Polynomial Time Algorithms for Testing Probabilistic Bisimulation and Simulation

Christel Baier

Fakultät für Mathematik & Informatik
Universität Mannheim, 68131 Mannheim, Germany
baier@pi1.informatik.uni-mannheim.de

Abstract. Various models and equivalence relations or preorders for probabilistic processes are proposed in the literature. This paper deals with a model based on labelled transition systems extended to the probabalistic setting and gives an $\mathcal{O}(n^2 \cdot m)$ algorithm for testing probabilistic bisimulation and an $\mathcal{O}(n^5 \cdot m^2)$ algorithm for testing probabilistic simulation where n is the number of states and m the number of transitions in the underlying probabilistic transition systems.

1 Introduction

Transition systems have proved to be very useful for modelling concurrent processes. A variety of widely accepted equivalence relations and preorders for such systems support the use of transition systems for the design and verification of concurrent systems. In this context, testing equivalences and preorders become important and have been studied e.g. in [3, 4, 8, 11, 17]. For instance, (strong) bisimulation can be decided in time $\mathcal{O}(m \cdot \log n)$ [22], weak bisimulation in time $\mathcal{O}(n^3)$ [3, 17] and strong and weak simulation in time $\mathcal{O}(n^4 \cdot m)$ [4] where n is the number of states and m the number of transitions of the underlying transition system.

In recent years, many researchers have focussed on reasoning about probabilistic distributed transition systems, see e.g. [15, 18, 23, 25, 28, 29, 30]. A lot of work has been done to extend those models and methods which have been successful for the non-probabilistic case to probabilistic systems. In the literature a variety of models for probabilistic processes has been proposed, most of them based on transition systems. Two kinds of models can be distinguished: on the one hand, models that replace the concept of non-determinism by probabilistic choice, e.g. [5, 13, 18, 26, 28], on the other hand, models which distinguish between non-deterministic and probabilistic choice, e.g. [6, 12, 16, 25, 27, 30]. As pointed out in [27], the distinction between non-determinism and probabilistic choice is essential for concurrent probabilistic systems since some states of a concurrent system are inherently non-deterministic.

Several kinds of equivalences and preorders for probabilistic processes are proposed: [5, 16, 30, 28] consider testing preorders for probabilistic processes. Probabilistic bisimulation for processes whose behaviour are described by "determinsitic" probabilistic transition systems are introduced in [18]. [25] extends

probabilistic bisimulation to non-deterministic probabilistic transition systems and defines a notion of probabilistic simulation which refines Milners notion of a simulation for non-probabilistic transition systems [21]. [15] defines an alternative notion of a simulation which relates a process given by a probabilistic transition system and a specification which is given by a "generalized" probabilistic transition system.

Various authors presented model-checking-algorithms for the verification of probabilistic processes e.g. [1, 6, 13, 14, 19, 23, 24, 27]. But – as far as the author knows – algorithms for testing probabilistic (bi-)simulation are missing until now. In this paper we present algorithms for testing probabilistic simulation and bisimulation in the sense of [18, 25]. The main idea of testing simulation is to reduce the question of whether a state s of a probabilistic transition system simulates a state s' to a maximum flow problem in a suitable network. Using the $\mathcal{O}(n^3)$ algorithm of Malhotra et al [20] to determine the maximum flow we get an $\mathcal{O}(n^5 \cdot m^2)$ algorithm for testing probabilistic simulation where n is the number of states and m the number of transitions. The idea for testing bisimulation is similar to the non-probabilistic case [17, 22]: the algorithm for testing probabilistic bisimulation is based on refinement steps which split a given partition of states into a finer one. The resulting time complexity of our algorithm is $\mathcal{O}(n^2 \cdot m)$.

The remainder of the paper is organized as follows: Section 2 introduces the notions of a probabilistic transition system, probabilistic bisimulation and simulation. Section 3 presents the algorithm for testing probabilistic simulation, section 4 the algorithm for deciding probabilistic bisimulation. Section 5 contains some concluding remarks.

2 Probabilistic transition systems

In this section we present the notions of probabilistic transition systems, bisimulation and simulation. Our model of probabilistic transition systems is closely related to those of [16, 30], to the "simple probabilistic automata" of [25] and "concurrent Markov chains" considered e.g. in [6, 12, 27].

A distribution on a finite set S is a function $\mu : S \to [0,1]$ such that $\sum_{s \in S} \mu(s) = 1$. We extend a distribution μ to a function which assigns to each subset U of S the probability $\mu(U) = \sum_{s \in U} \mu(s)$. In what follows, we suppose Act to be a nonempty and finite set of actions. A *probabilistic transition system* is a pair $\mathcal{S} = (S, \to)$ where S is a finite set of states and $\to$ a finite transition relation, i.e. $\to$ is a finite subset of $S \times \mathrm{Act} \times \mathcal{D}(S)$ where $\mathcal{D}(S)$ denotes the set of distributions on S. We write $s \xrightarrow{\alpha} \mu$ instead of $(s, \alpha, \mu) \in \to$. Informally, the outgoing transitions $s \xrightarrow{\alpha} \mu$ represent the non-deterministic alternatives in the state s. It is convenient to suppose that a scheduler resolves the non-deterministic choices. A transition $s \xrightarrow{\alpha} \mu$ asserts that in state s the action α can be performed and with probability $\mu(t)$ the state t is reached afterwards, i.e. every transition represents a probabilistic choice. (Finite-state) probabilistic processes can be described by a probabilistic transition system and

an initial state (or alternatively a distribution on the possible initial states). In what follows a transition system means a probabilistic transition system. By a non-probabilistic transition system we mean a transition system where for all transitions $s \xrightarrow{\alpha} \mu$: there is a state t with $\mu(t) = 1$. Following [18, 25] we define (probabilistic) bisimulation and simulation:

