# Probabilistic Model Checking of Perturbed MDPs with Applications to Cloud Computing

Anonymous Author(s)

## ABSTRACT

Cloud computing is becoming more popular every day, the demand to keep providing reliable cloud computing services to users has become a vital factor. Probabilistic model checking, a formal verification technique, has been successfully applied in a variety of domains by identifying possible errors and formally verifying quantitative properties of the system. However, the presence of external factors and environmental changes may affect the quantitive aspects of the system model, represented as probabilities, and they may lead to invalid verification results. To address this problem, we use Markov Decision Processes (MDPs) and a perturbation analysis for studying the consequences of these uncertain quantities in probabilistic model checking of MDPs. We study the presence of uncertainty, expressed as small perturbations attached to the transitions probabilities, in the reachability verification of MDPs. One of our main contributions includes a novel technique for analyzing MDPs efficiently. We demonstrate the practical effectiveness of our approach by applying it to two case studies of cloud computing technology.

## KEYWORDS

Uncertainty, Probabilistic Model Checking, Perturbation Analysis, Markov Decision Processes, Cloud Computing

**ACM Reference format:**
Anonymous Author(s). 2016. Probabilistic Model Checking of Perturbed MDPs
with Applications to Cloud Computing. In *Proceedings of 11th Joint Meeting of the European Software Engineering Conference and the ACM SIGSOFT Symposium on the Foundations of Software Engineering, Paderborn, Germany, 4–8 September, 2017 (ESEC/FSE 2017),* 11 pages.
DOI: 10.1145/nnnnnnn.nnnnnnn

## 1 INTRODUCTION

In recent years, cloud computing has been developed extensively, and it has experienced great success. The main idea of cloud computing is to allow consumers to self-provision cloud-based software systems, application services, development platforms and virtualized infrastructures [37].


*ESEC/FSE 2017, Paderborn, Germany*

DOI: 10.1145/nnnnnnn.nnnnnnn

The main characteristics of cloud computing systems are its reduced cost, its increased storage since the cloud can scale dynamically and its flexibility as the cloud can adapt itself to changing requirements or environmental context. Coupled with these features, cloud computing faces several challenges such as intermittent connectivity, interoperability, reliability and availability of services to users, etc. [29, 37]. Recent studies [18, 20, 26–28] have uncovered that the quantitative verification via *probabilistic model checking* provides to cloud administrators a way to analyze the operations, resources, and helps in providing reliable services.

Probabilistic model checking is a formal verification technique that aims to verify properties of systems that exhibit stochastic behavior [2, 22]. Probabilistic model checkers such as PRISM [23] build a mathematical model (using one of Markov chain variants) that captures the probabilistic system's behavior against a formal specification of properties via PCTL logic language [4, 21]. It is important to emphasize that the probabilities associated with transitions of the model are *fixed* in probabilistic model checking.

In cloud computing systems, transitions that represent certain system quantities such as reliability, fault-proneness, etc., are unknown a priori. Hence, they must be determined *empirically* by observations of the system or determined by the expertise of cloud administrators. However, these quantities may change over time due to the stochastic nature of the system. They may be affected by unpredictable environmental factors such as failures, disasters, and workload spikes. Consequently, the presence of these uncertain events during the probabilistic verification of a cloud computing system may lead to invalid verification results.

For the reasons outlined above, to keep providing reliable cloud computing services to the 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 under the presence of uncertain events in the cloud.

In brief, our approach consists of three steps. First, we model cloud computing systems as Markov Decision Processes (MDPs), because MDPs exhibit a combination of nondeterministic and probabilistic transitions. Second, we use a Parametric Markov Decision Process (PMDP), a parametric variant of an MDP, for modeling the presence of uncertainty. Third, we formally characterize the worst-case consequence based on the PMDP and the reachability property to be verified using perturbation analysis. Mainly, we capture the effect of uncertainty in the form of *condition numbers*.

In summary, the main contributions of this paper are:

(1) A perturbation approach that estimates the worst-case consequence of the effect of uncertainty, which

is expressed as small perturbations to model parameters, in the reachability verification of MDPs.

(2) An efficient heuristic algorithm based on the underlying graph of the MDP, which overcomes the challenge of analyzing the MDP exhaustively.

(3) A prototype of our perturbation approach in Python interfaced with PRISM [23] for extracting the general information about the MDP model. 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 for our study. 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 introduces an algorithmic method for computing the maximum condition number of an MDP. The experimental evaluation of our approach and results are presented in Section 6. Section 7 discusses related work. Section 8 summarizes the main contributions of this paper and discusses future work.

## 2 MOTIVATING EXAMPLE

Consider a real-life scenario in service migration decision in cloud computing [19, 20, 36] where the objective is to enable a user to always be connected to the optimal cloud. Suppose we have a mobile network under a typical 3GPP network topology, illustrated in Figure 1, that uses the concept of Follow-Me Cloud (FMC) [35] for optimized disrupted-free cloud-based services for mobile users [19]. As shown in Figure 1, 3GPP network topology is typically divided into hexagonal cells. For the sake of simplicity, we consider a topology of $N = 5$ rings of cells, where a cell $i, j$ depicts cell $j$ in ring $i$. Each cell is covered by a Micro-Cloud (MC), where all MCs are interconnected with each other and monitored by the FMC controller.

Particularly, we model the service migration decision of the FMC controller that decides whether a service consumed by a user and hosted by a determined MC should be migrated to an optimal MC. We consider a user moves to any cell of next ring in every new time slot. We assume that a user starts a service in the ring 0 (cell $0, 1$ in Figure 2). A user in ring 0 can move to any neighboring cell with the same probability. From ring $i \geq 1$, a user can move to its adjacent cells in ring $i + 1$ with probability $q$.

