CORA Cork Open Research Archive
Cartlann Taighde Oscailte Chorcaí

Title	Bloom filter variants for multiple sets: a comparative assessment
Authors	Calderoni, Luca;Maio, Dario;Palmieri, Paolo
Publication date	2022-02-28
Original Citation	Calderoni, L., Maio, D. and Palmieri, P. (2022) 'Bloom filter variants for multiple sets: a comparative assessment', Journal of Universal Computer Science, 28(2), pp. 120-140. doi: 10.3897/jucs.74230
Type of publication	Article (peer-reviewed)
Link to publisher's version	10.3897/jucs.74230
Rights	© 2022, the Authors. This is an open access article distributed under the Creative Commons Attribution-NoDerivatives 4.0 International (CC BY-ND 4.0) license. - https://creativecommons.org/licenses/by-nd/4.0/
Download date	2026-11-07 05:45:56
Item downloaded from	https://hdl.handle.net/10468/12646



UCC
University College Cork, Ireland
Coláiste na hOllscoile Corcaigh

Journal of Universal Computer Science, vol. 28, no. 2 (2022), 120-140
submitted: 30/4/2021, accepted: 28/12/2021, appeared: 28/2/2022 *CC BY-ND 4.0*

Bloom filter variants for multiple sets: a comparative assessment

Luca Calderoni
(Dept. of Computer Science and Engineering, University of Bologna, Italy
https://orcid.org/0000-0001-8294-7713, luca.calderoni@unibo.it)

Dario Maio
(Dept. of Computer Science and Engineering, University of Bologna, Italy
https://orcid.org/0000-0002-0094-0022, dario.maio@unibo.it)

Paolo Palmieri
(School of Computer Science & IT, University College Cork, Ireland
https://orcid.org/0000-0002-9819-4880, p.palmieri@cs.ucc.ie)

Abstract: In this paper we compare two probabilistic data structures for association queries derived from the well-known Bloom filter: the shifting Bloom filter (ShBF), and the spatial Bloom filter (SBF). With respect to the original data structure, both variants add the ability to store multiple subsets in the same filter, using different strategies. We analyse the performance of the two data structures with respect to false positive probability, and the inter-set error probability (the probability for an element in the set of being recognised as belonging to the wrong subset). As part of our analysis, we extended the functionality of the shifting Bloom filter, optimising the filter for any non-trivial number of subsets. We propose a new generalised ShBF definition with applications outside of our specific domain, and present new probability formulas. Results of the comparison show that the ShBF provides better space efficiency, but at a significantly higher computational cost than the SBF.

Keywords: Probabilistic data structures, Spatial bloom filter, Shifting bloom filter, Association queries
Categories: E.1, H.3.0, E.2
DOI: 10.3897/jucs.74230

1 Introduction

The term *big data* informally refers to data sets that are so large and complex that cannot be processed efficiently using traditional data structures and algorithms. The recent introduction of inexpensive information-gathering devices has fuelled the growth of pervasive sensing, which led to an increase in the size and number of data sets to be computed. Where traditional, deterministic data structures are inadequate to process big data, probabilistic data structures have been widely adopted by computer scientists as a suitable alternative [Tarkoma et al. 2012]. While standard data structures are designed to store elements and answer deterministically to queries, probabilistic data structures introduce an error probability. This drawback, however, is balanced by higher space efficiency and lower computational burden, which are crucial in big data applications [Chang et al. 2008].

Most probabilistic data structures, and specifically those relying on Bloom filters, use hash functions to randomize and compactly represent a set of items. The possibility of collisions introduces the potential for errors in the stored information, however the error probability is generally known and can be maintained under an arbitrary threshold by tuning the data structure parameters. Compared to errorless approaches, probabilistic data structures use significantly less memory, and have constant query time. They also usually support union and intersection operations and can therefore be easily parallelised.

Bloom filters (BF) are arguably the most prominent data structure belonging to this category. They were first introduced by Bloom [Bloom 1970] in 1970, and were extensively used during the last decades. Bloom filters were designed to support membership queries, i.e., to determine whether a given element is a member of a given set or not. The Bloom filter always determines positively if an element is in the set, while elements outside the set are generally determined negatively, but are subject to a (bounded) false positive error probability [Grandi 2018]. Several modifications of Bloom filters have been proposed over the years [Luo et al. 2019], with the purpose of adding additional functionalities. In this paper we compare two of these variants: the shifting Bloom filter (ShBF), and the spatial Bloom filter (SBF). Both variants allow the filter to store multiple sets[1] rather than a single one, and enable membership queries over these sets (called *association queries*). ShBF and SBF use different strategies to achieve this: the ShBF uses additional hash functions to determine a shift in the positions within the filter for each set, while the SBF writes a different index value for each set.

The article is organized as follows: in the following we discuss the relevant literature, in Section 2 we introduce the theoretical preliminaries to keep the paper self-contained and we propose a novel version of the ShBF. In Section 3 we conduct the comparative assessment between SBFs and ShBF and we comment the tests.

1.1 Related Works

Bloom filters and the derived data structures have been used in a number of database systems applications over the years [Liu and Özsu 2018]. Due to their ability to efficiently know whether or not a key belongs to a set of keys, they are widely adopted for first level indexes [Chang et al. 2008] and for query caching [Fan et al. 2000]. They were extensively tested over random access memories, traditional hard drives and, more recently on solid state drives. Bloom filters were also applied to semi-structured data queries [Wang et al. 2004], range queries and so forth. Due to their hash-based nature, Bloom filters were also applied in a number of privacy and security problems, such as biometric template protection [Gomez-Barrero et al. 2016].

Concerning association queries, following the classification proposed by Luo et al. [Luo et al. 2019], a number of Bloom filter variants proposed in the literature allow for multiple sets to be stored in the same filter. However, most data structures implement this functionality using a naive strategy: that is, using multiple instances of the data structure, one for each set.

Many such structures were proposed in the context of network routing strategies, where Bloom filters are widely used to address packet filtering and forwarding, and

[1] We note here that the relevant literature often mixes the definition of multi-set (as in multiple sets) and multiset. In this paper, we use the term multiset following the mathematical definition of a set which permits the coexistence of multiple instances of any element. In order to remove the ambiguity with multi-set, we only use the full expression “multiple sets” for datasets which include elements of different sets.

the application scenario often requires the use of multiple sets [Geravand and Ahmadi 2013, Broder and Mitzenmacher 2003, Patgiri et al. 2019]. For instance, the *longest prefix match* solution proposed in [Dharmapurikar et al. 2006] is one of the first strategies using the naive approach of one separate data structure for each set. In general, multiple BF strategies are viable when there is some a-priori knowledge regarding the number of sets to be mapped (and potentially the approximate number of items per set). This condition is often required for the design of hardware routing boards, where Bloom filters are parallelised and may thus be queried simultaneously. Several proposed solutions follow this principle: the IPv6 lookup scheme discussed in [Song et al. 2009] or the packet forwarding strategy presented in [Chang et al. 2004] are widely cited examples. The difference Bloom filter (DBF) is similar in this regard as it requires additional data structures linked to the main Bloom filter [Yang et al. 2017]. However, as the number of sets increases this approach becomes intuitively inefficient, as it increases the overhead linearly to the number of sets. It is also unsuitable when the number of sets cannot be predicted, or changes over time. We therefore exclude such Bloom filter variants from the comparison we present in this paper, to focus instead on structures that allow for multiple sets by design.

