Extended Directed Search for Probabilistic Timed Reachability

Husain Aljazzar and Stefan Leue

Department of Computer and Information Science
University of Konstanz, Germany
{Husain.Aljazzar,Stefan.Leue}@uni-konstanz.de

Abstract. Current numerical model checkers for stochastic systems can efficiently analyse stochastic models. However, the fact that they are unable to provide debugging information constrains their practical use. In precursory work we proposed a method to select diagnostic traces, in the parlance of functional model checking commonly referred to as failure traces or counterexamples, for probabilistic timed reachability properties on discrete-time and continuous-time Markov chains. We applied directed explicit-state search algorithms, like Z*, to determine a diagnostic trace which carries large amount of probability. In this paper we extend this approach to determining sets of traces that carry large probability mass, since properties of stochastic systems are typically not violated by single traces, but by collections of those. To this end we extend existing heuristics guided search algorithms so that they select sets of traces. The result is provided in the form of a Markov chain. Such diagnostic Markov chains are not just essential tools for diagnostics and debugging but, they also allow the solution of timed reachability probability to be approximated from below. In particular cases, they also provide real counterexamples which can be used to show the violation of the given property. Our algorithms have been implemented in the stochastic model checker PRISM. We illustrate the applicability of our approach using a number of case studies.

1 Introduction

Motivation. Software debugging is an important task in the design, implementation and integration of systems. In particular Model Checking techniques have recently been used extensively to aid the developer in fixing errors. To this end it is necessary that Model Checkers provide meaningful debugging information. In Model Checking of functional properties such information can be made available without additional computational cost. In the case of a safety property violation, Model Checkers like SPIN [1] deliver a single linear failure trace from the initial state to a property violating state that may later be used in locating the cause of a property violation. In model checking parlance such a failure trace is called a *counterexample* to the desired safety property. To obtain short and therefore easy to comprehend counterexamples, search techniques such as *Breadth-First Search* (BFS) or *Directed Model Checking* (DMC) [2], which relies on heuristics guided state space search, can be employed.

Performance and dependability models are usually represented as stochastic models describing how the system changes from state to state as time passes. In the presence of stochastic models we are not just interested in detecting functional failure behavior of the system but in the quantitative analysis of its dependability and performance. We use the terms *target state* for states which we are interested in, i.e. states satisfying a given state proposition, and *diagnostic traces* for traces leading to target states. As in the functional

setting, DMC algorithms can be employed in the Model Checking of safety properties to select diagnostic traces that are meaningful in the fault localization process. However, contrary to the functional setting, in the stochastic context we are faced with two main challenges. First, indicative of the quality of a diagnostic trace is not its length, but its probability mass. Hence, in order to use heuristics guided search techniques it is necessary to find a new quality measure based on the probability mass of traces as well as heuristics functions based on this measure that steer the search along traces with high probability mass. We first addressed this problem in [3]. Second, one diagnostic trace is in general not enough to provide meaningful error information for explaining why some probabilistic safety property is satisfied or not since all diagnostic traces contribute jointly to the probability of the property. Thus, the developer needs to consider a reasonably large set of diagnostic traces in order to debug the model. Obviously, the more probability this set carries, the more expressive it is. In this paper we address this challenge using advanced heuristic guided algorithms which make it possible to incrementally select a set of diagnostic traces with a high probability mass. This set forms a Markov chain which emulates the original model with respect to the given property.

In our approach, the set of diagnostic traces is incrementally selected. Its probability mass gradually grows during the search process with every iteration. However, it can always be ensured to be a lower bound of the total probability of the given property. In other words, the total probability of the given property to be satisfied is approximated from below. In particular cases, our method can be used to generate a counterexample which shows the violation of the given property. In this case a set of traces is computed whose probability is not smaller than the given probability upper bound. In order to repair the model, the developer has to consider the computed set. That is because, it is not possible to decrease the total probability to be under the given upper bound without applying changes to this part of the model.

Related Work and State of the Art. Many heuristic strategies and algorithms have been introduced to solve problems of, amongst others, graph search and optimization. In [4], Pearl has given a widespread overview of a set of general-purpose problem solving strategies, e.g. *Best First (BF)* and *Generalized Best First (GBF)*. Also a variety of specialized directed search algorithms, e.g. A* and Z*, have been proposed. An approach how to apply heuristics guided directed search algorithms to functional explicit state Model Checking, especially for the generation of counterexamples, has been presented in [2]. Discrete- and continuous-time Markovian models, e.g. Markov chains and Markov decision processes, build a very important and widely used class of stochastic models [5–7]. Prevalent stochastic Model Checkers, like PRISM [8], ETMCC [9] and its successor MRMC [10], apply efficient numerical methods to analyze Markovian models with respect to performance and dependability properties expressed in a stochastic temporal logic, like the *Probabilistic CTL* (PCTL) [11] or the *Continuous Stochastic Logic* (CSL) introduced in [12] and extended in [13]. These numerical approaches reach a high degree of numerical result accuracy. However, they are memory intensive because they keep the whole state space of the model in memory. Another disadvantage of these approaches is their inability to deliver debugging information, in particular diagnostic traces. Some approaches attempt to reduce the memory consumption of stochastic Model Checking using Monte-Carlo sampling methods [14–16]. Our method and these approaches have the generation of explicit paths through the stochastic model in common. However, while the goal of these approaches is to perform the stochastic model checking using Monte-Carlo sampling, ours is to dissect a meaningful portion of the model. In our own precursory work we proposed an approach based on Directed Model Checking to select a diagnostic trace which carries a high probability for a given probabilistic

safety property specified on a *discrete-time Markov chain* [3]. We have also proposed an approximation based on uniformization to deal with *continuous-time Markov chains*. Our approach is memory saving because it is performed on the fly.

Structure of the Paper. In Section 2 we introduce the stochastic models which we use as well as related notations and further preliminaries. Section 3 gives an overview on directed stochastic algorithms. In Section 4 we introduce our new algorithms and discuss their properties. Section 5 contains case studies and experimental evaluation of the approach. We conclude the paper in Section 6.

2 Stochastic Systems

2.1 Markov Chains

A *discrete-time Markov chains (DTMC)* is a probabilistic transition system consisting of states and transitions between them. Each transition is labeled with a numerical value called transition probability. It indicates the probability for firing this transition as the next step of the system if the system is in the origin state of the transition. Formally, we define a DTMC as follows:

Definition 1 *A labeled discrete-time Markov chain (DTMC) $\mathcal{D}$ is a quadruple (S, s_{init}, P, L), where*

- *S is a finite set of states*
- *$s_{init} \in S$ is an initial state*
- *$P : S \times S \longrightarrow [0, 1]$ is a probability matrix, satisfying that for each state s, $\sum_{s' \in S} P(s, s') = 1$.*
- *$L : S \longrightarrow 2^{AP}$ is a labeling function, which assigns each state a subset of the set of atomic propositions AP. We interpret this to define the set of valid propositions in the state.*

For each pair of states s and s', $P(s, s')$ gives the probability to move from s to s'. For a pair of states s and s', a transition from s to s' is possible if and only if $P(s, s') > 0$ holds. A state s is called absorbing if $P(s, s) = 1$ and consequently, $P(s, s') = 0$ for all other states $s' \neq s$.

