A New Analysis of the False-Positive Rate of a Bloom Filter

Ken Christensen[1], Allen Roginsky[2], and Miguel Jimeno[1]

(Correspondence author: Ken Christensen)

[1] Department of Computer Science and Engineering
University of South Florida
4202 East Fowler Avenue, ENB 118
Tampa, Florida USA 33620
Email: {*christen, mjimeno*}*@csee.usf.edu*
Phone: (813) 974-4761
FAX: (813) 974-5456

[2] Computer Security Division
National Institute of Standards and Technology
Gaithersburg, Maryland USA 20899
Email: *allen.roginsky@nist.gov*

Keywords

Data structures; Analysis of algorithms; Bloom filters

Submission editor

S. Hambrusch

1. INTRODUCTION

A Bloom filter is a space-efficient data structure used for probabilistic set membership testing. The Bloom filter was invented by Bloom in 1970 [1] and has found widespread application in many domains of Computer Science. Bloom filters have many uses in databases, network applications (a major survey is in [3]), and even by Google in the core of their search engine [4]. A Bloom filter can be implemented in hardware or software.

A Bloom filter is an array of *m* bits indexed from 1 to *m* that is initially clear (all bits set to 0). An object (for example, a string) is added, or mapped-in, to a Bloom filter by inputting it to a group of *k* independent hash functions. Each hash function hashes the object into an integer value between 1 and *m*, this value is then used as an index position in the Bloom filter array. The resulting *k* array index positions in the Bloom filter array are set to 1. An object is tested for set membership by inputting it to the same group of *k* hash functions. If all *k* generated array positions are set to 1, then the object is probably a member. Non-member objects may coincidentally map to set bit positions in the Bloom filter array, thus false positives are possible. False negatives are not possible. The probability of a false positive – or false positive rate – of a Bloom filter is a function of the randomness of the values generated by the hash functions and of *m*, *n*, and *k* (*n* is the number of objects mapped into the Bloom filter). Given the widespread application of Bloom filters, a thorough and correct understanding of the false positive rate is needed.

In the analysis of Bloom filter false positive rate it is typically assumed that the hash functions are perfect whereby they produce an independent and random index value for each object and thus the false positive rate is only a function of *m*, *n*, and *k*. The "classic" analysis of Bloom filter false positive rate is as follows. This analysis is often attributed to Bloom [1], but his original analysis was different. This classic analysis probably first appeared in Mullin [7]. The probability that an arbitrary bit is not set after *k* bit insertions from the mapping of one object is $(1-1/m)^k$. For *n* objects mapped-in, the probability that an arbitrary bit is not set is $(1-1/m)^{kn}$ and therefore the probability that an arbitrary bit is set is $p_{set} = 1-(1-1/m)^{kn}$. Thus, for *k* hashes for a test object not mapped in the Bloom filter, the probability of false positive is (we call this the "classic formula"),

$$p_{false} = p_{set}^k = \left(1-\left(1-\frac{1}{m}\right)^{kn}\right)^k . \qquad (1)$$

The value of *k* is usually chosen to minimize p_{false} and is a function of m/n. The false positive rate from eq. (1) can be approximated as $p_{false} = \left(1 - e^{-kn/m}\right)^k$ from which the value of *k* that minimizes p_{false} is found to be one of the two integer values closest to $(m/n)\ln(2)$ [5].

The classic formula of eq. (1) is wrong. This can be made clear if we look at the probability of false positives. For a Bloom filter with *s* bits set, $p_{set} = s/m$ and

$$p_{false} = \left(\frac{s}{m}\right)^k . \tag{2}$$

Bose et al. [2] is the first published account of the classic formula being "not quite correct". In Bose et al. a new formula is given by means of conditioning on *s* and then summing over all possible numbers of set bits. In this paper, we model and derive in detail the false positive rate of a Bloom filter and explore stable methods to compute the false positive probability value for various *m*, *n*, and *k* values thus expanding on the work by Bose et al. The key contributions of this paper are:

1. The detailed derivation of a correct formula for the mean false positive rate of a Bloom filter with a numerically stable method for solving it. Our derivation is different than that of Bose et al.
2. A presentation of a straight-forward and simple proof that the actual false positive rate is always greater than that predicted by the classic formula.
3. Numerical results showing the relative error of the classic versus new expression for false positive rate.

In addition, we show that the analysis and classic formula credited to Bloom is different from the one derived by Bloom in his paper [1]. Second, we note that Bose et al. [2] showed that the classic well-known formula for the probability of false positives is incorrect. We note that both this classic formula and Bloom's original result are incorrect, and we further correct and expand the analysis of Bose et al.

