Probabilistic Model Checking of Perturbed MDPs with Applications to Cloud Computing

Yamilet R. Serrano Llerena
National University of Singapore
Singapore
yserrano@comp.nus.edu.sg

Guoxin Su
University of Wollongong
Wollongong, Australia
guoxin@uow.edu.au

David S. Rosenblum
National University of Singapore
Singapore
david@comp.nus.edu.sg

ABSTRACT

Probabilistic model checking is a formal verification technique that has been applied successfully in a variety of domains, providing identification of system errors through quantitative verification of stochastic system models. One domain that can benefit from probabilistic model checking is cloud computing, which must provide highly reliable and secure computational and storage services to large numbers of mission-critical software systems.

For real-world domains like cloud computing, external system factors and environmental changes must be estimated accurately in the form of probabilities in system models; inaccurate estimates for the model probabilities can lead to invalid verification results. To address the effects of uncertainty in probability estimates, in previous work we have developed a variety of techniques for *perturbation analysis* of discrete- and continuous-time Markov chains (DTMCs and CTMCs). These techniques determine the consequences of the uncertainty on verification of system properties. In this paper, we present the first approach for perturbation analysis of Markov decision processes (MDPs), a stochastic formalism that is especially popular due to the significant expressive power it provides through the combination of both probabilistic and nondeterministic choice.

Our primary contribution is a novel technique for efficiently analyzing the effects of perturbations of model probabilities on verification of reachability properties of MDPs. The technique heuristically explores the space of *adversaries* of an MDP, which encode the different ways of resolving the MDP's nondeterministic choices. We demonstrate the practical effectiveness of our approach by applying it to two case studies of cloud systems.

CCS CONCEPTS

• **Mathematics of computing** → **Markov processes**; • **Theory of computation** → **Verification by model checking**; • **Software and its engineering** → **Formal software verification**; *Model checking*; • **Networks** → Cloud computing;

KEYWORDS

cloud computing, Markov decision processes, perturbation analysis, probabilistic model checking, uncertainty

ACM Reference format:
Yamilet R. Serrano Llerena, Guoxin Su, and David S. Rosenblum. 2017. Probabilistic Model Checking of Perturbed MDPs with Applications to Cloud Computing. In *Proceedings of 2017 11th Joint Meeting of the European Software Engineering Conference and the ACM SIGSOFT Symposium on the Foundations of Software Engineering, Paderborn, Germany, September 4–8, 2017 (ESEC/FSE'17),* 11 pages.
https://doi.org/10.1145/3106237.3106301

1 INTRODUCTION

Probabilistic model checking is a formal verification technique that aims to verify systems that exhibit stochastic behavior [2, 19]. It is a mature verification technology with many available tools, such as PRISM [20] . Probabilistic model checking has been applied successfully in a variety of domains, providing identification of system errors through quantitative verification of properties (typically expressed in PCTL) against stochastic system models (typically expressed in some variant of a Markov model).

One domain that can benefit from probabilistic model checking is cloud computing, which has matured extensively and has experienced great success over the last several years [36]. Cloud systems must provide highly reliable and secure computational and storage services to large numbers of mission-critical software systems in highly stochastic environments. In particular, they support dynamic and elastic provision of storage and compute services in environments that endure intermittent connectivity, and variable reliability and availability [26, 36]. Recent studies [17, 18, 23–25] have demonstrated that quantitative verification via probabilistic model checking can provide cloud administrators with an effective way of analyzing operations, resources, and the reliability of provided services.

Traditional stochastic formalisms are limited by their use of *constant*-valued probabilities. This limits the effectiveness of probabilistic model checking for real-world domains such as cloud computing, because the probabilities that represent certain system quantities such as reliability, fault-proneness and so on, are unknown a priori and must be determined empirically through observations of the system or from the experience of domain experts. Estimating these probabilities accurately is complicated by the fact that they may change over time due to the stochastic nature of the modeled system. They may be affected by unpredictable environmental factors such as failures, disasters and workload spikes. Consequently, uncertainty about model probabilities can lead to invalid verification results.

For the reasons outlined above, in order to support accurate and effective probabilistic verification of reliable and secure cloud services for system providers and users, it is necessary to understand *the effect of uncertainty on the probabilistic verification of cloud computing systems*. This paper aims to enable cloud administrators to verify reachability properties of cloud services in the presence of such uncertainty.

In order to address the effects of uncertainty in probabilistic verification, in previous work we have developed a variety of techniques for perturbation analysis of discrete- and continuous-time Markov chains (DTMCs and CTMCs). These techniques can determine the consequences of the uncertainty on verification of system properties [30–32]. In this paper, we present the first approach for perturbation analysis of Markov decision processes (MDPs), a stochastic formalism that is especially popular due to the significant expressive power it provides through the combination of both probabilistic and nondeterministic choice. Our approach is applied in three steps. First, we construct an MDP model that captures key behavioral characteristics of interest of a cloud system, and we specify the reachability properties the model must satisfy. Second, we extend the MDP model to a Parametric Markov decision process (PMDP), a parametric variant of an MDP, to model uncertainty in the MDP's probabilities. Third, we formally characterize the worst-case consequences of the uncertainty on the verification of the reachability properties; in particular, we capture the effect of uncertainty in the form of *condition numbers*, which express the maximum variation to verification results as a linear function of the perturbation of model probabilities. The main technical contributions of our approach are as follows:

(1) We present an efficient heuristic algorithm for exploring the space of *adversaries* of a PMDP, which encode the different ways of resolving the PMDP's nondeterministic choices. The algorithm exploits the underlying graph of the PMDP in an effort to overcome the intractability of analyzing the space of adversaries exhaustively.
(2) We present a prototype implementation of our approach in Python, which is interfaced with PRISM [20] to extract the basic features of an MDP model.
(3) We present an experimental evaluation of our approach on two case studies of cloud systems. Our experiments indicate that our technique is able to compute an accurate estimation of the effect of the uncertainty on the reachability verification of an MDP regardless of the model size and with efficient performance.

The remainder of the paper is organized as follows. Section 2 provides a motivating example of a cloud computing system. Section 3 recalls the necessary formal foundation with respect to MDPs, its parametric variant and its reachability verification in probabilistic model checking. We present our perturbation approach in Section 4. Section 5 presents our algorithm for computing the maximum condition number of an MDP. Section 6 presents our experimental evaluation and results. Section 7 discusses related work. And Section 8 summarizes the main contributions of the paper and discusses future work.

2 MOTIVATING EXAMPLE

Consider a real-life scenario in cloud service migration [18, 34, 35] where the objective is to enable a user always to be connected to the optimal cloud. Suppose we have a mobile network with a typical 3GPP network topology as illustrated in Figure 1, which employs the concept of Follow-Me Cloud (FMC) [33] to provide optimized, disruption-free cloud-based services for mobile users [34]. As shown in Figure 1, a 3GPP network topology typically is divided into hexagonal cells. For the sake of simplicity, we consider a topology of $N = 5$ rings of cells, where i, j indicates the j-th cell in ring i such that $0 \leq i \leq N - 1$. Each cell is covered by one of a number of micro-clouds (MCs) that are interconnected with each other and monitored by the FMC controller.

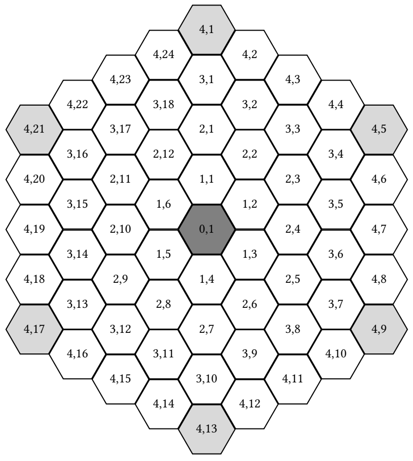

Figure 1: A Typical 3GPP Mobile Network.

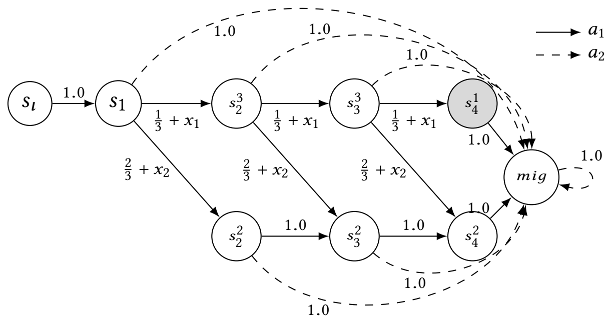

Figure 2: Mobile Network Modeled as a MDP.

We wish to model the service migration decisions of the FMC controller, which decides in each time slot whether a service consumed by a user at some MC should be migrated to a different, more optimal MC. A user starts using a service in the cell of ring 0 (cell 0, 1 in Figure 1). Then in each time slot, the user chooses a cell of the next enclosing ring with equal probability, until eventually the user reaches the last ring. Note that from ring 0, the user moves to an adjacent cell in ring 1 with probability $\frac{1}{6}$, while from ring $i \geq 1$, the user moves to an adjacent cell in ring $i + 1$ with probability $\frac{1}{3}$ (if there are three choices of successor) or $\frac{1}{2}$ (if there are two choices).

Figure 2 depicts the mobile network modeled as an MDP. For simplicity, rather than model each cell as a state, the states of the model represent groups of cells that induce similar behaviors. The state space is $S = \{s_\iota, s_1, s_2^3, s_2^2, s_3^3, s_3^2, s_4^1, s_4^2, mig\}$, with s_ι representing the initial state in which a user begins in ring 0 using a service on some MC. For each state s_i or s_i^j other than the initial state, the user is located in ring i, while the service is still hosted on the same MC. In those other states, the FMC controller nondeterministically chooses between two actions—either to keep running the service on the same MC, denoted as action a_1, or to migrate the service to a new, optimal MC, denoted as action a_2. These actions are depicted by solid and dashed lines in Figure 2, respectively, and the execution of the model terminates upon any migration of the service.

