Decision Algorithms for Probabilistic Bisimulation*

Stefano Cattani[1] and Roberto Segala[2]

[1] School of Computer Science, The University of Birmingham
Birmingham B15 2TT, United Kingdom
[2] Dipartimento di Informatica, Università di Verona
Strada Le Grazie 15, Ca' Vignal 2 37134 Verona, Italy

Abstract. We propose decision algorithms for bisimulation relations defined on probabilistic automata, a model for concurrent nondeterministic systems with randomization. The algorithms decide both strong and weak bisimulation relations based on deterministic as well as randomized schedulers. These algorithms extend and complete other known algorithms for simpler relations and models. The algorithm we present for strong probabilistic bisimulation has polynomial time complexity, while the algorithm for weak probabilistic bisimulation is exponential; however we argue that the latter is feasible in practice.

1 Introduction

Randomization is attracting increasing attention in computer science, and consequently the study of modeling and verification techniques for randomized concurrent systems becomes fundamental. An evidence of this fact is the existence of a considerable literature about models for concurrent probabilistic systems and related techniques [1, 7, 9, 11, 16, 18, 24, 25]. This paper focuses on the model of probabilistic automata [21], an extension of labeled transition systems with probabilities, and on verification techniques based on probabilistic extensions of bisimulation relations [17] as defined in [22].

Probabilistic automata [21] extend labeled transition systems by generalizing the notion of a transition, which leads to a probability distribution over states rather than to a single state. Then, several notions and techniques for labeled transition systems can be extended directly to probabilistic automata. Among these, bisimulation relations [22, 8, 16] are the focus of this paper. Bisimulation was first defined in the context of CCS [17], and turned out to be a fundamental relation for its simplicity and the elegance of its axiomatization. It was first extended to a model with randomization in [16] and then extended to a model with nondeterminism and randomization in [8]. The model of [8] is also known as the alternating model of concurrent probabilistic systems as opposed to the non-alternating model given by probabilistic automata. The main restriction in the alternating model is that each state either enables several transitions that

* Supported in part by EPSRC grants GR/N22960 and GR/M13046

lead to a single state (nondeterministic state), or a single unlabeled transition that leads to a probability distribution over states (probabilistic state). Probabilistic automata allow more general notions of bisimulation, which turn out to be different compared to the alternating model as soon as we use randomization to resolve nondeterminism. Such new bisimulation relations, both in their strong and weak versions, are studied in [22]. Bisimulation relations are also studied in the context of stochastic process algebras [12, 5, 10], where it is shown that bisimulation coincides with the notion of lumping for Markov chains [12].

A problem that is subject of considerable study is the search for decision algorithms for bisimulation [3, 20, 4]. The problem is solved already in the context of stochastic process algebras [3], in the context of the alternating model [20] and in the context of probabilistic automata with respect of strong bisimulation [2]. However, it is still open in the context of probabilistic automata, since the restrictions imposed to the alternating model seem to guarantee a lot of extra structure that simplifies decision procedures; in the non alternating model bisimulation gives rise to two different relations under deterministic and randomized schedulers respectively, while in the alternating model bisimulation gives rise to the same relation no matter whether randomization is used or not.

In this paper we give decision algorithms for strong bisimulation under randomized schedulers (strong probabilistic bisimulation) and for weak bisimulation under randomized schedulers (weak probabilistic bisimulation). The algorithms are instances of the well established partitioning technique [13, 19], where large equivalence classes are refined into smaller classes; thus, our presentation concentrates on how to define a splitter for each of the relations that we analyze. The splitter for strong probabilistic bisimulation considers two states s_1 and s_2 as equivalent if and only if for each transition that leaves s_1 there is a matching convex combination of transitions that leave from s_2 and vice versa, where the convex combination of transitions expresses the ability of the scheduler to use randomization. Operationally, if we take the extreme points of the convex hull generated by the transitions that are labeled by a specific action, then the extreme points of the convex hulls for s_1 and s_2 must coincide. The splitter for weak probabilistic bisimulation is based again on convex hulls; however, we have to consider arbitrary sequences of transitions that are labeled by internal actions. Since such transitions are potentially infinite, we identify a sufficiently powerful finite class of schedulers that can characterize the extremal points of the convex hulls to be examined.

The algorithms that we propose for the probabilistic relations are based on the problem of computing the extreme points of the convex hull generated by a set of n points in a d-dimensional space; the extreme points can be computed by solving $O(n)$ linear programming problems. Unfortunately, in the case of weak probabilistic bisimulation, the number of points generating the convex hull can be exponential in the number of states of the automaton. Despite the negative result concerning the complexity of our decision algorithms, we show that from a practical perspective our algorithms are feasible since the worst case scenarios are very unlikely to occur.

The rest of the paper is organized as follows. Section 2 contains some background on measure theory and convex hulls and describes our notational conventions; Section 3 introduces the model of probabilistic automata, and Section 4 describes the bisimulation relations of [22]; Section 5 describes the general structure of the decision algorithms, while Sections 6 and 7 describe the details of the algorithms for strong probabilistic bisimulation, and weak probabilistic bisimulation, respectively; finally, Section 8 contains some concluding remarks.

2 Preliminaries

Probability Spaces and Measures. A *σ-field* over a set X is a set $\mathcal{F} \subseteq 2^X$ that includes X and is closed under complement and countable union. Observe that 2^X is a σ-field over X. A *measurable space* is a pair $(X, \mathcal{F})$ where X is a set and $\mathcal{F}$ is a σ-field over X. The set X is also called the *sample space*. A measurable space $(X, \mathcal{F})$ is called *discrete* if $\mathcal{F} = 2^X$. Given a measurable space $(X, \mathcal{F})$, a *measure* over $(X, \mathcal{F})$ is a function $\mu : \mathcal{F} \to \mathbb{R}^{\geq 0}$ such that, for each countable collection $\{X_i\}_{i \in I}$ of pairwise disjoint elements of $\mathcal{F}$, $\mu[\cup_I X_i] = \sum_I \mu[X_i]$. A *probability measure* over a measurable space $(X, \mathcal{F})$ is a measure μ over $(X, \mathcal{F})$ such that $\mu[X] = 1$; a *sub-probability measure* over $(X, \mathcal{F})$ is a measure μ over $\mathcal{F}$ such that $\mu[X] \leq 1$. A measure over a discrete measurable space is called a *discrete measure*. Sometimes we refer to probability measures as *distributions*.

