Symmetry Reduction for Probabilistic Model Checking*

Marta Kwiatkowska, Gethin Norman, and David Parker

School of Computer Science, University of Birmingham,
Birmingham B15 2TT, United Kingdom
{mzk,gxn,dxp}@cs.bham.ac.uk

Abstract. We present an approach for applying symmetry reduction techniques to probabilistic model checking, a formal verification method for the quantitative analysis of systems with stochastic characteristics. We target systems with a set of non-trivial, but interchangeable, components such as those which commonly arise in randomised distributed algorithms or probabilistic communication protocols. We show, for three types of probabilistic models, that symmetry reduction, similarly to the non-probabilistic case, allows verification to instead be performed on a bisimilar quotient model which may be up to factorially smaller. We then propose an efficient algorithm for the construction of the quotient model using a symbolic implementation based on multi-terminal binary decision diagrams (MTBDDs) and, using four large case studies, demonstrate that this approach offers not only a dramatic increase in the size of probabilistic model which can be quantitatively analysed but also a significant decrease in the corresponding run-times.

1 Introduction

Probabilistic model checking is a formal verification technique for the analysis of systems which exhibit stochastic behaviour. It has been successfully applied to case studies from a wide range of application domains, including randomised distributed algorithms, communication and security protocols, dynamic power management schemes and biological systems. The key strength of the technique is the ability to automatically compute precise quantitative results based on an exhaustive analysis of a formal model. This allows reasoning about, for example, "the worst-case probability of system failure within T seconds" or "the minimum expected power consumption over all possible schedulings".

As with conventional model checking, the principal limiting factor with such techniques is the size of the models to be analysed. Although the development of symbolic implementations, which use binary decision diagrams (BDDs) and related data structures, have provided a significant increase in the applicability of the techniques, model size remains a major issue. In this paper we employ

* Supported in part by EPSRC grants GR/S11107 and GR/S46727 and Microsoft Research Cambridge contract MRL 2005-44.

symmetry reduction, a way of exploiting the presence of replication in a model which has yielded considerable success in non-probabilistic verification. In fact, this is particularly appealing in the context of probabilistic model checking since one of its more common, and indeed promising, applications is for the analysis of randomised distributed algorithms. These algorithms, which use electronic coin-flipping or random number generation, are increasingly used to provide elegant and efficient *symmetric* solutions to distributed coordination problems, as evidenced by the fact that they represent crucial components of many modern communication protocols such as Firewire, Bluetooth and Zeroconf.

We consider the case of *component symmetry*, in which any pair from a set of symmetric components in a model can be exchanged with no effect on the overall behaviour. We show that the key ideas of symmetry reduction in this case, namely that verification of a model can instead be performed on a bisimilar quotient model which is up to factorially smaller, carry across with relative ease to the probabilistic verification of three types of models: discrete-time Markov chains, continuous-time Markov chains and Markov decision processes.

We then propose an efficient algorithm for the construction of these quotient models which builds on the existing efficient symbolic implementations of probabilistic model checking in the tool PRISM [15, 24], based on the MTBDD (multi-terminal BDD) data structure [9, 3]. In doing so, we use ideas from the *dynamic symmetry reduction* technique of Emerson and Wahl [13]. Once the quotient model has been constructed, it can be analysed with existing algorithms and their implementations. Using experimental results from four large case studies, we show that our approach results in significant increases in both the size of models which can be verified and the speed with which this can be performed.

2 Background

2.1 Symmetry reduction

Symmetry reduction is a way of exploiting the occurrence of replication in a model. Consider a transition system $M = (S, R)$ comprising a (finite) set S of states and a transition relation $R \subseteq S \times S$. A permutation $\pi : S \to S$ on the state space is called an *automorphism* when it preserves the transition relation R, i.e. if $(s, s') \in R$, then $(\pi(s), \pi(s')) \in R$. Given a group G of such automorphisms under function composition, there is a corresponding equivalence relation θ on the set of states S where $(s, s') \in \theta$ if there is a permutation in G mapping s to s', i.e. if s and s' are symmetric. This relation θ is known as the *orbit relation* and its equivalence classes are called *orbits*.

If we then choose a set $\overline{S}$ containing a unique representative state for each equivalence class, we can define a function $rep : S \to \overline{S}$ which selects the corresponding unique representative $rep(s) \in \overline{S}$ for each state $s \in S$ and use this to induce a new transition relation $\overline{R} = \{(rep(s), rep(s')) \mid (s, s') \in R\}$. Since all permutations in G preserve the transition relation R, this *quotient transition system* $(\overline{S}, \overline{R})$ is bisimilar to the original transition system (S, R).

In many cases, this means that an analysis of the full system can instead be performed on the (smaller) quotient model. For example, if the states of the transition system (S, R) are labelled with atomic propositions from some set AP and we wish to model check a formula in the temporal logic CTL, then provided the atomic propositions in AP are preserved by the orbit relation, model checking can safely be carried out on the reduced model [11, 8].

In this paper, we consider *component symmetry* in which a model contains N symmetric components, any pair of which can be exchanged without any effect on the behaviour of the system. For a (global) state s of the model, we denote the local states of the N components by $s_1, \ldots, s_N$, respectively. A permutation π of S operates by mapping s to s' in which the values $s_1, \ldots, s_N$ are themselves permuted. Applying symmetry reduction to such a model where each of the N symmetric components has M local states provides, in the best case, a reduction in state space size from M^N to $\binom{M+N-1}{N}$, which tends towards $N!$ as M increases.