Figure 2 depicts the mobile network modeled as an MDP. The state space $S$ is $S = \{s', s_1, s_2^3, s_2^2, s_3^3, s_3^2, s_4^1, s_4^2, mig\}$, with $s'$ representing the initial state in which a user starts using a service in the ring 0.

A state $s_i$ represents the offset between the user location in the ring $i$ and the location of the MC running the service, before the FMC controller decides between two nondeterministic actions– either to keep running the service on the MC of the previous cell, denoted as action $a_1$; or to migrate the service to an optimal MC, denoted as action $a_2$. These actions are depicted by solid and dashed lines in Figure 2, respectively.

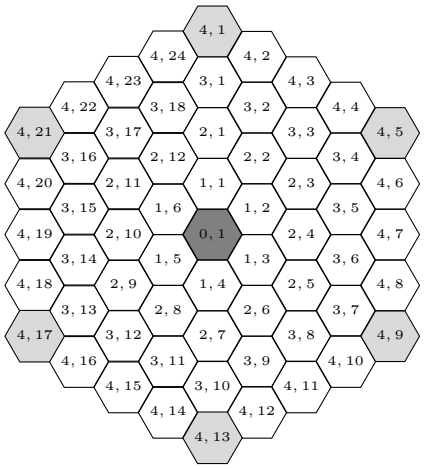


**Figure 1: A typical 3GPP mobile network**

As shown in Figure 2, all cells of ring 1 can be aggregated into one state $s_1$ since the user can move to any cell with the same probability. In contrast, the cells in ring 2 can be divided into two groups: (i) cells with 3 neighbors in ring 3, and (ii) cells with 2 neighbors in ring 3. Each group is aggregated into one state denoted as $s_2^3$ and $s_2^2$, respectively. Likewise, there are two groups of cells in ring 3: (i) cells with 3 neighbors in ring 4, and (ii) cells with 2 neighbors in ring 4, represented by states $s_3^3$ and $s_2^2$ respectively. Similarly, the groups with respect to ring 4 are: (i) cells with 1 neighbor in ring 3, and (ii) cells with 2 neighbors in ring 3, denoted as states $s_4^1$ and $s_4^2$, correspondingly.

Note that the *fixed* value of $q$ is a statistical estimation based on observations of the network behavior and/or application-specific measurements. For this reason and due to the cloud environment, this probability might be affected by tiny perturbations, which be originated from external factors (e.g., a power outage), environmental changes (e.g., an earthquake that could cause servers to go down), etc.

Suppose we are interested in verifying the reachability property: *the maximum probability of a mobile user is served by only one micro-cloud until he reaches one of the corners of the network (illustrated as the gray cells in Figure 1 and the colored state $s_4^2$ in Figure 2)*. Given that we are unable to measure the perturbations a priori, we capture the existence of perturbation in variables $x_1$ and $x_2$, which affect the values $\frac{1}{3}q$ and $\frac{2}{3}q$ respectively. Then, the *perturbed* transition probabilities can be expressed as $\frac{1}{3}q \pm x_1$ and $\frac{2}{3}q \pm x_2$. Therefore, it is reasonable to determine *the worst-case consequence of the effect of uncertainty on the reachability verification*. However, suppose we consider a small total perturbation embodied in variables $x_1$ and $x_2$. There exist infinite combinations of assigning values to $x_1$ and $x_2$ such that $x_1 + x_2 = 0$. This circumstance implies that the *exact* worst-case consequence

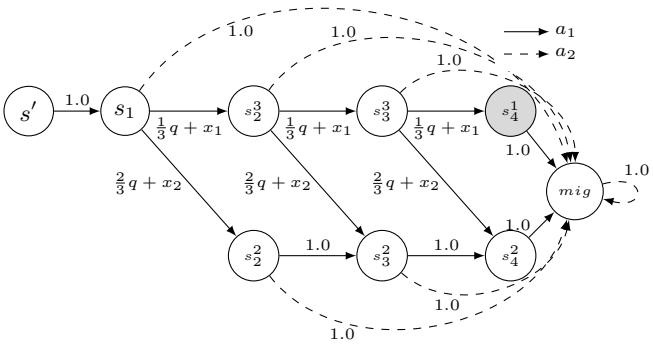


**Figure 2: Mobile network modeled as an MDP**

is hard to compute. In this context, we use a perturbation approach for *estimating* the worst-case consequence of uncertainty, which the goal of this paper.

Previous studies have proposed different perspectives to deal with uncertainty in probabilistic verification. Sen *et al.* [32] introduce intervals for characterizing the uncertainty in transitions and compute the exact bounds of the verification results on PCTL properties. However, these exact bounds are pointwise, which implies that wherever 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 Discrete Markov Chains (DTMCs) provides approximations rather than exact bounds when the transition probabilities are subject to small perturbations [33, 34].

To leverage this approach for MDP verification, we face the problem that an analysis of DTMCs induced by *all* memoryless adversaries (that always pick the same choice in a state and resolve the nondeterminism of the model) are needed for computing the largest asymptotic bounds. However, the computational complexity of such a method is exponential in the size of MDP. In this context, we present an approach that estimates the worst-case consequence of the effect of uncertainty, which is expressed as small perturbations to model parameters, in the reachability verification of MDPs.

