Symbolic Partition Refinement with Dynamic Balancing of Time and Space

Ralf Wimmer*
Institute of Computer Science, Albert-Ludwigs-University Freiburg, Germany
wimmer@informatik.uni-freiburg.de

Salem Derisavi†
IBM Toronto Software Lab, Canada
derisavi@ca.ibm.com

Holger Hermanns*
Department of Computer Science, Saarland University, Germany
hermanns@cs.uni-sb.de

Abstract

Bisimulation minimization is one of the classical means to fight the infamous state space explosion problem in verification. Particularly in stochastic verification, numerical algorithms are applied, which do not scale beyond systems of moderate size. To alleviate this problem, symbolic bisimulation minimization has been used effectively to reduce very large symbolically represented state spaces to moderate size explicit representations. But even this minimization may fail due to time or memory limitations. This paper presents a symbolic algorithm which relies on a hybrid symbolic partition representation. It dynamically converts between two known representations in order to provide a trade-off between memory consumption and runtime. The conversion itself is logarithmic in the partition size. We show how to apply it for the minimization of Markov chains, but the same techniques can be adapted in a straightforward way to other models like labeled transition systems or interactive Markov chains.

1 Introduction

Bisimulation equivalence [21, 20] equates two systems if their stepwise behavior is indistinguishable. Since bisimilar systems satisfy the same CTL* and μ-calculus formulae, bisimilar systems can be used interchangeably in compositional model construction [22] and model checking [8]. In particular one can switch from the original to the *minimal size* quotient system under bisimulation when verifying some system property, in order to fight the infamous state space explosion. Efficient algorithms for bisimulation minimization are available [23, 17, 6]. This minimization generalizes symmetry reduction, but at a higher computational cost. While it has been reported that bisimulation minimization is not favorable to speed up verification of invariance properties [12], it does pay off in many other areas of system verification, such as compositional applications [11, 13] where large systems are constructed from smaller building blocks, or when checking behavioral equivalence, or multiple system properties.

For some areas of quantitative verification, especially model checking of stochastic systems, bisimulation minimization appears almost indispensable. In this context, numerical algorithms are applied to calculate satisfaction probabilities of system properties [24], and these algorithms do not scale to large systems, since no effective technique is known to avoid the need to store at least one floating-point value per state. Thus, one strives for a reduction of the state spaces as much as possible before model checking, and bisimulation minimization does therefore pay off considerably [18]. This motivates the renaissance of work on bisimulation minimization algorithms. Rooted in the work of Blom and Orzan [2, 3, 4], who developed a distributed algorithm for (strong and branching) bisimulation, effective symbolic algorithms have been developed [28, 10, 9] and applied to models of sizes otherwise far out of reach of contemporary stochastic model checkers [5]. Here, the transition system is first con-

*This work was partly supported by the German Research Council (DFG) as part of the Transregional Collaborative Research Center "Automatic Verification and Analysis of Complex Systems" (SFB/TR 14 AVACS). See www.avacs.org for more information.

†Part of this work was done while the co-author was at Carleton University in Ottawa, Canada.

structed as an (MT)BDD, then it undergoes an aggressive minimization (using branching or stochastic bisimulation), before undergoing stochastic model checking.

While the above symbolic algorithms are similar in spirit, they have conceptual and practically relevant differences. The work of [28] utilizes a "fast" partition representation that makes it possible to have a very efficient algorithm in terms of computation time, but in particular for large numbers of equivalence classes it consumes considerable memory. On the other hand, the "compact" partition representation of [10] stays very small in terms of space requirements, but its drawback is that performing operations on the representation can become quite expensive timewise. The difference is caused by drastically different representation techniques for encoding the state space partitions as BDDs, which are accessed and refined by the algorithm.

To further push the limits of this technology, this paper investigates a combination of the two approaches. We develop an algorithm which is memory efficient by using the compact partition representation of [10] and runtime efficient by using the fast partition representation and refinement algorithm of [28]. Our "hybrid" approach offers a "spectrum" of representations whose extremes are the fast representation on one side and the compact representation on the other. It provides us a parameter by which we can control where in the spectrum a specific instance of the representation stands.

The contributions of the paper are 1) an algorithm that converts between fast and compact partition representations in a logarithmic number of BDD operations, 2) a simple but effective algorithm that automatically changes the parameter mentioned above to balance the time and space requirements of the algorithm such that the refinement works at maximal speed without exceeding the available memory, and 3) an implementation of the conversion and parameter selection algorithms into the principal refinement algorithm. We experimentally evaluate the benefits of our algorithm, and compare its performance with the algorithms of [9] (that uses the fast representation of [28]) and [10].

The entire work is presented here in the context of continuous-time Markov chain minimization. However, there is a much broader spectrum of possible applications, since the techniques are straightforwardly adaptable to labeled transition systems, discrete-time or interactive Markov chains, etc. The core contribution of this paper is thus a general, fast, and memory efficient algorithm for symbolic bisimulation minimization.

Organization of the paper: In the next section, we will briefly review the foundations of our hybrid algorithm. We will then present, in Section 3, our new hybrid algorithm. We will show our experimental results in Section 4 demonstrating the effectiveness of our approach. We finally conclude in Section 5.

2 Background

In this section, we first review the concepts of continuous-time Markov chains and stochastic bisimulation. We then give a brief account of the general principle of signature-based bisimulation minimization algorithms, to set the ground for the new algorithm developed in this paper. We finally review symbolic data structures and their use to represent various entities appearing in our context, like matrices and partitions.

2.1 MCs and Stochastic Bisimulation

A *continuous-time Markov chain* (MC) M is a pair $M = (S, \mathbf{R})$ where S is a finite non-empty set of states and $\mathbf{R} : S \times S \to \mathbb{R}^{\geq 0}$ is the transition rate matrix such that $\mathbf{R}(s, s) = 0$ for all $s \in S$. The *generator matrix* $\mathbf{Q} : S \times S \to \mathbb{R}$ is defined as $\mathbf{Q}(s, s) = -\sum_{s' \in S} \mathbf{R}(s, s')$ and $\mathbf{Q}(s, t) = \mathbf{R}(s, t)$ for all $s, t \in S$ with $s \neq t$.

A *partition* P of a set S is a set of pairwise disjoint, non-empty subsets of S, such that their union equals S. Elements of P are called *blocks* of P. In the sequel, a *subpartition* is a subset of a partition. For elements s and t in the same block of P, we write $s \equiv_P t$. Let P and P' be partitions of S. P is called a *refinement* of P' (or conversely P' coarser than P), denoted by $P \sqsubseteq P'$, if $\forall B \in P\ \exists B' \in P' : B \subseteq B'$.

