First. publ. in: Lecture notes in computer science, No. 3829 (2005) , pp. 177-195

Counterexamples for Timed Probabilistic Reachability

Husain Aljazzar[1], Holger Hermanns[2], and Stefan Leue[1]

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

[2] Department of Computer Science
Saarland University, Germany
hermanns@cs.uni-sb.de

Abstract. The inability to provide counterexamples for the violation of timed probabilistic reachability properties constrains the practical use of CSL model checking for continuous time Markov chains (CTMCs). Counterexamples are essential tools in determining the causes of property violations and are required during debugging. We propose the use of explicit state model checking to determine runs leading into property offending states. Since we are interested in finding paths that carry large amounts of probability mass we employ directed explicit state model checking technology to find such runs using a variety of heuristics guided search algorithms, such as Best First search and Z*. The estimates used in computing the heuristics rely on a uniformisation of the CTMC. We apply our approach to a probabilistic model of the SCSI-2 protocol.

1 Introduction

Overview. Stochastic models are widely used in system design to describe discrete phenomena that change randomly as time progresses. They are commonly employed to specify and reason about system performance and dependability characteristics. It is widely recognized that the availability of automated analysis tools is pivotal in the assurance of high quality system design, hence our interest in formal analysis of this type of properties. In this paper we are considering the use of explicit state Model Checking [1] (ESMC) in the formal analysis of stochastic system models. In particular, we use explicit state model checking to explain why probabilistic timed reachability properties are not satisfied by a stochastic model given in the form of a *Continuous-Time Markov Chain (CTMC)*.

A few model checking approaches for probabilistic models have been presented in the literature [2–9]. In [9], a branching time temporal logic named *Continuous Stochastic Logic (CSL)* for expressing real-time probabilistic properties on CTMCs has been proposed, based on [7]. Efficient approximative stochastic model checking algorithms to verify CSL formulae have been developed. We notice that in practice relevant CSL formulae are often non-nested and fall into the common fragment of (the timed stochastic variants of) CTL and LTL. In this paper we restrict ourselves to the consideration of an important class of safety properties that can be expressed in this restricted fragment of CSL, namely timed probabilistic reachability. A timed probabilistic reachability property is a property of the form:

> *"The probability to reach a state s violating a state proposition ϑ, i.e satisfying $\varphi := \neg\vartheta$, within the time interval $[0, t]$ does not exceed a probability $p \in [0, 1]$".*

The CSL syntax of this property is $\mathcal{P}_{<p}(\Diamond^{\leq t}\varphi)$.

The practical applicability of CSL model checking is constrained by the inability of this approach to produce offending system execution traces, also called counterexamples in model checking parlance, that illustrate why a property was violated. In particular, the probabilistic nature of the CTMC model checking problem means that it cannot generally be assumed that a single state-transition sequence through the model proves the violation of a probabilistic timed reachability property, in case it is invalid. In ordinary model-checking, such state-transition sequences are the prime debugging information provided by the model checker, in case the property is refuted. In the context of CSL, the probability of the property is determined by the probability mass flowing along all those state-transition sequences which lead into a target state, and consequently, the model checking algorithm cannot return a single state-transition sequence explaining why such states were reachable with a certain probability. Instead, the model checker can only give the actual probability in response to the specified CSL property. This means that the stochastic model checking algorithm will not be able to determine which part of the model is responsible for the undesired event of exceeding a given probability bound. It is then left up to the user to inspect the model in order to manually determine the reason for the property violation.

Approach. We address this problem by reconciling ESMC with CSL model checking. ESMC checks state properties of a system model by systematically exploring the state space of the system, usually given as an implicit graph. ESMC commonly uses graph search algorithms such as *Depth-First Search (DFS)* and *Breadth-First Search (BFS)* in the state space exploration. For safety properties, if an error is found the model checker returns an offending system run in the form of a state-transition sequence explaining how the property violating state can be reached from the initial system state. Reachability analysis performed by ESMC over-approximates the verification of timed probabilistic reachability since it does not respect the time and probability bounds that the property imposes.

In order to reconcile both approaches, an important question is to define a meaningful notion of a *counterexample* in the probabilistic context. Assume we are facing a refuted timed probabilistic reachability property. This means that the probability to reach an undesired state before the given time bound is higher than the bound specified in the property. This is caused by an infinite set of time-stamped runs (forming a tree structure of infinite depth and – owed to varying real-timed time stamps – infinite branching), which has a probability measure higher than required. Now the question is: what portion of this infinite set is the one that is undesired? The set itself does not provide useful information to answer this question. Even more, there surely is no general answer since the question is context-dependent. However it appears intuitively very beneficial to understand the structure of this set by determining its largest portions, especially by identifying a selection of runs through the system along which most of the relevant probability mass flows toward the undesired state. In this paper we are thus interested in identifying system runs in the state graph which are meaningful in the sense that they carry a lot of probability mass, and interpret these as meaningful counterexamples to the timed probabilistic reachability properties that we analyze. To discriminate more meaningful runs leading to property offending states from less meaningful ones during the state space exploration we employ heuristics guided directed search algorithms.

We envisage our approach to be applied in combination with a numeric probability analyser, such as the one included in the CADP tool set [10, 11]. A typical usage scenario would proceed in two steps. First, the system is checked on the given timed probabilistic reachability property using the numeric analyser, i.e., the probabilistic model checker. Second, if the property is determined to be violated, then a counterexample is elicited by our

directed state space exploration approach. Intermediate results of the numeric analysis can be applied in guiding the search process, as outlined in Section 3.2. While we assume that this is the most likely usage scenario, our approach does not depend on a preceding probabilistic model checking run. Even more, knowing that some state can be reached within some time with a relatively high probability can in itself constitute important information about the system, even if it is not a counterexample to a timed probabilistic reachability property in the above sense.

Structure of the Paper. In Section 2 we present some model foundations for performing reachability analysis on probabilistic models. In Section 3 we extend these concepts to include directed, heuristics guided model exploration. We introduce the modeling of the SCSI-2 protocol, which serves as case study in our paper, in Section 4. This section also gives the experimental results. We conclude in Section 5.

2 Explicit-State Analysis of Probabilistic Systems

2.1 Probabilistic Systems

In this section, we sketch the model for probabilistic systems that we use.

Definition 1 *A labelled discrete-time Markov chain (DTMC) is a triple* (S, P, L)*, where*

- S *is a finite set of states, and*
- $P : S \times S \longrightarrow [0, 1]$ *is a probability matrix, satisfying that for each state, probabilities of outgoing probabilistic transitions cumulate to 1.*
- $L : S \longrightarrow 2^{AP}$ *is labeling function, which assigns each state the subset of valid atomic propositions.*

We often assume that there is a unique initial state s_{init} of the system. Figure 1 illustrates a simple DTMC consisting of 3 states and 4 transitions. s_0 is the initial state, $AP = \{a, b\}$ and L is given through the subsets of AP labeling the states.

