On Incremental Quantitative Verification for Probabilistic Systems

Marta Kwiatkowska, David Parker, Hongyang Qu and Mateusz Ujma
Department of Computer Science, University of Oxford

Abstract

Quantitative verification techniques offer an effective means of computing performance and reliability properties for a wide range of systems. In many cases, it is necessary to perform repeated analyses of a system, for example to identify trends in results, determine optimal system parameters or when performing online analysis for adaptive systems. We argue the need for *incremental* quantitative verification techniques which are able to re-use results from previous verification runs in order to improve efficiency. We report on recently proposed techniques for incremental quantitative verification of Markov decision processes, based on a decomposition of the model into its strongly connected components. We give an overview of the method, describe a number of useful optimisations and show experimental results that illustrate significant gains in run-time performance using the incremental approach.

1 Introduction

Many real-world systems include probability, which is employed to quantify the likelihood of certain events, such as component failure or message loss over a wireless medium, or to obtain efficient randomised solutions to coordinate distributed systems. *Quantitative verification* is an automated method to establish quantitative properties of a system model. In quantitative *probabilistic* verification, models are typically variants of Markov chains and properties are stated in probabilistic temporal logics, which can express performance and reliability properties, for example, the probability of battery power dropping below minimum, the expected time for message delivery and the expected number of messages lost before protocol termination. Tools such as the probabilistic model checker PRISM [25] are widely used in a variety of application domains, including communication protocols, security and planning.

Quantitative verification for probabilistic systems is an extension of conventional, non-probabilistic verification, which, in addition to exhaustive exploration of a system model and graph-based analysis techniques, also requires numerical computation. Typically, this involves the solution of linear equation systems or linear optimisation problems. Although many efficient techniques exist for this purpose, in practice, since system models are large, computational performance of verification quickly becomes a crucial issue. Indeed, the development of efficient and scalable quantitative verification techniques represents one of the major topics of research in this area.

In this paper, we argue the need for *incremental* quantitative verification techniques, that is, those which are able to re-use results from previous verification runs in order to improve efficiency. Such techniques become particularly important in scenarios where verification needs to be performed many times and where fast execution times for verification are required. Below, we describe two specific instances where this is the case.

(i) System parameter exploration. Traditional verification techniques take as input a Boolean-valued system property, which is either proven to hold or shown to be violated, usually with the aid of a counterexample. For quantitative verification of probabilistic systems, properties are often in a numerical form, i.e., "what is the worst-case probability of battery power dropping below minimum?" or "what is the maximum expected time for message delivery?". Furthermore, it is common to evaluate such properties on a system model for a range of parameters of that model. This may help to identify erroneous or anomalous system behaviour. For example, an analysis of the anonymity protocol Crowds [31] demonstrated that, for some measures of anonymity, the algorithm's effectiveness decreases as the number of users increases. It is also common to explore the effect that system parameters have on the overall performance or reliability of a system. For example, in the context of randomised algorithm design, quantitative verification has been used to show that using biased coins can be beneficial for communication protocols [33, 27] and self-stabilisation algorithms [26]. Although some specific techniques have been developed for *parametric* quantitative verification [11, 20, 15], the most common approach in practice is simply to perform multiple verification runs, motivating the need for incremental techniques.

(ii) Predictable adaptive software. *Adaptive* software systems are designed to be able to react to changes in their environment in order to continuously satisfy their system requirement specifications. Quantitative verification has been proposed as a key ingredient of the predictable adaptive software framework of [6, 5], in which requirement specifications include reliability and Quality of Service properties. Quantitative verification is used repeatedly to analyse a system model in order to *detect* when requirements are violated, *predict* when such violations may occur and/or to *plan* self-adaptive actions to remedy the situation. Verification runs need to be executed many times and also in an online fashion, where a timely response is crucial. Furthermore, changes in the system model and its parameters are likely to occur gradually, over time, providing scope for incremental verification to add value. The framework of [6, 5] has been validated with an implementation based on the PRISM model checker [25], but incremental techniques have yet to be applied.

We report on research aimed at developing incremental quantitative verification for probabilistic systems initiated in [28]. The focus is on Markov decision processes, a widely used model that underpins a wide range of systems, including randomised distributed algorithms, communication protocols and planning. We target scenarios where the probability values for certain transitions in the model undergo changes, but the underlying graph structure is not altered. The idea is to re-use results from previous verification runs, based on a decomposition of the model into its strongly connected components (SCCs). To do so, we build upon existing SCC-based approaches to the verification of MDPs presented in [7]. We also describe a number of optimisations that can be used to improve the performance of the SCC-based approach. We evaluate the effectiveness of our incremental verification techniques on a selection of large benchmark case studies, illustrating significant gains in run-time.

Related work. In the context of non-probabilistic verification, A variety of incremental techniques have been proposed, e.g., [32, 9, 23], but typically the problems solved (e.g. speeding up state space generation or model checking of functional properties) are quite different to those for probabilistic systems.

For the quantitative setting, an alternative approach to the incremental techniques described in this paper is the run-time probabilistic model checking technique of [15]. This uses discrete-time Markov chains as system models (which can be seen as a special case of Markov decision processes) and employs *parametric* techniques: building a symbolic expression for the required system property, which can then be evaluated very rapidly for different parameter values. In related work, run-time approaches to verifying Markov decision process models are considered in [14], instead using techniques from control theory.