Example 1. Figure 1 illustrates a simple DTMC $\mathcal{D} = (S, s_0, P, L)$ with $S = \{s_0, s_1, s_2\}$. The probability matrix P is given on the left hand side of the figure. On the right hand side of the figure, the DTMC is illustrated as a state transition graph. The labeling function L can be, for example, defined as:

$$L = \{(s_0, \{idle\}), (s_1, \{active\}), (s_2, \{broken\})\},$$

where $AP = \{idle, active, broken\}$. The state s_2 is obviously absorbing.

Because of their simplicity, DTMCs are widely used in the modeling and analysis of stochastic systems. However, as the name already indicates, time is assumed to be discrete. If a more realistic modeling is required, then *continuous-time Markov chains (CTMCs)* are used. While each transition of a DTMC corresponds to a discrete time-step, in a CTMC transitions occur in real time. Instead of transition probability, each transition is labeled by a rate defining the delay which occurs before it is taken. The probability of a transition

$$P = \begin{pmatrix} 0.0 & 1.0 & 0.0 \\ 0.9 & 0.0 & 0.1 \\ 0.0 & 0.0 & 1.0 \end{pmatrix}$$

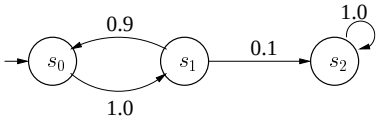

Fig. 1. A simple DTMC.

from s to some state s' being taken within t time units is described by a random variable which follows a negative exponential distribution with the transition rate as a parameter. To simplify matters, we illustrate our approach on DTMCs. However, we note that our approach can deal with CTMCs due to a uniformization based approximation presented in [3].

2.2 Paths and Traces

We now define the notions of *paths* and *traces* frequently used in this paper when talking about the probability mass of system execution. In this section, let $\mathcal{D} = (S, s_{init}, P, L)$ be a DTMC.

Definition 2 *An infinite path through $\mathcal{D}$ is a sequence $s_0 \rightarrow s_1 \rightarrow s_2 \rightarrow \ldots$ with, for $i \in \mathbb{N}$, $s_i \in S$ and $P(s_i, s_{i+1}) > 0$. A finite path is a finite sequence $s_0 \rightarrow s_1 \rightarrow \ldots s_{l-1} \rightarrow s_l$ with, for all $i \in \{0, \ldots, l\}$, $s_i \in S$, $P(s_i, s_{i+1}) > 0$ for all $i < l$ and s_l is absorbing.*

$Paths^{\mathcal{D}}$ denotes the set of all (finite and infinite) paths through $\mathcal{D}$. For some state s, $Paths^{\mathcal{D}}(s)$ denotes the set of all paths starting in s. For some path σ, define $states(\sigma)$ as the set of all states and $trans(\sigma)$ as the set of all transitions appearing in σ. $length(\sigma)$ is the number of transitions, i.e. $length(\sigma) = |trans(\sigma)|$. For a number i, with $0 \leq i \leq length(\sigma)$, $\sigma[i]$ is the i-th state of σ and $\sigma \uparrow i$ refers to the finite prefix of σ of the length i. The notations given in the paragraph are defined on finite prefixes as on paths.

Example 2. For the path $\sigma = s_0 \rightarrow s_1 \rightarrow s_0 \rightarrow s_1 \rightarrow s_2$ through the DTMC from Example 1, we get $states(\sigma) = \{s_0, s_1, s_2\}$ and $trans(\sigma) = \{(s_0, s_1), (s_1, s_0), (s_1, s_2))\}$. $length(\sigma)$ is 4. $\sigma[0]$, as well as $\sigma[2]$, is s_0, and $\sigma[4]$ is s_2. The 0th finite prefix of σ is s_0 and the 3rd is $s_0 \rightarrow s_1 \rightarrow s_0 \rightarrow s_1$.

The probability measure Pr of paths is defined in a standard manner on the smallest σ-algebra on $Paths^{\mathcal{D}}(s_{init})$ generated by sets of paths with a common finite prefix as follows:

$$Pr(\{\sigma \in Paths^{\mathcal{D}}(s_{init}) \mid \sigma \uparrow n = s_0 \rightarrow s_1 \rightarrow \ldots \rightarrow s_n\}) = P(s_0, s_1) \cdot \ldots \cdot P(s_{n-1}, s_n),$$

where s_0 is just another reference to the initial state, i.e. $s_0 = s_{init}$. Obviously, the set $\{\sigma \in Paths^{\mathcal{D}}(s_{init}) \mid \sigma \uparrow n = s_0 \rightarrow s_1 \rightarrow \ldots \rightarrow s_n\}$ is completely characterized by the prefix $s_0 \rightarrow s_1 \rightarrow \ldots \rightarrow s_n$. Thus, we rewrite the equation above replacing the set by the prefix itself: $Pr(s_0 \rightarrow s_1 \rightarrow \ldots \rightarrow s_n) = P(s_0, s_1) \cdot \ldots \cdot P(s_{n-1}, s_n)$.

Definition 3 *A trace is a finite sequence of states $< s_0, s_1, \ldots, s_n >$, including the self loops of all states $s_0, \ldots, s_n$, where, for all $i \in \{0, \ldots, n-1\}$, it holds that $s_i \neq s_{i+1}$ and $P(s_i, s_{i+1}) > 0$.*

For a given trace $R =< s_0, s_1, \ldots, s_n >$, we define the following notation: $first(R) = s_0$, $last(R) = s_n$, $states(R) = \{s_0, s_1, \ldots, s_n\}$, $trans(R) = \{(s_i, s_{i+1}) \mid 0 \leq i < n\} \cup \{(s_i, s_i) \mid 0 \leq i \leq n \ \wedge \ P(s_i, s_i) > 0\}$ and $length(R) = |states(R)| - 1$. We refer to the set of all traces through the DTMC $\mathcal{D}$ as $Traces^{\mathcal{D}}$. For a pair of states s and s', $Traces^{\mathcal{D}}(s, s')$ refers to the set containing all traces through $\mathcal{D}$ from s to s', i.e. $Traces^{\mathcal{D}}(s, s') = \{R \in Traces^{\mathcal{D}} \mid first(R) = s \ \wedge \ last(R) = s'\}$. If $\mathcal{D}$ is clear from the context we omit the respective superscript.

