Directed Explicit State-Space Search in the Generation of Counterexamples for Stochastic Model Checking

Husain Aljazzar and Stefan Leue, *Senior Member*, *IEEE*

Abstract—Current stochastic model checkers do not make counterexamples for property violations readily available. In this paper, we apply directed explicit state-space search to discrete and continuous-time Markov chains in order to compute counterexamples for the violation of PCTL or CSL properties. Directed explicit state-space search algorithms explore the state space on-the-fly, which makes our method very efficient and highly scalable. They can also be guided using heuristics which usually improve the performance of the method. Counterexamples provided by our method have two important properties. First, they include those traces which contribute the greatest amount of probability to the property violation. Hence, they show the most probable offending execution scenarios of the system. Second, the obtained counterexamples tend to be small. Hence, they can be effectively analyzed by a human user. Both properties make the counterexamples obtained by our method very useful for debugging purposes. We implemented our method based on the stochastic model checker PRISM and applied it to a number of case studies in order to illustrate its applicability.

Index Terms—Directed explicit state-space search, heuristic search, counterexamples, stochastic model checking.

1 Introduction

1.1 Motivation

Debugging is an important activity during the design, implementation, and integration of systems. Finite-state, algorithmic verification techniques, such as model checking [1], have recently been used extensively to aid the developer in locating errors. To this end, it is necessary that model checkers provide meaningful diagnostic information that facilitates debugging. During the model checking of functional properties, such information can be made available without additional effort. For instance, in the case of a safety property violation, an explicit-state model checker such as SPIN [2] delivers a *failure path* from the initial state of the system to a property violating state. In model checking parlance, such a failure path is called a *counterexample* to the desired safety property. Counterexamples for branching time logics like CTL [3] have the shape of trees of paths [4]. Counterexamples may later be used in locating the cause of a property violation, see, for instance, [5], [6]. While the computationally most efficient way to perform explicit-state model checking is to base it on depth-first search [2], in order to obtain short, and therefore easy to comprehend, counterexamples, the use of search techniques such as *Breadth-First Search* (BFS) or *Directed Model Checking* (DMC) [7], which relies on *directed explicit state-space search* [8], [9], have been proposed in model checking.

• *The authors are with the Department of Computer and Information Science, University of Konstanz, Universitätsstr. 10, PO Box D67, 78457 Konstanz, Germany.*
E-mail: {husain.aljazzar, Stefan.Leue}@uni-konstanz.de.

Manuscript received 27 Jan. 2008; revised 18 Feb. 2009; accepted 11 Aug. 2009; published online 17 Sept. 2009.
Recommended for acceptance by R. Cleaveland.
For information on obtaining reprints of this article, please send e-mail to: tse@computer.org, and reference IEEECS Log Number TSE-2008-01-0046.
Digital Object Identifier no. 10.1109/TSE.2009.57.

In performance and dependability analysis, one is not just interested in detecting functional errors in the design of the system, but also in quantitatively reasoning about a given system with respect to its performance and dependability characteristics. For this kind of analysis, systems are usually modeled as stochastic processes, e.g., as *discrete-time Markov chains* (DTMCs) or *continuous-time Markov chains* (CTMCs) [10]. These models describe how the system probabilistically changes from one state to another as time passes. Stochastic model checking is a technique to analyze stochastic models with respect to performance and dependability properties. These properties are commonly expressed in a stochastic temporal logic, such as, for instance, the *Probabilistic Computation Tree Logic* (PCTL) [11] or the *Continuous Stochastic Logic* (CSL) introduced in [12] and extended in [13]. Prominent stochastic model checkers such as PRISM [14] and MRMC [15] apply efficient numerical algorithms to compute the probabilities which determine the satisfaction or violation of a given property. Although these tools achieve a high degree of numerical result accuracy, they are unable to provide the user with diagnostic information, in particular, counterexamples. This is a result of the numerical nature of the employed algorithms. Some model checkers such as YMER [16] and APMC [17] verify properties using statistical execution sampling. They can be adopted to construct counterexamples from the statistically generated execution samples. However, it is not possible to focus on certain counterexamples which are informative for debugging purposes, e.g., which are small while carrying high probabilities.

The unavailability of informative counterexamples makes debugging very difficult, which constrains the practical usefulness of current stochastic model checking tools. In this paper, we propose a very scalable method for generating counterexamples for model checking of DTMCs and

0098-5589/10/$26.00 © 2010 IEEE Published by the IEEE Computer Society

CTMCs. The tenet of our approach will be the use of directed explicit state-space search on the state transition graph of the stochastic model in order to determine *diagnostic paths* which are meaningful in the fault localization process.

In contrast to the functional setting, in the stochastic context, we are faced with the following challenges when computing diagnostic paths, or counterexamples:

- First, one diagnostic path is, in general, not enough to provide meaningful information for explaining why some PCTL or CSL property is violated. More commonly, the accumulated probability of a larger set of paths jointly contributes to violating the given probability bound. Therefore, a counterexample determination method for probabilistic systems needs to compute such a set of diagnostic paths instead of a single one, as in the context of functional model checking.
- Second, the provided counterexamples have to convey crucial information in order to be useful for analysis and debugging purposes. Indicative of the significance of a diagnostic path is not its length, as in functional model checking [7], but its probability. Hence, our algorithms should hunt for the most probable diagnostic paths. It is hence necessary to find a quality measure based on the probability of paths as well as heuristic functions based on this measure that will guide the search along paths with high probabilities. As a consequence of this search strategy, the provided counterexamples will be relatively small. The fact that the counterexample is small while carrying a high probability greatly facilitates system analysis and debugging.

We introduce a directed explicit state-space search strategy called *eXtended Best-First* (XBF) which addresses these challanges. XBF explores the state transition graph of CTMCs and DTMCs and incrementally determines a subgraph of the state transition graph which covers the most probable diagnostic paths. The selected diagnostic subgraph has two essential properties. First, it proves the violation of the given property. Second, it is not possible to repair the model without considering this subgraph. XBF-based algorithms can be guided by heuristics to amplify the power of the search in terms of computational effort and counterexample quality.

1.2 Related and Precursory Work

Various approaches to stochastic model checking of DTMCs and CTMCs are known in the literature [18], [19], [11], [20], [21], [22], [23], [13]. As mentioned above, none of these approaches directly produces counterexamples. Our approach can be used in combination with most of the standard stochastic model checking approaches in order to add debugging support.

Our method exploits the power of directed search strategies which have been developed to solve, among others, graph search and optimization problems [8], [9]. In his seminal monograph on heuristics [9], Pearl gives a comprehensive overview of a set of general-purpose problem solving strategies like *Best First (BF)*. The monograph also presents a variety of specialized directed search algorithms such as A^* and Z^*. An approach on how to apply directed search algorithms to functional model checking, especially for the generation of counterexamples, has been presented in [7]. This approach became known as Directed (Explicit-State) Model Checking (DMC).

Counterexample generation in the context of stochastic model checking has recently been the subject of intensive research [24], [25], [26], [27], [28], [29]. In [24], [25], we presented an approach based on directed explicit-state search for counterexample generation for DTMCs and CTMCs against probablistic time-bounded reachability properties. The work presented in this paper extends this precursory work to completely cover PCTL and CSL logics.

Our approach bears some similarity with the approach presented in [26], [27]. In [26], the authors propose the use of algorithms for finding the k-shortest paths in graphs to compute the most probable diagnostic paths. Using this method, it is possible to compute counterexamples for PCTL model checking of DTMCs. An extension of this approach to deal with CTMCs is presented in [27]. This approach differs from ours mainly in two points. First, our method selects a diagnostic subgraph of the state transition graph, while the approach from [26], [27] enumerates diagnostic paths individually. It is therefore able to faithfully record the effect of every selected diagnostic path on the probability of the currently selected counterexample, whereas our method needs the help of a stochastic model checker to precisely compute the counterexample probability. Second, our approach is on the fly and uses directed search, whereas the approach from [26], [27] requires the full state space to be generated and stored in main memory. As a consequence, our approach has advantages when large models are analyzed under tight memory constraints. Experiments show that our approach significantly outperforms the approach from [26], [27] with respect to both runtime and memory consumption without detriment to the quality of the counterexamples. Of course, we included in the computational costs of our method the effort to compute the precise counterexample probability.

For DTMCs, Han et al. [26] present an algorithm to provide counterexamples in a more compact representation as regular expressions.

Counterexample generation is the starting point of a few applications. For instance, works on counterexample-guided abstraction refinement (CEGAR) in stochastic model checking like [30] can benefit out of this work. In [31], we demonstrate how to use visualization techniques in order to support the human user in analyzing counterexamples to localize causal factors of errors.

1.3 Structure of the Paper

We present fundamental concepts of stochastic model checking in Section 2. We introduce there also the notion of a counterexample. In Section 3, we review directed search algorithms. We also introduce our search strategy XBF which we have developed for the purpose of counterexample generation. The application of XBF to our problem setting is presented in Section 4. Section 5 finally presents an experimental evaluation of our method by applying it to a number of case studies. We conclude in Section 6.

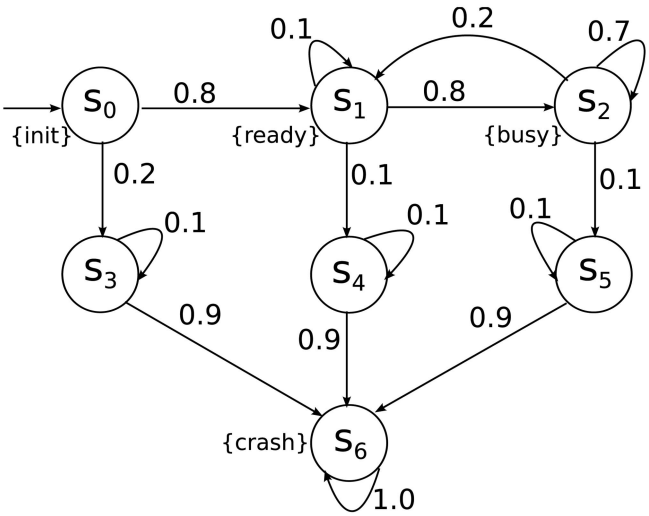

Fig. 1. An example of a DTMC.

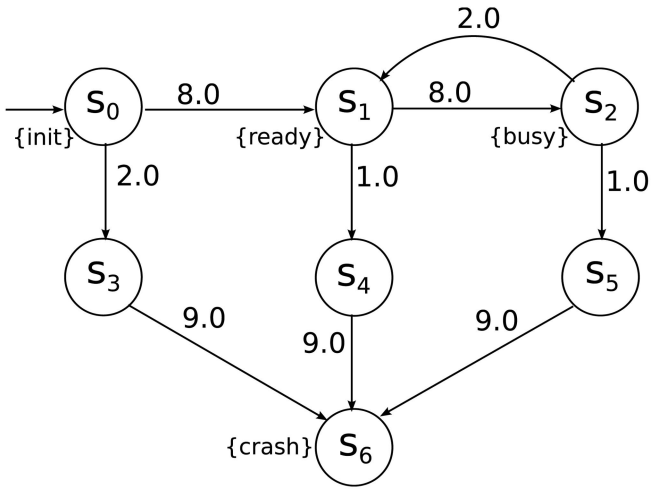

Fig. 2. An example of a CTMC.

2 Stochastic Model Checking

2.1 Markov Chains

System dependability and performance models are often represented by variants of Markov chains. They describe the system behavior as a stochastic process in which system transitions are labeled with probability and time-consumption values. The simplest kind of Markov chains used in system modeling are *discrete-time Markov chains*. A DTMC can be considered as a probabilistic transition system consisting of states and transitions between them. In DTMCs, time is modeled in a discrete way by assuming that the system fires exactly one transition at every discrete-time tick. Each transition is labeled with a numerical value referred to as transition probability. It indicates the probability of firing this transition as the next step of the system if the system is in the origin state of the transition. Formally, a DTMC is defined as follows:

Definition 1. *A labeled* discrete-time Markov chain $\mathcal{D}$ *is a tuple* $(\mathbf{S}, \hat{s}, \mathbf{P}, \mathbf{L})$, *where* $\mathbf{S}$ *is a finite set of states,* $\hat{s} \in \mathbf{S}$ *is the initial state,* $\mathbf{P} : \mathbf{S} \times \mathbf{S} \longrightarrow [0, 1]$ *is a transition probability matrix, satisfying that, for each state* s, $\sum_{s' \in S} \mathbf{P}(s, s') = 1$, *and* $\mathbf{L} : \mathbf{S} \longrightarrow 2^{\mathrm{AP}}$ *is a labeling function which assigns each state a subset of the set of atomic propositions* AP.

For any state s, we interpret $\mathbf{L}(s)$ as the set of valid propositions in the state. For each pair of states s and s', $\mathbf{P}(s, s')$ gives the probability to move from s to s'. A move from s to s' is possible if and only if $\mathbf{P}(s, s') > 0$ holds. In this case, we call (s, s') a *transition*. $succ(s)$ is the set of successor states of s, i.e., $succ(s) = \{\ s' \in \mathbf{S} \mid \mathbf{P}(s, s') > 0\ \}$. A state s is called absorbing if $\mathbf{P}(s, s) = 1$, and consequently, $\mathbf{P}(s, s') = 0$ for all other states $s' \neq s$.

Because of their conceptual simplicity, DTMCs are widely used in the modeling and analysis of stochastic systems. If a more realistic dense modeling of the time passing in states is required, then *continuous-time Markov chains* can be used. While each transition of a DTMC corresponds to a discrete-time step, in a CTMC, transitions occur in dense real time. In a CTMC, each transition is labeled by a rate which represents the probability of a transition from some state s to some state s' within t time units after being enabled. The duration t is determined by a random variable that follows a negative exponential distribution which uses the rate label as a parameter. A CTMC is formally defined as follows:

Definition 2. *A labeled* continuous-time Markov chain $\mathcal{C}$ *is a tuple* $(\mathbf{S}, \hat{s}, \mathbf{R}, \mathbf{L})$, *where* $\mathbf{S}$ *is a finite set of states,* $\hat{s} \in \mathbf{S}$ *is the initial state,* $\mathbf{R} : \mathbf{S} \times \mathbf{S} \longrightarrow \mathbb{R}_{\geq 0}$ *is a transition rate matrix, and* $\mathbf{L} : \mathbf{S} \longrightarrow 2^{\mathrm{AP}}$ *is a labeling function as in Definition 1.*

The transition probability matrix $\mathbf{P}$, which we defined for DTMCs, is replaced by a transition rate matrix $\mathbf{R}$. For any pair of states s and s', $\mathbf{R}(s, s')$ is the time rate for moving from s to s'. More precisely, $\mathbf{R}(s, s')$ is the parameter of a negative exponential distribution describing the probability to move from s to s' within a certain time interval. A move from s to s' is only possible if $\mathbf{R}(s, s') > 0$. In this case, we call (s, s') a *transition*. $succ(s)$ is the set of successor states of s, i.e., $succ(s) = \{s' \in \mathbf{S} \mid \mathbf{R}(s, s') > 0\}$. We use $\Lambda(s) = \sum_{s' \in \mathbf{S}} \mathbf{R}(s, s')$ to denote the total rate to leave s to any successor state. If $\Lambda(s) = 0$, then s is called absorbing.

Example 1. Fig. 1 illustrates a DTMC modeling a system which becomes ready (state s_1) after finishing some initialization routine (state s_0). When the system is ready, it may receive a task to process (state s_2) and return to be ready after finishing to process the task. The system may fail at any state (states s_3, s_4, and s_5), causing a system crash (state s_6). Fig. 2 illustrates a CTMC modeling a system which is very similar to the one modeled by the DTMC given in Fig. 1. The only difference here is that the time is interpreted to be continuous.

2.2 Paths and Probability Measures

A *path* is a concrete execution of the system, i.e., it encompasses a sequence of state transitions. Since systems considered in the domain of stochastic model checking are usually reactive systems, paths are assumed to be infinite. We often need to refer to finite-path prefixes. We use the

notions of *infinite path*, or simply *path*, for a full path and *finite path* for a finite-path prefix. Formally, let $\mathcal{D} = (\mathbf{S}, \hat{s}, \mathbf{P}, \mathbf{L})$ be a DTMC, then:

Definition 3. *A (*infinite*) path through $\mathcal{D}$ is an infinite sequence $s_0, s_1, s_2 \ldots$ with, for all i, $s_i \in S$ and $\mathbf{P}(s_i, s_{i+1}) > 0$. A* finite path *is a* finite prefix *of an infinite path.*

For a finite or an infinite path $\sigma = s_0, s_1, \ldots$, we use $len(\sigma)$ to denote the length of σ determined by the number of states. Note that for an infinite path σ, $len(\sigma)$ is equal to ∞. For a natural number k such that $0 \leq k < len(\sigma)$, $\sigma[k]$ refers to the kth state of σ, namely, s_k. With $\sigma^{(k)}$, we denote the kth prefix of σ, i.e., the prefix of σ consisting of the first k transitions, namely, $s_0, \ldots, s_k$. The term $first(\sigma)$ refers to the first state in σ. If σ is finite, then $last(\sigma)$ denotes the last state of σ. We use $Paths^{\mathcal{D}}$ to denote the set of all infinite paths in $\mathcal{D}$. For any state s, $Paths^{\mathcal{D}}(s)$ refers to the sets of infinite paths which start at s.

For a DTMC $\mathcal{D} = (\mathbf{S}, \hat{s}, \mathbf{P}, \mathbf{L})$ and a state $s_0 \in \mathbf{S}$, the probability of paths originating at s_0 is measurable by a probability measure $Pr^{\mathcal{D}}_{s_0}$. The underlying σ-algebra is formed by the cylinder sets which are induced by finite paths in $\mathcal{D}$ starting at s_0 [11]. Each finite path $s_0, \ldots, s_n$ induces a cylinder set $cyl(s_0, \ldots, s_n) = \{\sigma \in Paths^{\mathcal{D}}(s_0) \mid \sigma^{(n)} = s_0, \ldots, s_n\}$. The probability of this cylinder set is defined as follows:

$$Pr^{\mathcal{D}}_{s_0}(cyl(s_0, \ldots, s_n)) = \prod_{i=0}^{n-1} \mathbf{P}(s_i, s_{i+1}). \tag{1}$$

By $Pr^{\mathcal{D}}$, we denote the probability measure of paths in $\mathcal{D}$ starting at the initial state, i.e., $Pr^{\mathcal{D}} = Pr^{\mathcal{D}}_{\hat{s}}$.

