Hierarchical Counterexamples for Discrete-Time Markov Chains

Nils Jansen¹, Erika Ábrahám¹, Jens Katelaan¹, Ralf Wimmer², Joost-Pieter Katoen¹, and Bernd Becker²

¹ RWTH Aachen University, Germany
  abraham|nils.jansen|katoen@cs.rwth-aachen.de
² Albert-Ludwigs-University Freiburg, Germany
  wimmer|becker@informatik.uni-freiburg.de

Abstract. This paper introduces a novel counterexample generation approach for the verification of discrete-time Markov chains (DTMCs) with two main advantages: (1) We generate abstract counterexamples which can be refined in a hierarchical manner. (2) We aim at minimizing the number of states involved in the counterexamples, and compute a critical subsystem of the DTMC whose paths form a counterexample. Experiments show that with our approach we can reduce the size of counterexamples and the number of computation steps by several orders of magnitude.

1 Introduction

Discrete-time Markov chains (DTMCs) are a well-known modeling formalism for probabilistic systems. The probabilistic computation tree logic (PCTL) [6] is suited to express bounds on the probability mass of all paths satisfying some properties. Efficient algorithms and tools are available to verify PCTL properties of DTMCs. Prominent model checkers like PRISM [9] and MrMC [8] offer methods based on the solution of linear equation systems [6].

If verification reveals that a system does not fulfill a required property, the ability to provide diagnostic information is crucial for bug fixing. A counterexample carries an explanation why the property is violated. E. g., for Kripke structures and linear temporal logic (LTL) formulae, a counterexample is a path that violates the property, which can be generated by LTL model checking as a by-product without additional overhead. State-of-the-art model checking algorithms for probabilistic systems do not exhibit this feature. After model checking, current techniques have to apply additional methods to generate probabilistic counterexamples.

Even for large state spaces, a counterexample consisting of a single path gives an intuitive explanation why the property is violated. In the probabilistic setting, instead of a single path we need a set of paths whose total probability mass violates...
the bound specified by the PCTL formula [5]. It is much harder to understand
the behavior represented by such a probabilistic counterexample as it may
consist of a large or even infinite number of paths. To ease understanding, most
approaches aim at finding counterexamples with a small number of paths having
high probabilities. To generate more compact counterexamples, also the usage of
regular expressions [5], the detection of loops [11], and the abstraction of strongly
connected components (SCCs) [4] have been proposed, as well as diagnostic
subgraphs [3], which is most related to our counterexample representation.

We suggested in [2] a model checking approach based on the hierarchical
abstraction of SCCs. We abstract each SCC by a small loop-free graph in a
recursive manner by the abstraction of sub-SCCs. The result is an abstract
DTMC consisting of a single initial state and absorbing states, and transitions
carrying the total probabilities of reaching target states. In [2] we also gave an
idea of how to use the SCC-based model checking result for counterexample
generation. In this paper we generalize this approach and suggest a novel method
which computes a critical subsystem whose paths induce a counterexample. While
other methods concentrate on minimizing the number of paths, our computation
regards the system structure and aims at reducing the number of involved states
and transitions.

Critical subsystems are computed hierarchically. We refine a critical subsystem
by concretizing abstract states and reducing the concretized parts, such that the
reduced subsystem still induces a counterexample. This hierarchical approach
increases the usability of counterexamples for large state spaces. Concretization
of only the user-relevant parts of the abstract critical subsystem allows for an
intuitive approach for error correction.

The computation of critical subsystems is based on finding most probable
paths or path fragments to be contained in the critical subsystem. We propose
two approaches. The global method searches for paths through the entire system.
Our main contribution is the local search which aims at connecting most probable
path fragments. In contrast to most of the other approaches, our method is
complete, i.e., termination is always guaranteed.

Experiments for two well-known case studies show that our approach reduces
the size of counterexamples and the number of computation steps by several
orders of magnitude.

The paper is structured as follows: Section 2 contains some preliminaries.
We recall our model checking algorithm in Section 3. Section 4 describes our
counterexample generation method, for which we give some experimental results
in Section 5. A more detailed version of this paper, including examples and
illustrations, can be found in [1].