As shown in Figure 2, all cells of ring 1 can be aggregated into a single state s_1 since the cells in ring 1 are isomorphic with respect to future behaviors and are reached with equal probability from the cell of ring 0. However, within ring 1, the user may move either directly towards a "corner" cell of the network (illustrated as the grey cells in Figure 1 and the colored state s_4^2 in Figure 2), or towards some cell between the corners of the network. The resulting state of the former kind of move is denoted s_2^3 (since the move is to a cell in ring 2 that offers 3 choices of new cell in ring 3), while the resulting state of the latter kind of move is denoted s_2^2 (since the move is to a cell in ring 2 that offers 2 choices of new cell in ring 3). Similar behavior is possible in the moves from rings 2 to 3 and 3 to 4, with corresponding states in the model.

Among the various properties of this network, suppose we are interested in determining *the maximum probability that a mobile user is served by the same MC until reaching one of the corners of the network*. In a real-world cloud environment, the movement of the user through the cloud can be affected by external factors (e.g., a power outage), environmental changes (e.g., an earthquake that could cause servers to go down), and so on. In our model, we assume that such uncertainty in network conditions affects the probabilistic choice of whether or not to move toward the corner of the network. Therefore, it is important to determine *the worst-case consequence of the effect of this uncertainty* on the verification of the above reachability property. The main contribution of this paper is an approach for efficiently computing this worst-case effect.

In our model, we capture this uncertainty in the variables x_1 and x_2, which embody perturbations to the the probabilistic choice between moves to states s_i^3 and s_i^2, respectively. We require only that $x_1 + x_2 = 0$, in order to ensure that the probabilities over the two choices sum to 1. Our approach uses these variables plus an analysis of the structure of the MDP to compute a *condition number*, which as described in Section 4 provides an *estimate* of the worst-case effect of the perturbation.

Previous studies have proposed different perspectives to deal with uncertainty in probabilistic verification. Sen *et al.* [29] introduce intervals for characterizing the uncertainty in transition probabilities and compute exact bounds for the verification of PCTL properties. However, these exact bounds are point-wise, which implies that whenever the bounds of the intervals are modified, the verification bounds must be recomputed, making reuse of the intervals difficult.

An alternative approach is to compute a mathematical function that defines the relationship between the uncertain probabilities and the verification results [8, 10, 15]. In this context, asymptotic perturbation analysis of DTMCs provides efficient approximations rather than exact bounds when the transition probabilities are subject to small perturbations [31, 32].

To leverage this asymptotic approach for MDP verification and compute asymptotic bounds on the effect of perturbation on verification results, we must somehow analyze the full set of DTMCs induced by all *memoryless adversaries* of the MDP (which resolve the nondeterministic choice in a state by selecting the same action each time the state is visited). However, an exhaustive analysis of an MDP has computational complexity proportional to the number of adversaries it admits, which in general is exponential in the size of the MDP. In this paper we present an efficient algorithm that mitigates the intractability of an exhaustive analysis.

3 BACKGROUND

This section presents the necessary background for our approach. We first revisit the formal definition of MDPs and their parametric extension, which underlies our perturbation analysis. We then introduce reachability verification of properties on MDPs as developed in previous studies [1, 5, 9, 12].

3.1 Markov Decision Processes

For a countable set S, a discrete probability distribution on S is a function $\mu : S \to [0, 1]$ such that $\sum_{s \in S} \mu(s) = 1$. The set of all probability distribution over S is denoted $Dist(S)$.

Definition 3.1. Markov Decision Process (MDP). An MDP is a tuple $\mathcal{M} = (S, Act, \mathbf{Q}, s_\iota, AP, L)$ where S is a finite set of states, Act is a set of actions, $\mathbf{Q} : S \times Act \times Dist(S)$ is a probabilistic transition function such that $\forall s \in S$, $\forall a \in Act$, $\sum_{s' \in S} \mathbf{Q}(s, a, s') \in \{0, 1\}$, $s_\iota \in S$ is an initial state, AP is a set of atomic propositions, and $L : S \to 2^{AP}$ a labeling function.

Let $Act(s)$ denote the set of enabled actions in state s, where $Act(s) := \{a \in Act \mid \mathbf{Q}(s, a, \bar{s}) > 0 \text{ for some } \bar{s} \in S\}$. For any state $s \in S$, it is required that $Act(s) \neq \emptyset$ and $\sum_{\bar{s} \in S} \mathbf{Q}(s, a, \bar{s}) = 1$ for any action $a \in Act(s)$ [2, 12].

For each state $s \in S$, there are two steps for determining its successor state. First, a nondeterministic choice is made among the actions $Act(s)$. Second, assuming action $a \in Act(s)$ has been selected, a successor state $\bar{s}$ is chosen randomly based on the probability distribution given by a, such that the probability of moving from s to $\bar{s}$ is given by $\mathbf{Q}(s, a, \bar{s})$.

The set of *successors states* of $s \in S$ where $a \in Act(s)$ is enabled is $succ(s, a) = \{\bar{s} \in S \mid \mathbf{Q}(s, a, \bar{s}) > 0\}$. Given a state set $K \subseteq S$, we denote the set of output states of K as $Out(K)$, where $Out(K) = \{\bar{s} \in S \backslash K \mid \mathbf{Q}(s, a, \bar{s}) > 0 \text{ for some } s \in S \text{ and } a \in Act(s)\}$.

Paths. An *infinite path* in an MDP $\mathcal{M}$ is a non-empty sequence of the form $\pi = s_0 \xrightarrow{a_0} s_1 \xrightarrow{a_1} s_2 \ldots$ where $s_i \in S$, $a_i \in Act(s_i)$ and $\mathbf{Q}(s_i, a_i, s_{i+1}) > 0$ for all $i \geq 0$. We denote $\pi(i)$ as the $(i + 1)$th state in the path π. A *finite path* $\rho = s_0 \xrightarrow{a_0} s_1 \xrightarrow{a_1} \ldots \xrightarrow{a_{n-1}} s_n$, $|\rho| = n$ denoting its length and $last(\rho) = s_n$ its last state. The *probability measure* of a finite path ρ, denoted $p(\rho)$, is defined as the product $\prod_{i=0}^{n-1} \mathbf{Q}(s_i, a_i, s_{i+1})$. We denote $Path_{\mathcal{M},s}$ and $Path_{\mathcal{M},s}^{fin}$ as the set of all infinite and finite paths starting in state s of $\mathcal{M}$, respectively.

A path through an MDP represents an *execution* (i.e. one possible behavior) of the model.

Executions. An execution of an MDP $\mathcal{M}$ is an alternating sequence of states and actions ending with a state. Formally, an execution $v = s_0 a_0 s_1 a_1 \ldots a_{n-1} s_n$ such that $s_i \xrightarrow{a_i} s_{i+1}$ for all $0 \leq i < n$. We denote $Act(v) = a_0 \ldots a_{n-1}$ as the set of actions in the execution v and $p(v)$ as the probability measure of the execution v.

To trace a path in an MDP $\mathcal{M}$, we must resolve the nondeterministic choices that are presented in $\mathcal{M}$. This resolution is performed by an *adversary*, alternatively called a *scheduler* or a *policy* in the literature.

Definition 3.2. Memoryless Adversary. An adversary of an MDP $\mathcal{M} = (S, Act, \mathbf{Q}, s_\iota, AP, L)$ is a function $\mathcal{A} : Path^{fin}_{\mathcal{M}} \rightarrow Dist(Act)$ such that $\mathcal{A}(\rho)(a) > 0$ only if $a \in Act(last(\rho))$. An adversary $\mathcal{A}$ is memoryless if $\mathcal{A}(\rho)$ depends only on $last(\rho)$, that is, for any $\rho, \rho' \in Path^{fin}_{\mathcal{M}}$ such that $last(\rho) = last(\rho')$, we have $\mathcal{A}(\rho) = \mathcal{A}(\rho')$.

A memoryless adversary $\mathcal{A}$ always selects the same action in a given state independent of which states were visited previously. Therefore, $\mathcal{A}$ restricts the set of paths, denoted as $Path^{\mathcal{A}}_{\mathcal{M},s}$, where $Path^{\mathcal{A}}_{\mathcal{M},s} \subseteq Path_{\mathcal{M},s}$ and induces a probability space $Prob^{\mathcal{A}}_{\mathcal{M},s}$ over the paths $Path^{\mathcal{A}}_{\mathcal{M},s}$ [16]. We denote $Adv_{\mathcal{M}}$ as the set of all memoryless adversaries in $\mathcal{M}$.

Definition 3.3. Sub-MDP. A sub-MDP of an MDP $\mathcal{M}$ is a pair of state and action sets (C, D) where (i) $C \subseteq S$ is non-empty and the map $D : C \rightarrow 2^{Act}$ is a function such that $D(s) \subset Act(s)$ is non-empty for all states $s \in C$, and (ii) $s \in C$ and $a \in D(s)$ implies $succ(s, a) \subseteq C$.

3.2 Parametric Markov Decision Processes

Previous studies [15, 22] have used parametric Markov decision processes (PMDPs) for expressing non-fixed transition probabilities in MDPs. We use PMDPs for capturing the presence of uncertainty in the probabilistic transitions of MDPs.

Definition 3.4. Parametric Markov Decision Process (PMDP). A PMDP based on MDP $\mathcal{M} = (S, Act, \mathbf{Q}, s_\iota, AP, L)$ is a tuple $\mathcal{M}[\mathbf{x}] = (S, Act, \mathbf{Q}[\mathbf{x}], s_\iota, AP, L)$ where S, Act, s_ι, AP and L are defined as Definition 3.1, $\mathbf{x} = (x_1, \cdots, x_m)$ is a vector of pair-wise distinct symbolic variables where m represents the total number of symbolic variables, and $\mathbf{Q}[\mathbf{x}]$ is a parametric transition probability function based on $\mathbf{Q}$.