For a matrix $\mathbf{A}$ and $B, B' \subseteq S$, we define $\mathbf{A}(B, B') = \sum_{s \in B} \sum_{s' \in B'} \mathbf{A}(s, s')$. For $s \in S$ and $B \subseteq S$, we use $\mathbf{A}(s, B)$ and $\mathbf{A}(B, s)$ instead of $\mathbf{A}(\{s\}, B)$ and $\mathbf{A}(B, \{s\})$ respectively.

Definition 1 *Let $M = (S, \mathbf{R})$ be an MC with generator matrix $\mathbf{Q}$. A partition P of S is a* stochastic bisimulation *if $\mathbf{Q}(s, B) = \mathbf{Q}(t, B)$ for all blocks B of P and all states $s, t \in S$ satisfying $s \equiv_P t$.*

2.2 Bisimulation Minimization using Signature-based Refinement

Most bisimulation minimization algorithms in the literature use iterative partition refinement such that, in each iteration, the current partition is refined w. r. t. a block retrieved from a list of potential splitters. Blom and Orzan were the first to devise an iterative algorithm that, in each iteration, refines the current partition w. r. t. *all* blocks simultaneously [2]. Their algorithm works by computing, in each iteration, the *signature* of all states w. r. t. the current partition (as opposed to the current splitter in conventional algorithms). Defined formally in Eq. (1), the signature of a state with respect to a partition is the total transition rate from the state to each block of the partition. States are kept in the same block iff they have the same signature. The algorithm stops once the signatures reach a fixpoint, i. e., a partition that will not be split any further.

This signature-based principle is independent of the type of representation (i. e., explicit or symbolic) used.

Originally designed for an *explicit* distributed algorithm for branching bisimulation minimization of non-probabilistic transition systems, Wimmer et al. [28] and Derisavi [9] used it to develop *symbolic* minimization algorithms for non-probabilistic and probabilistic systems, respectively.

In the following, we present a signature-based algorithm, which computes the coarsest stochastic bisimulation that refines an initial partition, possibly induced by atomic labels, rewards, etc. attached to states.

For a given MC $M = (S, \mathbf{R})$ with generator matrix $\mathbf{Q}$ and initial partition $P^{(0)}$ of S, the signature-based algorithm is given as:

$$P^{(i+1)} = \mathsf{sigref}(P^{(i)}) \text{ for } i \geq 0$$
$$\mathsf{sigref}(P) = \left\{\{s \in S \mid \mathsf{sig}(P,s) = \mathsf{sig}(P,t) \wedge s \equiv_{P^{(0)}} t\} \mid t \in S\right\}$$
$$\mathsf{sig}(P,s) = \{(r,B) \in \mathbb{R} \times P \mid r = \mathbf{Q}(s,B)\} \quad (1)$$

Starting with the given initial partition $P^{(0)}$ (or $P^{(0)} = \{S\}$ if no initial partition is explicitly given), the algorithm iteratively applies the sigref-operator until a fixpoint is reached. Theorem 1 guarantees that the fixpoint is the coarsest stochastic bisimulation of the MC. For a proof, we refer the reader to [9].

Theorem 1 *There exists $f \leq |S| - |P^{(0)}|$ such that $P^{(f+1)} = P^{(f)}$ and $P^{(f)}$ is the coarsest refinement of $P^{(0)}$ that is also a stochastic bisimulation on M.*

2.3 (Multi-terminal) Binary Decision Diagrams

Algorithms that rely on explicit state space representations (e. g., sparse matrix representations) are severely limited by the size of systems that they can handle. To circumvent this problem, we employ symbolic data structures, in particular (Multi-terminal) Binary Decision Diagrams ((MT)BDDs) [7, 1] in our algorithm to represent data.

BDDs are a data structure for the compact representation of binary functions of h binary variables, i. e., $\{0,1\}^h \to \{0,1\}$. MTBDDs are a generalization of BDDs used to represent functions of h binary variables, i. e., $\{0,1\}^h \to A$ where A is a finite set. We use calligraphic letters to denote MTBDDs. We assume that all (MT)BDDs have a fixed variable ordering and are reduced. We denote the number of nodes of (or the size of) an MTBDD $\mathcal{G}$ by $|\mathcal{G}|$. In the following, we explain how to use them to represent sets, matrices, and in particular, partitions and signatures.

Set representation Let $N \subseteq \{0,1\}^n$ be a set of boolean vectors. It can be represented symbolically by an MTBDD $\mathcal{N}(x)$ such that $\mathcal{N}(x) = 1$ if $x \in N$ and 0 otherwise.

Matrix representation We use MTBDDs to efficiently represent transition matrices of MCs. A matrix $\mathbf{R} : \{0,1\}^h \times \{0,1\}^h \to \mathbb{R}$ can be represented using an MTBDD $\mathcal{R}$ with $2h$ binary variables. The first h variables encode the row index and the other h variables the column index. We use an interleaved variable ordering in which each row variable is immediately followed by its corresponding column variable or vice-versa. An interleaved variable ordering often leads to small MTBDDs for MCs which are generated from high-level models [14].

Partition representation The representation of state space partitions appearing in the refinement algorithm is a crucial aspect of the setting considered in this paper. We are aware of four distinct techniques for the symbolic representation of a partition $P = \{B_1, \ldots, B_n\}$. For the third and fourth technique, we presuppose an arbitrary, but fixed order on the blocks of each represented partition.

1. To use a BDD $\mathcal{P}$ to represent the corresponding equivalence relation $\equiv_P$ such that $\mathcal{P}(s,t) = 1$ iff $s \equiv_P t$. This representation is used, e. g., in [6], which pioneered symbolic bisimulation minimization.
2. To use one BDD per block. A partition representation is then a set $\{\mathcal{B}_1, \ldots, \mathcal{B}_n\}$ of BDDs such that $\mathcal{B}_i(s) = 1$ iff $s \in B_i$.
3. To use an extra set of BDD variables to denote the block index. The partition is represented by a BDD $\mathcal{P}$ such that $\mathcal{P}(s,k) = 1$ iff $s \in B_k$. This representation was introduced in [28] to compute branching bisimulation on transition systems.(FR)
4. To use a vector of $d = \lceil \lg n \rceil$ BDDs $(\mathcal{B}_0, \ldots, \mathcal{B}_{d-1})$ such that $\mathcal{B}_j(s) = 1$ iff $s \in B_i$ and the j^{th} bit of i is one. In other words, $\mathcal{B}_j$ is the union of all blocks whose indices have 1 in their j^{th} bit. This representation technique was introduced in [10], for symbolic computation of stochastic bisimulation in Markov chains. (CR)

These four representations differ in terms of their time and space efficiency when applied in a symbolic implementation of a refinement algorithm. We report on detailed experiments in Section 4, but provide a preview here for the sake of the exposition.