Given a set X, denote by $\mathit{Disc}(X)$ the set of discrete probability measures over the measurable space $(X, 2^X)$, and by $\mathit{SubDisc}(X)$ the set of discrete sub-probability measures over the measurable space $(X, 2^X)$. We call a discrete (sub-)probability measure a *Dirac* measure if either it assigns measure 1 to exactly one object or it assigns measure 0 to all objects. Given a discrete sub-probability measure μ of $\mathit{SubDisc}(X)$, denote by $\mu[\perp]$ the value $1 - \mu[X]$. In the sequel discrete sub-probability measures are used to describe progress. If the measure of the sample space is not 1, then it means that with some non-zero probability the system does not progress. We use the symbol $\perp$ to denote this fact.

In this paper we refer also to semi-Markov processes. A semi-Markov process is a pair (Q, μ) where Q is a set and $\mu : Q \times Q \to [0, 1]$ is a function, called a *transition function*, such that for each element $q \in Q$ the function is the probability of reaching an element from X when leaving from q.

Convex Sets and Convex Hulls. A subset S of $\mathbb{R}^d$ is *convex* if for each pair of points $s_1, s_2 \in S$, the segment joining s_1 and s_2, i.e., the set of points $T = \{ts_1 + (1-t)s_2 \mid 0 \leq t \leq 1\}$, is entirely contained in S. Given a subset S of $\mathbb{R}^d$, the *convex hull* of S, denoted by $\mathit{CHull}(S)$, is the smallest convex set that contains S. A convex set S is said to be *finitely generated* if there is a finite set T, called a generator, such that $S = \mathit{CHull}(T)$. In such case there is also a unique minimum set T that generates S. We denote such minimum set T by $\mathit{Gen}(S)$. The elements of $\mathit{Gen}(S)$ are called the *extreme points* of S.

Given a set of points $S = \{s_1, \ldots, s_n\}$, we say that s is a *convex combination* of the points in S if there exists a set of non negative weights $t_1, \ldots, t_n$ such that

$\sum_{i=1}^{n} t_i = 1$ and $s = \sum_{i=1}^{n} t_i * s_i$. The definition is extended in the natural way to give the convex combination of an infinite countable set.

In the sequel, we rely on the fact that there are algorithms to determine the extreme points of the convex hull generated by a given set of n points. This is also known as the "redundancy removal for a point set S" in $\mathbb{R}^d$. This problem has a polynomial complexity as it can be reduced to solving $O(n)$ linear programming problems, for which many polynomial time algorithms are available [6].

3 Probabilistic Automata

Probabilistic Automata. A *probabilistic automaton* $\mathcal{A}$ is a tuple $(S, \bar{s}, \Sigma, \mathcal{D})$ where S is a set of states, $\bar{s} \in S$ is the *start state*, Σ is a set of *actions* and $\mathcal{D} \subseteq S \times \Sigma \times Disc(S)$ is the *transition relation*. The set of actions Σ is partitioned into two sets E and H of *external* and *hidden* actions, respectively. For the purpose of this paper, we will consider only automata with finite state sets.

For convenience, states are ranged over by r, q, s, actions are ranged over by a, b, c, internal actions are ranged over by τ, and discrete distributions are ranged over by μ. We also denote the generic elements of a probabilistic automaton $\mathcal{A}$ by $S, \bar{s}, \Sigma, E, H$ and $\mathcal{D}$, and we propagate primes and indices when necessary. Thus, the elements of a probabilistic automaton $\mathcal{A}'_i$ are $S'_i, \bar{s}'_i, \Sigma'_i, E'_i, H'_i$ and $\mathcal{D}'_i$.

An element of $\mathcal{D}$ is called a *transition*. A transition (s, a, μ) is said to leave from state s, to be labeled by a, and to lead to μ. We also say that s enables action a and that action a is enabled from s. Finally, we say that the transition (s, a, μ) is enabled from s. The transition (s, a, μ) is denoted alternatively by $s \xrightarrow{a} \mu$. Given a state q, denote by $\mathcal{D}(q)$ the set of transitions that leave from q, that is, $\mathcal{D}(q) = \{(s, a, \mu) \in \mathcal{D} \mid s = q\}$.

The reactive models of [7] are probabilistic automata where each state enables at most one transition for each action. In line with the process algebraic terminology, we call such probabilistic automata *deterministic*. Ordinary automata are probabilistic automata where each transition leads to a Dirac distribution; we call them *Dirac* automata. The alternating model [9] can be seen as probabilistic automata where each state either enables a unique transition labeled by an internal action or several transitions that lead to Dirac distributions.

Executions and Traces. A *potential execution* of a probabilistic automaton $\mathcal{A}$ is a finite or infinite sequence of alternating states and actions, $\alpha = s_0 a_1 s_1 a_2 s_2 \ldots$ starting from a state and, if the sequence is finite, ending with a state. Define the *length* of a potential execution α, denoted by $|\alpha|$, to be the number of occurrences of actions in α. If α is an infinite sequence, then $|\alpha| = \infty$. For a natural number $i \leq |\alpha|$, denote by $\alpha[i]$ the state s_i. In particular $\alpha[0]$ is the first state of α. For a finite execution α denote by $\alpha[\perp]$ the last state of α. We say that a potential execution α is a *prefix* of a potential execution α', denoted by $\alpha \leq \alpha'$ if the sequence α is a prefix of the sequence α'. An *execution* of a probabilistic automaton $\mathcal{A}$ is a potential execution of $\mathcal{A}$, $\alpha = s_0 a_1 s_2 a_2 s_2 \ldots$ such that for each $i < |\alpha|$ there exists a transition (s_{i-1}, a_i, μ_i) of $\mathcal{D}$ where $\mu_i[s_i] > 0$. An execution is said to be *initial* if $\alpha[0] = \bar{s}$. We denote by $execs(\mathcal{A})$ the set of executions of $\mathcal{A}$, and by $execs^*(\mathcal{A})$ the set of finite executions of $\mathcal{A}$.

