Finite-Horizon Bisimulation Minimisation for Probabilistic Systems

Kamaleson, Nishanthan; Parker, David; Rowe, Jonathan

DOI:
10.1007/978-3-319-32582-8_10

Document Version
Peer reviewed version

Citation for published version (Harvard):
Kamaleson, N, Parker, D & Rowe, J 2016, Finite-Horizon Bisimulation Minimisation for Probabilistic Systems. in *Model Checking Software: 23rd International Symposium, SPIN 2016, Co-located with ETAPS 2016, Eindhoven, The Netherlands, April 7-8, 2016, Proceedings.* Lecture Notes in Computer Science, vol. 9641, Springer, pp. 147-164, 23rd International Symposium, SPIN 2016, Co-located with ETAPS 2016, Eindhoven, Netherlands, 7/04/16. https://doi.org/10.1007/978-3-319-32582-8_10

Finite-Horizon Bisimulation Minimisation for Probabilistic Systems

Nishanthan Kamaleson, David Parker, and Jonathan E. Rowe

School of Computer Science, University of Birmingham

Abstract. We present model reduction techniques to improve the efficiency and scalability of verifying probabilistic systems over a finite time horizon. We propose a finite-horizon variant of probabilistic bisimulation for discrete-time Markov chains, which preserves a bounded fragment of the temporal logic PCTL. In addition to a standard partition-refinement based minimisation algorithm, we present on-the-fly finite-horizon minimisation techniques, which are based on a backwards traversal of the Markov chain, directly from a high-level model description. We investigate both symbolic and explicit-state implementations, using SMT solvers and hash functions, respectively, and implement them in the PRISM model checker. We show that finite-horizon reduction can provide significant reductions in model size, in some cases outperforming PRISM's existing efficient implementations of probabilistic verification.

1 Introduction

Probabilistic verification is an automated technique for the formal analysis of quantitative properties of systems that exhibit stochastic behaviour. A probabilistic model, such as a Markov chain or a Markov decision process, is systematically constructed and then analysed against properties expressed in a formal specification language such as temporal logic. Mature tools for probabilistic verification such as PRISM [15] and MRMC [13] have been developed, and the techniques have been applied to a wide range of application domains, from biological reaction networks [11] to car airbag controllers [1].

A constant challenge in this area is the issue of scalability: probabilistic models, which are explored and constructed in an exhaustive fashion, are typically huge for real-life systems, which can limit the practical applicability of the techniques. A variety of approaches have been proposed to reduce the size of these models. One that is widely used is *probabilistic bisimulation* [18], an equivalence relation over the states of a probabilistic model which can be used to construct a smaller *quotient* model that is equivalent to the original one (in the sense that it preserves key properties of interest to be verified).

Typically, it preserves both infinite-horizon (long-run) properties, e.g., "the probability of eventually reaching an error state", finite-horizon (transient, or time-bounded) properties, e.g. "the probability of an error occurring within k time-steps", and, more generally, any property expressible in an appropriate

temporal logic such as PCTL [10]. It has been shown that, in contrast to non-probabilistic verification, the effort required to perform bisimulation minimisation can pay off in terms of the total time required for verification [12].

In this paper, we consider model reduction techniques for finite-horizon properties of Markov chains. We propose a *finite-horizon* variant of probabilistic bisimulation, which preserves stepwise behaviour over a finite number of steps, rather than indefinitely, as in standard probabilistic bisimulation. This permits a more aggressive model reduction, but still preserves satisfaction of PCTL formulae of bounded depth (i.e., whose interpretation requires only a bounded exploration of the model). Time-bounded properties are commonly used in probabilistic verification, e.g., for efficiency ("the probability of task completion within k steps") or for reliabilty ("the probability of an error occurring within time k").

We formalise finite-horizon probabilistic bisimulation, define the subset of PCTL that it preserves and then give a partition-refinement based algorithm for computing the coarsest possible finite-horizon bisimulation relation, along with a corresponding quotient model. The basic algorithm is limited by the fact it requires the full Markov chain to be constructed before it is minimised, which can be a bottleneck. So, we then develop on-the-fly approaches, which construct the quotient model directly from a high-level model description of the Markov chain, based on a backwards traversal of its state space. We propose two versions: one symbolic, based on SMT solvers, and one explicit-state.

We implemented all algorithms in PRISM and evaluated them on a range of examples. First, we apply the partition-refinement based approach to some standard benchmarks to investigate the size of the reduction that can be obtained in a finite-horizon setting. Then, we apply the on-the-fly approach to a class of problems to which it is particularly well suited: models with a large number of possible initial configurations, on which we ask questions such as "from which initial states does the probability of an error occurring within 10 seconds exceed 0.01?". We show that on-the-fly finite-horizon bisimulation can indeed provide significant gains in both verification time and scalability, demonstrated in each case by outperforming the existing efficient implementations in PRISM.

Related Work. For the standard notion of probabilistic bisimulation on Markov chains [18], various decision procedure and minimisation algorithms have been developed. Derisavi et al. [9] proposed an algorithm with optimal complexity, assuming the use of splay trees and, more recently, a simpler solution was put forward in [20]. Signature-based approaches, which our first, partition-refinement algorithm adapts, have been studied in, for example, [9,22]. Also relevant is the SMT-based bisimulation minimisation technique of [6] which, like our on-the-fly algorithm, avoids construction of the full model when minimising. Our SMT-based algorithm has an additional benefit in that it works on model descriptions with state-dependent probabilities. Other probabilistic verification methods have been developed based on backwards traversal of a model, for example for probabilistic timed automata [16], but this is for a different class of models and does not perform minimisation. Della Penna et al. considered finite-horizon verification of Markov chains [7], but using disk-based methods, not model reduction.

2 Preliminaries

We start with some background on probabilistic verification of Markov chains.

2.1 Discrete-time Markov Chains

A discrete-time Markov chain (DTMC) can be thought of as a state transition system where transitions between states are annotated with probabilities.

Definition 1 (DTMC). *A DTMC is a tuple* $\mathcal{D} = (\mathcal{S}, \mathcal{S}_{init}, \mathbf{P}, \mathcal{AP}, \mathcal{L})$*, where:*

- $\mathcal{S}$ *is a finite set of states and* $\mathcal{S}_{init} \subseteq \mathcal{S}$ *is a set of initial states;*
- $\mathbf{P} : \mathcal{S} \times \mathcal{S} \to [0,1]$ *is a transition probability matrix, where, for all states* $s \in \mathcal{S}$*, we have* $\sum_{s' \in \mathcal{S}} \mathbf{P}(s,s') = 1$*;*
- $\mathcal{AP}$ *is a set of atomic propositions and* $\mathcal{L} : \mathcal{S} \to 2^{\mathcal{AP}}$ *is a labelling function giving the set of propositions from* $\mathcal{AP}$ *that are true in each state.*

For each pair s, s' of states, $\mathbf{P}(s,s')$ represents the probability of going from s to s'. If $\mathbf{P}(s,s') > 0$, then s is a predecessor of s' and s' is a successor of s. For a state s and set $C \subseteq \mathcal{S}$, we will often use the notation $\mathbf{P}(s,C) := \sum_{s' \in C} \mathbf{P}(s,s')$.

