Coverage Estimation in Model Checking with Bitstate Hashing

Satoshi Ikeda, Masahiro Jibiki, and Yasushi Kuno, *Member*, *IEEE Computer Society*

Abstract—Explicit-state model checking which is conducted by state space search has difficulty in exploring satisfactory state space because of its memory requirements. Though bitstate hashing achieves memory efficiency, it cannot guarantee complete verification. Thus, it is desirable to provide a reliability indicator such as a coverage estimate. However, the existing approaches for coverage estimation are not very accurate when a verification run covers a small portion of state space. This mainly stems from the lack of information that reflects characteristics of models. Therefore, we propose coverage estimation methods using a growth curve that approximates an increase in reached states by enlarging a bloom filter. Our approaches improve estimation accuracy by leveraging the statistics from multiple verification runs. Coverage is estimated by fitting the growth curve to these statistics. Experimental results confirm the validity of the proposed growth curve and the applicability of our approaches to practical models. In fact, for practical models, our approaches outperformed the conventional ones when the actual coverage is relatively low.

Index Terms—Coverage estimation, model checking, bitstate hashing

1 Introduction

Model checking [1] has been widely used for verification of hardware design and communication protocols in recent years. In explicit-state model checking tools such as SPIN [2], Murφ [3], and Java PathFinder [4], verification is performed by exhaustive state space search, which explores counterexamples that violate formal specifications defined by temporal logic. One of the serious problems model checking faces is the state explosion problem. Because memory requirements for exhaustive state space search depend on the number of reachable states of a graph, it is infeasible to verify huge scale models. To deal with the state explosion problem, several probabilistic approaches have been proposed [5], [6], [7].

Bitstate hashing [6], proposed by Holzmann, is such a probabilistic approach designed to reduce the memory requirements. For memory efficiency, it uses a data structure called a bloom filter [8] to store reached states in place of a hash table. A bloom filter is probabilistic in the sense that there is a possibility of judging an unreached state as already reached. Therefore, model checking by bitstate hashing does not guarantee complete verification. Accordingly, in probabilistic approaches, it is desirable to exhibit a reliability indicator to users. One representative of such indicators is a coverage estimate. This is useful for sequential hashing [6], [9], which improves total coverage by repetitive verification with different hash functions or with randomization to decide how many searches should be performed.

However, coverage estimation for bitstate hashing is difficult, especially when actual coverage is low. The difficulty is mainly caused by the lack of information on a state space. From verification results, a few pieces of statistical information (such as the number of reached states, the number of hash conflicts, and the size of a bloom filter) are available for estimation. Thus, it is significantly difficult to achieve high accuracy since coverage needs to be estimated from this limited amount of information.

In this paper, we propose coverage estimation methods using a growth curve that approximates an increase in reached states by enlarging a bloom filter. The growth curve models a proportional increase in reached states with a small bloom filter and saturation of the increase with a large one. To improve the accuracy of estimation, our approach leverages statistical information from multiple verification runs that are performed with bloom filters of different sizes. By fitting the growth curve to results from multiple runs, the total number of reachable states, an unknown parameter of the curve, is estimated.

The rest of this paper is structured as follows: Section 2 reviews bitstate hashing and reliability indicators for probabilistic verification approaches. In Section 3, we discuss the characteristics of bitstate hashing and introduce a growth curve that describes the increase in reached states by enlarging a bloom filter. Then, we propose our coverage estimation approaches by curve fitting. Section 4 details the experimental results. In Section 4.1, we confirm whether the growth curve describes the increase in reached states suitably for practical models. In Section 4.2, we measure the accuracy of our approaches and compare it to the conventional approaches. In Section 5, we discuss the applicability of our approach to practical models. Finally, the conclusion ends the paper.

• S. Ikeda is with the Cloud System Research Laboratories, NEC Corporation, 1753 Shimonumabe, Nakahara, Kawasaki, Kanagawa 211-8666, Japan. E-mail: s-ikeda@fd.jp.nec.com.
• M. Jibiki is with the National Institute of Information and Communications Technology, 4-2-1 Nukuikitamachi, Koganei, Tokyo 184-8795, Japan. E-mail: jibiki@nict.go.jp.
• Y. Kuno is with the Faculty of Business Sciences, University of Tsukuba, 3-29-1 Otsuka, Bunkyo, Tokyo 112-0012, Japan. E-mail: kuno@gssm.otsuka.tsukuba.ac.jp.

Manuscript received 19 May 2011; revised 24 Mar. 2012; accepted 2 June 2012; published online 21 June 2012.
Recommended for acceptance by P. Inverardi.
For information on obtaining reprints of this article, please send e-mail to: tse@computer.org, and reference IEEECS Log Number TSE-2011-05-0154.
Digital Object Identifier no. 10.1109/TSE.2012.44.

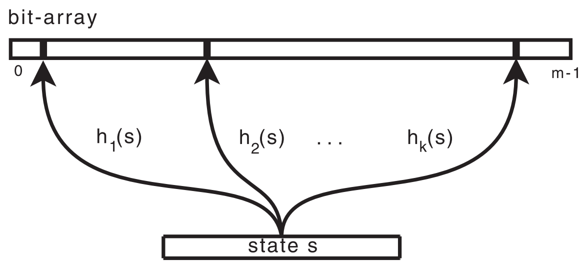

Fig. 1. Bloom filter.

2 Background

For bitstate hashing, a reliability indicator such as a coverage estimate is useful for users to decide their next action. However, the lack of statistical information on verification makes it difficult to provide a reliability indicator with high accuracy. Although several reliability indicators for probabilistic approaches including bitstate hashing have been proposed, these indicators are not accurate enough for a verification run that covers a small portion of state space or inapplicable to bitstate hashing.

2.1 Bitstate Hashing

Bitstate hashing, one of the probabilistic approaches, was introduced by Holzmann to reduce memory requirements to store reached states. In model checking, reachable states increase explosively as a target system becomes complex. Especially in explicit-state model checking, since an exhaustive search has to store all reached states, the requirement is proportional to the number of reachable states. Thus, it is difficult to verify complex systems by model checking. Bitstate hashing improves memory efficiency by managing reached states with a bloom filter. However, completeness of verification is sacrificed.