We point out that, using the notation of traces, we abstract from the number of repetitions of a cycle. Repeating the cycles in a concrete way results in a concrete finite prefix which induces a set of paths. Thus, each trace R can be considered as a compact representation of a set of paths $Paths(R)$ which we formally define as follows:

$$\begin{aligned} Paths(R) \ := \{ \ \sigma \in Paths(first(R)) \ \mid \ & \exists n \in \mathbb{N} : \ last(\sigma \uparrow n) = last(R) \\ & \wedge \ states(\sigma \uparrow n) = states(R) \\ & \wedge \ trans(\sigma \uparrow n) \subseteq trans(R) \ \}. \end{aligned} \tag{1}$$

This means that each path σ from $Paths(R)$ starts with a finite prefix from $first(R)$ to $last(R)$ which hits each state from R using exclusively transitions from R. Note that not all transitions of R have to appear in the path σ. Consequently, σ does not have to contain all self loops of R. We are usually interested in paths from $Paths(R)$ which reach $last(R)$ before a given time bound T. We call such paths *time bounded* and refer to this subset of $Paths(R)$ as $Paths(R, T)$. Formally, $Paths(R, T)$ is defined as follows: $Paths(R, T) \ := \{ \ \sigma \in Paths(R) \mid \exists n \leq T : \ last(\sigma \uparrow n) = last(R) \ \}$. Note that $Paths(R)$ is equal to $Paths(R, \infty)$.

Example 3. $R_1 =< s_0, s_1, s_2 >$ and $R_2 =< s_0, s_1, s_0, s_1, s_2 >$ are examples for traces in the DTMC $\mathcal{D}$ described in Example 1. The sets of paths induced by R_1 and R_2 for the time bound 4 are:

$$\begin{aligned} Paths(R_1, 4) = \quad & \{\sigma \in Paths(s_0) \mid \sigma \uparrow 2 = s_0 \rightarrow s_1 \rightarrow s_2\} \\ Paths(R_2, 4) = \quad & \{\sigma \in Paths(s_0) \mid \sigma \uparrow 2 = s_0 \rightarrow s_1 \rightarrow s_2\} \\ & \cup \{\sigma \in Paths(s_0) \mid \sigma \uparrow 4 = s_0 \rightarrow s_1 \rightarrow s_0 \rightarrow s_1 \rightarrow s_2\} \end{aligned}$$

Note that, $Paths(R_1, 4)$ is a subset of $Paths(R_2, 4)$.

Traces are relevant because search algorithms act on the state transition graph of the model and deliver, as will be shown later, traces as a result. Thus, it is important to define a mass to measure the stochastic quality of traces. For this purpose, we consider each trace as a set of paths. Accordingly, for a trace R and time bound T, we define a function ψ as follows:

$$\psi(R, T) = Pr(Paths(R, T)). \tag{2}$$

This definition presumes that $first(R)$ is a start state of the DTMC. Intuitively, ψ is the probability mass of the set of time bounded paths induced by the trace R.

Example 4. For the trace R_1 and R_2 from Example 3, we compute $\psi(R_1, 4)$ and $\psi(R_2, 4)$ as follows:

$$\begin{aligned} \psi(R_1, 4) &= Pr(Paths(R_1, 4)) = Pr(s_0 \rightarrow s_1 \rightarrow s_2) = 1.0 \cdot 0.1 = 0.1 \\ \psi(R_2, 4) &= Pr(Paths(R_2, 4)) = Pr(s_0 \rightarrow s_1 \rightarrow s_2) + Pr(s_0 \rightarrow s_1 \rightarrow s_0 \rightarrow s_1 \rightarrow s_2) \\ &= 1.0 \cdot 0.1 + 1.0 \cdot 0.9 \cdot 1.0 \cdot 0.1 = 0.19 \end{aligned}$$

Using the function ψ, we are now able to compare traces with respect to their stochastic quality. For two traces, the one with the higher ψ value is considered to be superior to the one with the lower value. Accordingly, we define the optimality of traces as follows.

Definition 4 *Let M be a set of traces. We call a trace $R \in M$ optimal over M, iff for any other trace $R' \in M$ the following inequality holds: $\psi(R,T) \geq \psi(R',T)$.*

We call traces, which do not contain cycles except for self loops, *forward traces*. More precisely, a trace $R =< s_0, s_1, \ldots, s_n >$ is a forward trace iff $\forall i,j \in \{0,1,\ldots,n\} : s_i = s_j \Rightarrow i = j$.

3 Directed Algorithms for Probabilistic Timed Reachability

3.1 Probabilistic Timed Reachability Properties

In our approach we address an important class of properties for stochastic systems namely *probabilistic timed reachability* (PTR) properties. Such properties express a constraint on the probability of reaching some state satisfying a given state proposition φ within a given time period T. We restrict ourselves to φ being state propositions and call states satisfying φ *target states*, and we write $s \models \varphi$ for any target state s. PTR properties are safety properties. They are widely used in specifying dependability and performance properties of systems. PTR properties can be cast as instances of the following pattern: *"What is the probability to reach a state satisfying a state proposition φ within time period T?* In stochastic temporal logics like PCTL [11] or CSL [13], such properties are formulated as follows: $\mathcal{P}(\Diamond^{\leq T}\ \varphi)$. We note that after a slight technical modification our method can easily be applied to time-bounded *Until* formulas of the form $\mathcal{P}(\varphi_1\ U_{\leq\ T}\ \varphi_2)$.

For a given DTMC $\mathcal{D} = (S, s_{init}, P, L)$, $\mathcal{P}(\Diamond^{\leq T}\ \varphi)$ computes the probability mass of the set of all paths through $\mathcal{D}$ which lead to any target state within the time period T. Technically, $\mathcal{P}(\Diamond^{\leq T}\ \varphi)$ is determined based on the computation of the corresponding *transient probability*. For a given state $s \in S$ and a time point t, the transient probability $\pi(s,t)$ is the probability to be in s exactly at the time point t. Formally, $\pi(s,t) := \Pr\{\sigma \in Paths^{\mathcal{D}}(s_{init}) \mid \sigma@t = s\}$. The procedure for the computation of $\mathcal{P}(\Diamond^{\leq T}\ \varphi)$ consists of two steps. First, the stochastic model checker renders all target states absorbing, i.e. for each target state s all outgoing transitions are ignored and a self loop with transition probability 1 is added to s. Second, the model checker computes the transient probability of all target states at the time point T as follows:

$$Prob(\Diamond^{\leq T}\ \varphi) := \sum_{s \models \varphi} \pi(s,T) \tag{3}$$

Note that in the modified model the system can never leave a reached target state because all target states have been made absorbing. We refer to the probability defined in Equation 3, i.e. the probability of satisfying the PTR property, as the *timed reachability probability*. Occasionally it is required that the timed reachability probability is bounded by an upper or a lower bound. In these cases a bounded version of the operator $\mathcal{P}$ is used, e.g. $\mathcal{P}_{\leq p}$ or $\mathcal{P}_{\geq p}$. We use the terms *upwards* and *downwards bounded* PTR property to refer to a PTR property with a bounded $\mathcal{P}$ operator.