For a CTMC $\mathcal{C} = (\mathbf{S}, \hat{s}, \mathbf{R}, \mathbf{L})$, a path through $\mathcal{C}$ comprises the time delays in each state along the path in addition to the sequence of state transitions:

Definition 4. *A (*infinite*) path through $\mathcal{C}$ is an infinite alternating sequence $s_0, t_0, s_1, t_1, s_2 \ldots$ with, for all i, $s_i \in S$, $\mathbf{R}(s_i, s_{i+1}) > 0$, and $t_i \in \mathbb{R}_{\geq 0}$. A* finite path *is a finite prefix of an infinite path ending with a state.*

All notions defined above for paths in DTMCs are defined in the same way for paths in CTMCs. Further, we define $\sigma@t$ for a finite or infinite path σ in a CTMC and a time point t as the state in which the system resides at time point t when it is executing the path σ. Formally, $\sigma@t$ is equal to s_k, where k is the smallest index with $t \leq \sum_{i=0}^{k} t_i$. The probability of paths in $\mathcal{C}$ starting at some state $s_0 \in \mathbf{S}$ is measurable. We define the probability measure $Pr^{\mathcal{C}}_{s_0}$ on $Paths^{\mathcal{C}}(s_0)$ following [13]: Let $s_0, \ldots, s_n$ be a sequence of states of $\mathcal{C}$ such that $\mathbf{R}(s_i, s_{i+1}) > 0$ for all $i \in \{0, \ldots, n-1\}$. Further, let $I_0, \ldots, I_{n-1}$ be a sequence of nonempty intervals in $\mathbb{R}_{\geq 0}$. Let $cyl(s_0, I_0, \ldots, I_{n-1}, s_n)$ denote the cylinder set consisting of all paths of the form $\sigma = s_0, t_0, \ldots, t_{n-1}, s_n \ldots$ with $t_i \in I_i$ for all $i \in \{0, \ldots, n-1\}$. The measure $Pr^{\mathcal{C}}_{s_0}$ is defined on the smallest σ-algebra on $Paths^{\mathcal{C}}(s_0)$ formed by all cylinder sets for all state sequences starting at s_0 and sequences of time intervals as described above. Intuitively, $Pr^{\mathcal{C}}_{s_0}(cyl(s_0, I_0, \ldots, I_{n-1}, s_n))$ gives the probability of $\mathcal{C}$ to visit the states $s_0, \ldots, s_n$ in that sequence while staying in state s_i a period of time $t \in I_i$ for $0 \leq i < n$. $Pr^{\mathcal{C}}_{s_0}$ is formally defined by the following induction, with $Pr^{\mathcal{C}}_{s_0}(cyl(s_0)) = 1$, as:

$$\begin{aligned} &Pr^{\mathcal{C}}_{s_0}(cyl(s_0, I_0, \ldots, I_{n-1}, s_n, I_n, s_{n+1})) \\ &\quad = Pr^{\mathcal{C}}_{s_0}(cyl(s_0, I_0, \ldots, I_{n-1}, s_n)) \cdot \frac{\mathbf{R}(s_n, s_{n+1})}{\Lambda(s_n)} \\ &\quad \cdot \int_{I_n} \Lambda(s_n) \cdot e^{-\Lambda(s_n) \cdot t} \cdot dt. \end{aligned} \tag{2}$$

The term $\int_{I_n} \Lambda(s_n) \cdot e^{-\Lambda(s_n) \cdot t} \cdot dt$ describes the probability of firing any transition outgoing from state s_n in the time interval I_n. The term $\frac{\mathbf{R}(s_n, s_{n+1})}{\Lambda(s_n)}$ is called the *branching probability* and indicates the probability to choose the transition (s_n, s_{n+1}) to be fired. Altogether, the term $\frac{\mathbf{R}(s_n, s_{n+1})}{\Lambda(s_n)} \cdot \int_{I_n} \Lambda(s_n) \cdot e^{-\Lambda(s_n) \cdot t} \cdot dt$ gives the probability to fire the transition (s_n, s_{n+1}) within the interval I_n. Like for DTMCs, we use the abbreviation $Pr^{\mathcal{C}}$ for the probability measure of paths in $\mathcal{C}$ starting at the initial state, i.e., $Pr^{\mathcal{C}} = Pr^{\mathcal{C}}_{\hat{s}}$.

2.3 Stochastic Temporal Logics

In performance and dependability analysis, system requirements are commonly expressed in a stochastic temporal logic. The *Probabilistic Computation Tree Logic* (PCTL) was introduced the first time by Hansson and Jonsson in 1994 as a stochastic extension of the CTL [11]. The syntax of a PCTL formula is defined as follows:

$$\phi \ := \ true \mid false \mid a \mid \neg\, \phi \mid \phi \ \vee \ \phi \mid \phi \ \wedge \ \phi \mid \mathcal{P}_{\bowtie p}(\varphi),$$

where $\bowtie$ is one of the operators $<$, $\leq$, $>$, or $\geq$, $p \in [0,\ 1]$, and φ is a path formula the syntax of which is defined as:

$$\varphi \ := \ \phi \ U \ \phi \mid \phi \ U^{\leq h} \phi,$$

where $h \in \mathbb{N}$.

The PCTL semantics is interpreted over a DTMC $\mathcal{D} = (\mathbf{S}, \hat{s}, \mathbf{P}, \mathbf{L})$ by the satisfaction relation $\models_{\mathcal{D}}$. The definition of $\models_{\mathcal{D}}$ is straightforward for almost all PCTL formulas. The interesting cases are the $\mathcal{P}$- and bounded Until-formulas, for which $\models_{\mathcal{D}}$ is defined as follows: Let $s \in \mathbf{S}$ be a state and σ be an infinite path in $\mathcal{D}$

$$s \models_{\mathcal{D}} \mathcal{P}_{\bowtie p}(\varphi) \quad \text{iff } Pr^{\mathcal{D}}_{s}(\{\sigma \in Paths^{\mathcal{D}}(s) \mid \sigma \models_{\mathcal{D}} \varphi\}) \ \bowtie \ p$$

$$\sigma \models_{\mathcal{D}} (\phi_1 \ U^{\leq h} \ \phi_2) \quad \text{iff } \exists i \leq h \text{ such that } \sigma[i] \models_{\mathcal{D}} \phi_2 \text{ and } \forall j < i \ : \ \sigma[j] \models_{\mathcal{D}} \phi_1.$$

For convenience, we abbreviate the term $Pr^{\mathcal{D}}_{s}(\{\sigma \in Paths^{\mathcal{D}}(s) \mid \sigma \models_{\mathcal{D}} \varphi\})$ by $Pr^{\mathcal{D}}_{s}(\varphi)$. Note that $Pr^{\mathcal{D}}_{s}(\varphi)$ is well defined for any PCTL path formula φ (see, for example, [32]).

For specifying properties over CTMCs, we use *Continuous Stochastic Logic* (CSL) [12], [13]. The syntax of CSL is the same as PCTL except that the time bound in a bounded *Until*-formula is a nonnegative real number. CSL also allows us to specify the time constraint as an interval. The semantics of a CSL formula over a CTMC $\mathcal{C} = (\mathbf{S}, \hat{s}, \mathbf{R}, \mathbf{L})$ is defined by the satisfaction relation $\models_{\mathcal{C}}$ in a similar way like PCTL over a DTMC. The only difference is in the semantics

of the bounded *Until*- operator. For an infinite path σ in $\mathcal{C}$ and a time interval I:

$$\sigma \models_{\mathcal{C}} (\phi_1 \; U^I \; \phi_2) \quad \text{iff } \exists x \in I \text{ such that } \sigma@x \models_{\mathcal{D}} \phi_2 \text{ and } \forall y < x \; : \; \sigma@y \models_{\mathcal{D}} \phi_1.$$

Throughout the paper, we use the term *time-bounded* Until-formula to refer to a bounded Until-formula either in PCTL or CSL. Notice that, in the case of a DTMC, the notion of time is identical to the notion hops. Each hop (state transition) in a DTMC corresponds to a time unit.

The procedure of model checking of PCTL or CSL differs from that of CTL in checking the $\mathcal{P}$-operator which requires the computation of probabilities. In order to determine the states of a Markov chain $\mathcal{M}$ which satisfy the formula $\mathcal{P}_{\bowtie p}(\varphi)$, for some path formula φ, we need to compute for each state s the probability of all paths in $\mathcal{M}$ which start at s and satisfy φ. Formally, we have to compute $Pr_s^{\mathbf{M}}(\varphi)$. The set $Sat(\mathcal{P}_{\bowtie p}(\varphi))$ contains then all states where $Pr_s^{\mathcal{M}}(\varphi) \bowtie p$ holds. This is usually done using numerical algorithms such as solving a large linear equation system. The numerical nature of these algorithms is responsible for the inability of stochastic model checkers to provide diagnostic information in case the model violates the property.

2.4 Counterexamples

In the context of functional model checking, the notion of a *counterexample* is widely known. For instance, in the case of a safety property violation, a counterexample consists of a (single) *failure path* from the initial state of the system to a property violating state. In this section, we derive the definition of the counterexample notion in the context of PCTL and CSL model checking. Let $\mathcal{M}$ be a Markov chain, i.e., a DTMC or a CTMC. Let ϕ be a PCTL or CSL property which is violated on $\mathcal{M}$. The notion of a counterexample for nonprobabilistic formula is straightforward and we focus on the definition of counterexamples for PCTL and CSL formulas of the form $\mathcal{P}_{\bowtie p}(\varphi)$. We first introduce the notion of a *diagnostic path* which we will need in the definition of counterexamples.

Definition 5. *In a Markov chain $\mathcal{M}$, a (finite) diagnostic path of $\mathcal{P}_{\bowtie p}(\varphi)$ is a (finite) path in $\mathcal{M}$ starting from the initial state that satisfies the path formula φ.*

In both PCTL and CSL, φ is either an Until or a bounded Until-formula. Any (infinite) diagnostic path of $\mathcal{P}_{\bowtie p}(\varphi)$ contains a finite prefix which satisfies φ. Diagnostic paths in our setting are therefore always finite paths.

We define the notion of a counterexample first for *upper-bounded* $\mathcal{P}$-formulas, i.e., formulas of the form $\mathcal{P}_{<p}(\varphi)$ or $\mathcal{P}_{\leq p}(\varphi)$.

Definition 6. *Let $\mathcal{M}$ be a DTMC or CTMC violating an upper-bounded formula $\mathcal{P}_{\bowtie p}(\varphi)$. A* counterexample *for $\mathcal{P}_{\bowtie p}(\varphi)$ is a measurable subset $X \subseteq \{\sigma \in Paths^{\mathcal{M}}(\hat{s}) \mid \sigma \models_{\mathcal{M}} \varphi\}$ of diagnostic paths such that $Pr^{\mathcal{M}}(X) \bowtie p$ does not hold.*

For lower-bounded $\mathcal{P}$-formulas, a direct definition of counterexamples does not make sense. For example, for a lower-bounded formula $\mathcal{P}_{\geq p}(\varphi)$, a measurable set X of diagnostic paths with $Pr^{\mathcal{M}}(X) < p$ does not carry useful information since even the empty set trivially fulfills this condition. We propose handling lower-bounded formulas by giving a condition that the accumulated probability of paths which do not satisfy φ is too high. In the remainder, we mainly focus on upper-bounded formulas, but discuss how our method can be adopted to deal with lower-bounded formulas in Section 4.6. In the following paragraphs (Sections 2.4.1-2.4.4), we explain some notions and basics related to counterexamples.

2.4.1 State Transition Graph (STG)

The state space of a Markov chain $\mathcal{M}$ can be viewed as a directed graph called *State Transition Graph* $\mathrm{STG}_{\mathcal{M}}$ (or simply STG if $\mathcal{M}$ is clear from the context), where the nodes represent the states of the model and each state transition (s, s') is represented by a directed edge from s to s'. The counterexample generation problem can be interpreted as a search problem on the state transition graph $\mathrm{STG}_{\mathcal{M}}$. Let $\mathcal{P}_{\bowtie p}(\varphi)$ be the property for which we are generating a counterexample. The path formula φ is an Until or bounded Until-formula, i.e., $\varphi = (\phi_1 \; U \; \phi_2)$ or $\varphi = (\phi_1 \; U^{\leq t} \; \phi_2)$. Only those paths which reach a ϕ_2 state without traversing a non-ϕ_1 state can be diagnostic paths. We hence search for ϕ_2 states, which we call *target states*. All non-ϕ_1 states will be considered absorbing since the paths leaving them will not contribute to the counterexample. Paths in an $\mathrm{STG}_{\mathcal{M}}$ neither carry stochastic nor time information. To avoid confusion, we refer to paths in an $\mathrm{STG}_{\mathcal{M}}$ as *traces*. We call a finite trace in $\mathrm{STG}_{\mathcal{M}}$ from $\hat{s}$ to a target state a *diagnostic trace*. On the other hand, there is a strong relationship between traces in $\mathrm{STG}_{\mathcal{M}}$ and paths in $\mathcal{M}$ that we shall exploit later to generate counterexamples.

2.4.2 Informative and Comprehensible Counterexamples

For a Markov chain $\mathcal{M}$ and an upper-bounded property $\mathcal{P}_{\bowtie p}(\varphi)$, a counterexample is a measurable set of diagnostic paths. A diagnostic trace in $\mathrm{STG}_{\mathcal{M}}$ represents a set of diagnostic paths in $\mathcal{M}$ which mimic this trace. If the accumulated probability of the set of diagnostic paths represented by a particular diagnostic trace is high, then this diagnostic trace represents system behavior that most significantly contributes to the property violation. Hence, a counterexample which includes such highly probable diagnostic traces is more useful for debugging than others. Moreover, such a counterexample exceeds the probability upper bound with fewer diagnostic traces than those counterexamples which only include diagnostic traces with low probabilities. In other words, such a counterexample tends to be smaller than a counterexample having the same probability without including highly probable diagnostic trace. In order to provide counterexamples that 1) are most informative in terms of the property violating behavior and 2) are easy to comprehend and manage, we strive in this paper to compute counterexamples which include diagnostic traces with the highest possible probability.

2.4.3 Counterexample Generation as a k-Shortest-Paths-Problem

The problem of counterexample generation can be viewed as an instance of the *k-Shortest-Paths problem* (KSP) [33], [34].

This is the problem of finding the k shortest paths from a given start node to a given target node in a weighted directed graph for an arbitrary number k. To provide a counterexample, we search on the state transition graph $\mathrm{STG}_{\mathcal{M}}$ for most probable diagnostic traces. We continue to collect diagnostic traces until their accumulated probability exceeds the probability bound p. Since we do not know at the beginning how many traces we will need to exceed p, we need KSP algorithms which enumerate the most probable diagnostic traces one-by-one until "enough" diagnostic traces are found. The best known examples for such algorithms are the algorithm of Eppstein [34] and the Recursive Enumeration Algorithm REA [35].

2.4.4 Counterexamples as Diagnostic Subgraphs

Instead of explicitly computing a set of diagnostic traces using KSP algorithms, our method provides a counterexample in a compact form by computing a subgraph of the $\mathrm{STG}_{\mathcal{M}}$ that we refer to as a *diagnostic subgraph*. It covers a set of diagnostic paths carrying the probability required to form a counterexample.

Definition 7. *For a Markov chain $\mathcal{M}$ and an upper-bounded $\mathcal{P}$-formula, a* diagnostic subgraph *is a subgraph of $\mathrm{STG}_{\mathcal{M}}$ with the following features:*

1. *It is either empty or it contains $\hat{s}$.*
2. *Each state of it is either a target state or a target state is reachable from it.*

A diagnostic subgraph is called *complete* iff it contains all states and transitions which lead to a target state.

A diagnostic subgraph for an upper-bounded property $\mathcal{P}_{\bowtie p}(\varphi)$ comprises a set X of diagnostic paths. This is the set $X \subseteq \{\sigma \in Paths^{\mathcal{M}}(\hat{s}) \mid \sigma \models_{\mathcal{M}} \varphi\}$ which comprises only those diagnostic paths which are built of states and transitions included in the diagnostic subgraph. This set is measurable which means that its accumulated probability $Pr^{\mathcal{M}}(X)$ is well-defined as we show now. We turn the subgraph into a *diagnostic Markov chain* as follows: For each state s contained in the diagnostic subgraph, we replace all outgoing transitions of s which are not included in the diagnostic subgraph by a transition to an extra absorbing state $sink$ that is substituting the remaining part of the whole Markov chain. The sought probability $Pr^{\mathcal{M}}(X)$ is equivalent to the result of checking the given property $\mathcal{P}_{\bowtie p}(\varphi)$ on the obtained diagnostic Markov chain. If $Pr^{\mathcal{M}}(X)$ violates the upper-bound $\bowtie p$, then X is a counterexample. Note that X is fully characterized by the corresponding diagnostic subgraph. In this paper, we often refer to diagnostic subgraphs as counterexamples.

3 Directed Explicit State Search

3.1 Best-First Search (BF)

Most of the prevalent directed search algorithms are based on a common optimizing search strategy called *Best-First (BF)* [9]. This strategy explores the given model on-the-fly which means that the set $\mathbf{S}$ does not have to be given explicitly at the beginning. It is sufficient to give the initial state $\hat{s} \in \mathbf{S}$ and a successor function $succ : \mathbf{S} \to 2^{\mathbf{S}}$. We will be dealing exclusively with *locally finite graphs* for which every state has got a finite number of successors. BF uses a set called closed to collect information about all explored states which have been *expanded*, which means that their successors have been generated. BF also uses a set called open to store all explored states which have not been expanded yet. The set open is organized as a priority queue which is sorted according to the "quality" of states, i.e., the promise of a state to lead to the sought solution. This quality is estimated numerically by an *evaluation function* f. f usually depends on local state information as well as information gathered by the search so far. In addition, f may depend on the specification of the target states as well as further knowledge about the problem domain. This knowledge is commonly encoded in a *heuristic function* h which is used by the evaluation function f. If such a heuristic function is used in the search, we call the resulting algorithm *heuristics-guided*, *informed*, or *directed*.

The core step in each BF iteration is the expansion of the state s from open with the best f value. Each successor state s' is labeled with a *parent link* to its parent s. If s' is a target state, the algorithm terminates with the path constructed by backtracking the parent links from s' up to the initial state $\hat{s}$. Otherwise, s is put into open. If s' has been explored before, i.e., it is contained in open or closed, then the better path to s' is taken and the other is discarded. Consequently, each explored state with the exception of the initial state has exactly one parent. These parent links form a tree which is called the *search tree*. For any state s, the unique path in the search tree from the root $\hat{s}$ to s is called the *solution base* of s.

BF is a complete search algorithm [9]. Many heuristic search strategies including BF have a worst-case exponential complexity [36], [37], [38] while showing a good average case performance. We will therefore not focus on complexity analysis of heuristic search algorithms in this paper.

BF is instantiated to concrete algorithms by specifying a concrete evaluation function f, or by requiring f to satisfy particular conditions. For example, BF becomes Z if the evaluation function f is computed recursively [9]. The recursive nature of the evaluation function allows shared computation and selective updating which are essential for efficient search, in particular, on large graphs [9]. A* is a specialization of Z which finds the shortest path where the path length is the sum of the transition weights.

3.2 Extended Best-First Search (XBF)

Applying any algorithm which is based on BF to $\mathrm{STG}_{\mathcal{M}}$ would deliver a single diagnostic trace as a solution. Because we are interested in selecting a diagnostic subgraph, we extended the search strategy BF to a new one called *eXtended Best-First* (XBF) by the following three modifications:

1. For each state, we record all parent states which we find during the search. We replace the single parent link used in BF by a list parents containing a link to each parent detected by the search.
2. In addition to open and closed, XBF maintains a graph Diag which contains, at any point in the search, the currently selected diagnostic subgraph.
3. XBF does not terminate when it finds the first target state. It continues the search for further target states until the whole state space has been processed, or termination is explicitly requested.

The pseudocode of XBF is given in Algorithm 1. The **while**-loop starting at code line 3 will run until open is empty or termination is explicitly requested. The termination can be requested using an arbitrary condition which we refer to as the *explicit termination condition*. In the context of counterexample generation, this condition is typically that the selected diagnostic subgraph has a sufficient amount of probability. In each iteration of the **while**-loop, the open state s with the optimal f-value is expanded. For each successor s' of s, a pointer to the parent s is added into the parents list of s' and $f(s')$ is computed. The **if**-statement starting at Line 8 checks whether s' is a target state or it has been already added to Diag. This check detects new increments of the diagnostic subgraph. If s' is a target state or it belongs to Diag, i.e., it is known, from prior search, that a target state is reachable from s', then the whole part, of the explored portion of the state transition graph, which leads to s' is added to Diag at Line 9. The code line 10 is useful to gather any information needed for the explicit termination condition. If s' is a new state, i.e., it has not been visited before, then it is added into open at Line 12. Otherwise, i.e., if s' already exists in closed or open, then we compare the newly computed $f(s')$ with the old one. If the new value of $f(s')$ is better than the old one, then we set the $f(s')$ to the new value. In this case, if s' is closed, then it is reopened, i.e., it is moved from closed to open (see Line 16). Finally, when the **while**-loop terminates, the algorithm returns the selected diagnostic subgraph Diag as a result.

Algorithm 1: XBF

```
   Data: A state transition graph given by its initial state ŝ ∈ S
         and its successor function succ
   Result: A diagnostic subgraph
 1 open ← an empty priority queue; closed ← an empty hash
   table; Diag ← an empty graph
 2 Insert ŝ into open
 3 while open is not empty and termination is not requested do
 4     Remove from open and place on closed the state s for which
       f is optimal.
 5     foreach s' ∈ succ(s) do
 6         Add a pointer to s into parents of s'.
 7         Compute f(s').
 8         if s' is a target state or s' is in Diag then
 9             Backtrack all parent links from s' up to the ŝ
               adding all touched states and transitions into Diag.
10             Signalize "update of Diag".
11         if s' is not already in open or closed then
12             Insert s' into open.
13         else
14             if The newly computed f(s') is better than the old
               value then
15                 Replace the old value of f(s') by the new one.
16                 if s' is in closed then Reopen s' (move it to
                   open).
17 return Diag and exit.
```

Note that every state may have several parents representing several solution bases, i.e., traces leading from $\hat{s}$ to the state. With the "solution base" of a state, we always mean the optimal one, i.e., the one causing the best f-value.