In the case of component symmetry, one way to define a unique representative for each set of symmetric states is to select the lexicographically least (given some ordering of the local states). This can be obtained by sorting the elements of the state. An alternative is to count the number of processes which are in each possible local state. Consider the case of 4 symmetric processes each with two local states A and B (where $A < B$). The states (A, B, A, A) and (A, A, B, A) are equivalent. Using the two schemes described above, they would both be mapped to (A, A, A, B) and $(A = 3, B = 1)$, respectively.

2.2 Symmetry reduction for symbolic model checking

The practicalities of exploiting symmetry reduction in the context of model checking have been studied at some length in the literature. Given a method of computing the representative $rep(s)$ for any state s it is possible, as shown by Ip and Dill [17], to compute the quotient transition system directly with a simple modification of a conventional, explicit state-space exploration algorithm: at every step of the exploration, each newly discovered state is immediately converted to its unique representative.

Clearly it would be desirable to combine *symbolic* model checking techniques, which have proven very successful for improving the efficiency of model checking, with symmetry reduction. Early results were discouraging: in [8], it was proved that for common types of symmetry, including component symmetry, the size of the BDD representing the orbit relation θ is exponential in either the number N of symmetric processes or the number M of local states that each can occupy. Progress was made in this area with several ways of reducing BDD sizes by, for example, allowing multiple representatives for each state [8], or attempting an under-approximation of reachability (aimed principally at falsification rather than verification) [5].

Recently, a promising approach known as *dynamic symmetry reduction* was proposed by Emerson and Wahl [13]. Their technique bypasses the construction of the orbit relation by only computing representatives for the set of states that are found during the process of model exploration (which, depending on the property, may not even be the set of all reachable states). This is achieved by sorting the components of all the currently explored states at each step. This procedure works using a bubble sort algorithm, applied directly to a BDD representation of the state set.

An alternative approach to symmetry reduction for systems exhibiting component symmetry, which is amenable to both explicit and symbolic model checking techniques, is *counter abstraction* [23] or *generic representatives* [12]. This is based on the idea of counting the number of processes in each local state, a process which in many cases can be carried out with a conversion of the model in the context of the high-level formalism in which it is described. Counter abstraction has been shown to perform particularly well, especially in situations where there are a large number of processes with a relatively small number of local states. For larger local state spaces, however, the exponential blow-up in variables required makes it very inefficient.

It should be noted that the work described above represents only a fraction of the available literature on symmetry reduction, the focus being on techniques for component symmetry and those targeted at a symbolic implementation.

2.3 Probabilistic model checking

Probabilistic model checking is an extension of model checking that is applied to transition systems augmented with information about the likelihood that each transition will occur. For a model with set of states S, its behaviour is specified not by a transition relation on S but a *transition function*. We most commonly deal with three types of probabilistic model: discrete-time Markov chains (DTMCs), continuous-time Markov chains (CTMCs) and Markov decision processes (MDPs). See e.g. [25] for an overview.

DTMCs are defined by a function $P : S \times S \to [0,1]$ satisfying $\sum_{s' \in S} P(s,s') = 1$ for each $s \in S$. This function, known as the *transition probability matrix*, gives the probability $P(s,s')$ of making a transition from each state s to any other state s'. DTMCs are typically used to represent synchronous systems with a discrete model of time. CTMCs, on the other hand, are defined by a *transition rate matrix* $R : S \times S \to \mathbb{R}_{\geq 0}$ giving the rate $R(s,s')$ at which transitions between state pairs (s,s') occur. This rate is interpreted as the parameter of a negative exponential distribution, resulting in a dense model of time. Lastly, MDPs are defined by a function $Steps : S \to 2^{Dist(S)}$ mapping each state s to a finite, nonempty set $Steps(s)$ of probability distributions over the state space S. Intuitively, this is interpreted as a nondeterministic choice between several probabilistic behaviours, and is thus useful for example to represent the asynchronous parallel composition of several stochastic processes.

Probabilistic model checking is applied to DTMCs, CTMCs and MDPs by formulating properties in the temporal logics PCTL and CSL which are probabilistic extensions of the logic CTL. This allows reasoning about, for example, "the probability of the algorithm terminating in error", "the probability that k packets are successfully transmitted within t seconds" or "the expected number of rounds for the protocol to complete". Because these are quantitative in nature, probabilistic model checking algorithms must perform numerical computation (typically iterative solution of linear equation systems or linear optimisation problems) in addition to the usual reachability-based algorithms.

Probabilistic model checking has been successfully applied to a large number of case studies from a wide range of application domains; see for example [20, 24]. In this paper, we have implemented our techniques using the open-source probabilistic model checker PRISM [15, 24].

2.4 Symbolic techniques for probabilistic model checking

Symbolic model checking techniques, i.e. those using BDDs or similar data structures have been successfully adapted to the field of probabilistic model checking. As in the non-probabilistic case, the key idea is that, by exploiting high-level structure and regularity in a model, it is possible to derive a very compact representation which can be efficiently manipulated. A popular data structure for this purpose, and the one used in the probabilistic model checker PRISM upon which our implementation is based, is multi-terminal BDDs (MTBDDs) [9, 3].

An MTBDD M is a directed acyclic graph, the nodes of which are labelled with Boolean variables from some set $\underline{\mathsf{x}} = \{\mathsf{x}_1, \ldots, \mathsf{x}_n\}$. The MTBDD represents a real-valued function $f_{\mathsf{M}}(\mathsf{x}_1, \ldots, \mathsf{x}_n) : \{0,1\}^n \to \mathbb{R}$. A BDD is thus a special case of an MTBDD which only maps to the two values 0 and 1. Given an encoding of the state space S of a probabilistic model into n Boolean variables, and two disjoint sets of n such variables $\underline{\mathsf{x}} = \{\mathsf{x}_1, \ldots, \mathsf{x}_n\}$ and $\underline{\mathsf{y}} = \{\mathsf{y}_1, \ldots, \mathsf{y}_n\}$, the transition probability/rate matrix of a DTMC or a CTMC can be represented as an MTBDD over these $2n$ variables. An MDP, although defined as a mapping to sets of probability distributions, can also be represented as a non-square $k{\cdot}|S| \times |S|$ matrix (where there are $k = \max_{s \in S} |\mathit{Steps}(s)|$ rows corresponding to each state) and represented in similar fashion [14].