A bloom filter is a data structure to test whether an element is a member of a set and consists of a bit array of m bits. There are two basic operations: *add* and *query*. These operations are realized with distinct k hash functions that have ranges from 0 to $m-1$. The k hash values of an element are used as indices to the bit array, as seen in Fig. 1. To *add* an element to a bloom filter, all bits corresponding to k hash values of the element are set to 1. To *query* whether an element is a member, all bits at which k hash values point are checked. The element is judged to be a member only if all the bits are 1. A bloom filter is probabilistic because it can cause a false positive. That is, a bloom filter misjudges a nonmember as a member when its hash values collide.

Because of the probabilistic behavior of a bloom filter, bitstate hashing is generally not complete. If a false positive occurs, the collided state is never visited, and its successor states might not be visited. Thus, some reachable states are omitted from exploration. As a result, bitstate hashing cannot guarantee exhaustive verification.

It is therefore highly desirable to provide a reliability indicator which represents the achievement of verification because a reliability indicator helps users decide their next action at the end of a verification run. For instance, if it were found that only a tiny portion of reachable states is explored, users would try to reduce states by abstracting models. If more than a quarter of states were covered, they might choose to perform repetitive verification runs by sequential hashing [6], [9] or by swarm verification [10].

However, it is difficult to achieve a reliability indicator with high accuracy because of the lack of statistical information. The statistics of bitstate hashing obtained from verification results are the numbers of reached states, hash conflicts, and bits in a bloom filter. The number of reached states (hash conflicts) can be measured simply by counting the number of additions (conflicts) of states in a bloom filter. Reliability indicators for bitstate hashing have to be calculated with these limited statistics.

2.2 Reliability Indicators

Including a coverage estimate, several reliability indicators for probabilistic verification have been proposed. We review such indicators for probabilistic verification approaches: bitstate hashing and random walk state space exploration.

2.2.1 Bitstate Hashing

For bitstate hashing, coverage estimates [2], [11] and hash factor [6] are common reliability indicators. These indicators are calculated from the number of reached states N and the size of a bloom filter m. Though the existing reliability indicators are calculated from these statistics, the difference originating from graph structures of models is not considered at all.

SPIN version 4 provides a coverage estimate at the end of verification [2]. When a run has stored N states in a hash array of m bits, the coverage estimate Cov_{Stern} is computed using the next equation by Stern:

$$Cov_{Stern} = N\frac{\ln(1-1/m)}{\ln(1-N/m)}. \tag{1}$$

This equation takes neither the number of hash functions nor the difference among models into account. The accuracy is not very high when the actual coverage is low.

Another coverage estimation, which was proposed by Dillinger and Manolios [11], uses the expectation of the number of state omissions $E(o)$. A state omission represents a state being excluded from a verification run by false positives. The expectation of state omissions $E(o)$ is defined as follows:

$$E(o) \approx 2\sum_{i=0}^{N-1}\{1-(1-1/m)^{ik}\}^k,$$

where k is the number of hash functions. With $E(o)$, the coverage estimate $Cov_{Dillinger}$ is calculated as

$$Cov_{Dillinger} = \frac{N-E(o)}{N}. \tag{2}$$

Though (2) leverages the number of hash functions, the difference in omission rates among models is ignored. Therefore, if the number of actual omissions significantly differs from that expected, an estimation error becomes large.

Hash factor [6], another reliability indicator that is used in SPIN, is defined by m/N. It represents how much memory per state is available. Though it does not directly represent a coverage estimate, hash factor is useful to determine whether the state coverage accomplished by a single run is sufficient, that is, whether the results are reliable. However, in the situation in which the actual coverage is low and one tries to improve the coverage by multiple verification runs, hash factor is not appropriate as a reliability indicator for whole verification.

2.2.2 *Random Walk Exploration*

Other than probabilistic model checking such as bitstate hashing, random walk state space exploration is another approach for memory-efficient verification [12], [13], [14]. Also, in random walk exploration reliability indicators should preferably be obtained.

Tronci et al. [15] used random sampling to estimate coverage. In their method, a coverage estimate is defined as the proportion of reached states among randomly sampled states. When trying to apply their approach to bitstate hashing, we have to sample states uniformly from reachable states. However, it is unclear whether a state is reachable at the start of verification. In addition, a large number of sample states, which increase memory requirements, will be necessary for high accuracy. For this reason, it is considered difficult to obtain an estimate with high accuracy.

Lurch [12] uses *saturation effects* [16] to determine when to stop a random walk. Lurch stores hash values for each state that it finds in order to check whether the hash value is new. In other words, a bloom filter with a single hash function is used to check whether a state is already reached. When the percentage of new states reaches some saturation point (close to 0 percent), Lurch is assumed to be unlikely to find any more interesting information. Thus, Lurch does not search further.

Monte Carlo Estimation [14] by Grosu and Smolka indicates how many random walks should be executed to guarantee the probability that an error found by further random walks is less than given δ. However, it needs to be given a lower bound of a predicted error rate ϵ. It is, in general, difficult to know such a lower bound ϵ in advance.

These indicators used in Lurch and Monte Carlo Estimation are utilized as a means to decide when to stop exploration. For both indicators, the parameters (the size of a bloom filter in Lurch and the pair of δ and ϵ in Monte Carlo Estimation) have to be chosen adequately, in tune with the characteristics of models. Moreover, these indicators are not applicable to bitstate hashing.

3 Coverage Estimation

Our strategy is to compensate for the lack of information by leveraging results from multiple verification runs. In the existing approaches, the number of reached states and the size of a bloom filter are the only statistics used for coverage estimation. The differences in the characteristics of models were not considered. Our strategy is based on an assumption that these statistics from several runs should reflect the characteristics of models such as a graph structure.

In our approaches, coverage is estimated by a curve fitting technique that uses the statistics from several runs. We use a growth curve that approximates the increase in reached states caused by enlarging a bloom filter for coverage estimation.