3.3 Properties of XBF

3.3.1 Correctness

The proof of correctness of XBF consists in proving that Diag is really a diagnostic subgraph. We have to prove that Diag is either empty or it contains $\hat{s}$ and every state in Diag leads to a target state. The subgraph Diag is constructed and incremented at code lines 8 and 9. If s' is a target state, then it is trivial that all ancestors of s' lead to a target state, namely, s', and hence, can be added to Diag. Among others, the state $\hat{s}$ will be added into Diag since it is an ancestor of s'. Moreover, if s' is in Diag, then we know that s' leads to a target state. As a consequence, all ancestors of s' lead to that target state and they can all be added into Diag. This means that any state or transition will be added to Diag only if it is a target state or if it leads to one. If no target state is reachable, then Diag will stay empty. In conclusion, Diag is a diagnostic subgraph.

3.3.2 Termination and Completeness

For proving the termination and completeness of XBF, we need the following observation. In the case that XBF does not reopen any state, it is guaranteed to terminate on a finite graph since it expands a new state in every iteration. XBF reopens a closed state only if it finds a new trace leading to a better f value of the state, or if it encounters the state at shorter depth in the depth-bounded case. In both cases, the new trace must be acyclic, because any cyclic trace causes a worse f than the trace which is obtained by not considering the cycle. Opening a new state or reopening a closed state hence means the detection of a new acyclic trace. The number of acyclic traces in a finite graph is finite. At some point, XBF will detect all acyclic traces and terminate.

If the external termination condition is not enabled, then XBF will run until open is empty. Let (s, s') be an arbitrary transition which belongs to the complete diagnostic subgraph. The queue open cannot turn empty before s is reached and expanded discovering the transition (s, s'). If s' is a target state or it belongs already to Diag, then (s, s') and the whole portion of the state space leading to s will be added to Diag, see Lines 8 and 9 in Algorithm 1. Otherwise, the same argument can be applied to all diagnostic transitions reachable from s'. This means that (s, s') is guaranteed to be added to Diag at some point. We can conclude that any diagnostic transition will be added into Diag at some point in the search which establishes the completeness of XBF.

3.3.3 Optimality

A counterexample is optimal when a diagnostic trace which is included in the diagnostic subgraph is at least as "good" as any diagnostic trace which is not included. XBF cannot guarantee optimality of the computed counterexample. The main reason for that is XBF keeps for every state all traces leading to it in order to be able to construct the diagnostic subgraph. This feature entails that diagnostic traces which are not optimal can be added to the counterexample. However, optimality is not the objective of our approach, and our experiments in Section 5 will show that it computes counterexamples with very high quality.

4 Generation of Counterexamples

In order to generate a counterexample for a Markov chain $\mathbf{M}$ and an upper-bounded property $\mathcal{P}_{\triangleright\!\!<p}(\varphi)$, we perform an XBF search on $\mathrm{STG}_{\mathcal{M}}$. XBF incrementally selects a diagnostic

subgraph of $\mathrm{STG}_{\mathcal{M}}$ until the probability of the diagnostic subgraph either exceeds or reaches p, depending on whether the property requires a strict or a nonstrict bound, respectively. This requires periodically computing the probability of Diag (cf., Line 10 in Algorithm 1) using a stochastic model checker. Doing so each time when Diag is updated would be fairly inefficient. Therefore, we perform this check only when the size of Diag has grown by q percent. In our experiments, $q = 20\%$.

4.1 State Evaluation Function

XBF is guided by a heuristic evaluation function f that we now define. As argued before, we are searching for informative counterexamples that carry high probability. We hence define f so that it guides the search along the most probable diagnostic traces.

A finite trace $R = \langle s_0, \ldots, s_n \rangle$ is a compact presentation of the set $Paths^{\mathcal{M}}(\langle s_0, \ldots, s_n \rangle) := \{\sigma \in Paths^{\mathcal{M}} \mid \forall 0 \leq i \leq n.\ \sigma[i] = s_i\}$ of all paths in $\mathcal{M}$ which mimic R. The set $Paths^{\mathcal{M}}(R)$ is measurable for both CTMCs and DTMCs. In the case of a DTMC $\mathcal{D}$, the trace $R = \langle s_0, \ldots, s_n \rangle$ represents the same set of paths as the finite sequence $s_0, \ldots, s_n$. In other words, $Paths^{\mathcal{D}}(R)$ is just the cylinder set $cyl(s_0, \ldots, s_n)$. The probability of $Paths^{\mathcal{D}}(R)$ is given as follows:

$$\begin{aligned} prob^{\mathcal{D}}(\langle s_0, \ldots, s_n \rangle) &= Pr^{\mathcal{D}}_{s_0}(Paths^{\mathcal{D}}(\langle s_0, \ldots, s_n \rangle)) \\ &= Pr^{\mathcal{D}}_{s_0}(cyl(s_0, \ldots, s_n)) \\ &= \prod_{i=0}^{n-1} \mathbf{P}(s_i, s_{i+1}). \end{aligned}$$

In the case of a CTMC $\mathcal{C}$, we allow the residence time in each state s_i to vary from 0 up to ∞, and hence, $Paths^{\mathcal{C}}(R)$ includes all paths which mimic R independently of any time delays. $Paths^{\mathcal{C}}(R)$ is then equal to the cylinder set $cyl(s_0, [0,\ \infty[, \ldots, [0,\ \infty[, s_n)$. This implies that

$$\begin{aligned} prob^{\mathcal{C}}(\langle s_0, \ldots, s_n \rangle) &:= Pr^{\mathcal{C}}_{s_0}(Paths^{\mathcal{C}}(\langle s_0, \ldots, s_n \rangle)) \\ &= Pr^{\mathcal{C}}_{s_0}(cyl(s_0, [0,\ \infty[, \ldots, [0,\ \infty[, s_n)). \end{aligned}$$

The term $\int_{I_i} \Lambda(s_i) \cdot e^{-\Lambda(s_i) \cdot t} \cdot dt$ from (2) represents the probability of firing any transition outgoing from state s_i after an arbitrary time delay in s_i, which means $I_i = [0,\ \infty[$. This corresponds to a value of 1.0. The probability to fire particularly the transition (s_i, s_{i+1}) after an arbitrary time can therefore be given by the branching probability $\frac{\mathbf{R}(s_i, s_{i+1})}{\Lambda(s_i)}$. By using (2), we obtain the following closed expression:

$$\begin{aligned} prob^{\mathcal{C}}(R) &= Pr^{\mathcal{C}}_{s_0}(cyl(s_0, [0,\ \infty[, \ldots, [0,\ \infty[, s_n)) \\ &= \prod_{i=0}^{n-1} \frac{\mathbf{R}(s_i, s_{i+1})}{\Lambda(s_i)}. \end{aligned}$$

Now we are able to compare traces with respect to their suitability for inclusion into a counterexample. Let R_1 and R_2 be two traces in a Markov chain $\mathcal{M}$. The trace R_1 is considered to be more suitable than R_2 iff $prob^{\mathcal{M}}(R_1) > prob^{\mathcal{M}}(R_2)$. We use this metric in defining the state evaluation function f used by XBF as follows: We aim to define the evaluation function f such that for every state s, $f(s)$ estimates the probability of the optimal diagnostic trace through s. Let R_s be the solution base of s. We define $g(s) := prob^{\mathcal{M}}(R_s)$. In case we possess special knowledge regarding the application domain, we can exploit this to estimate the probability of the optimal trace, denoted here as R^*, from s to a target state. We formulate a heuristic function h, such that $h(s)$ estimates the probability $prob^{\mathcal{M}}(R^*)$. Note that, during the search, R^* is not known. It is even not guaranteed that a trace from s to a target state exists at all. Thus, $h(s)$ can only be a heuristic value. We then define

$$f(s) = g(s) \cdot h(s). \tag{3}$$

The determination of a suitable h is a manual process and tailored to the particular application. Some work on automatically synthesizing heuristic estimate functions is ongoing [39]. The automated synthesis of heuristic estimate functions for stochastic model checking is beyond the scope of this paper.

XBF uses the evaluation function f in order to steer the search such that the most probable diagnostic traces will be selected first. Notice that f does not take the time bound into account that may be specified if φ is a bounded Until-formula.

4.2 State Evaluation in the Time-Bounded Case

Let $\mathcal{M}$ be a Markov chain and R be a trace in $\mathrm{STG}_{\mathcal{M}}$. In the time-bounded property case, we are interested only in those paths from $Paths^{\mathcal{M}}(R)$ which run along R completely within at most t time units. We refer to the set of such paths as $Paths^{\mathcal{M}}_{\leq t}(R)$. We refer to the probability of $Paths^{\mathcal{M}}_{\leq t}(R)$ as the *time-bounded probability* of R, denoted as $prob^{\mathcal{M}}_{\leq t}(R)$. In the case of a DTMC $\mathbf{D}$, the set $Paths^{\mathcal{D}}_{\leq t}(R)$ is empty iff $|R| > t$, or otherwise, identical to $Paths^{\mathcal{D}}(R)$. Consequently,

$$prob^{\mathcal{D}}_{\leq t}(R) = \begin{cases} prob^{\mathcal{D}}(R), & \text{if } |R| \leq t, \text{or} \\ 0 & \text{otherwise.} \end{cases}$$

We define the time-bounded variant of the evaluation function as

$$f_t(s) = g_t(s) \cdot h(s), \tag{4}$$

where $g_t(s) = prob^{\mathcal{D}}_{\leq t}(R_s)$. Remember that R_s is the solution base of s.

In the continuous-time case, we first consider the case of a time upper bound. That means the time constraint is an interval of the form $[0, t]$. For a CTMC $\mathcal{C}$, the set $Paths^{\mathcal{C}}_{\leq t}(R)$ is defined as

$$\begin{aligned} Paths^{\mathcal{C}}_{\leq t}(\langle s_0, \ldots, s_n \rangle) \ = \Bigg\{ & \sigma \in Paths^{\mathcal{C}} \mid \\ & \sigma = s_0, t_0, \ldots, t_{n-1}, s_n, \ldots \text{and} \sum_{i=0}^{n-1} t_i \leq t \Bigg\}. \end{aligned}$$

The time-bounded probability $prob^{\mathcal{C}}_{\leq t}(R)$ is then defined as follows:

$$\begin{aligned} prob^{\mathcal{C}}_{\leq t}(\langle s_0, \ldots, s_n \rangle) &= Pr^{\mathcal{C}}_{s_0}\big(Paths^{\mathcal{C}}_{\leq t}(R)\big) \\ &= Pr^{\mathcal{C}}_{s_0}\left(\bigcup_{t_0, \ldots, t_{n-1} : \sum_{i=0}^{n-1} t_i \leq t} cyl(s_0, [0,\ t_0], \ldots, [0,\ t_{n-1}], s_n) \right). \end{aligned}$$

The direct computation of this value is very expensive and afflicted with numerical instability problems. Thus, $prob^{\mathcal{C}}_{\leq t}(R)$ cannot be used directly in evaluating traces during the search. We address this problem using an approximation based on the uniformization of the CTMC [40], [41], [42], [10]. Briefly speaking, the uniformization method turns a given CTMC $\mathcal{C} = (\mathbf{S}, \hat{s}, \mathbf{R}, \mathbf{L})$ into a DTMC $\mathcal{D}_{\mathcal{C}} = (\mathbf{S}, \hat{s}, \mathbf{P}, \mathbf{L})$ by normalizing all rates in $\mathcal{C}$ with respect to a *uniformization rate* q, where $q \geq max\{\ \Lambda(s) \mid s \in \mathbf{S}\ \}$. In the uniformized DTMC $\mathcal{D}_{\mathcal{C}}$, the transition probability $\mathbf{P}(s,\ s') = \frac{\mathbf{R}(s,\ s')}{q}$ is assigned to each pair of states $(s,\ s')$. Every transition in $\mathcal{D}_{\mathcal{C}}$ corresponds to a time delay, which is exponentially distributed with a rate q. For any state s with $\Lambda(s) < q$, the residence time of s is probabilistically longer than a single exponentially distributed delay with the rate q. Therefore, a self-loop with the transition probability $\mathbf{P}(s,\ s) = \frac{q-\Lambda(s)}{q}$ is added. This self-loop allows the system to stay in s after a transition.

Example 2. The uniformization of the CTMC given in Fig. 2 using the uniformization rate $q := max\{\ \Lambda(s) \mid s \in \mathbf{S}\ \} = 10$ will yield to the DTMC defined in Fig. 1.

The uniformized DTMC $\mathcal{D}_{\mathcal{C}}$ is embedded into a Poisson process $PP_{q,t}$ which describes the probability that $\mathcal{D}_{\mathcal{C}}$ makes a certain discrete number k of transitions within the continuous-time budget t. $PP_{q,t}$ is formally defined as $PP_{q,t}(k) := \frac{(q\cdot t)^k}{k!} \cdot e^{-q\cdot t}$.

In our counterexample generation method, we perform the XBF search on the state transition graph $\mathrm{STG}_{\mathcal{D}_{\mathcal{C}}}$ of the uniformized DTMC $\mathcal{D}_{\mathcal{C}}$. This results in selecting a diagnostic subgraph of $\mathrm{STG}_{\mathcal{D}_{\mathcal{C}}}$. The question is now what the selected subgraph means in $\mathcal{C}$. It is hard to interpret the set of paths, which are represented by R, in $\mathcal{C}$. To solve this problem, we also take those paths into account which execute the self-loops introduced by the uniformization. Formally, we define a path set $Paths^{\mathcal{D}_{\mathcal{C}}}_{\bigcirc\leq k}(R)$ for a discrete time bound (i.e., step bound) as follows:

$$Paths^{\mathcal{D}_{\mathcal{C}}}_{\bigcirc\leq k}(\langle s_0 \dots s_n\rangle) =$$
$$\left\{ \sigma \in Paths^{\mathcal{D}_{\mathcal{C}}} \mid \sigma = s_0^{k_0} \dots s_n^{k_n} \dots \text{for some } k_0, \dots, k_n > 0 \text{ with} \sum_{i=0}^{n-1} k_i \leq k \right\}.$$

Then, we define $prob^{\mathcal{D}_{\mathcal{C}}}_{\bigcirc\leq k}(R) = Pr^{\mathcal{D}_{\mathcal{C}}}_{s_0}(Paths^{\mathcal{D}_{\mathcal{C}}}_{\bigcirc\leq k}(R))$. Note that $Pr^{\mathcal{D}_{\mathcal{C}}}_{s_0}(Paths^{\mathcal{D}_{\mathcal{C}}}_{\bigcirc\leq k}(R))$ is the probability to run a path along R including possible self-loops within at most k steps. It is also the probability to reach s_n restricted to R within k steps. This, in turn, is equivalent to the transient probability $\pi_R(s_n, k)$ of being in s_n at step k when considering s_n as being absorbing:

$$prob^{\mathcal{D}_{\mathcal{C}}}_{\bigcirc\leq k}(R) = prob^{\mathcal{D}_{\mathcal{C}}}_{\bigcirc\leq k}(\langle s_0 \dots s_n\rangle) = \pi_{\langle s_0 \dots s_n\rangle}(s_n, k). \quad (5)$$

We will show later how the function $\pi_{\langle s_0 \dots s_n\rangle}$ can be computed. Notice that $prob^{\mathcal{D}_{\mathcal{C}}}_{\bigcirc\leq k}(R)$ is the probability measured in the uniformized DTMC $\mathcal{D}_{\mathcal{C}}$. In order to compute $prob^{\mathcal{C}}_{\leq t}(R)$, i.e., the probability of R in the CTMC $\mathcal{C}$, we have to involve the Poisson probability distribution for all possible discrete numbers of steps, as mentioned above. This results in the following equation:

$$prob^{\mathcal{C}}_{\leq t}(\langle s_0 \dots s_n\rangle) = \sum_{k=0}^{\infty} PP_{q,t}(k) \cdot prob^{\mathcal{D}_{\mathcal{C}}}_{\bigcirc\leq k}(\langle s_0 \dots s_n\rangle).$$

The infinite sum has to be truncated to a finite one consisting of the first K terms. It is possible to compute the number K such that the truncation error is confined to a very small bound ϵ [10]

$$prob^{\mathcal{C}}_{\leq t}(\langle s_0 \dots s_n\rangle) \approx \sum_{k=0}^{K} PP_{q,t}(k) \cdot prob^{\mathcal{D}_{\mathcal{C}}}_{\bigcirc\leq k}(\langle s_0 \dots s_n\rangle).$$

Together with (5), we derive the following equation:

$$prob^{\mathcal{C}}_{\leq t}(\langle s_0 \dots s_n\rangle) \approx \sum_{k=0}^{K} PP_{q,t}(k) \cdot \pi_{\langle s_0 \dots s_n\rangle}(s_n, k).$$

Recall that our goal is to come up with a suitable evaluation function to guide the search rather than a precise computation of the probabilities. Hence, we want to avoid computing the summation given above, which can be very expensive, in particular, for large K. For this reason, we make the following simplification: Instead of considering all numbers of discrete hops and their probability distribution described by the Poisson process, we simply allow the $\mathcal{D}_{\mathcal{C}}$ to make the maximal number of hops K. In other words, we simplify the problem by assuming that K discrete hops will happen with the probability 1. We define a function g_π based on this simplification which will replace the functions g and g_t used in f and f_t. Let s be a state explored by XBF. Then, $g_\pi(s) = \pi_{R_s}(s, K)$, where R_s is the solution base of s. $g_\pi(s)$ is then an overapproximation of the probability of reaching s via the trace R_s within time t. In principle, various alternative simplifications could be used, for instance, to consider the expected value of the Poisson process instead of K. However, we chose K because it ensures that g_π is an optimistic estimate of the probability of R_s. Optimistic estimates are preferable in heuristic search because they usually improve the quality of the solution. Note also that an optimistic heuristic estimate is necessary in A* to guarantee the optimality of the solution [8], [9]. Similar to the evaluation function f defined in (3), we use g_π to define a new state evaluation function f_π as:

$$f_\pi(s) = g_\pi(s) \cdot h(s). \quad (6)$$

We next show how to compute the transient probability distribution $\pi_{\langle s_0 \dots s_n\rangle}$. We assume s_0 to be the start point. Formally,

$$\pi_{\langle s_0 \dots s_n\rangle}(s_i, 0) = \begin{cases} 1, & \text{if } i = 0 \\ 0, & \text{otherwise.} \end{cases}$$

Now we compute $\pi_{\langle s_0 \dots s_n\rangle}(s_i, k)$ for time point $k > 0$. The probability to be in s_0 at any time point greater than 0 arises from firing the self-loop at s_0. More precisely, $\pi_{\langle s_0 \dots s_n\rangle}(s_0, k) = \pi_{\langle s_0 \dots s_n\rangle}(s_0, k-1) \cdot \mathbf{P}(s_0, s_0)$. For any other state s_i, with the exception of the last state s_n, a portion of the probability comes from the predecessor s_{i-1} and another portion comes from firing the self-loop at s_i. Therefore,

$$\pi_{\langle s_0 \ldots s_n \rangle}(s_i, k) = \pi_{\langle s_0 \ldots s_n \rangle}(s_{i-1}, k-1) \cdot \mathbf{P}(s_{i-1}, s_i) + \pi_{\langle s_0 \ldots s_n \rangle}(s_i, k-1) \cdot \mathbf{P}(s_i, s_i).$$

We treat s_n, the last state of R as an absorbing state. That means

$$\pi_{\langle s_0 \ldots s_n \rangle}(s_n, k) = \pi_{\langle s_0 \ldots s_n \rangle}(s_{n-1}, k-1) \cdot \mathbf{P}(s_{n-1}, s_n) + \pi_{\langle s_0 \ldots s_n \rangle}(s_n, k-1).$$

We notice that $\pi_{\langle s_0 \ldots s_n \rangle}$ is recursively computed. In particular, for computing $\pi_{\langle s_0 \ldots s_n \rangle}(s_n, K)$, we need the values $\pi_{\langle s_0 \ldots s_n \rangle}(s_{n-1}, k)$ for all $k \in \{0, \ldots, K-1\}$. This means that, during the XBF search, we have to store a vector $(\pi_{R_s}(s,k))_{0 \le k < K}$, which we denote by $\underline{\pi}(s)$, for each open state s. This vector is needed to compute g_π for the successors of s when s is expanded.