A path σ of a DTMC $\mathcal{D}$ is a finite or infinite sequence of states $\sigma = s_0 s_1 s_2 \ldots$ such that $\forall i \geq 0,\ s_i \in \mathcal{S}$ and $\mathbf{P}(s_i, s_{i+1}) > 0$. The i^{th} state of the path σ is denoted by $\sigma[i]$. We let $Path^{\mathcal{D}}(s)$ denote the set of infinite paths of $\mathcal{D}$ that begin in a state s. To reason formally about the behaviour of a DTMC, we define a probability measure Pr_s over the set of infinite paths $Path^{\mathcal{D}}(s)$ [14]. We usually consider the behaviour from some initial state $s \in \mathcal{S}_{init}$ of $\mathcal{D}$.

2.2 Probabilistic Computation Tree Logic

Properties of probabilistic models can be expressed using *Probabilistic Computation Tree Logic* (PCTL) [10] which extends Computation Tree Logic (CTL) with time and probabilities. In PCTL, state formulae Φ are interpreted over states of a DTMC and path formulae ϕ are interpreted over paths.

Definition 2 (PCTL). *The syntax of PCTL is as follows:*

$$\Phi ::= \text{true} \mid a \mid \neg\Phi \mid \Phi \wedge \Phi \mid \mathtt{P}_{\bowtie p}[\,\phi\,]$$
$$\phi ::= \Phi_1 \,\mathtt{U}^{\leq k}\, \Phi_2$$

where a *is an atomic proposition,* $\bowtie \in \{<, \leq, \geq, >\}$*,* $p \in [0,1]$ *and* $k \in \mathbb{N} \cup \{\infty\}$*.*

The main operator in PCTL, in addition to those that are standard from propositional logic, is the probabilistic operator $\mathtt{P}_{\bowtie p}[\phi]$, which means that the probability measure of paths that satisfy ϕ is within the bound $\bowtie p$. For path formulae ϕ, we allow the (bounded) until operator $\Phi_1 \,\mathtt{U}^{\leq k}\, \Phi_2$. If Φ_2 becomes true within k time steps and Φ_1 is true until that point, then $\Phi_1 \,\mathtt{U}^{\leq k}\, \Phi_2$ is true. In the case where k equals ∞, the bounded until operator becomes the unbounded until operator and is denoted by $\mathtt{U}$. For simplicity of presentation, in this paper, we omit the next ($\mathtt{X}\,\Phi$) operator, but this could easily be added.

Definition 3 (PCTL semantics). *Let $\mathcal{D} = (\mathcal{S}, \mathcal{S}_{init}, \mathbf{P}, \mathcal{AP}, \mathcal{L})$ be a DTMC. The satisfaction relation $\vDash_{\mathcal{D}}$ for PCTL formulae on $\mathcal{D}$ is defined by:*

- $s \vDash_{\mathcal{D}} \text{true}$ $\quad \forall s \in \mathcal{S}$
- $s \vDash_{\mathcal{D}} a$ $\quad$ *iff* $a \in \mathcal{L}(s)$
- $s \vDash_{\mathcal{D}} \neg\Phi$ $\quad$ *iff* $s \nvDash_{\mathcal{D}} \Phi$
- $s \vDash_{\mathcal{D}} \Phi_1 \wedge \Phi_2$ $\quad$ *iff* $s \vDash_{\mathcal{D}} \Phi_1$ *and* $s \vDash_{\mathcal{D}} \Phi_2$
- $s \vDash_{\mathcal{D}} \mathtt{P}_{\bowtie p}[\phi]$ $\quad$ *iff* $Pr_s\{\sigma \in Path^{\mathcal{D}}(s) \mid \sigma \vDash_{\mathcal{D}} \phi\} \bowtie p$
- $\sigma \vDash_{\mathcal{D}} \Phi_1 \,\mathtt{U}^{\leq k}\, \Phi_2$ $\quad$ *iff* $\exists i \in \mathbb{N}.(i \leq k \wedge \sigma[i] \vDash_{\mathcal{D}} \Phi_2 \wedge (\forall j.0 \leq j < i.\sigma[j] \vDash_{\mathcal{D}} \Phi_1))$

For example, a PCTL formula such as $\mathtt{P}_{<0.01}[\neg \mathit{fail}_1 \,\mathtt{U}^{\leq k}\, \mathit{fail}_2]$ means that the probability of a failure of type 2 occurring within k time-steps, and before a failure of type 1 does, is less than 0.01. Common derived operators are $\mathtt{F}\,\Phi \equiv \text{true}\,\mathtt{U}\,\Phi$, which means that Φ eventually becomes true, and $\mathtt{F}^{\leq k}\,\Phi \equiv \text{true}\,\mathtt{U}^{\leq k}\,\Phi$, which means that Φ becomes true within k steps.

2.3 Probabilistic Bisimulation

Larsen and Skou [18] defined (strong) *probabilistic bisimulation* for discrete probabilistic transition systems, which is an equivalence relation used to identify states with identical labellings and (probabilistic) step-wise behaviour.

Definition 4 (Probabilistic bisimulation). *Let $\mathcal{D} = (\mathcal{S}, \mathcal{S}_{init}, \mathbf{P}, \mathcal{AP}, \mathcal{L})$ be a DTMC and $\mathcal{R}$ an equivalence relation on $\mathcal{S}$. Then $\mathcal{R}$ is a (strong) probabilistic bisimulation on $\mathcal{D}$ if, for $(s_1, s_2) \in \mathcal{R}$:*

$$\textit{(i) } \mathcal{L}(s_1) = \mathcal{L}(s_2) \;\; \textit{and (ii) for all } C \in \mathcal{S}/\mathcal{R} : \;\; \mathbf{P}(s_1, C) = \mathbf{P}(s_2, C)$$

where $\mathcal{S}/\mathcal{R}$ denotes the set of equivalence classes of set $\mathcal{S}$ by relation $\mathcal{R}$. States s_1, s_2 are bisimilar if there exists a bisimulation on $\mathcal{D}$ containing (s_1, s_2).

Two states that are probabilistically bisimilar will satisfy the same properties, including both infinite-horizon (long-run) and finite-horizon (transient) properties. Aziz et al. [3] proved that any property in the temporal logic PCTL is also preserved in this manner. Thanks to these results, the analysis of the original Markov chain, such as probabilistic model checking of PCTL, can be equivalently performed on the *quotient* Markov chain, in which equivalence classes of bisimilar states are lumped together into a single state.

Usually, we are interested in the coarsest possible probabilistic bisimulation for a DTMC $\mathcal{D}$ (or, in other words, the union of all possible bisimulation relations). We denote the coarsest possible probabilistic bisimulation by $\sim$. The quotient model $\mathcal{D}/\!\sim$ derived using this relation is defined as follows.

Definition 5 (Quotient DTMC). *Given DTMC $\mathcal{D} = (\mathcal{S}, \mathcal{S}_{init}, \mathbf{P}, \mathcal{AP}, \mathcal{L})$, the quotient DTMC is defined as $\mathcal{D}/\!\sim = (\mathcal{S}', \mathcal{S}'_{init}, \mathbf{P}', \mathcal{AP}, \mathcal{L}')$ where:*

- $\mathcal{S}' = \mathcal{S}/\!\sim = \{[s]_\sim \mid s \in \mathcal{S}\}$
- $\mathcal{S}'_{init} = \{[s]_\sim \mid s \in \mathcal{S}_{init}\}$
- $\mathbf{P}'([s]_\sim, [s']_\sim) = \mathbf{P}(s, [s']_\sim)$
- $\mathcal{L}'([s]_\sim) = \mathcal{L}(s)$

and $[s]_\sim$ denotes the unique equivalence class of relation $\sim$ containing s.