A number of variants of the Bloom filter allowing multiple sets exist in the literature: shifting Bloom filters [Yang et al. 2016], combinatorial Bloom filters [Hao et al. 2009, Hao et al. 2012], spatial Bloom filters [Palmieri et al. 2014, Calderoni et al. 2015], the kBF [Xiong et al. 2014], coloring embedders [Yang et al. 2019], and the Bloomier filter [Chazelle et al. 2004, Charles and Chellapilla 2008]. However, only a few are designed specifically for storing multiple sets. The kBF is instead a Bloom filter designed for key-value storage, with applications on approximate state machines; while Bloomier filters are a data structure for static support lookup tables. While both data structures can implement multiple sets, we do not include them in the comparison for reasons explained below. In the case of the kBF, the focus of the data structure is to store a value associated to each element of the originating set (following the key-value paradigm, where the key is the element and the value is the information stored in the data structure). The value is stored in a 32-bits space, divided into 29 bits used for encoding and a 3-bit counter. While we can use the value for storing a set identifier (and therefore group the elements into subsets) this is not the core functionality of the data structure, and, as it implies writing a 32-bit value for each element, is relatively inefficient. Similarly, the core functionality of Bloomier filters is not association queries on multiple sets: instead, Bloomier filters are designed for computing arbitrary functions defined only on the originating set. It is technically possible to implement a Bloomier filter defined on a membership function returning different values for elements of different subsets (and indeed this is used as an example by the authors). However, Bloomier filters can be implemented on arbitrary functions, which is a more complex problem and therefore introduce a significant overhead. In particular, Bloomier filters allow dynamic updates to the function, and also use two separate tables within the data structure, increasing memory usage. For the reasons detailed above, we restrict the comparative assessment presented in this paper to the data structures that are designed specifically for membership queries over multiple sets, commonly referred to as association queries. Filters that are designed for the purpose of storing multiple sets normally adopt either of two different strategies: making the filter non-binary (to allow for the set information to be stored in a filter position) as in the case of the spatial Bloom filter; or using an increased number of hash functions, where different hashes are used for different sets. The latter strategy is used by the shifting Bloom filters, as well as by the combinatorial Bloom filters and coloring embedders. As our objective is to compare the two main strategies to do so

(detailed above), we select the spatial Bloom filter to represent the non-binary filter approach, and the shifting Bloom filter for the multiple hashes approach. The shifting Bloom filter is in fact more efficient than the older combinatorial Bloom filters; while the coloring embedder has been optimised for 2 sets and can be extended to support a higher number (up to 16 in the original paper experiments) by following the same strategy as the shifting Bloom filter. Therefore, the shifting Bloom filter is arguably the best candidate to represent the multiple hashes approach.

1.2 Contribution

In this paper we present a comparative assessment of two probabilistic data structures for association queries, the shifting Bloom filter and the spatial Bloom filter. In the case of the shifting Bloom filter, we also provide a new, generalised definition that removes the upper limit on the number of sets to be stored (contrary to the original definition that focuses on two sets only), and we provide adapted error probability formulas to reflect this change. The comparative assessment is based on an experimental analysis of the behaviour of the two data structures over large test datasets. Extensive tests have been performed using comparable implementations of both primitives. The empirical results validate the theoretical analysis of the error probabilities, space efficiency and computational costs.

2 Theory

In this section, we first introduce some useful notions and definitions, and later we re-define both data structures used in the comparison in a coherent and comparable way. In particular, we discuss the original Bloom filter (BF) data structure as proposed by Bloom (Section 2.1) in Section 2.1. The two Bloom filter variants on which the comparative assessment will be based on are introduced in two separate sections: the shifting Bloom Filter in Section 2.2, and the spatial Bloom filter in Section 2.3. While for the latter we follow the original definition as presented in [Calderoni et al. 2015], in the case of the shifting Bloom Filter we introduce a novel and significantly expanded definition, which improves the generality of the data structure. Extensive justification of this, including experimental result, is provided in the relevant section. Given the new definition, we also present a novel probabilistic model for the data structure, in Sections 2.2.1 and 2.2.2.

2.1 Bloom Filter

While several definitions of Bloom filters have been provided over time, both formal and functional, we choose to use in this paper a set-based formal definition, followed by a description of the construction function. We will follow this same approach for the ShBF and the SBF in the following.

Definition 1 *We define a Bloom filter $B(S)$ representing a set $S = \{a_1, \ldots, a_n\} \subseteq \{0,1\}^*$ as the set*

$$B(S) = \bigcup_{a \in S, h \in H} h(a) \ , \tag{1}$$

where $H = \{h_1, \ldots, h_k\}$ is a set of k hash functions such that each $h_i \in H : \{0,1\}^* \to \{1, \ldots, m\}$, that is, the hash functions take binary strings as input and output a number uniformly chosen in $\{1, \ldots, m\}$.

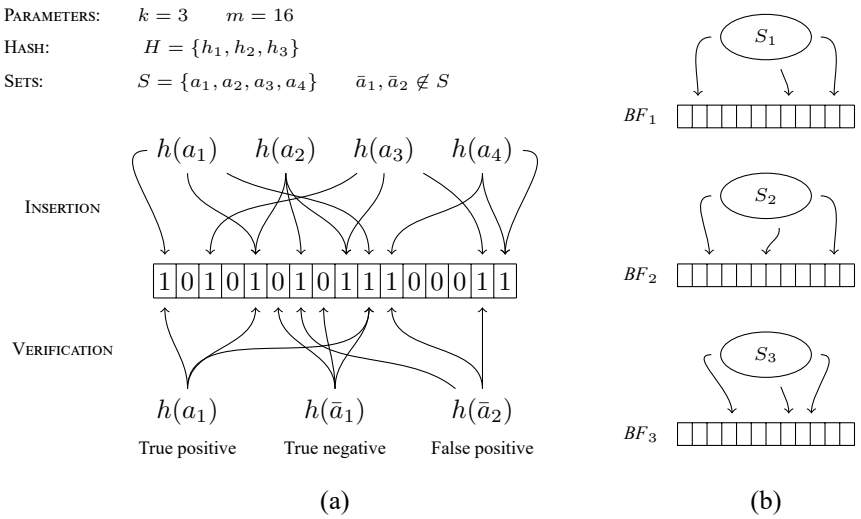


Figure 1: (a) Insertion of a set S and verification of elements in a Bloom filter of length $m = 16$, featuring three hash functions ($k = 3$). With respect to the verification process, it is possible to observe true positives, true negatives and false positives. Each scenario is depicted in sequence. (b) A naive approach to store elements belonging to multiple sets using the classic BF is to assign elements of each set to a separate, dedicated filter.

Please note that in the above definition we use a simplified notation: by $\bigcup x$, we mean the union with a set containing only element x. We omit the curly braces around x to avoid confusion with operators generating sets of sets.

A Bloom filter $B(S)$ can be represented as a binary vector b composed of m bits (or *cells*), where the i-th bit

$$b[i] = \begin{cases} 1 & \text{if } i \in B(S) \\ 0 & \text{if } i \notin B(S) \end{cases} . \tag{2}$$

The filter construction may be summarized as follows. First of all, each bit of the array is set to 0. Then we compute $h(a) = i$ for each item $a \in S$ and for each hash function $h \in H$. The corresponding i-th bit of the array b is thus set to 1. It follows that m bits are required in order to store b.

Subsequently, in order to check whether or not an element a_u belongs to the originating set S, the following condition should be satisfied:

$$\forall h \in H, b[h(a_u)] = 1 \ . \tag{3}$$

Therefore, if any bit in b that corresponds to a value output by one of the hash functions for a_u is 0, then $a_u \notin S$. If, instead, all the hash functions map to bits of value 1, then $a_u \in S$ minus a false positive probability p [Bose et al. 2008]. The nature of this probability is well known and it is out of the scope of this work to investigate it further. We will instead insist on those issues related to false positives and inter-set errors throughout our discussion on SBF and ShBF.

A schematisation of the BF insertion and verification procedures is presented in

Figure 1.