If the time constraint is an interval $[t_1, t_2]$, then our heuristic would just consider the upper bound t_2. This is justified since each trace which represents paths with execution times less than t_1 also represents paths with execution time in $[t_1, t_2]$. The probability of diagnostic paths with a duration of less than t_1 will be excluded by the model checking of the counterexample.

Note that the time bound t induces a depth bound for the search. Both evaluation functions f_t and f_π depend on the depth of the currently explored state. In order to ensure completeness of the search, we consider the minimal depth of a state when computing its f_t or f_π-value.

Both state evaluation functions f and f_t, cf., (3) and (4), are relatively easy to compute. Computing f_π requires additional runtime and memory effort for computing and storing the $\underline{\pi}$ vectors. This could have a negative impact on the performance of the search algorithms that we use, in particular, for large time bounds when the $\underline{\pi}$ vectors become bigger. On the other hand, for any trace R in a Markov chain $\mathcal{M}$, the bigger the time bound t is, the closer $prob^{\mathcal{M}}_{\le t}(R)$ is to $prob^{\mathcal{M}}(R)$. In other words, the advantage gained by the more precise evaluation using f_π is less significant for large time bounds. In our experiments, we hardly found a case where the effort needed to compute f_π paid off, see Section 5. Hence, from the practical point of view, in the case of a CTMC, we recommend using f instead of f_π, in particular, when the time bound is large or if a decline in the quality of the counterexample is acceptable.

4.3 XBF-Based Algorithms

As we will show soon, the evaluation functions f, f_t, and f_π, as defined in (3), (4), and (6), are computed by a recursive procedure.

This means that the algorithms obtained from XBF by using f, f_t, or f_π are extended variants of the Z algorithm (see Section 3.1). When f or f_t is used, we refer to the algorithm as XZ. If the algorithm uses f_π, then we refer to it as XZ_π. Further, if no heuristic estimate h is used in the evaluation function, we get undirected variants of the algorithms which we refer to as XUZ and XUZ_π.

In order to show that f_π is recursively computed, we have to verify that $f_\pi(s) = F(\chi(r), f_\pi(r), h(s))$ when an arbitrary state r is expanded generating a successor state s. F stands for an arbitrary function.

From the definition of g_π, we can see that $g_\pi(s) = F'(\underline{\pi}(r), \mathbf{P}(r,s))$ for a particular combination F'. Furthermore, $g_\pi(s) \cdot h(s) = F'(\underline{\pi}(r),\ \mathbf{P}(r,s)) \cdot h(s) = F''(\underline{\pi}(r),\ \mathbf{P}(r,s), h(s))$ for a further combination function F''. $\underline{\pi}(r)$ and $\mathbf{P}(r,s)$ are local parameters of r. Thus, we define $\chi(r)$ as $\{\underline{\pi}(r), \mathbf{P}(r,s)\}$. Consequently, $F''(\underline{\pi}(r), \mathbf{P}(r,s), h(s)) = F''(\chi(r), h(s))$. In summary,

$$f_\pi(s) = g_\pi(s) \cdot h(s) = F'(\underline{\pi}(r), \mathbf{P}(r,s)) \cdot h(s) = F''(\chi(r), h(s)) = F(\chi(r), f_\pi(r), h(s)).$$

The recursiveness of f and f_t can be derived by similar considerations.

4.4 Example

We illustrate the functioning of the algorithm using the following example. Let $\mathbf{C}$ be the CTMC given in Fig. 2. We consider the property $\mathcal{P}_{\le 0.4}(true\ U^{\le 1.0}\ crash)$. The CTMC $\mathcal{C}$ refutes the property because $Pr^{\mathcal{C}}(true\ U^{\le 1.0}\ crash)$, according to the model checker PRISM, is 0.63 which is greater than 0.4. To generate a counterexample, we applied XUZ_π on the uniformized DTMC given in Fig. 1. We observed the events illustrated in Fig. 3. At some point of the search, the algorithm will select the diagnostic subgraph highlighted by bold lines in Fig. 3a. The corresponding diagnostic CTMC, denoted here as $\mathcal{C}_1$, is given in Fig. 3b. This diagnostic subgraph is not a counterexample, since at this point, the model checker computes the probability $Pr^{\mathcal{C}_1}(true\ U^{\le 1.0}\ crash) = 0.30$ while the probability bound is 0.4. After some iterations, the solution will grow into the subgraph highlighted in Fig. 3c. The corresponding diagnostic CTMC, denoted here as $\mathcal{C}_2$, violates the property since the model checker computes the probability $Pr^{\mathcal{C}_2}(true\ U^{\le 1.0}\ crash) = 0.50 > 0.4$. Hence, this diagnostic subgraph will be delivered to the user as a counterexample. Note that it is not possible to decrease the total probability to be under 0.4, the given upper bound, without applying changes to the part of the model returned as a counterexample. It is easy to see that the best way to debug the model is to confine the factors leading to a *crash* during initialization (state s_0) and running (state s_2). This means reducing the rates of the transitions (s_0, s_3) and (s_2, s_5). Crashes during the ready state (state s_1) have a secondary impact.

If the algorithm would continue, it would select the complete diagnostic subgraph which can be found as given in Fig. 3d.

4.5 Further Application Scenarios

The main application of our approach is to provide counterexamples when a given Markov chain $\mathcal{M}$ refutes a given upper-bounded property $\mathcal{P}_{\bowtie p}(\varphi)$. However, this does not mean that our approach is only applicable after model checking. An important feature of this approach is that it allows the analysis of $\mathcal{P}_{\bowtie p}(\varphi)$ even if $\mathcal{M}$ is too large to fit completely into the memory, and hence, a complete model checking is not possible. Since our approach is based on directed search, which does not require generating the whole state transition graph $\mathrm{STG}_{\mathcal{M}}$, it can be applied to such huge models. Often, a diagnostic subgraph with probability greater than p can be found by exploring a relatively small portion of $\mathrm{STG}_{\mathcal{M}}$, which can be accommodated in the memory. Our method can therefore be

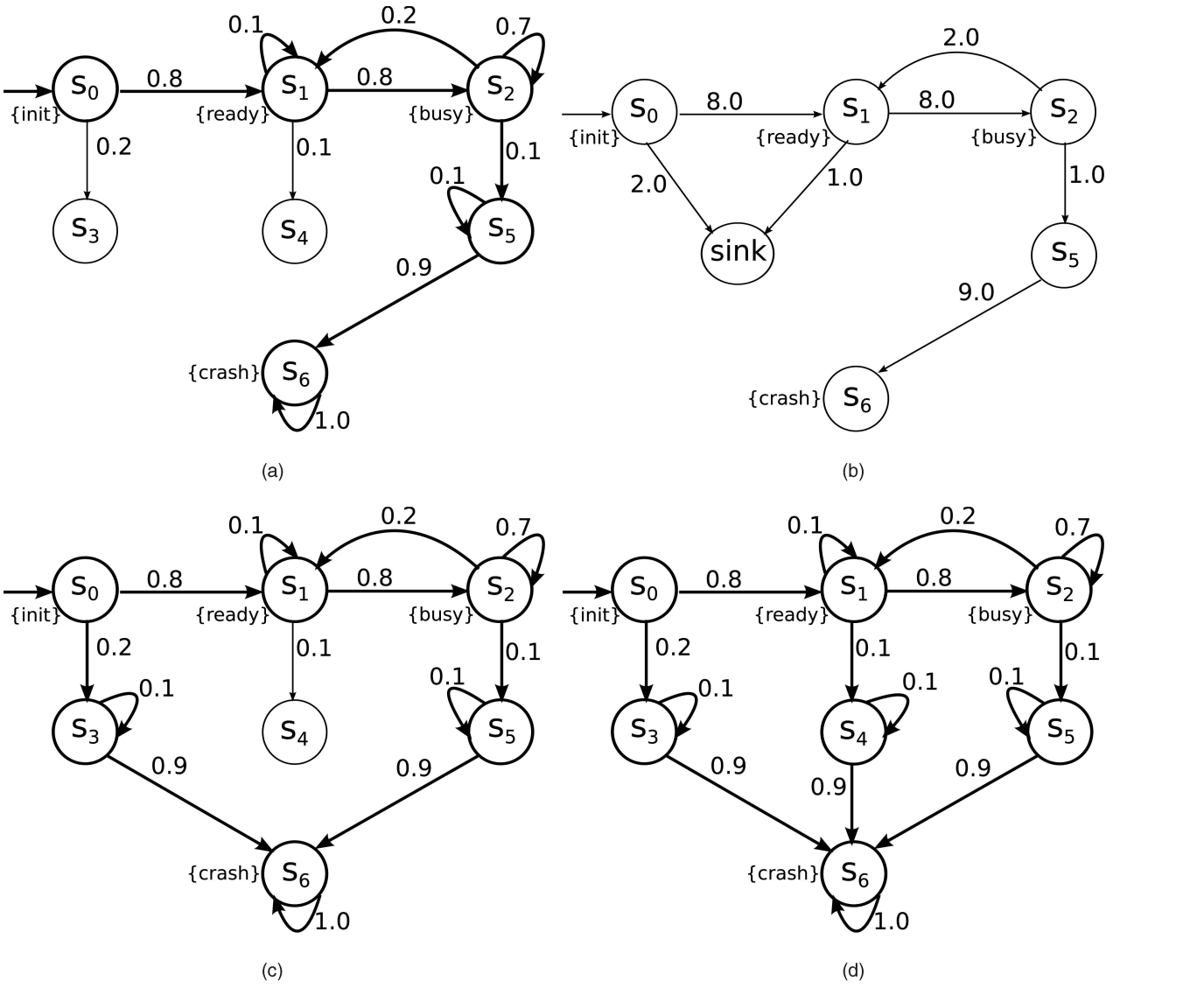

Fig. 3. Incremental selection of a counterexample. (a) Diagnostic subgraph with a probability of 0.30. (b) The diagnostic CTMC constructed from the diagnostic subgraph from (a). (c) Diagnostic subgraph with a probability of 0.50. (d) The complete diagnostic subgraph with a probability of 0.65.

applied in cases where a complete stochastic model checking would fail because the complete model is too large to fit into the memory.

Sometimes, one is interested in knowing the probability $Pr^{\mathcal{M}}(\varphi)$, or usually an approximation of it, rather than knowing whether $\mathcal{P}_{\leq p}(\varphi)$ is satisfied or not. XBF-based algorithms approximate the probability $Pr^{\mathcal{M}}(\varphi)$ from below. Checking the property on the diagnostic Markov chain obtained from the complete diagnostic subgraph results in the same probability as checking it on the original Markov chain. A sufficient approximation can often be achieved by selecting even a very small diagnostic subgraph. In such cases, XBF allows fast analysis since selecting a suitable diagnostic subgraph and checking the property on it can be performed faster than checking the property on the original model; see, for example, the case study in Section 5.3.

4.6 Dealing with Lower-Bounded Formulas

We argued in Section 2.4 that a direct definition of counterexamples for lower-bounded $\mathbf{P}$-formulas is not possible. Instead, we provide a counterexample for $\mathcal{P}_{\geq p}(\varphi)$ as a measurable set $X \subseteq \{\sigma \in Paths^{\mathbf{M}} \mid \sigma \not\models_{\mathcal{M}} \varphi\}$ such that $Pr^{\mathcal{M}}(X) > 1 - p$. The set X proves that $Pr^{\mathcal{M}}(\varphi) < p$ and can be considered as a counterexample for $\mathcal{P}_{\geq p}(\varphi)$. The challenge is then to collect paths violating φ.

First, we handle the case when φ is a time-unbounded Until-formula, i.e., $\varphi = (\phi_1\ U\ \phi_2)$. Our objective is to select a measurable set X of paths violating $(\phi_1 U \phi_2)$ such that $Pr^{\mathcal{M}}(X) > 1 - p$. A path violates $(\phi_1 U \phi_2)$ if it leads to a state which either violates both ϕ_1 and ϕ_2, or it never reaches a state satisfying ϕ_2. We formally state this fact as follows:

$$\sigma \not\models_{\mathcal{M}} (\phi_1\ U\ \phi_2)$$
$$\Leftrightarrow \quad \sigma \models_{\mathcal{M}} \left(\underbrace{(\phi_1 \wedge \neg\phi_2)}_{=:\bar{\phi}_1}\ U\ \underbrace{((\neg\phi_1 \wedge \neg\phi_2) \vee \phi^*)}_{=:\bar{\phi}_2} \right).$$

ϕ^* is a new state proposition which indicates for each state whether it belongs to a *bottom strongly connected component* (BSCC), the states of which satisfy $\bar{\phi}_1 = (\phi_1 \wedge \neg\phi_2)$. A BSCC

is a maximal strongly connected component which does not have any outgoing transitions. That means, a state s satisfies ϕ^* iff all states which are reachable outgoing from s satisfy $\bar{\phi}_1$. We can handle a strictly lower-bounded property $\mathcal{P}_{>p}(\phi_1\ U\ \phi_2)$ which results in the upper-bounded property $\mathcal{P}_{<1-p}(\bar{\phi}_1\ U\ \bar{\phi}_2)$. To generate a counterexample, we need to compute all BSCCs first, using an adoption of an algorithm for strongly connected components like [43] and states which belong to BSCCs satisfying $\bar{\phi}_1$ are labeled with the new proposition ϕ^*. Afterward, we can apply the described approach for upper-bounded properties to $\mathcal{P}_{\leq 1-p}(\bar{\phi}_1\ U\ \bar{\phi}_2)$ or $\mathcal{P}_{<1-p}(\bar{\phi}_1\ U\ \bar{\phi}_2)$, depending on the strictness of the bound.

Next, we consider the case when φ is a time-bounded until-formula of the form $\varphi = (\phi_1\ U^{\leq t}\ \phi_2)$. A path in $\mathcal{M}$ violates $(\phi_1\ U^{\leq t}\ \phi_2)$ if either it does not reach a ϕ_2-state within t time units, or if it reaches a state violating ϕ_1 before it reaches a ϕ_2-state

$$\sigma \not\models_{\mathcal{M}} (\phi_1\ U^{\leq t}\ \phi_2) \quad \Leftrightarrow \quad \sigma \models_{\mathcal{M}} \underbrace{(\neg\phi_2\ U^{\leq t}\ \neg\phi_1)}_{=:\bar{\varphi}_1} \vee \sigma \models_{\mathcal{M}} \underbrace{\Box^{\leq t}(\phi_1 \wedge \neg\phi_2)}_{=:\bar{\varphi}_2}.$$

$\bar{\varphi}_1$ characterizes those paths which violate $(\phi_1\ U^{\leq t}\ \phi_2)$ because they reach a state violating ϕ_1 before reaching a ϕ_2-state within time t. On the other hand, $\bar{\varphi}_1$ characterizes those paths which violate $(\phi_1\ U^{\leq t}\ \phi_2)$ because they do not reach a ϕ_2-state within time t. This equivalence holds for PCTL formulas on DTMCs and CSL formulas on CTMCs. In the case of a DTMC, we apply XBF to search for traces satisfying $\bar{\varphi}_1$ or $\bar{\varphi}_2$. Every time we want to measure the probability of the selected diagnostic subgraph, we need to run the model checker twice to compute the probability of both $Pr^{\mathcal{M}}(\bar{\varphi}_1)$ and $Pr^{\mathcal{M}}(\bar{\varphi}_2)$. Notice that $\bar{\varphi}_2 = \Box^{\leq t}(\phi_1 \wedge \neg\phi_2)$ is not a PCTL path formula. Nevertheless, standard stochastic model checkers such as PRISM have algorithms available to model check it. We point out that no path can satisfy both $\bar{\varphi}_1$ and $\bar{\varphi}_2$. We can, therefore, easily add both probabilities together to obtain the total probability of the diagnostic subgraph. A counterexample is found once the sum of both probabilities exceeds $1 - p$.

In the case of a CTMC, we apply XBF search to the STG of the uniformized DTMC, as described above. We substitute the continuous-time bound t with the discrete bound K, as mentioned in Section 4.2.

5 Experimental Evaluation

We evaluated our approach experimentally with respect to the quality of the delivered counterexamples and the computational costs. The computational costs involve runtime and memory consumption. We measure the quality of a counterexample in terms of the ratio between its size and the size of the entire model. The size here is defined as the sum of the number of states and transitions. The ratio between the size of the counterexample and the entire model is a good measure to assess which portion of the complete model the user needs to analyze in order to understand why the model refutes the given property.

We implemented the XBF search strategy and instantiated from it the algorithms XZ, XUZ, XZ_π, and XUZ_π. We also implemented Eppstein's algorithm to experimentally compare our approach with the KSP-based approach of [26], [27]. Our implementation uses the *PRISM Simulation Engine* to generate the state transition graph of the DTMCs and CTMCs that we use in our experiments.

In all experiments, we proceeded as follows: Let $\mathcal{M}$ be the Markov chain and φ the path formula which we are interested in. We generated a counterexample for the upper-bounded property $\mathcal{P}_{<p}(\varphi)$, where $p = Pr^{\mathcal{M}}(\varphi)$ computed using the PRISM model checker. In this way, we can obtain experimental information about generating counterexamples not only for the upper-bound p but for any probability bound less than or equal to p. We truncated insignificant decimal places from p in order to dispose of rounding errors, which may be made during the checking of the model or the counterexamples. We ran all experiments on a machine having an Intel Pentium CPU with 3.2 GHz speed[1] and 2 GB of memory. For technical reasons, we limited the runtime of all algorithms to at most one hour (3,600 seconds). The algorithms were implemented in Java.

We considered five case studies, cf., Sections 5.1-5.5. We study the scalability of our method regarding the model size in Section 5.6. We then demonstrate the impact of specifying different time bounds in the property on the various algorithms in Section 5.7. In Section 5.8, we analyze how a variation of the value of $Pr^{\mathcal{M}}(\varphi)$ influences the counterexample generation. Finally, we summarize our conclusions from the experiments in Section 5.9.

5.1 Workstation Cluster

This case study is a CTMC model of a dependable cluster of workstations, as first presented in [44]. It represents a system consisting of two subclusters connected via a backbone. Each subcluster consists of N workstations with a central switch that provides the interface to the backbone. Each of the components of the system (workstations, switches, and backbone) can break down with a certain rate. At least $\frac{2}{3}N$ workstations have to be operationally connected to each other via operational switches in order to provide minimum quality of service (QoS). We refer to this model with the short name *Cluster*.

We are interested in the likelihood that the QoS drops below the minimum within 100 time units, i.e., the path formula is $(true\ U^{\leq 100}\ \neg\ minimum)$. Here, we set the parameter N to 64 which results in a CTMC with 151,060 states and 733,216 transitions. PRISM computed the probability $Pr(true\ U^{\leq 100}\ \neg\ minimum) = 5.022E-5$ within 95.465 seconds using 5,077.4 KB. We ran the algorithms XZ, XUZ, XZ_π, XUZ_π, and Eppstein to generate counterexamples for the property $\mathcal{P}_{<5.022E-5}(true\ U^{\leq 100}\ \neg\ minimum)$. We invented the following, relatively simple heuristic function h to be used in XZ and XZ_π. The switches are central components of the system. Thus, our heuristic focuses on them and ignores other components. For each state s, we define two values $h_l(s)$ and $h_r(s)$. If a switch is down, then the subcluster is not operational. In this case, the corresponding values of $h_l(s)$ and $h_r(s)$ are equal to 1. Otherwise, at least one transition is required for a switch to break down. This is indicated by the

1. The CPU had two cores with 3.2 GHz each. However, we do not benefit from both the cores because our implementation does not contain any kind of parallelism.