Diagnostics for PTR Properties. The verification of a given PTR property relies on the analysis of paths leading to target states within the given time period, as Equation 3 and the definition of π given above highlight. Thus, the developer would surely be interested in such paths when she or he is debugging the model with respect to a given PTR property. We call such paths *diagnostic paths*. However, a single path has a very low probability mass compared to the whole probability of the property. In the case of a CTMC, the probability of a single path is even zero. For this reason, we consider *diagnostic traces* instead of *diagnostic paths*.

Definition 5 *For a PTR property $\mathcal{P}(\Diamond^{\leq T}\ \varphi)$ specified on a DTMC $\mathcal{D} = (S, s_{init}, P, L)$, a diagnostic trace R is an element from the set $\bigcup_{s \models \varphi} \{R \in Traces^{\mathcal{D}}(s_{init}, s) \mid length(R) \leq T\}$.*

Intuitively, a diagnostic trace R is a trace from the initial state to any target state with the length of at most $T^{\star}$. It can be easily shown that $Paths(R, T)$ contains only diagnostic paths.

3.2 Best-First Search (BF)

Most of the prevalent path directed search algorithms are based on a common optimizing search strategy called Best-First (BF) [4]. This strategy uses a hash table called *CLOSED* to collect information about all explored states which have been expanded, i.e. their successors have been generated. We call such states *closed* states. Additionally, it uses a priority queue called *OPEN* to store all explored states which have not been expanded yet. We call such states *open* states. The queue *OPEN* is organized as a priority queue which is sorted according the "quality" of states. This quality is estimated numerically by an *evaluation function* f. Amongst others, f usually depends on local state information, e.g. values of the state variables, and information gathered by the search up to that point. f may also depend on the specification of the target, in our case the given PTR property, as well as further knowledge about the problem domain that may be available, in which case we call the resulting algorithm *informed* or *directed*. This knowledge is mostly encoded in a heuristic function h which is used by the evaluation function f. In each iteration BF expands the optimal open state s, i.e. the state from *OPEN* with the best f value, moves it from *OPEN* to *CLOSED*. Each successor state s' is checked on being a target state. If s' is a target state, the algorithm terminates with the solution. Otherwise, s is put into *OPEN*. Once *OPEN* is empty, the algorithm terminates without a solution. The strategy BF* is derived from BF strategy by a slight modification called *termination delay*. The termination delay means that the termination of the algorithm is delayed until a target state is selected for expansion. In both strategies BF and BF*, the evaluation function f remains arbitrary. Both strategies do not specify how f is computed. This is a major issue which has a significant impact on the search effort and the solution's nature. BF and BF* are instantiated to concrete algorithms by specifying a concrete evaluation function f or requiring f to satisfy particular conditions. For instance, the prominent A* algorithm is obtained from BF* if an additive computation of the path length is used as an evaluation function f [4]. If the evaluation function f is computed recursively, then BF becomes Z and BF* becomes Z* [4]. More precisely, in Z or Z*, when a state s is expanded, for a successor state s', $f(s')$ is computed by an arbitrary combination of the form $f(s') = F[\chi(s), f(s), h(s')]$, where $\chi(s)$ is a set of local parameters characterizing s, e.g. the weight of the transition (s, s').

3.3 Stochastic Directed Search Algorithms

In [3] we have shown how to explore the state space of a given Markov chain using (directed) search algorithms in order to generate a diagnostic trace for a given PTR property. To this end, we proposed a stochastic evaluation function for DTMCs based on the stochastic quality of traces. To be able to deal with CTMCs we proposed an approximation based on uniformization (see Section 2.1). To facilitate understanding the algorithms used in this approach we briefly recall, in this section, the algorithms which we used in [3].

* If $length(R)$ is greater that T, then $Paths(R, T)$ is empty. Consequently, $\psi(R, T)$ is equal to zero.

Stochastic Evaluation Function. The search algorithm spans a tree called search tree. For each explored state s there exists exactly one trace R leading from the initial state to s. We define a new function γ as follows:

$$\gamma(s) := \psi(R, T), \tag{4}$$

where T is the time bound in the considered PTR property and ψ is as defined in Equation 2. Additionally, we expect a heuristic function h which estimates the stochastic quality of the optimal diagnostic trace starting in the current state, i.e. the sate we are computing the f value for. More precisely, for a given state s, let R^* be an optimal diagnostic trace starting in s, i.e. R^* is optimal over the set of all diagnostic traces starting in s (see Definition 4). $h(s)$ estimates the stochastic quality of R^*, i.e. the value $\psi(R^*, T)$, which means that we act as if s were the initial state. Note that $h(s)$ can only give a heuristic estimate based on the description of s, information gathered by the prior search and general knowledge about the problem. Although we can not give a general definition of h because it is application dependent, we proposed in [3] a method to obtain heuristic functions starting from given estimation for atomic state propositions.

The algorithms Z and Z* used in our approach use the product of γ and h as an evaluation function f. Formally, f is defined as follows:

$$f(s) := \gamma(s) \cdot h(s). \tag{5}$$

Equation 4 might induce the impression that one has to traverse the trace from a given state up to the initial state in order to compute γ for the given state. This is surely not the case. The computation is performed using information which we have gathered during the prior search and attached to the predecessor of the state which we are computing the γ value for. More concretely, we mark each open state s with a vector $\pi'(s)$ which gives the transient probabilities of s restricted on the trace from the initial state to s for the time from 0 to T. When s is expanded, for any successor state s', $\gamma(s')$ is computed according the following expression: $\gamma(s') = P(s, s') \cdot \sum_{k=0}^{t-1} \pi'(s, k)$. Altogether, f is computed recursively as follows (c.f. Section 3.2):

$$f(s') = F[\chi(s), f(s), h(s')] = F[\chi(s), h(s')] = F[\{\pi'(s), P(s, s')\}, h(s')] = \gamma(s') \cdot h(s').$$

In order to illustrate during our later experimental evaluation the advantage of using informed search approaches we introduce two auxiliary algorithms. They are derived from Z and Z* by omitting heuristic estimate function h in the evaluation function f, i.e. $f := \gamma$. As we can observe from Equation 4, γ depends only on local state information and information gathered by the prior search. We call the resulting algorithms undirected Z (UZ) and undirected Z* (UZ*). If we use BF with just h, as described above, as an evaluation function, we get a greedy algorithm. The optimality of this algorithm is not guaranteed in general. However, it has usually a very good performance in terms of runtime as well as memory consumption. In the remainder of the paper, we refer to this algorithm simply as Greedy.

