound, we write

$$\begin{aligned} q_u &= \Pr(A[1, g_1(u) = 1]) \\ &= \Pr(\exists x \in S : g_1(x) = g_1(u)) \\ &\geq \max_{x \in S} \Pr(g_1(x) = g_1(u)) \\ &\geq \max_{x \in S} p_L(d(x, u)) \\ &= p_L(d(u, S)), \end{aligned}$$

where the last step follows from the fact that p_L is non-increasing.

For the upper bound, we write

$$\begin{aligned} q_u &= \Pr(A[1, g_1(u) = 1]) \\ &= \Pr(\exists x \in S : g_1(x) = g_1(u)) \\ &\leq \sum_{x \in S} \Pr(g_1(x) = g_1(u)). \end{aligned}$$

Now, if $V = [m']$, then for any $x \in S$,

$$\Pr(g_1(x) = g_1(u)) = \Pr(h_1(x) = h_1(u)) \leq p_H(d(x, u)),$$

and if $V \neq [m']$, then for any $x \in S$,

$$\begin{aligned} &\Pr(g_1(x) = g_1(u)) \\ &\leq \Pr(h_1(x) = h_1(u)) \\ &\qquad + \Pr(h'_1(h_1(x)) = h'_1(h_1(u)) \mid h_1(x) \neq h_1(u)) \\ &\leq p_H(d(x, u)) \\ &\qquad + \Pr(h'_1(h_1(x)) = h'_1(h_1(u)) \mid h_1(x) \neq h_1(u)) \\ &\leq p_H(d(x, u)) + \frac{1}{m'}, \end{aligned}$$

so

$$\begin{aligned} q_u &\leq \sum_{x \in S} \Pr(g_1(x) = g_1(u)) \\ &\leq \sum_{x \in S} \left(p_H(d(x, u)) + \frac{1}{m'} \cdot \mathbf{1}(V \neq m') \right) \\ &= \sum_{x \in S} p_H(d(x, u)) + \frac{nk}{m} \cdot \mathbf{1}(V \neq m'). \end{aligned}$$

□

For real-valued random variables X and Y, we write $X \leq_{\text{st}} Y$ or if Y stochastically dominates X. That is $X \leq_{\text{st}} Y$ if for all $x \in \mathbb{R}$, $\Pr(X \geq x) \leq \Pr(Y \geq x)$. The following corollary now follows readily from Proposition 2.1.

Corollary 2.1. *For any $u \in U$,*

$$\begin{aligned} &\text{Bin}(k, p_L(d(u, S))) \\ &\leq_{\text{st}} B(u) \\ &\leq_{\text{st}} \text{Bin}\left(k, \sum_{x \in S} p_H(d(x, u)) + \frac{nk}{m} \cdot \mathbf{1}(V \neq [m'])\right). \end{aligned}$$

Copyright © by SIAM. Unauthorized reproduction of this article is prohibited

Corollary 2.1 suggests that if the filter is configured properly then whenever $u \in U$ is particularly close to some $x \in S$ and $v \in U$ is particularly far from every $x \in S$, we should have that q_u is substantially larger than q_v. Since binomial distributions are reasonably well-concentrated around their expectations, this intuition suggests the existence of some threshold t (that does not depend on the particular strings in S) such that $B(u)$ is unlikely to be below t and $B(v)$ is unlikely to be above t. It follows that for any $w \in U$ that is either particularly close to some $x \in S$ or particularly far from every $x \in S$, we can effectively determine which of these two cases applies by comparing $B(w)$ and t.

As an example, we consider $t = k$, which corresponds to the technique used by a standard Bloom filter. More specifically, we suppose that when the filter is queried with parameters $0 \leq \varepsilon < \delta$ and $u \in U$, it responds that "$d(u,S) \leq \varepsilon$" if all of u's hash locations are 1 and "$d(u,S) \geq \delta$" otherwise. For this scheme, Corollary 2.1 immediately tells us that

- if $d(u,S) \leq \varepsilon$, then Pr(the filter is incorrect) $\leq 1 - p_L(\varepsilon)^k$, and
- if $d(u,S) \geq \delta$, then Pr(the filter is incorrect) $\leq \left(n\left[p_H(d(\delta)) + \frac{k}{m} \cdot \mathbf{1}\left(V \neq [m']\right)\right]\right)^k$.