The *trace* of a potential execution α, denoted by $trace(\alpha)$ is the sub-sequence of α composed by its external actions. A trace is used to observe the externally visible communication that takes place within a potential execution.

Combined Transitions. Nondeterminism in a probabilistic automaton arises from the fact that a state may enable several transitions. Resolving the nondeterminism in a state s means choosing one of the transitions that are enabled from s. However, being in a randomized environment, transitions may be chosen randomly. In this case we obtain an object which we call a combined transition.

Given a state s and a distribution $\sigma \in SubDisc(\mathcal{D}(s))$, define the *combined transition* according to σ to be the pair (s, μ_σ), where $\mu_\sigma \in SubDisc(\Sigma \times S)$ is defined for each pair (a, q) as

$$\mu_\sigma[(a,q)] = \sum_{(s,a,\mu)} \sigma\,[(s,a,\mu)]\ \mu[q]. \tag{1}$$

In practice a transition is chosen among the transitions enabled from s according to σ, and then the chosen transition is scheduled. The chosen transitions may be labeled by different actions, and thus the distribution that appears in a combined transition includes actions as well. Observe that, since σ is a sub-probability measure, we consider also the fact that with some non-zero probability no transition is scheduled from s. We denote a combined transition (s, μ) alternatively by $s \longrightarrow_{\mathrm{C}} \mu$. Whenever there exists an action a such that $\mu[(a, S)] = 1$ we denote the corresponding combined transition alternatively by $s \xrightarrow{a}_{\mathrm{C}} \mu'$ where $\mu' \in Disc(S)$ is defined for each state q as $\mu'[q] = \mu[(a, q)]$.

Schedulers. A *scheduler* for a probabilistic automaton $\mathcal{A}$ is a function $\sigma : execs^*(\mathcal{A}) \to SubDisc(\mathcal{D})$ such that for each finite execution α, we have that $\sigma(\alpha) \in SubDisc(\mathcal{D}(\alpha[\bot]))$. In other words, a scheduler is the entity that resolves nondeterminism in a probabilistic automaton by choosing randomly either to stop or to perform one of the transitions that are enabled from the current state. The choices of a scheduler are based on the past history.

We say that a scheduler is *Dirac* if it assigns a Dirac distribution to each execution. We say that a scheduler is *simple* if for each pair of finite executions $\alpha_1, \alpha_2 \in execs(\mathcal{A})$ such that $\alpha_1[\bot] = \alpha_2[\bot]$ we have that $\sigma(\alpha_1) = \sigma(\alpha_2)$. Finally, we say that a scheduler is *determinate* is for each pair of finite executions $\alpha_1, \alpha_2 \in execs(\mathcal{A})$ such that $\alpha_1[\bot] = \alpha_2[\bot]$ and $trace(\alpha_1) = trace(\alpha_2)$ we have that $\sigma(\alpha_1) = \sigma(\alpha_2)$.

A Dirac scheduler is a scheduler that does not use randomization in its choices. We could call it a deterministic scheduler; however the name Dirac is more consistent with the rest of our terminology and avoids overloading of the term nondeterministic. Simple schedulers base their choices only on the current state without looking at the past history, while determinate scheduler may look at the externally visible part of the past history. Determinate scheduler were introduced in [20] for the purpose of defining a finite class of schedulers that would characterize probabilistic bisimulation in the alternate model. In this paper we use determinate schedulers for the same purpose as in [20].

Consider a scheduler σ and a finite execution α with last state q. The distribution $\sigma(\alpha)$ describes how to move from q. The resulting combined transition is the combined transition according to the distribution $\sigma(\alpha)$, which we denote by $(q, \mu_{\sigma(\alpha)})$.

Probabilistic Executions. The result of the action of a scheduler from a start state s can be represented as a semi-Markov process whose states are finite executions of $\mathcal{A}$ with start state s. These objects are called *probabilistic executions*.

Formally, the probabilistic execution of $\mathcal{A}$ identified by a scheduler σ and a state s is a semi-Markov process (Q, μ) where Q is the set of finite executions of $\mathcal{A}$ that start with s, and μ is 0 except for pairs of the form (q, qar), where it is defined as follows: $\mu[q, qar] = \mu_{\sigma(\alpha)}[(a, r)]$.

Given a probabilistic execution we can define a probability measure over executions as follows. The sample space is the set of executions that start with s; the σ-field is the σ-field generated by the set of *cones*, sets of the form $C_\alpha = \{\alpha' \in \mathit{execs}(\mathcal{A}) \mid \alpha \leq \alpha'\}$; the measure is the unique extension $\mu_{\sigma,s}$ of the measure defined over the cones by the following equation: $\mu_{\sigma,s}[C_{sa_1s_1\ldots a_ns_n}] = \prod_{i\in\{1,\ldots n\}} \mu_{\sigma(sa_1s_1\ldots a_{i-1}s_{i-1})}[(a_i, s_i)]$. This definition is justified by standard measure theoretical arguments [15]. In the sequel we denote a probabilistic execution alternatively by the probability measure over executions that it identifies.

Observe that any reachability property is measurable since it can be expressed as a countable union of cones. Furthermore, observe that the probability of a finite execution α is $\mu_{\sigma,s}[\alpha] = \mu_{\sigma,s}[C_\alpha]\mu_{\sigma(\alpha)}[\perp]$. The probability of α represents the probability of executing α and then stopping. The probability of the set of finite executions represents the probability of stopping eventually.