4 Extended Directed Search Algorithms

In this section we present a search strategy which extends the algorithms presented in our precursory work [3] and and which we discussed in Section 3. For a given PTR property, the new strategy makes it possible to generate not only one diagnostic trace but a set of diagnostic traces.

4.1 Extended Best-First Search (XBF)

We extend the Best-First strategy (BF) to a new strategy which we call *Extended Best-First* (XBF). The primary aim of this extension is to select a reasonable set of diagnostic traces which approximates the relevant behavior of the model, with respect to a given PTR property, from below. This set is helpful in diagnostics and can be used as a counterexample in the case of upwards bounded PTR properties. Algorithms based on BF explore the state transition graph of the model spanning a tree called traversal or search tree. Consequently, in the explored part of the state space, each state has exactly one parent. This excludes the possibility to accommodate cycles in the solution. Hence, the solution space is restricted to forward diagnostic traces. The idea is to develop a directed search strategy which explores the state transition graph of the model using a subgraph. We thus allow an explored state to have more than one parent. Additionally, XBF is designed so that it is able to select more than one target state. XBF is mainly obtained from BF by three modifications.

1. For each state we record all parent states which we find during the search. Therefore, we replace the single parent reference used in BF by a list *PARENTS* containing all parents detected by the search.
2. Additionally to *OPEN* and *CLOSED*, XBF maintains a list *TARGETS* which contains a reference for each target state encountered during the search.
3. XBF does not terminate when it finds the first solution. It continues the search for further solutions until the whole state space is processed or termination is explicitly requested.

The pseudo code of XBF is given in Algorithm 1. In the body of the **while**-loop (starting at code lines 3), there is no statement which directly causes a termination of the loop. Thus, the loop will run until the condition "*OPEN* is empty or termination is requested" is fulfilled. *OPEN* becomes empty when the state space is completely explored. The termination can also be explicitly requested using an arbitrary external condition which we refer to as the *external termination condition*. An example for that is that the number of the found diagnostic traces or the probability mass of the found solution exceeds a given bound. The **if**-statement starting at line 7 is used to detect new diagnostic traces. If the considered state s' is a target state or it is known, from prior search, that a target state is reachable from s', then we know that a new diagnostic trace is found. In this case, all known ancestor states of s' are marked as solution states. The code line 10 is useful to gather any information needed for the external termination condition, for example the number of found diagnostic traces is increased or/and the probability mass of the solution can be updated. Additionally, if s' is a target state, then it is also added to the *TARGETS* list. For each explored state s, a list *PARENTS* is used to keep pointers to all known parents of s (c.f. code lines 13, 14 and 16). Consequently, we are able to record all found traces leading to s including cycles.

Note that the stochastic evaluation function f from Equation 5 is defined based on the assumption that the explored part of the state transition graph is a tree, c.f. the definition of γ in Equation 4. This is not the case for XBF. Strictly speaking, we should redefine the function f taking into account that the explored state space is not a tree but a subgraph of the whole state space. Consequently, for a given state s, when a new trace leading to s is found, we should recompute $f(s)$ and correct the transient probability vector $\pi'(s)$ taking this trace into account. To do this, the old vector $\pi'(s)$ is needed. Therefore, we would have to store the vector π' not only for open but also for closed states. Additionally, $\pi'(s)$ was used to compute the transient probabilities of the successors of s. Thus, the whole explored part of the state space rooted at s would have to be re-explored in order to recompute π' for all states in that part. This would drastically decrease the performance

```
   Data: Safety property φ, the initial state s_init and a state transition relationship
   Result: A solution if any target state is reachable
 1 Initializations: OPEN ← an empty priority queue, CLOSED ← an empty hash table,
   TARGETS ← ∅ ;
 2 Insert s_init 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 |   Expand s generating all its successors ;
 6 |   foreach s' successor of s do
 7 |   |   if s' is a target state or s' is marked as a solution state then
 8 |   |   |   if s' is a target state then Insert s' into TARGETS ;
 9 |   |   |   Back track all pointers from s' up to s_init marking each touched states as a
   |   |   |   solution state ;
10 |   |   |   Signal a new diagnostic trace ;
   |   |   end
11 |   |   Compute f(s') ;
12 |   |   if s' is not already in OPEN or CLOSED then
13 |   |   |   Attach to s' a new list PARENTS ;
14 |   |   |   Add a pointer to s into PARENTS of s' ;
15 |   |   |   Insert s' into OPEN ;
   |   |   end
   |   |   else
16 |   |   |   Add a pointer to s into PARENTS of s' ;
17 |   |   |   if the newly computed f(s') is better than the old value then
18 |   |   |   |   Replace the old value of f(s') by the new one ;
19 |   |   |   |   if s' is in CLOSED then  Reopen s' (move it to OPEN) ;
   |   |   |   end
   |   |   end
   |   end
   end
20 if TARGETS is not empty then
21 |   Construct the solution
   end
22 Exit without a solution.
```

Algorithm 1: Pseudo code of Extended Best First (XBF).

of the algorithm in terms of both memory consumption and runtime. Therefore, we keep the definition of f based on the tree formed by the optimal traces. For each explored state s, let $Traces_{expl}(s_{init}, s)$ be the set of explored traces from s_{init} to s. We slightly modify the definition of γ as $\gamma(s) := \psi(R, T)$, where $R \in Traces_{expl}(s_{init}, s)$ is an optimal trace over $Traces_{expl}(s_{init}, s)$. Consequently, we reopen a state only if a better trace leading to it is found.

When the search algorithm terminates the solution is constructed using the **if**-statement at line 20. In our application, this step includes back tracking the pointers from all target states up to the initial state. Additionally, an extra absorbing state *sink* is added to substitute the remaining part of the model. Furthermore, for each state s contained in the solution, all outgoing transitions of s, which are not contained in the solution, are redirected to the *sink*. It is easy to show that the final result is a Markov chain. We call it the *diagnostic Markov chain* (DiagMC).