# 3 BACKGROUND

This section briefly recalls the background that is required for the description of our approach. We first revisit the formal definition of Markov decision processes and some general notions. Then, we present its parametric extension which underlies our perturbation analysis. Finally, we introduce the reachability verification of properties on Markov decision processes researched 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 $\mathcal{M} = (S, Act, \mathbf{Q}, \iota_{init}, AP, L)$ 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 $\forall s \in S$, $\forall a \in Act$, $\sum_{s'\in S} \mathbf{Q}(s,a,s') \in \{0,1\}$, $\iota_{init} : S \to [0,1]$ is the initial state distribution such that $\sum_{s\in S} \iota_{init}(s) = 1$, $AP$ 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 the 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 to determine its successor in an MDP. First, a nondeterministic choice between the enable actions $Act(s)$. Suppose action $a \in Act(s)$ has been selected. Second, based on the distribution given by $a$, a successor state $\bar{s}$ is chosen randomly 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 $\pi(i)$ to denote the $(i+1)$th state in the path $\pi$. A *finite path* $\rho = s_0 \xrightarrow{a_0} s_1 \xrightarrow{a_1} \ldots \xrightarrow{a_{n-1}} s_n$, $|\rho| = n$ denotes its length and $last(\rho) = s_n$ its last state. The *probability measure* of a finite path $\rho$, denoted $p(\rho)$, is defined as the product $\prod_{i=0}^{n-1} \mathbf{Q}(s_i, a_i, s_{i+1})$. We denote $Path_s$ and $Path_s^{fin}$ the set of all infinite and finite paths starting in state $s$, respectively. A path through an MDP represents an *execution* (i.e. one possible behavior) of the system.

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

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 $\mathcal{M} = (S, Act, \mathbf{Q}, \iota_{init}, AP, L)$ is a function $\mathcal{A} : Path^{fin} \to Dist(Act)$ 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 $\rho, \rho' \in Path^{fin}$ 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 $Path_s^{\mathcal{A}}$, where $Path_s^{\mathcal{A}} \subseteq Path_s$ and induces a probability space $Prob_s^{\mathcal{A}}$ over the paths $Path_s^{\mathcal{A}}$ [17]. 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 states and actions 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, 25] have used parametric Markov decision processes (PMDPs) for expressing *not* fixed transition probabilities in MDPs. We use PMDPs for capturing the presence of uncertain events in the probabilistic transitions of MDPs.

*Definition 3.4. Parametric Markov Decision Process (PMDP)* A PMDP based on $\mathcal{M} = (S, Act, \mathbf{Q}, \iota_{init}, AP, L)$ is a tuple $\mathcal{M}[\mathbf{x}] = (S, Act, \mathbf{Q}[\mathbf{x}], \iota_{init}, AP, L)$ where $S$, $Act$, $\iota_{init}$, $AP$ 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 and its parametric variant 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 study will be explained in Section 4. For the sake of our study, similar than 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.* The minimum and maximum probability of reaching a set of target states $T \subseteq S$ from $s$, over all possible adversaries is defined as follows:

$$p_s^{min}(T) = \inf_{\mathcal{A} \in Adv_{\mathcal{M}}} p_s^{A}(T) \tag{1}$$

$$p_s^{max}(T) = \sup_{\mathcal{A} \in Adv_{\mathcal{M}}} p_s^{A}(T) \tag{2}$$

where $p_s^{A}(T) = Prob_s^{\mathcal{A}}(\{\pi \in Path_s^{A} \mid \exists i \cdot \pi(i) \in T\})$.

*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 follows:

$$p_{s\xrightarrow{a}}^{min}(T) = \inf_{\mathcal{A} \in Adv_{\mathcal{M}}} Prob_s^{\mathcal{A}}(Path_{s\xrightarrow{a}}^{\mathcal{A}}(T)) \tag{3}$$

$$p_{s\xrightarrow{a}}^{max}(T) = \sup_{\mathcal{A} \in Adv_{\mathcal{M}}} Prob_s^{\mathcal{A}}(Path_{s\xrightarrow{a}}^{\mathcal{A}}(T)) \tag{4}$$

where $Path_{s\xrightarrow{a}}^{\mathcal{A}}(T) = \{\pi \in Path_s^{\mathcal{A}} \mid \pi(0) = s, \mathcal{A}(s) = a, \exists i \cdot \pi(i) \in T\}$.

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, $Adv_{s\xrightarrow{a}}^{min}$ and $Adv_{s\xrightarrow{a}}^{max}$ denote the sub-sets of optimal adversaries that achieve the probabilities in Definition 3.6.

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

$$\Upsilon_{s\xrightarrow{a}}^{min} = \{\upsilon \mid \upsilon = sas_1A'(s_1)\dots s_n, \forall A' \in Adv_{s\xrightarrow{a}}^{min}\} \tag{5}$$

$$\Upsilon_{s\xrightarrow{a}}^{max} = \{\upsilon \mid \upsilon = sas_1A'(s_1)\dots s_n, \forall A' \in Adv_{s\xrightarrow{a}}^{max}\} \tag{6}$$

where adversary $A' : S^+ \rightarrow Act$ such that $A(ss_1s_2\dots s_n)$, $\forall ss_1s_2\dots s_n \in S^+$.

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 utilized for computing *quantitatively* the minimum and maximum reachability probabilities using some standard technique such as value iteration, linear programming or policy iteration [12, 24]. In practice, probabilistic model checkers including PRISM [23] 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.* [24] 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 the uncertainty in the reachability verification of MDPs. After the modeling of a cloud computing system as an MDP, we follow two steps: (i) we introduce the uncertainty, expressed as small perturbations to model parameters, using a parametric variant of an MDP, and (ii) we formally characterize and capture the presence of uncertain phenomena in the form of condition numbers.