Weak Transitions. We are often interested in transitions that abstract from internal computation. Weak transitions as defined in [17] serve this purpose. In [23] weak transitions are generalized to probabilistic automata by stating that a weak transition is essentially a probabilistic execution that stops with probability 1 and whose traces at the stopping points consist of a unique action.

Formally, we say that there is a *weak transition* from state s to μ labeled by an action a, denoted by $s \stackrel{a}{\Longrightarrow} \mu$, if there is a probabilistic execution $\mu_{\sigma,s}$, with σ a Dirac scheduler, such that

1. $\mu_{\sigma,s}[\mathit{execs}^*(\mathcal{A})] = 1$, and
2. for each $\alpha \in \mathit{execs}^*(\mathcal{A})$, if $\mu_{\sigma,s}[\alpha] > 0$ then $\mathit{trace}(\alpha) = a$.
3. for each state q, $\mu_{\sigma,s}[\{\alpha \in \mathit{execs}^*(\mathcal{A}) \mid \alpha[\perp] = q\}] = \mu[q]$.

If we remove the Dirac condition on the scheduler σ, then we say that there is a *combined weak transition* from s to μ labeled by a, denoted by $s \stackrel{a}{\Longrightarrow}_{\mathrm{C}} \mu$.

Condition 1 states that the probability of stopping is 1; Condition 2 states that at each stopping point only action a has occurred among the external actions; Condition 3 states that the distribution over states at the stopping points is μ.

4 Bisimulation

We can now define the bisimulation relations that we are interested to study following the approach of [23]. We distinguish between strong and weak bisimulations, and within each class we distinguish between bisimulations based on Dirac schedulers (strong bisimulation and weak bisimulation relations) and bisimulations based on general schedulers (strong probabilistic bisimulation and weak probabilistic bisimulation relations).

We first need to lift an equivalence relation on states to an equivalence relation on distributions over states. Following [16], two distributions are equivalent if they assign the same probabilities to the same equivalence classes. Formally, given an equivalence relation $\mathcal{R}$ on a set of states Q, we say that two probability distributions μ_1 and μ_2 of $Disc(Q)$ are equivalent according to $\mathcal{R}$, denoted by $\mu_1 \equiv_{\mathcal{R}} \mu_2$, iff for each equivalence class $\mathcal{C}$ of $\mathcal{R}$, $\mu_1[\mathcal{C}] = \mu_2[\mathcal{C}]$.

Strong Probabilistic Bisimulation. Let $\mathcal{A}_1, \mathcal{A}_2$ be two probabilistic automata. An equivalence relation $\mathcal{R}$ on $S_1 \cup S_2$ is a *strong probabilistic bisimulation* if, for each pair of states $q, r \in S_1 \cup S_2$ such that $q \mathcal{R} r$, if $q \xrightarrow{a} \mu$ for some distribution μ, then there exists a distribution μ' such that $\mu \equiv_{\mathcal{R}} \mu'$ and $r \xrightarrow{a}_{\mathrm{C}} \mu'$.

The probabilistic automata $\mathcal{A}_1, \mathcal{A}_2$ are said to be strongly probabilistically bisimilar if there exists a strong probabilistic bisimulation $\mathcal{R}$ on $S_1 \cup S_2$ such that $\bar{s}_1 \mathcal{R} \bar{s}_2$. We denote a strong probabilistic bisimulation relation also by $\sim_c$.

The non probabilistic version of strong bisimulation [16, 8, 22] is obtained by disallowing combined transitions in the definition above. Strong probabilistic bisimulation was first defined in [22], but it also gives rise to meaningful relations for reactive systems and for the alternating model. However, in the two restrictive models strong bisimulation and strong probabilistic bisimulation coincide.

Weak Probabilistic Bisimulation. Let $\mathcal{A}_1, \mathcal{A}_2$ be two probabilistic automata. An equivalence relation $\mathcal{R}$ on $S_1 \cup S_2$ is a *weak probabilistic bisimulation* if, for each pair of states $q, r \in S_1 \cup S_2$ such that $q \mathcal{R} r$, if $q \xrightarrow{a} \mu$ for some distribution μ, then there exists a distribution μ' such that $\mu \equiv_{\mathcal{R}} \mu'$ and $r \overset{a}{\Longrightarrow}_{\mathrm{C}} \mu'$.

The probabilistic automata $\mathcal{A}_1, \mathcal{A}_2$ are said to be weakly probabilistically bisimilar if there exists a weak probabilistic bisimulation $\mathcal{R}$ on $S_1 \cup S_2$ such that $\bar{s}_1 \mathcal{R} \bar{s}_2$. We denote a weak probabilistic bisimulation relation also by $\approx_c$.

The non probabilistic version of weak bisimulation [22] is obtained by disallowing combined transitions in the definition above. Weak probabilistic bisimulation was first defined in [22]. Decision algorithms for weak probabilistic bisimulation in the alternating model are studied in [20]. Similarly to the strong case, there is a very close relationship between weak bisimulation and weak probabilistic bisimulation in the alternating model.

5 The Algorithms

In this section we define the general scheme of the algorithms that decide whether two probabilistic automata are bisimilar according to one of the definitions of

Section 4. The approach we use in all cases is the partitioning technique of [13, 19]: we start with a single equivalence class containing all the states and we refine it until we get the equivalence classes induced by the bisimulation relation under examination.

To refine a partition, we find an element (*splitter*) of the partition that violates the definition of bisimulation and then subdivide it further. Formally, given a probabilistic automaton $\mathcal{A} = (S, \bar{s}, \Sigma, \mathcal{D})$ and a partition $\mathcal{W}$ of S, we say that a triplet $(\mathcal{C}, a, \mathcal{W})$, $\mathcal{C} \in \mathcal{W}$ and $a \in \Sigma$, is a splitter if there are two states $s, t \in \mathcal{C}$ that prevent the partition from being a bisimulation.