In fact, the parametric approach to probabilistic verification, as used in [15], is an active area of research. Calculation of symbolic expressions over system parameters was originally proposed by Daws [11] and then improved further in [20, 19], both for the case of discrete-time Markov chains. A closely related problem is the synthesis of the set of parameter values for which a system model satisfies its specification. Methods have been proposed both for time-bounded properties of continuous-time Markov chains [21] and also for Markov decision processes [18].

Finally, we also mention the work of [34], which performs verification in a *run-time* manner, i.e., by monitoring a system during its execution. This approach is also based on the analysis of a probabilistic system model (a hidden Markov model), not because the system being verified is stochastic, but because its state space is only partially observable during monitoring.

2 Quantitative Verification of Markov Decision Processes

We begin with some background material on Markov decision processes and techniques for their verification.

2.1 Markov Decision Processes

Markov decision processes (MDPs) are widely used to model systems that exhibit both probabilistic and nondeterministic behaviour. Real-life systems are often inherently stochastic, for example due to the presence of failures, unpredictable delays or randomisation. In addition, nondeterminism may be essential, for example to capture *concurrency*, i.e., the possible interleavings of multiple components operating in parallel, or *underspecification*, where a probability or other parameter is not known or is not relevant.

Formally, we define MDPs as follows. We let $Dist(S)$ denote the set of all discrete probability distributions over S, i.e., the set of functions $\mu : S \to [0,1]$ such that $\sum_{s\in S} \mu(s) = 1$. An MDP is then a tuple $\mathcal{M} = (S, \overline{s}, Steps, r)$ where:

- S is a finite set of *states*,
- $\overline{s} \in S$ is the *initial state*,
- $Steps : S \to 2^{Dist(S)}$ is a *probabilistic transition function*,
- $r : S \times Dist(S) \to \mathbb{R}_{\geqslant 0}$ is a *reward function*.

The transition probability function $Steps$ maps each state $s \in S$ to a finite, non-empty set $Steps(s)$ of probability distributions. There are two steps to determine the successor of a state s in the MDP: first, a distribution μ is chosen non-deterministically from the set $Steps(s)$; second, the next state s' is chosen randomly according to μ, i.e., the probability of moving to each state s' is given by $\mu(s')$. For simplicity, we do not include action labels in MDPs. Distributions are, however, augmented with *rewards* (sometimes called impulse rewards).

A *path* in an MDP, representing a possible execution of the system being modelled, is a non-empty (finite or infinite) sequence of the form: $s_0 \xrightarrow{\mu_0} s_1 \xrightarrow{\mu_1} s_2 \ldots$ where $s_i \in S$, $\mu_i \in Steps(s_i)$ and $\mu_i(s_{i+1}) > 0$ for all $i \geqslant 0$. We use $\omega(i)$ to denote the $(i{+}1)$th state in the path ω, i.e., $\omega(i) = s_i$, and $step(\omega, i)$ is the distribution taken in state $\omega(i)$, i.e., $step(\omega, i) = \mu_i$. We let $Path_s$ denote the set of all (infinite) paths starting in state s.

In order to reason formally about the probabilistic behaviour of an MDP $\mathcal{M}$, we require the notion of *adversary* (sometimes called strategy, policy or scheduler), which is one possible resolution of the nondeterministic choices in $\mathcal{M}$. Formally, an *adversary* selects an available distribution in each state based on the history of choices made so far. An adversary A restricts the behaviour of the MDP to a set of paths $Path_s^A \subseteq Path_s$. It also induces a probability space [24] $Prob_s^A$ over the paths $Path_s^A$. We use $Adv_{\mathcal{M}}$ to denote the set of all possible adversaries for $\mathcal{M}$.

2.2 Quantitative Verification of MDPs

Usually, properties to be verified against MDPs are expressed in temporal logics, such as PCTL [22, 2] and LTL [29]. These logics can also be augmented to specify reward- or cost-based properties [12, 16]. Performing verification reduces to the computation of a few key properties of MDPs [10, 2, 12, 16]. The first are *minimum* or *maximum reachability probabilities*, i.e., the minimum or maximum probability that a path through the MDP eventually reaches a state in some target set $F \subseteq S$, quantified over all possible adversaries:

$$p_s^{\min}(F) = \inf_{A \in Adv_{\mathcal{M}}} p_s^A(F)$$
$$p_s^{\max}(F) = \sup_{A \in Adv_{\mathcal{M}}} p_s^A(F)$$

where:

$$p_s^A(F) = Prob_s^A(\{\omega \in Path_s^A \mid \exists i \,.\, \omega(i) \in F\})\,.$$

Secondly, we may require the *minimum* or *maximum expected reward* accumulated until target $F \subseteq S$ is reached:

$$e_s^{\min}(F) = \inf_{A \in Adv_{\mathcal{M}}} e_s^A(F)$$
$$e_s^{\max}(F) = \sup_{A \in Adv_{\mathcal{M}}} e_s^A(F)$$

where:

$$e_s^A(F) = \textstyle\int_{\omega \in Path_s^A} r_F(\omega)\, dProb_s^A$$

and $r_F(\omega)$ gives, for any path $\omega \in Path_s^A$, the total reward accumulated along ω until a state in F is reached:

$$r_F(\omega) = \begin{cases} \sum_{i=1}^{n_F} r(\omega(i{-}1), step(\omega, i{-}1)) & \text{if } \exists j \,.\, \omega(j) \in F \\ \infty & \text{otherwise} \end{cases}$$

and $n_F = \min\{j \mid \omega(j) \in F\}$.