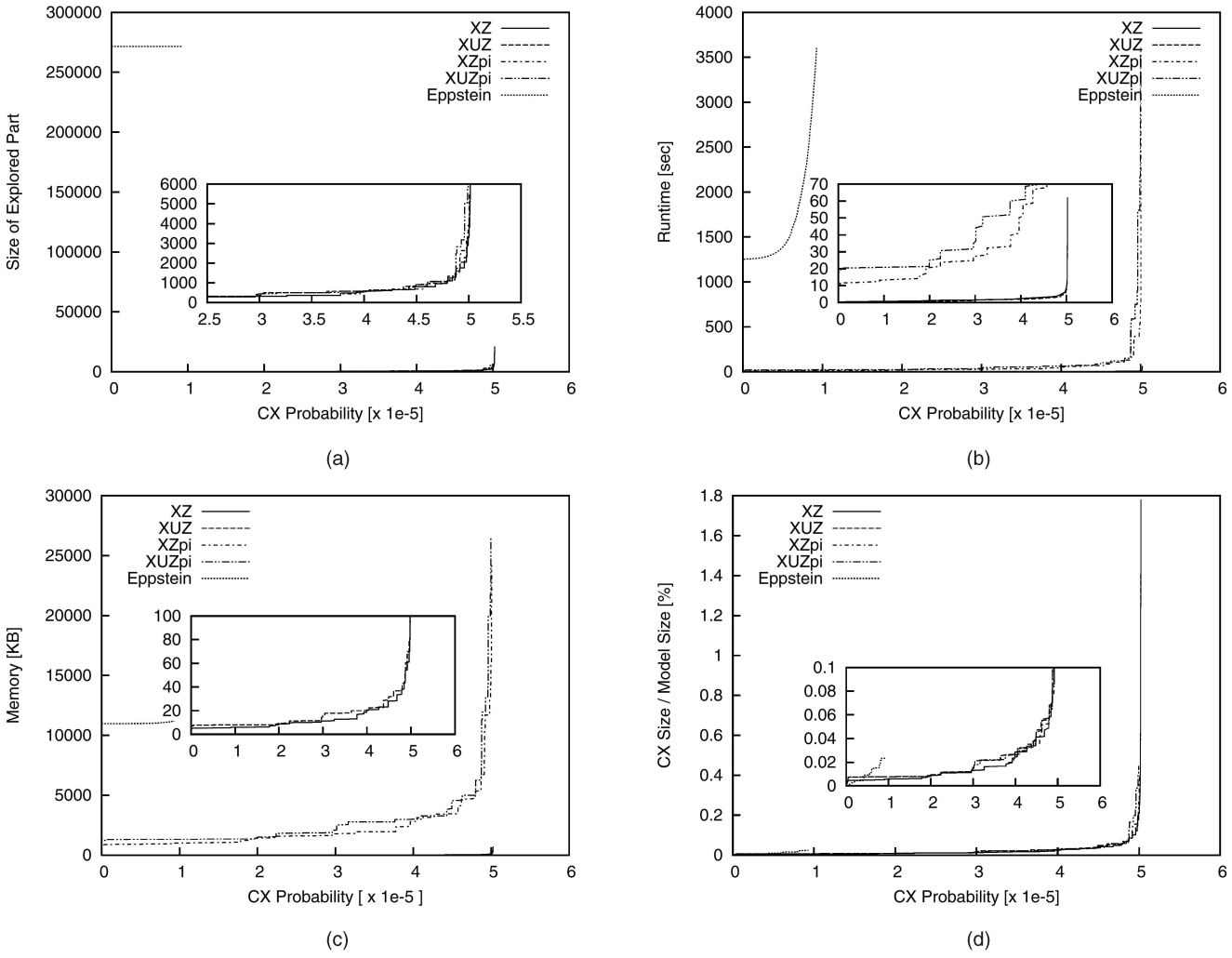

Fig. 4. Counterexample generation for $\mathcal{P}_{<5.022E-5}(true\ U^{\leq 100}\ \neg\ minimum)$ on Cluster ($N = 64$). (a) Size of explored part. (b) Runtime. (c) Memory consumption. (d) Counterexample quality.

value 0.5 of $h_l(s)$ or $h_r(s)$, respectively. Both switches need to break down in order to reach a state where both subclusters are not operational. Thus, we set $h(s) = h_l(s) \cdot h_r(s)$.

The results of the counterexample generation are illustrated in Fig. 4. The X-axis indicates the probability of the counterexample which has been selected so far. Figs. 4b and 4c show the runtime and memory effort needed by the various algorithms. Fig. 4a shows, for each algorithm, the size of portion of the state transition graph which has been explored to provide a counterexample with a certain amount of probability. Fig. 4d shows the quality of the counterexample. We use similar diagrams to present the results for each case study. The diagrams should be read as follows: To provide a counterexample for a probability upper bound 3 E-5, XZ needed about 2 seconds of runtime (cf., Fig. 4b), 15 KB of memory (cf., Fig. 4c) and explored about 350 states and transitions (cf., Fig. 4a). XZ_π needed about 30 seconds, approximately 1,700 KB and explored about 420 states and transitions. The counterexample in both cases covers about 0.03 percent of the entire model (cf., Fig. 4d). Eppstein failed to provide a counterexample for this probability bound after exploring 270,000 states and transitions and 1 hour of runtime and more than 11,000 KB.

Fig. 4a shows that all XBF-based algorithms provided a counterexample with the total probability $p = 5.022$ E-5 by exploring less than 23,000 states and transitions. Eppstein's algorithm had to explore the complete state transition graph, i.e., about 270,000 states and transitions, before it could provide counterexamples for any probability bound. This is also reflected in the required runtime and memory, as shown in Figs. 4b and 4c. Eppstein's algorithm delivers the first diagnostic trace after approximately 1,250 seconds and after consuming more than 10,000 KB of memory. After 1 hour of runtime, Eppstein could provide counterexamples only for probability bounds less than $1\ E{-}5$.

All XBF-based algorithms explored portions of the STG of approximately the same size, cf., Fig. 4a. Figs. 4b and 4c show that the computational costs, in particular, the memory consumption, of XZ_π and XUZ_π are higher than those of XZ and XUZ since additional effort is needed for storing and computing the $\underline{\pi}$-vectors.

Fig. 4d shows that the counterexample covers less than 1.8 percent of the entire model in all cases. In order to debug the model, the user needs to analyze at most 1.8 percent, and in most cases, even less than 0.2 percent of the entire model.

The curves of XZ, XUZ, XZ_π, and XUZ_π in all plots of Fig. 4 increase very slowly at first and then increase rapidly

just before reaching the total probability p. This confirms that our proposed algorithms discover the most probable diagnostic traces quite early on in the search and that the remaining parts of the model contribute with relatively small probability. When only diagnostic traces with low probabilities remain, a noticeable increase in the probability requires relatively high search effort. If we compare the curves of XZ_π and XUZ_π, we notice some advantage in guiding the search by a heuristic estimate h in terms of both runtime and memory consumption. This advantage is smaller in the case of XZ and XUZ. The more precise evaluation function f_π in algorithms XZ_π and XUZ_π did not lead to a significant improvement of the counterexample quality in this case study.

5.2 Embedded Control System

This case study models an embedded control system based on the one presented in [45]. The system consists of a main processor (**M**), an input processor (**I**), an output processor (**O**), three sensors, and two actuators. The input processor I reads data from the sensors and forwards it to M. Based on this data, M sends instructions to the output processor O which controls both actuators according to the received instructions. This model, which we refer to as *Embedded*, is translated by PRISM into a CTMC **C** comprising 8,548 states and 36,041 transitions (model parameter $\mathrm{MAX\_COUNT} = 8$).

Various failure modes, e.g., failure of I, M, or O or sensor or actuator failure, can lead to shut down of the system. We are interested in the likelihood that the system shuts down within one hour. One hour corresponds to 3,600 time units as one time unit is one second according to the description of the PRISM model. That means we consider the path formula of $(true\ U^{\leq 3{,}600}\ down)$. PRISM computes the probability $Pr^{\mathbf{C}}(true\ U^{\leq 3{,}600}\ down)$ as 3.0726 E-4. This required approximately 0.3 seconds and 314.4 KB. We applied the various algorithms in order to generate counterexamples for the upper-bounded property $\mathcal{P}_{\leq 3.0726\ E-4}(true\ U^{\leq 3{,}600}\ down)$.

We defined the heuristic function h used in the algorithm XZ and XZ_π based on the following idea. For each failure mode, we first determine transitions which are necessary to reach from s, a state which represents the corresponding failure. For instance, according to the model, the system is shut down when two sensors fail. Thus, in order to reach a failure state of the sensor component, at least two sensor failure transitions have to be fired. The heuristic estimate then corresponds to the probability of firing these necessary transitions. This is calculated as the product of the probabilities of these transitions. The probability that the system is shut down is, roughly speaking, equal to the probability that any of failure modes occurs. Hence, we define $h(s)$ to be the sum of the estimates for all failure modes. The summation of the component probabilities is justified since the various failure modes are independent.

We present the results of the counterexample generation in Fig. 5. Fig. 5a shows that all XBF-based algorithms provided counterexamples for the most upper bounds by exploring less than 800 states and transitions. They required less than 15 seconds, cf., Fig. 5b. XZ and XUZ consumed about 100 KB to achieve this, whereas XZ_π and XUZ_π needed about 500 KB. Meanwhile, Eppstein explored the complete STG, with approximately 20,000 states and transitions, before it provided a counterexample. It was able to provide a large portion of the counterexample after about 50 seconds. However, after this, it failed to make a noticeable progress with realistic effort. This effect can be explained as follows: Since the STG is relatively small, Eppstein explored quickly and selected the first most probable diagnostic traces. These traces carried the most probability. The remaining diagnostic traces had insignificant individual probabilities which impeded the progress of the algorithm. This also explains the rapid increase in the curves of XZ, XUZ, XZ_π, and XUZ_π just before the end, as shown in Fig. 5. In particular, the steep increase of the counterexample size, to achieve the total probability of 3.0726 E-4 (see Fig. 5d), confirms that the remaining probability is distributed over a relatively large portion of the model. The XBF algorithms tackled the problem more effectively than Eppstein for the following reason. Whenever Eppstein finds a diagnostic trace, it adds the individual probability of that trace to the counterexample. XBF algorithms, however, incorporate the accumulated probability of all diagnostic traces formed by the states and transitions of the found trace. This can be an infinite set of traces if the found trace contains a cycle. The accumulated probability of this set can be very high even if the probability of the found trace is itself insignificant.

As can be inferred from Fig. 5d, the quality of the counterexamples provided by XZ and XZ_π is almost identical (both curves overlap) and it is better than the quality of those provided by XUZ. We also notice a minor advantage of XUZ_π in the range between 1 E-4 and 2.1 E-4 which is explained by the usage of f_π. However, the advantage of guiding the search by a heuristic is more effective. We also see an advantage of using a heuristic in the runtime and memory consumption. The counterexamples provided by Eppstein for upper bounds in the range 1.2 E-4 to 2.9 E-4 are very small. The reason is that Eppstein's algorithm delivers only the most probable traces whereas XBF algorithms collect also traces with low probabilities (see Section 3.3.3).

5.3 Fibroblast Growth Factor Signaling

This is a case study from the area of system biology presented in [46], [47]. Fibroblast Growth Factors (*FGFs*) are a family of proteins which play a key role in the process of cell signaling within a variety of contexts such as wound healing. This case study models the early stages of FGF signal propagation as a CTMC. The model incorporates the main features of the process, namely: protein-protein interactions (e.g., competition for partners), phosphorylation and dephosphorylation, and protein complex relocation and protein complex degradation.

Here, we consider the probability that a particular cause of degradation/relocation, namely, relocation of FRS2 ($\mathrm{relocFRS2}{=}1$), has occurred first. The path formula is precisely formulated as $(relocFRS2 = 0 \wedge degFRS2 = 0 \wedge degFGFR = 0\ U\ relocFRS2 = 1)$. PRISM constructed the CTMC, which consists of 80,616 states and 562,536 transitions, and computed the probability

$$Pr(relocFRS2 = 0 \wedge degFRS2 = 0 \wedge degFGFR = 0\ U\ relocFRS2 = 1)$$

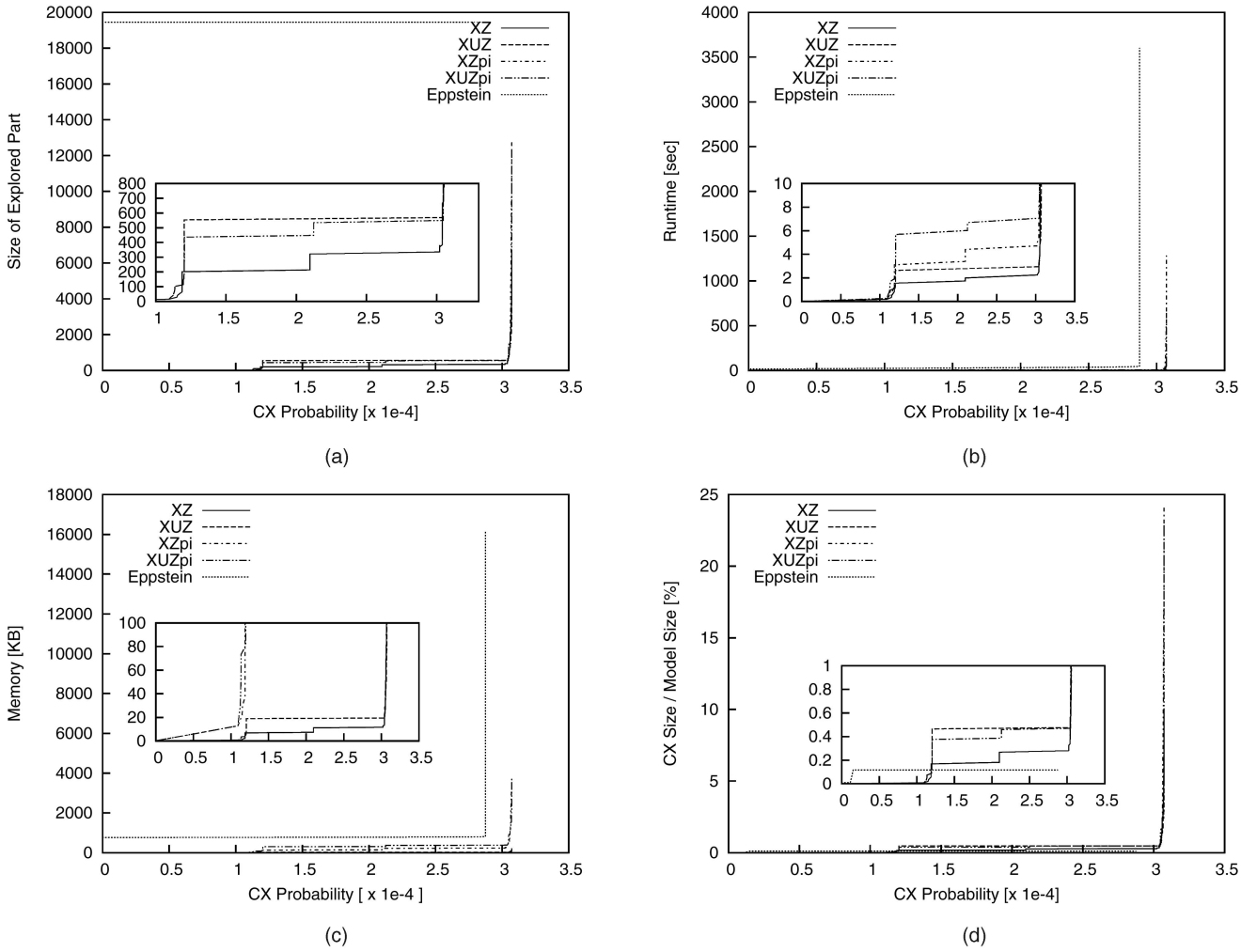

Fig. 5. Counterexample generation for $\mathcal{P}_{\leq 3.0726\ E-4}(true\ U^{\leq 3,600}\ down)$ on embedded. (a) Size of explored part. (b) Runtime. (c) Memory consumption. (d) Counterexample quality.

as 0.621258247 within 12,369.239 seconds (≈3.4 hours) while it consumed 54,975.6 KB of memory. We then generated counterexamples for the property $\mathcal{P}_{<0.621258247}(relocFRS2 = 0 \wedge degFRS2 = 0 \wedge degFGFR = 0\ U\ relocFRS2 = 1)$. As a heuristic, we used a function h which roughly estimates the number x of transitions which are required to reach a state where $relocFRS2 = 1$ and returns 0.5^x as a value. The results are reported in Fig. 6.

We deal here with a time-unbounded property which means that XZ_π and XUZ_π are not applicable. The first observation we make is that the curves of Eppstein in Figs. 6a, 6b, and 6c run in parallel to the Y-axis at counterexample probability 0. This indicates that Eppstein failed to deliver a counterexample for any probability bound within one hour. XZ and XUZ managed to provide counterexamples for almost all probability bounds after exploring about 100,000 states and transitions of the STG and consuming about 5,000 KB of memory. In Fig. 6b, we see that XZ is slightly faster than XUZ due to the use of a heuristic. Note that the runtime of XZ and XUZ exceeds the time limit of one hour. This is because both algorithms ended with a long counterexample checking step which we did not abort. XZ and XUZ required for this model very long time compared to both models Cluster and Embedded from Sections 5.1 and 5.2. This is because the model checking of this model, and consequently, of the generated counterexamples is much harder than the model checking in the case of Cluster or Embedded. Notice that PRISM needed about 3.4 hours for checking the entire model.

We see a steady increase of the curves of XZ and XUZ in all plots in Fig. 6 following a counterexample probability of approximately 0.35. This indicates that both algorithms had to explore larger portions of the STG and to capture more in the counterexample in order to achieve a probability increase. This is a typical effect for models in which the probability of the property is consistently distributed across widespread diagnostic traces in the model. Nevertheless, a counterexample which carries almost the total model probability comprises not more than 10 percent of the entire model.

5.4 Dynamic Power Management—IBM Disk Drive

This case study, which we refer to as *Power*, is a DTMC model of the power management of an IBM disk drive taken from [48]. Here, we are interested in the probability that 100 or more requests get lost within 400 milliseconds because of power management. This means that we are interested in the path formula $(true\ U^{\leq 800}\ lost = 100)$. Notice that, according to the description of the PRISM

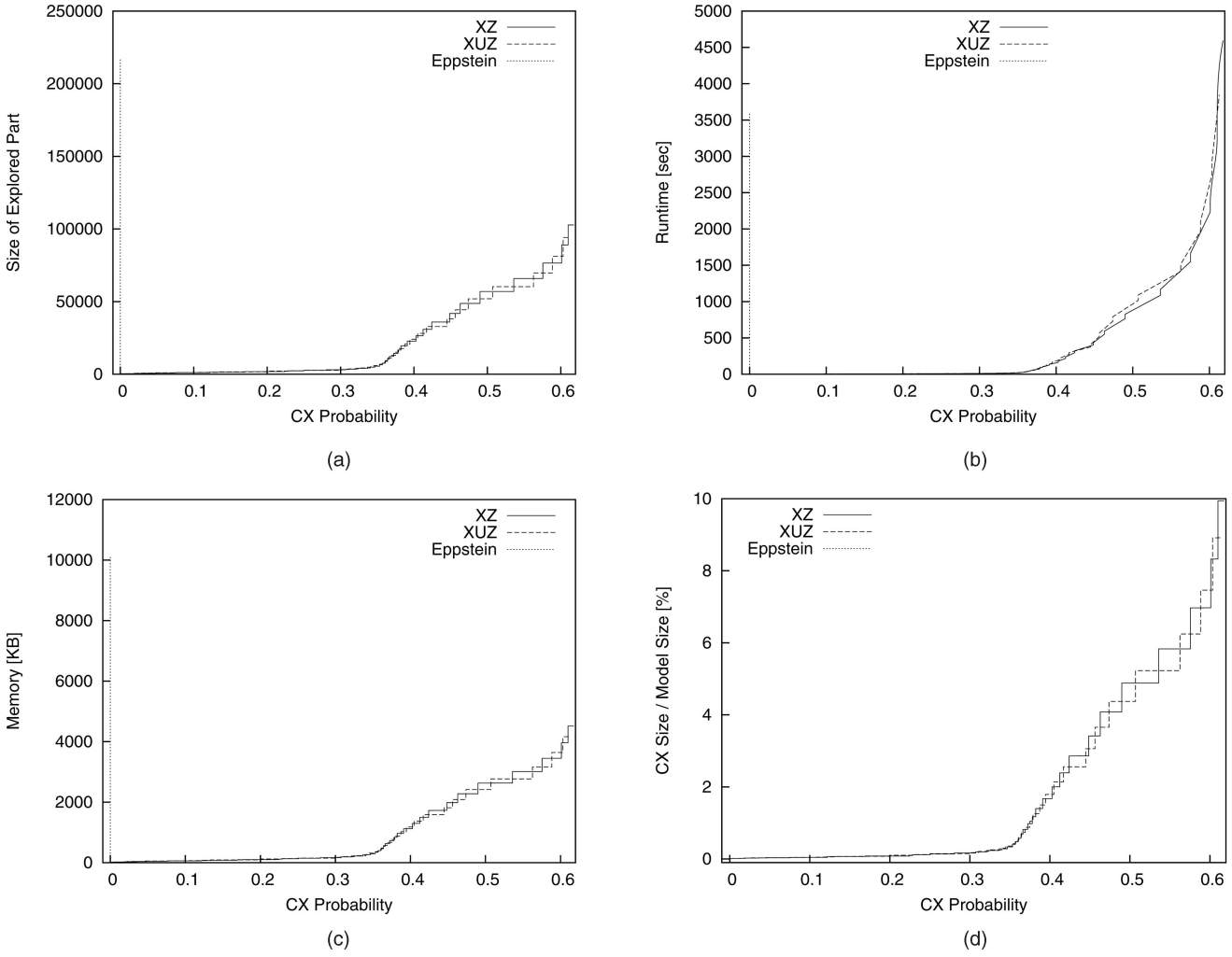

Fig. 6. Counterexample generation for $\mathcal{P}_{<0.621258247}(relocFRS2 = 0 \wedge degFRS2 = 0 \wedge degFGFR = 0\ U\ relocFRS2 = 1)$ on FGF. (a) Size of the explored STG. (b) Runtime. (c) Memory consumption. (d) Counterexample quality.

model, the duration of a DTMC transition is 0.5 milliseconds. PRISM checked the property within 0.5 seconds and consumed 132.4 KB of memory. It computed the probability $Pr(true\ U^{\leq 800}\ lost = 100)$ as 0.014725297. We applied XZ, XUZ, and Eppstein to generate counterexamples for the property $\mathcal{P}_{<0.014725297}(true\ U^{\leq 800}\ lost = 100)$. We used a heuristic h which simply returns a value from the interval $]0, 1]$ that is proportional to the number of lost requests. The results are illustrated in Fig. 7. Note that we are analyzing a time-bounded property defined on a DTMC. This means that XZ and XUZ use f_t from (4) as an evaluation function. Both algorithms are bounded to a maximal depth of 800.