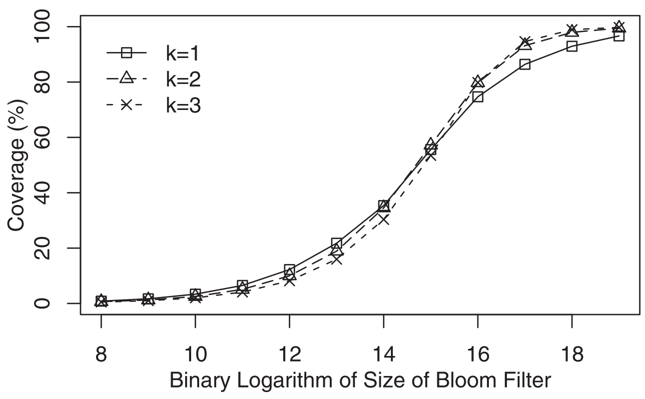

Fig. 2. Coverage achieved with bitstate hashing for a practical model.

3.1 Analysis

In this section, we analyze the increase in reached states by enlarging a bloom filter in bitstate hashing and introduce an approximated growth curve. The growth curve models the increase of reached states, which is proportional to the size of a bloom filter if it is small enough. In addition, the growth curve models the saturation of the increase in reached states with enlargement of a bloom filter.

Coverage achieved for some practical model is plotted in Fig. 2. The x-axis is a binary logarithm of the size of a bloom filter and the y-axis is the measured coverage. As shown in Fig. 2, coverage in bitstate hashing describes a growth curve when a bloom filter enlarges exponentially. The curves have horizontal asymptotes at coverage of 0 and 100 percent.

In the discussion below, let $2^t(=m)$ be the size of a bloom filter, k be the number of hash functions, N be the number of reached states in a single verification run, and M be the number of reachable states.

3.1.1 *Growth Rate of Reached States*

Obviously, reached states increase as a bloom filter enlarges. This is because a part of states that were regarded as reached by hash collisions turns out to be regarded as unreached. With a small enough bloom filter, the increase in reached states is proportional to the size of the bloom filter.

In the situation in which a bloom filter is small enough for a reachable state space, all bits are to be set to 1 at the ends of exploration. We define this condition as a saturated condition. A verification run is considered to end in a saturated condition if it uses a bloom filter smaller than that used in a saturated condition.

Let X represent a random variable describing the number of bits that are altered to 1 from 0 by a state transition. The average total number of on-bits after adding n states, B_n, is represented as

$$\begin{aligned} B_{n+1} &= B_n + E_{B_n}[X|X>0] \\ &= B_n + \sum_{j>0}^{k} j\,\frac{P_{B_n}(X=j)}{P_{B_n}(X>0)} \\ &= B_n + \frac{E_{B_n}[X]}{P_{B_n}(X>0)}, \end{aligned}$$

where $B_0 = 0$. $P_i(A)$ and $E_i[X]$ respectively represent the probability of an event A and the expectation of X when the

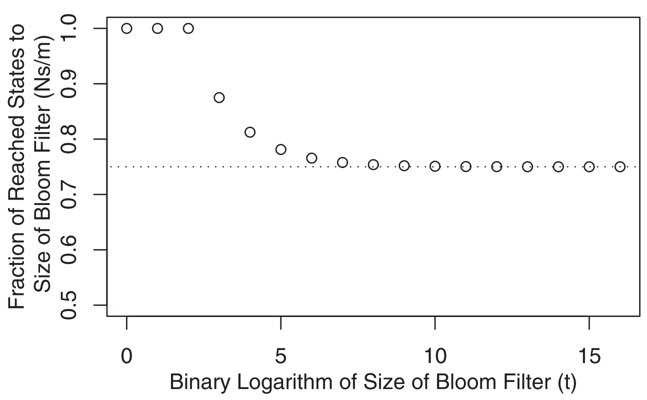

Fig. 3. Fraction of reached states to size of bloom filter with all bits on $(k = 2)$.

number of on-bits is i. $E_i[X|X > 0]$ represents the conditional expectation of X given the event $X > 0$. If a newly visited state is regarded as already visited, the state is not added to a bloom filter, and in this case $X = 0$ (in short, n is not incremented). This is the reason the conditional expectation given the event $X > 0$ is used.

In the case of $k = 1$, for example, B_n is simply defined as

$$B_{n+1} = B_n + 1,$$

because a single bit is altered to 1 when a new state is added. This implies that, at the end of exploration, the number of on-bits is equal to the number of reached states. Thus, in a saturated condition, reached states also halve if a bloom filter halves, that is, the number of reached states is proportional to the size of a bloom filter.

Next, let us consider the case of $k = 2$. When i bits are set to 1, the next addition of a state sets $E_i[X|X > 0] = (2m - 1)/(m + i)$ bits to 1 from 0 on average since

$$P_i(X = 1) = (m - i)(2i + 1)/m^2, P_i(X = 2) \\ = (m - i)(m - i - 1)/m^2,$$

and $P_i(X > 0) = 1 - (i/m)^2$. Then, we have

$$B_{n+1} = B_n + \frac{2m - 1}{m + B_n}. \tag{3}$$

The smallest N_S such that $B_{N_S} \geq m$ is considered as the number of reached states that is necessary to fill all m bits with 1. In other words, N_S is the average number of reached states in a saturated condition.

Fig. 3 plots the fraction N_S/m which is computed by (3). As seen in Fig. 3, N_S/m approaches about 0.75 as m increases. This indicates that, in saturated condition, reached states increases approximately in proportion to the size of a bloom filter m when m is larger than 2^{10}.

A similar discussion is applicable for k larger than 2. Thus, in a saturated condition, reached states can be generally regarded as proportional to the size of a bloom filter. Therefore, with a growth rate of reached states r, the relation between t and N in a saturated condition is approximated by the following differential equation:

$$\frac{dN}{dt} = rN.$$

The growth rate r is considered to be $\ln 2$ since the number of reached states N is proportional to the size of a bloom filter 2^t.

3.1.2 Inhibition Rate

The proportional increase never continues because the number of reachable states is finite. The increase slows down as the number of reached states comes close to the number of reachable states. We model this phenomenon by using an inhibition rate q, which describes the inhibition of the increase. An inhibition rate q is defined so that it reflects the differences in the number of hash functions.

By using a bloom filter large enough to cover a whole state space, reached states no longer increase because the number of reachable states is an upper bound of the number of reached states. Therefore, the differential dN/dt approaches 0 as a bloom filter enlarges. This tendency can be explained by the inhibition rate q as follows:

$$\frac{dN}{dt} = r(1 - q)N.$$

Moreover, the increase in the number of reached states N is considered to be more inhibited as N approaches the number of reachable states M. From this observation, the inhibition rate q needs to satisfy the conditions below:

1. q approaches 0 as t decreases,
2. q approaches 1 as t increases, and
3. q increases monotonically as N increases.