## 4.1 Modeling Uncertainty

Inspired by previous studies [11, 33, 34], we use a parametric variant of an MDP $\mathcal{M}$ for introducing the presence of uncertainty in the nondeterministic model. Before building

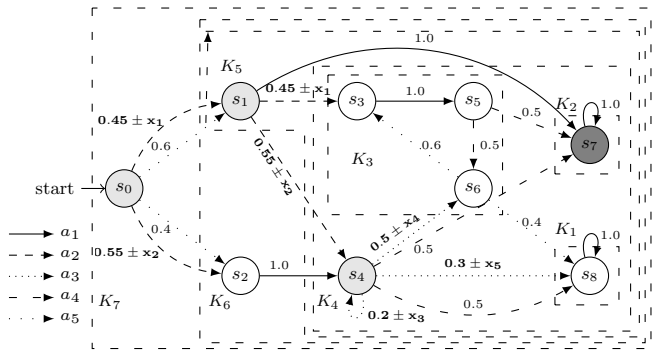


**Figure 3: An Illustrative Example**

the parametric variant, we define the parametric probabilistic transition function by identifying the actions and the 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 such that $\forall s, \bar{s} \in S$ and $a \in Act(s)$, the 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 $x_i \in \mathbf{x}$ and $1 \leq i \leq m$, 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 [1]. Based on $\mathbf{Q}[\mathbf{x}]$, we state the following propositions:

Proposition 4.2. *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 $1 \leq j < k \leq m$. 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 $Act_I = \{a \in Act \mid \mathbf{Q}[\mathbf{x}](s, a, \bar{s}) = p + x_i, \forall x_i \in I\}$ and a set of perturbed states $S_I = \{s \in S \mid \mathbf{Q}[\mathbf{x}](s, a, \bar{s}) = p(s, a, \bar{s}) + x_i, \forall x_i \in I\}$. It is always assumed that the vector $\mathbf{x} = (x_1, \cdots, x_m)$ is within the set $\mathcal{U} := \{\mathbf{x} \in \mathbb{R}^m \mid \forall I \in \mathscr{I}, \sum_{x_i \in I} x_i = 0, \mathbf{Q}[\mathbf{x}] \equiv \mathbf{Q}\}$. 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 $a_1$, $a_2$ and $a_3$, which are depicted as solid, dashed and 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 $Act_{I_1} = \{a_3\}$ and $S_{I_1} = \{s_0, s_1\}$, and, (2) $I_2 = (x_3, x_4, x_5)$ associated with $Act_{I_2} = \{a_2\}$ and $S_{I_2} = \{s_4\}$.

## 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 $p_s^{\mathcal{A}}(T)$ is the probability of reaching a target set $T$ from state $s$ under the adversary $\mathcal{A}$. 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 $\lambda(\mathbf{x}) = \sup_{\mathcal{A}[\mathbf{x}] \in Adv_{\mathcal{M}[\mathbf{x}]}} \{p_\iota^{\mathcal{A}[\mathbf{x}]}(T)\} - \sup_{\mathcal{A} \in Adv_{\mathcal{M}}} \{p_\iota^{\mathcal{A}}(T)\}$, where $\mathbf{x} \in \mathcal{U}$.

Through Definition 4.4, note that adversary $A$ represents the unperturbed version of adversary $\mathcal{A}[\mathbf{x}]$. 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. The difficulty is that there might be infinite combinations of $\mathbf{x}$. For this reason, we define the *perturbation characterization function* $\gamma$ 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 $\gamma(\mathbf{x}) = \sup_{\mathcal{A}[\mathbf{x}] \in Adv_{\mathcal{M}[\mathbf{x}]}} \{p_\iota^{\mathcal{A}[\mathbf{x}]}(T) - p_\iota^{\mathcal{A}}(T)\}$, where $\mathbf{x} \in \mathcal{U}$.

*4.2.2 Maximum Condition Number.* Inspired by previous studies [33, 34] 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 $\delta$ is a small positive number. We define the MaxCN as follows:

[1] This is because transitions with probabilities 0 or 1 are not probabilistic, and thus not subject to perturbations.

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

$$\kappa = \lim_{\delta \to 0} \sup \left\{ \frac{\gamma(\mathbf{x})}{\delta} \mid \mathbf{x} \in \mathcal{U}, \|\mathbf{x}\| \leq \delta, \delta > 0 \right\} \tag{7}$$

Let $\rho_1(\mathbf{x}) = \mathbf{h} \cdot \mathbf{x}$ be the linear approximation of $\gamma$ for some vector $\mathbf{h}$ such that $|\mathbf{h}| = |\mathbf{x}|$. Note that the existence of the limit in Eq. (7) is guaranteed by the fact that $\gamma$ is differentiable at $\mathbf{0}$ [6].

We formulate that the coefficients of vector $\mathbf{h}$ is the concatenation of all sub-vectors $\mathbf{h}_I = (h_j, \ldots, h_k)$ for all $I \in \mathscr{I}$ and $1 \leq j < k \leq m$. 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:

Proposition 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 $\kappa$ is defined as:*

$$\kappa = \frac{1}{2} . \max_{I \in \mathscr{I}} \triangle h_I \tag{8}$$

The key point is to compute the largest $h^+$ and smallest $h^-$ coefficients of each sub-vector $I \in \mathscr{I}$. However, it may not be feasible to compute these coefficients in an algorithmic time since each adversary in the PMDP model provides a possible value for $h^+$ and $h^-$. To overcome this problem, we introduce a heuristic method by analyzing the PMDP efficiently and consequently to compute the MaxCN of the model.