The first representation is inefficient in terms of its memory requirements (see Section 4) and that leads to its inefficiency in performing partition operations as well [28]. As shown in Section 4 by a number of example models, the second representation is not space-efficient in practice either.

We classify the third technique as the one that enables us to implement the key operations of the algorithm very efficiently in terms of running time. We therefore call it the *fast* representation (FR). One disadvantage of the fast representation is its size: the size

```
BisimMin(P^(0))
1 P' := P^(0)
2 repeat
3    P := P'
4    σ(s,k) := Q_t^+(R(s,t) · P(t,k)) − D(s) · P(s,k)
5    P' := SigRefine(σ)
6 until (P = P')
7 return P
```

Figure 1: The Original Bisimulation Minimization Algorithm using Signature-based Refinement

of BDD $\mathcal{P}$ is at least linear in the number of blocks. More precisely, $|\mathcal{P}| = \Omega(|P|)$, and that makes it unsuitable for partitions with large number of blocks. The fourth technique, in turn, is considerably slower to use, but almost always has the smallest number of nodes among the four representations. Therefore, we call it the *compact* representation (CR) hereafter.

Signature representation We use an MTBDD $\sigma_P(s,k)$ to represent the signature of states in S w. r. t. a partition P which is defined as follows: $\sigma_P(s,k) = r$ iff $(r, B_k) \in \mathsf{sig}(P,s)$.

2.4 Symbolic Implementation of the Refinement Algorithm

Fig. 1 shows the symbolic implementation of the signature-based algorithm explained in Section 2.2. Line 4 computes the MTBDD representation of the signatures and SigRefine, in line 5, returns the partition refined w. r. t. the signatures. More details follow.

Let $\mathcal{Q}$ be the MTBDD of the generator matrix and $\mathcal{P}$ be the compact representation of the current partition. We then compute the MTBDD representation $\sigma(s,k)$ of the signatures as follows:

$$\sigma(s,k) = \mathrm{Q}_t^+\big(\mathcal{Q}(s,t) \cdot \mathcal{P}(t,k)\big) \tag{2}$$

in which $\mathrm{Q}_x^{\odot}$ is the quantification operator w. r. t. an associative and commutative operator $\odot$:

$$\mathrm{Q}_x^{\odot}(\mathcal{A}(x,y)) = \bigodot_{a\in\{0,1\}^h} \mathcal{A}(a,y)$$

The problem with Eq. (2) is that $|\mathcal{Q}|$ might be excessively large due to the non-zero diagonal elements. To tackle this problem, we rewrite Eq. (2) as follows:

$$\sigma(s,k) = \mathrm{Q}_t^+(\mathcal{R}(s,t) \cdot \mathcal{P}(t,k)) - \mathcal{D}(s) \cdot \mathcal{P}(s,k) \tag{3}$$

in which $\mathcal{D}(s) = \mathrm{Q}_t^+(\mathcal{R}(s,t))$ and $\mathcal{R}$ is the MTBDD representation of the transition rate matrix. Remarkably, $|\mathcal{D}|$ often is very small.

To implement the refinement operation sigref symbolically, we exploit the following observation. Assume that we have a variable order in which all state variables precede the block number variables. Then, each state corresponds to a path in the MTBDD which ends in a node that represents the signature of that state. Furthermore, since we use reduced MTBDDs, the paths of all states with the same signature must lead to the same node. To obtain the refined partition, we simply replace the nodes representing signatures by new block numbers. For more details, see [28].

Function SigRefine implements the sigref operator, takes $\sigma(s,k)$ as input, and runs in $O(|\sigma|)$ time.

To take an arbitrary initial partition into account, we have two possibilities: either we refine only one block of the initial partition at a time [27] or we add an extra entry to each signature such that the signatures of states belonging to different blocks of the initial partition cannot be identical [9].

3 Hybrid Representation

Section 2.3 has presented four different partition representations two of which have desirable properties: the compact representation (CR) is very efficient in terms of memory requirement, but partition manipulation (such as adding and removing a block) is relatively expensive. On the other hand, the fast representation (FR) enables us to perform the operation sigref very efficiently in terms of speed but its space requirement is high for partitions with a large number of blocks.

3.1 Overall Idea

To get the best of both representations, our conceptual innovation is to provide a "hybrid" representation of the partition that uses FR for computation and CR for storage. Recall that the core computational step, in each iteration, is computing the signature of *all states* w. r. t. the *current partition*. Our key idea is to use CR to represent the current partition, but to represent the subpartition of states for which the signature is computed in FR, and to convert that into CR after signature-based refinement. To enable this, we do not necessarily compute the signatures of *all* states simultaneously as in BisimMin.

Instead, we do that in a number of steps. In each step, the signatures of all states in a *chunk* are computed. A chunk is a collection of blocks of the current partition. This leads to a time-space trade-off depending on the number of blocks we convert to FR for refinement. For the approach to be effective, it is important that the additional overhead due to conversion is low.

The pseudocode of BisimMinHybrid, the hybrid algorithm, is given in Fig. 2. Its outer loop corresponds to the main loop of BisimMin. In each iteration of the inner loop of BisimMinHybrid, Chunk computes a chunk consisting of *csize* blocks of the current partition $\mathcal{P}_{\mathrm{CR}}$. Then, Signature computes the signatures of all states in the chunk. Based on the signatures, SigRefine refines the chunk into a subpartition in FR, and

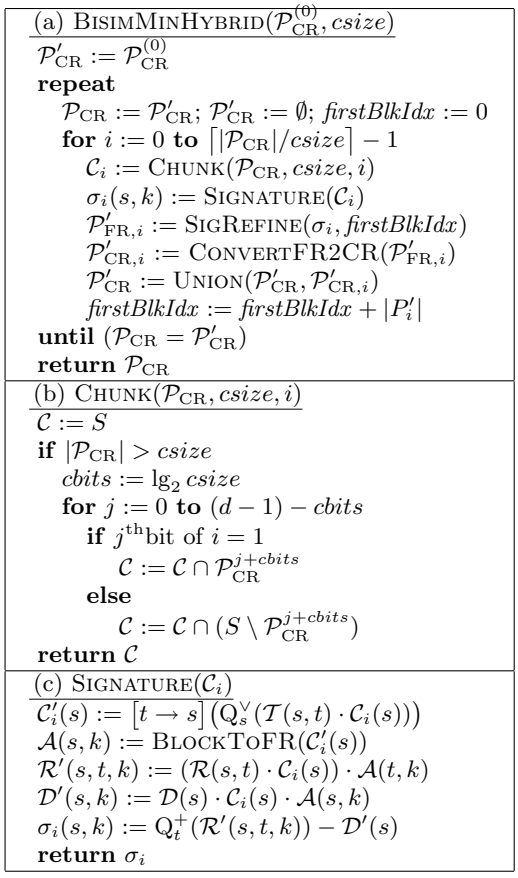


Figure 2: Bisimulation Minimization Algorithm using Hybrid Partition Representation

ConvertFR2CR converts the subpartition into compact representation. Finally, Union adds it to $\mathcal{P}'_{\mathrm{CR}}$, the compact representation of the (new) refined subpartition computed so far.