Given a high-level description of a probabilistic model in some formalism, e.g. a stochastic process algebra or, as in our case, the PRISM modelling language, it is possible to construct the corresponding MTBDD directly, in a compositional fashion [14], often resulting in a very compact representation. Subsequently, we usually implement probabilistic model checking in one of two ways: (a) entirely with MTBDDs; or (b) using a combination of MTBDDs and explicit data structures such as sparse matrices and arrays [19]. In the PRISM tool, these are referred to as the "MTBDD" and "hybrid" engines, respectively. The former, where applicable, has successfully been applied to huge probabilistic models (see e.g. [21]) but this is highly dependent on model regularity and in many cases, where irregularities introduced during numerical solution become a factor, has infeasible time and/or memory usage. The "hybrid" approach is usually faster but, due to storage requirements linear in the size of the state space, is generally limited to models of approximately 10^7–10^8 states. It is frequently the case that compact symbolic representations can be constructed for extremely large probabilistic models, but that verification, in particular via numerical computation, of these models is prohibitively expensive, both in terms of time and space.

3 Symmetry reduction for probabilistic model checking

The notion of symmetry in DTMCs, CTMCs and MDPs can be formulated analogously to the non-probabilistic case, described in Section 2.1. We consider permutations $\pi : S \to S$ on the state space which preserve the transition function. For DTMCs, we require that $P(\pi(s), \pi(s')) = P(s, s')$ for all $s, s' \in S$. Similarly, for CTMCs, we need $R(\pi(s), \pi(s')) = R(s, s')$ for all $s, s' \in S$. In the case of MDPs, for each $s \in S$ and each distribution $\mu \in \mathit{Steps}(s)$, there must be a distribution $\mu' \in \mathit{Steps}(\pi(s))$ such that $\mu'(\pi(s')) = \mu(s')$ for all $s' \in S$.

As before, we consider a group G of such permutations on S and the corresponding orbit relation θ. From the equivalence classes of the latter we define a reduced state space $\overline{S}$ containing a unique representative for each orbit and a function $\mathit{rep} : S \to \overline{S}$ which computes the representative for each state. Construction of the quotient model can then be carried out as follows. For a DTMC (S, P), we build $(\overline{S}, \overline{P})$ where for each pair of states $\overline{s}, \overline{s}' \in \overline{S}$:

$$\overline{P}(\overline{s}, \overline{s}') = \sum_{\{s' \in S \;\mid\; \mathit{rep}(s') = \overline{s}'\}} P(\overline{s}, s')\,. \tag{1}$$

For a CTMC (S, R), the quotient model is $(\overline{S}, \overline{R})$ where for $\overline{s}, \overline{s}' \in \overline{S}$:

$$\overline{R}(\overline{s}, \overline{s}') = \sum_{\{s' \in S \;\mid\; \mathit{rep}(s') = \overline{s}'\}} R(\overline{s}, s')\,. \tag{2}$$

Finally, for an MDP (S, Steps), the quotient model is $(\overline{S}, \overline{\mathit{Steps}})$ where if $\overline{s} \in \overline{S}$, then $\overline{\mathit{Steps}}(\overline{s})$ contains a distribution $\overline{\mu}$ if and only if there exists $\mu \in \mathit{Steps}(\overline{s})$ such that for each $\overline{s}' \in \overline{S}$:

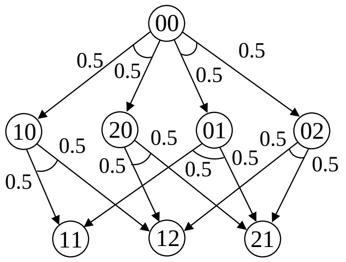
$$\overline{\mu}(\overline{s}') = \sum_{\{s' \in S \;\mid\; \mathit{rep}(s') = \overline{s}'\}} \mu(s')\,. \tag{3}$$

Constructed in such a way, the quotient model in each of the three cases can easily be shown to be equivalent to the original unreduced model in the context of (strong) probabilistic bisimulation, which is well understood for DTMCs [22], CTMCs [6] and MDPs [26]. Furthermore, from results in [2, 4, 26], we can hence deduce that for formulas in the temporal logics PCTL (for DTMCs or MDPs) and CSL (for CTMCs) using a set of atomic propositions AP which is preserved by symmetry, probabilistic model checking can be performed equivalently on the quotient model rather than the original, unreduced model. Additionally, quantitative analysis of cost- and reward-based specifications are similarly preserved.

3.1 Example

We now consider a simple illustrative example: an MDP representing two symmetric processes with three states (0, 1 and 2) which interact as follows. Initially, both are in state 0. First, one moves randomly to state 1 or 2 (each with probability 0.5). The other then does likewise. If, in the second step, state 2 is chosen and a process is already in state 2, the one which moved first moves to state 1.

(a)



(b)

$rep : S \to \overline{S}$
$00 \to 00$
$10 \to 01$
$20 \to 02$
$01 \to 01$
$02 \to 02$
$11 \to 11$
$12 \to 12$
$21 \to 12$

(c)

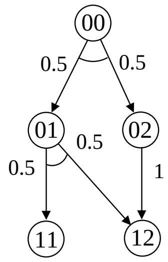


Fig. 1. A simple example: (a) full MDP (b) state representatives (c) quotient MDP

Figure 1(a) shows the MDP. States are denoted by circles labelled ij, where i and j represent the state of process 1 and 2, respectively. Each of the possible probability distributions from a state is denoted by a set of probability-labelled transitions (arrows) grouped by an arc. Figure 1(b) gives the function *rep* which maps each state from the MDP to a unique symmetric representative and Figure 1(c) shows the resulting quotient model over the reduced state space.