The setting of $t = k$ is simply an example. For any real application, it is important to assess the relative severity of false positives and false negatives and experimentally determine the value of t that optimizes the tradeoff between the observed frequencies of the two types of errors.

Before continuing, we note that the bound for the false positive probability is unfortunately rather weak in this general setting, due to the fact that the probability that a particular hash location is set to 1 (from the occurrence of the event $\{\exists x \in S : h_i(x) = h_i(u)\}$ for some fixed $i \in [k]$ and $u \in U$ with $d(u,S) \geq \delta$) is bounded only by $np_H(\delta)$; this requires that $p_H(\delta)$ is certainly $O(1/n)$ and preferably $o(1/n)$. In practice, this weakness may be avoided, although this depends on the set S (and the underlying metric); in particular, $\max_u \sum_{x \in S} p_H(d(x,u))$ will often be smaller than $np_H(\delta)$, and one may be able to bound $\max_u \sum_{x \in S} p_H(d(x,u))$ easily given S. Alternatively, the bound $np_H(\delta)$ may be reasonably tight, such as when $p_H(\delta)$ is small and collisions between elements of S are unlikely. Similar issues arise in nearest-neighbor problems [1]; this issue clearly warrants further study.

3 The Relative Hamming Metric on $U = \Sigma^\ell$

To give more insight and make the problem more concrete, we now focus on the special case where $U = \Sigma^\ell$ for some ℓ and alphabet Σ, and d is the relative Hamming metric on U (that is, $d(u,v) = \sum_{i=1}^{\ell} \mathbf{1}\left(u_i \neq v_i\right)/\ell$, where $\mathbf{1}(\cdot)$ is the indicator function and for $u \in U$ and $i \in [\ell]$, we let u_i denote the ith character of u). Without loss of generality, we suppose that $\Sigma = \{0, \ldots, r-1\}$ for some $r \geq 2$. An obvious alternative approach in this situation would be to simply choose s character locations (independently and uniformly at random) and for each string in S use these s characters as a sketch for the string. To check if any string in S has distance at most ε from some input x the sketches can be checked to see if any match in (slightly less than) a $1-\varepsilon$ fraction of locations. Such schemes require checking at least $\Omega(n)$ characters to find a potential match; while this may be suitable for some applications, we aim (in the spirit of standard Bloom filter constructions) to use only a constant number of character lookups into the data structure, making this approach unsuitable (although it may also be useful in practice).

We first give a general analysis of a distance-sensitive Bloom filter and show that this analysis does not yield performance tradeoffs nearly as good as a standard Bloom filter. Then we show that by limiting our goals appropriately, our analysis yields results that are suitable in practice for the important case where $r = 2$. Finally, we present the results of simple experiments that demonstrate the potential practicability of the scheme.

Recall that, given parameters $0 \leq \varepsilon < \delta \leq 1$, our goal is to effectively distinguish between the strings $u \in U$ where $d(u,S) \leq \varepsilon$, which we call *ε-close to S*, and those where $d(u,S) \geq \delta$, which we call *δ-far from S*.

We define

$$c = \begin{cases} 1 & \text{if } r = 2 \\ 1/2 & \text{if } r > 2 \end{cases}$$

$$n = |S|$$

$$\ell' = \left\lceil \log_{\frac{1-c\varepsilon}{1-c\delta}} 4n \right\rceil$$

$$m' = 2^{\ell'}$$

$$m = km'$$

$$t = k(1-c\varepsilon)^{\ell'}/2.$$

Here n is the number of items, m is the total size of the filter (in bits), $k \geq 1$ is the number of hash functions, and ℓ' is the number of locations read per hash function. If $r = 2$, the bits yield a location in the hash table in the natural way. Specifically, we define the hash family $\mathcal{H} : U \to [m']$ in the case where $r = 2$ as

Copyright © by SIAM. Unauthorized reproduction of this article is prohibited

follows: we choose $h \leftarrow \mathcal{H}$ by choosing $i_1, \dots, i_{\ell'} \leftarrow [\ell]$ uniformly and independently, and then defining $h(u) = u_{i_1} \cdots u_{i_{\ell'}} + 1$ (where we are considering $u_{i_1} \cdots u_{i_{\ell'}}$ as a number in binary). If $r > 2$, we do essentially the same thing, but with an added level of pairwise independent hashing from Σ to $\{0,1\}$. More precisely, if $r > 2$, we let $\mathcal{H}' : \Sigma \to \{0,1\}$ be a pairwise independent hash family, and we define the hash family $\mathcal{H} : U \to [m']$ as follows: we choose $h \leftarrow \mathcal{H}$ by choosing $i_1, \dots, i_{\ell'} \leftarrow [\ell]$ uniformly and independently, $h'_1, \dots, h'_{\ell'} \leftarrow \mathcal{H}'$ independently, and then defining $h(u) = h'_1(u_{i_1}) \cdots h'_{\ell'}(u_{i_{\ell'}}) + 1$. Using these definitions, we construct the filter described in Section 2.