As mentioned before, the step of signaling a new trace at line 10 is used to gather information which is needed for the external termination condition. Hence, we can cause a termination delay, if this step is delayed until a target or a solution state is chosen to be expanded. Similar to BF*, we call the derived strategy XBF*. Similar to Greedy, Z, Z*, UZ and UZ*, we obtain XGreedy, XZ, XZ*, XUZ, XUZ* by modifying the evalution function f as descriped in Section 3.3. Note that XUZ and XUZ* are undirected algorithms. We consider them in this paper only to illustrate the advantage of using a heuristic function.

Example 5. Figure 2 shows a simple transition system. We can view it as the transition system of a DTMC. We assume that the model contains only one target state which we labeled by $\times$. The figure illustrates the incremental growth of the selected solution. At some point of the search the algorithm will find a diagnostic trace, for instance the one highlighted by bold lines in Figure 2 (a). If the algorithm is not explicitly stopped, it will continue to find more diagnostic traces. After some iterations the solution will grow into the subgraph highlighted in Figure 2 (b). The largest solution which can be found is given in Figure 2 (c). At the end of the algorithm, the solution is transformed into the diagnostic Markov chain illustrated in Figure 3. We refer to the DiagMC obtained from the largest solution as the *complete* DiagMC.

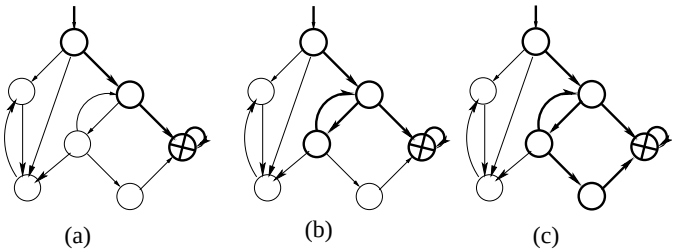

Fig. 2. Incrementally selected solution

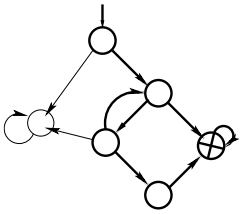

Fig. 3. The DiagMC resulting from Fig. 2 (c)

4.2 Impact of Cycles

To illustrate the effect of cycles on the timed reachability probability, we consider, again, the traces R_1 and R_2 from Example 3. Obviously, R_2, including the cycle $s_0 \rightarrow s_1 \rightarrow s_0$, has a higher probability mass than R_1. Thus, we would expect our algorithm to deliver R_2 as a solution. However, this is very difficult to realize. One reason for that is that prevalent search algorithms based on BF traverse the graph in the form of a traversal tree, i.e., each state has exactly one parent. As a consequence, cycles can not be included in the solution. If we enable the algorithm to accommodate cycles we have to allow states to have more than one parent. However, the more involved reason is the evolving complexity of the computation of f, as mentioned in Section 4.1. We try to make this point clear using the following example.

Example 6. Again, we consider the DTMC $\mathcal{D}$ given in Example 1. After the expansion of s_0, the following transient probability vectors are assigned to the states s_0 and s_1: $\pi'(s_0) = (1, 0, 0, \ldots, 0)$ and $\pi'(s_1) = (0, 1, 0, \ldots, 0)$. By the expansion of s_1, s_0 and s_2 are generated. On the one hand, $\pi'(s_1)$ is used to compute $\pi'(s_2) = (0, 0, 0.1, \ldots, 0)$ and the

cycle $s_0 \rightarrow s_1 \rightarrow s_0$ is detected. We know now that a part of the probability is circulated back to s_0. So, the vector $\pi'(s_0)$ has to be corrected to $(1, 0, 0.9, 0, \ldots, 0)$. However, the vector $\pi'(s_0)$ was used in the computation of $\pi'(s_1)$. Hence, we have to reopen the state s_0 in order to correct the vectors $\pi'(s_1)$. Then, s_1 also has to be reopened in order to correct $\pi'(s_2)$. We have to repeat this procedure until the vectors $\pi'(s_0)$, $\pi'(s_1)$ and $\pi'(s_2)$ are not changed any more.

As Example 6 illustrates, we might have to repeat the search linearly in the number of cycles and the time bound T. This would drastically decrease the performance of the algorithm in terms of both memory consumption and runtime. In order to accommodate cycles while avoiding these excessive computational costs, XBF uses the following strategy. For a state s, if a new parent of s is detected, XBF records this information adding a reference of the new parent into the *PARENT* list of s. However, s is only reopened, if the newly detected trace carries a higher probability than the old one. In Example 6, $f(s_0)$ computed regarding the newly detected trace $< s_0, s_1, s_0 >$ is $\psi(< s_0, s_1, s_0 >, 4) \cdot h(s_0) = 0.9 \cdot h(s_0)$ is less than the old value $\psi(< s_0 >) = 1.0 \cdot h(s_0)$. Thus, the vector $\pi'(s_0)$ will not be corrected and s_0 will not be reopened, although the cycle $s_0 \rightarrow s_1 \rightarrow s_0$ is included.

4.3 Under-Approximation of Timed Reachability Probability

An important contribution of this paper is that the timed reachability probability is approximated from below when using our extended algorithms. As mentioned in Section 4.1, the probability mass of the selected solution incrementally grows during the search process (see Example 5). Certainly, the probability mass of the solution highlighted in Figure 2 (c) is not smaller than that of the solution given in Figure 2 (b). To show this fact we reason as follows: The probability mass of the solution is the probability mass of the set of diagnostic paths induced by all traces contained in the solution. All diagnostic paths which are possible in (b) are also possible in (c). Hence, the set of diagnostic paths induced by the solution from (b) is contained in the set of diagnostic paths induced by the solution from (c). Similar reasoning can be used for the solution from (a) and (b).

The delivered DiagMC under-approximates the relevant behavior of the original model. The *complete* DiagMC, i.e. the DiagMC obtained from the largest solution, e.g. the DiagMC given in Figure 3, contains all diagnostic paths of the original model. As a consequence, checking the property on the complete DiagMC will deliver the same result as checking it on the original model. Normally the complete DiagMC consists of only a small portion of the complete model. Thus, checking the property on the complete DiagMC can be performed much faster than checking it on the original model. In many cases it is even not required to determine the complete DiagMC for the purposes of debugging or generating counterexamples.

Counterexamples. If the PTR property which we are interested in is upwards bounded, then our method can be used to generate a real counterexample. In this case a DiagMC is computed whose probability is not smaller than the given upper probability bound. For instance, in Example 5, if the probability mass of the solution from Fig. 2(b) is higher than the bound given in the property, then the computed DiagMC suffices to show the violation of the property. It can also be used as a counterexample. Hence, the search algorithm can be stopped at this point. In order to repair the model, the developer has to consider the computed DiagMC. Note that it is not possible to decrease the total probability to be under the given upper bound without applying changes to this part of the model. Note that during the search we are unable to compute the accurate probability mass of the currently

selected DiagMC. We can only compute an approximated value which is not higher than the accurate probability mass. The reason is that we avoid refreshing the transient probability vectors when a cycle is detected (see Section 4.2. In the case of a CTMC, the approximation using uniformization is another reason which prevents us from computing the accurate probability mass of the selected DiagMC. As a consequence, sometimes the algorithm runs longer than required for under-approximating an upwards bounded PTR.