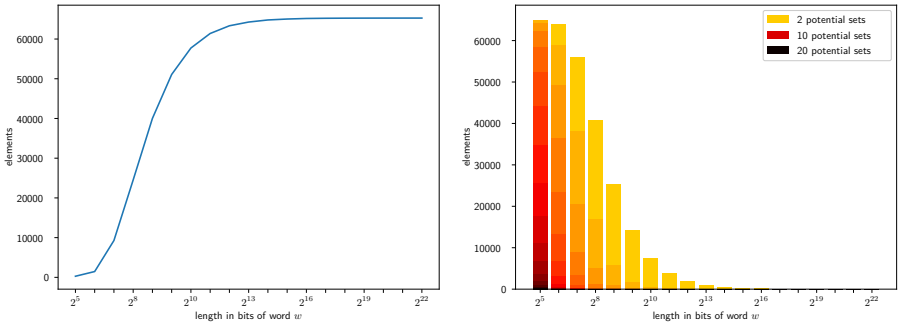
The Bloom filter characteristic of space-efficiency and of constant-time membership query lead to a rapid propagation of BF variants [Luo et al. 2019].

As briefly discussed above, one of the main constraints concerning BF consist in their ability to store elements belonging to a single originating set. A naive method to surpass this constraint is to build a separate filter for each set we need to represent. Although several among the proposed variants follow this principle [Luo et al. 2019], some BF variant able to store more than one set in a single filter were lately proposed. Spatial Bloom filters [Palmieri et al. 2014, Calderoni et al. 2015] and shifting Bloom filters [Yang et al. 2016] are both able to store elements belonging to several different sets, even then they differ one each other for a number of features. In the following, we discuss some basic principle concerning these data structures in order to better understand the comparison proposed in Section 3.

2.2 Shifting Bloom Filter

The shifting Bloom filter [Yang et al. 2016] is a variant of Bloom filters which supports membership queries as well as association queries and multiplicity queries. However, it cannot provide all the aforementioned features at once. The user needs to select which kind of queries the filter should answer to, and construct it according to procedures that are specific to each functionality. The interrogation procedures are also feature-specific. As the comparison proposed in this work focuses on association queries, we limit our discussion to ShBF filters set up for this specific type of queries. In the following we refer to the ShBF built for association queries simply as ShBF. The ShBF as proposed in the original work by Yang et al. [Yang et al. 2016] focuses only on two sets, and provides a mechanism for identifying elements that are present in both sets (therefore the two sets are not disjoint, and their intersection can be $\neq \emptyset$). However, the limitation of being able to store only two sets is a direct consequence of allowing the two originating sets to be non-disjoint, and requires the use of an auxiliary data structure thus increasing memory usage. We believe that this limitation significantly reduces the potential application scenarios for the ShBF, and therefore in this paper we extend the ShBF data structure to remove the upper limit on the number of originating sets. However, in doing so we remove the possibility to store sets with intersecting elements, and we allow for disjoint sets only. We feel the benefits of our proposed modification far outweight the downsides, and we note that intersections can potentially be re-introduced using a naive strategy (defining an intersection as a separate set). This modification also allows for a more meaningful comparison of the ShBF with the SBF data structure, with can also store an unbounded number of disjoint sets.

The shifting Bloom filter allows multiple sets to be stored by shifting the position at which the values are stored in the filter: that is, an *offset* is added to the output of each hash function, and the offset specifies which set the element belongs to. The offset is not static, but is determined by an additional hash function, so that each set uses a different hash function to determine the offset. In practice, this means that two elements of the same set will have different offsets, but these offsets are both calculated using the same hash function, which is set-specific. For memory-efficiency reasons, Yang et al. propose in the original paper to limit the offset space (that is, the output size of this additional hash function), so that the position pointed by the hash function plus the offset will be in the same memory word as the position pointed by the hash function only (without the offset). The authors do not specify a particular size for the memory word. While this approach can reduce memory reads when only two sets are stored, in the proposed



(a) Correctly recognised elements. *(b) Elements returning multiple potential sets.*

Figure 2: Experimental results showing the negative impact of a limited word size in a shifting Bloom filter.

generalisation that allows for a large number of sets to be stored, a small word size can significantly increase the number of elements for which the set cannot be determined with accuracy.

In Figure 2 we provide the results of an experiment where multiple word lengths are used over the same test dataset. For this experiment, we store 255 sets each containing 256 elements, for a total of 65280 elements in a filter of length 2^{23} bits. As the two figures clearly depict, a small word value (2^{10}) significantly increases the number of errors returned by the filter (a formal definition of such errors is provided in the following). The best results are obtained for word sizes of 2^{16} or above. Considering that the original goal of reducing the memory reads cannot be achieved with such a high word size, we propose here to discard the limitation on the offset space entirely. As shown by the experiment, limiting the offset space to any meaningful word size would negatively affect the correctness of the association queries result. Moreover, removal of the word requirement reduces the computational burden by eliminating a modulo operation for each offset calculation.

In the following, we provide a formal definition of the proposed generalised ShBF, as described above. In particular, the definition we propose differs from the original ShBF as proposed by Yang et al. [Yang et al. 2016] by removing the upper bound on the number of disjoint sets, and also removing the offset space limitation.

Definition 2 *Given the originating sets $\Delta_1, \Delta_2, \ldots, \Delta_s$ to be represented in the filter, let $\bar{S}$ be the union set $\bar{S} = \bigcup_{\Delta_i \in S} \Delta_i$ and S be the set of sets such that $S = \{\Delta_1, \ldots, \Delta_s\}$. Let $H = \{h_1, \ldots, h_k\}$ be a set of k hash functions such that each $h_i \in H : \{0,1\}^* \to \{1, \ldots, m\}$, that is, each hash function takes binary strings as input and outputs a number uniformly chosen in $\{1, \ldots, m\}$. Let also $H^\curlyvee = \{h_1^\curlyvee, \ldots, h_{s-1}^\curlyvee\}$ be a set of $s-1$ hash functions such that each $h_i^\curlyvee \in H^\curlyvee : \{0,1\}^* \to \{1, \ldots, m\}$, where m is a positive integer.*

We define a shifting Bloom filter $B^\curlyvee(S)$ over S as the set

$$B^\curlyvee(S) = \bigcup_{\delta \in \bar{S}, h \in H} (h(\delta) + o(\delta)) \bmod m \ , \tag{4}$$

where the offset $o(\delta)$ is intended as follows:

$$o(\delta) = \begin{cases} 0 & \text{if } \delta \in \Delta_1 \\ h_i^{\curlyvee}(\delta) & \text{if } \delta \in \Delta_{i+1}, i \neq 0 \end{cases} . \tag{5}$$

A shifting Bloom filter $B^{\curlyvee}(S)$ can be represented as a binary vector $b^{\curlyvee}$ composed of m bits (or *cells*), where the i-th bit

$$b^{\curlyvee}[i] = \begin{cases} 1 & \text{if } i \in B^{\curlyvee}(S) \\ 0 & \text{if } i \notin B^{\curlyvee}(S) \end{cases} . \tag{6}$$

In the following, when referring to a shifting Bloom filter, we refer to its vector representation $b^{\curlyvee}$.

The ShBF is built as follows. Initially all bits are set to 0. Then, for each $\Delta_i \in S$, for each element $\delta \in \Delta_i$ and for each $h \in H$ we calculate $(h(\delta) + o(\delta)) \bmod m = i$, and set the corresponding i-th bit of $b^{\curlyvee}$ to 1. Thus, m bits are needed in order to store $b^{\curlyvee}$.

The verification procedure is formalized as follows. Let us suppose to test an element $\delta \in \mathcal{E}$ against the filter (where $\mathcal{E}$ represents a generic domain), in order to learn which originating set it belongs to (or whether it does not belong to any originating set). First of all we need to compute k hash digests, i.e., $\forall h \in H$, we compute $h(\delta)$. Then, as each originating set is combined with a specific hash offset, we need to check separately whether the element belongs to each set. We check whether $\delta \in \Delta_i$ if

$$\forall h \in H, b^{\curlyvee}[(h(\delta) + o(\delta)) \bmod m] = 1 . \tag{7}$$

During the verification process of the set Δ_i, should we find an index i such that $b^{\curlyvee}[i] = 0$, we can conclude $\delta \notin \Delta_i$ and we proceed to another set $\Delta_j \in S$. Conversely, when the condition (7) is fulfilled, we add Δ_i to the set of positives matches Γ. At the end of the verification procedure three are the possible scenarios:

- $\Gamma = \emptyset$. Then we may assert $\delta \notin \bar{S}$.
- $|\Gamma| = 1, \Delta_i \in \Gamma$. Hence $\delta \in \Delta_i$.
- $|\Gamma| > 1$. Hence δ belongs to one among the sets included in Γ.