The hybrid version of the algorithm consumes less memory because it avoids representing both the signatures of all states and also the complete partition using FR (σ, $\mathcal{P}$, and $\mathcal{P}'$ in BisimMin). Instead, at each point of time only one chunk ($\mathcal{C}_i$), signatures of its states (σ_i), and its refinement represented as FR ($\mathcal{P}'_{\mathrm{FR},i}$) are stored. Since the number of nodes in the last two MTBDDs grows at least linearly with $csize$, we can adjust the memory consumption of the algorithm and achieve various time-space trade-offs by varying $csize$. That enables the hybrid algorithm to handle MCs out of reach of the original FR-based algorithm due to memory limitations.

The extraction of the quotient system after bisimulation computation can be performed exactly as described in [10], since the final partition is given in CR.

3.2 Signature Refinement in the Hybrid Representation

In the following, we will explain in detail how each of the steps of BisimMinHybrid is performed.

Computing the chunks As mentioned above, BisimMinHybrid partitions the state space into chunks and then it refines each chunk separately. Chunk($\mathcal{P}_{\mathrm{CR}}, csize, i$) computes the i^{th} chunk consisting of $csize$ blocks of P represented compactly as $\mathcal{P}_{\mathrm{CR}}$. In other words, it returns the union of blocks of P with indices $i \cdot csize$ through $\min((i+1) \cdot csize, |P|) - 1$.

Fig. 2(b) shows the pseudocode of Chunk. The variable d is the number of MTBDDs of the compact representation of P. We restrict $csize$ to be a power of 2, which enables us to perform Chunk using $O(\lg |P|)$ symbolic operations.

Conversion from FR to CR Let $\mathcal{P}_{\mathrm{FR}}(s,k)$ be the fast representation of the partition $P = \{B_1, \ldots, B_n\}$ and $(\mathcal{P}^0_{\mathrm{CR}}, \ldots, \mathcal{P}^{d-1}_{\mathrm{CR}})$ be its compact representation. Recall that $\mathcal{P}^j_{\mathrm{CR}}$ is the union of all blocks B_i such that the j^{th} bit of i is one. Consequently, $S \setminus \mathcal{P}^j_{\mathrm{CR}}$ is the union of all blocks B_i such that the j^{th} bit of i is zero.

Therefore, to convert $\mathcal{P}_{\mathrm{FR}}$ to the compact representation, we have

$$\mathcal{P}^j_{\mathrm{CR}}(s) = \mathrm{Q}^{\vee}_k \left(\mathcal{P}_{\mathrm{FR}}(s,k)|_{k_j=1}\right). \tag{4}$$

Observe that the conversion takes only $O(d) = O(\lg |P|)$ symbolic operations. Notably, the conversion from CR to FR can also be done in a logarithmic number of steps, but is not needed for the algorithm.

Computing the signatures Signature($\mathcal{C}_i$) (see Fig. 2(c)) computes the signatures of all states in $\mathcal{C}_i$, the i^{th} chunk of the current partition $\mathcal{P}_{\mathrm{CR}}$. In Signature, $\mathcal{T}$ is the MTBDD of the 0-1 transition matrix, i. e., $\mathcal{T}(s,t) = 1$ if there is a transition from s to t, and $\mathcal{T}(s,t) = 0$ otherwise. We first compute $\mathcal{C}'_i$ the set of successor states of $\mathcal{C}_i$. Then, BlockToFR attaches to each state in $\mathcal{C}_i$ its block number and stores it in $\mathcal{A}$, i. e., $\mathcal{A}(s,k) = 1$ iff $s \in B_k \cap C'_i$, and $\mathcal{A}(s,k) = 0$ otherwise. We explain below how this is performed efficiently. The remaining three lines provide the restriction of the source states in the generator matrix $\mathcal{Q}(s,t)$ to the current chunk $\mathcal{C}_i$, and replace the target states therein with their block numbers. As discussed in 2.4 for Eq. (3), we avoid the representation of the generator matrix in this computation.

The central novelty in Signature is BlockToFR($\mathcal{U}$). It computes the fast representation of the restriction of a partition on the set $U \subseteq S$. The straightforward approach to do so is to compute $\mathcal{P}_{\mathrm{FR}}(s,k) \cdot \mathcal{U}(s)$. However, that necessitates the generation of the fast (and possibly large) representation of P which we want to avoid. BlockToFR exploits the compact representation and performs only

$O(d) = O(\lg |P|)$ symbolic operations to do so:

$$\textsc{BlockToFR}(\mathcal{U}) = \bigwedge_{j=0}^{d-1} \left(k_j \wedge (\mathcal{P}^j_{\mathrm{CR}} \cap \mathcal{U})\right) \vee \left(\overline{k_j} \wedge \left((S \setminus \mathcal{P}^j_{\mathrm{CR}}) \cap \mathcal{U}\right)\right) \quad (5)$$

The intuition behind Eq. (5) is that $k_j \wedge (\mathcal{P}^j_{\mathrm{CR}} \cap \mathcal{U})$ (resp. $\overline{k_j} \wedge ((S \setminus \mathcal{P}^j_{\mathrm{CR}}) \cap \mathcal{U})$) sets to 1 (resp. 0) the j^{th} block index variable of all states in U that belong to a block whose index has 1 (resp. 0) in its j^{th} bit.

Computing the union of two partitions In BisimMinHybrid, we refine a chunk using the signatures of its states resulting in $\mathcal{P}'_{\mathrm{CR},i}$, a partition of the chunk. Then, Union computes a new subpartition that contains all blocks of $\mathcal{P}'_{\mathrm{CR},i}$ and $\mathcal{P}'_{\mathrm{CR}}$, the computed subpartition of the state space. In the new subpartition, all the blocks have the same index as they had in $\mathcal{P}'_{\mathrm{CR}}$ or $\mathcal{P}'_{\mathrm{CR},i}$.