The main difference between an MDP $\mathcal{M}$ and its parametric variant $\mathcal{M}[\mathbf{x}]$ lies in the extension of parameters and the definition of the probabilistic transition function $\mathbf{Q}[\mathbf{x}]$. Informally, $\mathbf{Q}[\mathbf{x}]$ is obtained by associating variables from $\mathbf{x}$ with some *specific* entries of $\mathbf{Q}$. In particular, the definition of $\mathbf{Q}[\mathbf{x}]$ for our approach will be explained in Section 4. For the sake of our approach, similar to the non-parametric case, we denote $Adv_{\mathcal{M}[\mathbf{x}]}$ as the set of optimal adversaries of a PMDP $\mathcal{M}[\mathbf{x}]$.

3.3 Reachability Verification in MDPs

A fundamental verification problem for quantitative analysis of MDPs is to compute the probability of reaching a target set $T \subseteq S$ [2, 12]. This property of probabilistic reachability is denoted $\Diamond T$.

Definition 3.5. Reachability Probabilities. Given an MDP $\mathcal{M}$, the minimum and maximum probability of reaching a set of target states $T \subseteq S$ from s, over all possible adversaries is defined as: $p^{min}_{\mathcal{M},s}(T) = \inf_{\mathcal{A} \in Adv_{\mathcal{M}}} \{p^{\mathcal{A}}_{\mathcal{M},s}(T)\}$, and $p^{max}_{\mathcal{M},s}(T) = \sup_{\mathcal{A} \in Adv_{\mathcal{M}}} \{p^{\mathcal{A}}_{\mathcal{M},s}(T)\}$ where $p^{\mathcal{A}}_{\mathcal{M},s}(T) = Prob^{\mathcal{A}}_{\mathcal{M},s}(\{\pi \in Path^{\mathcal{A}}_{\mathcal{M},s} \mid \exists i \cdot \pi(i) \in T\})$.

Definition 3.6. The minimum and maximum probability of reaching a set of target states $T \subseteq S$ from state s under its enabled action $a \in Act(s)$ can be defined, in similar way to Definition 3.5, as $p^{min}_{s \xrightarrow{a}}(T) = \inf_{\mathcal{A} \in Adv_{\mathcal{M}}} \{Prob^{\mathcal{A}}_{\mathcal{M},s}(Path^{\mathcal{A}}_{s \xrightarrow{a}}(T))\}$, and $p^{max}_{s \xrightarrow{a}}(T) = \sup_{\mathcal{A} \in Adv_{\mathcal{M}}} \{Prob^{\mathcal{A}}_{\mathcal{M},s}(Path^{\mathcal{A}}_{s \xrightarrow{a}}(T))\}$ where $Path^{\mathcal{A}}_{s \xrightarrow{a}}(T) = \{\pi \in Path^{\mathcal{A}}_{\mathcal{M},s} \mid \pi(0) = s, \mathcal{A}(s) = a, \exists i \cdot \pi(i) \in T\}$.

Adversaries that achieve the minimum and maximum probabilities defined in Definition 3.5 are called *optimal* adversaries, denoted as Adv^{min} and Adv^{max} respectively. Similarly, $Adv^{min}_{s \xrightarrow{a}}$ and $Adv^{max}_{s \xrightarrow{a}}$ denote the subsets of optimal adversaries that achieve the probabilities in Definition 3.6.

Definition 3.7. Sets of Optimal Executions. Given the sets of optimal adversaries $Adv^{min}_{s \xrightarrow{a}}$ and $Adv^{max}_{s \xrightarrow{a}}$, we define the sets of optimal executions from a state s and its enabled action $a \in Act(s)$ as follows:

$$\Upsilon^{min}_{s \xrightarrow{a}} = \{v \mid \forall A' \in Adv^{min}_{s \xrightarrow{a}}, v = s a s_1 A'(s_1) \ldots s_n\} \qquad (1)$$

$$\Upsilon^{max}_{s \xrightarrow{a}} = \{v \mid \forall A' \in Adv^{max}_{s \xrightarrow{a}}, v = s a s_1 A'(s_1) \ldots s_n\} \qquad (2)$$

where adversary $A' : S^+ \rightarrow Act$ such that $A(s s_1 s_2 \ldots s_n), \forall s s_1 s_2 \ldots s_n \in S^+$.

The computation of reachability probabilities is performed in two steps. The first step analyzes *qualitatively* the underlying graph of an MDP to identify sets of states for which the minimum or maximum reachability probability is either 0 or 1. Second, these sets are used to compute the *quantitative* minimum and maximum reachability probabilities using some standard technique such as value iteration, linear programming or policy iteration [12, 21]. In practice, probabilistic model checkers such as PRISM [20] use value iteration for their quantitative analysis. Value iteration is an iterative numerical method that approximates the values of a reachability probability up to some accuracy [2, 12]. In this context, Kwiatkowska *et al.* [21] developed an improved and efficient version of the value iteration technique based on a decomposition of the model into its strongly connected components (SCCs). This improved version optimizes the verification speed for MDPs.

4 PERTURBATION APPROACH

In this section, we present our perturbation approach for estimating the effect of uncertainty in the reachability verification of MDPs. After modeling a cloud computing system as an MDP, we first introduce uncertainty (small perturbations) to the MDP. Then, we characterize uncertainty via symbolic variables, and we capture the verification effect of uncertainty in the form of condition numbers.

4.1 Modeling Uncertainty

Inspired by previous studies [11, 31, 32], we use a parametric variant of an MDP $\mathcal{M}$ for modeling uncertainty in the nondeterministic model. First, we define the parametric probabilistic transition function by identifying actions and probabilistic transitions that may be affected by uncertain events.

Definition 4.1. Parametric Probabilistic Transition Function. Given a vector $\mathbf{x} = (x_1, \ldots, x_m)$ and a non-parametric function $\mathbf{Q}$, $\mathbf{Q}[\mathbf{x}]$ is the parametric probabilistic transition function where $\forall s \in S$ and $a \in Act(s)$, $\sum_{\bar{s} \in S} \mathbf{Q}[\mathbf{x}](s,a,\bar{s}) = 1$ and each entry $\mathbf{Q}[\mathbf{x}](s,a,\bar{s})$ of $\mathbf{Q}[\mathbf{x}]$ is defined as follows:

- $\mathbf{Q}(s,a,\bar{s})$, if $\mathbf{Q}(s,a,\bar{s}) \in \{0,1\}$, and
- $\mathbf{Q}(s,a,\bar{s})$ or $\mathbf{Q}(s,a,\bar{s}) + x_i$ where $1 \le i \le m$, if $\mathbf{Q}(s,a,\bar{s}) \notin \{0,1\}$.

Through Definition 4.1, we can see that only transitions with probabilities between, but not including, 0 and 1 can be updated by the appended variables. This is because transitions with probabilities 0 or 1 are not probabilistic, and thus not subject to perturbations.

Based on $\mathbf{Q}[\mathbf{x}]$, we state the following definition and proposition:

Definition 4.2. Perturbed Entry. A perturbed entry $\mathbf{Q}[\mathbf{x}](s,a,\bar{s})$ is of the form $p(s,a,\bar{s}) + x_i$ where $p(s,a,\bar{s}) \notin \{0,1\}$ represents the non-symbolic part of the perturbed entry and $x_i \in \mathbf{x}$ captures the presence of the uncertainty.

Proposition 4.3. *Given a vector $\mathbf{x} = (x_1, \ldots, x_m)$, let $\mathscr{I}$ be a partition on $\{1, \cdots, m\}$. A sub-vector $I = (x_j, \ldots, x_k)$ is an independent perturbed sub-vector in the partition $\mathscr{I}$ such that $1 \le j < k \le m$. For all $I \in \mathscr{I}$, $\sum_{\bar{s} \in S} \mathbf{Q}[\mathbf{x}](s,a,\bar{s}) = \sum_{\bar{s} \in S} p(s,a,\bar{s}) + \sum_{x_i \in I} x_i$ for some $s \in S$ and $a \in Act(s)$.*

In words, Proposition 4.3 states that each element of $\mathbf{x}$ falls into an *independent* perturbed sub-vector I and all variables of a sub-vector I are used. Each sub-vector I generates a singleton set of a perturbed action $Act_I = \{a \in Act \mid \exists s, \bar{s} \in S, \forall x_i \in I, \mathbf{Q}[\mathbf{x}](s,a,\bar{s}) = p + x_i\}$ and a set of perturbed states $S_I = \{s \in S \mid \exists \bar{s} \in S, \exists a \in Act(s), \forall x_i \in I, \mathbf{Q}[\mathbf{x}](s,a,\bar{s}) = p(s,a,\bar{s}) + x_i\}$. It is always assumed that the vector $\mathbf{x} = (x_1, \cdots, x_m)$ is within the set $\mathcal{U} \coloneqq \{\mathbf{x} \in \mathbb{R}^m \mid \forall I \in \mathscr{I}, \sum_{x_i \in I} x_i = 0, \mathbf{Q}[\mathbf{x}] \equiv \mathbf{Q}\}$. Note that we say $\mathbf{Q}[\mathbf{x}] \equiv \mathbf{Q}$ when they have exactly the same non-zero entries.