State coverage N/M, for instance, satisfies these conditions on q because a verification run with a larger bloom filter achieves better coverage in general.

However, since the number of hash functions varies the behavior of the inhibition rate q, state coverage is not adequate as q. Fig. 2 shows that reached states increase quickly for larger k with the same coverage. This is because a state can be reached when one of the hash values avoids a collision. Therefore, even with the same coverage, q decreases faster when k is larger. Let p be state coverage N/M. Then, $q = p^k$ satisfies the condition. We adopt $(N/M)^k$ as the inhibition rate q.

3.1.3 Growth Curve

From the discussion above, the next differential equation is considered to approximate the increase in reached states:

$$\frac{dN}{dt} = r\left\{1 - \left(\frac{N}{M}\right)^k\right\}N.$$

The equation models both the proportional increase by using a small bloom filter and the saturation of the increase by using a large bloom filter. The solution of the equation is

$$N(t) = \frac{M}{\sqrt[k]{1 + Ce^{-krt}}}, \tag{4}$$

where C is an integral constant.

Fig. 4 shows the growth curves described by (4). N is particularly a well-known logistic function when $k = 1$. When $k > 1$, the curve is not symmetric about the inflection point, unlike when $k = 1$.

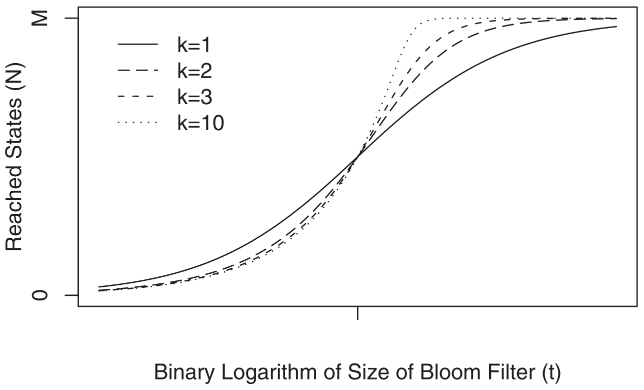

Fig. 4. Sigmoid curves of (4).

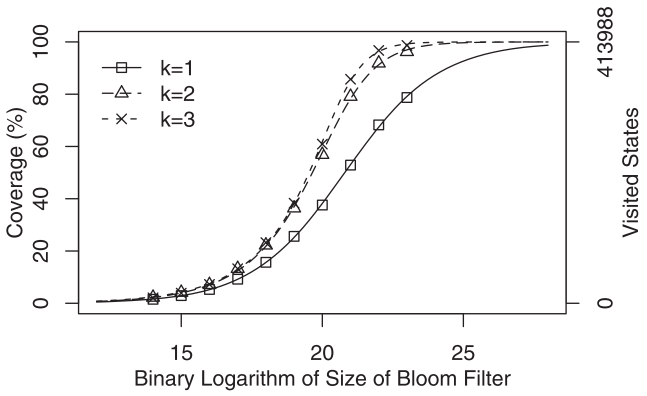

Fig. 5. Fitted curve and measured coverage of brp.red graph.

3.2 Estimation with Growth Curve

We propose two coverage estimation approaches with the growth curve defined by (4). Our approaches leverage the statistics from multiple verification runs with boom filters of different sizes to compensate for the lack of information reflecting the characteristics of models. The basic idea is to estimate the number of reachable states in a model by fitting the growth curve to the statistics of multiple runs. We introduce two approaches: a curve fitting approach and a simple formula approach.

The strategy of using the statistics from multiple verification runs matches up precisely with swarm verification techniques [10]. In swarm verification, to achieve high-quality results with fast turnaround time, parallel verification runs with relatively small bloom filters are performed on many processing cores. Our approaches can estimate coverage with these results from parallel verification runs.

3.2.1 Curve Fitting

A curve fitting technique can extrapolate the number of reachable states. Equation (4) contains three unknown parameters M, C, and r. If the formula accurately describes the relation between t and N, these parameters, particularly M, can be estimated by a curve fitting technique such as a nonlinear least-squares method from the statistics of multiple verification runs.

With the estimate M_E of the number of all reachable states, a coverage estimate $Cov_{fitting}$ is calculated as

$$Cov_{fitting} = \frac{N}{M_E}. \tag{5}$$

3.2.2 Simple Formula

The other approach is to use a simple formula derived from (4) with some assumptions. With the simple formula, only two verification runs are required for estimation.

Let F_N be the fraction $N(t)/N(t-n)$ for a positive number n. It is apparent that F_N approaches 1.0 when coverage approaches 100 percent since $N(t)$ is asymptotic to M. Thus, how close F_N is to 1.0 can be utilized for coverage estimation. From (4), coverage $N(t)/M$ is estimated by the following equation using F_N:

$$Cov_{simple} = \frac{N(t)}{M} = \sqrt[k]{\frac{e^{krn} - F_N^k}{e^{krn} - 1}}. \tag{6}$$

Therefore, if a growth rate r is known in advance, a couple of runs suffice for estimation.

Let r be ln2 as discussed in Section 3.1.1 and F_m be the abbreviation of e^{rn}. F_m is equal to the fraction of the sizes of bloom filters $2^n = 2^t/2^{t-n}$. Hence, with F_m, (6) is rewritten as

$$Cov_{simple} = \sqrt[k]{\frac{F_m^k - F_N^k}{F_m^k - 1}}.$$

This implies that the formula estimates coverage from the fraction of the sizes of bloom filters and the fraction of the numbers of reached states in two verification runs.

When $F_m < F_N$, an estimate cannot be computed because the formula yields a negative or complex number. In such cases, we can use another n for estimation until it yields a positive number.

4 Experiments

For evaluating our approaches, first we experimented to see how well the growth curve defined by (4) approximates the increase in reached states in practical models. Then, we compared the estimation results from our approaches and the conventional ones.