3 Finite-Horizon Bisimulation

We now formalise the notion of *finite-horizon bisimulation*, a step-bounded variant of standard probabilistic bisimulation for Markov chains [18]. We fix, from this point on, a DTMC $\mathcal{D} = (\mathcal{S}, \mathcal{S}_{init}, \mathbf{P}, \mathcal{AP}, \mathcal{L})$. Intuitively, a *$k$-step finite-horizon bisimulation*, for non-negative integer k, preserves the stepwise behaviour of $\mathcal{D}$ over a finite horizon of k steps. We use the following inductive definition.

Definition 6 (Finite-horizon bisimulation). *A* k-step finite-horizon bisimulation*, for $k \in \mathbb{N}_{\geq 0}$, is an equivalence relation $\mathcal{R}_k \subseteq \mathcal{S} \times \mathcal{S}$ such that, for all states $(s_1, s_2) \in \mathcal{R}_k$, the following two conditions are satisfied:*

(i) $\mathcal{L}(s_1) = \mathcal{L}(s_2)$*;*
(ii) $\mathbf{P}(s_1, C) = \mathbf{P}(s_2, C)$ *for each equivalence class* $C \in \mathcal{S}/\mathcal{R}_{k-1}$*,*

where $\mathcal{R}_{k-1}$ is a $(k-1)$-step finite-horizon bisimulation. A 0-step finite-horizon bisimulation *is an equivalence relation $\mathcal{R}_0$ satisfying only condition (i) above.*

Definition 7 (Finite-horizon bisimulation equivalent). *We say states s_1, s_2 are (k-step)* finite-horizon bisimulation equivalent *(*bisimilar*), denoted $s_1 \sim_k s_2$, if there exists a k-step finite-horizon bisimulation $\mathcal{R}_k$ such that $(s_1, s_2) \in \mathcal{R}_k$.*

Two states s_1 and s_2 satisfying $s_1 \sim_k s_2$ have the same stepwise behaviour over k steps. The following simple, but useful, properties hold.

Proposition 1. *Let $s_1, s_2 \in \mathcal{S}$ be two states. Then:*

(a) if $s_1 \sim_k s_2$, then $s_1 \sim_j s_2$ for any $0 \leq j \leq k$.
(b) if $s_1 \sim s_2$, then $s_1 \sim_k s_2$ for any $k \geq 0$.
(c) if $s_1 \sim_k s_2$ and $s_1 \to s_1'$, then $s_1' \sim_{k-1} s_2'$ for some state s_2' such that $s_2 \to s_2'$.

From a model checking perspective, if $s_1 \sim_k s_2$, then s_1 and s_2 satisfy the same PCTL formulae up to a bounded depth k. We formalise this as follows.

Definition 8 (Formula depth). *The* depth *of a PCTL formula Φ, denoted $d(\Phi)$, is a value in $\mathbb{N} \cup \{\infty\}$ defined inductively as follows:*

- $d(\text{true}) = d(a) = 0$ *for atomic proposition a;*
- $d(\neg\Phi) = d(\Phi)$*;*
- $d(\Phi_1 \wedge \Phi_2) = \max(d(\Phi_1), d(\Phi_2))$*;*
- $d(\mathtt{P}_{\bowtie p}[\Phi_1 \,\mathtt{U}^{\leq j}\, \Phi_2]) = j + \max(d(\Phi_1) - 1, d(\Phi_2))$.

For example, if a and b are atomic propositions, we have $d(\mathtt{P}_{\bowtie p}[\text{true}\,\mathtt{U}^{\leq 5}\, a]) = 5$, $d(\mathtt{P}_{\bowtie p}[\text{true}\,\mathtt{U}^{\leq 5}\, a] \wedge \mathtt{P}_{\bowtie p}[\text{true}\,\mathtt{U}^{\leq 6}\, a]) = 6$, and $d(\mathtt{P}_{\bowtie p}[\text{true}\,\mathtt{U}^{\leq 5}\, \mathtt{P}_{\bowtie p}[a\,\mathtt{U}^{\leq 3}\, b]]) = 8$.

If states s_1 and s_2 are (k-step) finite-horizon bisimilar, then they satisfy exactly the same PCTL formulae of depth at most k.

Theorem 1. *Let s_1 and s_2 be two states such that $s_1 \sim_k s_2$, and Φ be a PCTL formula with depth $d(\Phi) \leq k$, then $s_1 \models \Phi$ if and only if $s_2 \models \Phi$.*

Proof. We prove the result by induction over the structure (see Definition 2) of PCTL formula Φ. Propositional operators are straightforward since s_1 and s_2 satisfy the same atomic propositions, by the definition of $\sim_k$, and, for $\Phi = \neg\Phi_1$ or $\Phi = \Phi_1 \wedge \Phi_2$, the subformulae Φ_1 and Φ_2 have depth at most k so, by induction, we can assume that $s_1 \models \Phi_i \Leftrightarrow s_2 \models \Phi_i$ for $i \in \{1, 2\}$.

The remaining case to consider is $\Phi = \mathtt{P}_{\bowtie p}[\Phi_1 \,\mathtt{U}^{\leq j}\, \Phi_2]$. We know, from Definition 8, that the depths $d(\Phi_1)$ and $d(\Phi_2)$ of the two subformulae are at most $k-j+1$ and $k-j$. From the semantics of PCTL, we have that, for any state s:

$$s \models \mathtt{P}_{\bowtie p}[\,\Phi_1 \,\mathtt{U}^{\leq j}\, \Phi_2\,] \;\Leftrightarrow\; Pr_s(\Phi_1 \,\mathtt{U}^{\leq j}\, \Phi_2) \bowtie p$$

which means it suffices to show that:

$$Pr_{s_1}(\Phi_1 \,\mathtt{U}^{\leq j}\, \Phi_2) = Pr_{s_2}(\Phi_1 \,\mathtt{U}^{\leq j}\, \Phi_2) \tag{1}$$

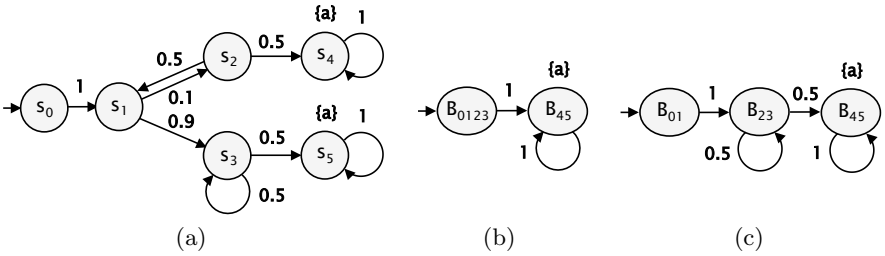
We in fact show this to be true for any states s_1, s_2, values $j \leq k$ and PCTL subformulae Φ_1, Φ_2 satisfying $s_1 \sim_k s_2$ and $\max(d(\Phi_1)-1, d(\Phi_2)) \leq k-j$, which we prove inductively over j. From the model checking algorithm for PCTL [10], we know that, for any state s:

$$Pr_s(\Phi_1 \,\mathtt{U}^{\leq j}\, \Phi_2) = \begin{cases} 1 & \text{if } s \models \Phi_2 \\ 0 & \text{if } s \models \neg\Phi_1 \wedge \neg\Phi_2 \\ 0 & \text{if } s \models \Phi_1 \wedge \neg\Phi_2 \text{ and } j = 0 \\ \sum_{s' \in \mathcal{S}} \mathbf{P}(s,s') Pr_{s'}(\Phi_1 \,\mathtt{U}^{\leq j-1}\, \Phi_2) & \text{if } s \models \Phi_1 \wedge \neg\Phi_2 \text{ and } j > 0. \end{cases}$$