Definition 1. Let $(S, \rightarrow)$ be a transition system. A *bisimulation* on S is an equivalence relation R on S such that for all $(s, s') \in R$: If $s \xrightarrow{\alpha} \mu$ then there is a transition $s' \xrightarrow{\alpha} \mu'$ with $\mu(A) = \mu'(A)$ for all $A \in S/R$. Here S/R denotes the set of equivalence classes w.r.t. R. Two states s_1 and s_2 are called *bisimilar* (denoted by $s_1 \sim s_2$) iff there exists a bisimulation which contains (s_1, s_2).

An alternative description of bisimulation is based on weight functions for distributions [15]:

Definition 2. Let S be a finite set, $R \subseteq S \times S$ and $\mu, \mu' \in \mathcal{D}(S)$. A *weight function* for (μ, μ') w.r.t. R is a function $\delta : S \times S \rightarrow [0, 1]$ which satisfies:

1. For all $s, s' \in S$: $\sum_{s' \in S} \delta(s, s') = \mu(s), \quad \sum_{s \in S} \delta(s, s') = \mu'(s')$
2. If $\delta(s, s') > 0$ then $(s, s') \in R$.

Let $(S, \rightarrow)$ be a transition system and R an equivalence relation on S. Then R is a bisimulation if and only if for all $(s, s') \in R$: Whenever $(s, s') \in R$ and $s \xrightarrow{\alpha} \mu$ then there exists a transition $s' \xrightarrow{\alpha} \mu'$ and a weight function for (μ, μ') w.r.t. R. Intuitively, the weight function δ shows how to split the probabilities $\mu(s)$ and $\mu'(s')$, s, $s' \in S$, so that the relation R is preserved: we "combine" the $\delta(s, s')$-part of s and s'. As in the non-probabilistic case, simulation is defined as "uni-directional bisimulation": in the above characterization of bisimulation we drop the requirement that R is an equivalence relation.

Definition 3. Let $(S, \rightarrow)$ be a transition system. A *simulation* for $(S, \rightarrow)$ is a subset R of $S \times S$ such that for all $(s, s') \in R$: Whenever $s \xrightarrow{\alpha} \mu$ then there exists a transition $s' \xrightarrow{\alpha} \mu'$ and a weight function δ for (μ, μ') w.r.t. R. We say s *implements* s' (denoted by $s \sqsubseteq s'$) iff there exists a simulation which contains (s, s').

In the non-probabilistic case this notion of a simulation agrees with Milners notion of a simulation [21]. This is because the only weight function for (μ, μ') where μ, μ' are distributions with $\mu(s) = \mu'(s') = 1$ is $\delta(u, u') = 0$ if $(u, u') \neq (s, s')$ and $\delta(s, s') = 1$. Hence if $(S, \rightarrow)$ is a non-probabilistic transition system and $R \subseteq S \times S$ then R is a simulation in the sense of Definition 3 if and only if R is a simulation in the sense of Milner. It is clear that $\sqsubseteq$ is a preorder whose kernel $\sim_{\text{sim}} = \sqsubseteq \cap \sqsubseteq^{-1}$ is coarser than bisimulation equivalence, i.e. $s \sim s'$ implies $s \sim_{\text{sim}} s'$. As in the non-probabilistic case, $\sim_{\text{sim}}$ does not coincide with bisimulation.

Example 4. Let $(S, \rightarrow)$ be the transition system where $S = \{s_0, \ldots, s_5\}$ and

$$s_0 \xrightarrow{\alpha} \mu,\ s_5 \xrightarrow{\alpha} \mu',\ s_2 \xrightarrow{\beta} \rho,\ s_3 \xrightarrow{\beta} \rho,\ s_3 \xrightarrow{\gamma} \rho,\ s_4 \xrightarrow{\alpha} \rho.$$

Here $\rho(s_1) = 1$, $\mu(s_1) = \mu(s_2) = \mu(s_3) = 1/3$ and $\mu'(s_1) = 1/4$, $\mu'(s_3) = 17/24$ and $\mu'(s_4) = 1/24$. Then

$$s_1 \sqsubseteq s_2 \sqsubseteq s_3,\ s_1 \sqsubseteq s_4 \sqsubseteq s_0 \sqsubseteq s_5.$$

The weight function δ for (μ, μ') w.r.t. $\sqsubseteq$ is given by: $\delta(s_1, s_1) = 1/4$, $\delta(s_1, s_3) = \delta(s_1, s_4) = 1/24$, $\delta(s_2, s_3) = \delta(s_3, s_3) = 1/3$. □