It is easy to verify that $\mathcal{H}$ is a (p_L, p_H)-distance sensitive hash function for $p_L(z) = p_H(z) = (1 - cz)^{\ell'}$. (Indeed, if $r = 2$, then $\mathcal{H}$ is a canonical example of a locality-sensitive hash family [5].) Proposition 2.1 now immediately yields the following result.

Corollary 3.1. *Consider some $u \in U$.*

- *If $d(u, S) \le \varepsilon$, then $q_u \ge (1 - c\varepsilon)^{\ell'}$.*
- *If $d(u, S) \ge \delta$, then $q_u \le n(1 - c\delta)^{\ell'}$.*

In Section 2, we gave some intuition as to why results of the above form should allow for a properly configured filter to effectively distinguish between those $u \in U$ that are ε-close to S and those $u \in U$ that are δ-far from S. Theorem 3.1 formalizes that intuition.

Theorem 3.1. *When $t = k(1 - c\varepsilon)^{\ell'}/2$, then for any fixed $u \in U$, over the random choices made constructing the filter:*

- *If $d(u, S) \le \varepsilon$, then*

$$\begin{aligned}\Pr(B(u) < t) &< \exp\left[-2t^2/k\right] \\ &= \exp\left[-k(1 - c\varepsilon)^{2\ell'}/2\right].\end{aligned}$$

- *If $d(u, S) \ge \delta$, then*

$$\begin{aligned}\Pr(B(u) \ge t) &\le \exp\left[-t^2/2k\right] \\ &= \exp\left[-k(1 - c\varepsilon)^{2\ell'}/8\right].\end{aligned}$$

Remark 3.1. While the correct choice of the threshold t in practice may depend on the relative importance of false positive and false negatives, the choice of $t = k(1 - c\varepsilon)^{\ell'}/2$ presented here allows provable statements that give insight into the performance tradeoffs involved in designing the filter; specifically, the bounds are the same order of magnitude in the exponent, and allow for the asymptotic analyses in Sections 3.1 and 3.2.

Proof. For the first inequality, we have $d(u, S) \le \varepsilon$. Then

$$t - \mathbf{E}[B(u)] = t - kq_u \le t - k(1 - c\varepsilon)^{\ell'} = -t$$

by Corollary 3.1. Therefore,

$$\begin{aligned}\Pr(B(u) < t) &= \Pr(B(u) - \mathbf{E}[B(u)] < t - \mathbf{E}[B(u)]) \\ &\le \Pr(B(u) - \mathbf{E}[B(u)] < -t) \\ &< \exp\left[-2t^2/k\right],\end{aligned}$$

by the Azuma-Hoeffding inequality.

For the second inequality, if $d(u, S) \ge \delta$, then

$$\begin{aligned}t - \mathbf{E}[B(u)] &= t - kq_u \\ &\ge t - kn(1 - c\delta)^{\ell'} \\ &= k(1 - c\varepsilon)^{\ell'}\left[\frac{1}{2} - n\left(\frac{1 - c\delta}{1 - c\varepsilon}\right)^{\ell'}\right] \\ &\ge k(1 - c\varepsilon)^{\ell'}\left[\frac{1}{2} - \frac{1}{4}\right] \\ &= t/2,\end{aligned}$$

where the second step follows from Corollary 3.1, the fifth step follows from the fact that $\ell' \ge \log_{\frac{1-c\varepsilon}{1-c\delta}} 4n$, and the other steps are obvious. Therefore,

$$\begin{aligned}\Pr(B(u) \ge t) &= \Pr(B(u) - \mathbf{E}[B(u)] \ge t - \mathbf{E}[B(u)]) \\ &\le \Pr(B(u) - \mathbf{E}[B(u)] \ge t/2) \\ &\le \exp\left[-2(t/2)^2/k\right] = \exp\left[-t^2/2k\right],\end{aligned}$$

by the Azuma-Hoeffding inequality. □

3.1 Asymptotic Analysis: The Bad News.

We now consider what Theorem 3.1 suggests about the performance of our basic distance-sensitive Bloom filter construction. While the experiments in Section 4 show that our construction is likely to be useful in many practical situations, it does not scale well to very large numbers of items. By comparison, we recall that standard Bloom filters have the following three very desirable properties when properly configured:

1. They use $O(n)$ bits (and the hidden constant is small).
2. They guarantee a small constant error probability (asymptotically and in practice).
3. To answer a query, one must only read a small constant number of bits in the array.

Our construction does not appear to meet these goals for constant (with respect to n) values of the

Copyright © by SIAM. Unauthorized reproduction of this article is prohibited

parameters ε and δ. In fact, it does not even seem to meet the last two goals for constant ε and δ. For example, for $r = 2$, if we take $k = O(1)$ and $\varepsilon = \Omega(1)$, then the bounds in Theorem 3.1 yield constant error probabilities only if $\ell' = O(1)$, in which case

$$\log_{\frac{1-\varepsilon}{1-\delta}} 4n = O(1) \quad \Longrightarrow \quad \delta = 1 - \frac{1}{n^{\Omega(1)}}.$$

Similarly, if $k = O(1)$ and $\delta = 1-\Omega(1)$, then the bounds in Theorem 3.1 give constant error probabilities only if $(1-\varepsilon)^{\ell'} = \Omega(1)$, implying that

$$\left\lceil \log_{\frac{1-\varepsilon}{1-\delta}} 4n \right\rceil = \ell' = O\left(\frac{1}{\log \frac{1}{1-\varepsilon}}\right),$$

which cannot hold for constant ε. Therefore, the only way that Theorem 3.1 allows us to achieve the desired goals is if we restrict our attention to cases where the gap between ε and δ is extremely large for sufficiently large n.

3.2 Asymptotic Analysis: The Good News. If we loosen the desired properties for a distance-sensitive Bloom filter, we can still obtain results that appear useful in practice. Specifically, in the case of the relative Hamming metric, we note that the total length of the strings in the original set is at least $n\ell \log_2 r$ bits. Therefore, we should seek that the total length of the filter m is much less than $n\ell$, and not be concerned if $m = \omega(n)$ so long as this condition holds. Furthermore, and more importantly, we consider cases where δ is constant but ε is only $O(1/\log n)$. That is, we only seek to differentiate between query strings that differ from all strings in S on a (constant) δ-fraction of bits and query strings that share a $1 - \varepsilon = 1 - O(1/\log n)$-fraction of bits with some string in S. This restriction clearly limits the scalability of the construction. However, it still allows very useful results for n in the thousands or tens of thousands. (Larger n can be handled, but only with quite small values of ε or quite large values of ℓ.) In fact, as our experiments in Section 4 show, the asymptotic restriction that $\varepsilon = O(1/\log n)$ still allows for excellent performance with reasonable values of n, ℓ, and δ. And if a particular application allows for an even smaller value of ε, say $\varepsilon = c'/n$ for some constant c', then the performance for reasonable values of n and ℓ only improves, although the gap between ε and a reasonable value of δ may be quite large.

For convenience we focus on the binary case where $r = 2$, so $U = \{0,1\}^{\ell}$. In this case, $k\ell'$, the total number of sampled characters, is still logarithmic in n. The space used in the filter is

$$m = k2^{\ell'} = O\left(n^{1/\log_2 \frac{1-\varepsilon}{1-\delta}}\right).$$

For $\delta < 1/2$ (which is the normal case, as even random bit strings will agree on roughly 1/2 of the entries) and $\varepsilon = O(1/\log n)$, we have that $m = \omega(n)$. However, in many cases we can still configure the filter with reasonable parameters so that $m \ll n\ell$. To gain some insight (that will guide our experiments) it is worth considering some sample cases. For $n = 1000$, $\varepsilon = 0.1$, and $\delta = 0.4$, we find $\ell' = 21$. Hence for $k \leq 32$ the number of bits required is less than 2^{26}, giving an improvement over the total number of bits $n\ell$ whenever $\ell \geq 2^{16}$ bits or 8 Kilobytes. Similarly, for $n = 10000$, $\varepsilon = 0.05$, and $\delta = 0.4$, we find $\ell' = 24$, and again for $k \leq 32$ there is a space savings whenever $\ell \geq 2^{16}$.