For simplicity, in the remainder of this paper, we will focus on the case of maximum reachability probabilities, i.e., computing $p_s^{\max}(F)$, but the techniques described also adapt easily to minimum probabilities and expected rewards.

Throughout the remainder of the paper, we will assume a fixed MDP $\mathcal{M} = (S, \overline{s}, Steps, r)$ and target set F. For clarity, we will abbreviate $p_s^{\max}(F)$ to just $p_s^{\max}$. We use $\underline{p}^{\max}$ to denote the vector of probabilities $p_s^{\max}$ for all states $s \in S$.

Calculation of reachability probabilities (or expected reward values) proceeds in two steps. The first step, referred to as *precomputation*, executes an analysis of the underlying graph of the MDP to identify states that have reachability probabilities of 0 or 1. Second, *numerical computation* is performed to determine values for the remaining states; this can be done with a variety of standard techniques, including value iteration, linear programming and policy iteration. We describe value iteration in more detail in this section since the other techniques discussed in the paper depend on it. Details of the other methods can be found in standard texts on MDPs [30].

Precomputation. This phase is used to partition the state space S into sets S^{no}, S^{yes} and $S^?$, containing states s for which the probability $p_s^{\max}$ is 0, 1 or in $(0,1)$, respectively. Determining these sets is performed using an analysis of the underlying graph of an MDP, that is, the actual probabilities of transition between states are unimportant, only the existence of a transition. We omit here the precise details of the precomputation algorithms to determine sets S^{no}, S^{yes}. See, for example, Section 4.1 of [16] for an explanation.

Value iteration. One commonly used method to compute the probabilities $p_s^{\max}$ for the remaining states $s \in S^?$ is to use *value iteration*, an iterative numerical method which can approximate the values up to some desired accuracy. In practice, this method is widely used since it scales well to large MDPs.

Value iteration works by computing a sequence of vectors $\underline{p}^{\max,k}$ for increasing k. Initially, i.e., for the case $k=0$, we set $p_s^{\max,0}$ to 1 if $s \in S^{yes}$ and 0 otherwise. Then, the kth iteration of computation is defined, for each $s \in S$, as:

$$p_s^{\max,k} \quad := \quad \begin{cases} 1 & s \in S^{yes} \\ 0 & s \in S^{no} \\ \max\limits_{\mu \in Steps(s)} \sum\limits_{s' \in S} \mu(s') \cdot p_{s'}^{\max,k-1} & s \in S^?. \end{cases} \tag{1}$$

The sequence of vectors $\underline{p}^{\max,k}$ is guaranteed to converge eventually to $\underline{p}^{\max}$. In practice, though, the computation is terminated when a pre-specified convergence criterion is met. One common approach is to check that the maximum (absolute) difference between the corresponding elements of successive vectors is below some fixed threshold δ, i.e.:

$$\max_{s\in S} |p_s^{\max,k} - p_s^{\max,k-1}| < \delta.$$

Another is to check the maximum *relative* difference:

$$\max_{s\in S} |(p_s^{\max,k} - p_s^{\max,k-1})/p_s^{\max,k}| < \delta.$$

A useful optimisation for value iteration is the so-called *Gauss-Seidel* variant of the algorithm. This increases the rate of convergence of the computation by using the most up-to-date probability values that are available for each state. More precisely, when computing $p_s^{\max,k}$ in the kth step of value iteration, the values $p_{s'}^{\max,k-1}$ in (1) can be replaced with $p_{s'}^{\max,k}$ for states s' where this value has already been computed. See, for example, [30, 16] for more detail.

2.3 SCC-based Value Iteration

We now describe an improved version of value iteration, first presented in [7], based on a decomposition of the MDP $\mathcal{M}$ to be analysed. The first step of this process is to remove *maximal end components* (MECs). An *end component* [12] of $\mathcal{M}$ is a pair $(S', Steps')$, where $S' \subseteq S$ and $Steps'(s) \subseteq Steps(s)$, which is closed and strongly connected, i.e.:

1. $\forall \mu \in Steps'(s), \forall s' \in S \ . \ (\mu(s') > 0 \rightarrow s' \in S')$
2. $\forall s, s' \in S'$, there is a path in $(S', Steps')$ from s to s'.

A *maximal* end component is one for which there is no larger end component that contains it. It is known [12] that all states s within an MEC have the same probability value $p_s^{\max}$. Furthermore, we can safely compress each MEC into a single state [7]. From this point on, we assume that all MECs have already been compressed in this way.

Algorithm 1 The k-th iteration (for states in C_i)

```
1: $X := \{x \in C_i \mid p_x^{\max,k-1} - p_x^{\max,k-2} \geqslant \delta\}$
2: for all $x \in X$ do
3:   $Y := \{y \in C_i \mid p_y^{\max,k-1} < 1$ and $\exists \mu \in Steps(y)\ .\ \mu(x) > 0\}$
4:   for all $y \in Y$ do
5:     $Steps'(y) := \{\mu \in Steps(y) \mid \mu(x) > 0\}$
6:     $p_y^{\max,k} := \max_{\mu \in Steps'(y)} \sum_{s' \in S} \mu(s') \cdot p'_{s'}$
7:   end for
8: end for
```

Next, we identify *strongly connected components* (SCCs) in the MDP. An SCC C is a set of states that is strongly connected (there is a path between any two states in C) and maximal (no superset of C is also strongly connected).