# 5 MAXCN ALGORITHM

In this section, we present our algorithmic approach to computing the MaxCN of a PMDP. In the following let $\mathcal{M}[\mathbf{x}] = (S, Act, \mathbf{Q}[\mathbf{x}], \iota_{init}, AP, L)$ be a PMDP with target set $T \subseteq S$ and the partition $\mathscr{I}$ over the vector $\mathbf{x}$. Given that for each perturbed sub-vector $I \in \mathscr{I}$, we need to compute the largest and the smallest coefficient for calculating $\kappa$ (Eq. 8), we first demand to compute the sub-vector of coefficients $\mathbf{h_I}$.

Consider again the example PMDP from Figure 3, the key point is to compute $\mathbf{h_{I_1}} = \{h_1, h_2\}$ and $\mathbf{h_{I_2}} = \{h_3, h_4, h_5\}$. For instance, the value of coefficient $h_1$, which is related to variable $x_1$, depends on the reachability probabilities from states $s_1$ and $s_3$ to the target set $T$. On account of these states are successors of the perturbed states $s_0$ and $s_1$ under the perturbed action $a_3$ and associated with variable $x_1$, respectively. In similar way, $h_2$ depends on the reachability probabilities from states $s_2$ and $s_4$ to $T$.

The challenge is to determine which probabilities (similarly which adversaries) assign to the coefficient $h_1$ and $h_2$ such that they maximize $\triangle h_{I_1}$. Our algorithmic method 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, 24], 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 [30]. Then we determine the *reversed topological order* $\zeta$ among the set of SCCs such that $\Pi = \{K_1, \ldots, K_N\}$ where $N$ denotes the number of SCCs in the PMDP and $K_j$ will appear before $K_i$ in $\zeta$ 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 $\zeta$: $K_1 = \{s_8\}$, $K_2 = \{s_7\}$, $K_3 = \{s_3, s_5, s_6\}$, $K_4 = \{s_4\}$, $K_5 = \{s_1\}$, $K_6 = \{s_2\}$ and $K_7 = \{s_0\}$.

In the following let $I = (x_j, \ldots, x_k)$ 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 $I$, for each successor state $s(x_i) \in S$ we generate a Sub-PMDP of $\mathcal{M}[\mathbf{x}]$. 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 $\Pi^\star = \{K_1, \ldots, K_x\}$ such that $s(x_i) \in K_x$ and $\Pi^\star \subseteq \Pi$;
2 $\mathcal{M}^* \leftarrow$ Sub-PMDP $(S^*, Act^*)$ where $S^* = Act^* = \emptyset$;
3 **for** *SCC* $K \in \Pi^*$ **do**
4 $\quad S^\star \leftarrow K$;
5 $\quad Act(K) = \bigcup_{s \in K} Act(s)$ `// Set of actions of` $K$ ;
6 $\quad Act^\star \leftarrow Act(K)$;
7 $\quad$ **if** *$K$ is no MEC* **then**
8 $\quad\quad Act^{out}(K) = \{a \in Act(K) \mid \exists s \in K \cdot succ(s, a) \subseteq Out(K)\}$ `// Set of outgoing actions of` $K$;
9 $\quad\quad Act^+ :=$ prune_actions$(\mathcal{M}^*, Act^{out}(K))$;
10 $\quad\quad Act^\star \leftarrow Act^+$ `// Update` $Act^\star$ `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 $\Pi^*$. 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) that 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). Last, generatorSubPMDP returns a Sub-PMDP $\mathcal{M}^*$.

*5.1.1 Pruning actions.* Let us recall that our target is to find coefficients that maximize $\triangle h_I$ and each SCC can be analyzed independently. With this in mind, we evaluate the set of outgoing actions of each SCC, and we discard actions that 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 $\quad \bar{S} \leftarrow succ(s, a^+)$ for some $s \in K$ ;
4 $\quad \Upsilon(\bar{S}) = \left\{ \Upsilon^{min}_{\bar{s} \xrightarrow{a^*}} \cup \Upsilon^{max}_{\bar{s} \xrightarrow{a^*}} \mid \forall \bar{s} \in \bar{S}, \forall a^* \in Act^*(\bar{s}) \right\}$
$\quad$ `// Set of optimal executions in` $\mathcal{M}^*$ ;
5 $\quad T^* \leftarrow (\Upsilon(\bar{S}), \bar{S})$ `// Build an execution tree` ;
6 $\quad \Omega(T^*) = \{\omega_1, \ldots, \omega_m\}$ `// Set of branches` $T^*$;
7 $\quad \Omega^* = \text{argmax}_{\omega \in \Omega(T)} \triangle p(\omega)$ where
$\quad \triangle p(\omega) = p^{max}(\omega) - p^{min}(\omega)$;
8 $\quad Act' = \{Act(v) \mid \forall \omega \in \Omega^* \cdot \forall (v, p) \in \omega\}$ `// Set of relevant actions based on` $\Omega^*$ ;
9 $\quad Act^+ \leftarrow Act'$
10 **return** $Act^+$

As Algorithm 2 shows, for each outgoing action $a^+ \in Act^{out}(K)$, prune_actions first computes the set of successor states of state $s \in K$ via action $a^+$ (line 3). Second, given the Sub-PMDP $\mathcal{M}^*$, prune_actions computes the set of optimal executions from the set of all successors states $\bar{S}$, denoted as $\Upsilon(\bar{S})$ and 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 parent of node $b = (v_b, p_b)$ if only if $v_b \subseteq v_a$ or $v_a \subset v_b$,

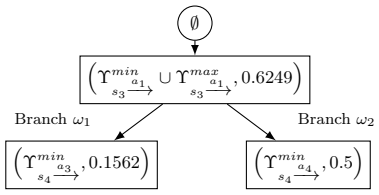


**Figure 4: Example of an Execution Tree**

- A branch $\omega$ of $T^*$ is a path from $v^*$ to a leaf node where $|\omega| = |\bar{S}|$,
- The minimum and the maximum probability of a branch $\omega$ 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 $\Omega^*$ (line 8).