2 Preliminaries

**Definition 1.** Assume a set $AP$ of atomic propositions. A discrete-time Markov
chain (DTMC) is a tuple $M = (S, I, P, L)$ with a non-empty finite state set $S$, an
initial discrete probability distribution $I : S \to [0, 1]$ with $\sum_{s \in S} I(s) = 1$, a
transition probability matrix $P : S \times S \to [0,1]$ with $\sum_{s' \in S} P(s,s') = 1$ for all $s \in S$, and a labeling function $L : S \to 2^{AP}$.

To reduce notation, we refer to the components of a DTMC $M_i^n$ by $S_i^n$, $I_i^n$, $P_i^n$, and $L_i^n$. E.g., we use $S'$ to denote the state set of the DTMC $M'$. Assume in the following a set $AP$ of atomic propositions and a DTMC $M = (S,I,P,L)$.

We say that there is a transition from a state $s \in S$ to a state $s' \in S$ if $P(s,s') > 0$. A path of $M$ is a finite or infinite sequence $\pi = s_0 s_1 \ldots$ of states $s_i \in S$ such that $P(s_i,s_{i+1}) > 0$ for all $i$. We say that the transitions $(s_i,s_{i+1})$ are contained in the path $\pi$, written $(s_i,s_{i+1}) \in \pi$. We write $Paths_{\text{inf}}^M$ for the set of all infinite paths of $M$, and $Paths_{\text{init}}^M(s)$ for those starting in $s \in S$. Analogously, $Paths_{\text{fin}}^M$ is the set of all finite paths of $M$, $Paths_{\text{fin}}^M(s)$ of those starting in $s$, and $Paths_{\text{fin}}^M(s,t)$ of those starting in $s$ and ending in $t$. A state $t$ is called reachable from another state $s$ iff $Paths_{\text{fin}}^M(s,t) \neq \emptyset$.

A state set $S' \subseteq S$ is called absorbing in $M$ iff there is a state in $S'$ from which no state outside $S'$ is reachable in $M$. We call $S'$ bottom in $M$ if this holds for all states in $S'$. States $s \in S$ with $P(s,s) = 1$ are also called absorbing states.

We call $M$ loop-free, if all of its loops are self-loops on absorbing states. A set $S' \subseteq S$ is strongly connected in $M$ iff for all $s,t \in S'$ there is a path from $s$ to $t$ visiting states from $S'$ only. A strongly connected component (SCC) of $M$ is a maximal strongly connected subset of $S$.

The probability measure for finite paths $\pi \in Paths_{\text{fin}}^M$ is defined by $Pr_{\text{fin}}^M(\pi) = \prod_{(s_i,s_{i+1}) \in \pi} P(s_i,s_{i+1})$. For a set $R \subseteq Paths_{\text{fin}}^M$ of paths we have $Pr_{\text{fin}}^M(R) = \sum_{\pi \in R} Pr_{\text{fin}}^M(\pi)$ with $R' = \{ \pi \in R | \forall \pi' \in R. \pi'$ is no prefix of $\pi \}$.

The syntax of probabilistic computation tree logic (PCTL) [6] is given by $^3$

$$\varphi ::= p | \neg \varphi | \varphi \land \varphi | \mathbb{P}_{\leq \lambda}(\varphi U \varphi)$$

for (state) formulae with $p \in AP$, $\lambda \in [0,1] \subseteq \mathbb{R}$, and $\sim \in \{<,\leq,\geq,>\}$. We define $\Diamond$ and $\Box$ in the usual way.

For a property $P_{\leq \lambda}(\varphi_1 U \varphi_2)$ refuted by $M$, a counterexample is a set $C \subseteq Paths_{\text{fin}}^M$, $Pr_{\text{fin}}^M(C) > \lambda$ of finite paths starting in an initial state and satisfying $\varphi_1 U \varphi_2$. For $P_{\leq \lambda}(\varphi_1 U \varphi_2)$, the probability mass has to be at least $\lambda$. We consider upper probability bounds; see [5] for the reduction of lower bounds to this case.