SCCs are particularly important in value iteration. Let C be an SCC, and $Pre^*(C) \subseteq S\backslash C$ be the set of states that can reach C, but are not contained within it. Any change of a state's probability value in C affects probability values of all other states in C, as well as those of states in $Pre^*(C)$. Furthermore, until the probability values of the states in C converge, the probability values of states in $Pre^*(C)$ cannot converge. In fact, the computation of probability values for states in $Pre^*(C)$ can be postponed until the values in C converge [7].

The set of SCCs in $\mathcal{M}$ forms a partition of its states S. Let $\Pi = \{C_1, \ldots, C_m\}$ be this partition. The *successor set* $Succ(C_i)$ of C_i is the set of states outside C_i that are immediate successors of states in C_i. We say that C_i *depends* on C_j if $Succ(C_i) \cap C_j \neq \emptyset$. As there is no cyclic dependence among SCCs, we generate a *reversed topological order* $\mathcal{C}$ among SCCs such that C_j will appear before C_i in $\mathcal{C}$ if C_i depends on C_j. The set of SCCs, along with their topological order, can be obtained efficiently with well-known methods such as the Tarjan algorithm [35], whose time and space complexity is linear in the size of the model.

SCC-based value iteration processes each SCC separately, according to the ordering $\mathcal{C}$, and then terminates. For each SCC, a sequence of approximations is computed, like for value iteration. For each state s in an SCC, $p_s^{\max,k}$ denotes the value computed for s in the kth iteration and $p_s^{\max,0}$ the initial value for s. For any SCC, we set $p_s^{\max,0}$ to 1 if $s \in S^{yes}$ and 0 otherwise. We also let $p_s^{\max}$ denote the final value for s. Consider now a particular SCC C_i. The first iteration is performed as follows. For each $s \in C_i$:

$$p_s^{\max,1} \quad := \quad \begin{cases} \max\limits_{\mu \in Steps'(s)} \sum\limits_{s' \in S} \mu(s') \cdot p'_{s'} & p_s^{\max,0} < 1 \wedge Steps'(s) \neq \emptyset \\ p_s^{\max,0} & \text{otherwise.} \end{cases}$$

where $Steps'(s) = \{\mu \in Steps(s) \mid \exists s' \in Succ(C_i)\ .\ \big(\mu(s') > 0 \wedge p_{s'}^{\max} > 0\big)\}$ and $p'_{s'}$ is $p_{s'}^{\max}$ if $s' \in Succ(C_i)$, or $p_{s'}^{\max,0}$ otherwise.

In the remaining iterations, we only update probabilities for states that are affected by the previous iteration. Other states simply keep their probability from the previous iteration. Algorithm 1 describes the k-th iteration (for $k > 1$), where $p'_{s'} = p_{s'}^{\max}$ if $s' \in Succ(C_i)$, and otherwise $p'_{s'} = p_{s'}^{\max,k-1}$. The iteration on C_i terminates at the k-th iteration when X in Algorithm 1 is empty. Note that Algorithm 1 also works when we use δ as a maximum relative difference, e.g., the condition $p_x^{\max,k-1} - p_x^{\max,k-2} \geqslant \delta$ in Algorithm 1 can be replaced by $\frac{p_x^{\max,k-1} - p_x^{\max,k-2}}{p_x^{\max,k-2}} \geqslant \delta$.

Algorithm 2 Generate $\overline{\Pi}$

```
1: $\overline{\Pi} := \emptyset$
2: for all $i \in 1, \ldots, m$ do
3:     if $\exists s \in C_i \,.\, \mathit{Steps}(s) \neq \overline{\mathit{Steps}}(s)$ or $\exists C \in \overline{\Pi} \,.\, \mathit{Succ}(C_i) \cap C \neq \emptyset$ then
4:         $\overline{\Pi} := \overline{\Pi} \cup \{C_i\}$
5:     end if
6: end for
```

3 Incremental Verification of MDPs

We now describe an *incremental* approach to quantitative verification of MDPs [28], which builds on the SCC-based version of value iteration outlined above. Incremental verification techniques aim to accelerate the process of analysing a model that has undergone minor changes, by exploiting the presence of existing verification results.

The techniques that we describe here target cases where the probabilities of some transitions in an MDP undergo changes. It is assumed, though, that the transition structure of the model remains untouched. This means that transitions with probability one or zero cannot be changed; otherwise, some transitions with non-zero probability would be added or deleted from the model. We use $\mathcal{M} = (S, \bar{s}, \mathit{Steps}, r)$ to denote the original MDP and $\overline{\mathcal{M}} = (S, \bar{s}, \overline{\mathit{Steps}}, r)$ for the modified one. Notice that only *Steps* is modified.

3.1 Incremental SCC-based Value Iteration

When some probabilities in *Steps* are changed, it may be unnecessary to recompute probability values for all states. We first identify the set $\overline{\Pi}$ of SCCs that have been affected by the changes. It can be generated using Algorithm 2.