The result of Milner [21] that in every (image-)finite non-probabilistic transition system bisimulation can be approximated by "finitary bisimulation" carries over to the probabilistic case. If $(S, \rightarrow)$ is a transition system then we define inductively equivalence relations $\sim_n$ on S: $\sim_0 = S \times S$ and $s \sim_{n+1} s'$ if and only if: Whenever $s \xrightarrow{\alpha} \mu$ then there is a transition $s' \xrightarrow{\alpha} \mu'$ with $\mu(A) = \mu'(A)$ for all $A \in S/\sim_n$ and vice versa. Similarly, we define "finitary simulation": $s \sqsubseteq_0 s'$ for all states s, s' and $s \sqsubseteq_{n+1} s'$ iff whenever $s \xrightarrow{\alpha} \mu$ then there exists a transition $s' \xrightarrow{\alpha} \mu'$ and a weight function δ for (μ, μ') w.r.t. $\sqsubseteq_n$. As shown in [2]:

Lemma 5. *Let $(S, \rightarrow)$ be transition systems and s, $s' \in S$. Then*

(a) $s \sqsubseteq s'$ if and only if $s \sqsubseteq_n s'$ for all $n \geq 0$.
(b) $s \sim s'$ if and only if $s \sim_n s'$ for all $n \geq 0$.

3 Testing simulation

We present an $\mathcal{O}(n^5 \cdot m^2)$ algorithm for testing simulation where n is the number of states and m the number of transitions in the underlying transition system. The results of this section yield also an $\mathcal{O}(n^5 \cdot m^2)$ algorithm for testing bisimulation. In section 4 we improve the costs and give an $\mathcal{O}(n^2 \cdot m)$ algorithm for testing bisimulation. Lemma 6 shows that for a (finite) transition systems there is a natural number N which is polynomial in the size of the underlying transition system such that $\sqsubseteq = \sqsubseteq_N$. Our algorithm successively computes the relations $\sqsubseteq_0$, $\sqsubseteq_1$,..., $\sqsubseteq_N$. We show that the relation $\sqsubseteq_{j+1}$ can be derived from $\sqsubseteq_j$ by solving maximum flow problems in suitable networks.

Lemma 6. *Let $(S, \rightarrow)$ be a transition system, n the number of states in S and $N = n^2$. Then $\sim = \sim_N$ and $\sqsubseteq = \sqsubseteq_N$.*

Proof. We only show $\sqsubseteq = \sqsubseteq_N$. We have $\sqsubseteq_0 \supseteq \sqsubseteq_1 \supseteq \ldots$ and $s \sqsubseteq s'$ iff $s \sqsubseteq_j s'$ for all j (Lemma 5). Since $\sqsubseteq_0 = S \times S$ contains N elements there exists j with $0 \leq j \leq N$ and $\sqsubseteq_{j+1} = \sqsubseteq_j$. Then $\sqsubseteq_j = \sqsubseteq_i$ for all $i \geq j$ and hence $\sqsubseteq = \sqsubseteq_j = \sqsubseteq_N$. □

Lemma 6 tells us that in order to compute the simulation preorder $\sqsubseteq$ for finite transition systems one has to compute the relation $\sqsubseteq_{n^2}$. We do this by successively computing the relations $\sqsubseteq_j$, $j = 0, 1, \ldots, N$. In order to compute the

relation $\sqsubseteq_{j+1}$ (where $\sqsubseteq_j$ is already computed) we need an algorithm which tests whether or not a weight function for given distributions w.r.t. $\sqsubseteq_j$ exists. We present a polynomial time algorithm which tests whether a weight function for distributions μ, μ' w.r.t. a given relation R exists. The idea of the algorithm is to reduce the problem of finding a weight function to a maximum flow problem in networks. Algorithms to compute the maximum flow are given in [7, 10, 20]. For further details about maximum flow problems see e.g. [9].

A network is a tuple $\mathcal{N} = (N, E, \bot, \top, c)$ where (N, E) is a finite directed graph – where N denotes the set of nodes, $E \subseteq N \times N$ the set of edges – with two specified nodes $\bot$ (the source) and $\top$ (the sink) and a capacity c, i.e. a function c which assigns to each edge $(v, w) \in E$ a non-negative number $c(v, w)$. A flow function f is a function which assigns to edge e a real number $f(e)$ such that

1. For all edges e: $0 \leq f(e) \leq c(e)$
2. Let $in(v)$ be the set of incoming edges to node v and $out(v)$ the set of outgoing edges from node v. Then for each node $v \in N \setminus \{\bot, \top\}$:

$$\sum_{e \in in(v)} f(e) = \sum_{e \in out(v)} f(e)$$

The flow $\mathcal{F}(f)$ of f is given by

$$\mathcal{F}(f) = \sum_{e \in out(\bot)} f(e) - \sum_{e \in in(\bot)} f(e).$$

The maximum flow in $\mathcal{N}$ is the supremum over the flows $\mathcal{F}(f)$ where f is a flow function in $\mathcal{N}$.

Let S be a finite sets, R a subset of $S \times S$ and let μ, $\mu' \in \mathcal{D}(S)$. Let $S' = \{t' : t \in S\}$ where t' are pairwise distinct "new" states (i.e. $t' \notin S$). We choose new elements $\bot$ and $\top$ not contained in $S \cup S'$, $\bot \neq \top$. We associate with (μ, μ') the following network $\mathcal{N}(\mu, \mu', R)$: The nodes are the elements of S and S' and $\bot$ (the source) and $\top$ (the sink), i.e. $N = \{\bot, \top\} \cup S \cup S'$. The edges are

$$E = \{(s, t') : (s, t) \in R\} \cup \{(\bot, s) : s \in S\} \cup \{(t', \top) : t \in S\}.$$

The capacities $c(e) \in [0, 1]$ are given by: $c(\bot, s) = \mu(s)$, $c(t', \top) = \mu'(t)$ and $c(s, t') = 1$.

