

# An Optimization Problem Related to Bloom Filters with Bit Patterns

Peter Damaschke<sup>1</sup>(✉) and Alexander Schliep<sup>2</sup>

<sup>1</sup> Department of Computer Science and Engineering,  
Chalmers University, 41296 Gothenburg, Sweden

`ptr@chalmers.se`

<sup>2</sup> Department of Computer Science and Engineering,  
University of Gothenburg, Gothenburg, Sweden

`schliep@cse.gu.se`

**Abstract.** Bloom filters are hash-based data structures for membership queries without false negatives widely used across many application domains. They also have become a central data structure in bioinformatics. In genomics applications and DNA sequencing the number of items and number of queries are frequently measured in the hundreds of billions. Consequently, issues of cache behavior and hash function overhead become a pressing issue. Blocked Bloom filters with bit patterns offer a variant that can better cope with cache misses and reduce the amount of hashing. In this work we state an optimization problem concerning the minimum false positive rate for given numbers of memory bits, stored elements, and patterns. The aim is to initiate the study of pattern designs best suited for the use in Bloom filters. We provide partial results about the structure of optimal solutions and a link to two-stage group testing.

**Keywords:** Bloom filter · Genomics · Antichain · Group testing  
Disjunct matrix

## 1 Introduction

The following scenario appears in various applications of computing: A large set  $S$  of data is maintained. Further elements may be added to  $S$ , but usually elements are never removed. Many queries of the form “ $s \in S$ ?” must be answered, where  $s$  is any element from the domain of discourse. To facilitate quick answers, a certain rate of false positives is permitted: The system may sometimes claim  $s \in S$  although actually  $s \notin S$ . However, false negatives are not allowed: The system must recognize that  $s \in S$  whenever this is the case. A popular example of a data structure providing this functionality is the Bloom filter [2].

The word filter in the name indicates their use in avoiding accesses to the set  $S$ , in particular if slow accesses over networks or to disks become necessary. If these costly operations only happen for items passing the filter, the computational cost of many operations can be reduced, if the filter data structure allows inserts and queries in a time- and space-efficient manner. A number of database products (Google BigTable, Apache Cassandra, and Postgresql)

and web proxies (Squid) use Bloom filters; they have also been used to accelerate network router performance [23].

A specific application area in which Bloom filters have become a central data structure is the bioinformatics analysis of high-throughput DNA sequencing data from clinical or genomics experiments. In its analysis the  $k$ -mer, a consecutive substring of  $k$  characters in a *DNA read*, or string obtained from a sequencing instrument, is a fundamental unit for two reasons: First, the nodes of the *de Bruijn*-graph [7] are  $k$ -mers. The *de Bruijn*-graph is the most frequently used data structure in *de novo* genome assembly [25], the process of assembling complete genomes from the many fragments obtained from a sequencing instrument. Second, due to the way the errors are distributed in DNA sequencing reads, frequent  $k$ -mers seen a number of times represent the error-free sequence of the genome, whereas  $k$ -mers seen only once are erroneous. This explains the importance of identifying frequent  $k$ -mers [21] and their use in error correction, gene expression analysis and metagenomics, to list just a few applications.

The size of the problem instances are huge. A DNA sequencing data set of a human genome might contain 240 billion  $k$ -mers in its 2.4 billion reads. One expects about 3 billion  $k$ -mers to be frequent, and there could be up to 270 billion distinct erroneous  $k$ -mers for  $k = 31$  [21]. Consequently, the filters have to be large too, in the tens of Gigabyte range, which amplifies the effect of cache misses. As the cost of the Bloom filter operations makes up a large proportion of the total running time, constants, complexity and details of cache behavior matter and a better understanding of these aspects of the filters will have impact on many practical applications. In the following, we first introduce the particular type of Bloom Filter we will study and motivate the hypothesis pursued in this study.

### 1.1 Blocked Bloom Filters with Bit Patterns

Bloom filters [2] are a classic implementation of the filtering idea introduced in the previous section. Although they have some issues (deletions are not supported, and the space is 1.44 times larger than optimal) and alternatives came up more recently in theory [11, 19] and practice [13], Bloom filters remain an important and widely used data structure due to their conceptual simplicity.

Some basic notation is needed for the subsequent descriptions. A *vector* always means a bit vector, if not said otherwise. For vectors  $x, y$  of equal length, let  $x \cap y$  and  $x \cup y$  denote the bit-wise AND and OR, respectively. We write  $y \leq x$  if  $y$  is contained in  $x$ , that is, for every entry 1 in  $y$ , the entry at the same position in  $x$  is 1 as well. We write  $y < x$  if  $y \leq x$  and  $y \neq x$ . For a set or multiset  $X$  of vectors,  $\bigcup X$  is the vector obtained by joining all vectors in  $X$  by  $\cup$ . The *complement* of a vector  $x$  is obtained by flipping all bits. To *set a bit* means to assign value 1 to it. We wish to store a subset  $S$  of a universal set  $U$ .

*Bloom filters with bit patterns* have been proposed in [20], and standard Bloom filters appear as a special case of them. Their work can be described as follows. (A formal definition would be lengthy.) The filter consists of a vector of  $m$  bits, whose length  $m$  is chosen depending on parameters of the application. The filter uses a hash function  $h : U \rightarrow \{0, 1\}^m$ , assigning a vector to every element  $s \in U$ . We call this vector the *pattern* of  $s$ . Let  $d$  be the number of elements

in  $S \subset U$ , and let  $x_1, \dots, x_d$  be their patterns. Only the vector  $x := x_1 \cup \dots \cup x_d$  is stored. In order to test whether  $s \in S$ , one takes the pattern  $y$  of  $s$  and checks whether  $y \leq x$ . If not, then clearly  $s \notin S$ . If  $y \leq x$ , then  $s \in S$  is assumed. Hence  $s \notin S$  is a false positive if and only if  $y \leq x_1 \cup \dots \cup x_d$ . In particular, any collision of patterns,  $y = x_i$ , causes a false positive.