For the base case $j = 0$, only the first three cases of the definition above can apply, and we know that $s_1 \models \Phi_i \Leftrightarrow s_2 \models \Phi_i$ for $i \in \{1, 2\}$, so we have that $Pr_{s_1}(\Phi_1 \,\mathtt{U}^{\leq 0}\, \Phi_2) = Pr_{s_2}(\Phi_1 \,\mathtt{U}^{\leq 0}\, \Phi_2)$. For the inductive case, where $j > 0$, we can assume that $Pr_{s_1}(\Phi_1 \,\mathtt{U}^{\leq j-1}\, \Phi_2) = Pr_{s_2}(\Phi_1 \,\mathtt{U}^{\leq j-1}\, \Phi_2)$, as long as $s_1 \sim_{j-1} s_2$. Considering again the possible cases in the above definition, the first two follow as for $j = 0$ and the third cannot apply since $j > 0$. For the fourth case, since $j > 0$, we know there exists a $(j-1)$-step finite-horizon bisimulation $\mathcal{R}_{j-1}$. Let us further assume an (arbitrary) function $rep : \mathcal{S}/\mathcal{R}_{j-1} \to \mathcal{S}$, which selects a unique representative from each equivalence class of $\mathcal{R}_{j-1}$. We have:

$$\begin{aligned}
& Pr_{s_1}(\Phi_1 \,\mathtt{U}^{\leq j}\, \Phi_2) \\
&= \textstyle\sum_{s' \in \mathcal{S}} \mathbf{P}(s_1, s') Pr_{s'}(\Phi_1 \,\mathtt{U}^{\leq j-1}\, \Phi_2) && \text{by definition} \\
&= \textstyle\sum_{C \in \mathcal{S}/\sim_{j-1}} \sum_{s' \in C} \mathbf{P}(s_1, s') Pr_{s'}(\Phi_1 \,\mathtt{U}^{\leq j-1}\, \Phi_2) && \text{since } \sim_{j-1} \text{ partitions } \mathcal{S} \\
&= \textstyle\sum_{C \in \mathcal{S}/\sim_{j-1}} Pr_{rep(C)}(\Phi_1 \,\mathtt{U}^{\leq j-1}\, \Phi_2) \sum_{s' \in C} \mathbf{P}(s_1, s') && \text{by induction on } j \\
&= \textstyle\sum_{C \in \mathcal{S}/\sim_{j-1}} Pr_{rep(C)}(\Phi_1 \,\mathtt{U}^{\leq j-1}\, \Phi_2) \mathbf{P}(s_1, C) \\
&= \textstyle\sum_{C \in \mathcal{S}/\sim_{j-1}} Pr_{rep(C)}(\Phi_1 \,\mathtt{U}^{\leq j-1}\, \Phi_2) \mathbf{P}(s_2, C) && \text{since } s_1 \sim_j s_2 \\
&= \textstyle\sum_{C \in \mathcal{S}/\sim_{j-1}} Pr_{rep(C)}(\Phi_1 \,\mathtt{U}^{\leq j-1}\, \Phi_2) \sum_{s' \in C} \mathbf{P}(s_2, s') \\
&= \textstyle\sum_{C \in \mathcal{S}/\sim_{j-1}} \sum_{s' \in C} \mathbf{P}(s_2, s') Pr_{s'}(\Phi_1 \,\mathtt{U}^{\leq j-1}\, \Phi_2) && \text{since } s' \sim_{j-1} rep(C) \\
&= \textstyle\sum_{s' \in \mathcal{S}} \mathbf{P}(s_2, s') Pr_{s'}(\Phi_1 \,\mathtt{U}^{\leq j-1}\, \Phi_2) && \text{since } \sim_{j-1} \text{ partitions } \mathcal{S} \\
&= Pr_{s_2}(\Phi_1 \,\mathtt{U}^{\leq j}\, \Phi_2) && \text{by definition}
\end{aligned}$$

which proves (1), as required, and concludes the proof. □



(a) (b) (c)

Fig. 1. (a) Example DTMC; (b-c) Finite-horizon quotient DTMCs for $k = 0, 1$.

In similar fashion to the standard (non-finite-horizon) case, we are typically interested in the *coarsest possible* k-step finite-horizon bisimulation relation for a given DTMC (labelled with atomic propositions) and time horizon k, which we denote by $\sim_k$. We can also define this as the union of all possible k-step finite-horizon bisimulation relations. Furthermore, for $\sim_k$ (or any other finite-horizon bisimulation relation), we can define a corresponding *quotient* DTMC, whose states are formed from the equivalence classes of $\sim_k$, and whose k-step behaviour is identical to the original DTMC $\mathcal{D}$.

This is similar, but not identical, to the process of building the quotient Markov chain corresponding to a full minimisation (see Definition 5). We must take care since, unlike for full bisimulation, given a state $B \in \mathcal{S}/\sim_k$ of the quotient model, the probabilities $\mathbf{P}(s, B')$ of moving to other equivalence classes $B' \in \mathcal{S}/\sim_k$ can be different for each state $s \in B$ (according to the definition of $\sim_k$, probabilities are the same to go states with the same $(k-1)$-step, not k-step, behaviour). However, when they do differ, it suffices to pick an arbitrary representative from B. We formalise the quotient DTMC construction below, and then present some examples.

Definition 9 (Finite-horizon quotient DTMC). *If $\mathcal{D} = (\mathcal{S}, \mathcal{S}_{init}, \mathbf{P}, \mathcal{AP}, \mathcal{L})$ is a DTMC and $\sim_k$ is a finite-horizon bisimulation on $\mathcal{D}$, then a quotient DTMC can be constructed as $\mathcal{D}/\sim_k = (\mathcal{S}', \mathcal{S}'_{init}, \mathbf{P}', \mathcal{AP}, \mathcal{L}')$ where:*

- $\mathcal{S}' = \mathcal{S}/\sim_k = \{[s]_{\sim_k} \mid s \in \mathcal{S}\}$
- $\mathcal{S}'_{init} = \{[s]_{\sim_k} \mid s \in \mathcal{S}_{init}\}$
- $\mathbf{P}'(B, B') = \mathbf{P}(rep(B), B')$ *for any* $B, B' \in \mathcal{S}'$
- $\mathcal{L}'(B) = \mathcal{L}(rep(B))$ *for any* $B \in \mathcal{S}'$,

where $rep : \mathcal{S}/\sim_k \to \mathcal{S}$ is an arbitrary *function that selects a unique representative from each equivalence class of $\sim_k$, i.e., $B = [rep(B)]_{\sim_k}$ for all $B \in \mathcal{S}'$.*

Example 1. Fig. 1 illustrates finite-horizon bisimulation on an example DTMC, shown in part (a). Fig.s 1 (b) and (c) show quotient DTMCs for 0-step and 1-step finite-horizon bisimulation minimisation, respectively, where quotient state names indicate their corresponding equivalence class (e.g., B_{23} corresponds to DTMC states s_2 and s_3). For 2-step minimisation (not shown), blocks B_{23} and B_{01} are both split in two, and only the states s_4 and s_5 remain bisimilar.