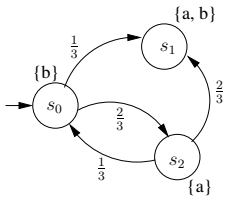

Fig. 1. A simple DTMC, or a CTMC with $E := \{(s_0, 3), (s_1, 0), (s_2, 5)\}$.

Definition 2 *A labelled continuous-time Markov chain (CTMC) is a quadruple* (S, P, E, L)*, where*

- (S, P, L)*, is a DTMC, and*
- $E : S \longrightarrow \mathbb{R}_{>0}$ *is a rate vector, assigning exit rates to states.*

For example, the DTMC illustrated in Figure 1 can be extended to a CTMC with $E := \{(s_0, 3), (s_1, 0), (s_2, 5)\}$.

Paths, runs and probability measures. For a given DTMC (S, P, L), an infinite *run* is a sequence $s_0 \rightarrow s_1 \rightarrow s_2 \rightarrow \ldots$ with, for $i \in \mathbb{N}$, $s_i \in S$ such that $P(s_i, s_{i+1}) > 0$ for all i. A finite run σ is a sequence $s_0 \rightarrow s_1 \rightarrow \ldots s_{l-1} \rightarrow s_l$ such that s_l is absorbing[3], and $P(s_i, s_{i+1}) > 0$ for all $i < l$. A (finite or infinite) *path* through a CTMC (S, P, E, L) is a sequence $s_0 \xrightarrow{t_0} s_1 \xrightarrow{t_1} s_2 \xrightarrow{t_2} \ldots$ such that $s_0 \rightarrow s_1 \rightarrow s_2 \rightarrow \ldots$ is a run through (S, P, L) and $t_i \in \mathbb{R}_{>0}$ for all relevant i. For a given initial state s_0 in CTMC $\mathcal{C}$, a unique probability measure Pr on $Path(s_0)$ exists, where $Path(s_0)$ denotes the set of paths starting in s_0 [9]. The probability measure induces a probability measure on $Run(s_0)$, where $Run(s_0)$ denotes the set of runs starting in s_0. Because runs are time-abstract, the latter measure only depends on the embedded DTMC, and can be defined directly using $\Pr(s_0, s_0 \rightarrow s_1 \rightarrow s_2 \rightarrow \ldots) = P(s_0, s_1) \cdot \Pr(s_1, s_1 \rightarrow s_2 \rightarrow \ldots)$.

Transient probabilities. In a CTMC, the time-dependent state probability can be written as: $\pi(s', s, t) = \Pr\{\sigma \in Path(s) \mid \sigma@t = s'\}$ where $\sigma@t$ denotes the state occupied at time t on path σ. $\pi(s', s, t)$ determines the probability to be in state s' at time t, if starting in state s at time 0. Efficient techniques based on uniformisation exist to compute these probabilities. These thechniques use a specific *uniformised* DTMC to reduce the required computations to the discrete setting. In a DTMC the corresponding time dependency is given by the number of hops along runs, and the hop-dependent probability is $\pi(s', s, k) = \Pr\{\sigma \in Run(s) \mid \sigma(k) = s'\}$ where $\sigma(k)$ denotes the k-th state in run σ. Intuitively, $\pi(s', s, k)$ denotes the probability to be in state s' after k hops, if starting in state s (with 0 hops). The values of $\pi(\cdot, \cdot, k)$ are given by the entries of the matrix P^k as $\pi(s', s, k) = P^k(s, s')$.

Timed reachability probabilities. For a CTMC, the time-bounded reachability probability can be defined as $\rho(s', s, t) = \Pr\{\sigma \in Path(s) \mid \exists t' \in [0, t] : \sigma@t' = s'\}$. $\rho(s', s, t)$ determines the probability to hit state s' at the latest at time t, if starting in state s at time 0. The computation of $\rho(s', s, t)$ can be reduced to that of $\pi(s', s, t)$ by making s' absorbing (i.e., cutting all transitions) prior to the computation of $\pi(s', s, t)$ [9].

2.2 Reachability Analysis of Probabilistic Models

Consider a timed probabilistic reachability property of the previously discussed form $\phi := \mathcal{P}_{<p}(\Diamond^{\leq t}\varphi)$, to be checked on a given CTMC with initial state s_{init}. According to the semantics of CSL [7] (which we omit here), the validity of ϕ can be decided by comparing the probability bound p with the cumulated timed reachability probability $\sum_{s' \models \varphi} \rho(s', s_{init}, t)$. The computation of this quantity can be done in one transient analysis where all states satisfying φ are made absorbing [9].

This answers the question how timed probabilistic reachability properties are decided effectively. The goal of the present paper is more refined since we are aiming to provide debugging information in case the property is refuted. When that is the case, at least one offending state s' satisfying φ is reachable. For typical error state specifications, the number of such offending states is large, leading to a large number of counterexamples that all contribute to exceeding the probability bound p. We expect the user to be interested in a counterexample which carries a high probability and which is hence most informative. We aim specifically at identifying a finite run such that the contribution of this run to the timed reachability probability is large or even maximal.

[3] A state of a Markov chain is called absorbing if it has no outgoing transition leading to another state.

To arrive there, we first have to introduce *timed run probabilities*. Recall that runs as such are time-abstract. Given a specific finite run $r = s_0 \rightarrow \ldots \rightarrow s_k$ of length k through the CTMC, the time-bounded run probability can be defined as

$$\gamma(r,t) = \Pr\{\sigma \in Path(s_0) \mid \exists t' \in [0,t] : \sigma@t' = s_k \ \wedge \ \sigma{\downarrow_k} = r\},$$

where $\sigma\downarrow$ is the projection of a path $s_0 \xrightarrow{t_0} s_1 \xrightarrow{t_1} s_2 \xrightarrow{t_2} \ldots$ on the run $s_0 \rightarrow s_1 \rightarrow s_2 \rightarrow \ldots$ obtained by removing the transition time stamps. The subscript $_k$ denotes the run truncated at depth k (thus ending in s_k which is contained in σ by construction). Intuitively, $\gamma(r,t)$ gives the probability that the CTMC moves along the run r and reaches $last(r)$ at the least at time t. For a finite run $r = s_0 \rightarrow s_1 \rightarrow s_2 \rightarrow \ldots \rightarrow s_n$ this timed run probability is given by

$$\gamma(r,t) = \int_0^t \left(p(s_1,s_0,t_1) \cdot \left(\ldots \left(\int_0^{t-t_{n-1}} p(s_n,s_{n-1},t_n) \cdot dt_n \right) \ldots \right) \right) \cdot dt_1, \qquad (1)$$

where $p(s',s,t) = P(s,s') \cdot \left(1 - e^{-E(s)\cdot t}\right)$ is the probability to move from s to s' in the interval [0,t]. One can show that $\gamma(r,t)$ can be computed by $\rho(last(r), first(r), t)$ (with the obvious meaning of $first$ and $last$) on a CTMC where all states which are not touched along the run r are made absorbing.

