

Bloom Filters in Probabilistic Verification

Peter C. Dillinger and Panagiotis Manolios

Georgia Institute of Technology
College of Computing, CERCS
801 Atlantic Drive
Atlanta, GA 30332-0280
{peterd,manolios}@cc.gatech.edu

Abstract. Probabilistic techniques for verification of finite-state transition systems offer huge memory savings over deterministic techniques. The two leading probabilistic schemes are hash compaction and the bitstate method, which stores states in a Bloom filter. Bloom filters have been criticized for being slow, inaccurate, and memory-inefficient, but in this paper, we show how to obtain Bloom filters that are simultaneously fast, accurate, memory-efficient, scalable, and flexible. The idea is that we can introduce large dependences among the hash functions of a Bloom filter with almost no observable effect on accuracy, and because computation of independent hash functions was the dominant computational cost of accurate Bloom filters and model checkers based on them, our savings are tremendous. We present a mathematical analysis of Bloom filters in verification in unprecedented detail, which enables us to give a fresh comparison between hash compaction and Bloom filters. Finally, we validate our work and analyses with extensive testing using 3SPIN, a model checker we developed by extending SPIN.

1 Introduction

Despite its simplicity, explicit-state model checking has proved to be an effective verification technique and has led to numerous tools, including SPIN [14], Murø [18], TLC [23], Java Pathfinder [20], etc. The *state explosion problem* is especially acute in explicit-state model checking because the amount of memory required depends linearly on the number of reachable states, which is often too large to enumerate in main memory. Disk can be utilized intelligently [17, 23], but such algorithms will probably continue to be outperformed by algorithms that take advantage of the fast random access time of main memory. Storing states more compactly in memory, therefore, is very desirable, and because of the huge memory savings available, storing states in a probabilistic data structure has become a popular approach and the topic of significant research.

Virtually all of the proposed probabilistic verification approaches utilize one of two data structures: a Bloom filter [14] or a compacted hash table [18]. The Bloom filter, dating back to 1970 [1], is the data structure underlying “supertrace” [12], “multihashing” [21], and “bitstate hashing” [13]. Compacted hash tables are utilized by “hashcompact” [21] and the first version of “hash compaction” [18], but the technique was not perfected until [19].

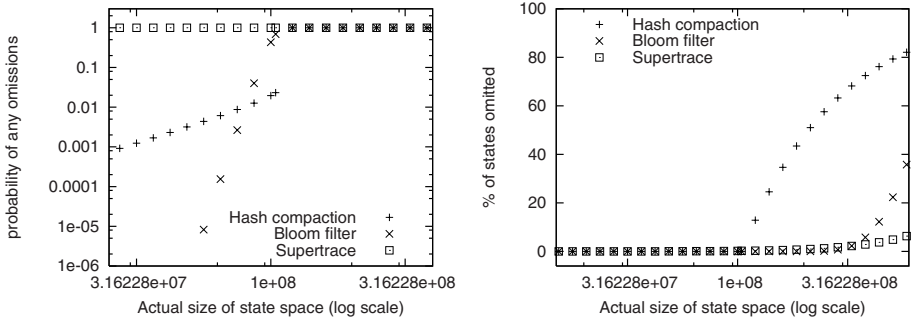


Fig. 1. These graphs show the accuracy of three probabilistic verification techniques/configurations for various state space sizes. In both graphs, lower is better. The data points for “Hash compaction” and “Bloom filter” are obtained with data structures optimized for a state space size of 10^8 , using 400MB of memory. The graphs show the accuracy of the data structures as the size of the state space varies. The left graph shows the probability that even a single omission occurs, while the right graphs shows the expected percentage of states omitted. The “Bloom filter” ($k = 24$) and “Supertrace” ($k = 2$) accuracies are computed using the analyses in Section 3. To compute the “Hash compaction” values we need to know the number of expected collisions in the table for the verifier run represented by each data point [18]. We determine this experimentally by counting the number of collisions in 3SPIN’s ordered, compacted table implementation, which in this case uses 32 bits per state and has a maximum visitable size of 104857589.

The literature contains explanations on both sides of the Bloom filter vs. hash compaction debate as to why each data structure is the best. We have found that neither is best, but that there are scenarios under which each is the best choice. More specifically, we identify three probabilistic verification techniques (based on the two data structures), each of which we believe is the best choice for some level of knowledge of the state space size.

When the state space size is completely unknown, Holzmann’s supertrace method, which uses a Bloom filter with two hash functions, is the best choice because of its high expected coverage over a wide range of state space sizes (Figure 1, right graph). Supertrace is fast but starts omitting states for state spaces much smaller than other approaches (see Figure 1, left graph). Nevertheless, we can estimate actual state space sizes rather accurately with supertrace (see Section 3.3).

When we know the size of the state space rather accurately – if we have an estimate that we are reasonably certain is within about 15% of the actual size – the best choice is a compacted hash table configured with slightly more cells than the maximum estimated state space size. This technique gives exceptionally high accuracy within this narrow range of state space sizes (Figure 1, left graph), and its speed is similar to supertrace’s. If, however, the data structure overflows (right graph) or is underpopulated (left graph), a Bloom filter configured for the same estimate would have been a better choice.

When we have a rough estimate of the state space size, a Bloom filter configured for that estimate can tolerate much more deviation from the estimate than hash compaction can, and is likely to be much more accurate than supertrace. Such a configuration remains a respectable choice even if the estimate is off by a factor of five or more.