Model checking of PCTL properties can be reduced to checking properties of the form $P_{\leq \lambda}(\Diamond \varphi)$. The $\varphi$-states are also called target states. We concentrate on this case and assume DTMCs to have single initial and target states. Note that each DTMC can be equivalently transformed to satisfy these requirements.

### 3 SCC-based Model Checking

Next we recall our model checking algorithm from [2]. Given a DTMC $M$, we are interested in the total probability of reaching its target state from its initial

---

$^3$ In this paper we only consider unbounded properties.
We use the notation \( \text{Inp} \) for all transitions from each input state \( s \) in the following definition.

**Definition 2.** The DTMC induced by \( S' \) in \( M \), written \( \text{DTMC}(S', M) \), is \( M_{\text{ind}} = (S_{\text{ind}}, I_{\text{ind}}, P_{\text{ind}}, L_{\text{ind}}) \) with

1. \( S_{\text{ind}} = S' \cup \text{Out}^M(S') \),
2. \( \forall s \in S_{\text{ind}}. (I_{\text{ind}}(s) > 0 \leftrightarrow s \in \text{Inp}^M(S')) \),
3. \( P_{\text{ind}}(s, t) = \begin{cases} P(s, t) & \text{for } s \in S' \text{ and } t \in S_{\text{ind}}, \\ 1 & \text{for } s = t \in \text{Out}^M(S'), \\ 0 & \text{else.} \end{cases} \)
4. \( \forall s \in S_{\text{ind}}. L_{\text{ind}}(s) = L(s) \).

We use the notation \( \text{Inp}(M_{\text{ind}}) = \{ s \in S_{\text{ind}} \mid I_{\text{ind}}(s) > 0 \} \) and \( \text{Out}(M_{\text{ind}}) = \{ s \in S_{\text{ind}} \mid P_{\text{ind}}(s, s) = 0 \} \).

The model checking procedure replaces inside \( M \) the subgraph \( M_{\text{ind}} \) by a smaller subgraph \( M_{\text{abs}} \) with the input and output states as state set and transitions from each input state \( s \) to each output state \( t \) carrying the total probability mass \( P_{r_{\text{ind}}}(\text{Paths}_{\text{fin}}^M(s, t)) \).

**Definition 3.** Let \( \text{DTMC}(S', M) = M_{\text{ind}} = (S_{\text{ind}}, I_{\text{ind}}, P_{\text{ind}}, L_{\text{ind}}) \) and

\[
p_{s,t} = Pr_{r_{\text{fin}}}^M(\{ ss_1 \ldots s_n t \in \text{Paths}_{\text{fin}}^M \mid \forall 1 \leq i \leq n, s_i \neq s \land s_i \neq t \})
\]

for all \( s \in \text{Inp}(M_{\text{ind}}) \) and \( t \in \text{Out}(M_{\text{ind}}) \). We define \( \text{Abs}(M_{\text{ind}}) \), to be the DTMC \( M_{\text{abs}} = (S_{\text{abs}}, I_{\text{abs}}, P_{\text{abs}}, L_{\text{abs}}) \) with

1. \( S_{\text{abs}} = \text{Inp}(M_{\text{ind}}) \cup \text{Out}(M_{\text{ind}}) \),
2. \( I_{\text{abs}}(s) = I_{\text{ind}}(s) \) for all \( s \in S_{\text{abs}} \),
3. \( P_{\text{abs}}(s, t) = \begin{cases} p_{s,t} & \text{for } s \in \text{Inp}(M_{\text{ind}}), t \in \text{Out}(M_{\text{ind}}), \\ 1 & \text{for } s = t \in \text{Out}(M_{\text{ind}}), \\ 0 & \text{else.} \end{cases} \)
4. \( L_{\text{abs}}(s) = L_{\text{ind}}(s) \) for all \( s \in S_{\text{abs}} \).

Next we formalize the abstraction and the concretization of an SCC.