Lemma 1 forms the basis of the Union operation:

Lemma 1 *Let P be a partition of $U \subseteq S$ and $\{P', P''\}$ be a partition of P. Furthermore, let $(P_0, \ldots, P_{d-1})$, $(P'_0, \ldots, P'_{d-1})$, and $(P''_0, \ldots, P''_{d-1})$ be the CR of P, P', and P'' respectively. Then,*

$$P_j = P'_j \cup P''_j. \quad (6)$$

If the CR of P' and P'' have different lengths, we append empty sets to the shorter representation to bring them to the same length. Using Eq. (6), $\textsc{Union}(P', P'')$ takes only $\lg(|P'| + |P''|)$ symbolic union operations.

To make Lemma 1 applicable and perform $\textsc{Union}(\mathcal{P}'_{\mathrm{CR}}, \mathcal{P}'_{\mathrm{CR},i})$ efficiently, we need to assign the block numbers carefully. In particular, we need the indices of $\mathcal{P}'_{\mathrm{CR}}$ and $\mathcal{P}'_{\mathrm{CR},i}$ to be disjoint and also not to have "holes" (i. e., to have consecutive block indices). Hence, we only have to compute the union of the corresponding BDDs. To facilitate this, we always make the indices of blocks of $\mathcal{P}'_{\mathrm{CR}}$ start from zero in each iteration of the outer loop. When we want to index the blocks of $\mathcal{P}'_{\mathrm{FR},i}$, we set our starting index to be $|\mathcal{P}'_{\mathrm{CR}}|$, i. e., the next available index in $\mathcal{P}'_{\mathrm{CR}}$. That is reflected in the second parameter of SigRefine in Fig. 2(a).

3.3 Optimizations

Avoiding unnecessary conversions During the initial iterations of the outer loop of BisimMinHybrid, the number of blocks of P is normally smaller than the chunk size *csize*. That means the whole partition consists only of one chunk. In that case, we completely avoid the overhead of the conversions by only using FR until the number of blocks exceeds *csize*. Only then do we start the refinement with the hybrid partition representation.

As a result, the algorithm dynamically switches the partition representation from FR to hybrid only when it is necessary. The advantage is that the algorithm achieves the efficiency of the algorithm in [9] (solely based on FR) as long as the resulting bisimulation does not exceed *csize*.

Automatic selection of *csize* As explained above, *csize* enables us to change the time–space balance of the algorithm. As we increase *csize*, the running time of the algorithm decreases and its space consumption increases. Therefore, it is important to be able to automatically choose an appropriate value for *csize*. In the following, we present an algorithm to do so.

Our goal is to use as much of the available physical memory as possible, thereby minimizing the computation time. There are ample possibilities to deal with the time–space trade-off behind the chunk size. We implemented the following.

We let the user provide a memory limit. Initially, we set $csize = 2^{\lceil \log_2 |S| \rceil}$ (upper bound) such that all blocks fit into one chunk. Therefore, the algorithm starts using only FR. If the memory limit is exceeded, we set $csize = 2^{\lceil \log_2 |P| \rceil - 1}$ where P is the last successfully computed partition. The algorithm is then restarted using P as the initial partition. Since $csize < |P|$, the algorithm uses a hybrid of CR and FR, thereby reducing the memory consumption at the price of higher runtime. Afterwards, each time the memory limit is exceeded, the chunk size is divided by two and the algorithm is restarted using the last successfully computed partition as the initial partition.

If the chunk size becomes smaller than one, the bisimulation computation cannot be completed within the given memory bound. Either the algorithm has to be aborted or the memory limit has to be increased.

4 Experimental Study

4.1 Example Models and Implementation

We have implemented our hybrid algorithm in C++ using the CUDD package [26] as the MTBDD library. To generate the MTBDD representations of the input MCs, we used the probabilistic model checking tool PRISM [15]. All the code involved in the experiments was compiled using gcc 4.1.2. The experiments were conducted on a Dual Core AMD Opteron™ 2.4 GHz CPU with 4 GB of main memory running Linux in 32 bit mode. We have stopped any experiment that takes more than 2000 seconds.

We consider three different example models from the literature to study the performance of the algorithm: A fault-tolerant parallel computer system (FPCS) [25], a peer-to-peer (`P2P`) protocol based on BitTorrent (studied in [19]), and a cyclic server polling system [16]. For the first model, we converted the SAN (Stochastic Activity Network) specification to the

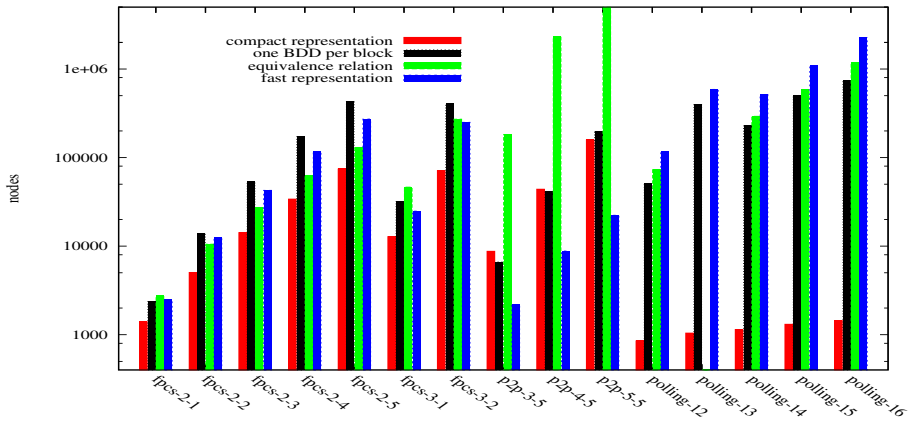


Figure 3: Sizes of the partitions using various representation techniques

PRISM input language. We obtained the PRISM specifications of the other two models from `http://www.prismmodelchecker.org/casestudies/index.php`.

The first two models have two parameters. For FPCS, they denote the number of computers in the system and the number of memory modules in each computer, respectively. For `P2P`, they represent the number of clients and the number of blocks of the file to be transmitted, respectively. The third model has only one parameter which denotes the number of servers.

4.2 Results

Partition sizes We first computed the coarsest bisimulation partition for the example models and converted them to the four partition representations described in Section 2.3 to compare their sizes.