To illustrate our perturbation approach, consider a simple example of a PMDP depicted in Figure 3. For now, ignore the dashed rectangles. As the figure shows, the state space S is $S = \{s_0, s_1, s_2, s_3, s_4, s_5, s_6, s_7, s_8\}$ where s_0 is the initial state and the target set $T = \{s_7\}$. The action set Act is given by a_1, a_2, a_3, a_4 and a_5, which are depicted as solid, dashed, dotted, loosely dashed and loosely dotted lines respectively. The perturbation in the model is depicted by the vector $\mathbf{x} = (x_1, x_2, x_3, x_4, x_5)$ where each variable of $\mathbf{x}$ falls into two independent perturbed sub-vectors: (1) $I_1 = (x_1, x_2)$ associated with $Act_{I_1} = \{a_2\}$ and $S_{I_1} = \{s_0, s_1\}$, and (2) $I_2 = (x_3, x_4, x_5)$ associated with $Act_{I_2} = \{a_3\}$ and $S_{I_2} = \{s_4\}$.

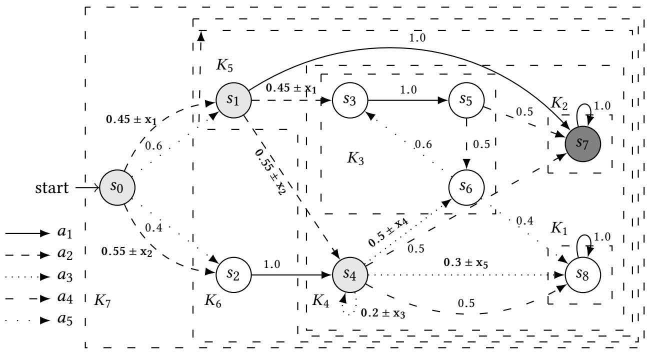

Figure 3: An Illustrative Example

4.2 Estimating the Worst-Case Effect

4.2.1 Perturbation Characterization of PMDPs. We first characterize the perturbation of PMDP $\mathcal{M}[\mathbf{x}]$ and the reachability property $\Diamond T$ to be verified. Recall that $p^{\mathcal{A}}_{\mathcal{M},s}(T)$ is the probability of reaching a target set T from state s under the adversary $\mathcal{A}$ in MDP $\mathcal{M}$. In this context, we formally define the *exact* difference between satisfying a reachability property in a *perturbed* model $\mathcal{M}[\mathbf{x}]$ and its an associated *unperturbed* model $\mathcal{M}$ as follows:

Definition 4.4. Exact Perturbation Characterization Function. Given an MDP $\mathcal{M}$ and its parametric variant PMDP $\mathcal{M}[\mathbf{x}]$ with the target set T, the exact perturbation characterization function for $\mathcal{M}[\mathbf{x}]$ against T is $\lambda : \mathcal{U} \to [0,1]$ such that $\lambda(\mathbf{x}) = \sup_{\mathcal{A} \in Adv_{\mathcal{M}[\mathbf{x}]}} \{p^{\mathcal{A}}_{\mathcal{M}[\mathbf{x}],s_\iota}(T)\} - \sup_{\mathcal{A} \in Adv_{\mathcal{M}}} \{p^{\mathcal{A}}_{\mathcal{M},s_\iota}(T)\}$, where $\mathbf{x} \in \mathcal{U}$.

Definition 4.4 reveals that for each set of values of the vector $\mathbf{x}$, it is possible to compute the *exact* difference $\lambda(\mathbf{x})$ between $\mathcal{M}[\mathbf{x}]$ and $\mathcal{M}$. However, our purpose is to capture the worst-case effect of $\mathbf{x}$ in $\mathcal{M}$, for which we need to evaluate $\lambda(\mathbf{x})$ for all combination of $\mathbf{x}$. Then, we select the worst-case consequence based on them. For this reason, we define the *perturbation characterization function* γ against the perturbed parameters $\mathbf{x}$ that *estimates* the *maximum difference* between satisfying a reachability property in $\mathcal{M}[\mathbf{x}]$ and its an associated $\mathcal{M}$ as follows:

Definition 4.5. Estimated Perturbation Characterization Function. Given an MDP $\mathcal{M}$ and its parametric variant PMDP $\mathcal{M}[\mathbf{x}]$ with the target set T, the estimated perturbation characterization function for $\mathcal{M}[\mathbf{x}]$ against T is $\gamma : \mathcal{U} \to [0,1]$ such that $\gamma(\mathbf{x}) = \sup_{\mathcal{A} \in Adv_{\mathcal{M}[\mathbf{x}]}} \{p^{\mathcal{A}}_{\mathcal{M}[\mathbf{x}],s_\iota}(T) - p^{\mathcal{A}}_{\mathcal{M},s_\iota}(T)\}$, where $\mathbf{x} \in \mathcal{U}$.

4.2.2 Maximum Condition Number. Inspired by previous studies [31, 32] and based on perturbation analysis, we use the concept of condition numbers, which capture the sensitivity of verification results to perturbed parameters. The maximum condition number (MaxCN) measures how much the characterization function $\gamma(\mathbf{x})$ can change due to small perturbations. Let $\delta > 0$ be a perturbation distance. We assume δ is a small positive number. We define the MaxCN as follows:

Definition 4.6. Maximum Condition Number. Let $\delta > 0$ be a perturbation distance. The maximum condition number, denoted as κ, for the estimated characterization function γ against $\mathbf{x}$ is:

$$\kappa = \lim_{\delta \to 0} \sup \left\{ \frac{\gamma(\mathbf{x})}{\delta} \mid \mathbf{x} \in \mathcal{U}, \|\mathbf{x}\| \le \delta, \delta > 0 \right\} \quad (3)$$

Let $\rho_1(\mathbf{x}) = \mathbf{h} \cdot \mathbf{x}$ be the linear approximation of γ for some vector $\mathbf{h}$ such that $|\mathbf{h}| = |\mathbf{x}|$. Note that the existence of the limit in Eq. (3) is guaranteed by the fact that γ is differentiable at $\mathbf{0}$ [6].

We formulate that the coefficients of vector $\mathbf{h}$ is the concatenation of all sub-vectors $\mathbf{h}_I = (h_j, \dots, h_k)$ for all $I \in \mathscr{I}$ and $1 \le j < k \le m$. We assume that the vector $\mathbf{h}$ preserves the order after the concatenation. In this context, we alternatively formulate the MaxCN of PMDP $\mathcal{M}[\mathbf{x}]$ as follows:

Definition 4.7. Let $h^+ = \max(\mathbf{h}_I)$ and $h^- = \min(\mathbf{h}_I)$ be the largest and smallest coefficients of $\mathbf{h}_I$. Let $\triangle h_I = h^+ - h^-$ be the maximum difference between all coefficients of $\mathbf{h}_I$. Then, the maximum condition number $\kappa = \frac{1}{2} . \max_{I \in \mathscr{I}} \triangle h_I$.

The key point is to compute the largest h^+ and smallest h^- coefficients of each sub-vector $I \in \mathscr{I}$. However, it may no be feasible to compute these coefficients by analyzing all adversaries in the PMDP model. To overcome this problem, we introduce a heuristic method that analyzes the PMDP efficiently and computes the MaxCN of the model.

5 MAXCN ALGORITHM

In this section, we present our algorithmic approach that computes the maximum condition number of a PMDP, named as *MaxCNAlg*. In the following, let $\mathcal{M}[\mathbf{x}] = (S, Act, \mathbf{Q}[\mathbf{x}], s_\iota, AP, L)$ be a PMDP with target set $T \subseteq S$ and the partition $\mathscr{I}$ over the vector $\mathbf{x}$.

Suppose we aim to compute the maximum condition number κ of the PMDP depicted in Figure 3, based on Def. 4.7, we require analyzing the perturbed sub-vectors $I_1 = (x_1, x_2)$ and $I_2 = (x_3, x_4, x_5)$, and its corresponding coefficient vectors $\mathbf{h}_{\mathbf{I}_1} = \{h_1, h_2\}$ and $\mathbf{h}_{\mathbf{I}_2} = \{h_3, h_4, h_5\}$. Note that the coefficient h_1 is related to variable x_1, h_2 to variable x_2 and so on.

Let us first analyze the vector $\mathbf{h}_{\mathbf{I}_1} = \{h_1, h_2\}$. To compute the coefficients of $\mathbf{h}_{\mathbf{I}_1}$, we first need to detect the transitions associated with variables x_1 and x_2. In our example, as can be seen from Figure 3, the transitions $s_0 \xrightarrow{a_2} s_1$ and $s_1 \xrightarrow{a_2} s_3$ are associated with x_1 and $s_0 \xrightarrow{a_2} s_2$ and $s_1 \xrightarrow{a_2} s_4$ with x_2. To determine the value of h_1, we depend on the reachability probabilities from the successors states s_1 and s_3 to the target set T. Similarly, h_2 depends on the reachability probabilities from states s_2 and s_4 to T. The challenge here is to determine which probabilities (similarly which adversaries) should be assigned to the coefficients h_1 and h_2 such that they maximize $\triangle h_{I_1}$. Our algorithm deals with this challenge.

5.1 Sub-PMDP Generator

The basic concept of our approach is to generate a sub-PMDP $\mathcal{M}^* = (S^*, Act^*)$ that contains the relevant information for maximizing the difference between the coefficients of a sub-vector $I \in \mathscr{I}$, denoted as $\triangle h_I$.

Inspired by a former method for computing reachability probabilities in the non-parametric case [7, 13, 21], we analyze the underlying graph of the PMDP by detecting the set of *strongly connected components* (SCCs) as a heuristic for selecting the set of states to be analyzed. The SCC-based decomposition of the graph is generated by Tarjan's algorithm [27]. Then, we determine the *reversed topological order* ζ among the set of SCCs such that $\Pi = \{K_1, \dots, K_N\}$ where N denotes the number of SCCs in the PMDP. K_j will appear before K_i in ζ if K_i depends on K_j. We say that K_i *depends on* K_j if $Succ(K_i) \cap K_j \neq \emptyset$ where $Succ(K_i)$ is the set of states that are immediate successors of states in K_i.