3 Directed Reachability Analysis of CTMCs

3.1 Search Algorithms

In state space search, the most commonly used algorithms are depth-first search (DFS) and breadth-first search (BFS). Depth-first search (DFS) is memory efficient, but typically finds goal states only very deep in the search space which leads to very long counterexamples. Breadth-first search (BFS) is complete and can provide shortest counterexamples. However, BFS is in general too memory inefficient to be applied to models of realistic size [12]. For weighted graphs, Dijkstra's algorithm is known to deliver optimal solutions [13]. Originally, Dijkstra's algorithm uses the summation as a cost measure, i.e. the cost of path is the sum of the costs of the path's transitions. The cost (or merit) measure is a function used by the search algorithm to assign each explored state a number quantifying the costs of the path from the start state to a goal state over the current state, e.g., path length. The merit is often considered to be the negation of the costs. An optimal solution is a path whose merit is maximal (i.e., costs are minimal). Markov chains can easily be cast into weighted graphs. In the stochastic setting, however, the summation of costs that Dijkstra's algorithm requires makes little sense. We hence use Dijkstra's algorithm with a suitable but non-additive cost measure.

The algorithms DFS, BFS and Dijkstra are uninformed search algorithms. To the contrary, the exists a large class of informed algorithms that take advantage of knowledge about structural properties of the state graph or the goal state specification in order to improve the search. Belonging to this class, directed search algorithms use such knowledge to perform heuristics guided state space exploration. The guiding aims at finding optimal (e.g., shortest or cost-minimal) paths in the state space. Heuristics guided search algorithms exploit their knowledge during state expansion when deciding which node to expand next. The problem knowledge manifests itself in a heuristic evaluation function f which estimates the desirability of expanding a node. Amongst others, f relies on an estimate h of

the optimal costs of the cheapest solution rooted in the current state. This algorithmic skeleton in its general form is called Best-first (BF) [14] search. If f is identical to h, the resulting greedy best first algorithm (GBestFS) will expand the successor node with the optimal value for h first. It often finds a solution fast but it is not optimal with respect to the cost of the path to a goal node since it can get stuck in sinks or local minima, as has been observed in work on directed explicit-state model checking (DESMC) which reconciles classical DFS based state space search with heuristics guided search [12].

In addition to GBestFS we also consider the Z*[4] search algorithm. According to Pearl [14], Z* is derived from BF by complying with the two following requirements:

- The use of a *recursively computed* evaluation function f, and by
- The use of a *delaying termination test* when deciding optimality of the expanded nodes.

The first requirement means that for each pair of states s and s', where s is the predecessor of s' in the traversal tree, $f(s')$ is computed by $f(s') := F[\psi(s), f(s), h(s')]$ for an arbitrary combination function F. $\psi(s)$ stands for a set of local parameters characterizing s. This requirement results in two efficiency improving features, namely computation sharing and selective updating [14]. The second requirement is needed to guarantee optimality of the search result that Z* delivers.

The optimality of Z* can only be guaranteed if the following two conditions are satisfied, in addition to the delaying termination test requirement:

1. The estimate h is *optimistic*, i.e. $h(s)$ is always an overestimate of the profit of expanding s.
2. The combining function F satisfies the *order-preservation* property illustrated in equation 2.

The property of h to be optimal has to be assured through its concrete definition in an application context. We then need to ascertain that F satisfies the order-preservation property:

$$\begin{aligned} &F[\psi(s_1), f(s_1), h_1(s')] \geq F[\psi(s_2), f(s_2), h_1(s')] \Rightarrow \\ &F[\psi(s_1), f(s_1), h_2(s')] \geq F[\psi(s_2), f(s_2), h_2(s')], \end{aligned} \tag{2}$$

for all states s_1, s_2 and $s' \in succ(s_1) \cap succ(s_2)$ and all optimistic heuristics h_1 and h_2 (where $succ(s)$ enumerates the successor nodes of s). The order-preservation property states that if two paths σ_1 and σ_2 from s to s' are found, where σ_1 can be reached with less cost than σ_2, then the further search process will not reverse this order. In other words, by discarding the more expensive path we do not throw away the optimal path. This property is fundamental for the optimality of the algorithm.

3.2 Probabilistic Search Algorithms

In case a timed reachability property, say $\mathcal{P}_{<p}(\Diamond^{\leq t}\varphi)$, is refuted, we are aiming at identifying a run r through the CTMC leading to a state s with $s \vDash \varphi$ with a high, if not the highest, timed run probability. In a heuristics guided state-space search algorithm, the timed run probability $\gamma(r, t)$ will therefore serve as optimization goal. However, the determination of the precise value of the integral in equation 1 is computationally very expensive and prone

[4] The more prominent A* heuristics guided search algorithm is a variant of Z*, where an additive estimation function f is used. Since the costs in our context are probabilities that are multiplied along the path, we need a multiplicative estimation function. Therefore, A* is not applicable in this setting.

to numerical instability problems. Thus $\gamma(r,t)$ cannot be used as a merit measure in the search process. Therefore we propose in the following section an approximation of γ. The approximation relies on a uniformisation of the CTMC.

Approximative Cost Measure We use a uniformisation method to turn the CTMC into a DTMC, which is then embedded in a Poisson process. Let $A = (S, P, E, L)$ be a CTMC. Using uniformisation we obtain an embedded DTMC $A' = (S, M, L)$, with M being defined as follows:

$$M = I + \frac{1}{\Gamma} \cdot E(s) \cdot (P - I), \tag{3}$$

where Γ is not smaller than the maximum of E. The Poisson process in which A' is embedded looks as follows:

$$Prob\{N(t) = k\} := \frac{(\Gamma \cdot t)^k}{k!} \cdot e^{-\Gamma \cdot t}, \qquad k,\ t \geq 0. \tag{4}$$

For fixed t, $N(t)$ is a random variable giving the (discrete) number of hops made in the uniformised model A' during the (continuous) time interval $[0,t]$ of A. For instance, the CTMC defined in Figure 1 can be uniformised using $\Gamma := max\{\ E(s_0),\ E(s_1),\ E(s_2)\ \} = 5$. The uniformised model differs from the original model only in so far as the branching probabilities of the state s_0 are changed as follows: $s_0 \xrightarrow{\frac{1}{5}} s_1$, $s_0 \xrightarrow{\frac{2}{5}} s_2$ and additionally $s_0 \xrightarrow{\frac{2}{5}} s_0$.

The expected value of the Poisson process above is $N := \Gamma \cdot t$. Intuitively, N corresponds to the expected number of hops in the uniformised DTMC that may occur in t time units. The probability that N hops occur in time t is maximal. Now let t be the time bound given in the reachability property. Our search algorithm is performed on A' and selects a path leading to an error state which has at most N transitions and carries a maximal probability. Thus we limit the search process to states reachable within at most N transitions, i.e. states probably reachable in the time interval $[0,t]$.