Specifically, the *standard Bloom filter* uses a fixed  $k$  and builds the pattern by setting up to  $k$  bits, chosen by  $k$  independent hash functions with values in  $\{1, \dots, m\}$ ; note that the  $k$  bits are not necessarily distinct. For variations of this original idea and theoretical analysis of their properties, in particular, the false positive rate versus space, see, e.g., [3, 8, 15, 18].

A *blocked Bloom filter* consists of many small blocks. A hash function first chooses a block (or blocks are partly predetermined, as the elements may be already grouped in some way), and then the blocks work like usual Bloom filters. Blocked Bloom filters with bit patterns have been proposed in [20]. It is one way to reduce both cache misses and hashing, which make up for the major part of the running time in some applications. Different applications can have very different demands on the false positive rate, memory space, time complexity, cache- and hash-efficiency, etc., therefore it is worthwhile to have a variety of filters with different strengths regarding these parameters.

Getting back to the description of blocked Bloom filters with bit patterns, the  $n$  patterns to be used are precomputed, by random sampling of  $k$  bits per pattern, and stored. While the space needed for the patterns is negligible compared to the entire filter, too large an  $n$ , say larger than the Level-1 or Level-2 cache will lead to deteriorating performance. Too small an  $n$  will lead to more collisions and increase the errors rate, which can be remediated by increasing filter size. Collision resolution mechanisms may be used to get roughly the same number  $d$  of elements per block. Hashing can be drastically reduced even without deteriorating the asymptotic false positive rate [15], but these results are shown for  $d \rightarrow \infty$  whereas we are interested in blocks with small  $d$  and prescribed  $n$ .

## 1.2 Specific Problems and Our Contributions

Our hypothesis, prompted by observations with random  $k$ -set designs in combinatorial group testing [16], is that the exact choice of random  $k$ -bit patterns does have an effect in particular in light of small  $n$ . Also, it is not obvious that random  $k$ -bit patterns indeed attain optimal performance.

To our best knowledge, a study is missing that asks which set of bit patterns (i.e., which image of the hash function) is optimal to use in this approach to blocked Bloom filters, depending on the mentioned parameters  $m, d, n$ . The present paper is mainly devoted to this question.

We consider arbitrary probability distributions on the  $m$ -bit vectors (resulting from the part of the hash function that assigns patterns to elements in a block), and we ask which ones minimize the *false positive rate (FPR)*:

**Definition 1.** *Let  $m$  and  $d$  be fixed positive integers. Consider any probability distribution on the set of  $m$ -bit vectors. Draw  $d + 1$  vectors  $x_1, \dots, x_d$  and  $y$*

randomly and independently from the given distribution. We define the false positive rate (FPR) of the distribution, for the given  $d$ , as the probability of the event  $y \leq x_1 \cup \dots \cup x_d$ . The true negative rate (TNR) is the probability of the complementary event. We abbreviate the FPR and TNR for a fixed  $d$  by  $FPR(d)$  and  $TNR(d)$ , respectively. (Thus  $TNR(d) + FPR(d) = 1$ .) We refer to the probability of the event  $y \leq x_1 \cup \dots \cup x_d$ , where the  $x_i$  are fixed and only  $y$  is random, as a conditional FPR.

**Problem.** Given  $m, d, n$ , devise a probability distribution on the  $m$ -bit vectors that assigns positive probabilities to at most  $n$  vectors and minimizes  $FPR(d)$ .

First we look at  $d = 1$ . This is not yet the realistic case, because a block would barely hold just one element. We start with this case rather for theoretical reasons, as it already provides some structural insights. We show that the best  $FPR(1)$  is achieved by a random vector with a fixed number of 1s, which is  $m/2$  (rounded if  $m$  is odd). Note that these vectors form the largest antichain (with respect to  $\leq$ ) due to Sperner's theorem [24]. While the result is not surprising, its proof is not that straightforward: Clearly, the more vectors we use, the smaller can we make the probability of collisions ( $y = x_1$ ), and this is the only type of false positives caused by an antichain. But it must also be shown that using even more vectors, partly being in  $\leq$  relation, is not beneficial. Our argument is based on edge colorings in bipartite graphs and resembles the "sizes of shadows" in one of the proofs of Sperner's theorem, yet our objective is different. (The proof also closes a gap in a proof step of a side result, Theorem 4, in [5].) This start result raises further interesting points addressed in the sequel.

First, notice that a random vector with some *fixed* number of *distinct* bits has the optimal FPR, and its specification needs less than  $\log_2 2^m = m$  random bits, whereas, e.g., setting a fixed number of independent random bits has a worse FPR and needs  $\Theta(m \log m)$  random bits (more than  $m/2$  specifications of one out of  $m$  position, requiring  $\Theta(\log m)$  bits each).

Getting to the real case with general  $d$  (and also with limited  $n$ ), one may wonder if random vectors with some fixed number  $k$  of 1s (depending on  $m$  and  $d$ ) yield optimal  $FPR(d)$  as well. This conjecture will be disproved already by a counterexample being as small as  $m = 2$  and  $d = 2$ , but then we also obtain the following general result (that will point to a modified conjecture; see below): Some distribution with minimum  $FPR(d)$  has a support of a form that we call a weak antichain. While an antichain forbids any vectors in  $\leq$  relation, in a weak antichain we do allow such pairs of vectors provided that they differ in only one bit. (In [4] we proved that another combinatorial optimization problem shares the same property, also the basic proof idea of "quartet changes" is the same, however the proof details are problem-specific.) The relevance of this general theorem is that families of bit patterns in Bloom filters can be restricted to weak antichains, since other designs have only worse FPR values.