Formally, we have the following asymptotic relationship, which shows that ℓ need only grow polynomially with n to have $m = o(n\ell)$. (Little effort has been made to optimize the constants below.)

Proposition 3.1. *For any constant δ and $r \geq 2$, if $\log_n \ell = (4 - 6c\delta)/c\delta + \Omega(1/\log n)$, then we may choose $\varepsilon = \Omega(1/\log n)$ and have $m = o(n\ell)$.*

Proof. Let $\gamma = (\log_n \ell)/2$. Since $\ell \geq \log_r n$, we have that

$$\ell = \omega(1) = 2^{\omega(1)} = 2^{(\log_2 n)\omega(1/\log n)} = n^{\omega(1/\log n)},$$

so $\gamma = \omega(1/\log n)$. Therefore, we have that $m = o(n\ell)$ if

$$\frac{1}{\log_2 \frac{1-c\varepsilon}{1-c\delta}} + \gamma \leq \log_n n\ell$$

for sufficiently large n, since in that case

$$m/n\ell = O\left(n^{-\gamma}\right) = O\left(2^{-\gamma \log_2 n}\right) = O\left(2^{-\omega(1)}\right) = o(1).$$

Solving the above inequality for $c\varepsilon$ gives

$$c\varepsilon \leq 1 - (1 - c\delta)2^{1/\log_n n^{1-\gamma}\ell}.$$

Therefore, we may choose $\varepsilon = \Omega(1/\log n)$ if

$$1 - (1 - c\delta)2^{1/\log_n n^{1-\gamma}\ell} \geq \Theta(1/\log n).$$

Equivalently, we may choose $\varepsilon = \Omega(1/\log n)$ if

$$2^{1/\log_n n^{1-\gamma}\ell} \leq \frac{1 - \Theta(1/\log n)}{1 - c\delta}$$

Since $\ell = n^{2\gamma}$, we have that $\log_n n^{1-\gamma}\ell = 1 + \gamma$, and

$$\begin{aligned} 2^{1/\log_n n^{1-\gamma}\ell} &= \exp\left[\frac{\ln 2}{\log_n n^{1-\gamma}\ell}\right] \\ &= \exp\left[\frac{\ln 2}{1+\gamma}\right] \\ &\leq \exp\left[\frac{1}{1+\gamma}\right] \\ &\leq 1 + \frac{2}{1+\gamma}, \end{aligned}$$

Copyright © by SIAM. Unauthorized reproduction of this article is prohibited

where we have used the facts that $\gamma > 0$ and $e^x \le 1+2x$ for $x \in [0,1]$. Therefore, we may choose $\varepsilon = \Omega(1/\log n)$ if

$$1 + \frac{2}{1+\gamma} \le \frac{1-\Theta(1/\log n)}{1-c\delta},$$

or, equivalently,

$$\begin{aligned} 1+\gamma &\ge 2(1-c\delta)\left(\frac{1}{c\delta - \Theta(1/\log n)}\right) \\ &= \frac{2(1-c\delta)}{c\delta}\left(\frac{1}{1-\Theta(1/\log n)}\right) \\ &= \frac{2(1-c\delta)}{c\delta}\left(1+\Theta(1/\log n)\right) \\ &= \frac{2(1-c\delta)}{c\delta} + \Theta(1/\log n), \end{aligned}$$

where we have used the Taylor series for $1/(1+x)$ (for $|x| < 1$). Therefore, we may choose $\varepsilon = \Omega(1/\log n)$ if

$$\gamma \ge \frac{2-3c\delta}{c\delta} + \Theta(1/\log n),$$

which holds by our assumption on $\log_n \ell = 2\gamma$. $\square$

Proposition 3.1 tells us that distance-sensitive Bloom filters can be very space efficient. Unfortunately, for certain reasonable settings of n, ε, and δ, the length ℓ of the strings may need to be very large in order for the filter to require less space than the natural encoding of the set, particularly when $r > 2$ (so $c = 1/2$). (And in these cases, one might be willing to sacrifice very fast query times to reduce the storage requirement using, for instance, the sketching approach mentioned in Section 3.) For example, consider the case where the characters in the alphabet Σ are bytes, so $r = 256$. Then if, as before, $n = 1000$, $\varepsilon = 0.1$, and $\delta = 0.4$, we now have $m' = 2^{49}$. Therefore, even if $k = 1$ (which is almost certainly too small to achieve good performance), $m \le n\ell \log_2 r$ only if $\ell > 7 \times 10^{10}$. We are unaware of any current applications where ℓ can be this large, although there may be future applications of distance-sensitive Bloom filters to extremely long strings, such as DNA analysis. Thus, while distance-sensitive Bloom filters work reasonably well for binary strings, there is much room for improvement in dealing with larger alphabets.