In all plots in Fig. 7, the curves of Eppstein run almost parallel to the Y-axis for a counterexample probability close to 0. Within one hour of running and after exploring the STG (with approximately 14,000 states and transitions), Eppstein could provide counterexamples for probability upper bounds up to 4.298 E-25, which is insignificant compared to the total probability of 0.014725297. XZ and XUZ succeeded with a reasonable computational effort to provide counterexamples for all possible probability upper bounds. They required about 3 seconds and about 300 KB of main memory.

Fig. 7a shows that the heuristic helped XZ to find the first increment of the counterexample after exploring a slightly smaller part of the STG than XUZ. However, the first counterexample increment found by XUZ carried a probability greater than 0.009, which is higher than the probability of that found by XZ. XUZ seems to have included a portion of the STG, for instance, a cycle, which increased the counterexample probability. The heuristic deflected XZ from exploring this portion while trying to guide it toward the most probable diagnostic traces. Both algorithms found a second increment of the counterexample after exploring roughly 2,000 states and transitions. The second increment completed the counterexample such that it covered the total probability. For all probability bounds, except the range between 0.006 and 0.01, XZ provided a counterexample by exploring less states and transitions than XUZ. Nevertheless, we did not observe here a significant advantage when using the heuristic. We even see that XZ provided counterexamples which are mostly larger than the ones provided by XUZ. This means that the heuristic had a negative impact on the counterexample quality.

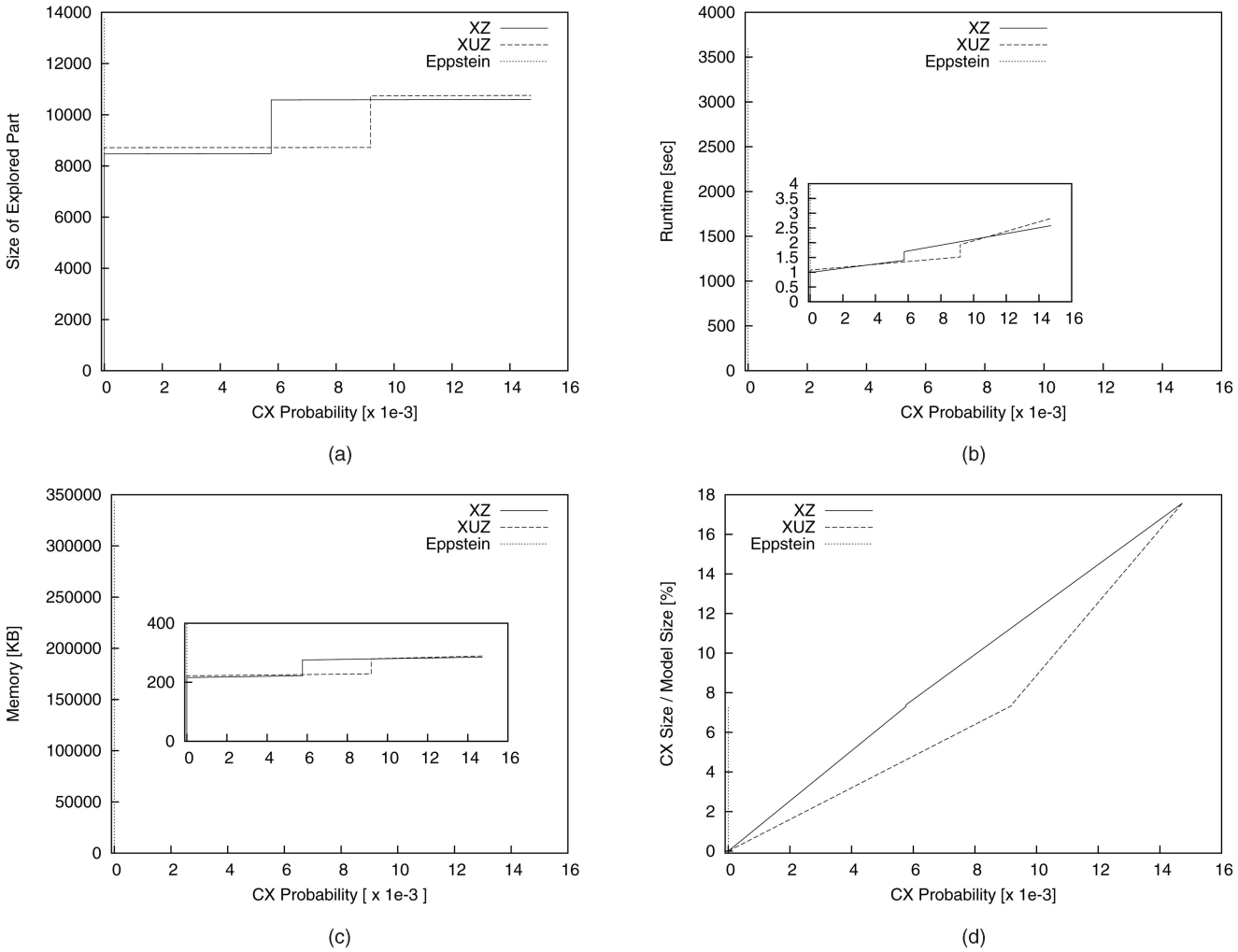

Fig. 7. Counterexample generation for $\mathcal{P}_{<0.014725297}(true\ U^{\leq 800}\ lost = 100)$ on power. (a) Size of the explored STG. (b) Runtime. (c) Memory consumption. (d) Counterexample quality.

5.5 Tandem Queuing Network

This model, which we simply refer to as *Tandem*, describes a simple tandem queuing network, taken from [49]. The queuing network consists of an $M/Cox2/1$-queue sequentially composed with an $M/M/1$-queue. We set the capacity of the queues to 127 (parameter c = 127) which results in a CTMC with 32,640 states and 113,283 transitions.

We are interested in the probability that both queues get full within 100 time units. We inspect the path formula $(true\ U^{\leq 100}\ both\_queues\_full)$. PRISM computes

$$Pr(true\ U^{\leq 100} both\_queues\_full) = 8.9536$$

E-43. We generated counterexamples for the property

$$\mathcal{P}_{\leq 8.9536\ E-43}(true\ U^{\leq 100}\ both\_queues\_full).$$

As a heuristic function h, we use in XZ and XZ_π the following estimate. For an arbitrary state s, let n_1 and n_2 be the current size of both queues. We roughly estimated the probability to add $(2 \cdot c - n_1 - n_2)$ elements into the first queue and multiply it with the estimate of the probability to forward $(c - n_2)$ elements to the second queue.

The results of counterexample generation are illustrated in Fig. 8. We see in Fig. 8a that XZ and XUZ managed to provide counterexamples for all probability upper bounds by exploring less than half, or even just a quarter in the case of XZ, of the STG. Eppstein could not finish the state space exploration within the given time budget. The curves of XZ_π and XUZ_π overlap with the curves of XZ, XUZ, and Eppstein up to approximately 8,500 states and transitions. This means that within one hour XZ_π and XUZ_π could not explore more of the STG because they were overloaded by handling the large $\underline{\pi}$-vectors. Remember that the length of the $\underline{\pi}$-vectors depends on the uniformization rate of the model and the time bound of the property. The uniformization rate of this CTMC (524.280) is very large compared to the models Cluster (41.057) and Embedded (0.085) for which XZ_π and XUZ_π scaled relatively better. The length of the $\underline{\pi}$-vectors was 4,578 in Cluster and 448 in Embedded, whereas it was 54,013 in Tandem. Thus, the computation of f_π in this case was too expensive. In summary, XZ_π, XUZ_π, and Eppstein failed, within one hour, to provide a counterexample for any probability bound. We see also that XZ outmatches XUZ significantly in terms of both search effort and counterexample quality. This means that guiding the search using a heuristic led to a significant advantage in this case study.

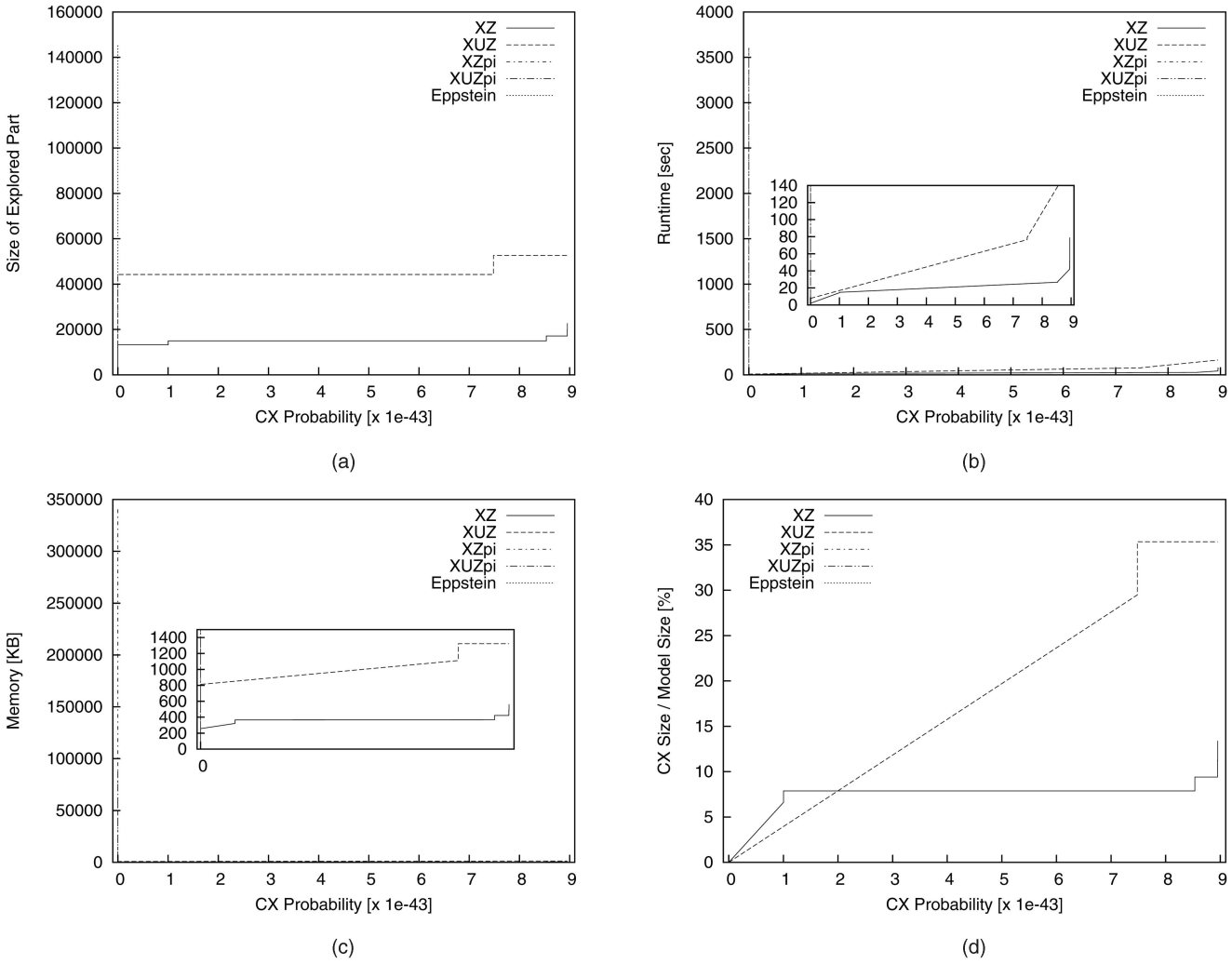

Fig. 8. Counterexample generation for $\mathcal{P}_{\leq 8.9536\,E-43}(true\ U^{\leq 100}\ both\_queues\_full)$ on tandem. (a) Size of explored part. (b) Runtime. (c) Memory consumption. (d) Counterexample quality.

5.6 Scalability When Varying Model Size

We consider several model variants of the Cluster and Tandem models which differ significantly in size in order to assess how the scalability of our algorithms depends on the model size. We scaled the size of the Cluster model by varying the parameter N, which gives the number of workstations in each subcluster. In the Tandem model, we considered several values of the parameter c, which defines the capacity of each queue. Table 1 summarizes the data of the different model variants and the model checking of the corresponding properties. We give the uniformization rate in the table because it, together with the time bound, rules the length of the $\underline{\pi}$-vectors in XZ_π and XUZ_π.

We ran the various counterexample generation algorithms on all models and recorded the results for Cluster models in Table 2 and for Tandem in Table 3. For each experiment, we report the result for the probability upper bounds 10, 40, 80, and 100 percent (cf., Column **B** in each table) of the total model probability, which is given in Table 1. Table cells with a content of the pattern "$? > x$", for some value x, mean that the algorithm failed to provide a counterexample for the corresponding bound although it achieved x, i.e., it explored already x states and transitions,

TABLE 1
Characteristics of Variants with Different Sizes of the Models Cluster and Tandem

Cluster *(true $U^{\leq 100}$ ¬ minimum)*			
N	**16**	**64**	**256**
States	10,132	151,060	2,373,652
Transitions	48,160	733,216	11,583,520
Uniformization	40.861	41.057	41.841
MC Probability	4.99E-5	5.02E-5	5.52E-5
MC Time [sec]	4.623	95.465	595.965
MC Memory [KB]	414.9	5,077.4	62,739.6
Tandem *(true $U^{\leq 100}$ both_queues_full)*			
c	**31**	**127**	**511**
States	2,016	32,640	523,776
Transitions	6,819	113,283	1,829,379
Uniformization	132.60	524.280	2,091.0
MC Probability	2.14E-9	8.95E-43	9.54E-208
MC Time [sec]	1.712	116.383	16,261.065
MC Memory [KB]	84.5	1,309.7	14,075.7

TABLE 2
Counterexample Generation for $\mathcal{P}_{<p}(true\ U^{\leq 100}\ \neg minimum)$ on Cluster with Different Sizes

N	B	XZ	XUZ	XZπ	XUZπ	Eppstein
16	10	80	122	80	122	19,266
	40	100	146	100	146	19,266
	80	212	229	212	229	19,266
	100	4,567	4,387	? > 6,208	? > 5,596	*19,266
64	10	154	225	154	225	271,632
	40	249	266	249	266	*271,632
	80	593	569	536	577	*271,632
	100	20,955	20,207	? > 5,456	? > 6,139	*271,632
256	10	616	819	775	803	? ≥ 549,650
	40	1,498	1,441	1,403	1,453	? ≥ 549,650
	80	4,074	4,348	4,279	4,499	? ≥ 549,650
	100	132,424	146,893	? > 13,775	? > 14,503	? ≥ 549,650

(a)

N	B	XZ	XUZ	XZπ	XUZπ	Eppstein
16	10	0.507	0.384	5.449	9.530	13.975
	40	0.654	0.542	6.907	11.905	17.899
	80	1.447	0.998	15.773	19.370	444.376
	100	33.311	27.019	? > 1h	? > 1h	? > 1h
64	10	0.678	0.532	12.444	21.405	1,389.210
	40	1.307	0.774	21.459	26.000	? > 1h
	80	2.732	1.992	51.218	61.052	? > 1h
	100	62.165	50.069	? > 1h	? > 1h	? > 1h
256	10	1.465	1.521	100.172	124.342	? > 1h
	40	4.105	3.415	191.338	233.905	? > 1h
	80	10.308	10.307	633.589	775.211	? > 1h
	100	540.476	580.442	? > 1h	? > 1h	? > 1h

(b)

N	B	XZ	XUZ	XZπ	XUZπ	Eppstein
16	10	3.0	4.5	503.6	755.4	785.2
	40	3.7	5.3	629.5	863.5	787.9
	80	7.7	8.3	1,223.4	1,313.3	1,019.1
	100	154.4	148.2	? > 20,665.4	? > 15,825.6	? > 2,047.91
64	10	5.6	8.2	935.3	1,348.9	10,987.9
	40	9.0	9.6	1,456.9	1,546.9	? > 11,191.1
	80	21.1	20.2	2,807.1	3,005.1	? > 11,191.1
	100	596.5	577.5	? > 24,157.1	? > 26,498.5	? > 11,191.1
256	10	22.0	29.2	4,226.9	4,406.4	? > 20,601.9
	40	52.5	50.4	7,034.9	7,286.4	? > 20,601.9
	80	139.7	148.8	18,554.4	19,560.2	? > 20,601.9
	100	4,318.4	4,784.1	? > 57,813.6	? > 58,183.9	? > 20,601.9

(c)

N	B	XZ	XUZ	XZπ	XUZπ	Eppstein
16	10	3.95E-4	6.00E-4	3.95E-4	6.00E-4	2.23E-4
	40	5.32E-4	7.72E-4	5.32E-4	7.72E-4	3.95E-4
	80	1.30E-3	1.41E-3	1.30E-3	1.41E-3	2.13E-3
	100	5.91E-2	5.67E-2	? > 8.77E-2	? > 7.81E-2	? > 3.65E-3
64	10	5.20E-5	8.03E-5	5.20E-5	8.03E-5	9.05E-5
	40	9.16E-5	9.84E-5	9.16E-5	9.84E-5	? > 2.39E-4
	80	2.90E-4	2.62E-4	2.47E-4	2.69E-4	? > 2.39E-4
	100	1.78E-2	1.68E-2	? > 4.02E-3	? > 4.44E-3	? > 2.39E-4
256	10	1.67E-5	2.24E-5	2.19E-5	2.17E-5	?
	40	5.04E-5	4.71E-5	4.59E-5	4.60E-5	?
	80	1.54E-4	1.70E-4	1.65E-4	1.67E-4	?
	100	7.94E-3	8.85E-3	? > 5.56E-4	? > 5.94E-4	?

(d)

(a) Size of explored STG. (b) Runtime in seconds. (c) Memory consumption in kilobytes. (d) Counterexample quality.