Note that setting  $k$  independent random bits with replacement violates the weak antichain property, which naturally leads to the idea of patterns, too. On the other hand, the proposal in [20] was just to use a "table of random  $k$ -bit patterns". The small example of non-optimality and the weak antichain

property suggest that it might be good to use some mixture of patterns with two consecutive numbers,  $k$  and  $k + 1$ , of 1 entries. This seems also plausible because for any given  $d$  and  $m$  one would hardly expect one optimal  $k$  that jumps when  $m$  grows.

We do not manage to solve the general optimization problem considered here, however its difficulty is explained by our last contribution that might be the main result: We show that FPR minimization is, essentially, equivalent to the (notoriously difficult) construction of optimal almost disjunct matrices, which are designs being known from the group testing problem. The connection between Bloom filters and group testing has been noticed earlier here and then, but we are not aware of an explicit result on their relationship, as provided here.

## 2 Preliminaries

In this section we collect some special notation and known facts, in the order of appearance in the paper, except disjunct matrices and group testing which fit more naturally in the technical sections.

We call the number of 1s in a vector  $u$  its *level*. (This number is also known as the Hamming weight, but later we want to avoid confusion with another weight.) We also use the phrase “level  $k$ ” to denote the *set* of all vectors with the same number  $k$  of bits 1.

We consider probability distributions  $\Phi$  on finite sets only. The *support* of  $\Phi$ , denoted  $supp(\Phi)$ , is the set of elements  $u$  (in our case: vectors) with nonzero probability  $p(u) > 0$ . The distribution  $\Phi$  is *uniform* on  $supp(\Phi)$  if all these  $p(u) > 0$  are equal.

As said before,  $p(u)$  denotes the probability of vector  $u$  in a given distribution. Sometimes it is more convenient to write  $p_U$  instead, where  $U$  is the set of positions of bits 1 in  $u$ . We also omit commas and brackets. For instance,  $p(1, 0, 1)$  is written as  $p_{13}$ . If  $U = \emptyset$ , we write  $p_0$ .

We presume that standard graph- and order-theoretic concepts not explained here are widely known. A classic theorem by König (1916) states that every bipartite graph with maximum degree  $\Delta$  allows an *edge coloring* with  $\Delta$  colors. That is, we can color the edges in such a way that edges with the same color always form a *matching*, i.e., they are pairwise disjoint. Different proofs and many algorithmic versions have been given later, see, e.g., [14].

An *antichain* in a partial order (e.g., in the partial order of  $m$ -bit vectors under the  $\leq$  relation), is a subset without any pairs  $y < x$ . Sperner’s theorem [24] states (rephrased) that the largest antichain in the set of  $m$ -bit vectors is simply the set of all vectors on level  $k$ , where  $k = \lfloor m/2 \rfloor$  or  $k = \lceil m/2 \rceil$ . Slightly relaxing the notion of antichain, we call a set  $A$  of  $m$ -bit vectors a *weak antichain* if for all vectors  $u, v \in A$  with  $u \leq v$ , vector  $v$  has at most one 1 entry more than vector  $u$ .

Let  $y$  be a vector and  $X$  a tuple of  $d$  vectors (which are in general not distinct). We define  $f(y, X) = 1$  if  $y \leq \bigcup X$ , and  $f(y, X) = 0$  otherwise. Note that  $FPR(d)$  in Definition 1 is the weighted sum of all  $f(y, X)$ , where the weight of  $(y, X)$

is the probability that  $d + 1$  vectors independently drawn from the distribution happen to be  $y$  and the vectors of  $X$  in the given order.

We say that a probability distribution  $\Phi$  on the  $m$ -bit vectors is *dominated* by another distribution  $\Psi$  if, for every  $d$ , the FPR of  $\Psi$  is no larger than that of  $\Phi$ . We call  $\Phi$  *undominated* if  $\Phi$  is not dominated by any  $\Psi \neq \Phi$ . Clearly, from the point of view of getting optimal FPR, only undominated distributions need to be considered.

Every probability distribution on finitely many elements and with rational numbers as probability values can be equivalently viewed as a uniform distribution on copies of the elements: Let  $q$  be a common denominator of all probabilities. Then we may represent every element of probability  $p/q$  as  $p$  distinct copies labeled by the element. We refer to these copies as *units*, and every unit is chosen with probability  $1/q$ . Notationally we may not always distinguish between a unit and its label, if this causes no confusion.

Let  $e$  denote Euler’s number. Logarithms are meant with base 2 if not said otherwise.

### 3 False Positive Rate for One Element

**Theorem 1.** *For every  $m > 1$ , the uniform distribution on a median level, i.e., on the level  $\lfloor m/2 \rfloor$  or  $\lceil m/2 \rceil$ , attains the smallest FPR(1).*

*Proof.* For any probability distribution  $\Phi$  on the  $m$ -bit vectors, observe that  $FPR(1) = \sum_u p(u)^2 + \sum_{u < v} p(u) \cdot p(v)$ , where the sums are taken over all vectors. Let  $k$  be the lowest level containing any vectors in  $supp(\Phi)$ , and assume that  $k \leq (m - 1)/2$ . We construct a weighted bipartite graph as follows. The vertices are all vectors  $u \in supp(\Phi)$  on level  $k$ , and all vectors  $v$  on level  $k + 1$  (including those with zero probability). We use the terms vector and vertex interchangeably. The edges are all pairs  $(u, v)$  with  $u < v$ . The weight of a vertex is its probability, and the weight of an edge  $(u, v)$  is  $p(u) \cdot p(v)$ .