Both the second and the third case are subject to a false positive probability, as ShBF preserves the probabilistic nature of the classic Bloom filter. Concerning the latter case, please also note that it is not possible to assign a different weight to those sets included in Γ. Consequently, the element δ has the same probability to belong to each set, resulting in an inter-set error.

A schematization of the insertion and verification procedures of an ShBF is proposed in Figure 3.

2.2.1 False positives

As we redefined the ShBF data structure it is important to investigate its behaviour in terms of false positives.

Definition 3 *Given a filter $b^{\curlyvee}$, a false positive event occurs when the verification procedure performed on an element $\delta \notin \bar{S}$ terminates with a non-empty set of positives matches, i.e. $\Gamma \neq \emptyset$.*

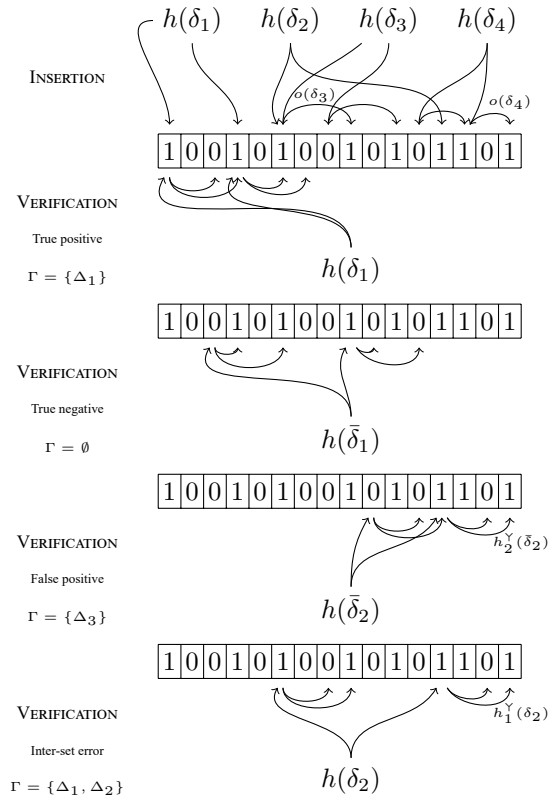


Figure 3: Insertion of three originating sets Δ_1, Δ_2 and Δ_3 and verification of elements in a shifting Bloom filter of length $m = 16$, featuring two hash functions ($k = 2$). With respect to the verification process, it is possible to observe true positives, true negatives, false positives and inter-set errors. Each scenario is depicted in sequence.

As we may see, a false positive event reported by a ShBF may pertain several originating sets, due to the verification procedure which performs a separate lookup for each originating set. Consequently, we define a false positive event on a specific set as follows:

Definition 4 *Given a filter $b^{\curlyvee}$, a false positive event on a specific originating set Δ_i occurs when the verification procedure performed on an element $\delta \notin \bar{S}$ terminates with the set Δ_i included in the set of positives matches, i.e. $\Delta_i \in \Gamma$.*

Since during the verification procedure each set is checked for membership through

a separate filter lookup performed on the same data structure, each originating set has the same probability of being included in the set of positives matches Γ. Specifically, the set-specific false positive probability ($\text{FPP}_i^{\curlyvee}$) in a ShBF coincides with the overall false positives probability of a common BF, assuming both filters were filled with the same amount of items, i.e. $b^{\curlyvee}$ was filled with $n = |\bar{S}|$ items:

$$\text{FPP}_i^{\curlyvee} = \left(1 - \left(1 - \frac{1}{m}\right)^{kn}\right)^k \quad . \tag{8}$$

Conversely, the overall false positives probability ($\text{FPP}^{\curlyvee}$) is increased due to the multiple filter lookups performed during the verification procedure. In order for the verification procedure to produce a false positive, it sufficient that one among the s set-specific lookups produces a set-specific false positive. As the probability for this event to occur is known ($\text{FPP}_i^{\curlyvee}$), we may address this problem using the well known success-failure scheme (Bernoulli trials). Given an event which occurs with probability p, the probability to observe b successes among a trials is:

$$\binom{a}{b} p^b (1-p)^{a-b} \quad . \tag{9}$$

The probability to report at least one set-specific false positive (i.e. at least one success) among s trials is the probability of the certain event minus the probability to never report a set-specific false positive among s trials. Hence

$$\text{FPP}^{\curlyvee} = 1 - \left(\binom{s}{0} \text{FPP}_i^{\curlyvee\,0} (1 - \text{FPP}_i^{\curlyvee})^{s-0}\right) \quad ; \tag{10}$$

it follows that:

$$\text{FPP}^{\curlyvee} = 1 - \left(1 - \left(1 - \left(1 - \frac{1}{m}\right)^{kn}\right)^k\right)^s \quad . \tag{11}$$

2.2.2 Inter-set errors

As discussed above, when the verification procedure produces a set of positives matches composed of more than one set, it is not possible to discern which among those sets is the one that effectively contains the element. Thus, supposing $|\Gamma| = u$, we have only $1/u$ chances to assign the element to the correct set.

Definition 5 *Given a filter $b^{\curlyvee}$, an inter-set error event occurs when the verification procedure performed on an element $\delta \in \bar{S}$ terminates with a set of positives matches such that $|\Gamma| > 1$.*

As per the classic BF, the ShBF is not affected by false negatives. Thus, when we test the filter for an element $\delta \in \Delta_i$, the correct set (Δ_i) is for sure included in Γ. However, the verification procedure is composed of s ShBF-lookups, one for each set. During the remaining $s - 1$ lookups, should the procedure report a false-positive, the corresponding set would be wrongly added to the set of positives matches. Hence, the probability to

observe an inter-set error ($\text{ISEP}^{\curlyvee}$) may be derived as the probability of the certain event minus the probability to never report a false positive throughout $s-1$ trials.

Following the same principle described in (10) we may derive:

$$\text{ISEP}^{\curlyvee} = 1 - \left(\binom{s-1}{0} \text{FPP}_i^{\curlyvee 0} (1 - \text{FPP}_i^{\curlyvee})^{s-1-0} \right) \quad ; \tag{12}$$

hence

$$\text{ISEP}^{\curlyvee} = 1 - \left(1 - \left(1 - \left(1 - \frac{1}{m} \right)^{kn} \right)^{k} \right)^{s-1} \quad . \tag{13}$$

As the amount of correct information produced by the verification procedure depends on the number of sets included in Γ, it is also useful to derive the probability $\text{ISEP}^{\curlyvee}_{u_i}$ for it to produce a set of positives matches with a specific cardinality i. Following the aforementioned principle, it is straightforward that the probability to obtain $|\Gamma| = i$ is the probability to report exactly $i-1$ false positives among $s-1$ trials. Hence:

$$\text{ISEP}^{\curlyvee}_{u_i} = \binom{s-1}{i-1} \text{FPP}_i^{\curlyvee i-1} (1 - \text{FPP}_i^{\curlyvee})^{s-i} \quad . \tag{14}$$

Finally, for ease of comparison with SBF, we discuss the probability for a specific set to be involved in a inter-set error event ($\text{ISEP}_i^{\curlyvee}$).

Definition 6 *Given a filter $b^{\curlyvee}$, an inter-set error event on a specific originating set Δ_i occurs when the verification procedure performed on an element $\delta \in \bar{S}$ terminates with a set of positives matches such that $\Delta_i \in \Gamma$, $|\Gamma| > 1$.*

Following the same principle discussed in Section 2.2.1 concerning false positive events on specific sets, it is straightforward to note that $\text{ISEP}_i^{\curlyvee}$ coincides with $\text{FPP}_i^{\curlyvee}$ (see (8) for reference).