In addition, $\gamma(r,t)$ is reduced to its discrete variant $\gamma'(r,N)$. While $\gamma(r,t)$ denotes the reachability probability in CTMC A along run r and bounded by time t, $\gamma'(r,N)$ denotes the reachability probability in DTMC A' along run r and bounded by hop count N. In the traversal tree spanned by the search algorithm there is always at most one run r between each pair of states, i.e., r is completely characterized by $first(r)$ and $last(r)$. Thus, instead of $\gamma'(r,N)$ we write $\gamma'(last(r), first(r), N)$, or even $\gamma'(last(r), N)$ if $first(r)$ is the start state. If $first(r)$ is different from the start state, $\gamma'(last(r), first(r), N)$ is computed as if $first(r)$ was the start state. We use π' to denote the restriction of π to the traversal tree. $\pi'(s,k)$ is just $\pi(s, s_{init}, k)$ (for start state s_{init}), but on the DTMC obtained from A' by redirecting all transitions which are not contained in the traversal tree – except for self-loops – to an absorbing state. For any state visited by the search algorithm, π' is computed as follows:

$$\begin{aligned} \pi'(s_{init}, 0) &= \begin{cases} 1, & k = 0 \\ M(s_{init}, s_{init}) \cdot \pi'(s_{init}, k-1), & k > 0 \end{cases} \\ \pi'(s, 0) \quad &= \begin{cases} 0, & k = 0 \\ M(pred(s), s) \cdot \pi'(pred(s), k-1) + M(s,s) \cdot \pi'(s, k-1), & k > 0 \end{cases} \end{aligned} \tag{5}$$

where $pred(s)$ stands for the unique predecessor of s in the traversal tree. For some state s, $\gamma'(s,N)$ can be efficiently computed according the following expression:

$$\gamma'(s,N) = M(pred(s), s) \cdot \sum_{k=0}^{N-1} \pi'(pred(s), k), \tag{6}$$

where $pred(s)$ stands for the unique predecessor of s in the traversal tree. Using the approximation explained above we are now able to define an efficiently computable merit measure, namely $\gamma'(s, N)$.

In the following we demonstrate the computation of γ' by means of the uniformised variant of the the example illustrated in Figure 1. In the search process, at first s_0 is expanded generating s_1 and s_2. At this point $\gamma'(s_1, 2)$ is computed as follows:

$$\gamma'(s_1, 2) = \tfrac{1}{5} \cdot (\pi'(s_0, 0) + \pi'(s_0, 1)) = \tfrac{1}{5} \cdot (1 + \tfrac{2}{5} \cdot 1) = \tfrac{7}{25}.$$

Now we assume that the search algorithm expands the state s_2 generating the states s_0 and s_1. A new computation of $\gamma'(s_1, 2)$ is performed on the following way:

$$\begin{aligned}\gamma'(s_1, 2) &= \tfrac{2}{3} \cdot (\pi'(s_2, 0) + \pi'(s_2, 1)) \\ &= \tfrac{2}{3} \cdot (\pi'(s_2, 0) + \tfrac{2}{5} \cdot \pi'(s_0, 0) + 0 \cdot \pi'(s_2, 0)) = \tfrac{2}{3} \cdot (0 + \tfrac{2}{5} \cdot 1) = \tfrac{4}{15}\end{aligned}$$

Since $\frac{7}{25} > \frac{4}{15}$, the run $s_0 \rightarrow s_1$ is preferred over $s_0 \rightarrow s_2 \rightarrow s_1$ in order to reach s_1.

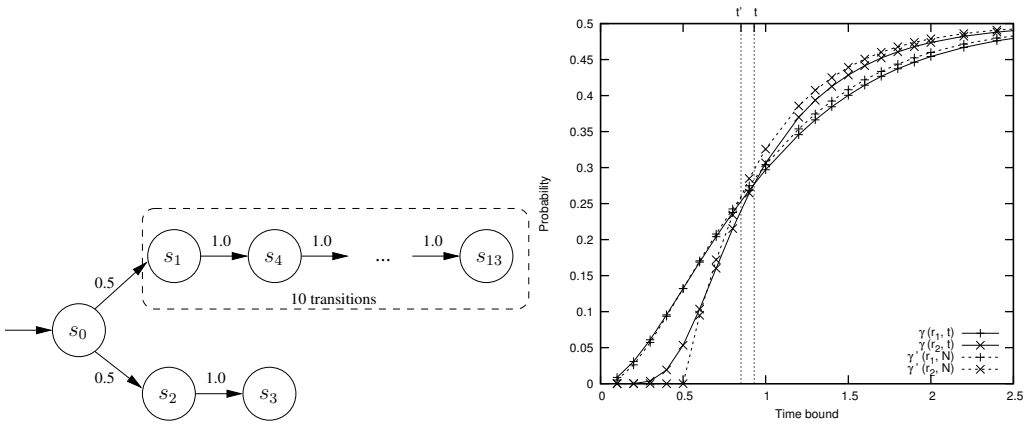

Fig. 2. An example of a CTMC

To give an impression of this approximation we consider an example CTMC that emphasizes a characteristic challenge in timed probabilistic search. We study the CTMC given on the left hand side of Figure 2, where E is defined as follows:

$$E(s) := \begin{cases} 2, & \text{if } s \in \{s_0, s_2\} \\ 20, & \text{otherwise} \end{cases}$$

Let s_0 be the start state, s_3 and s_{13} be goal states. We refer to the run from s_0 to s_{13} by r_1 and to s_3 by r_2. For small time bounds $t < 1.0$, $\gamma(r_1, t)$ is smaller than $\gamma(r_2, t)$, but for large time bounds $t \geq 1.0$, $\gamma(r_1, t)$ is higher than $\gamma(r_2, t)$. This observation implies that our search algorithm should select run r_1 for small time-bounds, and r_2 for larger bounds, in case it is a well-designed heuristics. The plot on the right side of figure 2 illustrates $\gamma(r_1, t)$ and $\gamma(r_2, t)$ and their approximations $\gamma'(r_1, N)$ and $\gamma'(r_2, N)$ depending on the time bound. We can see that the approximated curves run rather closely by the accurate curves. The point of inflexion of the approximative curves where the higher probability mass changes from r_1 to r_2 is located quite close to the inflection point of the non-approximated curves.

This illustrates that for a given time bound we are able to rather accurately determine the optimal path using our approximation.