Note that every vertex on level  $k + 1$  has a degree at most  $k + 1$ , and every vertex on level  $k$  has exactly the degree  $m - k \geq k + 1$ . By König’s theorem there exists an edge coloring with  $m - k$  colors. Clearly, every vertex on level  $k$  is incident to exactly one edge of each color. Since  $m > 1$ , we have  $m - k \geq 2$ . The total edge weight of the bipartite graph is  $b := \sum_{u < v} p(u) \cdot p(v)$ , where  $u$  and  $v$  are restricted to vertices on level  $k$  and  $k + 1$ , respectively. The color class of a color  $c$  is the set of all edges of this color  $c$ . Let  $M$  be a color class with minimum total edge weight, among all colors  $c$ . This weight can be at most  $b/2$ , since  $m - k \geq 2$ .

Now we modify the probabilities. For every vertex  $u$  on level  $k$  and its partner  $v$  in  $M$ , we set  $p(u) := 0$  and  $p(v) := p(v) + p(u)$ . Notice that  $v$  exists, and different vertices  $u$  have different partners  $v$ . The contribution of levels  $k$  and  $k + 1$  to  $FPR(1)$  decreases by  $b$  as we destroy all edges, and at the same time it increases by at most  $2b/2 = b$  because every new  $p(v)^2$  becomes  $(p(v) + p(u))^2 = p(v)^2 + 2p(u) \cdot p(v) + p(u)^2$ . In words: For every  $(u, v) \in M$ , the squared vertex weight  $p(u)^2$  just “moves into”  $p(v)^2$ , and the doubled edge weight is added. The

sum of all doubled edge weights in  $M$  is bounded by  $2b/2$ . Finally, no further positive terms in  $FPR(1)$  are created by moving probability mass to level  $k + 1$ : There are no further vertices  $w$  with  $p(w) > 0$  on lower levels, and for any  $w > v$  on higher levels we have already  $w > u$  by transitivity. Altogether it follows that we can empty the level  $k$  without increasing  $FPR(1)$ .

By symmetry,  $FPR(1)$  is not affected if we take the complements of all vectors. Thus the same reasoning applies also to the highest level  $k$  that intersects  $supp(\Phi)$ , assuming that  $k \geq (m + 1)/2$ . By iterating the procedure we can move all probability mass into the level  $m/2$  if  $m$  is even, or into one of  $\lfloor m/2 \rfloor$  or  $\lceil m/2 \rceil$  if  $m$  is odd. As the last step, the sum of squares of a fixed number of values with a fixed sum is minimized if these values are equal.  $\square$

In Theorem 1 we did not limit the size of the support, i.e., the number  $n$ . of patterns. Now let  $n$  be prescribed. Due to Theorem 1, if  $n > \binom{m}{\lfloor m/2 \rfloor}$ , we would still take only a median level and no further vectors, and if  $n \leq \binom{m}{\lfloor m/2 \rfloor}$ , we can take the uniform distribution on any antichain of  $n$  vectors to achieve the best  $FPR(1)$  which is then  $1/n$ .

### 4 Weak Antichains

**Theorem 2.** *For every  $m$ , every probability distribution on the  $m$ -bit vectors is dominated by some probability distribution whose support is a weak antichain.*

*Proof.* Let  $u$  and  $v$  be vectors such that  $u \leq v$ , and  $v$  has at least two 1 entries more than  $u$ . Clearly, we can get two vectors  $w$  and  $w'$  such that  $w \cap w' = u$ ,  $w \cup w' = v$ , and  $u, v, w, w'$  are four distinct vectors. Now let  $\Phi$  be any probability distribution on the vectors with  $u, v \in supp(\Phi)$ , that is,  $\Phi$  contains two such units  $u$  and  $v$ , and is therefore not a weak antichain. We replace one unit  $u$  with one unit  $w$ , and we replace one unit  $v$  with one unit  $w'$ . We call such a replacement a *quartet change*. We study how a quartet change affects the FPR.

In certain subsets (of sequences of vectors) with even cardinality we will pair up all members, i.e., divide them completely into disjoint pairs, and we refer to the members of every such pair as *partners*. In the following, observe that distinct units carrying the same label are still considered distinct (as units), and that to “appear” in a sequence means “at least once”.

Every argument  $(y, X)$  of  $f$ , where  $y$  is a unit and  $X$  is a sequence of  $d$  units, belongs to exactly one of the following cases:

- (a1) Both  $u$  and  $v$  are not  $y$ , nor do they appear in  $X$ .
- (a2) Both  $u$  and  $v$  are not  $y$ , and exactly one of them appears in  $X$ .
- (a3) Both  $u$  and  $v$  are not  $y$ , and both appear in  $X$ .
- (b1) Unit  $y$  is one of  $u$  and  $v$ , and both  $u$  and  $v$  do not appear in  $X$ .
- (b2) Unit  $y$  is one of  $u$  and  $v$ , and  $y$  appears in  $X$ .
- (b3) Unit  $y$  is one of  $u$  and  $v$ , and only the unit other than  $y$  appears in  $X$ .

Note that, in general,  $y$  itself may appear in  $X$ .

Since we are working with units, all  $(y, X)$  have the same probability, hence  $FPR(d)$  is simply the unweighted sum of all  $f(y, X)$ . In each of the cases we prove that  $FPR(d)$  cannot increase by the quartet change.