**Definition 4.** Let \( \text{DTMC}(S', M) = M_1 = (S_1, I_1, P_1, L_1) \), and \( M_2 = (S_2, I_2, P_2, L_2) \) a DTMC satisfying \( S_2 \cap (S' \setminus S_1) = \emptyset \) such that either \( M_2 = \text{Abs}(M_1) \) or \( M_1 = \text{Abs}(M_2) \). Then the result of the substitution of \( M_1 \) by \( M_2 \) in \( M \), written \( M[M_2/M_1] \), is the DTMC \( M_{\text{sub}} = (S_{\text{sub}}, I_{\text{sub}}, P_{\text{sub}}, L_{\text{sub}}) \) with
Algorithm 1

Model_check(DTMC $M = (S, I, P, L)$, PCTL-formula $\mathbb{P}_{\sim \lambda} (\Diamond p)$)
begin
$(M, Sub) := $ Abstract_SCC($M, \emptyset$); (1)
result := \left( \sum_{s \in \text{Inp}(M)} \sum_{t \in \text{Out}(M)} (I(s) \cdot P(s, t)) \sim \lambda \right); (2)
return (result, M, Sub) (3)
end

Abstract_SCC(DTMC $M = (S, I, P, L)$, Abstractions $Sub$)
begin
for all non-bottom SCCs $K$ in DTMC($S \setminus \text{Inp}(M), M$) do (4)
$M_K := $ DTMC($K, M$); (5)
$M := M[M_{abs}^K/M_K]$ (6)
end for (7)
$M_{abs} := $ Abs($M$); $Sub := Sub \cup \{(M, M_{abs})\}$; (8)
return ($M_{abs}, Sub$) (9)
end

1. $S_{sub} = (S \setminus S_1) \cup S_2$,
2. $I_{sub}(s) = I(s)$ for $s \in S_{sub}$ and 0 otherwise,
3. $P_{sub}(s, t) = P^2_2(s, t)$ for $s \in (S_2 \setminus \text{Out}(M_2))$ and $t \in S_2$, and $P(s, t)$ otherwise,
4. $L_{sub}(s) = L_2(s)$ for $s \in S_2$ and $L(s)$ otherwise.

The replacement of an SCC by its abstraction and vice versa does not affect the total probabilities of reaching a target state from an initial state in $M$ [1].

To compute the abstraction $M_{abs}$ of an induced DTMC $M_{ind}$, we determine the probabilities $p_{s,t}$ recursively as follows. We detect all non-bottom SCCs in $M_{ind}$ that do not contain any input states of $M_{ind}$, and replace them by their abstractions recursively. The result is a DTMC $M'_{ind}$ which is loop-free in case $M_{ind}$ has a single input state (multiple input states need a special treatment, see [2]), such that the probabilities $p_{s,t}$ can be computed easily.

The model checking algorithm is shown in Algorithm 1. We use a global variable $Sub$ to store the pairs of abstracted DTMCs and their abstractions for the concretization during counterexample generation.\footnote{Instead of copying, the implementation uses different markings to specify sub-graphs.}

4 Counterexample Generation

Our computation is based on the detection of single paths, which we use to determine a subgraph (closure) of the original system. We call the closure a critical subsystem if its paths form a counterexample for the violated property.

The closure is computed according to a selection $m \subseteq S \times S$. We use $\text{extend}^M: (2^{S \times S} \times \text{Paths}_{fin}^M) \rightarrow 2^{S \times S}$ defined by $\text{extend}(m, \pi) = \{(s, s') \in S \times S \mid (s, s') \in m \lor (s, s') \in \pi\}$ to extend a selection $m$ with the transitions of a path $\pi$.\footnote{Instead of copying, the implementation uses different markings to specify sub-graphs.}
Algorithm 2

SearchAbstractCex(DTMC $M$, PCTL-formula $\mathbb{P}_{\sim \lambda} (\diamond p)$)
begin
  (result, $M_{ce}$, Sub) := ModelCheck($M$, $\mathbb{P}_{\sim \lambda} (\diamond p)$); (10)
  if result = true then return $\perp$ (11)
  else
    $m_{max} := \{(s_0, t)\}$; (13)
    while true do
      $m_{min} := m_{max}$; (15)
      (ready, $M_{ce}$, $m_{min}$, $m_{max}$) := Concretize($M_{ce}$, $m_{min}$, $m_{max}$, Sub); (16)
      if (ready = true) then return $\text{closure}_{M_{ce}}(m_{max})$ (17)
      else $m_{max} := \text{CriticalSubsystem}(M_{ce}, m_{min}, m_{max}, \mathbb{P}_{\sim \lambda} (\diamond p))$; (18)
      end if
      end while
  end if
end