First, $\overline{\Pi}$ is initialised to an empty set. Then, we scan the SCC partition according to the reverse topological order and add C_i to $\overline{\Pi}$ if C_i satisfies one of two conditions:

1. There exists a state $s \in C_i$ such that one distribution from s is involved in the changes;
2. There exists an SCC $C \in \overline{\Pi}$ that C_i depends on.

Let $p_s^{\max}$ be the maximum probability for state s computed previously on $\mathcal{M}$ and $\overline{p}_s^{\max}$ the one we need to compute after the changes occur. The SCC-based value iteration algorithm of Section 2.3 can be adapted to handle changes in probabilities by replacing Π by $\overline{\Pi}$ and initialising $\overline{p}_s^{\max}$ as follows:

$$\overline{p}_s^{\max,0} \quad := \quad \begin{cases} 1 & s \in S^{yes} \\ 0 & s \in S^{no} \\ p_s^{\max} & s \in S^{?} \text{ and } s \in C \text{ for some } C \in \Pi \backslash \overline{\Pi} \\ 0 & \text{otherwise.} \end{cases}$$

In addition, before we recompute the probability for an SCC C in $\overline{\Pi}$, we perform a test on its successor set $\mathit{Succ}(C)$. This test checks the following conditions:

1. the probability for each state $s \in \mathit{Succ}(C)$ is unaffected by the changes, i.e.:

$$\forall s \in \mathit{Succ}(C) \,.\, \overline{p}_s^{\max} = p_s^{\max}, \tag{2}$$

2. all distributions from a state in C are unaffected by the changes, i.e.:

$$\forall s \in C \;.\; \overline{Steps}(s) = Steps(s).$$

If both conditions hold, there is no need to recompute values in this SCC, i.e. we simply use:

$$\forall s \in C \;.\; \overline{p}_s^{\max} = p_s^{\max}.$$

Although the above test can eliminate unnecessary recomputation for SCCs that might be affected by the changes, condition (2) is quite restrictive since it requires all states in the successor set to have the same probability as before the changes occurred. Recomputation is executed even if, for all states in $Succ(C)$, there are only very small changes, e.g., $|\overline{p}_s^{\max} - p_s^{\max}| \in (0, \epsilon)$ for some small $\epsilon > 0$.

In this case, the change in the probability for a state in C is bounded by ϵ with respect to its original value. If ϵ is less than the required accuracy, we can use $p_s^{\max}$ as $\overline{p}_s^{\max}$ for state s in C, which speeds up the recomputation by introducing a small approximation error. Lemma 1 formalises this idea.

Lemma 1 ([28])**.** *Consider the SCC-based version of value iteration from above.*

1. *If the condition $\overline{p}_s^{\max} = p_s^{\max}$ in condition (2) is replaced by $|\overline{p}_s^{\max} - p_s^{\max}| < \epsilon$ and the test succeeds, then:*
$$\forall x \in C \;.\; |\overline{p}_x^{\max} - p_x^{\max}| < \epsilon. \tag{3}$$

2. *If condition $\overline{p}_s^{\max} = p_s^{\max}$ is replaced by $|\frac{\overline{p}_s^{\max} - p_s^{\max}}{p_s^{\max}}| < \epsilon$ and the above test succeeds, then:*
$$\forall x \in C \;.\; |\frac{\overline{p}_x^{\max} - p_x^{\max}}{p_x^{\max}}| < \epsilon. \tag{4}$$

In practice, we can use δ, the maximum absolute difference or maximum relative difference, as ϵ, or a smaller value than δ to increase the accuracy, but possibly also increase computation time.

3.2 Further Optimisations to SCC-based Value Iteration

The performance of SCC-based value iteration, with or without the incremental approach described above, can be further optimised in several ways.

Parallelisation. First, the decomposition of verification presents opportunities for parallelisation, which is particularly desirable to exploit, given the increasing prevalence of multi-core architectures in mainstream CPU design. The topological order among SCCs provides a natural structure for parallel computation. At any step, an SCC can be processed independently (and thus in parallel), as long as all of its successor sets have been processed. To achieve this, we need a queue to store SCCs that are ready to be processed. Initially, all SCCs that have an empty successor set are put in the queue. Each computation thread takes one SCC from the queue to process, and when it is done, it puts SCCs that newly become ready into the queue. The whole process terminates when the queue is empty. Let $\overline{Succ}(C)$ be a copy of the successor set $Succ(C)$ of an SCC C in Π. Algorithm 3 shows the procedure for parallel computation. Note that, in the **while** loop, only line 5 can be executed in parallel. As an additional optimisation, MECs identification can also be parallelised. This is done by first partitioning into SCCs, then searching each one for MECs in parallel.

Algorithm 3 Parallel processing of SCCs

```
1: $Queue := \{C_i \in \Pi \mid \overline{Succ}(C_i) = \emptyset\}$
2: $\Pi := \Pi \backslash Queue$
3: while $Queue \neq \emptyset$ do
4:   $scc :=$ the head of $Queue$
5:   compute maximum probabilities for states in $scc$
6:   for all $C \in \Pi$ . $\overline{Succ}(C) \cap scc \neq \emptyset$ do
7:     $\overline{Succ}(C) := \overline{Succ}(C) \backslash scc$
8:     if $\overline{Succ}(C) = \emptyset$ then
9:       $\Pi := \Pi \backslash \{C\}$
10:      $Queue := Queue \cup \{C\}$
11:    end if
12:  end for
13: end while
```

State ordering heuristics. SCC-based value iteration works by exploiting the topological ordering of SCCs, as identified by Tarjan's algorithm for SCC detection. As mentioned in Section 2.2, implementations of value iteration can in general also be improved using Gauss-Seidel schemes, which re-use the latest values for each state that is available during each iteration. These can therefore be used when computing the values for the states within a single SCC.