2. FALSE POSITIVE RATE OF A BLOOM FILTER

The false positive rate of a Bloom filter can be considered from an experimental point of view. First, consider an experiment that generates *numTrials* Bloom filters (for $n \cdot numTrials$ unique objects) and determines the mean number of bits set in the *numTrials* generated Bloom filters. The probability of false positive is then calculated using eq. (2). The false positive rate from this experiment matches that of eq. (1) for a given *m*, *n*, and *k* – that is, it

matches the classic formula. Now, consider a similar experiment where the false positive rate of each Bloom filter instance is individually computed used eq. (2). For the first experiment, the false positive rate is computed as,

$$pFalse = \left(\frac{s_1 + \cdots + s_{numTrials}}{m \cdot numTrials} \right)^k , \tag{3}$$

where s_i is the number of bits set in the Bloom filter in iteration i ($i = 1, 2, \ldots, numTrials$). For the second experiment the mean false positive rate is computed as

$$pFalse = \frac{\left(\frac{s_1}{m} \right)^k + \cdots + \left(\frac{s_{numTrials}}{m} \right)^k}{numTrials} . \tag{4}$$

In practice, it is the mean value from the second experiment that will be seen in real applications of Bloom filters since it represents the mean false positive rate of multiple instances of a Bloom filter. The true probability of false positive is always at least as large as the old incorrect probability, this follows from

$$\frac{s_1^k + \cdots + s_N^k}{N} \geq \left(\frac{s_1 + \cdots + s_N}{N} \right)^k \tag{5}$$

which follows a well known Holder's inequality [6] (for any integer $N \geq 1$, any k, such that $1 \leq k < \infty$, and any non-negative $s_1, \ldots, s_N$). The proof of this is as follows:

Lemma. For any integer $N \geq 1$, any k, such that $1 \leq k < \infty$, and any non-negative $s_1, \ldots, s_N$, the following holds:

$$\frac{s_1^k + \cdots + s_N^k}{N} \geq \left(\frac{s_1 + \cdots + s_N}{N} \right)^k . \tag{6}$$

The proof is based on the Holder's inequality [6]. Note that k does not have to be an integer.

Proof of lemma. For $k = 1$ this is trivial. Assume now that $k > 1$ we will select $q = k/(k-1)$, so that $1/k + 1/q = 1$ and choose $y_1 = y_2 = \ldots = y_N = 1/N$. Then applying Holder's inequality with k and q as chosen to the sequences s_i and y_i we obtain,

$$\frac{s_1}{N} + \cdots + \frac{s_N}{N} \leq \left(\sum_{i=1}^N s_i^k \right)^{1/k} \left[N \left(\frac{1}{N} \right)^q \right]^{1/q} = \left(\sum_{i=1}^N s_i^k \right)^{1/k} N^{\frac{1}{q}-1} = \left(\sum_{i=1}^N s_i^k \right)^{1/k} N^{-\frac{1}{k}} = \frac{\left(\sum_{i=1}^N s_i^k \right)^{1/k}}{N^{1/k}} = \left(\frac{s_1^k + \cdots + s_N^k}{N} \right)^{1/k} . \tag{7}$$

Taking both sides to the power of k, we complete the proof. **End of proof.**

Thus, the classic formula of eq. (1) predicts too small of a value for the false positive rate of a Bloom filter. In a real application, this is a significant error as it can lead to using a too small Bloom filter in the terms of m for a given

(targeted) false positive rate. In addition, the error in the predicted false positive rate can also result in using an incorrect value of *k*.

We now derive an expression for the probability of false positive of a Bloom filter that corresponds to the mean false positive rate from multiple instances of a Bloom filter (this matches the results of the second experiment described above). We will use a balls-and-bins construct as a model for a Bloom filter. Let there be *N* balls and *M* bins. The *M* bins represent the *m* bits of a Bloom filter. The *N* balls represent the $k \cdot n$ hash values mapped into a Bloom filter. When the *N* balls are randomly thrown into the *M* bins we are interested in solving for the probability $\mathrm{P}(N, M, K)$ of exactly *K* bins ($K = 1, 2, \ldots, M$) having one or more balls in it. This mimics the number of bits set in a Bloom filter. The *N* balls being in exactly *K* bins can happen either when the first $N-1$ balls were in exactly *K* bins and the *N*th ball fell into one of these *K* bins, or when the first $N-1$ balls were in exactly $K-1$ bins and the *N*th ball fell into one of the other $M-K+1$ bins. Hence,

$$\mathrm{P}(N, M, K) = \mathrm{P}(N-1, M, K)\left(\frac{K}{M}\right) + \mathrm{P}(N-1, M, K-1)\left(\frac{M-K+1}{M}\right). \tag{8}$$

Eq. (8) is a recursive expression for the probability of *K* bins being occupied.