Definition 5 (Closure). For a DTMC $M = (S, I, P, L)$, target state $t$, and a selection $m \subseteq S \times S$, the closure $\text{closure}_M(m) = (S_{cl}, I_{cl}, P_{cl}, L_{cl})$ of $m$ in $M$ is given by $S_{cl} = S \cup \{s_{\perp}\}$, $I_{cl}(s) = I(s)$, $L_{cl}(s) = L(s)$ for $s \in S$ and $I_{cl}(s_{\perp}) = 0, L_{cl}(s_{\perp}) = \emptyset$ and

$$P_{cl}(s, s') = \begin{cases} 
  P(s, s') & \text{for } (s, s') \in m, \\
  1 - \sum_{(s, s'') \in m} P(s, s'') & \text{for } s \in S \setminus \{t\} \text{ and } s' = s_{\perp}, \\
  1 & \text{for } s = s' = t \text{ or } s = s' = s_{\perp}, \\
  0 & \text{otherwise.} 
\end{cases}$$

Given a PCTL property $\varphi$, we call a DTMC $M'$ a critical subsystem of $M$ for $\varphi$ if $M' = \text{closure}_M(m)$ for some selection $m$ and $M'$ violates $\varphi$.

4.1 The Basic Hierarchical Algorithm

We compute counterexamples in a hierarchical manner (see Algorithm 2): Intuitively, at first we compute a critical subsystem for the resulting abstract DTMC of the model checking procedure. Then we refine the DTMC stepwise hand in hand with its critical subsystem. For each refinement step, the abstract and the refined critical subsystems differ only in states and transitions affected by the refinement step.

The initial critical subsystem is given by the closure $\text{closure}_{M_{ce}}(m_{max})$ where the selection $m_{max}$ contains the only transition from the initial state $s_0$ to the target state $t$ of $M_{ce}$ (line 13). Note that this initial subsystem represents all paths of $M$ from its initial to its target state.

The Concretize method (Algorithm 3) concretizes some heuristically determined abstract states in $M_{ce}$. Thereby we remove all transitions from $m_{max}$ that were removed by the concretization and add all transitions added by the
Algorithm 3

Concretize(DTMC $M_{ce}$, Selection $m_{min}$, Selection $m_{max}$, Abstractions $Sub$)
begin
  first = true;
  while true do
    $s_a := \text{ChooseAbstractState}(closure^{M_{ce}}(m_{max}));$
    if ($s_a = \bot$) then return (first, $M_{ce}$, $m_{min}$, $m_{max}$)
    else
      first := false;
      Let ($M_{abs}$, $M_{con}$) $\in Sub$ s.t. $s_a \in \text{Inp}(M_{abs});$
      $Tr_{abs} := \{(s, s') \in S_{abs} \times S_{abs} \mid s \notin \text{Out}(M_{abs}) \land P_{abs}(s, s') > 0\};$
      $m_{min} := m_{min} \setminus Tr_{abs};$
      $m_{max} := (m_{max} \setminus Tr_{abs}) \cup Tr_{con};$
      $M_{ce} := M_{ce}[M_{con}/M_{abs}];$
    end if
  end while
end

Algorithm 4 Global Search

CriticalSubsystem(DTMC $M_{ce}$, Selection $m_{min}$, Selection $m_{max}$, Formula $\mathcal{P}_{\sim \lambda} (\diamond p)$)
begin
  $k := 0; \quad M_{max} := closure^{M_{ce}}(m_{max});$
  Let $s_0$ be the initial and $t$ the target state of $M_{max};$
  repeat
    $k := k + 1; \quad \pi := \text{FindNextPath}(s_0, t, M_{max}, k); \quad m_{min} := \text{extend}(m_{min}, \pi);$
  until ModelCheck($closure^{M_{ce}}(m_{min})$, $\mathcal{P}_{\sim \lambda} (\diamond p)$) reports violation;
  return $m_{min};$
end

Concretization (line 31). If the closure of $m_{max}$ in $M_{ce}$ represents a counterexample, then also the closure of the updated selection $m_{max}$ in the concretization of $M_{ce}$ represents a counterexample with the same probability. However, this counterexample may be unnecessarily large. CriticalSubsystem searches for a smaller selection included in $m_{max}$ that still contains all transitions that were not affected by the concretization.