4.1 Evaluation of Growth Curve

To check the validity of the growth curve defined by (4), we fitted it to the measured coverage of 12 model checking graphs from http://fi.muni.cz/xpelanek/state_spaces/ [13]. States in three out of the 12 graphs are reduced by partial order reduction. These reduced graphs have the suffix `.red` in their names.

For a graph with 2^i reachable states, we measured reached states for 10 verification runs with bloom filters that range in size from $2^{\lceil i \rceil - 5}$ to $2^{\lceil i \rceil + 4}$. Then, we fitted the growth curve of (4) to the measured reached states by a nonlinear least-squares method.

Fig. 5 shows the fitted curves (drawn by lines) and the measured coverage of the graph `brp.red`. As can be seen, the fitted curves closely approximate the measured points (plotted by square, triangle, and x-mark, respectively, for $k = 1, 2$, and 3), regardless of the number of hash functions k.

For the other graphs, the fitted curves approximate the measured coverage as well. Table 1 shows the estimated

TABLE 1
Results of Curve Fitting

	$k=1$		$k=2$		$k=3$	
	R^2	r	R^2	r	R^2	r
arbiter	0.98	0.57	1.00	0.63	1.00	0.62
brp.red	1.00	0.60	1.00	0.58	1.00	0.55
cambridge00	1.00	0.95	1.00	0.76	1.00	0.68
elevator2	1.00	0.79	1.00	0.67	1.00	0.63
leader	1.00	0.88	1.00	0.73	1.00	0.69
n-s-original.red	1.00	0.79	1.00	0.60	1.00	0.62
pako	1.00	0.83	1.00	0.69	1.00	0.65
petersonN	1.00	0.76	1.00	0.64	1.00	0.62
pftp	1.00	0.71	1.00	0.63	1.00	0.60
phil5.red	1.00	0.82	1.00	0.69	1.00	0.65
random_30_4	1.00	0.82	1.00	0.69	1.00	0.65
random_30_6	1.00	0.82	1.00	0.69	1.00	0.65

growth rate r and the coefficient of determination R^2, which is a measure of how well the fitted curve represents the data for each graph. The estimated growth rate r fits into the range from $\ln 1.8$ to $\ln 2.3$ for most graphs, that is, it is approximately $\ln 2$ as predicted in Section 3.1.1. However, r tends to decrease slightly as k increases. From the results in which R^2 was approximately 1.0 for all the graphs, the growth curve of (4) closely approximates the increase in reached states of bitstate hashing. This holds true for practical models regardless of whether or not partial order reduction is used. The growth curve could approximate the measured data of the reduced graphs (`brp.red`, `n-s-original.red`, and `phil5.red`).

4.2 Coverage Estimation

To evaluate our estimation, we compared the following approaches:

- curve fitting approach ($Cov_{fitting}$, (5)),
- simple formula approach (Cov_{simple}, (6)),
- Stern's approach (Cov_{Stern}, (1)) and
- Dillinger's approach ($Cov_{Dillinger}$, (2)).

We used the same graphs used in Section 4.1 and evaluated errors in estimation results. For a graph with 2^i reachable states, we estimated coverage for 10 verification runs with bloom filters that ranged in size from $2^{\lceil i \rceil - 5}$ to $2^{\lceil i \rceil + 4}$. In the curve fitting approach, coverage of a verification run with a bloom filter of 2^l bits was estimated from results of six verification runs with bloom filters that ranged in size from 2^{l-5} to 2^l bits by a nonlinear least-squares method. When the fitting failed, we tried another fitting with a fixed growth rate ($r = \ln 2$) as a fallback. When the second fitting failed, it was treated as a missing value. In the simple formula approach, we used a fixed growth rate ($r = \ln 2$) and took 1 as n at the start. Then, we incremented n by 1 until the formula yielded a positive number.

First, we take one of the graphs and give an overview of a tendency of our approaches. Fig. 6 shows the estimation results of the graph `brp.red`. The graph `brp.red` is one of those whose coverage is not very accurately estimated by the conventional approaches. Both of our approaches achieved more accurate results than the others. Especially for single bitstate hashing ($k = 1$), our approaches yielded good results even with a small bloom filter. Though our approaches are better than the existing approaches regardless of k, coverage estimates for $k = 2, 3$ are not very accurate compared with $k = 1$. A similar tendency can be seen in the other graphs.

Next, we describe the distribution of residual errors in coverage estimates. The distribution of residual errors for single bitstate hashing ($k = 1$) for each approach is plotted in Fig. 7. The x-axis is the actual coverage, that is, the proportion of reached states to reachable states. The y-axis is the residual error in percentage point, that is, the absolute value of the difference between an actual coverage and a coverage estimate. Residual errors are basically lower in our approaches than in the conventional approaches. There are many points that have a residual error over 40 percentage

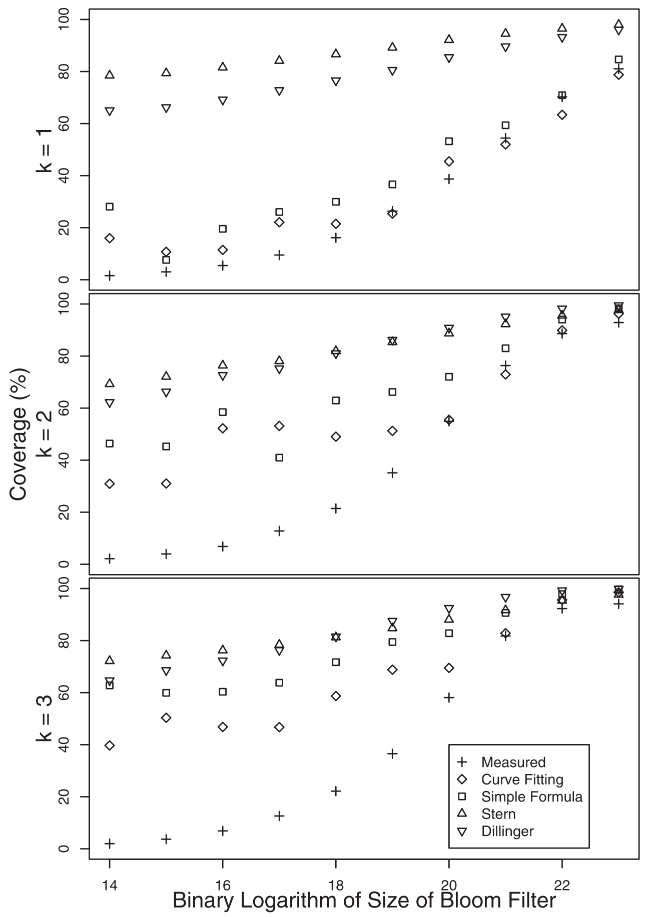

Fig. 6. Coverage estimation of brp.red graph.