As an illustration, consider again the PMDP shown in Figure 3, the dashes rectangles indicate the SCC-based decomposition in ζ: $K_1 = \{s_8\}$, $K_2 = \{s_7\}$, $K_3 = \{s_3, s_5, s_6\}$, $K_4 = \{s_4\}$, $K_5 = \{s_1\}$, $K_6 = \{s_2\}$ and $K_7 = \{s_0\}$.

In the following let $I = (x_j, \dots, x_k)$ be the sub-vector to be analyzed and let $s(x_i)$ be the shorthand for representing the successor state $\bar{s}$ of a perturbed entry $\mathbf{Q}[\mathbf{x}] = p(s, a, \bar{s}) + x_i$. Given the sub-vector I, we generate a Sub-PMDP of $\mathcal{M}[\mathbf{x}]$ for each successor state $s(x_i) \in S$. This procedure is named as generatorSubPMDP and depicted in Algorithm 1.

```
Algorithm 1: generatorSubPMDP($s(x_i)$)
   Input: A state $s(x_i) \in S$
1  $\Pi^* = \{K_1, \dots, K_x\}$ such that $s(x_i) \in K_x$ and $\Pi^* \subseteq \Pi$;
2  $\mathcal{M}^* \leftarrow$ Sub-PMDP $(S^*, Act^*)$ where $S^* = Act^* = \emptyset$;
3  for SCC $K \in \Pi^*$ do
4      $S^* \leftarrow K$;
5      $Act(K) = \bigcup_{s \in K} Act(s)$            // Set of actions of $K$ ;
6      $Act^* \leftarrow Act(K)$;
7      if $K$ is no MEC then
8          $Act^{out}(K) = \{a \in Act(K) \mid \exists s \in K \cdot succ(s, a) \subseteq$
             $Out(K)\}$            // Set of outgoing actions of $K$;
9          $Act^+ :=$ prune_actions($\mathcal{M}^*, Act^{out}(K)$);
10         $Act^* \leftarrow Act^+$            // Update $Act^*$ based on $Act^+$ ;
11 return Sub-PMDP $\mathcal{M}^*$
```

As can be seen in Algorithm 1, generatorSubPMDP($s(x_i)$) first computes the subset of SCCs to be analyzed, denoted as $\Pi^* \subseteq \Pi$, where the successor state $s(x_i)$ belongs to the last SCC to be analyzed (line 1). This step is important due to the facts that (a) an SCC can be analyzed independently, as long as all of its successors set have been already analyzed, and (b) the method only analyzes the relevant SCCs in the best case scenario.

Second, generatorSubPMDP iterates through all SCCs in Π^*. For each SCC, $K \in \Pi^*$, the algorithm assigns K and the set of actions of K to the set of states S^* and the set of actions Act^*, respectively (lines 4-6). Third, generatorSubPMDP evaluates whether K is not a *maximum end component* (MEC). In the affirmative case, generatorSubPMDP computes the set of outgoing actions of K (line 8) and invokes function prune_actions($\mathcal{M}^*, Act^{out}(K)$) (line 9). This function takes as input the current sub-PMDP $\mathcal{M}^*$ and the set of outgoing actions $Act^{out}(K)$, and returns the set of relevant actions Act^+. The update of Act^* is subject to Act^+ (line 10). Finally, generatorSubPMDP returns a sub-PMDP $\mathcal{M}^*$.

5.1.1 Pruning actions. Let us recall that each SCC can be analyzed independently and our target is to find coefficients that maximize $\triangle h_I$. With this in mind, we evaluate the set of outgoing actions of each SCC, and we discard actions that do not maximize $\triangle h_I$. This procedure is named as prune_actions and it takes as input a sub-PMDP $\mathcal{M}^*$ and a set of outgoing actions of an SCC K ($Act^{out}(K)$). The pseudo code of prune_actions is described in Algorithm 2.

Algorithm 2: prune_actions($\mathcal{M}^*, Act^{out}(K)$)

```
Input: A Sub-PMDP $\mathcal{M}^*$ and a set of outgoing actions
       $Act^{out}(K)$ of SCC $K$
1  $Act^+ \leftarrow \emptyset$;
2  for action $a^+ \in Act^{out}(K)$ do
3      $\bar{S} \leftarrow succ(s, a^+)$ for some $s \in K$ ;
4      $\Upsilon(\bar{S}) = \left\{\Upsilon^{min}_{\bar{s}\xrightarrow{a^*}} \cup \Upsilon^{max}_{\bar{s}\xrightarrow{a^*}} \mid \forall \bar{s} \in \bar{S}, \forall a^* \in Act^*(\bar{s})\right\}$   // Set of
       optimal executions in $\mathcal{M}^*$ ;
5      $T^* \leftarrow (\Upsilon(\bar{S}), \bar{S})$        // Build an execution tree ;
6      $\Omega(T^*) = \{\omega_1, \ldots, \omega_m\}$        // Set of branches $T^*$;
7      $\Omega^* = \text{argmax}_{\omega \in \Omega(T)} \triangle p(\omega)$ where
       $\triangle p(\omega) = p^{max}(\omega) - p^{min}(\omega)$;
8      $Act' = \{Act(v) \mid \forall \omega \in \Omega^*, \forall (v, p) \in \omega\}$ // Set of relevant actions
       based on $\Omega^*$ ;
9      $Act^+ \leftarrow Act'$
10 return $Act^+$
```

As Algorithm 2 shows, for each outgoing action $a^+ \in Act^{out}(K)$, prune_actions first computes the set of all successor states of state $s \in K$ via action a^+, denoted as $\bar{S}$ (line 3). Second, given the sub-PMDP $\mathcal{M}^*$ and the set $\bar{S}$, prune_actions computes the set of optimal executions $\Upsilon(\bar{S})$, as defined in Def. 3.7 (line 4). To reason about $\Upsilon(\bar{S})$, we build an execution tree T^* defined as follows:

Definition 5.1. Executions Tree. Given a set of optimal executions $\Upsilon(\bar{S})$ and the set of successors states $\bar{S}$, the execution tree T^* is a rooted tree where:

- The root of T^* is a distinguished node $v^* = \emptyset$.
- The height of T^* is equal to $|\bar{S}| + 1$.
- Every internal node u is a pair (v, p), where v is an execution in $\Upsilon(\bar{S})$ and p is the probability measure of v.
- A node $a = (v_a, p_a)$ is the parent of node $b = (v_b, p_b)$ if only if $v_b \subseteq v_a$ or $v_a \subset v_b$.
- A branch ω of T^* is a path from v^* to a leaf node where $|\omega| = |\bar{S}|$.
- The minimum and maximum probability of a branch ω is defined as $p^{min}(\omega) = \min\{p \mid \forall (v, p) \in \omega\}$ and $p^{max}(\omega) = \max\{p \mid \forall (v, p) \in \omega\}$, respectively.

After building the execution tree T^*, prune_actions computes the set of branches of T^*, denoted as $\Omega(T^*)$ (line 6). As a heuristic for filtering the actions, the method selects the subset of branches $\Omega^* \subseteq \Omega(T^*)$ at which $\triangle p(\omega)$ is maximized (line 7). Finally, it selects the set of existing actions in the sub-set Ω^* (line 8).

Figure 4 illustrates the execution tree with respect to state $s_1 \in K_5$ and action $a_2 \in Act^{out}(K_5)$. Given that $succ(s_1, a_2) = \{s_3, s_4\}$, the first level of the execution tree corresponds to state s_3 and the second level to state s_4. The set of branches $\Omega(T^*) = \{\omega_1, \omega_2\}$, and since $\triangle p(\omega_1) > \triangle p(\omega_2)$ the subset $\Omega^* = \{\omega_1\}$. As a result, the action a_4 is pruned.

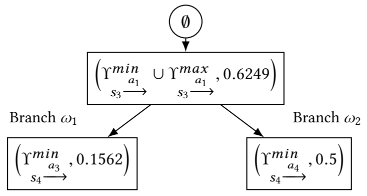
$\emptyset$
$\left(\Upsilon^{min}_{s_3\xrightarrow{a_1}} \cup \Upsilon^{max}_{s_3\xrightarrow{a_1}}, 0.6249\right)$
Branch ω_1
Branch ω_2
$\left(\Upsilon^{min}_{s_4\xrightarrow{a_3}}, 0.1562\right)$
$\left(\Upsilon^{min}_{s_4\xrightarrow{a_4}}, 0.5\right)$

Figure 4: Example of an Execution Tree

5.2 Computing the Maximum Difference

The procedure MaxDiff, depicted in Algorithm 3, performs the computation of the maximum difference $\triangle h_I$ of a coefficient sub-vector h_I.