From the above, we see that $s_2 \sim_1 s_3$, but $s_2 \not\sim_2 s_3$. Consider the PCTL formula $\Phi = \mathtt{P}_{\bowtie p}[\mathrm{true}\,\mathtt{U}^{\leq k}\,a]$, which has depth $d(\Phi) = k$. Satisfaction of Φ is equivalent in states s_2 and s_3 for $k = 1$, but not for $k = 2$. To give another example, for $\Phi' = \mathtt{P}_{>0}[\mathtt{P}_{>0.5}[\mathrm{true}\,\mathtt{U}^{\leq 2}\,a]\,\mathtt{U}^{\leq 1}\,a]$, which has $d(\Phi') = 1 + 2 - 1 = 2$, we have $s_3 \models \Phi'$, but $s_2 \not\models \Phi'$.

In constructing the 1-step quotient model (Fig. 1 (c)), we used s_1 as a representative of equivalence class $B_{01} = \{s_0, s_1\}$, which is why there is a transition to B_{23}. We could equally have used s_0, which would yield a different quotient DTMC, but which still preserves 1-step behaviour.

4 Finite-Horizon Bisimulation Minimisation

Bisimulation relations have a variety of uses, but our focus here is on using them to minimise a probabilistic model prior to verification, in order to improve the efficiency and scalability of the analysis. More precisely, we perform *finite-horizon bisimulation minimisation*, determining the coarsest possible finite-horizon bisimulation relation $\sim_k$, for a given k, and then constructing the corresponding quotient Markov chain. Theorem 1 tells us that it is then safe to perform verification on the smaller quotient model instead.

We begin, in this section, by presenting a classical *partition-refinement* based minimisation algorithm, which is based on an iterative splitting of an initially coarse partition of the state space until the required probabilistic bisimulation has been identified. In the next section, we will propose on-the-fly approaches which offer further gains in efficiency and scalability.

4.1 A Partition-Refinement Based Minimisation Algorithm

The standard approach to partition refinement is to use *splitters* [19,9], individual blocks in the current partition which show that one or more other blocks contain states that should be split into distinct sub-blocks. An alternative approach is to use a so-called *signature-based* method [8]. The basic structure of the algorithm remains the same, however the approach to splitting differs: rather than using splitters, a *signature* corresponding to the current partition is computed at each iteration for each state s. This signature comprises the probability of moving from s in one step to each block in the partition. In the next iteration, all states with different signatures are placed in different blocks.

Because each iteration of the signature-based algorithm considers the one-step behaviour of every state in the model, it is relatively straightforward to adapt to finite-horizon bisimulation minimisation. Algorithm 1 shows the finite-horizon minimisation algorithm MinimiseFiniteHorizon. It takes a DTMC $\mathcal{D}$ and the time horizon k as input. The partition Π is first initialised to group states based on the different combinations of atomic propositions, i.e., states with identical labellings are placed in one block.[1] The partition is then repeatedly split, each time by computing the signatures for each state and splitting

[1] In the algorithm, we store the signatures with the partition, so Π is a list of pairs of blocks (state-sets) and signatures (distributions).

accordingly. The loop terminates either when k iterations have been completed or no further splitting is possible. Finally, the quotient model is constructed, as described in the previous section.

Correctness. The correctness of MinimiseFiniteHorizon, i.e. that it generates the coarsest k-step finite-horizon bisimulation, can be argued with direct reference to Definition 6. For $k = 0$, only the initialisation step at the start of the algorithm is needed. For $k > 0$ the ith iteration of the loop produces a partition Π which groups precisely the equivalence classes of $\sim_i$, which are constructed from those of $\sim_{i-1}$, as in Definition 6. It is also clear that we group *all* equivalent states at each step, yielding the coarsest relation. If the algorithm terminates early, at step j, then $\sim_i = \sim_k$ for all $j \leq i \leq k$.

Algorithm 1: MinimiseFiniteHorizon

Data: $\mathcal{D} = (\mathcal{S}, \mathcal{S}_{init}, \mathbf{P}, \mathcal{AP}, \mathcal{L})$, k

$\Pi, \Pi' := \emptyset$; // Initialise partition

for $A \subseteq \mathcal{AP}$ **do**
  $B_A := \{s \in \mathcal{S} \mid L(s) = A\}$
  if $B_A \neq \emptyset$ **then** $\Pi := \Pi \cup \{(\{B_A\}, \langle\rangle)\}$;

$i := 1$; // Splitting loop

while $i \leq k \ \wedge \ \Pi \neq \Pi'$ **do**
  $\Pi' := \Pi$; $\Pi := \emptyset$
  for $s \in \mathcal{S}$ **do**
    $Sig := \langle\rangle$; // Compute signature
    for $B \in \Pi'$ **do** $Sig(B) := 0$;
    for $s \to s'$ **do**
      $B_{s'} :=$ block of Π' containing s'
      $Sig(B_{s'}) := Sig(B_{s'}) + \mathbf{P}(s, s')$
    $B_s :=$ block of Π' containing s
    if $\exists (B', Sig) \in \Pi \wedge B' \subseteq B_s$ **then**
      $B' := B' \cup \{s\}$; // New blocks
    else
      $\Pi := \Pi \cup \{(\{s\}, Sig)\}$
  $i := i + 1$

$\mathcal{S}' := \emptyset$; $\mathcal{S}'_{init} := \emptyset$; // Build quotient

for $(B, Sig) \in \Pi$ **do**
  $\mathcal{S}' := \mathcal{S}' \cup \{B\}$
  if $B \cap \mathcal{S}_{init} \neq \emptyset$ **then** $\mathcal{S}'_{init} := \mathcal{S}'_{init} \cup \{B\}$;
  $\mathbf{P}'(B, \cdot) := Sig$
  $\mathcal{L}'(B) := \mathcal{L}(s)$ for any $s \in B$

return $\mathcal{D}' = (\mathcal{S}', \mathcal{S}'_{init}, \mathbf{P}', \mathcal{AP}, \mathcal{L}')$

5 On-the-Fly Finite-Horizon Minimisation

A key limitation of the partition-refinement approach presented in the previous section is that it takes as input the full DTMC to be minimised, the construction of which can be expensive in terms of both time and space. This can remove any potential gains in terms of scalability that minimisation can provide.

To resolve this, we now propose methods to compute a finite-horizon bisimulation minimisation in an *on-the-fly* fashion, where the minimised model is constructed directly from a high-level modelling language description of the original model, bypassing construction of the full, un-reduced DTMC. In our case, the probabilistic models are described using the modelling language of the PRISM model checker [15], which is based on guarded commands.

Our approach works through a backwards traversal of the model, which allows us to perform bisimulation minimisation on the fly. For simplicity, we focus on preserving the subclass of PCTL properties comprising a single P operator, more precisely, those of the form $\mathtt{P}_{\bowtie p}[\, b_1 \,\mathtt{U}^{\leq k}\, b_2 \,]$ for atomic propositions b_1 and b_2. This is the kind of property most commonly found in practice.

5.1 The On-the-Fly Minimisation Algorithm

The basic approach to performing finite-horizon minimisation on the fly is shown as FiniteHorizonOnTheFly, in Algorithm 2. This takes *model*, which is a description of the DTMC, B_1 and B_2, the sets of states satisfying b_1 and b_2, respectively, in the property $\mathtt{P}_{\bowtie p}[\, b_1 \,\mathtt{U}^{\leq k}\, b_2 \,]$, and the time horizon k. The algorithm does not make any assumptions about how sets of states are represented or manipulated. Below, we will discuss two separate instantiations of it.