Algorithm 1 describes the main structure of the decision procedure DecideBisim. It is based on two functions: function FindSplit returns a splitter $(\mathcal{C}, a, \mathcal{W})$ for the current partition $\mathcal{W}$ if one exists, and returns the empty set otherwise; function Refine, given a splitter $(\mathcal{C}, a, \mathcal{W})$, distinguishes the states in $\mathcal{C}$ that are incompatible, and divides $\mathcal{C}$ into subclasses, thus refining the current partition.

Algorithm 1. *Decide whether two probabilistic automata $\mathcal{A}_1 = (S_1, \bar{s}_1, \Sigma_1, \mathcal{D}_1)$ and $\mathcal{A}_2 = (S_2, \bar{s}_2, \Sigma_2, \mathcal{D}_2)$ are related according to some bisimulation $\mathcal{R}$.*

DecideBisim $(\mathcal{A}_1, \mathcal{A}_2) =$
  $\mathcal{W} = \{S_1 \cup S_2\}$;
  $(\mathcal{C}, a, \mathcal{W}) =$ FindSplit$(\mathcal{W})$;
  while $\mathcal{C} \neq \emptyset$ do
    $\mathcal{W} =$ Refine$(\mathcal{W}, \mathcal{C}, a)$;
    $(\mathcal{C}, a, \mathcal{W}) =$ FindSplit$(\mathcal{W})$;
  if ($\bar{s}_1$ *and* $\bar{s}_2$ *are related*) then
    return $\mathcal{W}$;
  else return false;

Algorithm 1 works as follows: it starts with one single equivalence class and then refines it through the while loop. It is easy to see that in each iteration, if a splitter is found, then the refinement produces a partition that is finer than the previous one. When no splitter is found, then DecideBisim simply checks whether the start states of $\mathcal{A}_1$ and $\mathcal{A}_2$ are equivalent according to the current partition.

Function FindSplit (see Algorithm 2) has a two phase structure which is independent of the relation we study. In the first phase, FindSplit computes information about the transitions enabled from each state s; then, in the second phase, it compares the information of each pair of states in the same equivalence class to check whether they should be separated.

Algorithm 2. *Finds a splitter in partition $\mathcal{W}$.*

FindSplit$(\mathcal{W}) =$
  foreach $s \in S_a \cup S_b,\ a \in \Sigma$
    $I(s, a) =$ ComputeInfo$(s, a, \mathcal{W})$
  foreach $\mathcal{C}_i \in \mathcal{W},\ s, t \in \mathcal{C}_i,\ a \in \Sigma$
    if $I(s, a)/\mathcal{W} \neq I(t, a)/\mathcal{W}$
      return $(\mathcal{C}_i, a, \mathcal{W})$;
  return $\emptyset$;

Finally, function Refine has a very simple structure: it refines each partition by comparing states pairwise and grouping those with the compatible information into new equivalence classes. We omit the pseudo code from this paper.

In the following sections we show how to compute the information $I(s, a)$ and how to compare such information for strong probabilistic bisimulation and weak probabilistic bisimulation. The technique can also be used for strong non probabilistic bisimulation as $I(s, a)$ is described by the finite set of transitions leaving state s and labeled by a (the resulting algorithm is proposed in [2]).

For strong (resp, weak) probabilistic bisimulation $I(s, a)$ describes the set of combined transitions (resp, weak combined transitions) that are enabled from s and labeled by a. In both cases, each transition can be described simply by the distribution over states that it leads to, which can be seen as well as a tuple in a n-dimensional space, where n is the number of states. Formally, if we express S as $\{s_1, s_2, \ldots, s_n\}$, thus numbering the states, a transition (s, a, μ) of $I(s, a)$ can be represented by the point $(p_1, p_2, \ldots, p_n)$ in $\mathbb{R}^n$ where $p_i = \mu(s_i)$ for each i in $\{1, \ldots, n\}$.

Define the *reachable space* for a point s with transitions labeled by a with respect to strong probabilistic bisimulation, denoted by $S_{\sim_c}(s, a)$, to be the set of points that are reachable from s with a combined transition $s \xrightarrow{a}_{\mathrm{C}} \mu$. Similarly, define the reachable space with respect to weak probabilistic bisimulation, denoted $S_{\approx_c}(s, a)$ by considering combined weak combined transitions.

Theorem 1. *Let $\simeq$ be either strong probabilistic bisimulation or weak probabilistic bisimulation, $\mathcal{A}$ a probabilistic automaton, and s_1 and s_2 two states. Then, $s_1 \simeq s_2$ if and only if $\forall a \in \Sigma$ we have that $S_{\simeq}(s_1, a)/\simeq\, = S_{\simeq}(s_2, a)/\simeq$.*

Proof outline. The proof is straightforward in the case of strong probabilistic bisimulation, since we have to match strong transitions of one state with strong combined transitions of the other and vice versa. The result for weak probabilistic bisimulation is similar, after proving that the main condition in the definition of probabilistic weak bisimulation can be replaced by the following: for each pair of states q and r, such that $q \approx_c r$ if $q \stackrel{a}{\Longrightarrow}_c \mu$ for some distribution μ, then there exists a distribution μ' such that $\mu \equiv_{\approx_c} \mu'$ and $r \stackrel{a}{\Longrightarrow}_c \mu'$. ∎

It follows that the information $I(s, a)$ that we need in FindSplit coincides with the reachable spaces defined above. Such spaces are then quotiented with respect to the equivalence relation induced by the current partition $\mathcal{W}$. In the following sections we show that for strong probabilistic bisimulation and weak probabilistic bisimulation these spaces are convex sets and, despite being potentially infinite because of combined transitions, they are generated by a finite set of points.