**Case (a1).** Trivially,  $f(y, X)$  is not affected by the quartet change.

**Case (a2).** We pair up the arguments of  $f$  that belong to this case: The partner of every  $(y, X)$  is defined by replacing all occurrences of our unit  $u$  with  $v$ , or vice versa. Let  $X_u$  and  $X_v$  be any such partners containing  $u$  and  $v$ , respectively, with unions  $x_u := \bigcup X_u$  and  $x_v := \bigcup X_v$ . We define  $X_w$  as the sequence obtained from  $X_u$  by replacing all occurrences of the unit  $u$  with  $w$ , and  $x_w := \bigcup X_w$ . Finally,  $X_{w'}$  and  $x_{w'}$  are defined similarly. In the distribution after the quartet change,  $X_w$  and  $X_{w'}$  are partners.

We claim that  $f(y, X_u) + f(y, X_v) \geq f(y, X_w) + f(y, X_{w'})$ . This claim follows from two observations: If both  $y \leq x_w$  and  $y \leq x_{w'}$ , then  $y \leq x_w \cap x_{w'} = x_u \leq x_v$ , where the inner equality is true by the distributive law for  $\cap$  and  $\cup$ . If only one of the former inequalities holds, say  $y \leq x_w$ , then we still have  $y \leq x_v$ .

**Case (a3).** Consider any such argument  $(y, X)$  as specified in this case, and let  $X'$  be the sequence obtained from  $X$  by the quartet change. Let  $x := \bigcup X$  and  $x' := \bigcup X'$ . We claim that  $f(y, X) \geq f(y, X')$ . To show this claim, we use that  $w \cup w' = v$ : If  $y \leq w \cup w'$  then trivially  $y \leq v$ . Together with the distributive law this shows: If  $y \leq x'$  then  $y \leq x$ .

**Case (b1).** Again we pair up the arguments of  $f$  that belong to the case: This time,  $(u, X)$  and  $(v, X)$  are partners, and the claim is that  $f(u, X) + f(v, X) \geq f(w, X) + f(w', X)$ . With  $x := \bigcup X$  observe the following: If both  $w \leq x$  and  $w' \leq x$ , then  $u \leq v = w \cup w' \leq x$ . If only one of the former inequalities holds, say  $w \leq x$ , then we still have  $u \leq x$ .

**Case (b2).** The same unit appears in the role of  $y$  and also in  $X$ , and it is replaced with the same unit at all occurrences. Thus we have  $f(y, X) = 1$  before and after the quartet change.

**Case (b3).** We pair up the arguments  $(u, X_v)$  and  $(v, X_u)$ , where  $X_u$  is obtained from  $X_v$  by replacing all occurrences of our unit  $u$  with  $v$ . Note that we also obtain  $X_v$  from  $X_u$  in the opposite direction. We define  $X_w$  and  $X_{w'}$  as in case (1). We also adopt the earlier notations for the unions.

We claim that  $f(u, X_v) + f(v, X_u) \geq f(w, X_{w'}) + f(w', X_w)$ . To show the claim, first note that trivially  $w \leq x_w$  and  $w' \leq x_{w'}$ . Now, if also both  $w \leq x_{w'}$  and  $w' \leq x_w$ , then  $u \leq v = w \cup w' \leq x_w \cap x_{w'} = x_u \leq x_v$ , where the first equality holds by definition, and the second equality was already used in case (1). If only one of the former inequalities holds, say  $w \leq x_{w'}$ , then it suffices to observe that  $u \leq v$ .

Finally, it is not hard to see that a sequence of quartet changes cannot cycle. Hence we always arrive at a weak antichain dominating the original distribution. □



## 5 Some Special Cases

The following propositions are proved by using extremal value calculations and Theorem 2; a full version is available at [www.cse.chalmers.se/~ptr](http://www.cse.chalmers.se/~ptr).

**Proposition 1.** *Among all distributions whose support is contained in the levels 0 and 1, the distribution minimizing  $FPR(d)$  is the following:*

*For  $d \geq m$ , assign probability  $1 - m/(d + 1)$  to the zero vector, and  $1/(d + 1)$  to every vector on level 1.*

*For  $d < m$ , assign probability  $1/m$  to every vector on level 1.*

*Moreover, for  $d < m$ , the (unrestricted) distribution minimizing  $FPR(d)$  does not have the zero vector in the support.*

**Proposition 2.** *For  $m \geq 2$  and  $d \geq 2$ , the support of any distribution minimizing  $FPR(d)$  does not include the vector on level  $m$ .*

Although these propositions treat only special aspects of our FPR minimization problem, they lead to some interesting conclusions:

Consider  $m = 2$  and  $d = 2$ . Proposition 2 yields  $p_{12} = 0$ . Thus we can apply Proposition 1, and therefore the best distribution is  $p_0 = p_1 = p_2 = 1/3$ . Already this small example shows that Theorem 1 does not generalize to  $d > 1$  in the way that the optimal  $FPR(d)$  is always attained by the uniform distribution on some single level. But together with Theorem 2 it suggests that the minimum  $FPR(d)$  might be attained by some distribution whose support is in at most two consecutive levels, and where all vectors on the same level have equal probabilities.

Whatever the conjecture is, it is not easy to see how the arguments in Theorem 1 can be generalized to  $d > 1$ . Informally, movements of probability mass from the lowest level upwards create larger unions  $x_1 \cup \dots \cup x_d$ . This makes it tricky to control the FPR, since probabilities can no longer be assigned to the edges of some graph.

So far we have usually assumed an unlimited number  $n$  of vectors in the support. The problem earns an extra dimension when a maximum  $n$  is prescribed as well, as in the following section.