The algorithm is based on a backwards traversal of the model. It uses a separate algorithm FindMergedPredecessors(*model*, *target*, *restrict*), which queries the DTMC (*model*) to find all (immediate) predecessors of states in *target* that are also in *restrict* (the *restrict* set will be used to restrict attention to the set B_1 corresponding to the left-hand side b_1 of the until formula). The algorithm also groups the predecessor states in blocks according to the probabilities with which they transition to *target* and returns these too. As above, each instantiation of Algorithm 2 will use a separate implementation of the FindMergedPredecessors algorithm.

The main loop of the algorithm iterates backwards through the model: after the ith iteration, it has found all states that can reach the target set B_2 within i steps with positive probability. The new predecessors for each iteration are stored in a set of blocks P. A separate set P' is used to store predecessors of blocks in P, which will then be considered in the next iteration.

More precisely, P (and P') store, like in Algorithm 1, a list of pairs (B, D) where B is a block (a set of states) and D is a (partial) probability distribution storing probabilities of outgoing transitions (from B, to other blocks). The set Π, which is used to construct the partition representing the finite-horizon bisimulation relation, is also stored as a list of pairs.

Algorithm 2: FiniteHorizonOnTheFly

Data: $model$, B_1, B_2, k

$P := \{\text{FindMergedPredecessors}(model, B_2, B_1)\}$; $P' := \emptyset$
$\Pi := \{(B_2, \langle\rangle)\}$

$i := 1$
while $P \neq \emptyset \wedge i \leq k$ **do**
  $(B, D) := \text{pop}(P)$; `// block` B`, (sub)distribution` D
  for $(B', D') \in \Pi \wedge B \neq \emptyset$ **do**
    if $B' \cap B \neq \emptyset$ **then**
      replace (B', D') in Π with $(B' \setminus B), D')$ and $(B' \cap B, D' \cup D)$
      $B := B \setminus B'$
      refine all $(B'', D'') \in \Pi$ and (B, D) with respect to the split of B'
    end
  end
  if $B \neq \emptyset$ **then**
    $\Pi := \Pi \cup \{(B, D)\}$
    $P' := P' \cup \{\text{FindMergedPredecessors}(model, B, B_1)\}$
  end
  if $(P = \emptyset \wedge P' \neq \emptyset)$ **then**
    $P := P'$; $P' := \emptyset$
    $i := i + 1$
  end
end

return FiniteHorizonQuotient(Π)

Algorithm 3: FiniteHorizonQuotient

Data: Π

$\mathcal{S}' := \{B_{sink}\}$; $\mathcal{L}'(B_{sink}) = \emptyset$; $\mathcal{S}'_{init} := \emptyset$; $\mathbf{P}'(B_{sink}, \cdot) := \langle B_{sink} \to 1\rangle$;
for $(B, D) \in \Pi$ **do**
  $\mathcal{S}' := \mathcal{S}' \cup \{B\}$
  if $B \cap \mathcal{S}_{init} \neq \emptyset$ **then** $\mathcal{S}'_{init} := \mathcal{S}'_{init} \cup \{B\}$;
  $p_{sink} = 1 - \sum_{(B', D') \in \Pi} D(B')$
  $\mathbf{P}'(B, \cdot) := D \cup \langle B_{sink} \to p_{sink}\rangle$
  $\mathcal{L}'(B) := \mathcal{L}(s)$ for any $s \in B$
end
return $\mathcal{D}' = (\mathcal{S}', \mathcal{S}'_{init}, \mathbf{P}', \mathcal{AP}, \mathcal{L}')$

Algorithm 2 begins by finding all immediate predecessors of states in B_2 that are also in B_1 and putting them in P. In each iteration, it takes each block-distribution pair (B, D) from P one by one: it will add this to the current partition Π. But, before doing so, it checks whether B overlaps with any existing blocks B' in Π. If so, B' is split in two, and the overlap is removed from B. At this point, the partition Π is refined to take account of the splitting of block B'. We repeatedly recompute the probabilities associated with each block in Π and, if these are then different for states within that block, it is also split.

Each iteration of the main loop finishes when all pairs (B, D) from P have been dealt with. If $i < k$, then newly found predecessors P' are copied to P and the process is repeated. If $i = k$, then the time horizon k has been reached and the finite-horizon bisimulation has been computed.

Finally, the quotient model is built. The basic construction is as in Algorithm 1 but, since on-the-fly construction only partially explores the model, we need to add an extra sink state to complete the DTMC.

Computing predecessors. One of the main challenges in implementing the on-the-fly algorithm is determining the predecessors of a given set of states from the high-level modelling language description. The PRISM language, used here, is based on guarded commands, for example:

$$c > 0 \;\rightarrow\; c/K \;:\; (c' = c - 1) \;+\; 1 - c/K \;:\; (c' = c + 1);$$

The meaning is that, when a state satisfies the *guard* ($c > 0$), the *updates* (decrementing or incrementing variable c) can be executed, each with an associated probability (c/K or $1 - c/K$). We assume here a single PRISM *module* of commands (multiple modules can be syntactically expanded into a single one [23]).

In the following sections, we describe two approaches to finding predecessors: one *symbolic*, which represents blocks (sets of states) as predicates and uses an SMT (satisfiability modulo theories) [5] based implementation; and one *explicit-state*, which explicitly enumerates the states in each block.

5.2 Symbolic (SMT-based) Minimisation

Our first approach represents state sets (i.e., blocks of the bisimulation partition) *symbolically*, as predicates over PRISM model variables. If *target* is a predicate representing a set of states, their predecessors, reached by applying some guarded command update *update*, can be found using the *weakest precondition*, denoted $\mathbf{wp}(update,\ target)$. More precisely, if the guard of the command is *guard*, and *bounds* represents the lower and upper bounds of all model variables, the following expression captures the set of states, if any, that are predecessors:

$$bounds \;\wedge\; guard \;\wedge\; \mathbf{wp}(update,\ target)$$

We determine, for each guarded command update in the model description, whether states can reach *target* via that update by checking the satisfiability of the expression above using an SMT solver. FindMergedPredecessors (see Algorithm 4) is used to determine predecessors in this way. It also restricts attention to states satisfying a further expression *restrict*.

The probability attached to an update in a guarded command is in general a state-dependent expression *prob* (see the earlier example command) so this must be analysed when FindMergedPredecessors groups states according to the probability with which they transition to *target*. If the SMT query in the algorithm is satisfiable, a valid probability is also obtained from the corresponding valuation (p' in Algorithm 4). The conjunction of the expression *predecessor* and

Algorithm 4: FindMergedPredecessors (SMT-based)

Data: *model*, *target*, *restrict*

$P := \emptyset$
bounds := variable bounds from *model*
foreach $(guard, updates)$ *in model* **do**
  foreach $(prob, update)$ *in updates* **do**
    $predecessor := restrict \wedge bounds \wedge guard \wedge \mathbf{wp}(update, target)$
    $query := predecessor \wedge (p = prob)$
    while *query is satisfiable* **do**
      p' := value of p in *query*
      if $(B, \langle target \to p' \rangle) \in P$ *for some* B **then**
        replace $(B, \langle target \to p' \rangle)$ in P with
        $(B \vee predecessor, \langle target \to p' \rangle)$
      else
        $P := P \cup \{(predecessor, \langle target \to p' \rangle)\}$
      end
      $query := query \wedge (prob \neq p')$
    end
  end
end
return P

$p = prob$ denotes the set of predecessors with the same probability. To obtain all such probabilities, the algorithm adds a *blocking expression* $prob \neq p'$ to the query and repeats the process.

SMT-based methods for probabilistic bisimulation minimisation have been developed previously [6]. One key difference here is that our approach handles transition probabilities expressed as state-dependent expressions, rather than fixed constants, which are needed for some of the models we later evaluate.