Lemma 7. *The following are equivalent:*

(i) There exists a weight function δ for (μ, μ') w.r.t. R.
(ii) The maximum flow in $\mathcal{N}(\mu, \mu', R)$ is 1.

Proof. (i) $\Longrightarrow$ (ii): For each flow function f in $\mathcal{N}(\mu, \mu', R)$:

$$\mathcal{F}(f) = \sum_{s \in S} f(\bot, s) \leq \sum_{s \in S} c(\bot, s) = \sum_{s \in S} \mu(s) = 1.$$

Let δ be a weight function for (μ, μ') w.r.t. R. Then we define a flow function f as follows: $f(\bot, s) = \mu(s)$, $f(t', \top) = \mu'(t)$, $f(s, t') = \delta(s, t)$. Then $\mathcal{F}(f) = 1$.

Hence the maximum flow of $\mathcal{N}(\mu,\mu',R)$ is 1.
(ii) $\Longrightarrow$ (i): Let f be a flow function with $\mathcal{F}(f)=1$. Since $f(\bot,s)\leq c(\bot,s)=\mu(s)$ and since

$$\sum_{s\in S} f(\bot,s) \;=\; \mathcal{F}(f) \;=\; 1 \;=\; \sum_{s\in S}\mu(s)$$

we get $f(\bot,s)=\mu(s)$ for all $s\in S$. Similarly, we get $f(t',\top)=\mu'(t)$ for all $t\in S$. Let $\delta(s,t)=f(s,t')$ for all $(s,t)\in R$ and $\delta(s,t)=0$ if $(s,t)\notin R$. Then

$$\sum_{t\in S}\delta(s,t) \;=\; \sum_{t\in S} f(s,t') \;=\; f(\bot,s) \;=\; \mu(s)$$

and similarly $\sum_{s\in S}\delta(s,t) = \mu'(t)$. Hence δ is a weight function for (μ,μ') w.r.t. R. □

With Lemma 7 we get an algorithm which tests whether a weight function for distributions μ, μ' w.r.t. a relation R exists: We apply an algorithm for finding the maximum flow F in $\mathcal{N}(\mu,\mu',R)$. The maximum flow in $\mathcal{N}(\mu,\mu',R)$ can be computed e.g. with the $\mathcal{O}(n^3)$ algorithm of Malhotra et al [20] where n is the cardinality of S.

Algorithm 1.
Input: *a finite set S, distributions μ, $\mu'\in\mathcal{D}(S)$ and $R\subseteq S\times S$*
Output: *a weight function δ for (μ,μ') w.r.t. R if there exists one, "No" otherwise.*
Method: *Compute the maximum flow F of the network $\mathcal{N}(\mu,\mu',R)$ and a flow function f with $\mathcal{F}(f)=F$. If $F<1$ then answer "No" else answer "Yes" and return*

$$\delta(s,t) \;=\; \begin{cases} 0 & : \text{if } (s,t)\in S\times S\setminus R \\ f(s,t') & : \text{if } (s,t)\in R. \end{cases}$$

Lemma 6 and Algorithm 1 yield an algorithm for testing simulation:

Algorithm 2. *for testing probabilistic simulation*
Input: *a transition system $(S,\rightarrow)$*
Output: *the simulation preorder $R = \{(s,t)\in S\times S : s\sqsubseteq t\}$*
Method: *Let $N=n^2$ where n is the number of states of S and let $R_0 = S\times S$.*

For $j=1,\ldots,N$ do:
begin $R_j := R_{j-1}$
For all $(s,t)\in R_{j-1}$ do
begin For all transitions $s\xrightarrow{\alpha}\mu$ do:
If there does not exist a transition $t\xrightarrow{\alpha}\mu'$
such that Algorithm 1 yields a weight function
for (μ,μ') w.r.t. R_{j-1} then $R_j := R_j\setminus\{(s,t)\}$.
end
end
Return $R := R_N$.

It is clear that $R_j = \sqsubseteq_j$ and hence $R = \sqsubseteq_N = \sqsubseteq$. The time complexity of the algorithm is $\mathcal{O}(n^5 \cdot m^2)$ where m is the number of transitions and n the number of states. Algorithm 2 can be implemented in space $\mathcal{O}(n^2 + m)$ because the maximum flow problem (and hence Algorithm 1) can be solved in space $\mathcal{O}(n + m)$ and the representation of the sets R_j needs $\mathcal{O}(n^2)$ space. Similar to Algorithm 2, an $\mathcal{O}(n^5 \cdot m^2)$ algorithm for testing bisimulation can be given. In the next section we improve the time complexity giving an $\mathcal{O}(n^2 \cdot m)$ algorithm.

4 Testing bisimulation

Following the idea of [17] which gives an $\mathcal{O}(n \cdot m)$ algorithm for testing (non-probabilistic) bisimulation we present a method for deciding probabilistic bisimulation that works with refinement steps of partitions on the states. Given a transition system $(S, \rightarrow)$ we start with the trivial partition $X_0 = \{S\}$. Then we successively refine the partition X_k by substituting $B \in X_k$ by the set of equivalence classes w.r.t. the relation $s \equiv s'$ iff

1. Whenever $s \xrightarrow{\alpha} \mu$ then there exists a transition $s' \xrightarrow{\alpha} \mu'$ with $\mu(B) = \mu'(B)$ for all $B \in X_k$.
2. Whenever $s' \xrightarrow{\alpha} \mu'$ then there exists a transition $s \xrightarrow{\alpha} \mu$ with $\mu(B) = \mu'(B)$ for all $B \in X_k$.