For the representation of the equivalence relation we used an interleaved variable order, since an interleaved variable order often leads to small (MT)BDDs [14]. For the fast representation we used a variable order such that the block number variables are placed at the end of the order (i. e. at the bottom of the BDD). We need such a variable order to be able to perform the refinement operation efficiently (see Section 2.4).

Fig. 3 shows the result of this experiment. Note that the scale of the vertical axis is logarithmic.

For `FPCS` and `Polling` models, the compact representation, shown using left-most (red) bars of each benchmark, deserves its name, while the other representations are, in some cases, significantly larger. The exception is `P2P` for which CR is slightly, but not prohibitively, larger than FR. The reason is that `P2P` models exhibit a large degree of symmetry, resulting in quite small bisimulation partitions (in the order of a few hundred blocks). In this case, the overhead introduced by the block number variables in FR is marginal and the MTBDD size is dominated by other effects.

Effect of the chunk size To evaluate the effect of the chunk size on the time and space requirements of the hybrid algorithm from Section 3, we applied it to the three models, varying the chunk size from one block per chunk to a size such that all blocks fit into one chunk. The maximal number of BDD nodes which have to be stored in memory simultaneously (peak number of nodes) is depicted in the left-hand side of Fig. 4 while the runtime is shown in the right-hand side.

We observe that for the `FPCS` and `Polling` benchmarks the memory requirement grows drastically as the chunk size increases. For `P2P`, the memory requirement slightly decreases or stays more or less constant depending on the configuration. That is because the compact representation for the `P2P` model accounts for the major part of the memory requirement of the algorithm, as reflected in Fig. 3.

Moreover, the runtime of the algorithm for all benchmarks decreases significantly when chunk size increases. The sensitivity of the runtime is milder for `P2P` examples because the chunk size has only little influence on the size of the partition representation.

Because of the optimization explained in Section 3.3, when *csize* is larger than the size of the bisimulation partition (i. e., the final result), the hybrid algorithm behaves exactly like the pure-FR algorithm (that only uses FR). That is the reason why the runtime and space requirement of the hybrid algorithm eventually stabilizes when the chunk size exceeds a specific threshold.

In general, the experiments show that we can indeed have a time–space trade-off. The smaller the chunk size, the less memory and the more time the algorithm consumes. That property enables us to have a simple and automatic chunk size selection algorithm, described in Section 3.3.

Evaluation of the automatic chunk size selection algorithm We ran the algorithm once without limiting the available memory, i. e., it was allowed to use as much physical memory as needed. In the other experiments, we limited the available memory to 75, 50, 30, and 15 MB for the MTBDDs. The results are shown in Table 1.

For each set of experiments, we give the running

Table 1: Runtime and memory consumption of the hybrid algorithm with automatic chunk size selection for different memory limitations

	unlimited memory			75 MB			50 MB			30 MB			15 MB		
Model	Time	Mem.	*cbits*	Time	Mem.	*cbits*	Time	Mem.	*cbits*	Time	Mem.	*cbits*	Time	Mem.	*cbits*
fpcs-2-1	0.04	10.25	11	0.04	10.25	11	0.04	10.25	11	0.04	10.25	11	0.04	10.25	11
fpcs-2-2	0.32	18.29	15	0.34	18.29	15	0.30	18.29	15	0.34	18.29	15	0.34	18.29	15
fpcs-2-3	1.62	50.02	19	1.56	50.02	19	1.54	50.02	19	1.62	38.64	19	19.41	19.40	**11**
fpcs-2-4	6.51	62.93	23	6.32	62.94	23	6.66	55.93	23	34.66	37.20	**11**	19.96	20.45	**9**
fpcs-2-5	44.83	96.32	27	149.03	79.66	**13**	40.81	65.00	**12**	102.74	41.56	**10**	memout		
fpcs-3-1	0.80	29.45	16	0.84	29.46	16	0.84	29.45	16	0.78	29.46	16	0.94	19.76	16
fpcs-3-2	19.78	105.66	22	64.41	80.08	**12**	28.57	63.08	**12**	47.06	40.41	**9**	30.94	25.05	**8**
fpcs-3-3	220.00	520.70	28	329.63	89.27	**10**	296.09	61.61	**8**	445.03	51.19	**7**	memout		
p2p-3-5	0.07	10.49	15	0.07	10.49	15	0.08	10.49	15	0.07	10.49	15	0.08	10.49	15
p2p-4-5	0.92	33.55	20	0.90	33.55	20	0.91	33.55	20	0.93	33.55	20	1.16	19.41	20
p2p-5-5	6.94	63.68	25	6.99	63.67	25	6.22	54.65	25	25.62	35.26	25	memout		
p2p-6-5	227.86	62.72	30	230.57	62.72	30	152.97	56.51	30	memout			memout		
polling-12	24.92	60.07	17	23.76	60.08	17	23.53	56.85	17	24.74	40.02	17	49.63	20.73	**11**
polling-13	169.73	116.95	18	228.19	83.13	**16**	248.88	64.42	**15**	237.77	36.66	**14**	490.47	20.61	**11**
polling-14	260.64	110.79	19	391.42	87.13	**14**	350.88	68.61	**13**	297.24	50.51	**12**	466.23	24.95	**9**
polling-15	710.72	231.48	20	789.63	85.47	**13**	802.45	66.48	**12**	826.32	48.64	**11**	memout		
polling-16	1868.24	433.97	21	1965.46	83.67	**12**	2525.20	68.51	**11**	3566.04	46.14	**8**	memout		
polling-18	15610.48	1814.29	24	memout			memout			memout			memout		

time, the memory usage of the nodes, and the number of chunk bits ($cbits = \lg_2 csize$) with which the bisimulation computation succeeded without violating the memory limit[1]. We have marked the number of chunk bits using a bold font if it was automatically decreased by the chunk size selection algorithm to adhere to the memory limit. An entry "mem out" means that the bisimulation computation failed since the memory limit was exceeded in spite of a chunk size of 1.

We observe that as we decrease the physical memory available to the algorithm, it adapts itself by decreasing the chunk size as much as necessary. This adaptation comes with a cost in the running time which in the worst case grows only by a small factor (< 2.5).

One would expect that the running time grows as the available memory decreases in each row of the table. However, there are a few exceptions. This is due to the caching behavior of the MTBDD package. If CUDD runs short of available memory, garbage collection is executed more frequently to free nodes which are no longer used. Furthermore, the size of internal caches is reduced to save memory. Both can influence the runtime in unpredictable ways. These memory saving strategies of CUDD are also the reason why sometimes less memory is used although the chunk size has not been decreased (see e. g., `polling-12`).