2.3 Spatial Bloom Filter

The spatial Bloom filter [Palmieri et al. 2014, Calderoni et al. 2015] was originally introduced with the purpose of efficiently storing an arbitrary number of disjoint sets representing geographic areas. Although its first application was in the location privacy domain (as the name suggests), the data structure can store any type of element and thus represents an efficient solution for any kind of scenarios where association queries over an arbitrary number of sets are required [Palmieri et al. 2017, Calderoni et al. 2020].

Similarly to the generalised ShBF we propose earlier in this paper, SBF natively supports multiple sets and element mapping is performed through a single data structure, without any need of auxiliary data structures or meta data. The spatial Bloom filter (SBF) is defined as follows [Calderoni et al. 2015]:

Definition 7 *Given the originating sets $\Delta_1, \Delta_2, \ldots, \Delta_s$ to be represented in the filter, let $\bar{S}$ be the union set $\bar{S} = \bigcup_{\Delta_i \in S} \Delta_i$ and S be the set of sets such that $S = \{\Delta_1, \ldots, \Delta_s\}$. Let O be the strict total order over S for which $\Delta_i < \Delta_j$ for $i < j$. Let also $H = \{h_1, \ldots, h_k\}$ be a set of k hash functions such that each $h_i \in H : \{0,1\}^* \to \{1, \ldots, m\}$, that is, each hash function in H takes binary strings as input and outputs a random*

number uniformly chosen in $\{1, \ldots, m\}$. *We define the spatial Bloom filter over* (S, O) *as the set of couples*

$$B^{\#}(S,O) = \bigcup_{i \in I} \langle i, \max L_i \rangle \tag{15}$$

where I *is the set of all values output by hash functions in* H *for elements of* $\bar{S}$

$$I = \bigcup_{\delta \in \bar{S}, h \in H} h(\delta) \tag{16}$$

and L_i *is the set of labels* l *such that:*

$$L_i = \{l \mid \exists \delta \in \Delta_l, \exists h \in H : h(\delta) = i\} \quad . \tag{17}$$

A spatial Bloom filter $B^{\#}(S,O)$ can be represented as a vector $b^{\#}$ composed of m values (or *cells*), where the i-th value

$$b^{\#}[i] = \begin{cases} l & \text{if } \langle i, l \rangle \in B^{\#}(S,O) \\ 0 & \text{if } \langle i, l \rangle \notin B^{\#}(S,O) \end{cases} \quad \text{[Calderoni et al. 2015].} \tag{18}$$

Throughout this paper, we refer to spatial Bloom filters using the vector representation $b^{\#}$.

The construction procedure of a SBF may be summarized as follows [Calderoni et al. 2015]. The first step consists in setting each value in $b^{\#}$ to 0. Then we compute $h(\delta) = i$ for each item $\delta \in \Delta_1$ and for each $h \in H$. The corresponding cell of the filter ($b^{\#}[i]$) is set to 1 (as 1 represents the label of Δ_1). The same procedure is performed for elements included in Δ_2 (writing the value 2), then for those included in Δ_3 and so forth. The construction procedure terminates when the last set Δ_s has been processed. It is important to point out that, as the construction procedure follows the strict total order defined over S, sets with lower labels are more likely to be overwritten than sets with higher label values, in case of collision.

Concerning collisions, we note that they occur when a specific hash function produces the same digest for two different elements (this is indeed what we commonly refer to as a hash collision), but also when two separate hash functions produce the same digest for a single element or for two distinct elements. When one among the aforementioned conditions is met, the same filter cell is written twice. We refer to this phenomenon as *cell overwrite*, or simply as *collision*, indifferently. An in-depth analysis concerning collisions and how they affect the probabilistic properties of this data structure is beyond the scope of this work and may be found in [Calderoni et al. 2018].

During the filter construction, the filter crosses several *states* [Calderoni et al. 2018]:

Definition 8 *Let us consider the spatial Bloom filter* $b^{\#}$. *Given* $i \in L$, *we say the filter is in state* i *(and we refer to its vector representation as* $b^{\#}_{\dashv i}$*) if and only if all the elements of set* Δ_i *have been inserted into the filter.*

Therefore, state 0 ($b^{\#}_{\dashv 0}$) represents the empty filter (with all of its cells set to zero). At the end of the construction process, the filter is in state s ($b^{\#}_{\dashv s}$).

In order to know whether or not an element belongs to one of the originating sets, we need to perform a single filter lookup. Given $\delta \in \mathcal{E}$ (where $\mathcal{E}$ represents a generic domain), we compute k hash digest, i.e., $\forall h \in H$, we compute $h(\delta)$. We check whether $\delta \in \Delta_i$ if

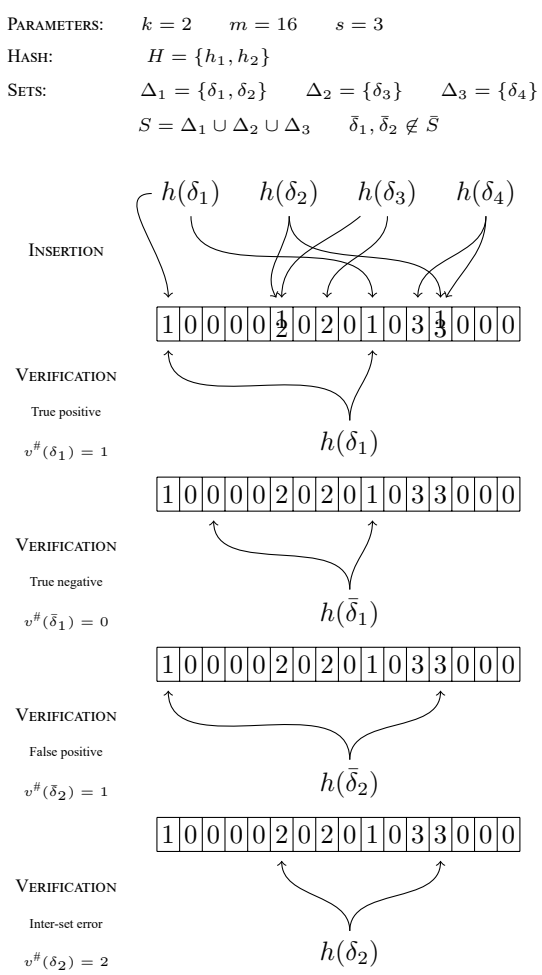


Figure 4: Insertion of three originating sets Δ_1, Δ_2 and Δ_3 and verification of elements in a spatial Bloom filter of length $m = 16$, featuring two hash functions ($k = 2$). With respect to the verification process, it is possible to observe true positives, true negatives, false positives and inter-set errors. Each scenario is depicted in sequence.

$$\exists h \in H : b^{\#}\left[h(\delta)\right] = i \quad \text{and} \quad \forall h \in H, b^{\#}\left[h(\delta)\right] \geq i \ . \tag{19}$$

Should one or more cells contain the value 0, we can conclude $\delta \notin \bar{S}$, i.e., it does not belong to any of the originating sets.

The verification procedure which classifies an element $\delta \in \mathcal{E}$ (where $\mathcal{E}$ represents a generic domain) as belonging to one of the originating sets may be formalized using a functional notation as follows:

$$\begin{aligned} v^{\#}: \quad & \mathcal{E} \rightarrow L_0 \\ & \delta \mapsto v^{\#}(\delta) \end{aligned} \tag{20}$$

where $L_0 = \{0, \ldots, s\}$. This function processes a generic element of the domain $\mathcal{E}$ and outputs a value included in $\{0, \ldots, s\}$. This integer indicates the set to which the element is supposed to belong to (if the output is > 0), or indicates that the element does not belong to any of the originating sets (when the output is 0). As thoroughly investigated in [Calderoni et al. 2018], false negatives are not possible, while positive matches are subject to false-positives and inter-set errors. An exemplification of insertion and verification procedures concerning a SBF is proposed in Figure 4.