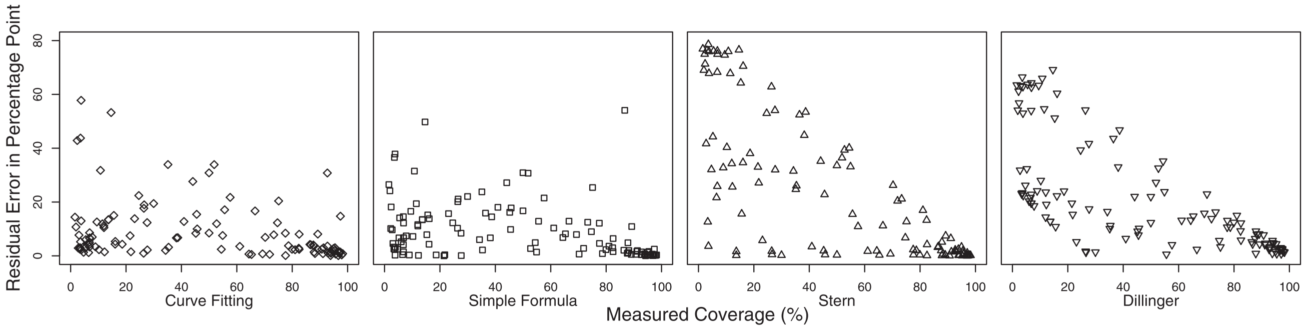

Fig. 7. Residual error distribution of coverage $(k = 1)$.

points in the conventional approaches. Meanwhile, our approaches have quite a few such points. Table 2 summarizes the distribution of Fig. 7. It compares residual errors among intervals of actual coverage (Act. Cov.). For instance, when the actual coverage was within 0 to 20 percent, the third quartile (3Q) values of residual errors by the curve fitting approach were lower than 12.70 percentage points. When the actual coverage is under 40 percent, the 3Q value of a residual error in our approaches is much lower than those of the conventional approaches. On the other hand, when the actual coverage is higher than 80 percent, no significant difference is found from Table 2. Thus, for single bitstate hashing, our approaches outperform the conventional ones when a verification run covers a small portion of a state space.

Finally, we study the difference among the numbers of hash functions. The estimation results for $k = 1, 2, 3$ are summarized in Table 3. The column labeled "failed" represents the number of failures (missing values) in estimation. As seen in Table 3, our approaches are more accurate than the others. Our approaches improve median values and 3Q values by about 10 percentage points regardless of k. However, the residual error tends to increase as k increases. This is considered to be because an increase in hash functions weakens the inhibition of growth even if the actual coverages are the same. In our approaches the inhibition rate q decreases as k increases in accordance with its definition in Section 3.1.2. Thus, even if the actual coverages are similar, an increasing rate of reached states is larger than that for small k. This causes a difficulty for estimation from the increasing rate of reached states because estimation becomes more sensitive to its variation. For instance, with the simple formula approach with $r = \ln 2$ and $n = 1$, an increase rate of $F_N = 1.9$ indicates coverage of 36.1 percent for $k = 2$, while it indicates coverage of 10.0 percent for $k = 1$. Since $F_N = 2.0$ indicates coverage of 0.0 percent, a slight increase in F_N greatly affects an estimate for large k. Therefore, estimation errors enlarge as k increases. Thus, using one hash function is recommended to estimate coverage of bitstate hashing. Once a coverage estimate is obtained, an estimate of the number of reachable states M can also be obtained. Therefore, a coverage estimate with high accuracy can be calculated from the estimated M and the number of reached states with k larger than 1.

When our two approaches are compared, the simple formula approach has more advantages. In contrast to the simple formula approach yielding estimates for all the cases, 29 out of 360 trials failed in the curve fitting approach. In Table 3, the median and 3Q values of the curve fitting approach are only a few percentage points better than those

TABLE 2
Comparison among Approaches for Each Interval of Actual Coverage (k = 1)

Act. Cov. (%)	Approach	Min.	1Q	Median	3Q	Max.	Mean
0 — 20	Curve Fitting	1.22	3.23	6.32	12.70	57.79	11.52
	Simple Formula	0.12	3.22	7.65	14.45	49.79	11.46
	Stern	0.19	32.25	66.03	74.97	78.59	50.62
	Dillinger	10.86	22.32	27.11	61.01	69.19	38.35
20 — 40	Curve Fitting	0.97	2.70	7.49	18.32	33.93	11.30
	Simple Formula	0.02	5.06	11.28	16.95	23.79	11.23
	Stern	0.23	19.98	31.76	52.56	62.83	31.45
	Dillinger	1.33	8.91	16.93	39.91	54.20	22.41
40 — 60	Curve Fitting	2.43	8.49	12.76	21.74	33.91	16.02
	Simple Formula	1.42	9.82	16.74	21.52	30.91	16.22
	Stern	0.34	3.42	22.80	35.19	40.11	21.01
	Dillinger	0.60	7.73	12.35	23.83	35.21	16.43
60 — 80	Curve Fitting	0.13	0.68	3.64	8.28	20.40	6.06
	Simple Formula	0.66	3.26	8.06	11.69	25.40	8.32
	Stern	0.32	0.99	2.16	12.35	26.24	7.74
	Dillinger	2.40	10.66	13.17	14.53	22.97	12.03
80 — 100	Curve Fitting	0.09	1.02	2.14	3.48	30.79	3.47
	Simple Formula	0.04	0.44	0.76	1.97	54.09	3.09
	Stern	0.00	0.17	0.73	2.84	16.95	2.35
	Dillinger	0.76	2.31	4.21	5.76	15.02	4.55