At most after n refinement steps the partition X_k cannot be refined. Then X_k is the set of bisimulation equivalence classes.

Definition 8. A *partition* of a transition system $(S, \rightarrow)$ is a set X consisting of pairwise disjoint subsets B of S with $\bigcup_{B \in X} B = S$ and such that for all $B \in X$ and $s \in B$: the bisimulation equivalence class $[s]$ of s is contained in B.

In what follows, we shortly write $\mu(X)$ to denote the vector $(\mu(B))_{B \in X}$. If $s \in S$ then we define $X(s) = \{ (\alpha, \mu(X)) : s \xrightarrow{\alpha} \mu \}$. Each partition X is associated with an equivalence relation $\equiv_X$ on S: $s \equiv_X s'$ iff $X(s) = X(s')$. Having a partition X we split the elements of X into the equivalence classes w.r.t. $\equiv_X$: We define

$$\mathcal{J}(X) = \bigcup_{B \in X} B / \equiv_X .$$

Lemma 9. *Let $(S, \rightarrow)$ be a transition system and X a partition.*

(a) $S/\sim$ is a partition with $\mathcal{J}(S/\sim) = S/\sim$.
(b) $\mathcal{J}(X)$ is a partition.
(c) If $\mathcal{J}(X) = X$ then $X = S/\sim$.

Proof. (a) is clear. Let X be a partition of $(S, \rightarrow)$. It is clear that the sets $B \in \mathcal{J}(X)$ are pairwise disjoint and that the union of them is S. Each $B \in X$ can be written as disjoint union of bisimulation equivalence classes. This is because

$s \in B$ implies $[s] \subseteq B$. Hence whenever μ, μ' are distributions with $\mu(A) = \mu'(A)$ for all $A \in S/\sim$ then

$$\mu(B) \;=\; \sum_{A \in B/\sim} \mu(A) \;=\; \sum_{A \in B/\sim} \mu'(A) \;=\; \mu'(B)$$

for all $B \in X$. Hence $s \;\sim\; s'$ implies $s \;\equiv_X\; s'$. Therefore: If $C \in \mathcal{J}(B)$, $s \in C$ then C is the equivalence class of s w.r.t. $\equiv_X$ and hence contains $[s]$. We conclude that $\mathcal{J}(X)$ is a partition of $(S, \to)$. If $\mathcal{J}(X) = X$ then $\equiv_X$ is a bisimulation. Hence $s \;\equiv_X\; s'$ implies $s \;\sim\; s'$. Therefore $s \;\equiv_X\; s'$ iff $s \;\sim\; s'$ and hence $\mathcal{J}(X) \;=\; S/\sim$. □

Lemma 10. *Let $(S, \to)$ be a transition system with n states and m transitions and let X be a partition of $(S, \to)$. Then $\mathcal{J}(X)$ can be computed in time $\mathcal{O}(n \cdot m)$ and space $\mathcal{O}(n \cdot m)$.*

Proof. For fixed $B \in X$ and $\alpha \in \mathrm{Act}$ let $\mathcal{L}_{B,\alpha}$ be the set of all pairs $(\mathbf{p}, L)$ where L is a nonempty subset of B and $\mathbf{p} = (p_C)_{C \in X}$ a real vector such that $s \in L$ if and only if there exists a transition $s \xrightarrow{\alpha} \mu$ with $\mu(X) = \mathbf{p}$. Let $\mathcal{L}_B$ be the set of all pairs (α, L) where $\alpha \in \mathrm{Act}$ and $(\mathbf{p}, L) \in \mathcal{L}_{B,\alpha}$ for some $\mathbf{p}$. Then $s \;\equiv_X\; s'$ if and only if:

$$\text{Whenever } (\alpha, L) \in \mathcal{L}_B \text{ then } s \in L \text{ iff } s' \in L.$$

The idea of computing $B/\equiv_X$ is to calculate first the sets $\mathcal{L}_{B,\alpha}$, $\alpha \in \mathrm{Act}$, and then to derive the equivalence classes of B w.r.t. $\equiv_X$.

Computation of $\mathcal{L}_{B,\alpha}$. For each $\alpha \in \mathrm{Act}$ and $B \in X$ we construct a tree $T_{B,\alpha}$ by successively inserting nodes and edges. The edges of $T_{B,\alpha}$ are labelled by real numbers $p \in [0,1]$. Each leaf v has depth l and is labelled by an element $(\mathbf{p}(v), L(v)) \in \mathcal{L}_{B,\alpha}$.
Let $X = \{B_1, \ldots, B_l\}$. We start with $T_{B,\alpha}$ to be a tree of depth 0, i.e. a tree consisting of its root. Then for each transition $s \xrightarrow{\alpha} \mu$ where $s \in B$ we traverse the tree starting at the root. Reaching a node v of depth k we do:

- If $k < l$ and there is an outgoing edge from v leading to the node w labelled by $\mu(B_{k+1})$ then we pass the edge $v \to w$ and continue to travel through $T_{B,\alpha}$ with node w.
- If $k < l$ and there is no outgoing edge from v labelled by $\mu(B_{k+1})$ then we insert a new node w and an edge from v to w labelled by $\mu(B_{k+1})$. In the case $k+1 < l$ we continue to travel through $T_{B,\alpha}$ with node w. If $k+1 = l$ then w is a leaf and we define $L(w) = \{s\}$ and $\mathbf{p}(w) = \mu(X)$.
- If v is a leaf of depth l then we insert s into the set $L(v)$.