## 6 Using Almost Disjunct Matrices

Disjunct matrices (see definitions below) are test designs for non-adaptive group testing [9], and relaxed versions are applied to two-stage group testing. In *non-adaptive group testing*,  $d$  unknown elements in a set of  $n$  elements have a specific property called *defective*, and these defective elements must be identified by  $m$  simultaneous group tests: A *group test* indicates whether a certain subset contains some defective or not. In *two-stage group testing* the aim is the same, but the job of the first stage is only to limit the possible defectives to a subset of candidates, which are then tested individually in the second stage. (There is also a version where the second stage can apply yet another non-adaptive group

testing scheme, but this problem version is not relevant in our current context.) Remarkably, two-stage group testing can accomplish a query number exceeding the information-theoretic lower bound only by a constant factor [6, 12], which is not possible in one stage.

We call a binary matrix  $(d, \varepsilon)$ -disjunct if  $y \leq x_1 \cup \dots \cup x_d$  happens with probability at most  $\varepsilon$ , when  $x_1, \dots, x_d$  are columns chosen independently and uniformly at random, and  $y$  is uniformly chosen among the remaining vectors, distinct from all  $x_i$ . (The definition in [1, 17] is slightly different, as it requires the  $x_i$  to be distinct as well, but the difference is marginal for  $d \ll n$ .) A  $(d, 0)$ -disjunct matrix is simply called  $d$ -disjunct. Informally we also refer to  $(d, \varepsilon)$ -disjunct matrices as *almost disjunct*. For the use of (almost) disjunct matrices in group testing we refer to the cited literature. In our context, the  $n$  columns are the patterns in a Bloom filter with  $m$  bits.

The event  $y \leq x_1 \cup \dots \cup x_d$  can occur for two reasons: either (1) a collision  $y = x_i$  happens for some  $i$ , or (2)  $y$  is in the union of  $d$  vectors other than  $y$ . We name the probabilities of (1) and (2) the *collision* and *containment probability*, respectively.

**Proposition 3.** *Among all distributions with a fixed support of size  $n \geq d + 1$ , the uniform distribution on the support has the smallest collision probability, which equals  $1 - (1 - 1/n)^d$ .*

*Proof.* We denote the  $n$  probabilities by  $q_1, \dots, q_n$ . The probability of no collision equals  $\sum_{i=1}^n q_i(1 - q_i)^d$ . We want to maximize this expression under the constraint  $\sum_{i=1}^n q_i = 1$ . From the first and second derivative one can see that the function  $q(1 - q)^d$  is increasing if and only if  $q < 1/(d + 1)$ , and concave if and only if  $q < 2/(d + 1)$ . It follows that, in an optimal solution, all  $q_i < 2/(d + 1)$  are equal, and  $q_i \geq 2/(d + 1)$  holds for at most one index. Denote the small and large value  $s$  and  $r$ , respectively. The assumption  $n \geq d + 1$  implies  $s < 1/n \leq 1/(d + 1)$ . Hence we can decrease  $r$  and increase  $s$  so as to preserve the sum constraint and improve the objective. It follows that, in an optimal solution, an index  $i$  with  $q_i = r$  cannot exist. Finally we get  $q_i = 1/n$  for all  $i$ .  $\square$

By virtue of Proposition 3 we focus now on filters that use a distribution being uniform on its support. We remark that, by simple calculation,  $1 - (1 - 1/n)^d = d/n - O((d/n)^2)$ , which is essentially  $d/n$ .

**Proposition 4.** *Any  $(d, \varepsilon)$ -disjunct  $m \times n$  matrix enables a Bloom filter with  $m$  bits,  $n$  patterns, and  $FPR(d) \leq 1 - (1 - 1/n)^d + (1 - 1/n)^d \varepsilon$ , where the patterns are assigned uniformly to the elements. The converse holds also true.*

*Proof.* Given a matrix as indicated, we take the uniform distribution on its columns. The collision probability is  $1 - (1 - 1/n)^d$ . The containment probability, in the event of no collision, is bounded by  $\varepsilon$ , by the definition of  $(d, \varepsilon)$ -disjunctness. Conversely, suppose that we have a Bloom filter as indicated. Again, the collision probability is equal to  $1 - (1 - 1/n)^d$  because of the uniform distribution. Since  $FPR(d) \leq 1 - (1 - 1/n)^d + (1 - 1/n)^d \varepsilon$  is assumed, the containment probability in the case of no collision cannot exceed  $\varepsilon$ . Hence we can view the patterns as the columns of some  $(d, \varepsilon)$ -disjunct  $m \times n$  matrix.  $\square$

Proposition 4 states that, at least for uniform distributions, constructing Bloom filters with bit patterns that are optimal (in terms of FPR, space, and amount of patterns and hash bits) is essentially equivalent to constructing optimal almost-disjunct matrices. The next natural question concerns the possible trade-offs between the parameters. The smallest possible row number of  $d$ -disjunct matrices behaves as  $m = \Theta(d^2/\log d) \log n$  [10]. Unfortunately, with  $\varepsilon := d/n$  this leads to  $m/d = \Theta(d/\log d)(\log d + \log(1/\varepsilon))$ , i.e., the space per element ratio is by a  $\Theta(d/\log d)$  factor worse than in standard Bloom filters where  $m/d = 1.44 \log(1/\varepsilon)$ . For  $d = 1$  we remark that the optimal 1-disjunct matrices are the Sperner families, and according to Theorem 2 they have optimal  $FPR(1)$ . But for  $d > 1$ , using  $d$ -disjunct matrices quickly becomes unsuitable.