4 A Simple Experiment

We present experimental results demonstrating the behavior of the basic distance-sensitive Bloom filter of Section 3 in the special case where $r = 2$. In our experiments, we populate our data set S with uniform random binary strings of 65,536 characters, and test whether we can distinguish strings that are near an element of S from strings far from every element of S (under the Hamming metric). In part, this test is chosen because it represents a worthwhile test case; surely we would hope that any worthwhile scheme would perform reasonably on random inputs. In application terms, for the setting described in Section 1 of a distance-sensitive Bloom filter taken over a collection of other Bloom filters, this random experiment roughly corresponds to sets of the same size with no overlapping elements; similar results would hold even if small overlaps between sets were allowed.

We consider sets of sizes $n = 1000$ and $n = 10000$ with $\ell = 65536$. For $n = 1000$, we take $\varepsilon = 0.1$ and $\delta = 0.4$, and for $n = 10000$, we take $\varepsilon = 0.05$ and $\delta = 0.4$. For each value of n and $k = \{1, \ldots, 25\}$, we repeat the following experiment 10 times. Generate 50000 independent *close* queries by randomly selecting a string in S and flipping an ε-fraction of its bits, and testing whether the filter returns a false negative. Then generate 50000 independent *far* queries by randomly selecting a string in S and flipping a δ-fraction of its bits, and testing whether the filter returns a false positive (here, the implicit assumption is that the query string is δ-far from *all* strings in S, not just the one chosen; this holds with very high probability). We compute observed false positive and false negative rates by averaging our results over all queries.

The results are given in Figures 1 and Figure 2, with a summary in Table 1. In the figures, the x-axis is given in terms of the number m of bits used, but recall that $m = km'$, so it also reflects the number k of hash functions. The abrupt jumps in the false positive rate correspond to values of k where the threshold value $\lfloor t \rfloor$ increases. At such a point, it naturally becomes much harder for a false positive to occur, and easier for a false negative to occur.

As the figures and table indicate, we successfully use fewer than $n\ell$ bits and require only a constant number of bit lookups into the data structure (the array A). The observed false positive and negative rates are quite reasonable; in particular, the false negative rate falls rapidly, which is good as false negatives are often very damaging for applications.

We emphasize that although the experimental results appear to give modest size improvements, this is because we have chosen a small value for the item size of ℓ bits. For larger values of ℓ, the results would be entirely the same, except that the ratio $m/n\ell$ would shrink further.

Before concluding, we point out a few interesting characteristics of these experiments. First, recall that these tests roughly correspond to the application described in Section 1 where a distance-sensitive Bloom

Copyright © by SIAM. Unauthorized reproduction of this article is prohibited

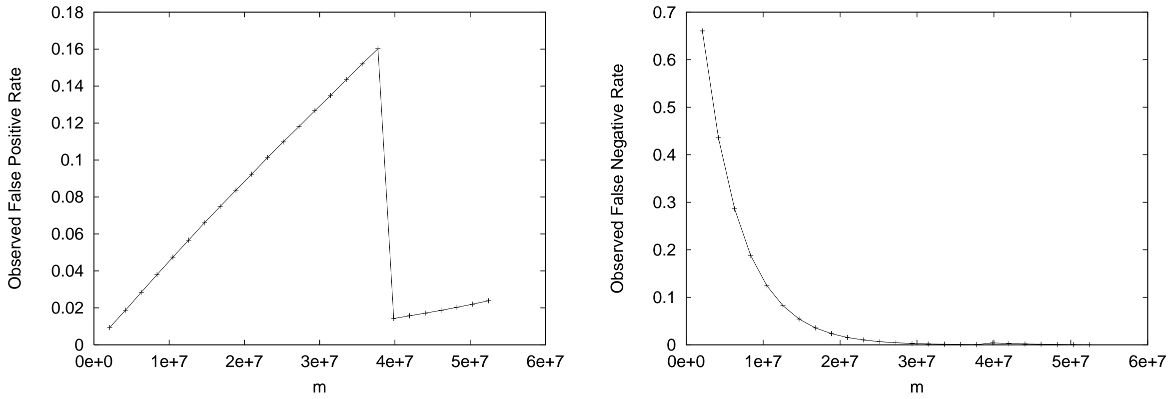

Figure 1: The observed false positive and false negative rates for $n = 1000$, $\ell = 65536$, $\varepsilon = 0.1$, and $\delta = 0.4$.