Gauss-Seidel schemes are, by their nature, sensitive to the order in which states are updated during value iteration. In [28], the order taken is arbitrary: for convenience, the implementation simply uses the order in which states had been created during model construction. However, this may not reflect the way that states are connected. Here, we consider an additional improvement, namely, we use the order in which states were visited during the Tarjan SCC detection algorithm (which is based on a depth-first search). In our experiments, this gave the greatest improvement in the speed of convergence of value iteration.

Eliminating precomputation. As discussed in [28], the precomputation phase of MDP verification (which identifies states satisfying a property with probability exactly 0 or 1) is sometimes relatively expensive in comparison to value iteration. However, SCC-based decomposition of an MDP yields the possibility to replace the standard precomputation algorithms with simpler and cheaper checks, resulting in significant improvements in performance. We refer the reader to [28] for more details.

Symbolic SCC identification. Finally, we discuss another technique from [28]: a *symbolic* version of SCC identification. Using standard explicit-state data structures to store the state space and transition relation of an MDP can limit the size of models that can be handled. Tools like PRISM make use of symbolic implementations, based on binary decision diagrams (BDDs) [4] and extensions such as multi-terminal BDDs (MTBDDs) [8, 1]. Symbolic versions of precomputation algorithms and value iteration already exist, but Tarjan's SCC identification algorithm is known to be poorly suited to these data structures. Various BDD-based versions of SCC identification have been developed [3, 17], but these make it inefficient to generate the topological ordering of SCCs, as required here. Thus, [28] develops a "hybrid" version of the Tarjan algorithm that combines symbolic and explicit-state data structures. Again, we refer the reader to [28] for full details of the implementation and resulting gains in performance.

3.3 Experimental Results

The techniques described in this paper have been implemented in an extension of PRISM [25], using its explicit-state probabilistic model checking library. We summarise here the performance of the techniques on four case studies:

- *zeroconf*: a model of the Zeroconf dynamic network configuration protocol. We compute "the maximum probability of the protocol correctly configuring a local network address within time T".
- *consensus*: a model of the shared coin protocol used in Aspnes & Herlihy's randomised consensus algorithm. The algorithm allows N processes in a distributed network to reach a consensus. We compute "the maximum probability of terminating without consensus being reached".
- *wlan*: a model of an IEEE 802.11 Wireless LAN featuring two stations sending data over a shared channel, each with a backoff counter of size N. We compute "the maximum probability of the backoff counters for both stations reaching their maximum value".
- *mer*: a simple model based on the flight software for JPL's Mars Exploration Rovers (MER), in which N threads compete for a set of R resources. We compute "the minimum probability that mutual exclusion is not violated within 2 cycles of system execution".

The first three models are taken from the PRISM benchmark suite [36]; the fourth is taken from [13]. The reader is referred to these sources for details of any model parameters not explained here (e.g. K for *zeroconf*, K for *consensus*) and for the models themselves. The experiments were performed on a 2.80GHz PC with 6 cores and 32GB RAM, running 64-bit Fedora.

To demonstrate the incremental verification algorithm without bias, we use the following scheme to make small changes to an MDP's transition probabilities. We randomly choose three states that are not in any maximal end component and have a distribution with probabilistic choices. For each state s, we pick such a distribution $\mu \in \mathit{Steps}(s)$ and modify the probability distribution as follows. Assume there exist m ($m > 1$) states $s_1, \ldots, s_m \in S$ such that $\mu(s_i) > 0$ for $1 \leqslant i \leqslant m$. The new distribution μ' is such that, for $1 \leqslant i \leqslant m-1$, we keep half of the value, i.e., $\mu'(s_i) = \mu(s)/2$; for $i = m$, we increase the value such that $\mu'(s_m) = \mu(s_m)+\sum_{i=1}^{m-1} \mu(s_i)/2$.

Table 1 summarises the results. For each example, the first three columns show the name and parameters of the model, and the number of states in the MDP. The remaining columns show the total time required to verify a single property, using four different variants of value iteration:

(i) the standard algorithm (see Section 2.2);

(ii), (iii) the SCC-based version of [7, 28], using parallelisation, without and with the state ordering heuristic described in Section 3.2, respectively;

(iv) the incremental algorithm (see Section 3.1), with parallelisation and the state ordering heuristic.

Where parallelisation is employed, we use 12 threads, i.e. 2 per core. It should also be noted that the results shown here are based on an improved implementation of parallelisation, compared to the one originally presented in [28].