4.4 Diagnostics of PTR Properties

As we mentioned in Section 3.1, our main goal is to obtain diagnostic tracesin order to facilitate diagnostics and debugging of PTR properties. Each diagnostic trace represents a potentially large set of diagnostic paths and it has a meaningful probability mass which is not equal to zero. A diagnostic trace with a higher probability mass is more essential in the debugging process than the one with a lower probability mass. In our precursory work we presented a method to select one forward diagnostic trace R which carries a large amount of probability. We suggested be used in debugging the system and called it a counterexample to the considered PTR property. Obviously, one trace might be insufficient to be a real counterexample to a PTR property. Even for diagnostics the developer should consider more than one critical diagnostic trace. However, in [3] we have developed the basic techniques for the exploration of state spaces of stochastic models using directed search strategies. The advanced algorithms presented in this paper make it possible to deliver more than one critical diagnostic trace represented in form of a diagnostic Markov chain.

5 Case Studies and Experimental Results

We have implemented our algorithms in Java 5.0 based on the *Data Structures Library* (JDSL) [17, 18]. Our algorithms uses the PRISM Model Checker [8] which is designed to analyze stochastic models. The DTMCs and CTMCs that we use in our experiments are modeled in the PRISM modeling language. We use the PRISM Simulation Engine in order to generate the model state spaces on-the-fly. Our search algorithms work on the thus generated state spaces. Whenever precise numerical stochastic model checking is required, for instance in order to compute total model probabilities, it is performed by the PRISM Model Checker. We next present two case studies which we used to experimentally evaluate our method. Space limitations do not permit us to present the experimental results in full. More detail is included in [19].

Case Study 1: A Query Processing System. In this section we consider a very simple model for a query processing system given as a CTMC in PRISM. The system receives queries from clients and puts them into a queue with a maximal capacity C queries where they await processing. The system works in two different modes. In the secure mode, the processing of the queries is safer but much slower than it in the normal mode. Hence, the secure mode is switched on only if it is necessary. The system is illustrated in Figure 4 in the form of a stochastic Petri net. The initial marking of this Petri net is as follows: C tokens in *Free-Slots*, one token in *Intact* and one token in *Ready*. For the maximal queue capacity $C = 500$, the CTMC consists of about 2500 states and 5000 transitions. We assume the following transition rates: $\lambda = 0.1$ is the rate to receive a new query, $\mu_1 = 3.0$ and $\mu_2 = 1.0$ are the rates for processing a query in normal and secure mode, respectively. $\kappa = 1.0 \cdot 10^{-5}$ is the rate for an attack to be successful, and $\nu = 9.0 \cdot 10^{-5}$ is the rate for an attack to be

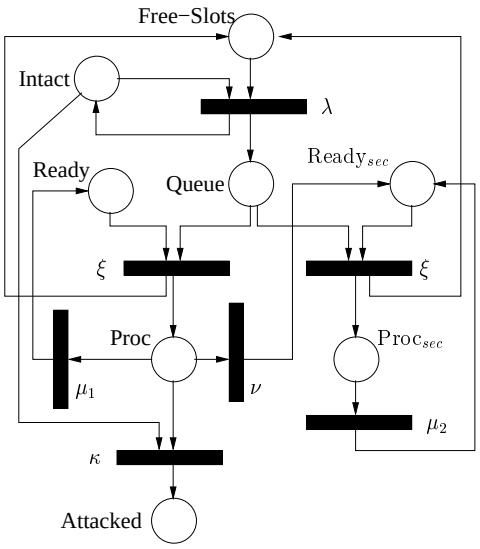

Fig. 4. SPN of the Query Processing System

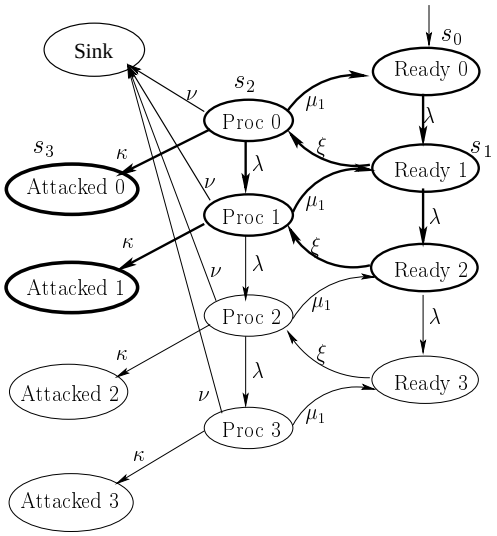

Fig. 5. The complete DiagMC in the Query Processing System

detected which causes the secure mode to be enabled. $\xi = 5.0$ is the rate to pick a query from the queue for processing.

An example of an interesting PTR property is *QPS-Attacked* which expresses the interest in the probability of the system to be conquered by an attack within a particular time period. Figure 5 shows the complete DiagMC for this property in the case that $C = 3$. The numbers by which the states are labeled indicate the number of requests in the queue.

Our experimental results showed that while Z* delivers only one diagnostic trace $< s_0, s_1, s_2, s_3 >$, which is represented in the figure by the state labels s_0 to s_3, XZ* delivers a set of diagnostic traces. For instance, if we first restrict the search to select only three diagnostic traces, XZ* delivers the DiagMC indicated by bold lines in Figure 5. If we allow the algorithm to run to the end it will select the complete DiagMC given in Figure 5 which reflects the complete failure behavior of the model. It is easy to see from this DiagMC that we have to increase the rate ν if we want to reduce the likelihood of an attack to be successful. This means we have to speed up the activation of the security component. Another approach is to perform a security check on the queue to detect attacks before they reach the processor. This results in an extra transition leading from *Ready* state to *Ready$_{sec}$* in Figure 4. It is mapped to a transition from *Ready* to *Sink* in the DiagMC in Figure 5.

When comparing the different algorithms in terms of solution quality, we observed that the solutions produced by the extended algorithms, especially XZ* and XUZ*, have a very high probability mass compared to the solutions of the basic algorithms. In some instances their probability masses were very close to the total probability measured for the model by precise model checking. This highlights the the effect of the guided under-approximation that we perform. We also observed that the non-optimal (respectively non-admissible) greedy algorithms Greedy and XGreedy delivered solutions with very low probabilities. However, these algorithms had the best computational performance in terms of both memory consumption and runtime measured in the number of search iterations. Generally, we observed that the directed algorithms Greedy, Z, Z*, XGreedy, XZ and XZ* explored much

less states and made much less search iterations than the undirected ones UZ, UZ*, XUZ and XUZ*. In the case of unreachability of target states, the directed algorithms detected that much earlier than the undirected ones.