4 A symbolic implementation

We now consider in more detail how to exploit symmetry reduction for probabilistic model checking. More precisely, we consider the problem of constructing, for a DTMC, CTMC or MDP exhibiting component symmetry, the corresponding quotient model upon which probabilistic model checking can instead be performed. Although the method of Ip and Dill [17] can readily be adapted for this purpose, it is only applicable only to the construction of models in an explicit fashion. As stated earlier, we wish to instead build upon the efficient *symbolic* model checking framework that already exists.

One way that this could be achieved is to use counter abstraction, which works by applying symmetry reduction at a higher level: on the language description of the model. Preliminary results for the application of such techniques to probabilistic model checking can be found in [10]. An advantage of this is that existing efficient methods for the construction of a symbolic model can be used unmodified; however it is known, from applications of this to non-probabilistic model checking, that performance is typically poor for examples where symmetric processes have many local states.

Previous experience with symbolic methods for probabilistic model checking suggests that the most desirable way of constructing an MTBDD representing the quotient model would be to do so directly from a high-level model description and in a compositional manner. Unfortunately, there is no obvious way of achieving this due to the implicit introduction, during symmetry reduction, of a large number of inter-component dependencies. Hence, our approach proceeds by first building a symbolic representation for the full model and then reducing it to the quotient model. Working in an explicit context, this would negate any benefit of symmetry reduction, since constructing and storing the model itself represents the bottleneck. In a symbolic setting, though, this is not so: it is very often the case that it is possible to build the MTBDD representing extremely large models but that model checking cannot be carried out, either in a fully symbolic fashion (because irregularity in the computation causes a blow-up in MTBDD sizes) or with the use of explicit data structures (because of excessive memory requirements). It is in such cases that symmetry reduction has the potential to be very useful. In Section 5 we will demonstrate exactly this on a number of large case studies.

4.1 The algorithm

We now present our algorithm for converting a probabilistic model with N symmetric components to its reduced quotient form which is applicable to all three model types: DTMCs, CTMCs and MDPs. In the following we shall refer to the *transition matrix*, which corresponds to P, R and *Steps*, respectively, for the three models. As described in Section 2.4, for an MDP, *Steps* can be thought of as non-square matrix where there are multiple rows corresponding to each state.

Our algorithm proceeds in three steps. Firstly, we identify the quotient state space: the set $\overline{S}$. Secondly, every row in the transition matrix corresponding to states not in $\overline{S}$ is removed (replaced with a row of zeros). Thirdly, we modify each of the remaining rows, moving entries in column s' to column $rep(s')$. More precisely, since a row may contain multiple non-zero entries which move to the same column, we replace each element of row s of the matrix with the sum of the values from columns s' for which $s = rep(s')$. Note that this corresponds precisely to our three definitions of the quotient model in Section 3.

The full algorithm, expressed in terms of the operations on BDDs and MTBDDs, is shown in Figure 2. The inputs to the algorithm are a BDD reach over set of variables $\underline{\mathsf{x}}$ representing the set of reachable states S and an MTBDD trans over the variable sets $\underline{\mathsf{x}}$ and $\underline{\mathsf{y}}$ representing the transition matrix (see Section 2.4). Since this MTBDD is constructed in a compositional fashion, the variable sets $\underline{\mathsf{x}}$ and $\underline{\mathsf{y}}$ are conveniently partitioned into subsets $\underline{\mathsf{x}_\mathsf{i}}$ and $\underline{\mathsf{y}_\mathsf{i}}$ for $1 \leq i \leq N$, one for each of the N symmetric processes. We can illustrate this as follows. Let us assume that the state space S of the model is simply the product of the (identical) local state spaces $\hat{S}$ of each of the N symmetric components, i.e. $S = \hat{S}^N$. We have both an encoding $\alpha : S \to \{0,1\}^n$ of the whole state space S into n Boolean variables and an encoding $\hat{\alpha} : \hat{S} \to \{0,1\}^{n/N}$ of the local state space $\hat{S}$ into Boolean variables. Consider for example an MTBDD M over variables $\underline{\mathsf{x}} = \{\mathsf{x}_1, \ldots, \mathsf{x}_n\}$. This MTBDD represents a real-valued function over the state space S which, for a state $s = (s_1, \ldots, s_N)$ in S, can be defined either as $f_\mathsf{M}(\alpha(s))$ or, equivalently, $f_\mathsf{M}(\hat{\alpha}(s_1), \ldots, \hat{\alpha}(s_N))$.

The output of the algorithm is a modified copy of the trans MTBDD representing the quotient transition matrix. This MTBDD is also over the variable sets $\underline{\mathsf{x}}$ and $\underline{\mathsf{y}}$, meaning that we retain the same encoding of the state space and remove (set to zero) all entries of rows and columns of the matrix corresponding to states not in $\overline{S}$, rather than select a new encoding for the set $\overline{S}$. This is important in order to preserve the regularity of the original data structure.

```
// Step 1: Identify quotient state space, i.e. all representative states
1. repr := reach
2. for (i := 1..N − 1)
3.     sorted_i := VariablesLessThanEquals(y_i, y_i+1) ∧ reach
4.     repr := repr ∧ sorted_i

// Step 2: Remove rows corresponding to non-representative states
5. trans := Apply(×, trans, SwapVariables(repr, y, x))

// Step 3: Construct matrix for quotient model via bubble sort
6. for (i := N, . . . , 2)
7.     trans_prev := trans
8.     for (j := 1, . . . , i−1)
9.         good := Apply(×, trans, sorted_j)
10.        bad := Apply(×, trans, ¬ sorted_j)
11.        fixed := SwapVariables(bad, y_j, y_j+1)
12.        trans := Apply(+, good, fixed)
13.    if (trans = trans_prev) return trans
14. return trans
```