Generally, we see that these four variants perform increasingly well in terms of verification time. Firstly, we see significant gains when using the SCC-based approach (including the faster

Model			Time(s)			
Name	Parameters	States	Original	SCC-based (parallel)		Incremental (parallel, ordered)
				Non-ordered	Ordered	
zeroconf $[K, T]$	2, 10	665,567	122	6.1	5.1	0.5
	2, 14	1,061,771	214	8.6	7.1	0.5
	3, 10	949,912	198	8.3	4.4	0.8
	3, 14	1,735,014	354	16.6	10.1	1.3
	4, 10	976,247	265	9.4	4.6	1.1
	4, 14	2,288,771	575	24.6	14.1	1.9
consensus $[N, K]$	2, 12	24,678	26	2.3	1.9	0.1
	2, 16	32,486	63	3.9	2.3	1.4
	2, 20	40,294	131	6.4	3.5	1.8
	3, 1	729,337	33	3.3	3.6	0.4
	3, 2	1,418,545	168	9.1	8.6	0.4
	3, 3	2,259,817	478	16.3	9.4	0.4
wlan $[N]$	2	28,480	0.3	0.4	0.4	0.1
	3	96,302	1.3	0.7	0.8	0.1
	4	345,000	8.6	2.4	1.7	0.4
	5	1,295,218	59	7.9	8.1	1.7
mer $[R, N]$	2, 100	592,264	20	3.4	3.1	0.4
	2, 200	1,182,964	39	6.1	5.9	1.0
	3, 300	1,773,664	59	15.0	13.3	1.5
	4, 400	2,364,364	86	20.3	19.6	2.0
	5, 500	2,955,064	97	16.2	14.9	2.6

Table 1: Performance comparison for SCC-based and incremental verification methods

precomputation phase and parallelisation of Section 3.2) compared to the standard algorithm. Next, we note that the state ordering heuristic offers (in almost all cases) an additional small gain. Lastly, and most importantly, we see that incremental verification is much faster than any other option. Comparing to the fastest of the three non-incremental variants (option (iii) above), we see an average speed-up by more than a factor of ten. In comparison to the original version of value iteration, the incremental algorithm is often hundreds of times faster.

4 Conclusions and Further Directions

We have argued the need for incremental approaches to quantitative verification of probabilistic systems, citing several scenarios where multiple verification runs need to be performed, after only small changes to the model being verified, and often with rapid response times required for the results of verification.

We reported on improvement to the recent work from [28], which performs incremental verification on Markov decision processes by exploiting a model's decomposition into strongly connected components, and demonstrated significant gains in run-time for incremental verification of MDPs in which localised changes were made to some transition probabilities.

There are several additional directions that warrant investigation. For example, we plan to develop incremental techniques for models that undergo structural alterations, as well as changes to their transition probabilities. We will also investigate incremental model construction. For both of these directions, we hope to exploit information about modifications to the system expressed in terms of their high-level model descriptions (e.g. the PRISM modelling language, in the current implementation), rather than a low-level examination of the changes to states and transitions. Another important direction is the development of incremental techniques for other classes of probabilistic models, in particular those that include timing information such as continuous-time Markov chains or probabilistic timed automata.

Acknowledgments. The authors are part supported by ERC Advanced Grant VERIWARE, EU FP7 project CONNECT (IST 231167), EPSRC project EP/F001096/1 and DARPA/Air Force Research Laboratory contract FA8650-10-C-7077 (PRISMATIC).

References

[1] 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.

[2] A. Bianco and L. de Alfaro. Model checking of probabilistic and nondeterministic systems. In P. Thiagarajan, editor, *Proc. 15th Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS'95)*, volume 1026 of *LNCS*, pages 499–513. Springer, 1995.

[3] R. Bloem, H. Gabow, and F. Somenzi. An algorithm for strongly connected component analysis in n log n symbolic steps. In *Proc. 3rd International Conference on Formal Methods in Computer-Aided Design (FMCAD'00)*, pages 37–54, 2000.

[4] R. Bryant. Graph-based algorithms for Boolean function manipulation. *IEEE Transactions on Computers*, C-35(8):677–691, 1986.

[5] R. Calinescu, C. Ghezzi, M. Kwiatkowska, and R. Mirandola. Self-adaptive software needs quantitative verification at runtime. *Communications of the ACM*, 2012. To appear.

[6] R. Calinescu, L. Grunske, M. Kwiatkowska, R. Mirandola, and G. Tamburrelli. Dynamic QoS management and optimisation in service-based systems. *IEEE Transactions on Software Engineering*, 37(3):387–409, 2011.

[7] F. Ciesinski, C. Baier, M. Größer, and J. Klein. Reduction techniques for model checking Markov decision processes. In *Proc. 5th International Conference on Quantitative Evaluation of Systems (QEST'08)*, pages 45–54. IEEE CS Press, 2008.

[8] 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.

[9] C. L. Conway, K. S. Namjoshi, D. Dams, and S. A. Edwards. Incremental algorithms for interprocedural analysis of safety properties. In *Proc. CAV'05*, volume 3576 of *LNCS*, pages 449–461. Springer, 2005.

[10] C. Courcoubetis and M. Yannakakis. The complexity of probabilistic verification. *Journal of the ACM*, 42(4):857–907, 1995.

[11] C. Daws. Symbolic and parametric model checking of discrete-time markov chains. In Z. Liu and K. Araki, editors, *Proc. 1st Int Coll. Theoretical Aspects of Computing (ICTAC 2004)*, volume 3407 of *LNCS*, pages 280–294. Springer, 2004.

[12] L. de Alfaro. *Formal Verification of Probabilistic Systems.* PhD thesis, Stanford University, 1997.