Complexity. Assuming that the complexity to compute the information $I(s, a)$ is c and the complexity to compare such information is d, then the complexity of FindSplit is $O(n * c + n^2 * d)$, the complexity of Refine is $O(n^2 * d)$ and the overall complexity is $O(n * (n * c + n^2 * d))$, since the while loop can be repeated at most n times, corresponding to the worst case when all the states are in a different equivalence class. Note that our algorithm computes $I(s, a)$ at each iteration and an efficient implementation would avoid this.

6 Splitter for Strong Probabilistic Bisimulation

In the case of strong probabilistic bisimulation, $S_{\sim_c}(s, a)$ is a potentially infinite set, but it is easy to see that such set is the convex set generated by the (non combined) transitions leaving the state.

Proposition 1. *Given a probabilistic automaton $\mathcal{A}$, for each state s and each action a, the reachable space $S_{\sim_c}(s, a)$ is a convex space.*

Proof. We need to prove that, given two combined transitions $s \xrightarrow{a}_c \mu_1$ and $s \xrightarrow{a}_c \mu_2$, each of their convex combinations is still a legal combined transition. To prove this fact, observe that each transition $(s, a, (p * \mu_1 + (1-p) * \mu_2))$, with $0 \leq p \leq 1$, can be seen as the combination of the two transitions $s \xrightarrow{a}_c \mu_1$ and $s \xrightarrow{a}_c \mu_2$ with weights p and $1-p$, respectively. ■

Proposition 2. *Given a probabilistic automaton $\mathcal{A}$, each point reachable with a combined transition $s \xrightarrow{a}_c \mu$ is a convex combination of those points reachable with non combined transitions.*

Compute. From Propositions 1 and 2, the distributions that are reachable from s with non combined transitions labeled by a, are sufficient to characterize the set $S_{\sim_c}(s, a)$. Thus, we can apply any algorithms for the determination of the extreme points to obtain $Gen(S_{\sim_c}(s, a))$. This means that we have to apply linear programming algorithms to the points corresponding to each deterministic transition.

Compare. We have to verify that the generators computed are the same set.

Complexity. The algorithm proposed above is polynomial in the number of states n and the number of transitions m of the automaton. The computation of the generator of the convex set can be done by solving at most $O(m)$ linear programming problems, for which polynomial algorithms are available (e.g. [14]). The comparison of two sets of extreme points is again polynomial. In practice, we expect to have a limited number of transitions leaving each point, thus limiting the number of linear programming problems to solve for each point.

7 Splitter for Weak Probabilistic Bisimulation

Again we can prove that the set $S_{\approx_c}(s, a)$ is a convex set; however, since in a weak transition we can consider arbitrary sequences of transitions labeled by internal actions, we need to identify a finite set of points that includes the generator of $S_{\approx_c}(s, a)$. A weak combined transition can be seen as the result of the action of a scheduler. We show that the generator of $S_{\approx_c}(s, a)$ is included in the set of weak transitions generated by determinate and Dirac schedulers, which are finite. The idea of determinate schedulers originates from [20]. The proof proceeds in two steps: first we show that we can use determinate schedulers (Proposition 3); then we show that we can remove randomization from the schedulers (Proposition 4).

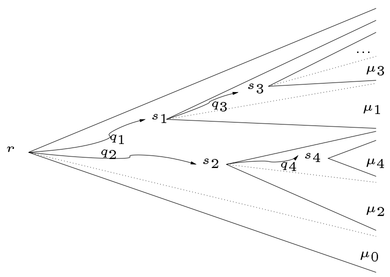

Fig. 1. Structure of the proof of Proposition 3.

Proposition 3. *Given a probabilistic automaton $\mathcal{A}$, any point reachable with a weak transition under the control of a (generic) scheduler can be obtained as a convex combination of the points reachable with determinate schedulers.*

Proof. To simplify our proof we augment our probabilistic automaton by adding some state information that records the last external action that occurred. Thus we can limit our analysis to simple schedulers.

Consider a probabilistic execution induced by a scheduler σ starting from a state r. Let s be a state of the automaton and call $s_1, s_2, \ldots$ its occurrences in the probabilistic execution (see Figure 1). There can be at most countably many occurences of s. For each $i > 0$, let π_i be the probability not to reach any other occurrence of s from s_i, and let μ_i by the final distribution reached from s_i conditioned on not reaching s any more. If $\pi_i = 0$, then define μ_i arbitrarily. Let π_0 be the probability not to reach s from r if $s \neq r$, 0 otherwise, and let μ_0 by the final distribution reached from r conditioned on not reaching s. As before, μ_0 is arbitrary if $\pi_0 = 0$. For each $i > 0$, define σ_i as a scheduler that behaves like σ whenever s does not occur in the history and that treats the last occurrence of s as the occurrence s_i otherwise. Then σ_i is a scheduler that does not look at the past history whenever s is reached. Furthermore, the weak combined transition identified by σ_i leads to distribution $\rho_i = \pi_0\mu_0 + (1 - \pi_0)\mu_i$.

Observe that the distribution μ induced by σ satisfies the equation $\mu = \sum_{i \geq 0} p_i\mu_i$, where p_i is the probability to reach the occurrence s_i from r multiplied by π_i for $i > 0$, and $p_0 = \pi_0$; note that $\sum_{i>0} p_i = 1 - \pi_0$. By defining p'_i to be $p_i/(1 - \pi_0)$, it is immediate to show that $\mu = \sum_{i>0} p'_i\rho_i$, which means that the weak combined transition associated with σ can be expressed as a convex combination of the weak combined transitions associated with each of the σ_i's.