2.3.1 False positives

While an in-depth investigation concerning SBF false positives falls outside the scope of this work, it is meaningful to recall some outcome for ease of comparison with the ShBF. Specifically, the overall false positive probability in a SBF coincides with the one of a common BF, assuming both filters were filled with the same amount of items, i.e. $|\bar{S}| = n$. In this case, the probability to observe a false positive when we query a SBF for membership of an element which does not belong to any of the originating sets is

$$\text{FPP}^{\#} = \left(1 - \left(1 - \frac{1}{m}\right)^{kn}\right)^{k} . \tag{21}$$

As widely discussed in [Calderoni et al. 2018], this probability may be divided in a set-specific one. Specifically, the probability to report a false positive on a given set Δ_i is:

$$\text{FPP}_i^{\#} = \left(1 - \left(1 - \frac{1}{m}\right)^{k \sum_{j=i}^{s} n_j}\right)^{k} - \sum_{j=i+1}^{s} \text{FPP}_j^{\#} . \tag{22}$$

2.3.2 Inter-set errors

As stated above, an in-depth investigation concerning SBF inter-set errors falls outside the scope of this work. However, we report the main outcomes discussed in [Calderoni et al. 2018] for ease of comparison with the ShBF. The probability to observe a set-specific inter-set error is:

$$\text{ISEP}_i^{\#} = \left(1 - \left(1 - \frac{1}{m}\right)^{kn_i^{\text{FILL}}}\right)^{k} , \tag{23}$$

where n_i^{FILL} represents the number of elements left for insertion after the filter $b^{\#}$ reaches the state i (see [Calderoni et al. 2018]).

Assuming $|\Delta_i| = n_i$, the overall inter-set error probability ($\text{ISEP}^{\#}$) may be derived as the weighted sum of each set-specific probability:

$$\text{ISEP}^{\#} = \frac{1}{n} \sum_{i=1}^{s} n_i \text{ISEP}_i^{\#} . \tag{24}$$

dataset	sets (s)	elements per set	elements (n)
uniform	255	256	65280
random	255	[209, 298]	65280
non-elements	-	-	500000

Table 1: Test datasets used for the experimental comparison of the two data structures [Calderoni and Palmieri 2017].

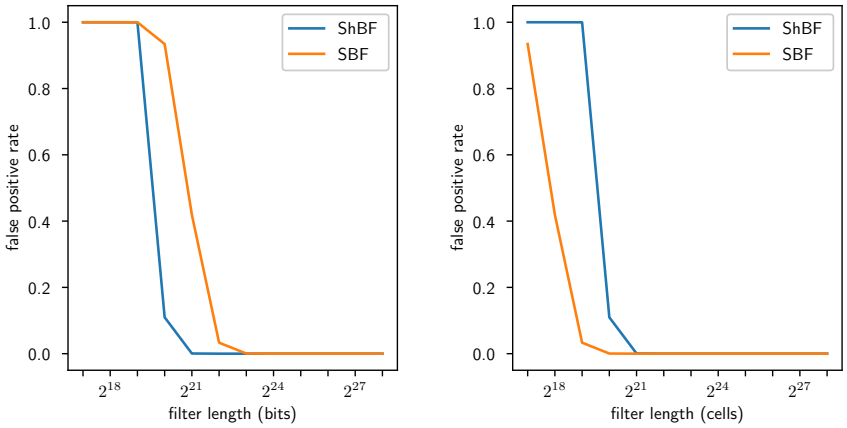
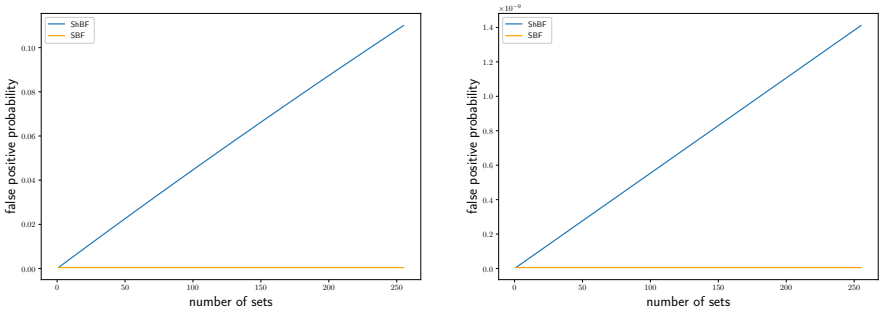


Figure 5: False positive rate for the two data structures over a uniformly distributed dataset, calculated over the same filter length in bits (left) and the same number of cells (right). The rate validates the probability functions discussed in Section 2.2.1 and 2.3.1.

3 Results and discussion

We compare the performances of the shifting Bloom filters and the spatial Bloom filters over the two most meaningful filter characteristics: the false positive probability and the inter-set error rate. Because of the different strategies used by the data structures to store multiple sets, these probabilities need to be evaluated with respect to two different parameters: the filter length in bits l (which expresses the memory usage of the corresponding filter), and the number of cells m (where each cell is a unique position within the filter). In the case of the ShBF, these two parameters coincide and $l = m$, as each cell can store either a 0 or a 1, following the binary nature of the original Bloom filter. The SBF allows instead multi-bit cells, where the data structure stores the index of the relevant set. The length of the filter in bits is therefore the number of cells times the size in bits of each cell, which is determined as the smallest power of 2 which is equal or greater than the number of sets to be stored plus one. So, if we need to store 255 sets, $l = 8 \cdot m$, as $2^8 = 256$.

We performed extensive experimental tests over implementations of both the shifting Bloom filter and the spatial Bloom filter using the test datasets described in Table 1 [Calderoni and Palmieri 2017]. Both datasets have a total of 65280 elements distributed over 255 sets. In the uniform dataset, each set contains exactly 256 elements, while in



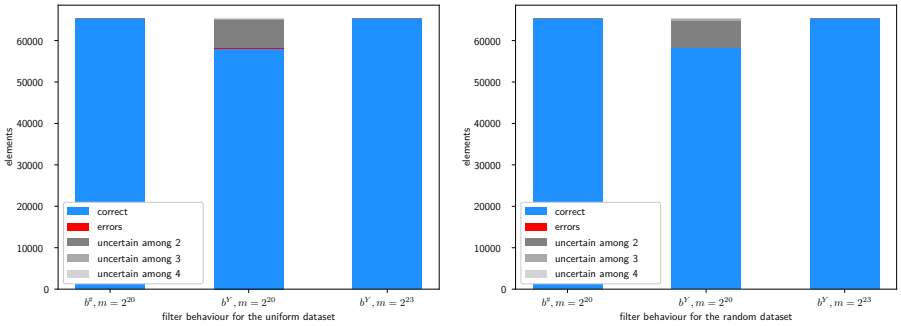
(a) $m = 2^{20}$ *(b)* $m = 2^{23}$

Figure 6: A plot of the false positive probability as expressed in (11) and (21) as the number of originating sets s increases. The filter length is $m = 2^{20}$ in (a), and $m = 2^{23}$ in (b).

the random dataset the elements are distributed randomly between sets. In order to test the false positive probability, we use an additional dataset of 500000 "non-elements" that are not part of either test datasets, and therefore should ideally be detected as not belonging to any of the originating sets.

Results of the false positive probability experiment are depicted in Figure 5. The figure plots the observed ratio of false positives over the uniform dataset. On the left hand side, the ratio is calculated over a varying filter legth in bits l, while on the right hand side this is done over the number of cells m. Given that $l = m$ for the ShBF, as described above, the experimentally observed probability remains the same, while the ratio changes for the SBF. The length of the filter (in bits and cells) over which the two graphs are plotted is aimed at covering the spectrum of different possible instances of use of the filters, and to encompass low expected false positive ratios (to the right of each graph) and high expected false positive ratios (to the left). For this reason, the length parameter in the graph ranges from low values starting at 2^{17} – which imply high false positive rate – to high values up to 2^{28}, where the false positive rate becomes (close to) 0. From the experiment, we can see that the SBF performs better if we consider the number of cells m, resulting in a negligible number of false positives for $m \geq 2^{20}$. However, if we consider the memory space used by the filter and therefore the filter length in bits, ShBF can provide optimal results starting at $l = 2^{21}$ while the SBF at $l = 2^{23}$. Based on these results, we select values of m and l in the range from 2^{20} to 2^{23} as the parameters for further experiments, as these values provide a probability – and therefore expected rate – of false positives and inter-set errors that is neither too low (negligible) nor too high (over 0.1, where the filters become unusable in most use cases), as seen in Figure 5.