As mentioned previously, the cost estimate function f used in Z* takes a heuristic function h into account. In our setting h is an optimistic heuristics, i.e. for each state $h(s)$ is an overestimate of the maximal probability $h^*(s)$ to reach a goal state started from s within at most N transitions, more precisely:

$$h(s) \geq h^*(s) := max\{\gamma'(s', s, N) \mid s' \text{ is a goal state}\}. \tag{7}$$

f is defined formally in the following equation:

$$\begin{aligned} f(s') &:= F[\psi(s), f(s), h(s')] = F[\{\pi'(s,k) | 0 \leq k \leq N\}, M(s,s'), h(s')] \\ &= -\gamma'(s', N) \cdot h(s'). \end{aligned} \tag{8}$$

$\gamma'(s', N) \cdot h(s')$ is an estimate of the merit of the state s' and the costs are the negation of this value. If it is technically possible, information from the numerical analysis can be used in the computation of $\gamma'(s', N)$ and $h(s')$. This would increase the performance by computation sharing. The quality of h can also be improved. We remark that the property of f of being optimistic relies on the currently unproven conjecture that for each triple of states s_1, s_2 and s_3, where s_i is an ancestor of s_j in the traversal tree of the search algorithm for $i < j$, the following inequation holds:

$$\gamma'(s_2, s_1, N) \cdot \gamma'(s_3, s_2, N) \geq \gamma'(s_3, s_1, N). \tag{9}$$

This implies admissibility of f, and consequently the optimality of the algorithm. Our experiments confirm this conjecture since the search always finds optimal counterexamples and a reopening of nodes, which can have detrimental effects on the computational performance, was never observed. We point out, however, that optimality is not our essential goal. The experience in directed model checking has been that even inadmissible heuristics deliver good (sub-)optimal goals, if only the heuristic estimate is informative. We also remark that we have to store with each open state s, i.e. leaf state in the traversal tree, a vector of the size $N - depth(s)$ saving the probabilities to be in the state s after k transitions, for each $depth(s) \leq k \leq N$. However the effect of this additional space is imperceptible, hence the set of open states is very small relative to the whole state space.

To establish the optimality of Z* we still have to establish order-preservation of f. This can be done by the following reasoning:

$$\begin{aligned} & F[\psi(s_1), f(s_1), h_1(s')] \geq F[\psi(s_2), f(s_2), h_1(s')] \\ \Rightarrow\ & -\gamma'(s_1, N) \cdot h_1(s') \geq -\gamma'(s_2, N) \cdot h_1(s') \\ \Rightarrow\ & -\gamma'(s_1, N) \geq -\gamma'(s_2, N) \\ \Rightarrow\ & -\gamma'(s_1, N) \cdot h_2(s') \geq -\gamma'(s_2, N) \cdot h_2(s') \\ \Rightarrow\ & F[\psi(s_1), f(s_1), h_2(s')] \geq F[\psi(s_2), f(s_2), h_2(s')] \end{aligned}$$

In the above proof we use the fact that h_1 and h_2 never take the value 0. Let $S' \subseteq S$ be the set containing all goal states. Note that $h_i(s) = 0$ for some state s implies that $h^*(s) = 0$ because $h_i(s) \geq h^*(s)$. That means, the probability to reach a goal state outgoing from s within N transitions is 0. In this case we are allowed to remove s and its in- and outgoing transitions from the state space.

3.3 Heuristic Functions

We now turn to the question how to efficiently compute informative heuristic estimates to be used in the search. Consider the property $\mathcal{P}_{<p}(\Diamond^{\leq t}\varphi)$. For every state s that we explore in the search we need an overestimating function for the maximal time-bounded run probability until a state satisfying φ is reached from s, more precisely, $max\{\gamma'(s', s, N) \mid s' \vDash \varphi\}$ Let h_φ be an estimation for the maximal time-bounded run probability until a state *satisfying* φ is reached and $\bar{h}_\varphi$ an estimation for the maximal time-bounded run probability until a state *violating* φ is reached. If φ is an atomic proposition, the concrete heuristic estimation values are application dependent and we shall provide examples in the context of our case study in Section 4. However, the state formulae characterizing the state reachability conditions consist of boolean combinations of atomic state propositions, as for instance in the formula $\mathcal{P}_{<p}(\Diamond^{\leq t}(\varphi_1 \wedge \neg\varphi_2))$. Table 1 illustrates how heuristic estimates can be obtained from boolean combinations of atomic formulae. If the given heuristic functions of the atomic propositions are admissible, then the heuristic functions built according to Table1 are admissible. We can therefore assume that Z* delivers paths leading into goal states with optimal costs.

φ	h_φ	$\bar{h}_\varphi$
$\neg\,\varphi_1$	$\bar{h}_{\varphi_1}$	h_{φ_1}
$\varphi_1 \vee \varphi_2$	$max\{h_{\varphi_1}, h_{\varphi_2}\}$	$min\{\bar{h}_{\varphi_1}, \bar{h}_{\varphi_2}\}$
$\varphi_1 \wedge \varphi_2$	$min\{h_{\varphi_1}, h_{\varphi_2}\}$	$max\{\bar{h}_{\varphi_1}, \bar{h}_{\varphi_2}\}$

Table 1. A method to build heuristic functions for complex reachability properties.

4 Case Study: The SCSI-2 Protocol

SCSI-2 Protocol. To illustrate our approach we analyze a storage system of realistic size. This system consists of up to 8 devices, one disk controller and up to 7 hard disks. These devices are connected by a bus implementing the *Small Computer System Interface-2* (SCSI-2) standard [15]. Each device is assigned a unique SCSI number between 0 and 7. In [16], this system was analyzed regarding starvation problems for disks having SCSI numbers smaller than the SCSI number of the disk controller. Within the scope of that work the system was modelled in LOTOS [17] and transformed into an *interactive Markov chain (IMC)* by the CADP toolbox [10, 11]. We will use this model in our analysis.

In accordance with the SCSI-2 protocol definition, the controller can send a *command* (CMD) to the disk d. After processing this command, the disk sends a *reconnect* message (REC) to the controller. CMD and REC messages of every disk are stored in eight-place FIFO queues. CMD and REC messages circulate on the SCSI bus, which is shared by all devices. To avoid access conflicts on the bus, a bus arbitration policy is used.

Directed Reachability Analysis of the SCSI-2 Protocol. We analyse the storage system described above with respect to the probability of reaching an overload situation of the hard disks. The system consists of one controller and three disks. On one disk (the main disk) the operating system and all applications are installed. The two other disks are used to backup the data of the main disk (backup disks). In stress cases, the main disk is intensely

loaded and has to serve many more access requests than the other disks. It is interesting to determine the probability to reach some state where the main disk is overloaded while the other disks are not busy. In this situation the command queue of the main disk is full and the queues of the backup disks are empty. Similarly we consider the system behavior during backup. In order to capture these situations we define the following atomic state propositions:

- For each disk d, the command queue of d is full: φ_d.
- For each disk d, the command queue of d is empty: ϑ_d.