By repeating the above procedure on the other states for each of the schedulers σ_i, we end up with a collection of simple schedulers whose corresponding transitions can be combined to give the transition of σ. The process ends since the states of $\mathcal{A}$ are finite. ∎

Proposition 4. *Given a probabilistic automaton $\mathcal{A}$, each point reachable with a weak transition under the control of a determinate scheduler can be obtained as a convex combination of the points reachable with Dirac determinate schedulers.*

Proof outline. Again we simplify the proof by augmenting our probabilistic automata so that we deal with simple schedulers. Then, we proceed state by state by replacing the combined transition returned by σ by one of the transitions that have non-zero probability. The resulting schedulers can be combined to lead to the distribution of σ. ■

Compute. To compute $S_{\approx_c}(s,a)$ we first compute the set of points reachable by transitions generated by Dirac determinate schedulers and then solve the extreme point problem in the set of reachable points as we did in Section 6 for strong probabilistic bisimulation.

To compute the set of points reachable by transitions generated by Dirac determinate schedulers we first check all the possible ways to resolve non determinism by visiting the transition graph of $\mathcal{A}$ and resolving nondeterminism in each possible way at each state that is reached in the visit. Since we are restricted to Dirac determinate schedulers, each time we encounter a state before or after the occurrence of a external action the same decision must be taken.

Let σ be a Dirac simple scheduler for $\mathcal{A}$ (again we consider augmented automata for simplicity), and let the states of $\mathcal{A}$ be $s_0, s_1, \ldots, s_n$. For each i, let μ_i be the target distribution of the transition given by σ and x_i be the distribution reachable from s_i under the control of σ. The following linear system holds, where e_i is the vector with the ith component set to 1 and all others set to 0.

$$\begin{cases} x_0 = \mu_0[s_0] \cdot x_0 + \ldots + \mu_0[s_n] \cdot x_n + \mu_0[\perp] \cdot e_0 \\ \quad \vdots \\ x_n = \mu_n[s_0] \cdot x_0 + \ldots + \mu_n[s_n] \cdot x_n + \mu_n[\perp] \cdot e_n \end{cases} \tag{2}$$

We are interested in the solution of the system above relative to state s. If such solution does not exist, then it means that scheduler σ does not describe a transition from s, and thus it should be discarded.

Compare. Once we have the set of extreme points, the problem is the same as with strong probabilistic bisimulation and we have to check whether the two generators are the same.

Complexity. Unfortunately, the algorithm described above is exponential: there can be exponentially many Dirac determinate schedulers to consider while resolving non determinism. In such a case, we have to run the extremal point algorithm on an exponential number of candidate points.

Example 1. The automaton of Figure 2 shows a scenario in which the exponential bound is reached. Assume that all $b_{i,j}$ states are in different equivalence classes (achievable by enabling different actions from the states), and that all the transitions shown are labeled by invisible actions. Let σ be a Dirac determinate scheduler; the distribution μ generated by σ is such that $\mu(b_{i,j_i}) = \frac{1}{2^i}$, $i = 1 \ldots k-1$ and $\mu(b_{k,j_k}) = \frac{1}{2^{k-1}}$ for some combination of $j_i \in \{0,1\}$ and probability 0 to all other states. We denote a scheduler σ with the set of indexes

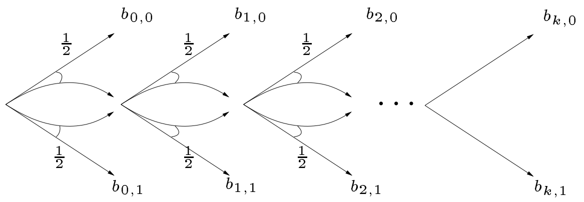

Fig. 2. An automaton with an exponential number of Dirac determinate schedulers.

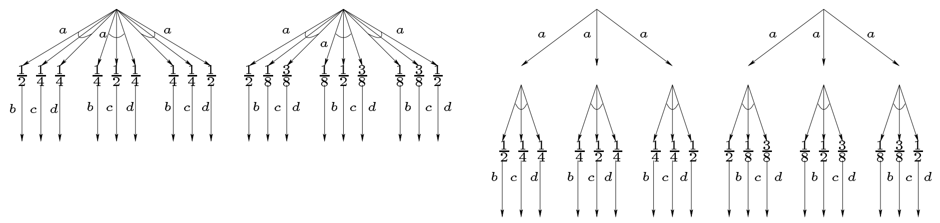

Fig. 3. Two non bisimilar probabilistic automata (left) and two non bisimilar alternating automata (right)

$\{j_i\}_{i \in 1..k}$ of the states reached with non zero probability under σ. It is clear that there are 2^k such schedulers. In order to show that all 2^k schedulers must be considered, it is necessary to show that no resulting point can be expressed as the convex combination of the others. Assume that the point reached with scheduler $\{j_i\}$ can be expressed as a convex combination of the other schedulers, then there must be another scheduler $\{l_i\}$ which has a positive weight in the convex combination. Since the two schedulers $\{j_i\}$ and $\{l_i\}$ are different, there must be an index k such that $j_k \neq l_k$, that is, they make a different choice at the k-th level. This leads to a contradiction, since $\{l_i\}$ would give a non zero probability to state b_{k,l_k} in the convex combination, while b_{k,l_k} has probability 0 under scheduler $\{j_k\}$.

The example above shows that, using the technique described in this section, all points reachable with Dirac determinate schedulers must be considered, resulting in an exponential complexity. We do not know whether this bound is inevitable or there are other techniques that avoid consideration of all these points, thus reaching a polynomial complexity.

However, despite the exponential complexity we have achieved, we argue that the algorithm is feasible in practice. Note that the worst case complexity bound is achieved when all states and transitions are reachable from a state via sequences of τ moves. We believe that such pathological cases are unlikely to occur since the sequences of τ moves in a typical probabilistic automaton are short.

In the description of the algorithms for both strong and weak probabilistic bisimulation, we have not considered efficiency: both algorithms can be made more efficient by not calculating the reachable states at each iteration and by reusing much of the previous computation. Making such improvement to the algorithms would not change their complexity classes; in particular, the algorithm for weak probabilistic bisimulation would still be exponential.