Fig. 2. MTBDD-based algorithm for computing the quotient model

In the first step of the algorithm (lines 1-4) we compute the BDD repr representing the set $\overline{S}$. This is in fact relatively easy. Since our unique representative function *rep* simply sorts the values of symmetric components, $\overline{S}$ is the subset of S containing states s in which $s_1 \leq s_2 \leq \cdots \leq s_N$. Hence symm is formed from the conjunction of reach and $N-1$ BDDs, the ith of which, denoted sorted_i, encodes all states in which $s_i \leq s_{i+1}$. Given the BDD variable sets $\underline{\mathsf{y}_\mathsf{i}}$ and $\underline{\mathsf{y}_\mathsf{i+1}}$, encoding the ith and $(i+1)$th processes, the latter are small (linear in the size of $\underline{\mathsf{y}_\mathsf{i}}$) and easy to construct (in the CUDD library which we use, there is a built-in function to do so).

The second step of the algorithm (line 5) requires a single application of the Apply function, a fundamental MTBDD operation which performs a pointwise application of a binary function on two MTBDDs. We use a pointwise multiplication of the BDD (0-1 valued MTBDD) repr and MTBDD trans. Since we are removing rows from the matrix, not columns, we first swap the variables in repr from $\underline{\mathsf{y}}$ to $\underline{\mathsf{x}}$.

The third and most important step (lines 6-14) permutes and sums matrix elements to compute the MTBDD for the quotient matrix. Essentially, each element in column s of the matrix needs to move to column s' where the latter can be determined by sorting the components $(s_1, \ldots, s_N)$ of s into ascending numerical order. We do this using a standard bubble sort algorithm comprising $N-1$ passes, the ith of which compares s_j and s_{j+1} for all $j<i$ and swaps the two values if $s_j > s_{j+1}$. The novelty is that we perform this sorting process for all elements of each row simultaneously and, furthermore, for all rows simulta-

$$
\text{(a)}\quad
\begin{array}{cc}
 & \begin{array}{cccccccc} 00 & 01 & 02 & 10 & 11 & 12 & 20 & 21 \end{array} \\
\begin{array}{c} 00 \\ \\ 01 \\ 02 \\ 10 \\ 11 \\ 12 \\ 20 \\ 21 \end{array} &
\begin{pmatrix}
\cdot & 0.5 & 0.5 & \cdot & \cdot & \cdot & \cdot & \cdot \\
\cdot & \cdot & \cdot & 0.5 & \cdot & \cdot & 0.5 & \cdot \\
\cdot & \cdot & \cdot & \cdot & 0.5 & \cdot & \cdot & 0.5 \\
\cdot & \cdot & \cdot & \cdot & \cdot & 0.5 & \cdot & 0.5 \\
\cdot & \cdot & \cdot & \cdot & 0.5 & 0.5 & \cdot & \cdot \\
\cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\
\cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\
\cdot & \cdot & \cdot & \cdot & \cdot & 0.5 & \cdot & 0.5 \\
\cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot
\end{pmatrix}
\end{array}
\qquad
\text{(b)}\quad
\begin{array}{cc}
 & \begin{array}{cccccccc} 00 & 01 & 02 & 10 & 11 & 12 & 20 & 21 \end{array} \\
\begin{array}{c} 00 \\ \\ 01 \\ 02 \\ 10 \\ 11 \\ 12 \\ 20 \\ 21 \end{array} &
\begin{pmatrix}
\cdot & 0.5 & 0.5 & \cdot & \cdot & \cdot & \cdot & \cdot \\
\cdot & \cdot & \cdot & 0.5 & \cdot & \cdot & 0.5 & \cdot \\
\cdot & \cdot & \cdot & \cdot & 0.5 & \cdot & \cdot & 0.5 \\
\cdot & \cdot & \cdot & \cdot & \cdot & 0.5 & \cdot & 0.5 \\
\cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\
\cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\
\cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\
\cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\
\cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot
\end{pmatrix}
\end{array}
$$

$$
\text{(c)}\quad
\begin{array}{cc}
 & \begin{array}{cccccccc} 00 & 01 & 02 & 10 & 11 & 12 & 20 & 21 \end{array} \\
\begin{array}{c} 00 \\ \\ 01 \\ 02 \\ 10 \\ 11 \\ 12 \\ 20 \\ 21 \end{array} &
\begin{pmatrix}
\cdot & 0.5 & 0.5 & \cdot & \cdot & \cdot & \cdot & \cdot \\
\cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\
\cdot & \cdot & \cdot & \cdot & 0.5 & \cdot & \cdot & \cdot \\
\cdot & \cdot & \cdot & \cdot & \cdot & 0.5 & \cdot & \cdot \\
\cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\
\cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\
\cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\
\cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\
\cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot
\end{pmatrix}
\end{array}
\qquad
\text{(d)}\quad
\begin{array}{cc}
 & \begin{array}{cccccccc} 00 & 01 & 02 & 10 & 11 & 12 & 20 & 21 \end{array} \\
\begin{array}{c} 00 \\ \\ 01 \\ 02 \\ 10 \\ 11 \\ 12 \\ 20 \\ 21 \end{array} &
\begin{pmatrix}
\cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\
\cdot & \cdot & \cdot & 0.5 & \cdot & \cdot & 0.5 & \cdot \\
\cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & 0.5 \\
\cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & 0.5 \\
\cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\
\cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\
\cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\
\cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\
\cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot
\end{pmatrix}
\end{array}
$$