As an illustration, Figure 4 shows 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 $\Omega(T^*)$ is $\Omega(T^*) = \{\omega_1, \omega_2\}$ and the subset $\Omega^*$ is $\Omega^* = \{\omega_1\}$ due to $\triangle p(\omega_1) > \triangle p(\omega_2)$. As a result, the action $a_4$ is pruned.

## 5.2 Computing $\triangle h_I$

The SCC-based procedure as introduced in the previous section contains the relevant information for the computation of the maximum difference $\triangle h_I$ of a sub-vector $h_I$. This procedure is named as MaxDiff and depicted in Algorithm 3.

**Algorithm 3:** MaxDiff($I$)

**Input**: A sub-vector $I$

1 $\bar{S}_I = \{\bar{s} \in S \mid \mathbf{Q}[\mathbf{x}](s, a, \bar{s}) = p(s, a, \bar{s}) + x_i, \forall x_i \in I\}$
$\quad$ `// Perturbed successors states`;
2 $\Upsilon_I \leftarrow \emptyset$;
3 **for** $s(x_i) \in \bar{S}_I$ **do**
4 $\quad (S^*, Act^*) \leftarrow$ generatorSubPMDP$(s(x_i))$;
5 $\quad \Upsilon_I(s(x_i)) \leftarrow \left\{ \Upsilon^{min}_{s(x_i) \xrightarrow{a*}} \cup \Upsilon^{max}_{s(x_i) \xrightarrow{a*}} \mid \forall a^* \in Act^*(s(x_i)) \right\}$;
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 first computes the set of perturbed successors states of the sub-vector $I$, denoted as $\bar{S}_I$ (line 1). Second, it constructs a dictionary $\Upsilon_I$ 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 $s(x_i)$, the function invokes generatorSubPMDP($s(x_i)$) which returns a sub-PMDP $(S^*, Act^*)$. Based on $(S^*, Act^*)$, MaxDiff selects the set of optimal executions with respect to $s(x_i)$ and stores into the dictionary $\Upsilon_I$.

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 $\Upsilon_I$ 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 $(s, a, \bar{s}, \upsilon)$ where $s$ is a perturbed state, $a$ is a perturbed action, $\bar{s}$ is a perturbed successor state and $\upsilon$ is an execution, and
  - $h_i$ is the coefficient related to the variable $x_i$.
- A coefficient $h_i = \sum_{(s,a,\bar{s},\upsilon)}\{p^{max}_{init}(s) * \Lambda\}$ where:
  - If $\mathbf{Q}[\mathbf{x}](s, a, s) > 0$ then $\Lambda = \frac{p(\upsilon)}{1-\mathbf{Q}[\mathbf{x}](s,a,s)}$
  - If $\bar{s} \notin T$ and $\mathbf{Q}[\mathbf{x}](s, a, s) = 0$ then $\Lambda = p(\upsilon)$
  - Otherwise, $\Lambda = 1$.
- A node $x = (\mathbf{v_i}, h_i)$ is a parent of node $y = (\mathbf{v_{i+1}}, h_{i+1})$, if only if $\mathbf{v_i}(\upsilon) \cap \mathbf{v_{i+1}}(\upsilon) \neq \emptyset$
- A branch $\omega$ of $T$ is a path from $v^*$ to a leaf node where $|\omega| = |I|$,
- The largest and smallest coefficients of a branch $\omega$ 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 help us 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 set of adversaries.

# 6 EXPERIMENTAL EVALUATION

In this section, we present two case studies of cloud computing which were inspired by previous studies [3, 19, 20, 36]. As we present each case study, we describe briefly the 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 [23] to extract information about the MDP model. The prototype computes the maximum condition number (MaxCN) for each MDP model using the algorithm presented in the previous sections. All our experiments have been conducted on a machine with Intel Core 2 Duo at 3.06 GHz and 8 GB RAM.

**Table 1: Experimental Results**

| Model | MDP Data | | | MaxCN ($\kappa$) | Time(sec) | |
|---|---|---|---|---|---|---|
| | $\sharp S$ | $\sharp T$ | $\sharp Adv$ | | *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 |

## 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 FMC controller that decides whether a service consumed by a user and hosted by a particular micro-cloud should be migrated to an optimal micro-cloud. The cloud computing services work under a typical 3GPP network topology of $N$ rings of cells. We performed experiments for systems with network topology of $N = 5$ and 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 verify the maximum probability of a user is served by only one micro-cloud until:

- P1 A user reaches one of the corners of the topology.
- P2 A user reaches any cell of the last ring except its corners.
- P3 A user reaches one of the corners of $\lfloor \frac{N}{2} \rfloor$ ring.

As mentioned in Section 2, due to the cloud environment and other external factors, the value of $q$ can be affected by small perturbations. We capture the perturbations in variables $x_1$ and $x_2$, which are attached to the probabilistic transitions $\frac{1}{3}q$ and $\frac{2}{3}q$, respectively (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 shown in Table 1.