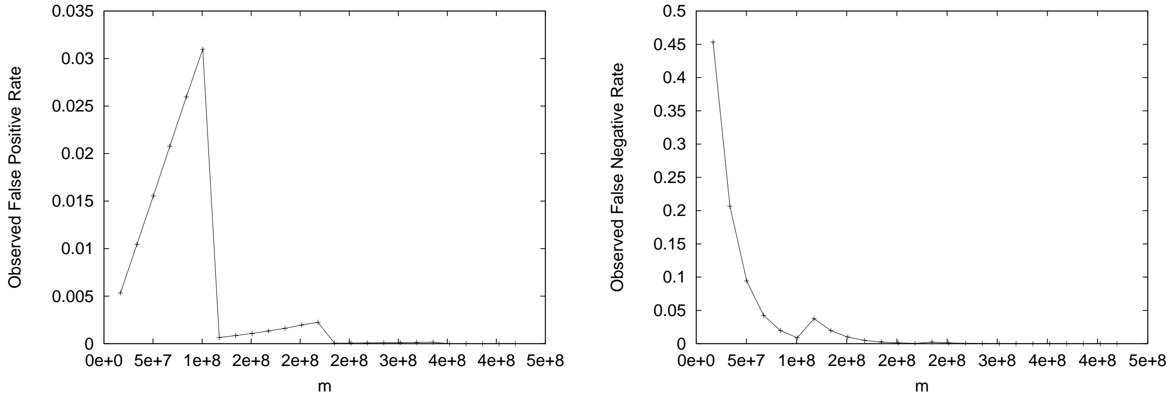

Figure 2: The observed false positive and false negative rates for $n = 10000$, $\ell = 65536$, $\varepsilon = 0.05$, and $\delta = 0.4$.

(a) $n = 1000$, $\ell = 65536$, $\varepsilon = 0.1$, and $\delta = 0.4$.

k	$k\ell'$	fp rate	fn rate	$m/n\ell$
5	105	0.04744	0.124236	0.16
10	210	0.09235	0.015366	0.32
15	315	0.134926	0.001934	0.48
20	420	0.01572	0.002816	0.64
25	525	0.023874	0.000372	0.8

(b) $n = 10000$, $\ell = 65536$, $\varepsilon = 0.05$, and $\delta = 0.4$.

k	$k\ell'$	fp rate	fn rate	$m/n\ell$
5	120	0.025958	0.019746	0.128
10	240	0.001338	0.00495	0.256
15	360	0.000068	0.00125	0.384
20	480	0.000158	0.000034	0.512
25	600	0.000006	0.000012	0.64

Table 1: The number $k\ell'$ of sampled bits, the observed false positive and false negative rates, and the corresponding storage requirements, for various values of k.

Copyright © by SIAM. Unauthorized reproduction of this article is prohibited

filter is formed for a collection of Bloom filters with no shared elements. In certain instances of this application, we might be justified in choosing a very small ε. For example, if the Bloom filters each represent sets of $n' = c_1 n$ elements and have size $c_2 n'$, and $\varepsilon = c_3/n$, where c_1, c_2, and c_3 are reasonable constants, then a query Bloom filter is ε-close to the set of Bloom filters if (roughly speaking) the set underlying the query filter shares all but a constant number of items with one of the other sets. While this threshold is certainly low, it may be suitable for some applications, and for such applications a properly configured distance-sensitive Bloom filter is likely to be extremely successful.

On a more general note, these experiments have the nice property that the set S is uniformly sampled from $\{0,1\}^{\ell}$. Thus, it is unlikely that any close query is ε-close to any element of S other than the one used to generate the query. Furthermore, for a far query, the events corresponding to hash collisions between the query and the elements of S are independent and, since ℓ' is reasonably sized, each occur with small probability. By looking back at the proof of Proposition 2.1, it follows that the bounds in Corollary 3.1 are fairly tight. Therefore, while those results may be very weak for certain data sets, it is impossible to substantially improve them without taking into account specific properties of S.