Comparison with the Alternating Model. A polynomial time algorithm for the alternating model was proposed in [20]. This algorithm uses the same partitioning technique we have used and it is based on the result that two states are weakly bisimilar if and only if the maximum probabilities of reaching each equivalence class are the same for each action. Informally, this works in the alternating model because for each probability distribution there is a state representing it, so checking maximum probabilities is enough to check exact probabilities. In our model a single state represents more than one probability distribution. Figure 3 shows how computing maximal probabilities fails to distinguish non bisimilar states for probabilistic automata, while it works in the alternating model. The start states of the two automata at the top of Figure 3 are not weakly bisimilar; however, the maximal probabilities of reaching each equivalence class (reachable with action b or c or d, respectively) are the same. The bottom part of Figure 3 shows the alternating automata obtained from those above by using the widely accepted decomposition of the transitions. In this case, the alternating automata first resolve the nondeterministic choice, then the probabilistic choice; this allows the algorithm to distinguish between non bisimilar states: the states reached after the a action have different maximal probabilities for each equivalence class.

8 Concluding Remarks

We have studied decision algorithms for bisimulation relations on probabilistic automata and we have obtained a polynomial time algorithm for strong probabilistic bisimulation and an exponential algorithm for weak probabilistic bisimulation. From a practical perspective we have argued that our exponential time algorithm should be feasible as the number of reachable equivalence classes from a state is small. It is still an open problem to find a decision algorithm for weak bisimulation, while the problem is solved already in the alternating model. However, given the results of [22], where it is shown that the probabilistic temporal logic PCTL is preserved by probabilistic bisimulation, we believe that the probabilistic version of our bisimulation relations is the most important.

References

[1] L. d. Alfaro. *Formal Verification of Probabilistic Systems.* PhD thesis, Stanford University, 1997. Available as Technical report STAN-CS-TR-98-1601.

[2] C. Baier, B. Engelen, and M. E. Majster-Cederbaum. Deciding bisimilarity and similarity for probabilistic processes. *Journal of Computer and System Sciences*, 60(1):187–231, 2000.

[3] C. Baier and H. Hermanns. Weak bisimulation for fully probabilistic processes. Technical Report 99/12, University of Twente, 1999.

[4] C. Baier and M. Stoelinga. Norm functions for bisimulations with delays. In *Proceedings of FOSSACS*. Springer-Verlag, 2000.

[5] M. Bernardo, L. Donatiello, and R. Gorrieri. Modeling and analyzing concurrent systems with MPA. In *Proceedings of the Second Workshop on Process Algebras and Performance Modelling* (PAPM), Erlangen, Germany, pages 175–189, 1994.

[6] K. Fukuda. *Polyhedral computation FAQ.* `http://www.ifor.math.ethz.ch/~fukuda/polyfaq/polyfaq.html`, 2000.

[7] R. v. Glabbeek, S. Smolka, and B. Steffen. Reactive, generative, and stratified models of probabilistic processes. *Information and Computation*, 121(1), 1996.

[8] H. Hansson. *Time and Probability in Formal Design of Distributed Systems*, volume 1 of *Real-Time Safety Critical Systems.* Elsevier, 1994.

[9] H. Hansson and B. Jonsson. A calculus for communicating systems with time and probabilities. In *Proceedings of the* 11^{th} *IEEE Symposium on Real-Time Systems,* Orlando, Fl., 1990.

[10] H. Hermanns, U. Herzog, and J. Katoen. Process algebra for performance evaluation. To appear in *Theoretical Computer Science*.

[11] J. Hillston. PEPA: Performance enhanced process algebra. Technical Report CSR-24-93, Department of Computer Science, Univ. of Edimburgh (UK), 1993.

[12] J. Hillston. *A Compositional Approach to Performance Modeling.* PhD thesis, Department of Computer Science, Univ. of Edimburgh (UK), 1994.

[13] P. Kanellakis and S. Smolka. CCS expressions, finite state processes, and three problems of equivalence. *Information and Computation*, 86(1):43–68, 1990.

[14] N. Karmakar. A new polynomial-time algorithm for linear programming. *Combinatorica 4*, pages 373–395, 1984.

[15] J. Kemeny, J. Snell, and A. Knapp. *Denumerable Markov Chains.* Graduate Texts in Mathematics. Springer-Verlag, second edition, 1976.

[16] K. Larsen and A. Skou. Bisimulation through probabilistic testing. In *Conference Record of the* 16^{th} *ACM Symposium on Principles of Programming Languages,* Austin, Texas, pages 344–352, 1989.

[17] R. Milner. *Communication and Concurrency.* Prentice-Hall International, Englewood Cliffs, 1989.

[18] C. Morgan, A. McIver, and K. Seidel. Probabilistic predicate transformers. *ACM Trans. Prog. Lang. Syst.*, 18(3):325–353, May 1996.

[19] R. Paige and R. Tarjan. Three partition refinement algorithms. *SIAM J. Comput.*, 16(6):973–989, 1987.

[20] A. Philippou, I. Lee, and O. Sokolsky. Weak bisimulation for probabilistic systems. In *Proceedings of CONCUR 2000*, pages 334–349. Springer-Verlag, 2000.

[21] R. Segala. *Modeling and Verification of Randomized Distributed Real-Time Systems.* PhD thesis, MIT, Dept. of Electrical Engineering and Computer Science, 1995. Also appears as technical report MIT/LCS/TR-676.

[22] R. Segala and N. Lynch. Probabilistic simulations for probabilistic processes. In *Proceedings of CONCUR 94*, pages 481–496. Springer-Verlag, 1994.

[23] R. Segala and N. Lynch. Probabilistic simulations for probabilistic processes. *Nordic Journal of Computing*, 2(2):250–273, 1995.

[24] K. Seidel. Probabilistic communicating processes. Technical Report PRG-102, Ph.D. Thesis, Programming Research Group, Oxford University, 1992.

[25] S. Wu, S. Smolka, and E. Stark. Composition and behaviors of probabilistic I/O automata. *Theoretical Comput. Sci.*, 176(1-2):1–38, 1999.