As can be seen from Table 1, for each model instance, the first four columns show the name model, the property, and the general information of an MDP model as the total number of states, transitions and (memoryless) adversaries in the model. The remaining columns show the MaxCN computed for the model and the running time (provided in seconds) of our algorithm, denoted as *MaxCNAlg*. For the sake of evaluating the performance of our algorithm, Table 1 also presents the running time of an exhaustive algorithm, named as *ExhAlg*. This algorithm analyzes exhaustively the total set of adversaries in an MDP. For each adversary, *ExhAlg* computes a condition number, then it returns the maximum of all computed condition numbers. The best total time is **boldfaced**.

As Table 1 shows that the experiments have been conducted over a considerable range of MDP sizes. From an MDP model with 32 adversaries to a large MDP model with 2048 adversaries. The size of the set of adversaries is crucial

**Table 2: Experimental Data of $\mathcal{M}^5$ model**

| Model | $\delta$ ($\times 10^{-3}$) | $Pr^{max}$ | $\triangle Pr$ ($\times 10^{-4}$) | $\alpha$ ($\times 10^{-4}$) |
|---|---|---|---|---|
| $\mathcal{M}^5$ | 0 | 0.03703 | - | - |
| $\mathcal{M}_1^5$ | 1 | 0.03687 | 1.6 | $\pm 1.11$ |
| $\mathcal{M}_2^5$ | 2 | 0.03670 | 3.2 | $\pm 2.22$ |
| $\mathcal{M}_3^5$ | 3 | 0.03653 | 4.9 | $\pm 3.33$ |

since it represents the set of paths and executions that the algorithms need to analyze.

Comparing the running time between the two depicted algorithms, Table 1 reveals that the time computed by *MaxCNAlg* is always considerable smaller than *ExhAlg*. These findings are consistent. 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 in all the experiment settings. 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 perturbation approach provides an adequate estimation of the worst-case effect of uncertain phenomena in the reachability verification of MDPs, we computed the perturbation estimation generated by the MaxCN algorithm. For this purpose, we conducted experiments on the model $\mathcal{M}^5$ and the reachability property $P1$. These experiments are depicted in Table 2.

First, we constructed several potential non-parametric models with sufficiently small perturbation distance, denoted as $\mathcal{M}_j^5$ ($j \geq 1$). It is worth mentioning that the values of the symbolic variables are assigned deliberately. Second, for each perturbed model, we computed the actual maximum probability, denoted as $Pr_j^{max}$. Third, we compared the difference between the maximum probability of an unperturbed model ($\delta = 0$) and a perturbed model ($\delta > 0$). Formally, $\triangle Pr = Pr^{max} - Pr_j^{max}$. Lastly, we calculated the perturbation estimation ($\alpha = \pm\kappa\delta$), the product of the maximum condition number $\kappa$ (shown in Table 1) and a perturbed distance $\delta > 0$. The perturbation estimation $\alpha$ allows us to compute the *perturbation bound*, which is the intuitive estimation of the worst-case effect of uncertainty. Particularly, the perturbation bound is given by an idealized reachability probability (maximum) in the absence of perturbation plus or minus the perturbation estimation.

As shown in Table 2, for each model the perturbed estimation represents a safe prediction of the worst-case scenario. For example, for the model $\mathcal{M}_1^5$, given that $\delta = 1 \times 10^{-4}$, the difference between the probability of the perturbed model and the unperturbed model is $1.6 \times 10^{-4}$. This result lies on the perturbation bound $[0.03691, 0.03714]$.

The correlation between the probability verifications of the perturbed model $Pr_j^{max}$ and the perturbation estimation $\alpha$ 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-effect uncertainty in the reachability verification of MDPs is captured by the maximum condition number computed by our approach.

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

| Model | $\delta$ ($\times 10^{-3}$) | $Pr^{max}$ | $\triangle Pr$ ($\times 10^{-3}$) | $\alpha$ ($\times 10^{-3}$) |
|---|---|---|---|---|
| $\mathcal{M}^c$ | 0 | 0.7839 | - | - |
| $\mathcal{M}_1^c$ | 1 | 0.7833 | 0.6 | $\pm 1.81$ |
| $\mathcal{M}_2^c$ | 2 | 0.7827 | 1.2 | $\pm 3.62$ |
| $\mathcal{M}_3^c$ | 3 | 0.7821 | 1.8 | $\pm 5.43$ |

## 6.2 Confidentiality in Cloud Computing

Inspired by Baldi *et al.* [3], we study the confidentiality in data dispersal algorithms against an intruder in the cloud. Data dispersal algorithms provide a methodology for storing information in distinct slices (dispersed) across multiple servers, which are located in different locations.

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

The intruder, named as *slice attacker*, intercepts every slice independently from the previously intercepted ones. A slice attacker needs to intercept at least $k$ slices of information for reconstructing the message. The probability that this intruder intercepts a server successfully is $p$.

Due to the nature of cloud computing systems, the statistical estimation 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 in 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: $m = 2$, $c = 3$, $n = 3$, $p = 0.7$ and $k = 2$. The MDP model size is 152 states and 373 transitions. The number of adversaries is $> 116 \times 10^{10}$ approximately. In similar way than the previous case study, we built three potential non-parametric models with a small perturbation distance. These models are denoted as $\mathcal{M}_1^c$, $\mathcal{M}_2^c$ and $\mathcal{M}_3^c$ and named as perturbed models.

In order to evaluate our approach, we compute the maximum condition number of the model by using *MaxCNAlg* algorithm. As a result, we obtain that the maximum condition number $\kappa = 1.81$. Since the computational complexity of *ExhAlg* is exponential with respect to the size of the MDP, we are unable to compare the running time of *MaxCNAlg* with the running time of *ExhAlg* algorithm for this case study.