5.3 Explicit-State Minimisation

As an alternative to the symbolic approach using SMT, we developed an explicit-state implementation of finite-horizon minimisation in which the blocks of equivalent states are represented by explicitly listing the states that comprise them. As in the previous algorithm, the blocks are refined at each time step such that states residing in the same block have equal transition probabilities to the required blocks. To improve performance and store states compactly, we hash them based on the valuation of variables that define them. This is done in such a way that the hash values are bi-directional (one-to-one).

The algorithm explicitly computes the predecessor state for each update and each state in the set *target*, the transition probability is then computed for each predecessor state and these are collected in order to group states into sets. The set *restrict* is not stored explicitly, but rather as a symbolic expression which is then evaluated against each state's variable values to compute the intersection.

6 Experimental Results

We have implemented the bisimulation minimisation techniques presented in this paper as an extension of the PRISM model checker [15], and applied them to a range of benchmark models. For both the partition-refinement based minimisation of Sect. 4, and the on-the-fly methods in Sect. 5, we build on PRISM's "explicit" model checking engine. For the SMT-based variant, we use the Z3 solver [4], through the Z3 Java API. All our experiments were run on an Intel Core i7 2.8 GHz machine, using 2 GB of RAM.

Our investigation is in two parts. First, we apply the partition-refinement algorithm to several DTMCs from the PRISM benchmark suite [17] to get an idea of the size of reductions that can be obtained on some standard models. We use: *Crowds* (an anonymity protocol), *EGL* (a contract signing protocol) and *NAND* (NAND multiplexing). Details of all models, parameters and properties used can be found at [24]. A common feature of these models is that they have a single initial state, from which properties are verified. Since on-the-fly approaches explore backwards from a target set, we would usually need to consider time horizons k high enough such that the whole model was explored.

So, to explore in more depth the benefits of the on-the-fly algorithms, we consider another common class of models in probabilistic verification: those in which we need to exhaustively check whether a property is true over a large set of possible configurations. We use *Approximate majority* [2], a population protocol for computing a majority value amongst a set of K agents, and two simple models of *genetic algorithms* [21] in which a population of K agents evolves over time, competing to exist according to a fitness value in the range $0, \ldots, N-1$. In the first variant, *tournament*, the agent with the highest value wins; in the second, *modulo*, the sum of the two scores is used modulo N. Again, details of all models, parameters and properties used can be found at [24].

6.1 The partition-refinement algorithm

Fig. 2 shows results for the partition-refinement algorithm. The top row of plots shows the number of blocks in the partition built by finite-horizon bisimulation minimisation for different values of k on the first three benchmark examples. For the largest values of k shown, we have generated the partition corresponding to the full (non-finite-horizon) bisimulation. In most cases, the growth in the number of blocks is close to linear in k, although it is rather less regular for the *NAND* example. In all cases, it seems that the growth is slow enough that verifying finite-horizon properties for a range of values of k can be done on a considerably smaller model than the full bisimulation.

The bottom row of plots shows, for the same examples, the time required to perform bisimulation minimisation and then verify a k-step finite-horizon property (details at [24]). The black lines show the time for finite-horizon minimisation, the grey lines for full minimisation. The latter are relatively flat, indicating that the time for verification (which is linear in k) is very small compared to the time needed for minimisation. However, we see significant gains in the total time required for finite-horizon minimisation compared to full minimisation.

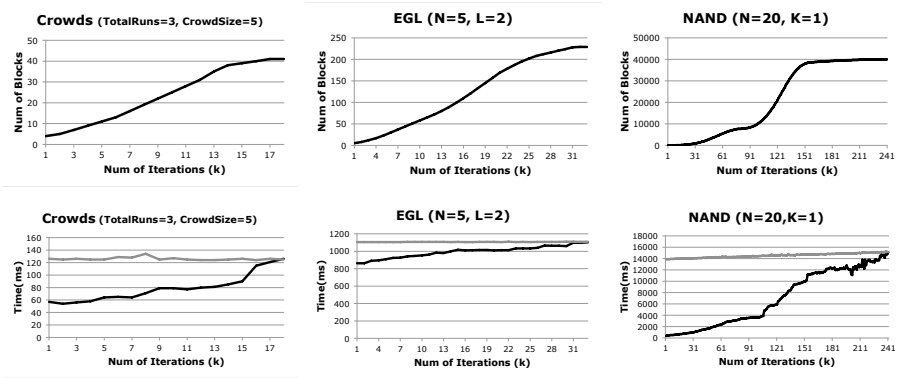


Fig. 2. Results for partition-refinement. Top: quotient size for varying time horizon k. Bottom: time for finite-horizon (black) and full (grey) minimisation/verification.

However, despite these gains, the times to minimise and verify the quotient model are still larger than to simply build and verify the full model. This is primarily because the partition refinement algorithm requires construction of the complete model first, the time for which eclipses any gains from minimisation. This was the motivation for the on-the-fly algorithms, which we evaluate next.

6.2 On-the-Fly Algorithms

Table 1 shows model sizes and timings for the on-the-fly algorithms on a range of models and scenarios. The left four columns show the model (and which on-the-fly algorithm was used), any parameters required (N or K) and the time horizon k. Next, under the headings 'Full Red.' and 'Finite Horiz.', we show the reductions in model size obtained using full (non-finite-horizon) and finite-horizon minimisation (for several k), respectively. In the first case, 'States' and 'Blocks' show the size of the full DTMC and the fully reduced quotient model, respectively. For the second case, 'Blocks' is the size of the finite-horizon quotient model and, to give a fair comparison, 'States' is the number of states in the full DTMC that can reach the target of the property within k steps (i.e., the number of states across all blocks). The rightmost three columns show the time required to build the model in three scenarios: 'Finite Horiz.' uses the on-the-fly approach over k steps; 'Full Red.' builds the full (non-finite-horizon) quotient by repeating the on-the-fly algorithm until all states have been found; and 'PRISM' builds the full model using its most efficient (symbolic) construction engine.

First, we note that finite-horizon minimisation yields useful reductions in model size in all cases, both with respect to the full model and to normal (non-finite horizon) minimisation. Bisimulation reduces models by a factor of roughly 2 and 5, for the *Approximate majority* and *Modulus* examples, respectively. For *Tournament*, a very large reduction is obtained since, for the property checked, the model ends up being abstracted to only distinguish two fitness values. Finite-horizon minimisation gives models that are smaller again, by a factor of between