Let 0 be the SCSI number of the main disk and 1 and 2 the numbers of the backup disks. We are interested in the following properties:

1. The main disk is overloaded and the backup disks are idle (MDOL):

$$\phi := \mathcal{P}_{<p}(\Diamond^{\leq t}\varphi_0 \wedge \vartheta_1 \wedge \vartheta_2) \tag{10}$$

2. One backup disk is overloaded and the two other disks are idle (BDOL):

$$\theta := \mathcal{P}_{<p}(\Diamond^{\leq t}\vartheta_0 \wedge (\varphi_0 \wedge \vartheta_1) \vee (\vartheta_0 \wedge \varphi_1)) \tag{11}$$

We first need heuristic estimation functions for the atomic propositions, as discussed in Section 3.3.[5] As we mentioned above, determining these estimates requires to exploit some domain specific insight.

Let D denote the set containing the SCSI numbers of the plugged disks. The map $cq : S \times D \longrightarrow \{0, 1, ..., 8\}$ gives for each disk the number of commands contained in its command queue in the current state.

1. The delay required to issue a new command to the disk d, for some $d \in D$, is modelled by a Markovian transition with rate λ_d.
2. The servicing time of the disk d is modelled by a Markovian transition with rate μ_d.

For the uniformisation of the model we have to determine a value which is not smaller than $E_{max} \geq max\{E(s) \mid s \in S\}$. It can easily be shown that $max\{E(s) \mid s \in S\} = \sum_{d \in D} (\lambda_d + \mu_d) =: E_{max}$. Thus in the uniformised model the branching probability of the transition $s \xrightarrow{rate} s'$ is $\frac{rate}{E_{max}}$. In each state s, transitions modeling sending a new command to disk d as well as processing a command by the disk d compete at least against the following Markovian delays (including the delay of the transition itself):

- λ_i for each disk i, where $cq(s, i) < 8$.
- μ_i for each disk i, where $cq(s, i) > 0$.

This leads to the following inequation, describing possible underestimations for the probabilities:

$$E(s) \;\geq\; \left(\sum_{\substack{i \in D \\ cq(s,i)<8}} \lambda_i + \sum_{\substack{i \in D \\ cq(s,i)>0}} \mu_i \right) =: E'(s). \tag{12}$$

[5] Since negations of the atomic propositions do not occur in the property which we analyse we do not need to give heuristic functions for the negations $\bar{h}$.

In other words, the branching probability of leaving s, which we denote by $p_{out}(s)$, is not smaller than $\frac{E'(s)}{E_{max}}$. Thus, if s' is the other end state of such transition, then

$$\gamma'(s', s, N) \leq p \cdot \sum_{k=0}^{N-1} (1 - p_{out}(s))^k,$$

where p is the branching probability of the transition, i.e. $\frac{\lambda_i}{E_{max}}$ or $\frac{\mu_i}{E_{max}}$.

In conclusion, relying on the the over-approximation above and the conjecture explained in Section 3.2, we can define the following optimistic heuristic functions:

$$h_{\varphi_d}(s) := (\frac{\lambda_d}{E_{max}} \cdot \sum_{k=0}^{N-1} (1 - p_{out}(s))^k))^{8-cq(s,d)} \tag{13}$$

$$h_{\vartheta_d}(s) := (\frac{\mu_d}{E_{max}} \cdot \sum_{k=0}^{N-1} (1 - p_{out}(s))^k))^{cq(s,d)} \tag{14}$$

More precisely, we conjecture the following relations:

$$h_{\varphi_d}(s) \geq h^*_{\varphi_d}(s) := max\{\gamma'(s, s', N) \mid cq(s', d) = 8\}$$
$$h_{\vartheta_d}(s) \geq h^*_{\vartheta_d}(s) := max\{\gamma'(s, s', N) \mid cq(s', d) = 0\}$$

The heuristic functions h_φ and h_ϑ are built according to Table 1[6].

Implementation and Experiments. As mentioned above we use GBestFS and Z* as directed search algorithms and compare them to the undirected algorithms DFS, BFS and Dijkstra. The variant of Dijkstra's algorithm that we use interprets the $-\gamma'(s, N)$ values as a weight of the state s. We expect that BFS delivers the shortest solution while our variant of Dijkstra delivers the path with maximal time-bounded run probability.

We generate the models using the OPEN/ CÆSAR environment [18], also referred to as CADP, with which it is possible to generate a C graph module representing an implicit labeled transition system (LTS) corresponding to the given LOTOS model. We then transform the generated LTS into a CTMC[7]. We explore this CTMC on-the-fly by our search algorithms. If property violations are found, the search algorithm delivers a path leading from the initial state to a state violating the property. In order to preserve all probabilistic information of the model, the remainder of the model is replaced by a special absorbing state s_{out} to which we redirect all transitions which originate from some state on the path to the goal states, but which are not part of the path.

Table 2 shows an overview on the probabilities computed for the properties MDOL and BDOL. In order to assess the quality of some counterexample that we found we compare the reachability probability of the counterexamples with the precise reachability probability of the property in the original model, as determined by a transient analyser. For this purpose we generate the explicit state graph of the model in the BCG format of CADP and modify it by making all goal states absorbing. After that we analyze the modified model using the transient analyser of CADP. The resulting probabilities are given in the row labeled by "Model". For the probabilistic algorithms (Dijkstra, GBestFS, Z*), two probability values

[6] Note that heuristics built according to this table are monotone in case the heuristics for the atomic propositions are monotone. For the SCSI-2 example this is the case. The monotonicity manifests itself in the fact that we did not observe state re-openings in the experiments.

[7] In fact, in general the model resulting from this transformation is an interactive Markov chain (IMC), which in this particular case happens to correspond to a CTMC [16].

Prob.	Time bound		1	2	3	4	5	6	7	8	9	10
MDOL	Model		0.235	0.312	0.327	0.329	0.329	0.329	0.330	0.330	0.330	0.330
	DFS		-	-	-	-	-	-	0.000	-	-	0.000
	BFS		-	0.161	0.161	0.161	0.161	0.161	0.161	0.161	0.161	0.161
	Dijkstra	estimated	-	0.049	0.049	0.049	0.049	0.049	0.049	0.049	0.049	0.049
		precise	-	0.161	0.161	0.161	0.161	0.161	0.161	0.161	0.161	0.161
	GBestFS	estimated	-	0.005	0.005	0.005	0.005	0.005	0.005	0.005	0.005	0.005
		precise	-	0.012	0.012	0.012	0.012	0.012	0.012	0.012	0.012	0.012
	Z^*	estimated	-	0.049	0.049	0.049	0.049	0.049	0.049	0.049	0.049	0.049
		precise	-	0.161	0.161	0.161	0.161	0.161	0.161	0.161	0.161	0.161
BDOL	Model		0.008	0.008	0.008	0.008	0.008	0.008	0.008	0.008	0.008	0.008
	DFS		-	-	-	0.000	0.000	0.000	0.000	0.000	0.000	0.000
	BFS		-	0.002	0.002	0.002	0.002	0.002	0.002	0.002	0.002	0.002
	Dijkstra	estimated	-	0.001	0.001	0.001	0.001	0.001	0.001	0.001	0.001	0.001
		precise	-	0.002	0.002	0.002	0.002	0.002	0.002	0.002	0.002	0.002
	GBestFS	estimated	-	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
		precise	-	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
	Z^*	estimated	-	0.001	0.001	0.001	0.001	0.001	0.001	0.001	0.001	0.001
		precise	-	0.002	0.002	0.002	0.002	0.002	0.002	0.002	0.002	0.002