Comparing the hybrid algorithm with algorithms of [9] (pure FR-based) and [10] (pure CR-based) To have a fair comparison of the effectiveness of the three algorithms, we apply them to different models while we set an equal memory limit (75 MB) and time limit (2000 s) for all of them.

Table 2 shows the results of our experiments. We observe that the hybrid algorithm finishes several experiments successfully for which the FR-based and the CR-based algorithms fail due to memory limits and time limits, respectively. For all experiments that both hybrid and FR-based algorithms finish successfully, the hybrid algorithm is at most 2 % slower. Moreover, for all experiments that both hybrid and CR-based algorithms finish successfully, the hybrid algorithm is extremely faster than the CR-based one.

Thus, our hybrid algorithm provides us with the core advantages of both representations/algorithms with a negligible running time penalty.

5 Conclusion

In this paper, we have developed a general, fast and memory efficient algorithm for symbolic bisimulation minimization. The algorithm is presented in the context of continuous-time Markov chains, but is easily adaptable to labeled transition systems, Kripke structures, discrete-time or interactive Markov chains, etc.

The particular strength of this algorithm is that it exploits the true potential of BDD-based representation with respect to time *and* space, in a way that so far was unavailable. Based on experimental analysis of different partition representation techniques, we have devised an algorithm that (1) exploits the compactness of the CR representation and (2) uses the efficiency of the FR representation for an iterative signature-based refinement of partition chunks. The algorithm is parametric in the chunk size it processes at once. We have also devised and evaluated a strategy that automatically chooses an appropriate value for the chunk size. Thanks to our hybrid representation and automatic chunk size selection, severe memory limitations caused only a worst case slowdown by a factor of 2.5 in running time compared to unlimited available memory. Moreover, given the same memory limits, the hybrid algorithm works virtually as fast as pure-FR algorithm

[1] The actual memory requirement is slightly higher than the limit, since the limit is placed on the memory used by the MTBDDs. The memory requirement of the program code and of other data structures than MTBDDs are not taken into account for the limit.

Table 2: Comparison of the hybrid algorithm with the algorithms based only on FR [9] and on CR [10].

	pure FR-based algorithm		hybrid algorithm with automatic *csize*		pure CR-based algorithm	
Model	Runtime	Node peak	Runtime	Node peak	Runtime	Node peak
fpcs-2-1	0.04	36411	0.04	37228	0.14	8685
fpcs-2-2	0.36	192342	0.34	194100	2.43	30114
fpcs-2-3	1.56	646403	1.56	649118	27.31	56076
fpcs-2-4	6.24	1740601	6.32	1744309	148.07	153937
fpcs-2-5	mem out		149.03	3479910	669.45	365859
fpcs-3-1	0.82	429480	0.84	433907	9.41	74562
fpcs-3-2	mem out		64.41	3471734	785.16	501825
fpcs-3-3	mem out		329.63	3499328	time out	
p2p-3-5	0.07	35439	0.07	35548	0.95	39144
p2p-4-5	0.93	173396	0.90	173545	11.46	223080
p2p-5-5	6.84	502239	6.99	502428	88.75	684236
p2p-6-5	226.79	1064528	230.57	1064757	320.89	1554417
polling-12	23.51	853853	23.76	854632	175.13	129042
polling-13	mem out		228.19	3478888	time out	
polling-14	mem out		391.42	3507504	1788.61	452243
polling-15	mem out		789.63	3483998	time out	
polling-16	mem out		1965.46	3462536	time out	

while drastically outperforming the pure-CR algorithm for models for which pure-FR runs out of memory.

References

[1] I. Bahar, E. Frohm, C. Gaona, G. Hachtel, E. Macii, A. Pardo, and F. Somenzi. Algebraic decision diagrams and their applications. *FMSD*, 10(2/3):171–206, 1997.

[2] S. Blom and S. Orzan. Distributed branching bisimulation reduction of state spaces. In *Proc. of PDMC*, vol. 89 of *ENTCS*, pp. 99–113. Elsevier, 2003.

[3] S. Blom and S. Orzan. A distributed algorithm for strong bisimulation reduction of state spaces. *STTT*, 7(1):74–86, 2005.

[4] S. Blom and S. Orzan. Distributed state space minimization. *STTT*, 7(3):280–291, June 2005.

[5] E. Böde, M. Herbstritt, H. Hermanns, S. Johr, T. Peikenkamp, R. Pulungan, R. Wimmer, and B. Becker. Compositional performability evaluation for statemate. In *Proc. of QEST*, pp. 167–178, 2006.

[6] A. Bouali and R. de Simone. Symbolic bisimulation minimisation. In *Proc. of CAV*, vol. 663 of *LNCS*, pp. 96–108, 1992.

[7] E. Clarke, M. Fujita, P. McGeer, K. McMillan, J. Yang, and X. Zhao. Multiterminal binary decision diagrams: An efficient data structure for matrix representation. *FMSD*, 10(2/3):149–169, 1997.

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

[9] S. Derisavi. Signature-based symbolic algorithm for optimal Markov chain lumping. In *Proc. of QEST*, pp. 141–150, Sept. 2007.

[10] S. Derisavi. A symbolic algorithm for optimal Markov chain lumping. In *Proc. of TACAS*, LNCS, pp. 139–154, 2007.

[11] J.-C. Fernandez, H. Garavel, A. Kerbrat, R. Mateescu, L. Mounier, and M. Sighireanu. CADP: A Protocol Validation and Verification Toolbox. In *Proc. of CAV*, vol. 1102 of *LNCS*, pp. 437–440, 1996.

[12] K. Fisler and M. Y. Vardi. Bisimulation minimization and symbolic model checking. *FMSD*, 21(1):39–78, 2002.

[13] J. F. Groote and A. Ponse. The syntax and semantics of μCRL. In *Algebra of Communicating Processes '94*, Workshops in Computing Series, pp. 26–62, 1995.

[14] H. Hermanns and M. Siegle. Bisimulation algorithms for stochastic process algebras and their BDD-based implementation. In *5th Int'l AMAST Workshop on Real-Time and Probabilistic Systems (ARTS'99)*, vol. 1601 of *LNCS*, pp. 144–264, 1999.

[15] A. Hinton, M. Kwiatkowska, G. Norman, and D. Parker. PRISM: A tool for automatic verification of probabilistic systems. In *Proc. of TACAS*, vol. 3920 of *LNCS*, pp. 441–444, 2006.

[16] O. Ibe and K. Trivedi. Stochastic Petri net models of polling systems. *IEEE Journal on Selected Areas in Communications*, 8(9):1649–1657, 1990.