While the experiment depicted in Figure 5 was carried out keeping the number of originating sets fixed, it is important to point out that, concerning false positive probability, the SBF outperforms the ShBF as the number s of originating sets increases. This condition is evident from (11) and (21), and is outlined in Figure 6. In particular, for $m = 2^{20}$ the false positive probability for the ShBF increases from close to 0 for a single set to over 0.1 for 250 sets (Figure 6a). In the case of a larger filter with $m = 2^{23}$, the false positive probability for the ShBF ranges from close to 0 to 1.4×10^{9}. The false positive probability of the SBF is instead independent of the number of sets, as expected.



(a) Uniform dataset. *(b) Random dataset.*

Figure 7: Filter behaviour of the two data structures over two datasets, (a) with a uniform distribution of elements among the 255 sets, (b) with a random distribution. The SBF is tested over a number of cells $m = 2^{20}$ and filter length in bits $l = 2^{23}$ (left). The ShBF is tested over the same number of cells $m = l = 2^{20}$ (centre); and over the same filter length in bits with $m = l = 2^{23}$ (right). For a ShBF, the number of cells and the filter length in bits coincide.

Figure 7 depicts the behaviour of the ShBF and SBF over both the uniform and random datasets with respect to inter-set errors. The bar on the left hand side for both graphs shows results for the SBF over a number of cells $m = 2^{20}$ and filter length in bits $l = 2^{23}$. The ShBF is tested over the same number of cells $m = 2^{20}$ as the SBF in the bar at the centre; and over the same filter length in bits with $l = 2^{23}$ in the bar on the right hand side. Since some outcomes concerning errors are not easily visible, results of the experiment are also provided in Table 2.

Given that the computational cost (in terms of number of hash computations) increases significantly as the number of sets increases for the ShBF (as discussed in the following section, and Table 3), we performed the tests over large but compact datasets of 65280 elements and 255 sets. Larger datasets would not provide significantly different results from the error probability perspective, given the filter parameters m and k are adjusted accordingly, but would incur in a largely increased computation time. Again, we notice here that the ShBF outperforms the SBF in terms of space efficiency, but not in terms of cell numbers. It is important to note here the different behaviour of the two data structures with respect to the occurrence of an inter-set error. In the case of the ShBF, an inter-set error happens when the originating set cannot be determined exactly, because more than one potential set is returned by the filter over a query. However, the correct set is always included in the returned list. In the case of the SBF, instead, and inter-set error returns a single set, which is not the correct originating set. In order to compare the different behaviours, we introduce an *entropy* metric here (modeled around the entropy concept first proposed by Shannon) which assesses the amount of information returned by a query. For the purpose of this work, we consider a correct result as having entropy 1, an incorrect result (possible only for the SBF) as having entropy 0, and a doubtful result (when multiple sets are returned by querying an ShBF) as $1/u$ where u is the number of sets returned. Average entropy results (ent) for the experiment are provided in Table 2, where we call an event where $u = i$ as u_i. The results indicate that the ShBF has a

filter	m	c	e	u_2	u_3	u_4	u_5	ent
				UNIFORM DATASET				
SBF	2^{20}	65276	4	-	-	-	-	0.99994
ShBF	2^{20}	58174	-	6739	352	15	0	0.94462
ShBF	2^{23}	65276	-	4	0	0	0	0.99997
				RANDOM DATASET				
SBF	2^{20}	65277	3	-	-	-	-	0.99995
ShBF	2^{20}	58282	-	6600	379	18	1	0.94536
ShBF	2^{23}	65278	-	2	0	0	0	0.99998

Table 2: Results of the experiment also depicted in Figure 7. c is the number of elements whose set was correctly identified. For the SBF, the number of elements recognised as belonging to a wrong set is e. For the ShBF, u_i is the number of elements that were identified as belonging to one out of i sets. The resulting entropy ent is also provided for both data structures, calculated as described in Section 3.

marginally higher entropy for the same filter length l for both distributions. Finally, we note here that while inter-set errors are distributed uniformly in the case of the ShBF, the SBF has a higher occurrence of inter-set errors for sets that have low index (that is, those that are entered first into the filter during construction). Therefore, the filter can be tuned to have different error probabilities for different sets, which may be an advantage in certain application scenarios.

3.1 Computational cost

In analysing the error probability of the two data structures, we referred to the memory usage of the filters. However, the operation of both data structures also implies a computational cost, which is different between the two constructions. In particular, in the following we analyse the computational cost associated with a single query to a filter. In general, cryptographic operations are the most computationally expensive, and therefore we analyse here the number of hash computations required for a query. Results are shown in Table 3.

A shifting Bloom filter requires $(k + s - 1)$ hash digests to be computed for each query (where k is the number of hash functions chosen at filter construction, and s is the number of sets), and each query also involves s lookups. Moreover, a single query involves reading a minimum of s cells (if a 0 is returned in the first position of all set offsets) and a maximum of $(s \cdot k)$ (in the worst case scenario for which a 1 is returned for each cell read). A spatial Bloom filter, instead, requires a constant number of hash computations k and a single lookup for each query, with a minimum of 1 and a maximum

filter	lookups/query	hashes/query	cells read/query [min,max]
ShBF	s	$k + s - 1$	$[s, (s \cdot k)]$
SBF	1	k	$[1, k]$

Table 3: Computational cost (expressed as number of hash computations required) for each lookup and query.

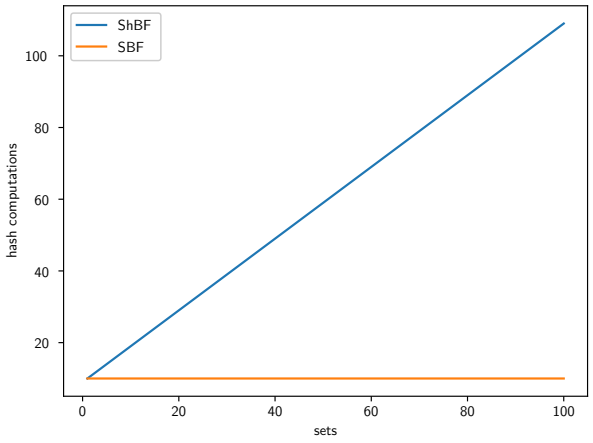


Figure 8: Number of hash computations required for a single query over the ShBF (linear) and SBF (constant) over the number of sets.

of k cells read. We can therefore conclude that the computational cost of a ShBF is significantly higher per query than the cost of an equivalent SBF, as evident in Figure 8.

4 Conclusions

In this paper we compared the shifting Bloom filters and the spatial Bloom filters, two probabilistic data structures designed to support association queries. For the former data structure, we also provide a novel generalised definition allowing an unbounded number of originating sets, and we discuss the resulting probabilistic model in detail. We implemented and tested both data structures over several datasets. Results show that the two data structures provide different benefits, and an adoption choice should depend on the application context. In particular, the probabilistic model of the spatial Bloom filter shows that it outperforms the shifting Bloom filter concerning both false positives and inter-set errors when the number of cells is considered. However, when effective memory consumption is considered, the shifting Bloom filter can achieve similar error rates to the spatial Bloom filter using less memory. With regards to computational costs, SBF requires a constant number of hash calculations, while ShBF requires a number of hash digest computations that increases linearly with the number of sets.

Acknowledgements

The authors would like to thank Herakliusz Oskar Lipiec, who worked on a python implementation of the shifting Bloom filter data structure as part of his final year project at University College Cork.

References

[Bloom 1970] Bloom, B. H. (1970). Space/Time Trade-offs in Hash Coding with Allowable Errors. Communications of the ACM 13 (7), 422–426.