4.2 Search Algorithms

Global search. An implementation for CriticalSubsystem, which we call the global search algorithm, is proposed in Algorithm 4. Similarly to [5], we search for most probable paths from the initial state to the target state in the subsystem $M_{max} = closure^{M_{ce}}(m_{max})$ (line 35). After a next most probable path has been found (line 38), the algorithm extends $m_{min}$ with the found path (line 38). This procedure is repeated until the closure of $m_{min}$ is large enough to represent a counterexample (line 39).
Algorithm 5 Local Search

CriticalSubsystem(DTMC $M_{ce}$, Selection $m_{min}$, Selection $m_{max}$, PCTL-formula $P_{\sim \lambda} (\Diamond p)$) begin

\[ M_{cl} := \text{closure}^{M_{ce}}(m_{min}); \] (41)

while ModelCheck($M_{cl}, P_{\sim \lambda} (\Diamond p)$) reports satisfaction do

\[ M_{\text{search}} := \text{closure}^{M_{ce}}(m_{max}\setminus m_{min}); \] (42)

\[ \Pi := \{ \pi' \in \text{Paths}_{m_{\text{search}}}^{M_{\text{search}}}(s,t) \mid s \in \text{Imp}(M_{\text{search}}) \land t \in \text{Out}(M_{\text{search}}) \}; \] (43)

\[ \pi := \arg \max_{\pi \in \Pi} P_{\text{fin}}(\pi); \] (44)

\[ m_{min} := \text{extend}(m_{min}, \pi); \] (45)

\[ M_{cl} := \text{closure}^{M_{ce}}(m_{min}); \] (46)

end while

return $m_{min}$ (47)

end

Local search. The global search is complete, but it may find most probable paths which do not extend the minimal selection $m_{min}$. This can be time-consuming, e.g., when many different traversals of loops are considered.

Our second implementation for CriticalSubsystem (Algorithm 5), which we call the local search, overcomes this problem and finds only paths that extend the minimal selection and increase the target reachability probability of its closure. Instead of searching for paths from the initial to the target state, it aims at finding most probable path fragments that connect fragments of already found paths to new paths. The path fragments should, as the paths for the global search, lie in the closure of $m_{max}$. But this time they should (1) start at states reachable from an initial state via transitions of $m_{min}$, (2) end in states from which the target state is reachable via transitions from $m_{min}$, and (3) contain transitions from $m_{max}\setminus m_{min}$ only. I.e., we only search for path fragments in the subgraphs inserted by the last concretization step, which connect path fragments in the closure of $m_{min}$ to whole paths from the initial to the target state.

5 Experimental results

We developed a C++ implementation with exact arithmetic for both search algorithms, and used it to run experiments on a 2.4 GHz dual core CPU with 4 GB RAM. We used PRISM [9] to generate models for different instances of the parametrized synchronous leader election protocol [7] and the crowds protocol [10].

The global and the local search work on hierarchical data types. However, they can also directly be applied to concrete models. We consider this non-hierarchical approach to obtain a fair comparison to [5]. Table 1 compares the global method with the $k$-shortest path search for the leader election protocol, where the probability of reaching a target state is always 1. Table 2 depicts results for the crowds benchmark additionally containing the local search. The global search finds paths in the same order as $k$-sp, but due to the closure computation earlier termination, a significantly smaller number of needed paths, and therefore a
Table 1. Results for the leader benchmark on concrete models (TO > 1h)