[17] P. C. Kanellakis and S. A. Smolka. CCS expressions, finite state processes, and three problems of equivalence. *Information and Computation*, 86(1):43–68, 1990.

[18] J.-P. Katoen, T. Kemna, I. Zapreev, and D. N. Jansen. Bisimulation minimization mostly speeds up probabilistic model checking. In *Proc. of TACAS*, vol. 4424 of *LNCS*, pp. 87–101, 2007.

[19] M. Kwiatkowska, G. Norman, and D. Parker. Symmetry reduction for probabilistic model checking. In *Proc. of CAV*, vol. 4114 of *LNCS*, pp. 234–248, 2006.

[20] K. G. Larsen and A. Skou. Bisimulation through probabilistic testing. In *16th Annual ACM Symposium on Principles of Programming Languages (POPL)*, pp. 344–352, 1989.

[21] R. Milner. A modal characterisation of observable machine-behaviour. In *6th Colloquium on Trees in Algebra and Programming (CAAP)*, vol. 112 of *LNCS*, pp. 25–34, 1981.

[22] R. Milner. Lectures on a calculus for communicating systems. In *Seminar on Concurrency*, vol. 197 of *LNCS*, pp. 197–220, 1985.

[23] R. Paige and R. E. Tarjan. Three partition refinement algorithms. *SIAM J. Comp.*, 16(6):973–989, 1987.

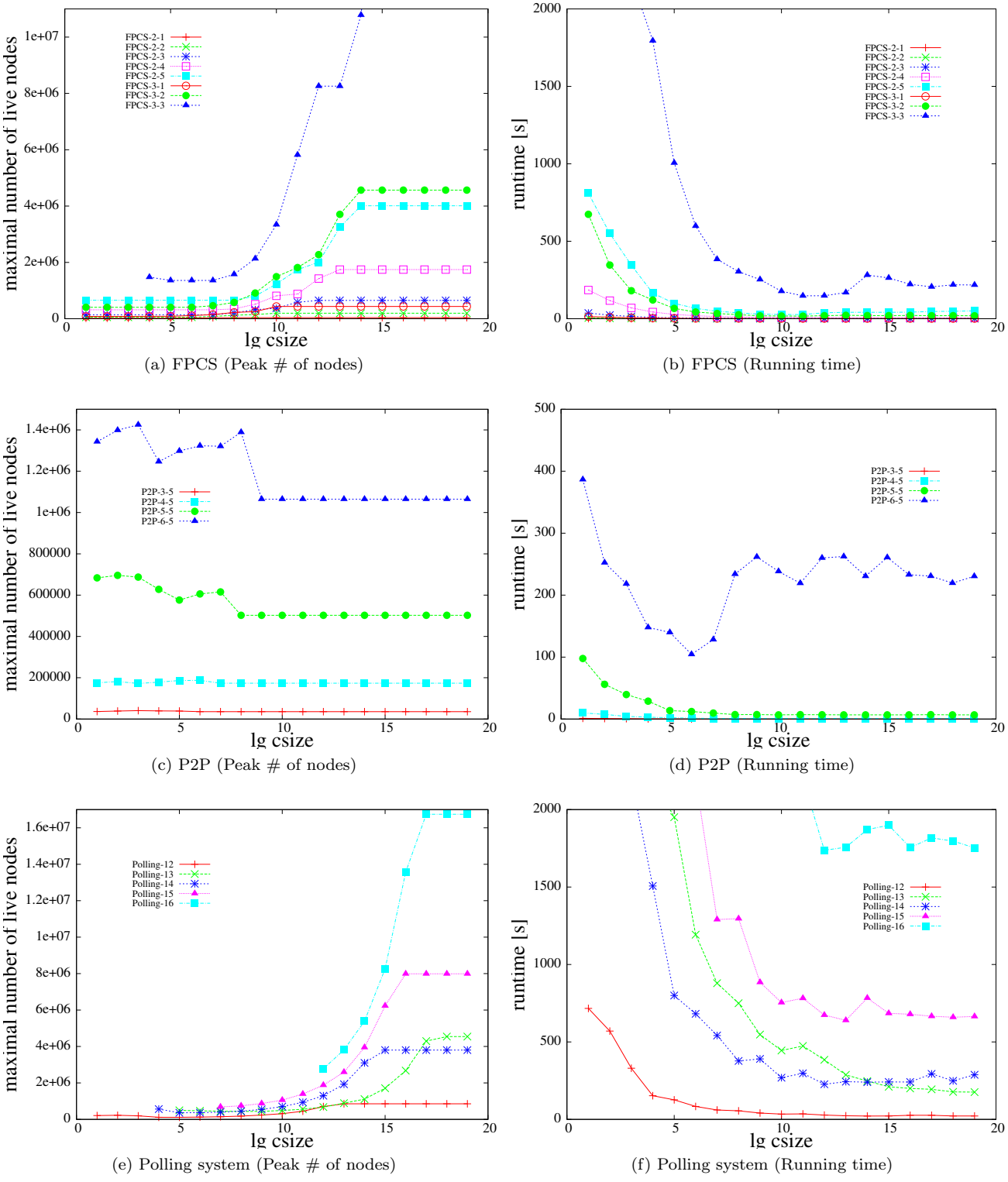
[24] D. Parker. *Implementation of Symbolic Model Checking for Probabilistic Systems*. Phd thesis, University of Birmingham, Great Britain, 2002.

[25] W. H. Sanders and L. M. Malhis. Dependability evaluation using composed SAN-based reward models. *J. Parallel Distr. Comp.*, 15(3):238–254, July 1992.

[26] F. Somenzi. CUDD: Colorado University decision diagram package. public software, Colorado University, Boulder, `http://vlsi.colorado.edu/~fabio/`, 2007.

[27] R. Wimmer, M. Herbstritt, and B. Becker. Optimization techniques for BDD-based bisimulation minimization. In *ACM Great Lakes Symposium on VLSI*, pp. 405–410, 2007.

[28] R. Wimmer, M. Herbstritt, H. Hermanns, K. Strampp, and B. Becker. Sigref – A symbolic bisimulation tool box. In *Proc. of ATVA*, vol. 4218 of *LNCS*, pp. 477–492, 2006.



(a) FPCS (Peak # of nodes)

(b) FPCS (Running time)

(c) P2P (Peak # of nodes)

(d) P2P (Running time)

(e) Polling system (Peak # of nodes)

(f) Polling system (Running time)

Figure 4: Maximal number of alive nodes and running times of the hybrid algorithm for different chunk sizes