Table 2. Overview on the probabilities computed for the properties MDOL and BDOL for different time bounds, for main and backup disk load = 10.

are recorded in Table 2, namely the *estimated* and the *precise* ones, c.f. the table rows labeled "estimated" and "precise". The estimated value is $\gamma'(r, N)$ computed by the search algorithm. The precise probability of the counterexample is $\gamma(r, t)$ computed by the numeric transient analyser of CADP. To compute the precise value we run the transient analyser on the counterexample interpreted as a truncated CTMC (see Figure 5). When the search algorithm was unable to find a counterexample, then we denote this with an entry of the form '-'.

To interpret the results, we can first see that for both properties the probability of the run delivered by Z^* corresponds to the probability of the run using Dijkstra. This supports the optimality of Z^* and the optimality conjecture that we propose in Section 3.2. The probabilities of the counterexamples delivered by DFS are very close to zero which supports our claim that DFS is unsuitable in this setting. In many cases the DFS algorithm failed to find a counterexample at all. The reason is that we limited the search depth of DFS to N[8]. For time bound 1 no counterexample could be found since no goal state was reachable in the approximation within N state transitions. The optimal counterexample for both properties happens to be the shortest one. Only due to this BFS was able to select the optimal path in all experiments. The counterexamples found by GBestFS are inferior to the optimal ones.

Figures 3 and 4 compare the performance of the different algorithms with respect to CPU runtime, explored states and used memory space, respectively. Generally, we can observe that the guided algorithms (GBestFS and Z^*) have a much better performance in terms of both runtime and memory space than the uninformed algorithms (DFS, BFS and Dijkstra). This suggests that the guided search algorithms will scale much better to larger problems than, in particular, Dijkstra's algorithm. In some situations GBestFS works more efficiently than Z^* but the difference is not very significant. The algorithms DFS and BFS perform badly in terms of both runtime and number of explored states. The number of

[8] C.f. also the incompleteness of depth bounded DFS discussed in [19].

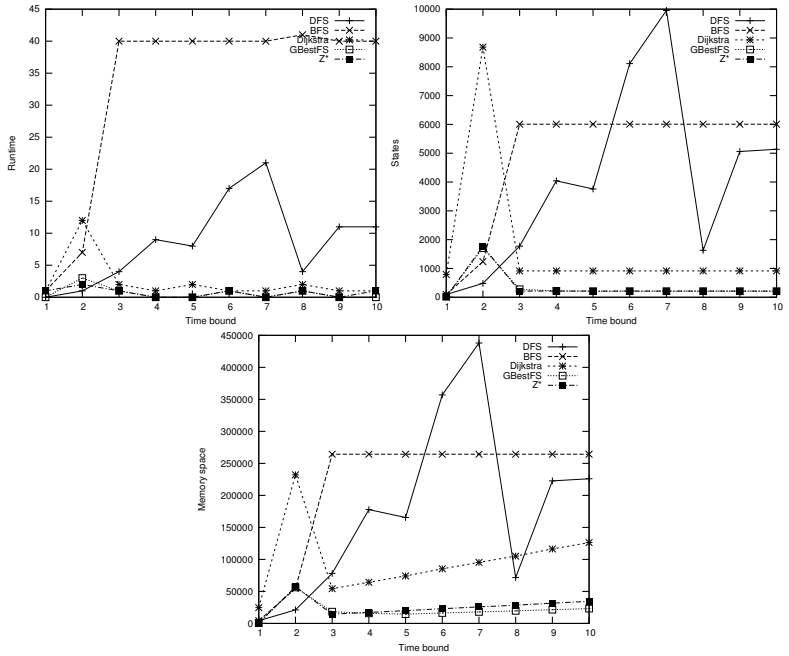

Fig. 3. Computational effort for MDOL depending on the time bound (main disk load = 5)

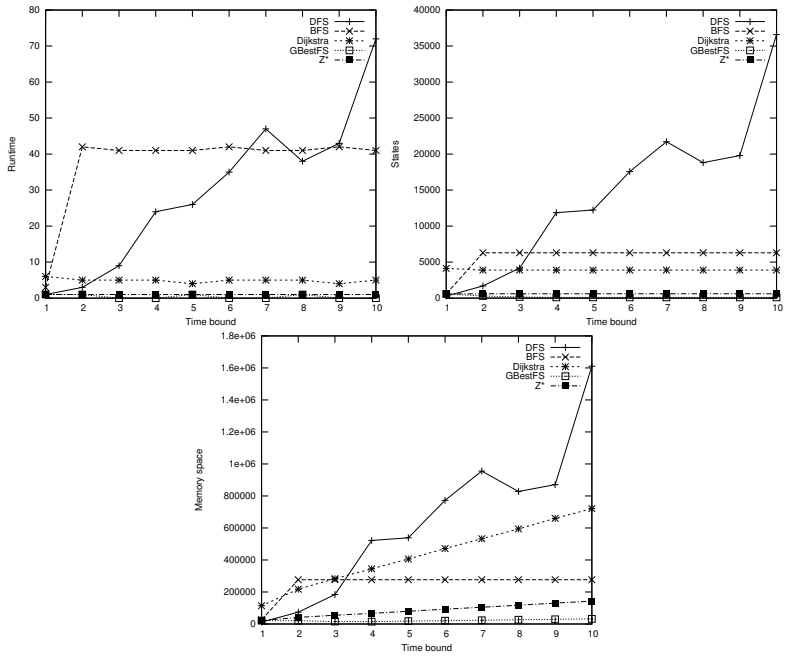

Fig. 4. Computational effort for BDOL depending on the time bound (backup disk load = 5)

explored states usually correlates with the memory space used by the algorithm. However the probabilistic algorithms (Dijkstra, GBestFS and Z*) use additionally a vector of the size $\leq N$ for each open state s in order to save the values of $\pi(k)$, for $depth(s) \leq k \leq N$, c.f. section 3.2. Normally the effect of this additional space is imperceptible, hence the set of open states is very small relative to the whole state space. However, the effect can be noticed in Figure 4. Although BFS explores more states than Dijkstra, it requires less memory than BFS.