5 Conclusions and Further Work

We strongly believe that efficient distance-sensitive Bloom filters will find significant use in many applications. Our initial work suggests that distance-sensitive Bloom filters can be constructed with parameters suitable in practice, but further improvements appear possible.

We suggest a number of open questions for further work:

- Is there a general approach that would allow constant $\varepsilon, \delta > 0$, a linear number of bits, a constant number of hash functions, and constant false positive/false negative probabilities?

- Are there simple and natural conditions one can place on a set S for natural metrics that would yield stronger bounds?

- There are very natural information-theoretic bounds on the number of bits required for Bloom-filter-like structures. Are there equivalent bounds in this setting? (Perhaps one must fix a metric, such as the relative Hamming metric.)

- Can closeness in edit distance be handled using data structures of this type?

References

[1] A. Andoni, M. Datar, N. Immorlica, P. Indyk, and V. Mirrokni. Locality-sensitive hashing using stable distributions. To appear in *Nearest Neighbor Methods in Learning and Vision: Theory and Practice*, MIT Press.

[2] A. Broder and M. Mitzenmacher. Network Applications of Bloom Filters: A Survey. *Internet Mathematics*, 1(4):485-509, 2004.

[3] N. Jain, M. Dahlin, and R. Tewari. Using Bloom filters to refine web search results. In *Proc. of the Eighth International Workshop on the Web and Databases (WebDB2005)*, 2005.

[4] M. Datar, N. Immorlica, P. Indyk, and V. Mirrokni. Locality-sensitive hashing scheme based on p-stable distributions. In *Proc. of the 20th ACM Symposium on Computational Geometry*, pp. 253-262, 2004.

[5] A. Gionis, P. Indyk, and R. Motwani. Similarity Search in High Dimensions via Hashing. In *Proc. of the 25th International Conference on Very Large Data Bases*, pp. 518-529, 1999.

[6] O. Goldreich, S. Goldwasser, and D. Ron. Property testing and its connection to learning and approximation. *JACM*, 45(4):653-750, 1998.

[7] P. Indyk. Approximate nearest neighbor under edit distance via product metrics. In *Proc. of the 15th Annual ACM-SIAM Symposium on Discrete Algorithms*, pp. 646-650, 2004.

[8] U. Manber and S. Wu. An algorithm for approximate membership checking with applications to password security. *Information Processing Letters*, 50(4):191-197, 1994.

[9] M. Mitzenmacher. Compressed Bloom Filters. *IEEE/ACM Transactions on Networking*, 10(5):613-620, 2002.

[10] M. Mitzenmacher and E. Upfal. *Probability and Computing: Randomized Algorithms and Probabilistic Analysis*. Cambridge University Press, 2005.

[11] R. Ostrovsky and Y. Rabani. Low distortion embeddings for edit distance. In *Proc. of the 37th Annual ACM Symposium on Theory of Computing*, 218-224, 2005.

[12] A. Pagh, R. Pagh, and S. Srinivas Rao. An Optimal Bloom Filter Replacement. In *Proc. of the 16th Annual ACM-SIAM Symposium on Discrete Algorithms*, pp. 823-829, 2005.

[13] M. Parnas, D. Ron, and R. Rubinfeld. Tolerant property testing and distance approximation. ECCC Report No. 10, 2004.

[14] R. Rubinfeld and M. Sudan. Robust characterization of polynomials with applications to program testing. *SIAM Journal on Computing*, 25(2):252-271, 1996.

[15] K. Shanmugasundaram, H. Brunnimann, and N. Memon. Payload attribution via hierarchical Bloom filters. In *Proc. of the 11th ACM Conference on Computer and Communications Security*, pp. 31-41, 2004.

Copyright © by SIAM. Unauthorized reproduction of this article is prohibited

[16] A. Snoeren, C. Partridge, L. Sanchez, C. Jones, F. Tchakountio, B. Schwartz, S. Kent, and W. Strayer. Single-Packet IP Traceback. *IEEE/ACM Transactions on Networking*, 10(6):721-734, 2002.

Copyright © by SIAM. Unauthorized reproduction of this article is prohibited