The picture becomes better with  $(d, \varepsilon)$ -disjunct matrices. As mentioned in [1, 17], it is possible to achieve  $m = \Theta(d \log n)$  due to [26] (and hence the additional  $\Theta(d/\log d)$  factor disappears), although the cited result was not constructive. But it was not noticed in [1, 17] that a special type of  $(d, \varepsilon)$ -disjunct matrices with  $m = \Theta(d \log n)$  rows and even better properties is known as well [6, 12]. We will utilize them now.

A binary matrix is called  $(d, f)$ -resolvable if, for any  $d$  distinct columns  $x_1, \dots, x_d$ , the inclusion  $y \leq x_1 \cup \dots \cup x_d$  holds for fewer than  $f$  columns  $y$  other than the  $x_i$  [12]. Note that any  $(d, f)$ -resolvable matrix is also  $(d, f/(n - d))$ -disjunct, and the resulting false positive probability bound holds even conditional on every tuple  $x_1, \dots, x_d$ , not only averaged over all tuples. A counterpart of Proposition 4 holds for resolvable matrices and conditional FPR.

Specifically, Theorem 2 in [12] provides, for every integer  $f > 0$ , a  $(d, f)$ -resolvable matrix with  $m = 2(d^2/f) \log(en/d) + 2d \log(en/f) + 2(d/f) \log n$  rows. This yields, in a few steps:  $m/d = 2(d/f) \log(en/d) + 2 \log(en/f) + (2/f) \log n = (2(d + f + 1)/f) \log(n/d) + 2(d/f) \log e + 2 \log(d/f) + 2 \log e + (2/f) \log d$ . For notational convenience we define  $r = d/n$ ,  $s = f/n$ , and  $t = (d+f)/n$ . We assume bounded ratios  $r/s$  and  $s/r$  and (for studying the asymptotics for growing  $n$ ) we neglect the terms that do not depend on  $n$ . Then the above equation simplifies to  $m/d = 2(1 + r/s) \log(1/r)$ . Further rewriting gives  $m/d = 2t/(t - r) \cdot \log(1/r)$ , which we use below.

It is not totally obvious that the most efficient resolvable matrices, that maximize  $n$  for given  $m$  and  $d$ , also yield the smallest  $FPR(d)$  of Bloom filters of this type. While the collision probability improves (i.e., decreases) for growing  $n$ , the containment probability increases, as the relation between  $n$  and the fixed  $m$  becomes worse. But, in fact, we can establish monotonicity.

**Proposition 5.** *When the columns of the  $(d, f)$ -resolvable matrices from [12] are used as patterns in a Bloom filter, then  $FPR(d)$  decreases for growing  $n$ .*

*Proof.* Let  $C$  denote the factor for which  $m = Cd \log n$ . Note that  $C \geq 2$ . Solving  $m/d = 2t/(t - r) \cdot \log(1/r)$  for  $t$  yields  $t = r/(1 - (2/\ln 2)(d/m) \ln(1/r))$ . Taking the derivative with respect to  $r$  by using the quotient rule yields the denominator  $1 - (2/\ln 2)(d/m)(\ln(1/r) + 1)$ . We have the following chain of equivalent inequalities:  $1 - (2/\ln 2)(d/m)(\ln(1/r) + 1) > 0 \iff \ln(1/r) < \ln 2 \cdot$

$(m/2d) - 1 \iff 1/r < (1/e) \cdot 2^{m/(2d)} \iff n/d < (1/e) \cdot 2^{Cd \log n/(2d)} \iff n < (d/e) \cdot 2^{(C/2) \log n} \iff n < (d/e) \cdot n^{C/2}$ , and the latter one is true for  $C \geq 2$ . Hence the derivative is positive in the relevant range of  $r$ , therefore  $t$  decreases with growing  $1/r$ , and the assertion follows.  $\square$

On the other hand, a larger  $n$  requires more space to store the patterns and more hashing. Still, the use of patterns is advantageous in this respect: In a design with  $n$  patterns, only  $\log n$  hash bits per element are needed. A standard Bloom filter needs  $\Theta((m/d) \log m) = \Theta(\log n \cdot \log m)$  hash bits per element. The constants depend on the desired FPR, but the extra  $\Theta(\log m)$  factor remains.

Another remark is that the resolvable matrices in [12] consist again of randomly chosen vectors where a fixed number of bits is set, and Bloom filters are explicitly mentioned as the inspiration. The opposite direction, namely, using randomly chosen vectors with a fixed number of 1s as bit patterns in blocked Bloom filters, was proposed in [20]. However, the known resolvable matrices are not necessarily optimal. (In general, constructions of improved test designs are a major theme in group testing research.) An intriguing question is whether there are better designs with a given number  $n$  of patterns, and this is our optimization problem.

## 7 Concluding Remarks

The actual construction of improved almost-disjunct matrices, and hence of better bit patterns for Bloom filters, is beyond the scope of this paper. Our partial results suggest that certain designs with vectors from *two* neighbored levels might be optimal. We notice that the construction of combinatorial designs with certain “almost-properties” gained new momentum recently [22].

We intend to design large-scale simulation experiments, to gain insights for real and simulated workloads of using Bloom filters with bit patterns. We expect to see differences between various pattern choices when viewing the FPR, as the number of items in the filter increases. There may not necessarily be differences in the FPR for the nominal design parameter representing the number of items, but in how the FPR behaves up to this point and beyond.

Yet another aspect could not be addressed here: Cache considerations prevent making the number  $n$  of patterns arbitrarily large, as the pattern storage needs to fit within primary or at least secondary caches. One possible approach to increase  $n$  (and thus reduce the FPR) in the same pattern storage would be the computation of the actually needed patterns on-the-fly from hash values, using only a small amount of auxiliary memory. Naturally, the design of the patterns must allow fast calculation. We are wondering if such designs exist, that do not compromise the other parameters too much.