It is easy to see that the leaves of $T_{B,\alpha}$ represent the elements of $\mathcal{L}_{B,\alpha}$. Hence $\mathcal{L}_B$ is the set of all pairs $(\alpha, L(v))$ where v is a leaf in $T_{B,\alpha}$.

Complexity. First we observe that the tuples $\mu(X)$ (where μ ranges over all distributions s.t. $s \xrightarrow{\alpha} \mu$ is a transition) can be computed in $\mathcal{O}(n \cdot m)$ time: For each distribution μ we set $a_B = 0$ for all $B \in X$. Then for all states $s \in S$: If $s \in B$ then we replace a_B by $a_B + \mu(s)$. Finally $\mu(X) = (a_B)_{B \in X}$. The representation of the tuples $\mu(X)$ needs $\mathcal{O}(n \cdot m)$ space.

The construction of $T_{B,\alpha}$ needs $\mathcal{O}(m_{B,\alpha} \cdot l)$ steps where $m_{B,\alpha}$ is the number of transitions $s \xrightarrow{\alpha} \mu$, $s \in B$. Since $\sum_B \sum_\alpha m_{B,\alpha} = m$ and since the cardinality l of X is bounded by n we get: Ranging over all $B \in X$ and $\alpha \in \mathrm{Act}$ the construction of all trees $T_{B,\alpha}$, $B \in X$, $\alpha \in \mathrm{Act}$, takes $O(n \cdot m)$ steps. The set of paths from the root to a leaf in $T_{B,\alpha}$ is bounded by $m_{B,\alpha}$. Since l is the depth of the leaves $T_{B,\alpha}$ has at most $m_{B,\alpha} \cdot l + 1$ nodes. Hence, all trees $T_{B,\alpha}$ together have $\mathcal{O}(m \cdot n)$ nodes and $\mathcal{O}(m)$ leaves. The representation of the sets $L(v)$ needs $\mathcal{O}(|B|)$ space (where v is a leaf of a tree $T_{B,\alpha}$). Since $|B| \leq n$ the representation of all trees $T_{B,\alpha}$ together needs $\mathcal{O}(n \cdot m)$ space.

Computation of $B/\equiv_X$. We construct a binary tree T_B by successively inserting nodes and edges. Each leaf v has depth r and is labelled by a subset $C(v)$ of B. Let (α_i, L_i), $i = 1, \ldots, r$, be an enumeration of the elements of $\mathcal{L}_B$. (Note that $\alpha_i = \alpha_j$, $i \neq j$ is possible.) We start with a tree of depth 0, a tree consisting of its root. For each $s \in B$ we traverse the tree in the following way: If we have reached a node v of depth $k - 1$, $k \leq r$ then:

- If v has a left son w and $s \in L_k$ then we go to w.
- If v does not have a left son and $s \in L_k$ then we create a new left son w of v and go to w. If $k = r - 1$ then we set $C(w) = \{s\}$.
- If v has a right son w and $s \notin L_k$ then we go to w.
- If v does not have a right son and $s \notin L_k$ then we create a new right son w of v and go to w. If $k = r - 1$ then we set $C(w) = \{s\}$.

If we have reached a node v of depth r then we insert s into the set $C(v)$ of states associated with v.

Then we have: If v is a leaf and $v_0, v_1, \ldots, v_r = v$ the unique path from the root v_0 to v then $C(v) = L'_1 \cap L'_2 \cap \ldots \cap L'_r$ where $L'_i = L_i$ if v_i is the left son of v_{i-1} and $L'_i = B \setminus L_i$ if v_i is the right son of v_{i-1}. Let $\mathbf{p}_i = \mathbf{p}(v)$ where v is the leaf in T_{B,α_i} with $(\alpha_i, L_i) = (\alpha, L(v))$. Then for all $s \in B$: $s \in C(v)$ if and only if $X(s) = \{ (\alpha_i, \mathbf{p}_i) : L_i = L'_i \}$. Hence, if $s, s' \in B$ then $s \equiv_X s'$ if and only if $s, s' \in C(v)$ for some leaf v in T_B. We conclude:

$$B/\equiv_X = \{ C(v) : v \text{ is a leaf in } T_B \}$$

Complexity. The computation of T_B needs $\mathcal{O}(|B| \cdot r)$ steps. It is clear that the cardinality r of $\mathcal{L}_B$ is bounded by m. Hence we have the time complexity $\mathcal{O}(|B| \cdot m)$ for the construction of T_B. Each leaf in T_B has depth $r \leq m$. Since the leaves of T_B correspond to the equivalence classes w.r.t. $\equiv_X$ T_B has at most $|B|$ leaves. Since T_B is binary it has at most $|B| \cdot r + 1$ nodes. Hence, all trees T_B, $B \in X$, have $\mathcal{O}(n \cdot m)$ nodes. Ranging over all v, the sets $C(v)$ can be represented in space $\mathcal{O}(n)$. Hence we get the time complexity $\mathcal{O}(n \cdot m)$ for computing the trees T_B, $B \in X$ and the space complexity $\mathcal{O}(n \cdot m)$ for their representation. □

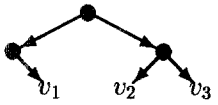
Algorithm 3. *for testing probabilistic bisimulation*
Input: *a transition system* $(S, \to)$
Output: *the set* $R = S/\sim$ *of bisimulation equivalence classes*
Method: *Let* $X := \{S\}$.
Repeat
$Y := X$; $X := \mathcal{J}(X)$;
until $Y = X$;
Return $R := X$.