Theorem. Let $\mathrm{P}(N, M, K)$ be the probability of having exactly *K* bins non-empty when *N* balls are thrown independently at random in *M* bins. This probability can be expressed as

$$\mathrm{P}(N, M, K) = \left(\frac{1}{M^{N-1}} \cdot \frac{(M-1)!}{(M-K)!}\right) \sum_{j=1}^{K} \left(\frac{(-1)^{K-j} \cdot j^{N}}{j!(K-j)!}\right), \tag{9}$$

which can be rewritten as

$$P(N, M, K) = \sum_{j=1}^{K} (-1)^{K-j} \left(\frac{j}{M}\right)^{N} \binom{M}{K} \binom{K}{j}. \tag{10}$$

Proof of theorem. We prove eq. (10) by induction where (I) is the base case and (II) is the induction step.

(I). $N = 1$. We need to prove the result for all values of *M* and *K* $(M \geq K)$. Note that $\mathrm{P}(1, M, 1)$ is 1, so the result is trivially true for $K = 1$. If $K > 1$, then $\mathrm{P}(1, M, K)$ is 0. Let us demonstrate that the right hand side of (10) also evaluates to 0. We have

$$\sum_{j=1}^{K} (-1)^{K-j} \left(\frac{j}{M}\right) \binom{M}{K} \binom{K}{j} = \binom{M}{K} \frac{1}{M} \sum_{j=1}^{K} (-1)^{K-j} \frac{K!}{j!(K-j)!} j = \binom{M}{K} \frac{K}{M} \sum_{j=1}^{K} (-1)^{(K-1)-(j-1)} \frac{(K-1)!}{(j-1)!(K-j)!}$$

(here we set $n = K-1$)

$$= \binom{M}{K}\frac{K}{M}\sum_{l=0}^{n}(-1)^{n-l}\binom{n}{l} = \binom{M}{K}\frac{K}{M}(1-1)^n = 0 \tag{11}$$

since $n \geq 1$.

(II) Let us now assume that eq. (10) holds for $N-1$ and all *M* and *K* such that $M \geq K$ and prove it for *N*. We will utilize the recursive formula for $\mathrm{P}(N,M,K)$ given in eq. (8). We will compute

$$P(N-1,M,K)\frac{K}{M} + P(N-1,M,K-1)\frac{M-K+1}{M}$$

$$= \sum_{j=1}^{K}(-1)^{K-j}\left(\frac{j}{M}\right)^{N-1}\binom{M}{K}\binom{K}{j}\frac{K}{M} + \sum_{j=1}^{K-1}(-1)^{K-1-j}\left(\frac{j}{M}\right)^{N-1}\binom{M}{K-1}\binom{K-1}{j}\frac{M-K+1}{M}$$

$$= \left(\frac{K}{M}\right)^{N-1}\binom{M}{K}\frac{K}{M} + \sum_{j=1}^{K-1}(-1)^{K-j}\left(\frac{j}{M}\right)^{N-1}\binom{M}{K}\binom{K}{j}\frac{K}{M} + \sum_{j=1}^{K-1}(-1)^{K-1-j}\left(\frac{j}{M}\right)^{N-1}\binom{M}{K-1}\binom{K-1}{j}\frac{M-K+1}{M} \tag{12}$$

In the above we have separated the term with $j = K$ in the first sum. Continuing the above and substituting

$$\binom{K-1}{j} + \binom{K-1}{j-1} = \binom{K}{j}$$

thus breaking the first sum into two parts, we have,

$$= \left(\frac{K}{M}\right)^{N-1}\binom{M}{K}\frac{K}{M} + \sum_{j=1}^{K-1}(-1)^{K-j}\left(\frac{j}{M}\right)^{N-1}\binom{M}{K}\binom{K-1}{j}\frac{K}{M} + \sum_{j=1}^{K-1}(-1)^{K-j}\left(\frac{j}{M}\right)^{N-1}\binom{M}{K}\binom{K-1}{j-1}\frac{K}{M}$$