**Acknowledgment.** We are grateful to the reviewers for encouragement and for careful comments that helped improve the notation and fix a calculation mistake.

## References

1. Barg, A., Mazumdar, A.: Almost Disjunct Matrices from Codes and Designs. CoRR abs/1510.02873 (2015)
2. Bloom, B.H.: Space/time trade-offs in hash coding with allowable errors. *Comm. ACM* **13**, 422–426 (1970)
3. Broder, A., Mitzenmacher, M.: Network applications of Bloom filters: a survey. *Internet Math.* **1**, 485–509 (2004)
4. Damaschke, P.: Calculating approximation guarantees for partial set cover of pairs. *Optim. Lett.* **11**, 1293–1302 (2017)
5. Damaschke, P., Muhammad, A.S.: Randomized group testing both query-optimal and minimal adaptive. In: Bieliková, M., Friedrich, G., Gottlob, G., Katzenbeisser, S., Turán, G. (eds.) SOFSEM 2012. LNCS, vol. 7147, pp. 214–225. Springer, Heidelberg (2012). [https://doi.org/10.1007/978-3-642-27660-6\\_18](https://doi.org/10.1007/978-3-642-27660-6_18)
6. De Bonis, A., Gasieniec, L., Vaccaro, U.: Optimal two-stage algorithms for group testing problems. *SIAM J. Comput.* **34**, 1253–1270 (2005)
7. de Bruijn, N.G.: A combinatorial problem. *Koninklijke Nederlandse Akademie v. Wetenschappen* **49**, 758–764 (1946)
8. Dillinger, P.C., Manolios, P.: Bloom filters in probabilistic verification. In: Hu, A.J., Martin, A.K. (eds.) FMCAD 2004. LNCS, vol. 3312, pp. 367–381. Springer, Heidelberg (2004). [https://doi.org/10.1007/978-3-540-30494-4\\_26](https://doi.org/10.1007/978-3-540-30494-4_26)
9. Du, D.Z., Hwang, F.K.: *Pooling Designs and Nonadaptive Group Testing*. World Scientific, New Jersey (2006)
10. Dyachkov, A.G., Vorobev, I.V., Polyansky, N.A., Shchukin, V.Y.: Bounds on the rate of disjunctive codes. *Probl. Inf. Transm.* **50**, 27–56 (2014)
11. Eppstein, D.: Cuckoo filter: simplification and analysis. In: Pagh, R. (ed.) SWAT 2016. LIPIcs, vol. 53, paper 8, Dagstuhl (2016)
12. Eppstein, D., Goodrich, M.T., Hirschberg, D.S.: Improved combinatorial group testing algorithms for real-world problem sizes. *SIAM J. Comput.* **36**, 1360–1375 (2007)
13. Fan, B., Andersen, D.G., Kaminsky, M., Mitzenmacher, M.: Cuckoo filter: practically better than Bloom. In: Seneviratne, A., et al. (eds.) CoNEXT 2014, pp. 75–88. ACM (2014)
14. Kapoor, A., Rizzi, R.: Edge-coloring bipartite graphs. *J. Algorithms* **34**, 390–396 (2000)
15. Kirsch, A., Mitzenmacher, M.: Less hashing, same performance: building a better Bloom filter. *Random Struct. Algorithms* **33**, 187–218 (2008)
16. Knill, E., Schliep, A., Torney, D.C.: Interpretation of pooling experiments using the Markov Chain Monte Carlo method. *J. Comput. Biol.* **3**, 395–406 (1996)
17. Mazumdar, A.: Nonadaptive Group Testing with Random Set of Defectives. CoRR abs/1503.03597 (2016)
18. Mitzenmacher, M., Upfal, E.: *Probability and Computing: Randomized Algorithms and Probabilistic Analysis*. Cambridge University Press, Cambridge (2005)
19. Pagh, A., Pagh, R., Srinivasa Rao, S.: An optimal Bloom filter replacement. In: SODA 2005, pp. 823–829 (2005)
20. Putze, F., Sanders, P., Singler, J.: Cache-, hash-, and space-efficient Bloom filters. *ACM J. Exp. Algorithms* **14**, Article 4.4 (2009)
21. Roy, R.S., Bhattacharya, D., Schliep, A.: Turtle: identifying frequent k-mers with cache-efficient algorithms. *Bioinformatics* **14**, 1950–1957 (2014)

22. Sarkar, K., Colbourn, C.J., de Bonis, A., Vaccaro, U.: Partial covering arrays: algorithms and asymptotics. In: Mäkinen, V., Puglisi, S.J., Salmela, L. (eds.) IWOCA 2016. LNCS, vol. 9843, pp. 437–448. Springer, Cham (2016). [https://doi.org/10.1007/978-3-319-44543-4\\_34](https://doi.org/10.1007/978-3-319-44543-4_34)
23. Song, H., Dharmapurikar S., Turner J., Lockwood, J.: Fast hash table lookup using extended Bloom filter: an aid to network processing. In: Guérin, R., Govindan, R., Minshall, G.: SIGCOMM 2005, pp. 181–192. ACM (2005)
24. Sperner, E.: Ein Satz über Untermengen einer endlichen Menge. *Math. Zeitschrift* **27**, 544–548 (1928)
25. Zerbino, D.R., Birney, E.: Velvet: algorithms for de novo short read assembly using de Bruijn graphs. *Genome Res.* **18**, 821–829 (2008)
26. Zhigljavsky, A.: Probabilistic existence theorems in group testing. *J. Stat. Plann. Infer.* **115**, 1–43 (2003)