TABLE 3
Counterexample Generation for $\mathcal{P}_{<p}(true\ U^{\leq 100}\ both\_queues\_full)$ on Tandem with Different Sizes

c	B	XZ	XUZ	XZπ	XUZπ	Eppstein
31	10	1,331	3,499	1,331	3,484	*8,832
	40	1,487	3,502	1,487	3,487	*8,832
	80	1,663	4,174	1,663	4,121	*8,832
	100	2,764	4,875	2,764	4,840	*8,832
127	10	13,222	44,229	? > 8,757	? > 8,401	*145,920
	40	14,971	44,232	? > 8,757	? > 8,401	*145,920
	80	14,983	44,241	? > 8,757	? > 8,401	*145,920
	100	22,718	52,631	? > 8,757	? > 8,401	*145,920
511	10	163,733	573,363	? > 3,505	? > 4,522	*2,353,152
	40	189,634	573,363	? > 3,505	? > 4,522	*2,353,152
	80	189,634	? > 573,363	? > 3,505	? > 4,522	*2,353,152
	100	? > 189,634	? > 573,363	? > 3,505	? > 4,522	*2,353,152

(a)

c	B	XZ	XUZ	XZπ	XUZπ	Eppstein
31	10	1.203	1.704	206.433	576.153	? > 1h
	40	1.712	1.704	259.858	577.144	? > 1h
	80	2.278	3.310	286.095	683.362	? > 1h
	100	5.445	5.229	435.368	798.974	? > 1h
127	10	14.354	76.550	? > 1h	? > 1h	? > 1h
	40	26.732	76.551	? > 1h	? > 1h	? > 1h
	80	26.734	76.554	? > 1h	? > 1h	? > 1h
	100	79.038	162.704	? > 1h	? > 1h	? > 1h
511	10	1,309.195	24,737.464	? > 1h	? > 1h	? > 1h
	40	5,748.411	24,737.463	? > 1h	? > 1h	? > 1h
	80	5,748.406	? > 6.8h	? > 1h	? > 1h	? > 1h
	100	? > 1.6h	? > 6.8h	? > 1h	? > 1h	? > 1h

(b)

c	B	XZ	XUZ	XZπ	XUZπ	Eppstein
31	10	32.0	87.6	14,463.8	8,092.3	? > 6386.63
	40	36.0	87.7	14,522.0	8,092.4	? > 6386.63
	80	40.5	104.5	14,525.3	8,104.2	? > 6386.63
	100	68.4	122.4	14,658.8	8,117.9	? > 6386.63
127	10	322.3	1,110.7	? > 340,856.8	? > 87,991.9	? > 8,931.5
	40	366.7	1,110.7	? > 340,856.8	? > 87,991.9	? > 8,931.5
	80	367.0	1,111.0	? > 340,856.8	? > 87,991.9	? > 8,931.5
	100	562.7	1,322.7	? > 340,856.8	? > 87,991.9	? > 8,931.5
511	10	4,024.8	14,425.0	? > 922,954.8	? > 1,046,517.9	? > 69,927.0
	40	4,677.0	14,425.0	? > 922,954.8	? > 1,046,517.9	? > 69,927.0
	80	4,677.0	? > 14,425.0	? > 922,954.8	? > 1,046,517.9	? > 69,927.0
	100	? > 4,677.0	? > 14,425.0	? > 922,954.8	? > 1,046,517.9	? > 69,927.0

(c)

c	B	XZ	XUZ	XZπ	XUZπ	Eppstein
31	10	9.13E-2	3.68E-1	9.13E-2	3.66E-1	? > 6.24E-2
	40	1.09E-1	3.69E-1	1.09E-1	3.66E-1	? > 6.24E-2
	80	1.30E-1	4.41E-1	1.30E-1	4.37E-1	? > 6.24E-2
	100	2.60E-1	5.28E-1	2.60E-1	5.24E-1	? > 6.24E-2
127	10	6.62E-2	2.95E-1	?	?	? > 6.11E-3
	40	7.88E-2	2.95E-1	?	?	? > 6.11E-3
	80	7.89E-2	2.95E-1	?	?	? > 6.11E-3
	100	1.34E-1	3.53E-1	?	?	? > 6.11E-3
511	10	5.63E-2	2.42E-1	?	?	? > 1.32E-3
	40	6.74E-2	2.42E-1	?	?	? > 1.32E-3
	80	6.74E-2	? > 2.42E-1	?	?	? > 1.32E-3
	100	? > 6.74E-2	? > 2.42E-1	?	?	? > 1.32E-3

(d)

(a) Size of explored STG. (b) Runtime in seconds. (c) Memory consumption in kilobytes. (d) Counterexample quality.

ran for x seconds, consumed x kilobytes, or found a counterexample with a size x of the entire STG, respectively. If no counterexample is found at all, then we just put "?" in the corresponding cell in the counterexample quality table. In the tables reporting the size of the explored part of the STG, we replace "$? >$" with an asterisk "$^*x$" in the case of Eppstein's algorithm. This asterisk indicates that Eppstein has explored the complete STG and x will not increase. If we are not sure about that, we use "$? \geq x$," indicating that STG might be bigger than x.

First, we see that XZ and XUZ scaled very well for almost all model variants. An exception is the Tandem model with $c = 511$, where XZ failed to provide a counterexample with the total probability and XUZ failed to provide counterexamples for probability bounds starting from 80 percent of the total probability. This observation is

due to the high effort for model checking the provided counterexamples.

XZ_π and XUZ_π also scaled well for the Cluster model, although their computational costs were higher than that of XZ and XUZ. While XZ_π and XUZ_π failed to achieve the total model probability, they came very close to it as additional figures, which are not reported in the tables, show: For $N = 16$, XZ_π achieved 99.97 percent and XUZ_π 99.95 percent of the total probability; for $N = 64$, they achieved 99.67 percent and 99.46 percent; and for $N = 256$, they achieved 96.68 percent and 96.18 percent. XZ_π and XUZ_π did not scale in the case of the Tandem model as good as in the case of the Cluster model. While they managed to provide counterexamples for all probability bounds for $c = 31$, they failed for $c = 127$ and $c = 511$ to provide a counterexample for any probability bound. This can be explained by the fact that the uniformization rate of the Tandem model increases rapidly with the value of c, as shown in Table 1. This leads to a rapid increase in the length of the $\underline{\pi}$-vectors. Such an effect is not observable in the Cluster model.

Regarding the Eppstein-based approach, it is expected that the larger the STG is, the more search effort is required to completely explore it before any counterexamples can be provided. In Table 2a, we see that Eppstein started to provide counterexamples for the Cluster models after exploring 19,266 states and transitions for $N = 16$ and 271,632 for $N = 64$. For $N = 256$, it did not deliver any diagnostic trace after exploring 549,650 states and transitions. In the case of the Tandem model, although Eppstein succeeded in exploring the whole STG, it failed in all cases to collect enough diagnostic traces to provide a counterexample for any probability bound listed in the table. The reason is that in the Tandem model, the probability of single diagnostic paths is very low. The total probability is widely distributed on a huge number of them. This becomes clear when we consider, for instance, that Eppstein found, for $c = 31$, 13 023 diagnostic paths with an accumulated probability of 3.39 E-15 which compromises less than 0.0002 percent of the total probability.

If we compare XZ with XUZ and XZ_π with XUZ_π, then we see in the case of the Tandem model that guiding the search using a heuristic estimate (i.e., in XZ and XZ_π) significantly reduced the size of the state space portion which has to be explored to provide a counterexample, see Table 3a. This is reflected in a significant improvement in terms of runtime and memory consumption, see Tables 3b and 3c. A similar effect, while less significant, can also be observed in the case of the Cluster model. For both models, using the heuristic estimate results, in many cases, in providing smaller counterexamples. This becomes clear by comparing XZ with XUZ and XZ_π with XUZ_π in Tables 2d and 3d, respectively. We hardly notice any advantage in using XZ_π or XUZ_π instead of XZ or XUZ, either in guiding the search more effectively or improving the counterexample quality.

5.7 Scalability When Varying the Time Bound

For all algorithms, we expect additional computation effort for larger time bounds. This even holds for XZ, XUZ, and Eppstein, where the search itself is actually independent of the time bound. The reason for this is that the larger the time bound is, the more diagnostic paths are feasible and the higher the total probability. Thus, in the case of a large time bound, the search algorithm has to explore more of the STG to achieve a certain percentage of the total probability. This effect is more drastic for Eppstein's algorithm since it has to process each diagnostic trace individually. In the case of XZ_π and XUZ_π, increasing the time bound increases the length of the $\underline{\pi}$-vectors. Thus, we expect XZ_π and XUZ_π to be more sensitive regarding the time bound than XZ, XUZ, and Eppstein.

TABLE 4
Characteristics of the Models Embedded and FGF and Model Checking Results for Different Time Bounds

Embedded States: 8,548, Transitions: 3,041, Uniformization Rate: 0.085002				
T	**0.0167**	**1.0**	**100.0**	**infinity**
MC Probability	1.91E-6	1.14E-4	1.10E-2	5.46E-2
MC Time [sec]	0.288	0.568	45.957	1.426
MC Memory [KB]	315.4	315.4	315.4	492.3
FGF States: 80,616, Transitions: 562,536, Uniformization Rate: 5,249.974				
T	**1.0**	**10.0**	**100.0**	**infinity**
MC Probability	1.13E-7	5.94E-4	5.72E-2	6.21E-1
MC Time [sec]	79.599	606.829	8,052.946	12,369.239
MC Memory [KB]	3,742.3	3,742.3	3,742.3	54,975.6

For the model Embedded, we analyzed the property that only the main processor (M) is shut down within a particular time period T, expressed by the path formula $(\neg down\ U^{\leq T}\ M\_Failure)$. We formulated a heuristic function, which we used in XZ and XZ_π, according to the same idea described in Section 5.2. For the FGF model, we checked again the property that the relocation of FRS2 has occurred first, restricted to a particular time period while using the same heuristic estimate described in Section 5.3. We summarized the model characteristics and model checking results of both models in Table 4. We ran the algorithms for different time bounds and report the results in Tables 5 and 6. Both tables are organized similarly to Tables 2 and 3.

In the example Embedded, while XZ and XUZ scaled for all time bounds, we observe that the runtime and memory consumption of XZ_π and XUZ_π rapidly increase in proportion to the time bound. This rapid increase cannot be explained by the increase in the size of the STG which the algorithms have to explore. Table 5a shows that XZ_π or XUZ_π explored almost equal parts of the STG as XZ or XUZ. Further, comparing Table 5a with Tables 5b and 5c shows that the increase of runtime and memory consumption is much greater than the increase in the size of the STG portion which the algorithms explored. The reason for this rapid increase is the growing length of the $\underline{\pi}$-vectors which leads to greater computation and storage effort. Eppstein explored 19,452 states and transitions in every case, cf., Table 5a. As expected, the larger the time bound is, the more difficult it was to provide counterexamples.

In the FGF example, we can observe similar effects. However, Table 6a shows for $T = 1.0$ that XZ_π and XUZ_π

TABLE 5
Counterexample Generation for $\mathcal{P}_{<p}(\neg down\ U^{\leq T}\ M\_Failure)$ on Embedded for Different Time Bounds

T	B	XZ	XUZ	XZπ	XUZπ	Eppstein
0.0167 (1 min)	10	15	15	15	15	19,452
	40	29	42	29	42	19,452
	80	42	72	42	72	19,452
	100	265	322	265	307	*19,452
1.0	10	15	15	15	15	19,452
	40	29	42	29	42	19,452
	80	42	72	42	72	19,452
	100	678	780	661	891	*19,452
100.0	10	15	15	15	15	19,452
	40	72	72	72	72	*19,452
	80	202	233	196	210	*19,452
	100	11,207	11,409	? > 5,500	? > 5,619	*19,452
Infinity	10	72	72	-	-	*19,452
	40	678	780	-	-	*19,452
	80	3,705	3,768	-	-	*19,452
	100	14,565	14,794	-	-	*19,452

(a)

T	B	XZ	XUZ	XZπ	XUZπ	Eppstein
0.0167 (1 min)	10	0.124	0.127	0.131	0.125	17.531
	40	0.250	0.375	0.265	0.378	17.830
	80	0.368	0.623	0.393	0.639	18.113
	100	1.495	1.641	1.610	1.587	? > 1h
1.0	10	0.125	0.177	0.227	0.187	19.747
	40	0.245	0.419	0.461	0.598	22.646
	80	0.378	0.671	0.676	1.035	27.252
	100	2.437	2.695	6.341	9.053	? > 1h
100.0	10	0.165	0.193	5.113	5.054	1,002.100
	40	0.767	0.809	34.407	32.746	? > 1h
	80	1.438	1.816	97.978	102.737	? > 1h
	100	52.529	54.700	? > 1h	? > 1h	? > 1h
Infinity	10	0.753	0.759	-	-	? > 1h
	40	2.779	2.984	-	-	? > 1h
	80	9.430	9.533	-	-	? > 1h
	100	53.831	52.589	-	-	? > 1h

(b)

T	B	XZ	XUZ	XZπ	XUZπ	Eppstein
0.0167 (1 min)	10	0.6	0.6	1.5	1.5	762.0
	40	1.0	1.4	2.4	3.4	762.1
	80	1.4	2.5	3.2	5.9	762.3
	100	8.9	10.8	18.0	23.4	? > 14,732.1
1.0	10	0.6	0.6	12.8	12.8	762.4
	40	1.0	1.4	20.3	29.4	764.6
	80	1.4	2.5	25.9	51.4	769.9
	100	23.0	26.3	363.8	589.1	? > 14,952.4
100.0	10	0.6	0.6	870.9	870.9	4,302.7
	40	2.5	2.5	2,986.2	3,483.5	? > 12,914.9
	80	6.8	7.9	7,465.6	9,703.8	? > 12,914.9
	100	354.4	360.4	? > 98,495.7	? > 163,382.0	? > 12,914.9
Infinity	10	2.5	2.5	-	-	? > 11,755.2
	40	23.0	26.3	-	-	? > 11,755.2
	80	118.8	120.6	-	-	? > 11,755.2
	100	463.3	470.3	-	-	? > 11,755.2

(c)

T	B	XZ	XUZ	XZπ	XUZπ	Eppstein
0.0167 (1 min)	10	6.73E-5	6.73E-5	6.73E-5	6.73E-5	6.73E-5
	40	1.79E-4	2.92E-4	1.79E-4	2.92E-4	8.97E-5
	80	2.92E-4	5.16E-4	2.92E-4	5.16E-4	8.97E-5
	100	2.11E-3	2.53E-3	2.11E-3	2.40E-3	? > 8.97E-5
1.0	10	6.73E-5	6.73E-5	6.73E-5	6.73E-5	8.97E-5
	40	1.79E-4	2.92E-4	1.79E-4	2.92E-4	8.97E-5
	80	2.92E-4	5.16E-4	2.92E-4	5.16E-4	8.97E-5
	100	5.61E-3	6.68E-3	5.49E-3	7.65E-3	? > 8.97E-5
100.0	10	6.73E-5	6.73E-5	6.73E-5	6.73E-5	8.97E-5
	40	5.16E-4	5.16E-4	5.16E-4	5.16E-4	? > 8.97E-5
	80	1.44E-3	1.64E-3	1.50E-3	1.64E-3	? > 8.97E-5
	100	1.86E-1	1.90E-1	? > 7.74E-2	? > 7.78E-2	? > 8.97E-5
Infinity	10	5.16E-4	5.16E-4	-	-	? > 8.97E-5
	40	5.61E-3	6.68E-3	-	-	? > 8.97E-5
	80	4.58E-2	4.68E-2	-	-	? > 8.97E-5
	100	2.66E-1	2.72E-1	-	-	? > 8.97E-5

(d)

(a) Size of explored STG. (b) Runtime in seconds. (c) Memory consumption in kilobytes. (d) Counterexample quality.

TABLE 6
Counterexample Generation for $\mathcal{P}_{<p}(relocFRS2 = 0 \wedge degFRS2 = 0 \wedge degFGFR = 0\ U^{\leq T}\ relocFRS2 = 1)$ on FGF for Different Time Bounds

T	B	XZ	XUZ	XZπ	XUZπ	Eppstein
1.0	10	360	298	225	280	? ≥ 214,710
	40	1,070	1,140	439	714	? ≥ 214,710
	80	2,341	2,446	738	846	? ≥ 214,710
	100	56,955	60,285	? > 22,385	? > 21,631	? ≥ 214,710
10	10	142	173	163	194	? ≥ 214,064
	40	313	316	300	331	? ≥ 214,064
	80	540	578	1,052	953	? ≥ 214,064
	100	22,749	20,651	? > 2,767	? > 2,558	? ≥ 214,064
100.0	10	360	400	360	? > 354	? ≥ 200,631
	40	540	707	? > 375	? > 354	? ≥ 200,631
	80	1,422	1,243	? > 375	? > 354	? ≥ 200,631
	100	? > 76,546	? > 81,226	? > 375	? > 354	? ≥ 200,631
Infinity	10	834	990	-	-	? ≥ 216,638
	40	2,341	2,446	-	-	? ≥ 216,638
	80	56,955	51,920	-	-	? ≥ 216,638
	100	? > 102,770	? > 94,100	-	-	? ≥ 216,638

(a)

T	B	XZ	XUZ	XZπ	XUZπ	Eppstein
1.0	10	1.264	0.854	20.908	29.248	? > 1h
	40	3.097	3.616	45.686	90.292	? > 1h
	80	6.551	7.470	91.297	110.201	? > 1h
	100	625.226	752.152	? > 1h	? > 1h	? > 1h
10	10	0.321	0.366	156.832	178.322	? > 1h
	40	1.517	1.436	279.619	328.919	? > 1h
	80	3.229	3.634	1,386.997	1,437.745	? > 1h
	100	204.484	179.091	? > 1h	? > 1h	? > 1h
100.0	10	4.057	5.097	3,538.028	? > 1h	? > 1h
	40	8.993	12.949	? > 1h	? > 1h	? > 1h
	80	25.831	22.895	? > 1h	? > 1h	? > 1h
	100	? > 1h	? > 1h	? > 1h	? > 1h	? > 1h
Infinity	10	3.115	3.648	-	-	? > 1h
	40	7.616	8.580	-	-	? > 1h
	80	1,085.215	1,016.053	-	-	? > 1h
	100	? > 1.2h	? > 1h	-	-	? > 1h

(b)

T	B	XZ	XUZ	XZπ	XUZπ	Eppstein
1.0	10	22.2	18.3	1,683.4	1,934.2	? > 89,096.2
	40	61.0	63.9	3,093.8	4,935.9	? > 89,096.2
	80	132.3	136.6	5,185.6	5,845.5	? > 89,096.2
	100	2,631.8	2,768.4	? > 96,589.6	? > 88,345.6	? > 89,096.2
10	10	9.1	10.9	12,051.7	14,377.2	? > 10,034.9
	40	19.4	19.5	23,255.9	25,158.9	? > 10,034.9
	80	31.8	33.4	67,445.3	63,215.8	? > 10,034.9
	100	1,121.8	1,022.4	? > 174,843.5	? > 154,342.8	? > 10,034.9
100.0	10	22.2	24.4	273,172.8	? > 266,964.6	? > 9,479.6
	40	31.8	40.8	? > 283,520.3	? > 266,964.6	? > 9,479.6
	80	81.0	70.7	? > 283,520.3	? > 266,964.6	? > 9,479.6
	100	? > 3,449.4	? > 3,643.3	? > 283,520.3	? > 266,964.6	? > 9,479.6
Infinity	10	48.1	56.9	-	-	? > 10,143.9
	40	132.3	136.6	-	-	? > 10,143.9
	80	2,631.8	2,413.5	-	-	? > 10,143.9
	100	? > 4,514.9	? > 4,162.9	-	-	? > 10,143.9

(c)

T	B	XZ	XUZ	XZπ	XUZπ	Eppstein
1.0	10	1.37E-4	8.71E-5	1.07E-4	1.34E-4	?
	40	4.26E-4	4.48E-4	1.94E-4	2.91E-4	?
	80	1.18E-3	1.24E-3	3.56E-4	4.28E-4	?
	100	4.88E-2	5.23E-2	? > 1.98E-2	? > 1.96E-2	?
10	10	4.51E-5	4.51E-5	6.53E-5	6.53E-5	?
	40	1.14E-4	1.09E-4	1.09E-4	1.21E-4	?
	80	2.43E-4	2.53E-4	5.83E-4	5.15E-4	?
	100	1.67E-2	1.50E-2	? > 1.56E-3	? > 1.37E-3	?
100.0	10	1.37E-4	1.55E-4	1.37E-4	? > 1.31E-4	?
	40	2.43E-4	3.08E-4	? > 1.37E-4	? > 1.31E-4	?
	80	6.76E-4	5.97E-4	? > 1.37E-4	? > 1.31E-4	?
	100	? > 6.97E-2	? > 7.46E-2	? > 1.37E-4	? > 1.31E-4	?
Infinity	10	3.56E-4	3.70E-4	-	-	?
	40	1.18E-3	1.24E-3	-	-	?
	80	4.88E-2	4.37E-2	-	-	?
	100	? > 9.94E-2	? > 8.92E-2	-	-	?