$$+ \sum_{j=1}^{K-1}(-1)^{K-1-j}\left(\frac{j}{M}\right)^{N-1}\binom{M}{K-1}\binom{K-1}{j}\frac{M-K+1}{M} = A + B + C + D \tag{13}$$

where *A*, *B*, *C*, and *D* are intermediate terms considered below. Now,

$$C = \sum_{j=1}^{K-1}(-1)^{K-j}\left(\frac{j}{M}\right)^{N-1}\binom{M}{K}\binom{K-1}{j-1}\frac{K}{M} = \sum_{j=1}^{K-1}(-1)^{K-j}\left(\frac{j}{M}\right)^{N}\binom{M}{K}\frac{(K-1)!}{(j-1)!(K-j)!}\frac{M}{j}\frac{K}{M}$$

$$= \sum_{j=1}^{K-1}(-1)^{K-j}\left(\frac{j}{M}\right)^{N}\binom{M}{K}\binom{K}{j}. \tag{14}$$

So

$$A + C = \left(\frac{K}{M}\right)^{N-1}\binom{M}{K}\frac{K}{M} + \sum_{j=1}^{K-1}(-1)^{K-j}\left(\frac{j}{M}\right)^{N}\binom{M}{K}\binom{K}{j} = \sum_{j=1}^{K}(-1)^{K-j}\left(\frac{j}{M}\right)^{N}\binom{M}{K}\binom{K}{j}$$

which is our expected formula for $\mathrm{P}(N,M,K)$. So, to prove that $A+B+C+D$ is equal to $\mathrm{P}(N,M,m)$, we only need to show that $B+D=0$. We have

$$B = \sum_{j=1}^{K-1} (-1)^{K-j} \frac{j^{N-1}}{M^{N-1}} \frac{M!}{K!(M-K)!} \frac{(K-1)!}{j!(K-1-j)!} \frac{K}{M} = \sum_{j=1}^{K-1} (-1)^{K-j} \frac{j^{N-1}}{M^{N}} \frac{M!}{(M-K)!\, j!(K-1-j)!}. \quad (15)$$

And,

$$D = \sum_{j=1}^{K-1} (-1)^{K-1-j} \frac{j^{N-1}}{M^{N-1}} \frac{M!}{(K-1)!(M-K+1)!} \frac{(K-1)!}{j!(K-1-j)!} \frac{M-K+1}{M}$$

$$= \sum_{j=1}^{K-1} (-1)^{K-1-j} \frac{j^{N-1}}{M^{N}} \frac{M!}{(M-K)!\, j!(K-1-j)!}. \quad (16)$$

Eqs. (15) and (16) show that *B* and *D* are the sums of the same summands, but each summand is entering the sum with an opposite sign, that is, where it is $(-1)^{K-j}$ for *B*, it is $(-1)^{K-1-j}$ for *D*. Hence, $B + D = 0$, which completes the induction proof of the correctness of eq. (10). **End of Proof.**

Having $\mathrm{P}(N, M, K)$ we can now write the probability of false positive for a Bloom filter as the total probability,

$$p_{false} = \sum_{i=1}^{m} \left(\mathrm{P}(k \cdot n, m, i) \left(\frac{i}{m} \right)^k \right) \quad (17)$$

where $(i/m)^k$ is the false positive rate for the Bloom filter with *i* bits set from eq. (2) and $\mathrm{P}(k \cdot n, m, i)$ is the probability of this occurrence. Combining eq. (10) and eq. (17) we get,

$$p_{false} = \frac{m!}{m^{k(n+1)}} \sum_{i=1}^{m} \sum_{j=1}^{i} (-1)^{i-j} \frac{j^{kn} i^k}{(m-i)!\, j!(i-j)!} \quad (18)$$

The direct relationship of eq. (18) with the result in Bose et al. [2] is described later in this paper.