[Bose et al. 2008] Bose, P., Guo, H., Kranakis, E., Maheshwari, A., Morin, P., Morrison, J., Smid, M. H. M. and Tang, Y. (2008). On the false-positive rate of Bloom filters. Information Processing Letters 108 (4), 210–213.

[Broder and Mitzenmacher 2003] Broder, A. Z. and Mitzenmacher, M. (2003). Survey: Network Applications of Bloom Filters: A Survey. Internet Mathematics 1 (4), 485–509.

[Calderoni et al. 2020] Calderoni, L., Bandini, S. and Maio, D. (2020). Privacy preservation in outsourced mobility traces through compact data structures. Journal of Information Security and Applications 55, 102630.

[Calderoni and Palmieri 2017] Calderoni, L. and Palmieri, P. (2017). libSBF-testdatasets v.1.1. https://github.com/spatialbloomfilter/libSBF-testdatasets.

[Calderoni et al. 2015] Calderoni, L., Palmieri, P. and Maio, D. (2015). Location privacy without mutual trust: The spatial Bloom filter. Computer Communications 68, 4–16.

[Calderoni et al. 2018] Calderoni, L., Palmieri, P. and Maio, D. (2018). Probabilistic Properties of the Spatial Bloom Filters and Their Relevance to Cryptographic Protocols. IEEE Transactions on Information Forensics and Security 13 (7), 1710–1721.

[Chang et al. 2008] Chang, F., Dean, J., Ghemawat, S., Hsieh, W. C., Wallach, D. A., Burrows, M., Chandra, T., Fikes, A. and Gruber, R. E. (2008). Bigtable: A Distributed Storage System for Structured Data. ACM Transactions on Computer Systems 26 (2), 4:1–4:26.

[Chang et al. 2004] Chang, F., Li, K. and Feng, W. (2004). Approximate Caches for Packet Classification. In The 23rd IEEE Conference on Computer and Communications, INFOCOM 2004, Hong Kong, China, March 7-11, 2004 pp. 2196–2207,.

[Charles and Chellapilla 2008] Charles, D. X. and Chellapilla, K. (2008). Bloomier Filters: A Second Look. In Algorithms - ESA 2008, 16th Annual European Symposium, Karlsruhe, Germany, Sept 15-17, 2008. Proceedings pp. 259–270,.

[Chazelle et al. 2004] Chazelle, B., Kilian, J., Rubinfeld, R. and Tal, A. (2004). The Bloomier filter: an efficient data structure for static support lookup tables. In Proceedings of the Fifteenth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2004, New Orleans, Louisiana, USA, January 11-14, 2004 pp. 30–39,.

[Dharmapurikar et al. 2006] Dharmapurikar, S., Krishnamurthy, P. and Taylor, D. E. (2006). Longest prefix matching using bloom filters. IEEE/ACM Trans. Netw. 14 (2), 397–409.

[Fan et al. 2000] Fan, L., Cao, P., Almeida, J. M. and Broder, A. Z. (2000). Summary cache: a scalable wide-area web cache sharing protocol. IEEE/ACM Transactions on Networking 8 (3), 281–293.

[Geravand and Ahmadi 2013] Geravand, S. and Ahmadi, M. (2013). Bloom filter applications in network security: A state-of-the-art survey. Computer Networks 57 (18), 4047–4064.

[Gomez-Barrero et al. 2016] Gomez-Barrero, M., Rathgeb, C., Galbally, J., Busch, C. and Fiérrez, J. (2016). Unlinkable and irreversible biometric template protection based on bloom filters. Information Sciences 370-371, 18–32.

[Grandi 2018] Grandi, F. (2018). On the analysis of Bloom filters. Information Processing Letters 129, 35–39.

[Hao et al. 2009] Hao, F., Kodialam, M. S., Lakshman, T. V. and Song, H. (2009). Fast Multiset Membership Testing Using Combinatorial Bloom Filters. In The 28th IEEE Conference on Computer Communications, INFOCOM 2009, 19-25 April 2009, Rio de Janeiro, Brazil pp. 513–521,.

[Hao et al. 2012] Hao, F., Kodialam, M. S., Lakshman, T. V. and Song, H. (2012). Fast Dynamic Multiple-Set Membership Testing Using Combinatorial Bloom Filters. IEEE/ACM Transactions on Networking 20 (1), 295–304.

[Liu and Özsu 2018] Liu, L. and Özsu, M. T., eds (2018). Encyclopedia of Database Systems, Second Edition. Springer.

[Luo et al. 2019] Luo, L., Guo, D., Ma, R. T. B., Rottenstreich, O. and Luo, X. (2019). Optimizing Bloom Filter: Challenges, Solutions, and Comparisons. IEEE Communications Surveys and Tutorials 21 (2), 1912–1949.

[Palmieri et al. 2014] Palmieri, P., Calderoni, L. and Maio, D. (2014). Spatial Bloom Filters: Enabling Privacy in Location-Aware Applications. In Information Security and Cryptology - 10th International Conference, Inscrypt 2014, Beijing, China, December 13-15, 2014, Revised Selected Papers pp. 16–36,.

[Palmieri et al. 2017] Palmieri, P., Calderoni, L. and Maio, D. (2017). Private inter-network routing for Wireless Sensor Networks and the Internet of Things. In Proceedings of the Computing Frontiers Conference, CF'17, Siena, Italy, May 15-17, 2017 pp. 396–401,.

[Patgiri et al. 2019] Patgiri, R., Nayak, S. and Borgohain, S. K. (2019). Hunting the Pertinency of Bloom Filter in Computer Networking and Beyond: A Survey. Journal of Computer Networks and Communications 2019, 2712417:1–2712417:10.

[Song et al. 2009] Song, H., Hao, F., Kodialam, M. S. and Lakshman, T. V. (2009). IPv6 Lookups using Distributed and Load Balanced Bloom Filters for 100Gbps Core Router Line Cards. In The 28th IEEE Conference on Computer Communications, INFOCOM 2009, 19-25 April 2009, Rio de Janeiro, Brazil pp. 2518–2526,.

[Tarkoma et al. 2012] Tarkoma, S., Rothenberg, C. E. and Lagerspetz, E. (2012). Theory and Practice of Bloom Filters for Distributed Systems. IEEE Communications Surveys and Tutorials 14 (1), 131–155.

[Wang et al. 2004] Wang, W., Jiang, H., Lu, H. and Yu, J. X. (2004). Bloom Histogram: Path Selectivity Estimation for XML Data with Updates. In The 30th Intl. Conference on Very Large Data Bases, VLDB 2004, Toronto, Canada, August 31 - Sept 3 2004 pp. 240–251, Morgan Kaufmann.

[Xiong et al. 2014] Xiong, S., Yao, Y., Cao, Q. and He, T. (2014). kBF: A Bloom Filter for key-value storage with an application on approximate state machines. In The 33rd IEEE Conference on Computer Communications, INFOCOM 2014, Toronto, Canada, April 27 - May 2, 2014 pp. 1150–1158,.

[Yang et al. 2017] Yang, D., Tian, D., Gong, J., Gao, S., Yang, T. and Li, X. (2017). Difference Bloom Filter: A probabilistic structure for multi-set membership query. In IEEE International Conference on Communications, ICC 2017, Paris, France, May 21-25, 2017 pp. 1–6,.

[Yang et al. 2016] Yang, T., Liu, A. X., Shahzad, M., Zhong, Y., Fu, Q., Li, Z., Xie, G. and Li, X. (2016). A Shifting Bloom Filter Framework for Set Queries. Proceedings of the VLDB Endowment 9 (5), 408–419.

[Yang et al. 2019] Yang, T., Yang, D., Jiang, J., Gao, S., Cui, B., Shi, L. and Li, X. (2019). Coloring Embedder: A Memory Efficient Data Structure for Answering Multi-set Query. In 35th IEEE International Conference on Data Engineering, ICDE 2019, Macao, China, April 8-11, 2019 pp. 1142–1153, IEEE.