Case Study 2: A Workstation Cluster. The second comprehensive case study that we conducted was a dependable cluster of workstations as first presented in [20]. It represents a system consisting of two sub-clusters connected via a backbone. Each sub-cluster 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 randomly. In order to provide minimum quality of service (QoS), at least k ($< N$) workstations have to be operational connected to each other via operational switches. The system is modeled as a CTMC. For the maximal number of workstations per cluster (=256) the CTMC consists of about 2.3 million states. We are interested in the likelihood that the quality of service drops below the minimum within a particular time period. We ran experiments on models for different N values and restricted the search to arbitrarily chosen number of diagnostic traces.

The DiagMCs that our analysis computes indicate the most critical portion of the failure behavior of the system. Their probability masses come very close to the full probability mass measured on the complete model. In most cases, the directed search algorithms outperformed the undirected algorithms in terms of computational cost and memory consumption. However, we observed that the performance of the undirected algorithm XUZ is often similar to the performance of XZ*. Also, the quality of solution delivered by XUZ is in some cases higher than that of the solutions delivered by XZ*. The use of the greedy algorithms further reduces the computational costs, at the expense of a loss of solution quality in the order of multiple orders of magnitude.

6 Conclusion

In this paper we have presented a heuristics guided method to generate diagnostic information for the debugging of probabilistic timed reachability properties on stochastic models. For this purpose we have developed an advanced heuristic search strategy called XBF which extends the framework presented in [3]. XBF is instantiated to concrete algorithms, namely XGreedy, XZ and XZ* as well as the undirected variants XUZ and XUZ*. We have evaluated our method using a number of experiments on two case studies. Overall, the experiments showed that 1) the DiagMCs that have been computed are meaningful and useful as diagnostic information in the analysis of the model, 2) the solution delivered by the algorithms XUZ, XZ* and XUZ* have high stochastic quality, i.e. they have high probability masses, and 3) in almost all situations the directed algorithms outperformed the undirected ones in terms of computational cost.

Currently, we are investigating how the probability vectors can be interpolated so that we can avoid to store the complete transient probability vectors during the search in order to reduce overall memory consumption of the method. We are also studying visualization techniques for state spaces in order to facilitate comprehension of the diagnostic Markov chains that are provided to the user. As future work includes the application of our method to Markov decision processes which include the concept of non-determinism that is essential in analyzing concurrent stochastic models.

Acknowledgments. We thank Marta Kwiatkowska and Holger Hermanns for enlightening discussions on the subject of this paper. We thank also Dave Parker for his advice on

choosing appropriate case studies. This work was carried out in the course of the DFG funded research project DiRePro.

References

1. Holzmann, G.J.: The Spin Model Checker: Primer and Reference Manual. Addision–Wesley (2003)
2. Edelkamp, S., Leue, S., Lluch-Lafuente, A.: Directed explicit-state model checking in the validation of communication protocols. International Journal on Software Tools for Technology Transfer STTT **5** (2004) 247–267
3. Aljazzar, H., Hermanns, H., Leue, S.: Counterexamples for timed probabilistic reachability. In Pettersson, P., Yi, W., eds.: FORMATS. Volume 3829 of Lecture Notes in Computer Science., Springer (2005) 177–195
4. Pearl, J.: Heuristics – Intelligent Search Strategies for Computer Problem Solving. Addision–Wesley (1986)
5. Feller, W.: An Introduction to Probability Theory and Its Applications. John Wiley & Sons (1968)
6. Stewart, W.J.: Introduction to the Numerical Solution of Markov Chains. Princeton University Press, New Jersey, USA (1994)
7. Kulkarni, V.G.: Modeling and analysis of stochastic systems. Chapman & Hall, Ltd., London, UK (1995)
8. Hinton, A., Kwiatkowska, M., Norman, G., Parker, D.: PRISM: A tool for automatic verification of probabilistic systems. In Hermanns, H., Palsberg, J., eds.: Proc. 12th International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS'06). Volume 3920 of LNCS., Springer (2006) 441–444
9. Hermanns, H., Katoen, J.P., Meyer-Kayser, J., Siegle, M.: A markov chain model checker. In: Tools and Algorithms for Construction and Analysis of Systems. (2000) 347–362
10. Katoen, J.P., Khattri, M., Zapreev, I.S.: A markov reward model checker. qest **0** (2005) 243–244
11. Hansson, H., Jonsson, B.: A logic for reasoning about time and reliability. Formal Asp. Comput. **6** (1994) 512–535
12. Aziz, A., Sanwal, K., Singhal, V., Brayton, R.: Model-checking continuous-time markov chains. ACM Trans. Comput. Logic **1** (2000) 162–170
13. Baier, C., Haverkort, B., Hermanns, H., Katoen, J.P.: Model-checking algorithms for continuous-time Markov chains. IEEE Transions on Software Engineering **29** (2003)
14. Grosu, R., Smolka, S.A.: Monte carlo model checking. In Halbwachs, N., Zuck, L.D., eds.: TACAS. Volume 3440 of Lecture Notes in Computer Science., Springer (2005) 271–286
15. Sen, K., Viswanathan, M., Agha, G.: On statistical model checking of stochastic systems. In Etessami, K., Rajamani, S.K., eds.: CAV. Volume 3576 of Lecture Notes in Computer Science., Springer (2005) 266–280
16. Younes, H.L.S., Simmons, R.G.: Probabilistic verification of discrete event systems using acceptance sampling. In Brinksma, E., Larsen, K.G., eds.: CAV. Volume 2404 of Lecture Notes in Computer Science., Springer (2002) 223–235
17. Goodrich, M., Tamassia, R.: Data Structures and Algorithms in Java (2nd edition). John Wiley & Sons, Inc., New York, NY, USA (2000)
18. JDSL Web Page: (http://www.cs.brown.edu/cgc/jdsl/)
19. Aljazzar, H., Leue, S.: Extended directed search for probabilistic timed reachability. Technical Report soft-06-03, Chair for Software Engineering, University of Konstanz, Gemany (2006) URL: http://www.inf.uni-konstanz.de/soft/research/publications/pdf/soft-06-03.pdf.
20. Haverkort, B.R., Hermanns, H., Katoen, J.P.: On the use of model checking techniques for dependability evaluation. In: SRDS. (2000) 228–237