TABLE 3
Summary of Residual Error of Coverage

	k	Min.	1Q	Median	3Q	Max.	Mean	failed
Curve Fitting	1	0.09	2.20	4.33	12.29	57.79	8.93	3
	2	0.09	1.33	5.48	17.81	73.75	13.10	12
	3	0.03	1.46	8.54	33.19	74.37	17.14	14
Simple Formula	1	0.02	1.33	5.63	13.58	54.09	9.14	0
	2	0.00	1.06	8.82	19.65	58.91	13.81	0
	3	0.00	1.83	14.94	36.45	80.24	21.65	0
Stern	1	0.00	1.40	12.69	39.54	78.59	24.00	8
	2	0.30	3.92	25.08	49.55	79.62	28.19	0
	3	0.02	4.87	31.14	58.92	79.14	32.56	0
Dillinger	1	0.60	4.81	14.12	26.48	69.19	20.92	0
	2	0.07	2.22	20.06	40.42	78.47	24.12	0
	3	0.02	1.62	24.11	47.50	80.45	27.10	0

of the simple formula approach. Though the curve fitting approach uses four more verification results than the other, our two approaches do not significantly differ. This is mainly because our approaches strongly depend on how much the growth is inhibited. The closer the number of reached states approaches is to the number of reachable states, the more gently reached states increases. In a sense, our approaches extrapolate the end of the growth from this observation. Thus, a few verification results that are close to the end of the growth have a conclusive impact. From the viewpoints of the number of verification runs and the risk of estimation failures, the simple formula approach has advantages over the curve fitting approach.

5 Discussion

Whether our approaches are applicable depends on the graph structure of models. Let V be a set of reachable states in a state space and $\{V_S, V_I, V_D\}$ be a partition of V such that V_S contains an initial state, there are no direct transitions from V_S to V_D and a state in V_D (V_I) is reachable from a state in V_I (V_S) via no states in V_S (V_D) as shown in Fig. 8. If the probability reaching V_D from V_S is very low for some partition, a search is highly probable to finish without visiting all states in V_D. In bitstate hashing, the difficulty reaching V_D from V_S is dependent on the graph structure of V_I. When states in V_I have an appropriate amount of branches leading to V_D, a search can approach V_D by tracking another branch even if hash values collide at a branch. On the other hand, if very few branches lead to V_D, it is exceedingly difficult to reach V_D. Especially, if V_I is a straight line structure, a single collision makes it impossible to reach V_D. Moreover, the shorter the interval between hash collisions, the more difficult it is to go through V_I.

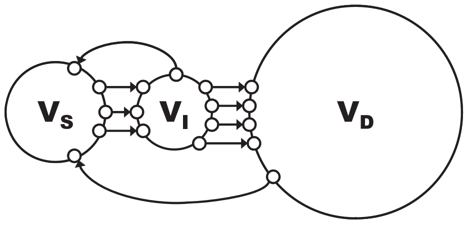

Fig. 8. Partition of state space.

An interval between hash collisions is very short for single bitstate hashing. Assuming that a proportion of on-bits in a bloom filter is p, the expectation of the number of states that can be visited before the next collision (uninterrupted transitions) is approximated by $\frac{1}{p^k} - 1$. One can observe that the expectation is very small for single bitstate hashing ($k = 1$). Only nine states, for instance, are expected to be visited before the next collision when 10 percent of bits are set to 1. One may think that the expectation value is sufficiently large since a search starts with an empty bloom filter. However, this is not the case. The expectation of uninterrupted transitions E before the first collision for single bitstate hashing is calculated by the equation below:

$$E = \sum_{i=1}^{m} i \frac{m^{\underline{i}}}{m^i} \frac{i}{m} = \sum_{i=1}^{m} \frac{m^{\underline{i}}}{m^i},$$

where $m^{\underline{i}}$ is the ith falling factorial of m. When a state and its hash value are regarded as a person and his/her birthday, $E + 1$ is equivalent to the average number of people required to find a pair with the same birthday in the birthday problem. E is identical to Ramanujan's Q-function and has asymptotic expansion [17]:

$$E = \sqrt{\frac{\pi m}{2}} - \frac{1}{3} + \frac{1}{12}\sqrt{\frac{\pi}{2m}} + \mathcal{O}(m^{-1}).$$

This implies that an interval between hash collisions for single bitstate hashing is short even at an early stage of exploration.

Thus, in a state space with a long straight line-like structure, frequent hash collisions increase the risk of a search terminating without visiting a large part of states even if a bloom filter has plenty of room. When a large part of states is omitted because of the graph structure, coverage estimation methods leveraging the number of visited states and the size of a bloom filter results in a high coverage estimate near 100 percent. The expected value of uninterrupted transitions for $k = 1$ takes a smaller value than for larger k at an early stage of exploration. This indicates that coverage estimation for single bitstate hashing is notably sensitive to a long straight line-like structure. However, the expected value of uninterrupted transitions ($\frac{1}{p^k} - 1$) can be improved by using multiple hash functions. This improvement generates a gap in which an estimation with a single hash function results in nearly 100 percent coverage and a search with multiple hash functions visits more states that are distinct in spite of using a bloom filter of the same size. By observing this gap, significant errors stemming from the graph structure can be detected.

Furthermore, the applicability of our approaches is strengthened by a characteristic of practical models. In the case of practical models, the height of the shortest path tree from an initial state is considerably smaller than the number of reachable states. In fact, the typical height of a shortest path tree is $\mathcal{O}(\log M)$ for the number of reachable states M [18] (http://anna.fi.muni.cz/models/). This feature is supported by the fact that practical models rarely have a long straight line structure because model checking aims at verifying systems with nondeterministic transitions, which are described as branches in a state space. Therefore, our approaches are considered applicable for practical models.

6 Conclusion

We have proposed coverage estimation methods for model checking by bitstate hashing. We have analyzed an increase in reached states by enlarging a bloom filter and have introduced a growth curve that approximates it. Our approaches estimate coverage on the basis of the proposed growth curve. The existing approaches are not accurate enough for a verification run with low actual coverage due to the lack of information. They use only the number of reached states and the size of a bloom filter from a single verification run. On the other hand, our approaches estimate coverage by fitting the growth curve to the statistics from multiple verification runs to compensate for the lack of information that reflects the characteristics of state spaces.