It is clear that the algorithm returns a partition R with $\mathcal{J}(R) = R$. By Lemma 9: R is the set of bisimulation equivalence classes. If the loop is performed n times then X consists of n one-element sets and hence $\mathcal{J}(X) = X$. Hence the loop is performed at most n times. By Lemma 10 the time complexity is $\mathcal{O}(n^2 \cdot m)$, the space complexity $\mathcal{O}(n \cdot m)$.

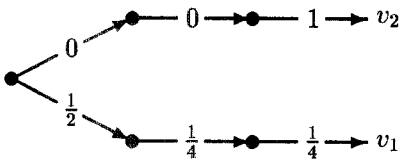
Example 11. Let $(S, \to)$ be given by: $S = \{s_1, s_2, s, t, u\}$ and

$$s_1 \xrightarrow{\alpha} \mu,\ s_2 \xrightarrow{\alpha} \mu,\ s_1 \xrightarrow{\alpha} \mu_1,\ s_2 \xrightarrow{\alpha} \mu_2,\ s \xrightarrow{\alpha} \mu,\ t \xrightarrow{\beta} \mu$$

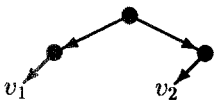
where $\mu(u) = 1$, $\mu_1(s_1) = \mu_1(s_2) = \mu_1(t) = \mu_1(u) = 1/4$ and $\mu_2(s_1) = 1/2$, $\mu_2(t) = \mu_2(u) = 1/4$. Initially we deal with the partition $\{S\}$ and compute $\mathcal{J}(\{S\})$ with the help of Lemma 10: The trees $T_{S,\alpha}$ and $T_{S,\beta}$ consist of a single edge labelled by 1. Their leaves $v_{S,\alpha}$ and $v_{S,\beta}$ are labelled by $(1, \{s_1, s_2, s\})$ and $(1, \{T\})$ respectively. This yields $\mathcal{L}_S = \{(\alpha, \{s_1, s_2, s\}), (\beta, \{t\})\}$ and the tree T_S



where $C(v_1) = \{s_1, s_2, s\}$, $C(v_2) = \{t\}$ and $C(v_3) = \{u\}$. Hence $\mathcal{J}(\{S\}) = \{B_1, B_2, B_3\}$ where $B_i = C(v_i)$. Next we compute $\mathcal{J}(\{B_1, B_2, B_3\})$. Since B_2 and B_3 consist of a single element we only have to consider B_1. The tree $T_{B_1,\alpha}$ can be depict as follows:



where $L(v_1) = \{s_1, s_2\}$ and $L(v_2) = \{s_1, s_2, s\}$. This yields the tree T_{B_1}:



where $C(v_1) = \{s_1, s_2\}$, $C(v_2) = \{s\}$. We obtain the partition X which consists of $\{s_1, s_2\}$, $\{s\}$, $\{t\}$ and $\{u\}$. The next step yields $\mathcal{J}(X) = X$ and hence $X = S/\sim$. □

5 Concluding remarks

We gave an algorithm for testing probabilistic bisimulation in time $\mathcal{O}(n^2 \cdot m)$. Compared with the non-probabilistic case where the best known algorithm for deciding bisimilarity has the time complexity $\mathcal{O}(m \cdot \log n)$ [22] the cost of our algorithm seem to be acceptable. It is an open problem whether the time complexity of our algorithm can be improved in a similar way as the $\mathcal{O}(m \cdot \log n)$ algorithm of [22] improves the $\mathcal{O}(n \cdot m)$ algorithm of [17]. The algorithm which is implemented in the Concurrency Workbench [4] tests non-probabilistic simulation in time $\mathcal{O}(n^4 \cdot m)$. It works similar to the bisimulation equivalence algorithm of [17]. It is an open question whether our $\mathcal{O}(n^5 \cdot m^2)$ result can be improved by a partioning technique. Our methods applied to "deterministic" probabilistic transition systems yield time complexity $\mathcal{O}(n^7)$ for deciding simulation and time complexity $\mathcal{O}(n^3)$ for deciding bisimulation. (In "deterministic" transition systems, for every state s and action α there is at most one outgoing transition labelled by α. Hence, for fixed action set, the total number m of transitions is $\mathcal{O}(n)$.)

In this paper we only considered strong (bi-)simulation which does not abstract from internal actions. It would be interesting if the algorithms presented here can be modified to check weak (bi-)simulation.