3. NUMERICALLY COMPUTING THE FALSE POSITIVE RATE

Directly computing p_{false} using eq. (18) is very difficult and is generally unstable for large *m*, *n*, and *k*. The factorial and exponent terms within the summations become very large and overflow for even modest values of *m*. However, a direct computation of the recursive expression in eq. (8) and then using eq. (17) to compute p_{false} is stable. In eq. (17) both terms within the sum are always non-negative, and thus the calculation of the summation is stable in that it will not overflow. Eq. (8) can be computed recursively with $\mathrm{P}(1, M, 1) = 1$, $\mathrm{P}(a, M, b) = 0$ if $a < \mathrm{b}$, and $\mathrm{P}(a, M, 0) = 0$. The recursive algorithm can be "unraveled" into an iterative table-based algorithm. Such a table-based method may be faster to execute when implemented in a stack-oriented machine. Figure 1 shows an iterative

table-based algorithm where the recursive loop is "unraveled" into two tables (*table*1 and *table*2, both of size *N* and indexed as *table*1[1], *table*1[2], …, *table*1[*N*]). If the algorithm for $\mathrm{P}(N, M, K)$ in Figure 1 is called iteratively (that is, with $K = 1, 2, \ldots, M$) as is the case when implementing eq. (17), then the tables do not need to be recomputed for each iteration and can be stored permanently (that is, as "static" in a C language implementation) between iterations once initially created for $K = 1$. The implementation of the table-based method is very straightforward where the algorithm in Figure 1 can be directly translated to a C language program.

Fig. 2 shows the relative error between the classic and new formulas as a function of *m* for four m/n values. As *m* increases and as m/n decreases, the relative error decreases. Note that for small values of *m* (such as $m = 32$ and $m = 64$) the relative error can be very significant. This implies that the performance (relative to false positive rate) of applications that use small Bloom filters would not be correctly predicted using the classic formula.

4. RELATED WORK

The Bloom filter was first proposed as a "Method 2" for hash coding with allowable errors by Bloom in 1970 [1]. The original analysis by Bloom represented the expected proportion of bits set to 0 after *n* objects have been mapped-in as $(1 - k/m)^n$, which is incorrect and *not* the same as $(1 - 1/m)^{kn}$ in the classic analysis that is usually attributed to Bloom. Bloom's expression for probability of false positive was then,

$$p_{false} = \left(1 - \left(1 - \frac{k}{m}\right)^n\right)^k . \tag{19}$$

For $m >> k$ this expression is numerically close to that of eq. (1). However, for a smaller *m* Bloom's original expression is significantly "off". The analysis usually attributed to Bloom may have first appeared in a short paper by Mullin in 1983 where eq. (1) was developed [7]. Thus, the attribution of the classic analysis to Bloom (an attribution made in countless papers) is incorrect; the concept of the Bloom filter is, however, correctly attributed to Bloom.

A very recent work by Bose et al. [2] is notably the first work to demonstrate that while the classic derivation of p_{false} corrects an error in Bloom's original analysis, as explained above in this Section, it is also incorrect. Bose et al. argue that the assumption that the probability of an arbitrary bit being set in a Bloom filter is independent is

incorrect. Bose et al. use an example of a 2-bit Bloom filter to demonstrate that the false positive rate predicted by eq. (1) is an under-estimate. Bose et al. develop an expression for p_{false} based on a balls-and-bins model using two colors of balls. The expression derived is,

$$p_{false} = \frac{1}{m^{k(n+1)}} \sum_{i=1}^{m} i^k i! \binom{m}{i} \left\{ {k \cdot n \atop i} \right\} \tag{20}$$

where,

$$\left\{ {k \cdot n \atop i} \right\} = \frac{1}{i!} \sum_{j=0}^{i} (-1)^{i-j} \binom{i}{j} j^{k \cdot n} \tag{21}$$

is the Stirling number of the second kind (in Bose et al. [2] this expression is in error; it contains $(-1)^j$ and not $(-1)^{i-j}$). The difference is very significant as it will lead to a completely erroneous result. Our expression for p_{false} in eq. (18) reduces to exactly eq. (21). In contrast to Bose et al., our work develops an experimental basis for understanding the false positive rate and derives a computationally solvable formula (even for large *m*, *n*, and *k*) based on a recursive expression. Bose et al. does not present any means of computing eq. (21). Bose et al. does, however, derive an upper bound for p_{false} that applies only to large *m*, *n*, and *k* that satisfy some additional constraints.

Most applications use large Bloom filter of many megabytes in length. However, there are applications that use small Bloom filters of few bytes in length. Predicting the performance of these applications using the classic formula will often result in significant errors. Mullin has investigated the use of small Bloom filters to speed-up string search [8]. Mullin employed a first algorithm to build an index of Bloom filters of one per document and a second algorithm to use the resulting Bloom filter index for searching. A specific example was developed and demonstrated. The expected (predicted) performance of the new algorithm was not studied by Mullin. Whitaker and Wetherall used small Bloom filters embedded in packets to detect and halt forwarding loops in networks [9]. In their scheme – named Icarus – an extra field is added to a packet header that consists of a small Bloom filter which registers the network interfaces the packet has been through by setting pre-determined random bits in the Bloom filter header. An analysis of expected hop count before a false positive was given as a function of Bloom filter size. The analysis was based on the classic formula for false positive rate. Given the use of small Bloom filters in Icarus, this analysis is likely incorrect.