Algorithm 3: MaxDiff(I)

```
Input: A sub-vector $I$
1  $\bar{S_I} = \{\bar{s} \in S \mid \forall x_i \in I, \mathbf{Q}[\mathbf{x}](s, a, \bar{s}) = p(s, a, \bar{s}) + x_i\}$   // Perturbed
   successors states;
2  $\Upsilon_I \leftarrow \emptyset$;
3  for $s(x_i) \in \bar{S_I}$ do
4      $(S^*, Act^*) \leftarrow$ generatorSubPMDP($s(x_i)$);
5      $\Upsilon_I(s(x_i)) \leftarrow \{\Upsilon^{min}_{s(x_i)\xrightarrow{a*}} \cup \Upsilon^{max}_{s(x_i)\xrightarrow{a*}} \mid \forall a^* \in Act^*(s(x_i))\}$;
6  $T \leftarrow (\Upsilon_I, I)$        // Build a coefficient tree ;
7  $\Omega(T) = \{\omega_1, \ldots, \omega_m\}$        // Set of branches $T$;
8  $\triangle h_I = \max\{h^+(\omega) - h^-(\omega)\}$;
9  return $\triangle h_I$
```

MaxDiff takes as input the perturbed sub-vector I and computes the set of perturbed successors states, denoted as $\bar{S_I}$, of I (line 1). Then, MaxDiff constructs a dictionary Υ_I that maps a successor state $s(x_i)$ to its set of executions (line 2). Next, MaxDiff iterates through all perturbed successors states in $\bar{S_I}$. For each successor state $s(x_i) \in \bar{S_I}$, the function invokes generatorSubPMDP($s(x_i)$) which returns a sub-PMDP (S^*, Act^*) (line 4). It is important to notice that the SCC-based procedure as introduced in the previous section contains relevant information for the computation of $\triangle h_I$. Based on (S^*, Act^*), MaxDiff selects the set of optimal executions with respect to $s(x_i)$ and stores into the dictionary Υ_I (line 5).

After the analysis of the set of successors states $\bar{S_I}$ and the computation of optimal executions that provide relevant information, MaxDiff builds a coefficient tree, which is defined as follows:

Definition 5.2. Coefficient Tree. Given a set of optimal executions Υ_I and the sub-vector $I = (x_a, \ldots, x_b)$, the coefficient tree T is a rooted tree where:

- The root of T is a distinguished node $v^* = \emptyset$.
- The height of T is equal to $|I| + 1$.
- Every internal node u is a pair $(\mathbf{v_i}, h_i)$ where:

  - $\mathbf{v_i}$ is a tuple list of the form $(s, a, \bar{s}, v)$ where s is a perturbed state, a is a perturbed action, $\bar{s}$ is a perturbed successor state and v is an execution.
  - h_i is the coefficient related to the variable x_i.
- A coefficient $h_i = \sum_{(s,a,\bar{s},v)}\{p^{max}_{\mathcal{M},s_\iota}(\{s\}) \cdot \Lambda\}$ where:

$$\Lambda = \begin{cases} \frac{p(v)}{1-\mathbf{Q}[\mathbf{x}](s,a,s)} & \text{if } \mathbf{Q}[\mathbf{x}](s,a,s) > 0 \\ p(v) & \text{if } \bar{s} \notin T \wedge \mathbf{Q}[\mathbf{x}](s,a,s) = 0 \\ 1 & \text{otherwise} \end{cases}$$

- A node $x = (\mathbf{v_i}, h_i)$ is the parent of node $y = (\mathbf{v_{i+1}}, h_{i+1})$, if only if $\mathbf{v_i}(v) \cap \mathbf{v_{i+1}}(v) \neq \emptyset$.
- A branch ω of T is a path from v^* to a leaf node where $|\omega| = |I|$.
- The largest and smallest coefficients of a branch ω are defined as $h^+ = \max_{(\mathbf{v_i},h_i)\in\omega}\{h_i\}$ and $h^- = \min_{(\mathbf{v_i},h_i)\in\omega}\{h_i\}$, respectively.

As last steps, MaxDiff evaluates the set of branches $\Omega(T)$. Next, it computes and returns the value of the maximum difference $\triangle h_I$ (lines 6-9).

Another key point of our algorithm is the building of trees. The execution and coefficient trees ensure to group a set of probabilities generated by the same set of executions. In this way, we can identify the largest and smallest coefficients of a similar set of adversaries.

5.3 Time Complexity

Theorem 5.3. *The time complexity of the MaxCNAlg algorithm is $O(y \cdot |S|^4 \cdot |Act|^2)$.*

Proof. Given a PMDP $\mathcal{M}[\mathbf{x}] = (S, Act, \mathbf{Q}[\mathbf{x}], s_\iota, AP, L)$ and the vector $\mathbf{x} = (x_1, \ldots, x_m)$, let $\mathscr{I}$ be a partition on $\{1, \ldots, m\}$ that contains $y < |\mathbf{x}|$ sub-vectors. *MaxCNAlg* calls y times the MaxDiff function, which takes as input a sub-vector $I \in \mathscr{I}$ and invokes generatorSubPMDP function at most $|S|$ times. To calculate generatorSubPMDP, we analyze at most $|S|$ SCCs. For each SCC, we invoke prune_actions which can be achieved in $O(|S|^2 \cdot |Act|^2)$ time. Thus, considering all these aspects, the complexity of *MaxCNAlg* is $O(y \cdot |S|^4 \cdot |Act|^2)$.

6 EXPERIMENTAL EVALUATION

In this section, we present two case studies of cloud computing which were inspired by previous studies [3, 18, 34, 35]. As we present each case study, we describe briefly the MDP model, the reachability property to be verified and the perturbed set to be analyzed. For evaluation purposes, we have developed a prototype implementation of our algorithms in Python and used it in conjunction with the probabilistic model checker PRISM [20] to extract basic information about the MDP model. The prototype computes the maximum condition number (MaxCN) for the MDP using the algorithm presented in the previous sections. All our experiments have been conducted on a machine with 3.06 GHz Intel Core 2 Duo processors and 8 GB RAM.

6.1 Service Migration Decision in Cloud Computing

This case study has been described in Section 2. In short, we analyze the service migration decision of an FMC controller that decides whether a service consumed by a user and hosted by a particular micro-cloud should be migrated to a different, optimal micro-cloud. The cloud computing services work within a typical 3GPP network topology of N rings of cells. We performed experiments for systems with network topology of $N = 5$ and $N = 8$ rings. We modeled each system as an MDP, where the state space S is composed of $2N - 1$ states. The MDP models are denoted as $\mathcal{M}^5$ and $\mathcal{M}^8$ respectively.

We wish to verify *the maximum probability a user is served by only one micro-cloud until*

P1 *a user reaches one of the corner cells of the topology*;

P2 *a user reaches one of the cell in the last ring between the corners*; and

P3 *a user reaches one of the corners of ring* $\lfloor \frac{N}{2} \rfloor$.

As mentioned in Section 2, due to the nature of the cloud environment and other external factors, the probabilistic transitions of the model can be affected by small perturbations. We capture the perturbations in variables x_1 and x_2, which are attached to the probabilistic transitions to states s_i^j shown in Figure 2. For the purpose of estimating the worst-case consequence of the effect of these perturbations, we compute the MaxCN for each model. These experimental results are presented in Table 1.

Table 1: Experimental Results

Model	MDP Data			MaxCN	Time(sec)	
	$\sharp S$	$\sharp T$	$\sharp Adv$	(κ)	*MaxCNAlg*	*ExhAlg*
$\mathcal{M}^5, P1$	09	17	32	0.1111	**0.15**	3.28
$\mathcal{M}^5, P2$				0.5	**0.21**	4.02
$\mathcal{M}^5, P3$				0.5	**0.10**	2.95
$\mathcal{M}^8, P1$	15	32	2048	0.0102	**141.06**	197.10
$\mathcal{M}^8, P2$				0.5	**202.42**	288.97
$\mathcal{M}^8, P3$				0.111	**139.75**	199.05

As can be seen from Table 1, for each model instance, the first four columns show the model and property being verified, plus the size of the model in terms of its number of states, transitions and (memoryless) adversaries. The remaining columns show the MaxCN computed for the model and the running time (in seconds) of our algorithm, denoted as *MaxCNAlg*. For the sake of evaluating the performance of our algorithm, the last column gives the running time of an exhaustive algorithm, denoted as *ExhAlg*, which exhaustively analyzes the total set of adversaries in the MDP. *ExhAlg* computes the condition number of each adversary and then returns the maximum of all computed condition numbers. As indicated by the **boldfaced** times, *MaxCNAlg* outperforms *ExhAlg* in all cases.

As Table 1 shows, our experiments have been conducted on a model with a significant number of adversaries. The number of adversaries is a critical determinant of performance, since it represents the set of paths and executions that the algorithms need to analyze.

According to the running time of the two algorithms, the time computed by *MaxCNAlg* is always considerably smaller than *ExhAlg*. Table 1 also reveals that the running times of the two algorithms are consistent between models, in that the largest and smallest running times of both algorithms correspond to the same

Table 2: Additional Experimental Results for $\mathcal{M}^5$

Model	δ ($\times 10^{-3}$)	Pr^{max}	$\triangle Pr$ ($\times 10^{-3}$)	α ($\times 10^{-3}$)
$\mathcal{M}^5$	0	0.96296	-	-
$\mathcal{M}^5_1$	1	0.96312	0.16	± 0.5
$\mathcal{M}^5_2$	2	0.96329	0.33	± 1.0
$\mathcal{M}^5_3$	3	0.96346	0.50	± 1.5

Table 3: Experimental Data of $\mathcal{M}^c$ model

Model	δ ($\times 10^{-3}$)	Pr^{max}	$\triangle Pr$ ($\times 10^{-3}$)	α ($\times 10^{-3}$)
$\mathcal{M}^c$	0	0.7839	-	-
$\mathcal{M}^c_1$	1	0.7833	-0.6	± 1.81
$\mathcal{M}^c_2$	2	0.7827	-1.2	± 3.62
$\mathcal{M}^c_3$	3	0.7821	-1.8	± 5.43

property. For example, the largest running times of *ExhAlg* and *MaxCNAlg* correspond to P2 for each model. In a similar way, there exists a correlation between P1 and P3. It is important to mention that the performance of our algorithm is subject to the perturbation input, the underlying graph of the MDP and the model connectivity.

To evaluate whether our approach provides an accurate estimate of the worst-case effect of uncertainty in the reachability verification of MDPs, we computed the perturbation estimate generated by the *MaxCNAlg* algorithm. For this purpose, we conducted experiments, depicted in Table 2, on the model $\mathcal{M}^5$ and the reachability property $P2$.