(d)

(a) Size of explored part. (b) Runtime. (c) Memory consumption. (d) Counterexample quality.

TABLE 7
Counterexample Generation for $\mathbf{P}_{\leq p}(true\ U^{\leq 100}\ both\_queues\_full)$ and $\mathbf{P}_{\leq p}(true\ U^{\leq 100}\ first\_queues\_full)$ on Tandem ($c = 127$)

(true $U^{\leq 100}$ both_queues_full) [total prob. = 8.9536E-43]

B	XZ	XUZ	XZπ	XUZπ	Eppstein
10	13,222	44,229	? > 8,757	? > 8,401	*145,920
40	14,971	44,232	? > 8,757	? > 8,401	*145,920
80	14,983	44,241	? > 8,757	? > 8,401	*145,920
100	22,718	52,631	? > 8,757	? > 8,401	*145,920

(true $U^{\leq 100}$ first_queues_full) [total prob. = 0.99999]

B	XZ	XUZ	XZπ	XUZπ	Eppstein
10	759	759	759	759	145,287
40	766	761	766	761	145,287
80	1,289	1,557	1,289	1,557	145,287
100	6,196	7,304	6,304	6,197	*145,287

(a)

(true $U^{\leq 100}$ both_queues_full) [total prob. = 8.9536E-43]

B	XZ	XUZ	XZπ	XUZπ	Eppstein
10	14.354	76.550	? > 1h	? > 1h	? > 1h
40	26.732	76.551	? > 1h	? > 1h	? > 1h
80	26.734	76.554	? > 1h	? > 1h	? > 1h
100	79.038	162.704	? > 1h	? > 1h	? > 1h

(true $U^{\leq 100}$ first_queues_full) [total prob. = 0.99999]

B	XZ	XUZ	XZπ	XUZπ	Eppstein
10	0.264	0.254	102.944	97.034	27.146
40	0.265	0.254	103.830	97.624	28.333
80	1.491	2.005	193.454	220.101	159.042
100	8.985	10.383	1,252.758	1,164.716	? > 1h

(b)

(true $U^{\leq 100}$ both_queues_full) [total prob. = 8.9536E-43]

B	XZ	XUZ	XZπ	XUZπ	Eppstein
10	322.3	1,110.7	? > 340,856.8	? > 87,991.9	? > 8,931.5
40	366.7	1,110.7	? > 340,856.8	? > 87,991.9	? > 8,931.5
80	367.0	1,111.0	? > 340,856.8	? > 87,991.9	? > 8,931.5
100	562.7	1,322.7	? > 340,856.8	? > 87,991.9	? > 8,931.5

(true $U^{\leq 100}$ first_queues_full) [total prob. = 0.99999]

B	XZ	XUZ	XZπ	XUZπ	Eppstein
10	17.8	17.8	53,335.9	53,335.9	4,309.3
40	18.0	17.9	53,546.5	53,335.9	4,310.8
80	30.6	37.3	71,886.5	79,906.9	4,453.7
100	154.7	182.5	81,487.9	79,994.2	? > 7,126.7

(c)

(true $U^{\leq 100}$ both_queues_full) [total prob. = 8.9536E-43]

B	XZ	XUZ	XZπ	XUZπ	Eppstein
10	6.62E-2	2.95E-1	?	?	? > 6.11E-3
40	7.88E-2	2.95E-1	?	?	? > 6.11E-3
80	7.89E-2	2.95E-1	?	?	? > 6.11E-3
100	1.34E-1	3.53E-1	?	?	? > 6.11E-3

(true $U^{\leq 100}$ first_queues_full) [total prob. = 0.99999]

B	XZ	XUZ	XZπ	XUZπ	Eppstein
10	1.75E-3	1.75E-3	1.75E-3	1.75E-3	1.75E-3
40	1.77E-3	1.77E-3	1.77E-3	1.77E-3	1.81E-3
80	4.16E-3	5.74E-3	4.16E-3	5.74E-3	6.15E-3
100	3.84E-2	4.59E-2	3.86E-2	3.84E-2	? > 7.83E-3

(d)

(a) Size of explored part. (b) Runtime. (c) Memory consumption. (d) Counterexample quality.

explored often much less of the STG than XZ and XUZ for the same probability bound. This is due to the fact that f_π was a better guide than f. However, the advantage of the better guide was ruined by the computational costs of f_π as we can see in Tables 6b and 6c. For small time bounds, the use of f_π offers an advantage with respect to the counterexample quality (see Table 6d for $T = 1.0$).

5.8 Impact of Low and High Probabilities

In general, algorithms need more time and more memory to compute counterexamples with higher probabilities. However, the probability is sometimes associated with a few highly probable diagnostic traces. In such cases, XBF algorithms would quickly find these traces and cover the whole probability. In other words, the level of probability does not affect the performance of the algorithms. Instead, the relevant factor is how this probability is distributed on the STG of the model. We illustrate this effect using the following experiment. We considered the Tandem model with $c = 127$. We analyzed two different properties, namely, the one used in Section 5.5 as well as the property stating the probability that the first queue becomes full within 100 time units. Checking both properties on the model using PRISM results in a very low probability of 8.9536 E-43 for the first property, and a very high probability of 0.99999 for the second one.

The results of the counterexample generation are reported in Table 7. We observe that all algorithms performed better with the second property. From Table 7a, we see that, in particular, XBF-based algorithms explored in the case of the second property much less of the STG than in the case of the first one. For the first property, XZ explored 22,718 and XUZ explored 52,631 states and transition while XZ_π and XUZ_π failed even to explore the needed part. Meanwhile, all XBF-based algorithms provided a counterexample for the second property with 100 percent of the total probability after exploring between 6,614 and 7,304 states and transitions. The reason for this is that the total probability of the first property is distributed over a large number of diagnostic traces and that each of these traces has a very low probability. Thus, the algorithms had to explore more from the STG to cover the needed diagnostic traces in order to form a counterexample with a certain probability. Eppstein explored almost the same number of states and transitions for both properties. It succeeded in providing counterexamples for a probability of up to 80 percent of the total probability in the case of the second property. However, it failed to provide a counterexample with any noticeable probability for the first property. The explanation for this phenomenon is that Eppstein's algorithm was engaged in collecting the diagnostic traces with low individual probabilities. For example, it collected 2,887 diagnostic paths in the case of the first property. The accumulated probability of these paths is ignorable. On the other hand, the 2,887 diagnostic paths first selected for the second property carry about 85 percent of the total probability.

5.9 Summary

We illustrated in our experiments that XBF-based algorithms allow for a very efficient and scalable counterexample generation. The KSP-based approach, which needs to explore the whole STG before it can provide any counterexamples, is only applicable to small models. We observed that the usage of f_π improved the counterexample quality in some cases but its computation is very expensive. In contrast to XZ and XUZ, XZ_π and XUZ_π were either inefficient or failed to provide the desired results. This occurred particularly in the case of models with large uniformization rate and properties with large time bounds. On the other hand, XZ and XUZ scaled for very large models and for very large time bounds. From the practical point of view, we recommend using XZ and XUZ, although the evaluation function f_π is theoretically interesting. We also saw that guiding the search using a heuristic function h in many cases results in an improvement in terms of performance and counterexample quality. In some cases, this improvement is so significant that it determines the scalability of the algorithm, for instance, in the Tandem model with $c = 511$.

6 Conclusion

In this paper, we have presented a heuristics-guided method to generate counterexamples for the model checking of PCTL formulas on DTMCs and CSL ones on CTMCs. For this purpose, we have developed an advanced heuristic search strategy called XBF which extends the framework presented in [24]. XBF is instantiated to concrete algorithms, e.g., XZ and XZ_π. The approach that we have proposed reconciles stochastic model checking with directed explicit-state search and illustrates the potential as well as the possible shortcomings of this approach.

We have evaluated our method using a number of experiments on significant case studies. In conclusion, the experiments illustrate that:

1. The provided counterexamples cover the most probable diagnostic traces.
2. The provided counterexamples are usually very small compared to the entire model. Hence, they can be effectively analyzed for the purpose of debugging.
3. In contrast to KSP-based approach, our approach scales for very large models.

Acknowledgments

This work was carried out in the course of the DFG-funded research project DiRePro. The authors wish to thank Holger Hermanns for inspiring discussions that led to the precursory work as documented in the paper [24].

References

[1] E.M. Clarke, O. Grumberg, and D.A. Peled, *Model Checking*, third ed. MIT Press, 2001.
[2] G.J. Holzmann, *The SPIN Model Checker: Primer and Reference Manual*. Addision-Wesley, 2003.
[3] E.M. Clarke and E.A. Emerson, "Design and Synthesis of Synchronization Skeletons Using Branching-Time Temporal Logic," *Logic of Programs*, pp. 52-71, Springer, 1981.
[4] E. Clarke, S. Jha, Y. Lu, and H. Veith, "Tree-Like Counterexamples in Model Checking," *Proc. 17th Ann. IEEE Symp. Logic in Computer Science*, pp. 19-29, 2002.
[5] A. Groce, S. Chaki, D. Kroening, and O. Strichman, "Error Explanation with Distance Metrics," *Int'l J. Software Tools for Technology Transfer*, vol. 8, no. 3, pp. 229-247, 2006.
[6] A. Groce and W. Visser, "What Went Wrong: Explaining Counterexamples," *Proc. Workshop Software Model Checking*, pp. 121-135, 2003.
[7] S. Edelkamp, S. Leue, and A. Lluch-Lafuente, "Directed Explicit-State Model Checking in the Validation of Communication Protocols," *Int'l J. Software Tools for Technology Transfer*, vol. 5, nos. 2/3, pp. 247-267, 2004.
[8] N.J. Nilsson, *Principles of Artificial Intelligence*. Tioga, 1980.
[9] J. Pearl, *Heuristics—Intelligent Search Strategies for Computer Problem Solving*. Addision-Wesley, 1986.
[10] W.J. Stewart, *Introduction to the Numerical Solution of Markov Chains*. Princeton Univ. Press, 1994.
[11] H. Hansson and B. Jonsson, "A Logic for Reasoning about Time and Reliability," *Formal Aspects of Computing*, vol. 6, no. 5, pp. 512-535, 1994.
[12] A. Aziz, K. Sanwal, V. Singhal, and R. Brayton, "Model-Checking Continuous-Time Markov Chains," *ACM Trans. Computational Logic*, vol. 1, no. 1, pp. 162-170, 2000.
[13] C. Baier, B. Haverkort, H. Hermanns, and J.-P. Katoen, "Model-Checking Algorithms for Continuous-Time Markov Chains," *IEEE Trans. Software Eng.*, vol. 29, no. 6, pp. 524-541, June 2003.
[14] A. Hinton, M.Z. Kwiatkowska, G. Norman, and D. Parker, "PRISM: A Tool for Automatic Verification of Probabilistic Systems," *Proc. 12th Int'l Conf. Tools and Algorithms for the Construction and Analysis of Systems*, pp. 441-444, 2006.
[15] J.-P. Katoen, M. Khattri, and I.S. Zapreev, "A Markov Reward Model Checker," *Proc. Second Int'l Conf. Quantitative Evaluaiton of Systems*, pp. 243-244, 2005.
[16] H.L.S. Younes, "Ymer: A Statistical Model Checker," *Proc. 17th Int'l Conf. Computer Aided Verification*, pp. 429-433, July 2005.
[17] T. Hérault, R. Lassaigne, and S. Peyronnet, "APMC 3.0: Approximate Verification of Discrete and Continuous Time Markov Chains," *Proc. Third Int'l Conf. Quantitative Evaluation of Systems*, pp. 129-130, 2006.
[18] C. Courcoubetis and M. Yannakakis, "Verifying Temporal Properties of Finite-State Probabilistic Programs," *Proc. 29th Ann. Symp. Foundations of Computer Science*, pp. 338-345, 1988.
[19] C. Courcoubetis and M. Yannakakis, "Markov Decision Processes and Regular Events (Extended Abstract)," *Proc. 17th Int'l Colloquium Automata, Languages and Programming*, pp. 336-349, 1990.
[20] C. Courcoubetis and M. Yannakakis, "The Complexity of Probabilistic Verification," *J. ACM*, vol. 42, no. 4, pp. 857-907, 1995.
[21] A. Aziz, V. Singhal, and F. Balarin, "It Usually Works: The Temporal Logic of Stochastic Systems," *Proc. Seventh Int'l Conf. Computer Aided Verification*, pp. 155-165, 1995.
[22] A. Bianco and L. de Alfaro, "Model Checking of Probabalistic and Nondeterministic Systems," *Proc. 15th Conf. Foundations of Software Technology and Theoretical Computer Science*, pp. 499-513, 1995.
[23] A. Aziz, K. Sanwal, V. Singhal, and R.K. Brayton, "Verifying Continuous Time Markov Chains," *Proc. Eighth Int'l Conf. Computer Aided Verification*, pp. 269-276, 1996.
[24] H. Aljazzar, H. Hermanns, and S. Leue, "Counterexamples for Timed Probabilistic Reachability," *Proc. Third Int'l Conf. Formal Modeling and Analysis of Timed Systems*, pp. 177-195, 2005.
[25] H. Aljazzar and S. Leue, "Extended Directed Search for Probabilistic Timed Reachability," *Proc. Fourth Int'l Conf. Formal Modeling and Analysis of Timed Systems*, pp. 33-51, 2006.
[26] T. Han, J.-P. Katoen, and B. Damman, "Counterexample Generation in Probabilistic Model Checking," *IEEE Trans. Software Eng.*, vol. 35, no. 2, pp. 241-257, Mar./Apr. 2009.
[27] T. Han and J.-P. Katoen, "Providing Evidence of Likely Being on Time: Counterexample Generation for ctmc Model Checking," *Proc. Fifth Int'l Symp. Automated Technology for Verification and Analysis*, pp. 331-346, 2007.
[28] H. Fecher, M. Huth, N. Piterman, and D. Wagner, "Hintikka Games for PCTL on Labeled Markov Chains," *Proc. Fifth Int'l Conf. Quantitative Evaluation of Systems*, pp. 169-178, 2008.

[29] M.E. Andrés, P.R. D'Argenio, and P. van Rossum, "Significant Diagnostic Counterexamples in Probabilistic Model Checking," *ACM Computing Research Repository,* vol. abs/0806.1139, 2008.
[30] H. Hermanns, B. Wachter, and L. Zhang, "Probabilistic CEGAR," *Proc. 20th Int'l Conf. Computer Aided Verification,* pp. 162-175, 2008.
[31] H. Aljazzar and S. Leue, "Debugging of Dependability Models Using Interactive Visualization of Counterexamples," *Proc. Fifth Int'l Conf. Quantitative Evaluation of Systems,* 2008.
[32] M.Y. Vardi, "Automatic Verification of Probabilistic Concurrent Finite-State Programs," *Proc. 26th Ann. Symp. Foundations of Computer Science,* pp. 327-338, 1985.
[33] S.E. Dreyfus, "An Appraisal of Some Shortest Path Algorithms," *Proc. Operations Research Soc. of Am./The Inst. of Management Sciences Joint Nat'l Mtg.,* vol. 16, p. 166, 1968.
[34] D. Eppstein, "Finding the k Shortest Paths," *SIAM J. Computing,* vol. 28, no. 2, pp. 652-673, http://dx.doi.org/10.1137/S0097539795290477, 1998.
[35] V.M. Jiménez and A. Marzal, "Computing the k Shortest Paths: A New Algorithm and an Experimental Comparison," *Algorithm Eng.,* pp. 15-29, 1999.
[36] A. Martelli, "On the Complexity of Admissible Search Algorithms," *Artificial Intelligence,* vol. 8, no. 1, pp. 1-13, 1977.
[37] R.E. Korf and M. Reid, "Complexity Analysis of Admissible Heuristic Search," *Proc. 15th Nat'l Conf. Artificial Intelligence and 10th Innovative Applications of Artificial Intelligence Conf.,* pp. 305-310, 1998.
[38] R.E. Korf, M. Reid, and S. Edelkamp, "Time Complexity of Iterative-Deepening-A*," *Artificial Intelligence,* vol. 129, nos. 1/2, pp. 199-218, 2001.
[39] S. Kupferschmid, K. Dräger, J. Hoffmann, B. Finkbeiner, H. Dierks, A. Podelski, and G. Behrmann, "Uppaal/DMC—Abstraction-Based Heuristics for Directed Model Checking," *Proc. 13th Int'l Conf. Tools and Algorithms for the Construction and Analysis of Systems,* pp. 679-682, 2007.
[40] W.K. Grassmann, "Transient Solutions in Markovian Queuing Systems," *Computers and OR,* vol. 4, no. 1, pp. 47-53, 1977.
[41] W.K. Grassmann, "Transient Solutions in Markovian Queuing Systems," *Computers and OR,* vol. 5, no. 2, p. 161, 1978.
[42] D. Gross and D.R. Miller, "The Randomization Technique as a Modeling Tool and Solution Procedure for Transient Markov Processes," *Operations Research,* vol. 32, no. 2, pp. 343-361, 1984.
[43] R.E. Tarjan, "Depth-First Search and Linear Graph Algorithms," *SIAM J. Computing,* vol. 1, no. 2, pp. 146-160, 1972.
[44] B.R. Haverkort, H. Hermanns, and J.-P. Katoen, "On the Use of Model Checking Techniques for Dependability Evaluation," *Proc. 19th IEEE Symp. Reliable Distributed Systems,* pp. 228-237, 2000.
[45] J. Muppala, G. Ciardo, and K. Trivedi, "Stochastic Reward Nets for Reliability Prediction," *Comm. in Reliability, Maintainability and Serviceability,* vol. 1, no. 2, pp. 9-20, July 1994.
[46] J. Heath, M.Z. Kwiatkowska, G. Norman, D. Parker, and O. Tymchyshyn, "Probabilistic Model Checking of Complex Biological Pathways," *Proc. Int'l Conf. Computational Methods in Systems Biology,* pp. 32-47, 2006.
[47] J. Heath, M.Z. Kwiatkowska, G. Norman, D. Parker, and O. Tymchyshyn, "Probabilistic Model Checking of Complex Biological Pathways," *Theoretical Computer Science,* vol. 391, no. 3, pp. 239-257, 2008.
[48] L. Benini, A. Bogliolo, G.A. Paleologo, and G.D. Micheli, "Policy Optimization for Dynamic Power Management," *IEEE Trans. Computer-Aided Design of Integrated Circuits and Systems,* vol. 18, no. 6, pp. 813-833, 1999.
[49] H. Hermanns, J.-P. Katoen, J. Meyer-Kayser, and M. Siegle, "A Markov Chain Model Checker," *Proc. Tools and Algorithms for Construction and Analysis of Systems,* pp. 347-362, 2000.

Husain Aljazzar received the degree of Diplom-Informatiker from the University of Marburg, Germany, in 2004. He is currently a research and teaching assistant at the chair for software engineering at the University of Konstanz, Germany. His research interests include methods for the design of dependable software systems including stochastic model checking and heuristic search.

Stefan Leue received the PhD degree in computer science from the University of Berne. He is a professor for computer science in the Department of Computer and Information Science of the University of Konstanz, where he holds the chair for software engineering. His research interests include formal methods for the design and verification of complex software systems, in particular, model checking. He has held faculty positions at the University of Waterloo and the University of Freiburg, as well as visiting positions at Bell Laboratories, Université Joseph Fourier, and ETH Zurich. He currently serves as the chair of the steering committee of the SPIN International Workshop Series. He is a senior member of the IEEE and a member of the ACM and the GI.