$$
\text{(e)}\quad
\begin{array}{cc}
 & \begin{array}{cccccccc} 00 & 01 & 02 & 10 & 11 & 12 & 20 & 21 \end{array} \\
\begin{array}{c} 00 \\ \\ 01 \\ 02 \\ 10 \\ 11 \\ 12 \\ 20 \\ 21 \end{array} &
\begin{pmatrix}
\cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\
\cdot & 0.5 & 0.5 & \cdot & \cdot & \cdot & \cdot & \cdot \\
\cdot & \cdot & \cdot & \cdot & \cdot & 0.5 & \cdot & \cdot \\
\cdot & \cdot & \cdot & \cdot & \cdot & 0.5 & \cdot & \cdot \\
\cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\
\cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\
\cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\
\cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\
\cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot
\end{pmatrix}
\end{array}
\qquad
\text{(f)}\quad
\begin{array}{cc}
 & \begin{array}{cccccccc} 00 & 01 & 02 & 10 & 11 & 12 & 20 & 21 \end{array} \\
\begin{array}{c} 00 \\ \\ 01 \\ 02 \\ 10 \\ 11 \\ 12 \\ 20 \\ 21 \end{array} &
\begin{pmatrix}
\cdot & 0.5 & 0.5 & \cdot & \cdot & \cdot & \cdot & \cdot \\
\cdot & 0.5 & 0.5 & \cdot & \cdot & \cdot & \cdot & \cdot \\
\cdot & \cdot & \cdot & \cdot & 0.5 & 0.5 & \cdot & \cdot \\
\cdot & \cdot & \cdot & \cdot & \cdot & 1 & \cdot & \cdot \\
\cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\
\cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\
\cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\
\cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\
\cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot
\end{pmatrix}
\end{array}
$$

Fig. 3. Executing the algorithm from Figure 2 on the MDP from Figure 1(a): (a) trans (initially) (b) trans (after step 2) (c) good (d) bad (e) fixed (f) trans (finally)

neously. This concurrent approach to algorithm design is typical of symbolic, i.e. BDD-based, implementations.

In terms of MTBDDs, each iteration of bubble sort, as described above, is achieved as follows. We partition the current copy of the matrix trans into two parts, good and bad, the latter containing all columns of the matrix for which in the column index s, $s_j > s_{j+1}$. We can then use the primitive MTBDD operation SwapVariables and the variable sets $\underline{\mathsf{y_j}}$ and $\underline{\mathsf{y_{j+1}}}$ to convert bad to an equivalent MTBDD fixed in which s_j and s_{j+1} have been swapped for all column indices. Summing (using Apply) the MTBDDs good and fixed gives the new version of trans for this iteration.

A feature of bubble sort is that, if no swaps are performed in an entire outer iteration, then we can deduce that the sort has been completed early. Thanks to the canonicity property of MTBDDs, it is trivial to compare two MTBDDs for equality. Hence, for each outer iteration of our algorithm, we check whether the MTBDD trans has been modified (lines 7 and 13), terminating early if not.

4.2 Example

We now return to our example of Section 3.1 (Figure 1). Figure 3 illustrates the execution of the algorithm from Figure 2 on the MDP from Figure 1(a). Figure 3(a) shows the matrix trans for the full MDP. Figure 3(b) shows trans after step 2 (line 5 of Figure 2). Since this example comprises just two processes, we require only a single iteration of bubble sort. The matrices good, bad and fixed from this iteration are shown in Figures 3(c), 3(d) and 3(e), respectively. The final resulting version of trans (i.e. good + fixed) is shown in Figure 3(f) which corresponds exactly to the quotient MDP in Figure 1(c).

4.3 Efficiency of the algorithm

Clearly, the most costly part of our method is the sorting process. As mentioned previously our decision to use the bubble sort algorithm for this is motivated by [13], where it is argued that, despite being a poor choice in general, bubble sort is well suited for a BDD-based implementation. Firstly, it allows each step of the algorithm to be applied in parallel to several separate sorting problems: in the case of [13], values for all states in a set are sorted simultaneously, whereas in our case we take this a step further, sorting values in all rows and columns of an entire real-valued matrix. Secondly, the most costly operation at the BDD or MTBDD level is the permutation of two sets of Boolean variables, the performance of which worsens with an increase in the distance between the two sets in the variable ordering. Bubble sort has the advantage that it only swaps the variables corresponding to adjacent processes which are very close in the ordering.

The complexity of bubble sort is $O(N^2)$. Since model sizes are typically exponential in the number of processes they contain, N will usually be comparatively small. More crucial from a performance point of view will be the effect that the sorting process will have on the size of the MTBDD representation of the matrix. The unavoidable loss in regularity associated with this process would be expected to have at least some detrimental effect in this respect but, as is always the case with BDD and MTBDD-based implementations, the exact effect is hard to predict. We will give an analysis based on empirical data in the next section.

5 Results

We now present experimental results to illustrate the performance of our approach, which has been implemented as a prototype extension of the PRISM probabilistic model checker [15, 24] using the CUDD BDD/MTBDD library [27]. For this, we have used four case studies: the IEEE 802.3 CSMA/CD communication protocol [16], the shared coin component of Aspnes and Herlihy's randomised consensus protocol [1], studied in [21]; the randomised Byzantine agreement protocol of Cachin, Kursawe and Shoup [7], studied in [18]; and a simple peer-to-peer (P2P) protocol based on BitTorrent. The first three have been modelled as MDPs, the latter as a CTMC. In all cases, we model check a single quantitative property. All PRISM models and properties are available [24]. At this stage, symmetry has been identified manually; future work will involve modifying the PRISM input language to facilitate automation of this process.