First, we constructed several non-parametric models with small perturbation distances, denoted as $\mathcal{M}^5_j$ ($j \geq 1$), which are simply instances of $\mathcal{M}^5$ with values supplied for the symbolic variables. Second, for each such perturbed model, we computed the actual maximum probability for reachability property $P2$, denoted as Pr^{max}_j. Third, we compared these actual maximum probabilities against the maximum probability for the unperturbed model (with the symbolic variables set to zero), computed as $\triangle Pr = Pr^{max}_j - Pr^{max}$. Lastly, we calculated the estimate $\alpha = \pm\kappa\delta$ as an estimate for $\triangle Pr$, which thus provides an estimate of the worst-case effect of uncertainty on $P2$ due to the perturbation.

As shown in Table 2, for each model the estimate α represents a safe prediction of the worst-case effect. For example, for the model $\mathcal{M}^5_1$, given that $\delta = 1 \times 10^{-3}$, the difference between the actual probabilities for the perturbed and unperturbed models is 0.16×10^{-3}, which lies within the interval $[0.96246, 0.96346]$ estimated by α.

The correlation between the probability verifications of the perturbed model Pr^{max}_j and the perturbation estimate α is interesting because it demonstrates that the increment of these values is subject to the perturbation distance. All these findings reveal that the prediction of the worst-case effect of uncertainty in the reachability verification of MDPs is captured by the maximum condition number computed by our approach.

Another key finding from our work is that the maximum condition number of $P2$ is generated by a non-optimal adversary (which does not produce minimum and maximum satisfaction probabilities). This finding demonstrates that the worst-case effect of uncertainty can be generated via adversaries other than those identified as optimal by conventional techniques. In general, this result offers crucial information for further investigation.

6.2 Confidentiality in Cloud Computing

Inspired by Baldi *et al.* [3], we study the problem of confidentiality provided by data dispersal algorithms against an intruder in the cloud. Data dispersal algorithms provide a method for storing information in distinct slices dispersed across multiple servers in different locations.

We model the behavior of a client that sends a sequence of n slices to m distinct servers each having a capacity of c slices, and an intruder that intercepts the traveling slices and reconstructs the message body. The sequence of n slices constitutes a message. A client can select any server as a storage server with the same probability. However, a client can send a slice to the selected server only if the server has not reached its full capacity. Otherwise, the client tries again, selecting another server.

The intruder, called the *slice attacker*, intercepts every slice independently from the previously intercepted ones. A slice attacker needs to intercept at least k slices to reconstruct the message. The probability that the attacker intercepts a slice successfully is p.

Due to the nature of cloud computing systems, the statistical estimate of p, which is associated with the performance of a slice attacker, could be affected by small perturbations. We investigate the effect of these perturbations on *the maximum probability that a slice attacker manages to reconstruct a message*.

We model the system as an MDP, denoted as $\mathcal{M}^c$, with the following parameters: $n = 3$, $m = 2$, $c = 3$, $k = 2$ and $p = 0.7$. The MDP model size is 152 states and 373 transitions. The number of adversaries is over 116×10^{10}. In a manner similar to the previous case study, we built three non-parametric models having a small perturbation distance from the unperturbed model. These perturbed models are denoted as $\mathcal{M}^c_1$, $\mathcal{M}^c_2$ and $\mathcal{M}^c_3$.

In order to evaluate our approach, we compute the maximum condition number of the unperturbed model using the *MaxCNAlg* algorithm. Due to the large number of adversaries, we were unable to complete any successful run of *ExhAlg* for a performance comparison.

Table 3 presents the perturbation estimates generated by our approach, in a manner similar to Table 2. As can be seen from Table 3, the maximum probability of the unperturbed model is 0.7839 and the probability of the perturbed models $\mathcal{M}^c_1$, $\mathcal{M}^c_2$ and $\mathcal{M}^c_3$ are 0.7833, 0.7827 and 0.7821, respectively. These findings show that the probability of a perturbed model is subject to the perturbation distance. These results are consistent with the experimental data of the previous case study.

We can observe in Table 3 that the estimates α provide an accurate estimate of the worst-case effect of the perturbations. For instance, for the model $\mathcal{M}^c_3$, α is ± 5.43, corresponding to the probability interval $[0.77847, 0.78933]$.

The experimental results for both case studies show a clear advantage of our algorithm over the exhaustive algorithm. *MaxCNAlg* successfully and efficiently computes the MaxCN of an MDP model without analyzing all adversaries. Likewise, the results demonstrate that our approach provides an accurate estimate of the worst-case effect of uncertainty on the reachability verification of MDPs.

7 RELATED WORK

In this section, we briefly discuss previous works related to the research described in this paper, focusing on the existent works on probabilistic model checking of cloud computing systems and studies that deal with the uncertainty in the quantitative verification of MDPs.

Recent studies have adopted probabilistic model checking with the purpose of analyzing the advantages and limitations of cloud computing systems. For example, Naskos *et al.* [23–25] developed an approach to enforcing elasticity (the most prominent advantage of cloud computing) and the provisioning of cloud resources through dynamic instantiation and online quantitative verification of MDPs. The authors demonstrated that quantitative verification at runtime also could be beneficial for the management of requirements of the cloud. On the other hand, researchers at Fujitsu [17] adopted probabilistic model checking to construct a performance model of concurrent live migrations in virtualized data centers. The authors claim their approach will help cloud administrators to orchestrate management operations in their cloud systems. Our experiments complement these previous studies.

In probabilistic model checking, there are previous studies on Markov models with uncertainties. Sen *et al.* [29] introduce Interval-valued Discrete-time Markov Chains (IDTMCs) for which the exact transition probabilities are given by a closed interval. The authors consider two semantic interpretations for the uncertainty in the transition probabilities of an IDTMC: One is Uncertain Markov Chains (UMCs), a family of DTMCs whose transition probabilities lie within the interval range given. The other is Interval Markov Decision Processes (IMDPs), where the uncertainty is resolved through nondeterminism. In addition, they introduce verification of PCTL properties on IDTMCs. In this context, Puggelli *et al.* [28] present P-completeness complexity bounds for an extension of IMDPs.

Despite the fact that the scope of our study is offline quantitative verification, other approaches, for example Calinescu *et al.* [4], ensure that requirements are continuously satisfied as the system evolves using a combination of continual verification and online updating of the verified models. Both techniques by their nature provide advantages and limitations.

Another study that deals with uncertainties in probabilistic model checking is given by Daws [8]. His study presents a language-theoretic approach to symbolic model checking of PCTL over DTMCs, introducing a parametric Markov chain formalism whose transition probabilities are viewed as letters in an alphabet of a finite automaton. Hahn *et al.* [14, 15] study the synthesis problem for reachability and PCTL properties in a parametric version of MDPs, improving the study of Daws. Similarly, Filieri *et al.* [11] develop a mathematical framework for runtime probabilistic model checking, presenting a parameterized version of the Gauss-Jordan elimination algorithm to address the uncertainty problem in DTMC models for reliability prediction.

Lastly, Su *et al.* [6, 30–32] introduce a mathematical framework for asymptotic analysis in the presence of small perturbations to model probabilities, which represent uncertainty for probabilistic verification of DTMCs and CTMCs. For DMTCs, the authors present a parametric variant of DTMCs for modeling the perturbation, and they define two asymptotic bounds, namely condition numbers and quadratic bounds. As discussed previously, it is impractical to apply this approach exhaustively to the analysis of all DTMCs induced by adversaries of an MDP, due to the exponential computational complexity. In this context, our work presents an efficient asymptotic approach to address the effect of uncertainty in the verification of MDPs. To the best of our knowledge, no previous publication in the literature has addressed this problem using perturbation analysis.

8 CONCLUSIONS

In this paper, we present an approach for dealing with uncertainty the probabilistic model checking of MDPs. Our main contribution is a perturbation approach that estimates the worst-case consequence of the effect of uncertainty—which is expressed as small perturbations to model probabilities—on the reachability verification of MDPs. We present an efficient algorithm that captures the effect of uncertainty in the form of condition numbers, by analyzing the underlying graph of the MDP and a subset of optimal executions. To evaluate the approach, we implemented a prototype in Python that interacts with PRISM [20]. We ran experiments on two cloud computing case studies, and the experimental results reveal that our approach is able to estimate the maximum effect of perturbation on quantitative verification of reachability properties of MDPs, and it does so with good performance. This work advances the state-of-the-art in perturbation analysis and probabilistic model checking of MDPs.

There are several directions for further study. First, we will explore efficient approaches for computing bounds through Monte Carlo sampling of the set of adversaries. Second, we will develop techniques for computing perturbation bounds for other kinds of properties besides reachability properties such as the full range of PCTL properties, reward properties, and so on.

ACKNOWLEDGMENTS

Y. Serrano, G. Su and D. S. Rosenblum were supported in part by the Singapore Ministry of Education under grants R-252-000-458-133 and MOE2015-T2-137-1, and by a grant from the Profile Partnership between Humboldt-Universität zu Berlin and the National University of Singapore. Y. Serrano was supported in part by a Research Scholarship from National University of Singapore. G. Su was supported in part by the National Natural Science Foundation of China under grant 61502260. The authors thank Quang-Trung Ta, Lars Grunske and Esteban Pavese for their valuable suggestions and discussions about this research, and the anonymous referees for their insightful comments.