5. SUMMARY

As Bloom filters – both large and small in size – become more widely used in a broad range of applications a complete and correct understanding of their false positive rate is needed. In this paper, we have shown that the false positive rate can be viewed experimentally in two ways – as a function of the average of the number of bits set in a Bloom filter over many instances and as an average of the individual false positive rate of each Bloom filter of many instances. The classic analysis of the Bloom filter results in the "classic formula" – eq. (1) – that correctly predicts the false positive rate of the first case, but not that of the second case. It is the second case that represents actual application performance (that is, of an application that uses Bloom filters) over many trials. We derive a computable recursive formula – eq. (8) – for the second case. A closed form of this formula is eq. (18). We also clearly show that the classic formula always predicts a too low false positive rate compared to the correct new formula in eq. (10). For a small Bloom filter (for example, of size 32 bits) the relative error in prediction of false positive rate can be off by several factors. For larger Bloom filters the relative error decreases.

FUNDING

This material is based on work supported by the National Science Foundation under CNS-0520081.

REFERENCES

[1] Bloom, B. (1970) Space/time tradeoffs in hash coding with allowable errors. *Communications of the ACM*, **13**, 7, 422-426.

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

[3] Broder A. and Mitzenmacher, M. (2004) Network applications of Bloom filters: a survey. *Internet Mathematics*, **1**, 4, 485-509.

[4] Chang, F., Dean, J., Ghemawat, S., Hsieh, W., Wallach, D., Burrows, M., Chandra, T., Fikes, A., and Gruber, R. (2006) Bigtable: a distributed storage system for structured data. *Proceedings of the 7th USENIX Symposium on Operating Systems Design and Implementation*, Seattle, WA, 6-8 November, pp. 205-218, USENIX Association, Berkley, CA.

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

[6] Hardy, G., Littlewood, J, and Pólya, G. (1934), Inequalities, Cambridge Univ. Press.

[7] Mullin, J. (1983) A second look at Bloom filters. *Communications of the ACM*, **26**, 8, 570-571.

[8] Mullin, J. (1987) Accessing textual documents using compressed indexes of arrays of small Bloom filters. *The Computer Journal*, **30**, 4, 343-348.

[9] Whitaker, A. and Wetherall, D. (2002) Forwarding without loops in Icarus. *Proceedings of the 5th IEEE Conference on Open Architectures and Network Programming*, New York City, 28-29 June, pp. 63-75, IEEE Computer Society, Los Alamitos, CA.

Figure

```
P(N, M, K)
1.  clear table1
2.  clear table2
3.  table1[1] ← 1
4.  for i ← 1 to K do
5.    for j ← i to N do
6.      if (i = 1) and (j = 1) continue
7.      table1[j] ← table1[j – 1] · (i / M) + table2[i – 1] · ((M – i + 1) / M)
8.    p ← table1[N]
9.    copy table1 to table2
10.   clear table1
11. return(p)
```

FIGURE 1. Algorithm for table-based computation of $\mathrm{P}(N, M, K)$

Figure

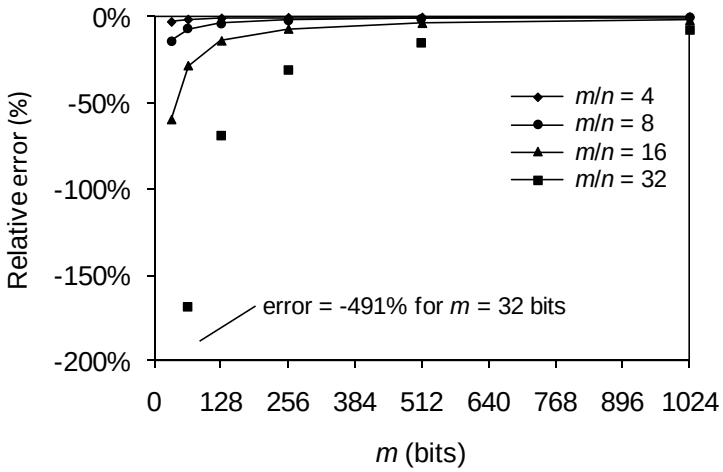

FIGURE 2. Relative error of classic versus new formula