The table in Figure 4 summarises the results of our experiments, which were executed on a 2.80GHz Pentium 4 PC with 1GB RAM running Linux. The first two columns show the range of values of N (number of symmetric processes) for which we have obtained results with each case study. The next three columns show the sizes of the full and symmetry-reduced state spaces (i.e. $|S|$ and $|\overline{S}|$) and the factor of reduction achieved. In the next two columns, we give the size of the MTBDD (number of nodes) representing the transition matrix for the original model and the reduced quotient model.

Case study	N	State space			MTBDD size (nodes)		Time (sec.)			
		Size		Reduct. factor			Model build	Model sort	Model checking	
		Normal	Symm.		Normal	Symm.			Normal*	Symm.*
CSMA	4	678,831	35,270	19.2	64,857	37,653	9.25	1.05	134 (H)	26.5 (H)
	5	1.7e+7	203,271	81.9	207,837	105,768	45.3	4.17	1,988 (H)	131 (H)
	6	3.2e+8	813,520	392	538,931	244,970	158	13.1	21,511 (M)	491 (H)
	7	5.7e+9	2.8e+6	2,073	1.1e+6	510,618	499	31.5	Mem. out	1,934 (H)
	8	1.0e+11	8.4e+6	11,962	2.1e+6	1.1e+6	1,831	76.8	Mem. out	5,281 (H)
Consensus	8	6.1e+7	46,482	1,313	15,529	15,883	1.41	0.46	5,447 (M)	74.6 (H)
	10	2.8e+9	136,708	20,198	29,419	29,939	3.01	1.02	27,668 (M)	412 (H)
	12	1.2e+11	339,729	352,407	50,037	50,741	5.71	2.11	> 24 hrs	2,047 (H)
	14	5.0e+12	747,243	6.7e+6	78,171	79,123	9.59	5.05	> 24 hrs	6,816 (H)
	16	2.1e+14	1.5e+6	1.4e+8	115,385	116,691	19.0	7.56	> 24 hrs	19,168 (H)
Byzantine	8	6.4e+8	298,993	2,142	713,143	167,587	21.0	5.15	111 (M)	10.8 (M)
	12	3.6e+12	3.3e+6	1.1e+6	2.6e+6	529,619	86.0	23.9	1,430 (M)	56.6 (M)
	16	1.9e+16	1.2e+7	9.3e+8	6.6e+6	1.2e+6	241	75.3	Mem. out	186 (M)
	20	9.5e+19	9.1e+7	1.0e+12	1.3e+7	2.3e+6	1,503	237	Mem. out	740 (M)
P2P	4	1.0e+6	52,360	20.0	2,735	28,684	0.10	0.34	34.1 (H)	8.75 (H)
	5	3.4e+7	376,992	89.0	12,230	67,764	0.08	0.88	1,756 (H)	70.1 (H)
	6	1.1e+9	2.3e+6	462	26,555	134,641	0.16	1.77	Mem. out	547 (H)
	7	3.4e+10	1.3e+7	2,723	40,880	233,068	0.28	3.38	Mem. out	3,556 (H)

* (M) denotes use of MTBDD engine, (H) denotes use of hybrid engine

Fig. 4. Experimental results for the four case studies

We first note that, as expected, we can obtain large (several orders of magnitude) reductions in state space size using component symmetry and that these reductions improve with an increase in N. Of equal importance, however, is the size of the MTBDD representation of the quotient model. Our result shows that in some cases this is actually smaller (approximately half the size for CSMA, even smaller for Byzantine), in others there is little change (Consensus) but that there can also be a significant increase in size (P2P). There are two contrasting factors which influence this. Firstly, the removal of a large number of rows decreases MTBDD size. The BDD representing the set of representative states (repr in Figure 2) is normally quite regular, unlike for example the set of all reachable states. On the other hand, the permutation of a large number of matrix elements during sorting destroys a great deal of the matrix's regularity and hence increases MTBDD size. This is confirmed by a more detailed analysis of the MTBDD size during the process of constructing the quotient model which usually exhibits an initial decrease followed by a steady increase.

We thus observe that the example exhibiting the most regularity before symmetry, P2P (compare the MTBDD size and state space), suffers the highest increase. Although the Consensus and Byzantine examples are also regular, the fact that we can work with larger values of N results in a greater reduction in state space and hence the gain from removal of rows outweighs the loss in structure. Similarly, we can see that the effect of loss in regularity (increase in MTBDD size) decreases with increasing N on the P2P example.

The last four columns in Figure 4 show the times taken for each part of our experiments: building the original full model (including reachability), construction of the quotient model via bubble sort and performing model checking on each of the two models. In the latter case we use the fastest PRISM engine which can complete the task: either the MTBDD (M) or hybrid (H) engine. All experiments requiring more than 24 hours were discounted.

We see that the times for constructing the quotient model (i.e. to run the algorithm in Figure 2), although slower on the CSMA and Byzantine examples for which the MTBDDs are larger, are generally fast, certainly with respect to the time required for model checking. More significantly, we observe that using symmetry reduction we can obtain a dramatic increase in the size of models that can be successfully verified. Note that this is true even on the P2P example for which the MTBDD sizes increase. For three of our case studies, this is because the large reduction in state space makes use of the hybrid engine possible after symmetry reduction. Another consequence of this is that we also observe a significant improvement in run-time using symmetry reduction. On the Consensus case study, the total time required for solution drops from several weeks to a few hours. For the Byzantine case study, where we use the MTBDD engine both before and after reduction, the decrease in MTBDD size means that we still obtain huge improvements in both state space and run-time.