<table>
<thead>
<tr>
<th></th>
<th>states</th>
<th>transitions</th>
<th>prob. threshold</th>
<th>k-sp</th>
<th># paths</th>
<th># states</th>
<th>prob.</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>3902</td>
<td>5197</td>
<td>0.92</td>
<td>1193</td>
<td>3593</td>
<td>0.9205</td>
<td></td>
</tr>
<tr>
<td></td>
<td>12302</td>
<td>16397</td>
<td>0.93</td>
<td>8043</td>
<td>3903</td>
<td>0.9302</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>0.95</td>
<td>41636</td>
<td>3903</td>
<td>0.9501</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>0.95</td>
<td>3892</td>
<td>11690</td>
<td>0.9502</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>0.96</td>
<td>16397</td>
<td>12302</td>
<td>0.9600</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>0.97</td>
<td>57728</td>
<td>12302</td>
<td>0.9700</td>
<td></td>
</tr>
</tbody>
</table>

Table 2. Results for the crowds benchmark on concrete models (TO > 1h)

<table>
<thead>
<tr>
<th></th>
<th>states</th>
<th>transitions</th>
<th>total prob.</th>
<th>prob. threshold</th>
<th>k-sp</th>
<th># paths</th>
<th># states</th>
<th>prob.</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>396</td>
<td>576</td>
<td>0.1891</td>
<td>0.12</td>
<td>1301</td>
<td>3974</td>
<td>0.1339</td>
<td></td>
</tr>
<tr>
<td></td>
<td>3515</td>
<td>6035</td>
<td>0.2346</td>
<td>0.15</td>
<td>26184</td>
<td>39874</td>
<td>0.1514</td>
<td></td>
</tr>
<tr>
<td></td>
<td>18817</td>
<td>32677</td>
<td>0.4270</td>
<td>0.21</td>
<td>3947</td>
<td>48864</td>
<td>0.1043</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>0.23</td>
<td>26981</td>
<td>18016</td>
<td>0.1018</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>0.2</td>
<td>1071</td>
<td>634</td>
<td>0.1251</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>0.25</td>
<td>11690</td>
<td>11815</td>
<td>0.2300</td>
<td></td>
</tr>
</tbody>
</table>

smaller number of computation steps are achieved. For probability thresholds near the total probability, the number of paths for k-sp is several orders of magnitude larger. The number of considered states can also be reduced significantly. The local search not only leads to smaller critical subsystems in most cases, but also needs a much smaller number of found path fragments in comparison to the global search. The probability mass for all types of counterexamples is always very close to the specified probability threshold. Note that for our methods we model check only extended subsystems, while for the local search actually every new path extends the system.

The search for hierarchical counterexamples is motivated by their usefulness and understandability. The results in Table 3 show that the hierarchical search leads to critical subsystems of comparable size (the third last column is the hierarchical version of the global search in the second last column of Table 2). The number of found paths is much larger in the hierarchical approach, because we have to search at each abstraction level. However, due to abstraction, the found paths are shorter, especially for the local search, and the concretization up to the concrete level seems not necessary for many cases. We did experiments using different heuristics for the number of abstract states that are concretized in one step (e.g., either a single one or \(\sqrt{n}\) with \(n\) the number of abstract states). We
Table 3. Results for a crowds instance (18817 states, 32677 transitions, 0.2 probability threshold) on the hierarchical model

<table>
<thead>
<tr>
<th>search type</th>
<th></th>
<th>global</th>
<th></th>
<th>local</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td># abstract states to concretize in one step</td>
<td></td>
<td>✓</td>
<td>single</td>
<td>✓</td>
<td>single</td>
</tr>
<tr>
<td>heuristic to choose the next abstract state</td>
<td>prob</td>
<td>none</td>
<td>prob</td>
<td>none</td>
<td>prob</td>
</tr>
<tr>
<td># paths</td>
<td>113525</td>
<td>912455</td>
<td>38379</td>
<td>504881</td>
<td>496</td>
</tr>
<tr>
<td># closures</td>
<td>728</td>
<td>730</td>
<td>728</td>
<td>729</td>
<td>496</td>
</tr>
<tr>
<td># states</td>
<td>457</td>
<td>457</td>
<td>458</td>
<td>457</td>
<td>319</td>
</tr>
<tr>
<td># refinements</td>
<td>13</td>
<td>10</td>
<td>37</td>
<td>37</td>
<td>9</td>
</tr>
</tbody>
</table>

also tried two different heuristics for the choice of the next abstract state, either being just the next one found (“none”), or the one whose outgoing transitions have the maximal average probability (“prob”).

References