References

1. R. Alur, C. Courcoubetis, D. Dill: Verifying Automata Specifications of Probabilistic Real-Time Systems, Proc. REX Workshop, Mook, The Netherlands, Real-Time: Theory in Practice, J.W. de Bakker, C. Huizing, W.P. de Roever, C. Rozenberg (eds.), Lecture Notes in Computer Science 600, pp 27-44, 1991.
2. C. Baier, M. Kwiatkowska: Domain Equations for Probabilistic Processes, submitted for publication.
3. T. Bolognesi, S. Smolka: Fundamental Results for the Verification of Observational Equivalence: a Survey, Protocol Specification, Testing and Verification, Elsevier Science Publishers, IFIP, pp 165-179, 1987.
4. R. Cleaveland, J. Parrow, B. Steffen: A Semantics-Based Verification Tool for Finite State Systems, Protocol Specification, Testing and Verification IX, Elsevier Science Publishers, IFIP, pp 287-302, 1990.
5. R. Cleaveland, S. Smolka, A. Zwarico: Testing Preorders for Probabilistic Processes, Proc. ICALP 1992, Lecture Notes in Computer Science 623, Springer-Verlag, pp 708-719, 1992.
6. C. Courcoubetis, M. Yannakakis: Verifying Temporal Properties of Finite-State Probabilistic Programs, Proc. 29th Annual Symp. on Foundations of Computer Science, pp 338-345, 1988.
7. E. Dinic: Algorithm for Solution of a Problem of Maximal Flow in a Network with Power Estimation, Soviet. Math. Dokl., Vol. 11, pp 1277-1280, 1970.
8. R. Enders, T. Filkorn, D. Taubner: Generating BDDs fir Symbolic Model checking in CCS, Distributed Computing, Vol. 6, pp 155-164, 1993.
9. S. Even: Graph Algorithms, Computer Science Press, 1979.
10. L. Ford, D. Fulkerson: Flows in Networks, Princeton University Press, 1962.

11. J. Groote, F. Vaandrager: An Efficient Algorithm for Branching Bisimulation and Stuttering Equivalence, Proc. 17th International Colloqium Warwick, Automata, Languages and Programming, Lecture Notes in Computer Science 443, pp 626-638, 1990.
12. H. Hansson: Time and Probability in Formal Design of Distributed Systems, Ph.D.Thesis, Uppsala University, 1994.
13. H. Hansson, B. Jonsson: A Logic for Reasoning about Time and Probability, Formal Aspects of Computing, Vol. 6, pp 512-535, 1994.
14. S. Hart, M. Sharir: Probabilistic temporal logic for finite and bounded models, Proc. 16th ACM Symposium on Theory of Computing, pp 1-13, 1984.
15. B. Jonsson, K.G. Larsen: Specification and Refinement of Probabilistic Processes, Proc. 6th IEEE Symp. on Logic in Computer Science, 1991.
16. B. Jonsson, W. Yi: Compositional Testing Preorders for Probabilistic Processes, Proc. 10th IEEE Symp. on Logic in Computer Science, pp 431-443, 1995.
17. P. Kannelakis, S. Smolka: CCS Expressions, Finite State Processes and Three Problems of Equivalence, Proc. 2nd ACM Symposium on the Pronciples of Distributed Computing, pp 228-240, 1983.
18. K. Larsen, A. Skou: Bisimulation through Probabilistic Testing, Information and Computation, Vol. 94, pp 1-28, 1991.
19. D. Lehmann, S. Shelah: Reasoning with Time and Chance, Information and Control, Vol. 53, pp 165-198, 1982.
20. V. Malhotra, M. Pramodh Kumar, S. Maheshwari: An $\mathcal{O}(|V^3|)$ Algorithm for Finding Maximum Flows in Networks, Computer Science Program, Indian Institute of Technology, Kanpur 208016, 1978.
21. R. Milner: Communication and Concurrency, Prentice Hall, 1989.
22. R. Paige, R. Tarjan: Three Partition Refinement Algorithms, SIAM Journal of Computing, Vol. 16, No. 6, pp 973-989, 1987.
23. A. Pnueli, L. Zuck: Verification of Multiprocess Probabilistic Protocols, Distributed Computing, Vol. 1, No. 1, pp 53-72, 1986.
24. A. Pnueli, L. Zuck: Probabilistic Verification, Information and Computation, Vol. 103, pp 1-29, 1993.
25. R. Segala, N. Lynch: Probabilistic Simulations for Probabilistic Processes, Proc. CONCUR 94, Theories of Concurrency: Unification and Extension, Lecture Notes in Computer Science 836, Springer-Verlag, pp 492-493, 1994.
26. R. van Glabbeek, S. Smolka, B. Steffen, C. Tofts: Reactive, Generative, and Stratified Models for Probabilistic Processes, Proc. 5th IEEE Symposium on Logic in Computer Science, pp 130-141, 1990.
27. M. Vardi: Automatic Verification of Probabilistic Concurrent Finite-State Programs, Proc. 26th Symp. on Foundations of Computer Science, pp 327-338, 1985.
28. S. Yuen, R. Cleaveland, Z. Dayar, S. Smolka: Fully Abstract Characterizations of Testing Preorders for Probabilistic Processes, Probabilistic Simulations for Probabilistic Processes, Proc. CONCUR 94, Theories of Concurrency: Unification and Extension, Lecture Notes in Computer Science 836, Springer-Verlag, pp 497-512, 1994.
29. W. Yi: Algebraic Reasoning for Real-Time Probabilistic Processes with Uncertain Information, Formal Techniques in Real Time and Fault Tolerant Systems, Lecture Notes in Computer Science 863, Springer-Verlag, pp 680-693, 1994.
30. W. Yi, K. Larsen: Testing Probabilistic and Nondeterminsitic Processes, Protocol, Specification, Testing and Verification XII, Elsevier Science Publishers, IFIP, pp 47-61, 1992.