Table 3 shows the perturbation estimation generated by our approach. Note that Table 3 has been constructed in a similar fashion than 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 $\mathcal{M}_1^c$, $\mathcal{M}_2^c$ and $\mathcal{M}_3^c$ 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. This results are consistent with the experimental data of the previous case study.

On the other hand, we can observe in Table 3 that the perturbation estimation $\alpha$ for the model $\mathcal{M}_3^c$ is $\pm 5.43$, which produces a perturbation bound between 0.77667 and 0.78753. This perturbation bound indicates a safe prediction of the worst-case scenario of the uncertainty in the model.

The experimental results for both case studies show a clear advantage of our method over the exhaustive method. Our algorithm, *MaxCNAlg*, successfully computes in the allocated time frame the MaxCN of an MDP model without analyzing all adversaries. Likewise, the results validate that our perturbation approach provides an adequate estimation of the worst-case effect of uncertain phenomena in the reachability of MDPs.

## 7 RELATED WORK

In this section, we briefly discuss works related to our study, 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 the limitations of cloud computing systems. For example, Naskos *et al.* [26–28] developed an approach to enforcing elasticity, the most prominent advantage of cloud computing, and the provisioning of cloud resources through the dynamic instantiation and online quantitative verification of MDPs. The authors demonstrated that the quantitative verification at runtime could be also beneficial for the management of requirements of the cloud. On the other hand, researchers at Fujitsu [18] 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.

In probabilistic model checking, there are previous studies on Markov models with uncertainties. Sen *et al.* [32] 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.* [31] present P-completeness complexity bounds for an extension of IMDPs.

Another study that deals with uncertainties in probabilistic model checking is given by Daws [8]. His study presents a language-theoretic approach to the 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 run-time probabilistic model, presenting a parameterized version of the Gauss-Jordan elimination algorithm to address the uncertainty problem in DTMC models for reliability prediction. Last, Su *et al.* [6, 33, 34] introduce a mathematical framework for small perturbations, also viewed as uncertainty, in reachability verification of DTMCs using perturbation analysis. In these studies, the authors present parametric DTMCs for modeling the perturbation and define two asymptotic bounds namely, condition numbers and quadratic bounds. This approach may not be practical for analysis all DTMCs induced by adversaries of an MDP, due to the exponential computational complexity. In this context, our work presents an efficient approach to address the effect of uncertainty in the verification of MDP. To the authors' best knowledge, no publication in the literature exists that addresses this issue using perturbation analysis

## 8 CONCLUSIONS

In this paper, we studied the presence of the uncertain phenomena in 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 parameters, in the reachability verification of MDPs. We use 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 [23]. We ran experiments on two cloud computing case studies, and the experimental results reveal that our approach is able to estimate the maximum effect perturbation on quantitative verification of an MDP with good performance. This work advances the state-of-the-art on the combination of 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 condition properties, PCTL properties, rewards properties, etc.

## 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. DOI:http://dx.doi.org/10.1109/HPCSim.2016.7568373

[4] Andrea Bianco and Luca de Alfaro. 1995. Model Checking of Probabilistic and Nondeterministic Systems. In *FSTTCS*. 499–513.

[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. DOI:http://dx.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. DOI:http://dx.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. DOI:http://dx.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. DOI:http://dx.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. DOI:http://dx.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. DOI:http://dx.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] David Henriques, Jo ao Martins, Paolo Zuliani, André Platzer, and Edmund M. Clarke. 2012. Statistical Model Checking for Markov Decision Processes. In *Ninth International Conference on Quantitative Evaluation of Systems, QEST 2012, London, United Kingdom, September 17-20, 2012*. 84–93. DOI:http://dx.doi.org/10.1109/QEST.2012.19

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

[18] 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. DOI:http://dx.doi.org/10.1109/CLOUD.2011.48

[19] 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. DOI:http://dx.doi.org/10.1109/TCC.2016.2525987

[20] 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. DOI:http://dx.doi.org/10.1109/ICC.2014.6883509

[21] Marta Kwiatkowska, Gethin Norman, and David Parker. 2010. Advances and Challenges of Probabilistic Model Checking. In *2010 48th Annual Allerton Conference on Communication, Control, and Computing (Allerton)*. IEEE, 1691–1698.

[22] 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.

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

[24] Marta Z. Kwiatkowska, David Parker, and Hongyang Qu. 2011. Incremental quantitative verification for Markov decision processes. In *DSN*. 359–370.

[25] 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. DOI:http://dx.doi.org/10.1007/s00165-006-0015-2

[26] 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

[27] 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. DOI:http://dx.doi.org/10.1109/CCGrid.2015.91

[28] 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. DOI:http://dx.doi.org/10.1007/978-3-319-23820-3_18

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

[30] 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. DOI:http://dx.doi.org/10.1016/0020-0190(94)90047-7

[31] 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. DOI:http://dx.doi.org/10.1007/978-3-642-39799-8_35

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

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

[34] Guoxin Su and David S. Rosenblum. 2014. Perturbation analysis of stochastic systems with empirical distribution parameters. In *ICSE*. 311–321.

[35] Tarik Taleb and Adlen Ksentini. 2013. Follow me cloud: interworking federated clouds and distributed mobile networks. *IEEE Network* 27, 5 (2013), 12–19. DOI:http://dx.doi.org/10.1109/MNET.2013.6616110

[36] 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

[37] Mahmood Zaigham. 2014. *Cloud Computing : challenges, limitation and R & D*. Springer.