[13] L. Feng, M. Kwiatkowska, and D. Parker. Automated learning of probabilistic assumptions for compositional reasoning. In D. Giannakopoulou and F. Orejas, editors, *Proc. 14th International Conference on Fundamental Approaches to Software Engineering (FASE'11)*, volume 6603 of *LNCS*, pages 2–17. Springer, 2011.

[14] A. Filieri, C. Ghezzi, A. Leva, and M. Maggio. Self-adaptive software meets control theory: A preliminary approach supporting reliability requirements. In *Proc. 26th IEEE/ACM International Conference on Automated Software Engineering (ASE'11)*, pages 283–292, 2011.

[15] A. Filieri, C. Ghezzi, and G. Tamburrelli. Run-time efficient probabilistic model checking. In *Proc. 33rd ACM/IEEE International Conference on Software Engineering (ICSE'11)*. ACM, 2011.

[16] V. Forejt, M. Kwiatkowska, G. Norman, and D. Parker. Automated verification techniques for probabilistic systems. In M. Bernardo and V. Issarny, editors, *Formal Methods for Eternal Networked Software Systems (SFM'11)*, volume 6659 of *LNCS*, pages 53–113. Springer, 2011.

[17] R. Gentilini, C. Piazza, and A. Policriti. Computing strongly connected components in a linear number of symbolic steps. In *Proc. SODA'03*, pages 573–582, 2003.

[18] E. M. Hahn, T. Han, and L. Zhang. Synthesis for PCTL in parametric Markov decision processes. In *Proc. 3rd NASA Formal Methods Symposium (NFM'11)*, volume 6617 of *LNCS*. Springer, 2011.

[19] E. M. Hahn, H. Hermanns, B. Wachter, and L. Zhang. PARAM: A model checker for parametric Markov models. In *Proc. 22nd International Conference on Computer Aided Verification (CAV'10)*, volume 6174 of *LNCS*, pages 660–664. Springer, 2010.

[20] E. M. Hahn, H. Hermanns, and L. Zhang. Probabilistic reachability for parametric Markov models. In C. Pasareanu, editor, *Proc. 16th International SPIN Workshop*, volume 5578 of *LNCS*, pages 88–106. Springer, 2009.

[21] T. Han, J.-P. Katoen, and A. Mereacre. Approximate parameter synthesis for probabilistic time-bounded reachability. In *Proc. IEEE Real-Time Systems Symposium (RTSS 08)*, pages 173–182. IEEE CS Press, 2008.

[22] H. Hansson and B. Jonsson. A logic for reasoning about time and reliability. *Formal Aspects of Computing*, 6(5):512–535, 1994.

[23] K. Heljanko, T. Junttila, and T. Latvala. Incremental and complete bounded model checking for full PLTL. In *CAV 05*, volume 3576 of *LNCS*, pages 98–111. Springer, 2005.

[24] J. Kemeny, J. Snell, and A. Knapp. *Denumerable Markov Chains*. Springer-Verlag, 2nd edition, 1976.

[25] M. Kwiatkowska, G. Norman, and D. Parker. PRISM 4.0: Verification of probabilistic real-time systems. In G. Gopalakrishnan and S. Qadeer, editors, *Proc. 23rd International Conference on Computer Aided Verification (CAV'11)*, volume 6806 of *LNCS*, pages 585–591. Springer, 2011.

[26] M. Kwiatkowska, G. Norman, and D. Parker. Probabilistic verification of herman's self-stabilisation algorithm. *Formal Aspects of Computing*, 2012. To appear.

[27] M. Kwiatkowska, G. Norman, and J. Sproston. Probabilistic model checking of deadline properties in the IEEE 1394 FireWire root contention protocol. *Formal Aspects of Computing*, 14(3):295–318, 2003.

[28] M. Kwiatkowska, D. Parker, and H. Qu. Incremental quantitative verification for Markov decision processes. In *Proc. IEEE/IFIP International Conference on Dependable Systems and Networks (DSN-PDS'11)*, pages 359–370. IEEE CS Press, 2011.

[29] A. Pnueli. The temporal logic of programs. In *Proc. 18th Annual Symposium on Foundations of Computer Science (FOCS'77)*, pages 46–57. IEEE Computer Society Press, 1977.

[30] M. Puterman. *Markov Decision Processes: Discrete Stochastic Dynamic Programming*. John Wiley and Sons, 1994.

[31] V. Shmatikov. Probabilistic model checking of an anonymity system. *Journal of Computer Security*, 12(3/4):355–377, 2004.

[32] O. V. Sokolsky and S. A. Smolka. Incremental model checking in the modal mu-calculus. In *CAV 94*, volume 818 of *LNCS*, pages 351–363. Springer, 1994.

[33] M. Stoelinga. *Alea jacta est: Verification of probabilistic, real-time and parametric systems*. PhD thesis, University of Nijmegen, 2002.

[34] S. Stoller, E. Bartocci, J. Seyster, R. Grosu, K. Havelund, S. Smolka, and E. Zadok. Runtime verification with state estimation. In *Proc. 2nd International Conference on Runtime Verification (RV'11)*, 2011.

[35] R. Tarjan. Depth-first search and linear graph algorithms. *SIAM Journal on Computing*, 1:146–160, 1972.

[36] `http://www.prismmodelchecker.org/benchmarks/`.