Model (method)	Param.s		k	Full Red.		Finite Horiz.		Time (s)		
	N	K		States	Blocks	States	Blocks	PRISM	Full Red.	Finite Horiz.
Approx. majority (explicit)	n/a	100	20	20300	10201	242	122	11.0	14.2	0.2
			40			882	442			0.3
			60			1922	962			0.4
		150	100	45450	22801	5202	2602	46.1	83.1	1.2
			150			11552	5777			5.1
			200			20402	10202			15.6
		200	250	80600	40401	31752	15877	memout	293.5	40.8
			300			45602	22802			93.9
			350			61952	30977			180.8
		250	375	125750	63001	71064	35533	memout	773.5	247.8
			400			80802	40402			323.2
			425			91164	45583			416.6
Genetic alg. tournament (explicit)	8	22	8	1184040	22	6435	10	19.2	5.3	0.3
			9			11440	11			0.4
			10			19448	12			0.4
		23	8	1560780	23	6435	10	31.1	7.0	0.3
			9			11440	11			0.4
			10			19448	12			0.4
	10	21	8	10015005	21	24310	10	59.0	43.6	0.5
			9			48620	11			0.6
			10			92378	12			0.7
		22	8	14307150	22	24310	10	61.3	51.3	0.5
			9			48620	11			0.6
			10			92378	12			0.7
Genetic alg. tournament (SMT)	4	9	3	165	9	20	5	0.03	155	4.5
			4			35	6			11.1
			5			56	7			23.5
		10	3	220	10	20	5	0.03	215	9.3
			4			35	6			15.1
			5			56	7			31.1
	5	9	3	330	9	35	5	0.04	723.4	22.1
			4			70	6			70.7
			5			126	7			180.9
		10	3	495	10	35	5	0.04	1998.7	48.8
			4			70	6			82.0
			5			126	7			233.7
Genetic alg. modulus (explicit)	7	19	8	177100	29565	22179	3638	0.4	475.3	6.8
			9			39404	6491			21.6
			10			66002	10914			64.3
		20	8	230230	38431	22179	3637	0.5	778.6	6.9
			9			39404	6488			20.3
			10			66068	10914			65.9
	9	11	6	75582	12707	24822	3435	0.3	79.9	7.7
			7			51756	8084			32.3
			8			70448	11745			58.3
		12	6	125970	21145	24906	3450	0.3	253.5	7.8
			7			54440	8482			37.4
			8			88642	14207			102.4

Table 1. Experimental results for on-the-fly bisimulation minimisation.

2 and 10 on these examples, even for relatively large values of k on the *Approximate majority* models. Comparing columns 7 and 8 in Table 1 shows that much of the reduction is indeed due to merging of bisimilar states, not just to a k-step truncation of the state space from the backwards traversal.

Regarding performance and scalability, we first discuss results for the SMT-based implementation. We were only able to apply this to the *Tournament* example, where a very large reduction in state space is achieved. On a positive note, the SMT-based approach successfully performs minimisation here and gives a symbolic (Boolean expression) representation for each block. However, the process is slow, limiting applicability to DTMCs that can already be verified without

minimisation. Our experiments showed that the slow performance was largely caused by testing for overlaps between partition blocks resulting in a very large number of calls to the SMT solver.

The explicit-state on-the-fly implementation performed much better and Table 1 shows results for all three models. In particular, for the *Tournament* example, finite-horizon minimisation and verification is much faster than verifying the full model using the fastest engine in PRISM. This is because we can bypass construction of the full models, which have up to 14 million states for this example. For the *Modulus* example, the model reductions obtained are much smaller and, as a result, PRISM is able to build and verify the model faster. However, for the *Approximate Majority* example, the minimisation approach can be applied to larger models than can be handled by PRISM. For this example, although the state spaces of the full model are manageable, the models prove poorly suited to PRISM's model construction implementation (which is based on binary decision diagram data structures).

7 Conclusions

We have presented model reduction techniques for verifying finite-horizon properties on discrete-time Markov chains. We formalised the notion of k-step finite-horizon bisimulation mininisation and clarified the subset of PCTL that it preserves. We have given both a partition-refinement algorithm and an on-the-fly approach, implemented in both a symbolic (SMT-based) and explicit-state manner as an extension of PRISM. Experimental results demonstrated that significant model reductions can be obtained in this manner, resulting in improvements in both execution time and scalability with respect to the existing efficient implementations in PRISM.

Future work in this area will involve extending the techniques to other classes of probabilistic models, and adapting the on-the-fly approaches to preserve the full time-bounded fragment of PCTL, including nested formulae.

Acknowledgements. This work has been supported by the EU-FP7-funded project HIERATIC.

References

1. Aljazzar, H., Fischer, M., Grunske, L., Kuntz, M., Leitner, F., Leue, S.: Safety analysis of an airbag system using probabilistic FMEA and probabilistic counterexamples. In: Proc. QEST'09 (2009)
2. Angluin, D., Aspnes, J., Eisenstat, D.: A simple population protocol for fast robust approximate majority. Distributed Computing 21(2), 87–102 (2008)
3. Aziz, A., Singhal, V., Balarin, F., Brayton, R.K., Sangiovanni-Vincentelli, A.L.: It usually works: The temporal logic of stochastic systems. In: CAV'95 (1995)
4. De Moura, L., Bjørner, N.: Z3: An efficient SMT solver. In: Proc. TACAS'08 (2008)
5. De Moura, L., Bjørner, N.: Satisfiability modulo theories: Introduction and applications. Communications of the ACM 54(9), 69–77 (2011)

6. Dehnert, C., Katoen, J.P., Parker, D.: SMT-based bisimulation minimisation of Markov models. In: Proc. VMCAI'13. pp. 28–47 (2013)
7. Della Penna, G., Intrigila, B., Melatti, I., Tronci, E., Zilli, M.V.: Finite horizon analysis of markov chains with the murϕ verifier. STTT 8(4-5), 397–409 (2006)
8. Derisavi, S.: Signature-based symbolic algorithm for optimal Markov chain lumping. In: Proc. QEST'07. pp. 141–150. IEEE Computer Society (2007)
9. Derisavi, S., Hermanns, H., Sanders, W.H.: Optimal state-space lumping in Markov chains. Information Processing Letters 87(6), 309–315 (2003)
10. Hansson, H., Jonsson, B.: A logic for reasoning about time and reliability. FAC 6(5), 512–535 (1994)
11. Heath, J., Kwiatkowska, M., Norman, G., Parker, D., Tymchyshyn, O.: Probabilistic model checking of complex biological pathways. In: Proc. CMSB'06 (2006)
12. Katoen, J.P., Kemna, T., Zapreev, I., Jansen, D.: Bisimulation minimisation mostly speeds up probabilistic model checking. In: Proc. TACAS'07. vol. 4424 (2007)
13. Katoen, J.P., Zapreev, I.S., Hahn, E.M., Hermanns, H., Jansen, D.N.: The ins and outs of the probabilistic model checker MRMC. Perform. eval. 68(2), 90–104 (2011)
14. Kemeny, J., Snell, J., Knapp, A.: Denumerable Markov Chains. Springer-Verlag, 2nd edn. (1976)
15. Kwiatkowska, M., Norman, G., Parker, D.: PRISM 4.0: Verification of probabilistic real-time systems. In: Proc. CAV'11. LNCS, vol. 6806, pp. 585–591. Springer (2011)
16. Kwiatkowska, M., Norman, G., Sproston, J., Wang, F.: Symbolic model checking for probabilistic timed automata. Information and Computation 205(7) (2007)
17. Kwiatkowska, M., Norman, G., Parker, D.: The PRISM benchmark suite. In: Proc. QEST'12. pp. 203–204 (2012)
18. Larsen, K.G., Skou, A.: Bisimulation through probabilistic testing. Information and Computation 94(1), 1–28 (1991)
19. Paige, R., Tarjan, R.E.: Three partition refinement algorithms. SIAM Journal on Computing 16(6), 973–989 (1987)
20. Valmari, A., Franceschinis, G.: Simple o(m logn) time Markov chain lumping. In: Proc. TACAS'10, pp. 38–52. Springer (2010)
21. Vose, M.: The simple genetic algorithm: Foundations and theory. MIT Press (1999)
22. Wimmer, R., Becker, B.: Correctness issues of symbolic bisimulation computation for markov chains. In: Proc. MMB'10 (2010)
23. `http://www.prismmodelchecker.org/doc/semantics.pdf`
24. `http://www.prismmodelchecker.org/files/spin16fh`