6 Conclusion

We have presented an efficient approach for the exploitation of component symmetry in three types of probabilistic models: discrete-time Markov chains, continuous time Markov chains and Markov decision processes. Our algorithm, based on multi-terminal BDDs and implemented in the PRISM tool, has been applied to four large case studies and demonstrates that the technique performs extremely well. In comparison to existing state-of-the-art implementations, it allows quantitative verification to be performed on models many orders of magnitude larger and results in significantly faster run-times.

Our work can be extended in several directions. Following ideas put forward in [13], we hope to able to apply our technique to a wider range of symmetric systems, including those based on rotational, rather than component, symmetry (i.e. rings). We would also like to extend the PRISM modelling language with a notion of scalarsets [17] in order to facilitate the automation of detecting and exploiting symmetry. Finally, we hope to undertake a comparison with symmetry reduction methods based on counter abstraction [10].

References

[1] J. Aspnes and M. Herlihy. Fast randomized consensus using shared memory. *Journal of Algorithms*, 15(1):441–460, 1990.

[2] A. Aziz, V. Singhal, F. Balarin, R. Brayton, and A. Sangiovanni-Vincentelli. It usually works: The temporal logic of stochastic systems. In *Proc. CAV'95*, 1995.

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

[4] C. Baier, B. Haverkort, H. Hermanns, and J.-P. Katoen. Model checking continuous-time Markov chains by transient analysis. In *Proc. CAV'00*, 2000.

[5] S. Barner and O. Grumberg. Combining symmetry reduction and under-approximation for symbolic model checking. In *Proc. CAV'02*, 2002.

[6] P. Buchholz. Markovian process algebra: Composition and equivalence. In *Proc. PAPM'94*, pages 11–30, 1994.
[7] C. Cachin, K. Kursawe, and V. Shoup. Random oracles in Constantinople: Practical asynchronous Byzantine agreement using cryptography (extended abstract). In *Proc. Symposium on Principles of Distributed Computing*, pages 123–132, 2000.
[8] E. Clarke, R. Enders, T. Filkorn, and S. Jha. Exploiting symmetry in temporal logic model checking. *Formal Methods in System Design*, 9(1/2):77–104, 1996.
[9] E. Clarke, M. Fujita, P. McGeer, K. McMillan, J. Yang, and X. Zhao. Multi-terminal binary decision diagrams: An efficient data structure for matrix representation. *Formal Methods in System Design*, 10((2/3):149–169, 1997.
[10] A. Donaldson and A. Miller. Symmetry reduction for probabilistic systems. In *Proc. 12th workshop on Automated Reasoning*, pages 17–18, 2005.
[11] E. Emerson and A. Sistla. Symmetry and model checking. *Formal Methods in System Design*, 9(1/2):105–131, 1996.
[12] E. Emerson and R. Trefler. From asymmetry to full symmetry: New techniques for symmetry reduction in model checking. In *Proc. CHARME'99*, 1999.
[13] E. Emerson and T. Wahl. Dynamic symmetry reduction. In *Proc. TACAS'05*, volume 3440 of *LNCS*, pages 382–396. Springer, 2005.
[14] H. Hermanns, M. Kwiatkowska, G. Norman, D. Parker, and M. Siegle. On the use of MTBDDs for performability analysis and verification of stochastic systems. *Journal of Logic and Algebraic Programming*, 56(1-2):23–67, 2003.
[15] A. Hinton, M. Kwiatkowska, G. Norman, and D. Parker. PRISM: A tool for automatic verification of probabilistic systems. In *TACAS'06*, volume 3920 of *LNCS*, pages 441–444. Springer, 2006.
[16] IEEE 802.3-2002. IEEE Standard for Carrier Sense Multiple Access with Collision Detection (CSMA/CD). http://standards.ieee.org/getieee802/802.3.html, 2002.
[17] C. Ip and D. Dill. Better verification through symmetry. *Formal Methods In System Design*, 9(1-2):41–75, 1996.
[18] M. Kwiatkowska and G. Norman. Verifying randomized Byzantine agreement. In *Proc. FORTE'02*, volume 2529 of *LNCS*, pages 194–209. Springer, 2002.
[19] M. Kwiatkowska, G. Norman, and D. Parker. Probabilistic symbolic model checking with PRISM: A hybrid approach. *International Journal on Software Tools for Technology Transfer (STTT)*, 6(2):128–142, 2004.
[20] M. Kwiatkowska, G. Norman, and D. Parker. Probabilistic model checking in practice: Case studies with PRISM. *ACM SIGMETRICS Performance Evaluation Review*, 32(4):16–21, 2005.
[21] M. Kwiatkowska, G. Norman, and R. Segala. Automated verification of a randomized distributed consensus protocol using Cadence SMV and PRISM. In *Proc. CAV'01*, volume 2102 of *LNCS*, pages 194–206. Springer, 2001.
[22] K. Larsen and A. Skou. Bisimulation through probabilistic testing. *Information and Computation*, 94:1–28, 1991.
[23] A. Pnueli, J. Xu, and L. Zuck. Liveness with (0, 1, ∞)-counter abstraction. In *Proc. CAV'02*, pages 107–122, 2002.
[24] PRISM web site. http://www.cs.bham.ac.uk/˜dxp/prism/.
[25] J. Rutten, M. Kwiatkowska, G. Norman, and D. Parker. *Mathematical Techniques for Analyzing Concurrent and Probabilistic Systems*, P. Panangaden and F. van Breugel (eds.). American Mathematical Society, 2004.
[26] R. Segala and N. Lynch. Probabilistic simulations for probabilistic processes. In *Proc. CONCUR'94)*, volume 836 of *LNCS*, pages 481–496. Springer, 1994.
[27] F. Somenzi. CUDD: Colorado University decision diagram package. Public software, Colorado Univeristy, Boulder, http://vlsi.colorado.edu/˜fabio/.