REFERENCES

[1] Husain Aljazzar and Stefan Leue. 2009. Generation of Counterexamples for Model Checking of Markov Decision Processes. In *QEST*. 197–206.
[2] Christel Baier and Joost-Pieter Katoen. 2007. *Principles of Model Checking*. The MIT Press.
[3] Marco Baldi, Alessandro Cucchiarelli, Linda Senigagliesi, Luca Spalazzi, and Francesco Spegni. 2016. Parametric and Probabilistic Model Checking of Confidentiality in Data Dispersal Algorithms. In *International Conference on High*

Performance Computing & Simulation, HPCS 2016, Innsbruck, Austria, July 18-22, 2016. 476–483. https://doi.org/10.1109/HPCSim.2016.7568373

[4] Radu Calinescu, Kenneth Johnson, and Yasmin Rafiq. 2013. Developing Self-Verifying Service-Based Systems. In *2013 28th IEEE/ACM International Conference on Automated Software Engineering, ASE 2013, Silicon Valley, CA, USA, November 11-15, 2013.* 734–737. https://doi.org/10.1109/ASE.2013.6693145

[5] Rohit Chadha, Vijay Anand Korthikanti, Mahesh Viswanathan, Gul Agha, and YoungMin Kwon. 2011. Model Checking MDPs with a Unique Compact Invariant Set of Distributions. In *QEST*. 121–130.

[6] Taolue Chen, Yuan Feng, David S. Rosenblum, and Guoxin Su. 2014. Perturbation Analysis in Verification of Discrete-Time Markov Chains. In *CONCUR 2014 - Concurrency Theory - 25th International Conference, CONCUR 2014, Rome, Italy, September 2-5, 2014. Proceedings.* 218–233. https://doi.org/10.1007/978-3-662-44584-6_16

[7] Frank Ciesinski, Christel Baier, Marcus Größer, and Joachim Klein. 2008. Reduction Techniques for Model Checking Markov Decision Processes. In *Fifth International Conference on the Quantitative Evaluaiton of Systems (QEST 2008), 14-17 September 2008, Saint-Malo, France.* 45–54. https://doi.org/10.1109/QEST.2008.45

[8] Conrado Daws. 2005. Symbolic and Parametric Model Checking of Discrete-time Markov Chains. In *Proceedings of the First International Conference on Theoretical Aspects of Computing (ICTAC'04).* Springer-Verlag, Berlin, Heidelberg, 280–294. https://doi.org/10.1007/978-3-540-31862-0_21

[9] Kousha Etessami, Marta Z. Kwiatkowska, Moshe Y. Vardi, and Mihalis Yannakakis. 2008. Multi-Objective Model Checking of Markov Decision Processes. *Logical Methods in Computer Science* 4, 4 (2008).

[10] Antonio Filieri and Giordano Tamburrelli. 2013. Probabilistic Verification at Runtime for Self-Adaptive Systems. In *Assurances for Self-Adaptive Systems - Principles, Models, and Techniques.* 30–59. https://doi.org/10.1007/978-3-642-36249-1_2

[11] Antonio Filleri, Carlo Ghezzi, and Giordano Tamburrelli. 2011. Run-time Efficient Probabilistic Model Checking. In *Proceedings of the 33rd International Conference on Software Engineering (ICSE '11).* ACM, New York, NY, USA, 341–350. https://doi.org/10.1145/1985793.1985840

[12] Vojtech Forejt, Marta Z. Kwiatkowska, Gethin Norman, and David Parker. 2011. Automated Verification Techniques for Probabilistic Systems. In *SFM*. 53–113.

[13] Lin Gui, Jun Sun, Songzheng Song, Yang Liu, and Jin Song Dong. 2014. SCC-Based Improved Reachability Analysis for Markov Decision Processes. In *Formal Methods and Software Engineering - 16th International Conference on Formal Engineering Methods, ICFEM 2014, Luxembourg, Luxembourg, November 3-5, 2014. Proceedings.* 171–186. https://doi.org/10.1007/978-3-319-11737-9_12

[14] Ernst Moritz Hahn, Tingting Han, and Lijun Zhang. 2011. Synthesis for PCTL in Parametric Markov Decision Processes. In *NASA Formal Methods*. 146–161.

[15] Ernst Moritz Hahn, Holger Hermanns, and Lijun Zhang. 2009. Probabilistic Reachability for Parametric Markov Models. In *SPIN*. 88–106.

[16] John Kemeny, James Snell, and Anthony Knapp. 1976. *Denumerable Markov Chains* (2 ed.). Vol. 40. Springer-Verlag New York.

[17] Shinji Kikuchi and Yasuhide Matsumoto. 2011. Performance Modeling of Concurrent Live Migration Operations in Cloud Computing Systems Using PRISM Probabilistic Model Checker. In *IEEE International Conference on Cloud Computing, CLOUD 2011, Washington, DC, USA, 4-9 July, 2011.* 49–56. https://doi.org/10.1109/CLOUD.2011.48

[18] Adlen Ksentini, Tarik Taleb, and Min Chen. 2014. A Markov Decision Process-based Service Migration Procedure for Follow Me Cloud. In *IEEE International Conference on Communications, ICC 2014, Sydney, Australia, June 10-14, 2014.* 1350–1354. https://doi.org/10.1109/ICC.2014.6883509

[19] Marta Z. Kwiatkowska, Gethin Norman, and David Parker. 2005. Probabilistic Model Checking in Practice: Case Studies with PRISM. *SIGMETRICS Performance Evaluation Review* 32, 4 (2005), 16–21.

[20] Marta Z. Kwiatkowska, Gethin Norman, and David Parker. 2011. PRISM 4.0: Verification of Probabilistic Real-Time Systems. In *CAV*. 585–591.

[21] Marta Z. Kwiatkowska, David Parker, and Hongyang Qu. 2011. Incremental Quantitative Verification for Markov Decision Processes. In *DSN*. 359–370.

[22] Ruggero Lanotte, Andrea Maggiolo-Schettini, and Angelo Troina. 2007. Parametric Probabilistic Transition Systems for System Design and Analysis. *Formal Asp. Comput.* 19, 1 (2007), 93–109. https://doi.org/10.1007/s00165-006-0015-2

[23] Athanasios Naskos, Emmanouela Stachtiari, Anastasios Gounaris, Panagiotis Katsaros, Dimitrios Tsoumakos, Ioannis Konstantinou, and Spyros Sioutas. 2014. Cloud Elasticity using Probabilistic Model Checking. *CoRR* abs/1405.4699 (2014). http://arxiv.org/abs/1405.4699

[24] Athanasios Naskos, Emmanouela Stachtiari, Anastasios Gounaris, Panagiotis Katsaros, Dimitrios Tsoumakos, Ioannis Konstantinou, and Spyros Sioutas. 2015. Dependable Horizontal Scaling Based on Probabilistic Model Checking. In *15th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing, CCGrid 2015, Shenzhen, China, May 4-7, 2015.* 31–40. https://doi.org/10.1109/CCGrid.2015.91

[25] Athanasios Naskos, Emmanouela Stachtiari, Panagiotis Katsaros, and Anastasios Gounaris. 2015. Probabilistic Model Checking at Runtime for the Provisioning of Cloud Resources. In *Runtime Verification - 6th International Conference, RV 2015 Vienna, Austria, September 22-25, 2015. Proceedings.* 275–280. https://doi.org/10.1007/978-3-319-23820-3_18

[26] Mohsin Nazir. 2012. Cloud Computing: Overview & Current Research Challenges. *IOSR Journal of Computer Engineering* 8, 1 (2012), 14–22.

[27] Esko Nuutila and Eljas Soisalon-Soininen. 1994. On Finding the Strongly Connected Components in a Directed Graph. *Inf. Process. Lett.* 49, 1 (1994), 9–14. https://doi.org/10.1016/0020-0190(94)90047-7

[28] Alberto Puggelli, Wenchao Li, Alberto L. Sangiovanni-Vincentelli, and Sanjit A. Seshia. 2013. Polynomial-Time Verification of PCTL Properties of MDPs with Convex Uncertainties. In *Computer Aided Verification - 25th International Conference, CAV 2013, Saint Petersburg, Russia, July 13-19, 2013. Proceedings.* 527–542. https://doi.org/10.1007/978-3-642-39799-8_35

[29] Koushik Sen, Mahesh Viswanathan, and Gul Agha. 2006. Model-Checking Markov Chains in the Presence of Uncertainties. In *TACAS*. 394–410.

[30] Guoxin Su, Taolue Chen, Yuan Feng, and David S. Rosenblum. 2017. ProEva: Runtime Proactive Performance Evaluation Based on Continuous-Time Markov Chains. In *Proceedings of the 39th International Conference on Software Engineering, ICSE 2017, Buenos Aires, Argentina, May 20-28, 2017.* 484–495. http://dl.acm.org/citation.cfm?id=3097426

[31] Guoxin Su and David S. Rosenblum. 2013. Asymptotic Bounds for Quantitative Verification of Perturbed Probabilistic Systems. In *ICFEM*. 297–312.

[32] Guoxin Su and David S. Rosenblum. 2014. Perturbation Analysis of Stochastic Systems with Empirical Distribution Parameters. In *ICSE*. 311–321.

[33] Tarik Taleb and Adlen Ksentini. 2013. Follow Me Cloud: Interworking Federated Clouds and Distributed Mobile Networks. *IEEE Network* 27, 5 (2013), 12–19. https://doi.org/10.1109/MNET.2013.6616110

[34] Tarik Taleb, Adlen Ksentini, and Pantelis Frangoudis. 2016. Follow-Me Cloud: When Cloud Services Follow Mobile Users. *IEEE Transactions on Cloud Computing* PP, 99 (2016), 1–1. https://doi.org/10.1109/TCC.2016.2525987

[35] Shiqiang Wang, Rahul Urgaonkar, Ting He, Murtaza Zafer, Kevin Chan, and Kin K. Leung. 2015. Mobility-Induced Service Migration in Mobile Micro-Clouds. *CoRR* abs/1503.05141 (2015). http://arxiv.org/abs/1503.05141

[36] Mahmood Zaigham. 2014. *Cloud Computing: Challenges, Limitation and R & D.* Springer.