We have experimented on the validity of the growth curve and have verified the applicability of our approaches to practical models. The experimental results have confirmed that the proposed growth curve accurately describes the increase in reached states in verification of practical models. Moreover, our approaches outperformed the conventional ones when a verification run covered a small portion of state space. The results have shown that the estimation errors were reduced by about 10 percentage points on average and have confirmed the effectiveness of our approaches regardless of the number of hash functions. Therefore, our approaches are applicable for coverage estimation for practical models. In addition, our approaches can be used as barometers for sequential or parallel verification techniques such as [6], [11], [19], [10], in which coverage of a single verification run is very important.

Note that while our approaches are applicable regardless of the number of hash functions, it is better to use one hash function for coverage estimation. With multiple hash functions, estimation becomes very sensitive to the variation in the increasing rate of reachable states. Therefore, it is difficult to achieve high accuracy for bitstate hashing with multiple hash functions.

Future work will be to confirm the effectiveness of our approaches for variations of bitstate hashing. For instance, the method introduced by Holzmann [6] probabilistically stores a newly visited state into a bloom filter. In such variations, since additional parameters might be required for coverage estimation, the effectiveness and applicability of our approaches are worth a discussion. Hence, we want to adapt our approaches to such variations of bitstate hashing.

References

[1] E.M. Clarke, O. Grumberg, and D. Peled, *Model Checking.* MIT Press, 1999.

[2] G.J. Holzmann, *The Spin Model Checker: Primer and Reference Manual.* Addison Wesley, 2003.

[3] D.L. Dill, A.J. Drexler, A.J. Hu, and C.H. Yang, "Protocol Verification as a Hardware Design Aid," *Proc. IEEE Int'l Conf. Computer Design on VLSI in Computer and Processors,* pp. 522-525, 1992.

[4] W. Visser, K. Havelund, G. Brat, S. Park, and F. Lerda, "Model Checking Programs," *Automated Software Eng.,* vol. 10, no. 2, pp. 203-232, 2003.

[5] P. Wolper and D. Leroy, "Reliable Hashing without Collision Detection," *Proc. Fifth Int'l Conf. Computer Aided Verification,* pp. 59-70, 1993.

[6] G.J. Holzmann, "An Analysis of Bitstate Hashing," *Formal Methods in System Design,* vol. 13, no. 3, pp. 289-307, 1998.

[7] U. Stern and D.L. Dill, "A New Scheme for Memory-Efficient Probabilistic Verification," *Proc. IFIP TC6/WG6.1 Joint Int'l Conf. Formal Description Technique for Distributed Systems and Comm. Protocols, and Protocol Specification, Testing, and Verification,* pp. 333-348, 1996.

[8] B.H. Bloom, "Space/Time Trade-Offs in Hash Coding with Allowable Errors," *Comm. ACM,* vol. 13, no. 7, pp. 422-426, 1970.

[9] J. Eckerle and T. Lais, "New Methods for Sequential Hashing with Supertrace," 1998.

[10] G. Holzmann, R. Joshi, and A. Groce, "Swarm Verification Techniques," *IEEE Trans. Software Eng.,* vol. 37, no. 6, pp. 845-857, Nov./Dec. 2011.

[11] P.C. Dillinger and P. Manolios, "Bloom Filters in Probabilistic Verification," *Formal Methods in Computer-Aided Design,* pp. 367-381, Springer, 2004.

[12] D. Owen and T. Menzies, "Lurch: A Lightweight Alternative to Model Checking," *Proc. Int'l Conf. Software Eng. and Knowledge Eng.,* pp. 158-165, 2003.

[13] R. Pelánek, T. Hanžl, I. Černá, and L. Brim, "Enhancing Random Walk State Space Exploration," *Proc. 10th Int'l Workshop Formal Methods for Industrial Critical Systems,* pp. 98-105, 2005.

[14] R. Grosu and S.A. Smolka, "Monte Carlo Model Checking," *Tools and Algorithms for Construction and Analysis of Systems,* vol. 3440, pp. 271-286, 2005.

[15] E. Tronci, G.D. Penna, B. Intrigila, and M.V. Zilli, "A Probabilistic Approach to Automatic Verification of Concurrent Systems," *Proc. Asia-Pacific Software Eng. Conf.,* pp. 317-324, 2001.

[16] T. Menzies, D. Owen, and B. Cukic, "Saturation Effects in Testing of Formal Models," *Proc. 13th Int'l Symp. Software Reliability Eng.,* pp. 15-26, 2002.

[17] P. Flajolet, P.J. Grabner, P. Kirschenhofer, and H. Prodinger, "On Ramanujan's Q-Function," *J. Computational and Applied Math.,* vol. 58, no. 1, pp. 103-116, 1995.

[18] R. Pelánek, "Typical Structural Properties of State Spaces," *Model Checking Software,* S. Graf and L. Mounier, eds., pp. 5-22, Springer, 2004.

[19] M.B. Dwyer, S. Elbaum, S. Person, and R. Purandare, "Parallel Randomized State-Space Search," *Proc. 29th Int'l Conf. Software Eng.,* pp. 3-12, 2007.

Satoshi Ikeda received the MS degree in informatics from the Graduate School of Informatics, Kyoto University, Japan. He joined NEC Corporation in 2007, where he is with the Cloud System Research Laboratories. His research interests include the areas of model checking and protocol engineering. He is a member of IEICE (Japan).

Masahiro Jibiki received the PhD degree in systems management from the University of Tsukuba, Japan, in 2003. In 1992, he joined NEC Corporation and was a researcher in the Central Research Laboratories until 2011. From 2006 to 2009, he was also a visiting professor at the University of Wakayama, Japan. Currently, he is an expert researcher in the National Institute of Information and Communications Technology. His research interests include networking, distributed systems, and software science.

Yasushi Kuno received the BS, MS, and PhD degrees in information science from the Tokyo Institute of Technology in 1979, 1981, and 1991, respectively. He is a professor in the Graduate School of Systems Management, University of Tsukuba, Tokyo. His current research interests include programming languages, user interface, and informatics education in general. He is a chair of the Primary and Secondary Educational Committee of the Information Processing Society of Japan (IPSJ). He has published more than 100 conference and journal papers, is a (co)author of 20 books, and has translated more than 10 books on the WWW, programming, and software. He is a member of the IEEE Computer Society.