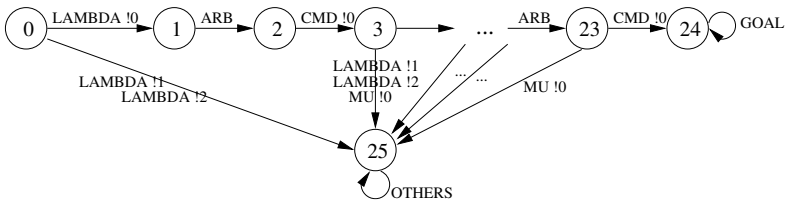

Fig. 5. A Counterexample for MDOL

To assess the counterexamples found by our probabilistic algorithms qualitatively, we consider the following comparison. For the properties MDOL and BDOL, DFS finds an error state, if at all, over a very intricate run. These runs contribute almost nothing to the whole reachability probability of the property. In models in which short runs do not carry necessarily a large probability mass, we will detect the same effect for BFS. Certainly, such runs do not provide much insight in order to to locate the cause of the property violation. On the other hand, there are other runs that obviously carry larger probability mass. We illustrate one such run in Figure 5. It also corresponds to the counterexample that Z* finds for the MDOL property. This counterexample models the behavior of the system in the case that right from the start permanently new commands are sent to the main disk (disk 0) while the disk does not get the chance to service any of these commands. The counterexample largely consists of eight repetitions of the events `Lambda !0, ARB, CMD !0`. These correspond to a Markovian delay (`Lambda !0`), access to the data bus (`ARB`) and the sending of a command to disk 0 (`CMD !0`). This in fact corresponds to the most direct way to overload the main disk, a situation that is represented by the state labeled `GOAL`. All transitions which are not touched along this run are redirected to a sink state labeled `OTHERS`. By analysing this counterexample we can identify the following two factors contributing to the high probability of the property violation:

1. The Markov delay `LAMBDA !0` modeling that the controller issues a new command for disk 0 is relative large, i.e. the disk is highly frequented.
2. Other Markov delays, especially `MU !0` modeling the servicing delay of disk 0, are relative small, i.e. the disk can barely service the requests.

This leads to the following solution strategy. The main disk has to be replaced by a faster one which means increasing `MU !0`. At the same time the storage system load could be reduced, i.e. the value of `LAMBDA !0` should be decreased.

In summary, the experiments show that our approach succeeds in efficiently finding counterexamples carrying large portion of the full probability and that are therefore meaningful in the debugging process. The estimated probabilities of the found counterexamples are relatively accurate, and comprise on average 50% for MDOL and 25% for BDOL of the total probability (c.f. line "Model" in the table).

5 Conclusion

We have described a method to determine counterexamples for timed probabilistic reachability properties of CTMCs. Our approach relies on heuristics directed explicit state search and employs uniformisation in order to efficiently compute heuristics. Using experimental data we illustrated that our approach succeeds in finding meaningful counterexamples in a way that is computationally superior to using other, non-directed search approaches.

Related work can largely be found in the area of model checking for stochastic systems, and in directed model checking, as cited earlier. We are not aware of other approaches to solving counterexample generation for stochastic model checking using heuristic search techniques. The method is also readily applicable to untimed probabilistic reachability counterexample generation.

Goals for future research include the use of appropriate search methods to approximate the timed probabilistic reachability problem from below, i.e., to determine a set of n runs in the model leading into goal states so that the combined probability of these runs exceeds the probability bound p. For appropriately structured models and small n this may be computationally advantageous compared to numerical stochastic model checking.

References

1. Clarke, E.M., Grumberg, O., Peled, D.A.: Model Checking. The MIT Press (1999 (third printing 2001))
2. Bianco, A., de Alfaro, L.: Model checking of probabalistic and nondeterministic systems. In: Proceedings of the 15th Conference on Foundations of Software Technology and Theoretical Computer Science, Springer-Verlag LNCS (1995) 499–513
3. A. Aziz, K. Sanwal, V. Singhal, R. K. Brayton: Verifying continuous-time Markov chains. In Rajeev Alur, Thomas A. Henzinger, eds.: Eighth International Conference on Computer Aided Verification CAV. Volume 1102., New Brunswick, NJ, USA, Springer Verlag LNCS (1996) 269–276
4. de Alfaro, L.: Formal Verification of Probabilistic Systems. Ph.d. dissertation, Stanford University (1997)
5. Baier, C.: On algorithmic verification methods for probabilistic systems. Habilitation Thesis. Habilitation thesis, University of Mannheim (1998)
6. Baier, C., Katoen, J.P., Hermanns, H.: Approximate symbolic model checking of continuous-time Markov chains. In: International Conference on Concurrency Theory, Springer Verlag LNCS (1999) 146–161
7. Aziz, A., Sanwal, K., Singhal, V., Brayton, R.: Model-checking Continuous-Time Markov Chains. ACM Transactions on Computational Logic **1** (2000) 162–170
8. Baier, C., Haverkort, B.R., Hermanns, H., Katoen, J.P.: Model checking continuous-time Markov chains by transient analysis. In: CAV '00: Proceedings of the 12th International Conference on Computer Aided Verification, Springer-Verlag LNCS (2000) 358–372
9. Baier, C., Haverkort, B., Hermanns, H., Katoen, J.P.: Model-checking algorithms for continuous-time Markov chains. IEEE Transions on Software Engineering **29** (2003)
10. H. Garavel, F. Lang, R.M.: An overview of CADP 2001. European Association for Software Science and Technology (EASST) Newsletter **4** (2002)
11. Hermanns, H., Joubert, C.: A set of performance and dependability analysis components for CADP. In Garavel, H., Hatcliff, J., eds.: Proceedings of the 9th International Conference on Tools and Algorithms for the Construction and Analysis of Systems TACAS'2003 (Warsaw, Poland). Volume 2619., Springer Verlag LNCS (2003) 425–430
12. Edelkamp, S., Lafuente, A.L., Leue, S.: Directed explicit-state model checking in the validation of communication protocols. Software Tools for Technology Transfer **5** (2004) 247–267

13. Dijkstra, E.W.: A note on two problems in connexion with graphs. Numerische Mathematik **1** (1959) 269–271
14. Pearl, J.: Heuristics – Intelligent Search Strategies for Computer Problem Solving. Addision–Wesley (1986)
15. (ANSI), A.N.S.I.: Small computer interface-2 – standard x3.131-1994 (1994)
16. Garavel, H., Hermanns, H.: On combining functional verification and performance evaluation using CADP. In: Proceedings of the International Symposium of Formal Methods Europe on Formal Methods - Getting IT Right, Springer-Verlag LNCS (2002) 410–429
17. ISO/IEC: LOTOS – a formal description technique based on temporal ordering of observational behaviour (1989)
18. Garavel, H.: OPEN/CÆSAR: An open software architecture for verification, simulation, and testing. In: TACAS '98: Proceedings of the 4th International Conference on Tools and Algorithms for Construction and Analysis of Systems, London, UK, Springer-Verlag LNCS (1998) 68–84
19. Holzmann, G.J.: The Spin Model Checker: Primer and Reference Manual. Addision–Wesley (2003)