Seminar paper on
Probabilistic Bisimulation

Author: Johannes Raufeisen
Supervisor: Shahid Khan

RWTH Aachen University, July 2020

1 Introduction

Bisimulation in general describes a relation between transition systems that guarantees similar behavior. The ultimate goal when defining bisimulation for a certain type of transition system is to make model checking more efficient by reducing the transition system's size. As a consequence, bisimulation relations are constructed in such a way, that two bisimilar systems are equivalent under a certain logic.

Baier and Katoen have given a detailed overview of known bisimulation strategies in [2]. As an introduction to the topic of probabilistic bisimulation, we first introduce the general notion of bisimulation between to transition systems as presented in Definition 7.7 of [2].

Definition 1.1 (Bisimulation)
Let $TS = (S, Act, \longrightarrow, I, AP, L)$ be a transition system. A bisimulation for TS is a binary relation $\mathcal{R}$ such that for all $(s_1, s_2) \in R$:

- $L(s_1) = L(s_2)$
- *If $s_1' \in \text{Post}(s_1)$, then there exists an $s_2' \in \text{Post}(s_2)$ with $(s_1', s_2') \in \mathcal{R}$*
- *If $s_2' \in \text{Post}(s_2)$, then there exists an $s_1' \in \text{Post}(s_1)$ with $(s_1', s_2') \in \mathcal{R}$*

States s_1 and s_2 are bisimulation-equivalent (or bisimilar), denoted $s_1 \sim_{TS} s_2$, if there exists a bisimulation $\mathcal{R}$ for TS with $(s_1, s_2) \in \mathcal{R}$.

In order to determine the largest possible bisimulation on a given transition system, a number of algorithms such as the Paige-Tarjan algorithm in [4] have been developed. Using such an algorithm, the quotient transition system can be constructed. For the following definition, we again quote [2], this time Definition 7.11.

Definition 1.2 (Bisimulation Quotient)
For transition system $TS = (S, Act, \longrightarrow, I, AP, L)$ and bisimulation $\sim_{TS}$, the quotient transition system $TS/\sim_{TS}$ is defined by

$$TS/\sim_{TS} = (S/\sim_{TS}, \tau, \longrightarrow', I', AP, L')$$

where:

- $I' = \{[s]_{\sim_{TS}} | \in I\}$
- $\longrightarrow'$ *is defined by*

$$\frac{s \xrightarrow{\alpha} s'}{[s]_{\sim_{TS}} \xrightarrow{\tau} [s']_{\sim_{TS}}}$$

- $L'([s]_{\sim_{TS}}) = L(s)$

It can be shown that the quotient transition system is equivalent to the original transition system under CTL. A detailed proof can be found in [2]. As a result, model checking in CTL can be made more efficient by first calculating the quotient transition system.

In this seminar report, we present a similar notion of bisimulation which is suitable for probabilistically labeled transition systems (PLTS). Such transition systems are useful for modelling concurrent processes, where non-determinism and probabilistic behavior both occur. The report focuses on an algorithm developed by Groote, Verduzco and de Vink [3] in 2018 to efficiently calculate probabilistic bisimulation.

2 Preliminaries

We start by introducing the necessary terms.

Definition 2.1 (Probabilistic distribution)
Let S be a finite set. A distribution f over S is a function $f : S \to [0, 1]$ such that $\sum_{s \in S} f(s) = 1$. As an abbreviated notation, we define $f[T] \coloneqq \sum_{s \in T} f(s)$ for subsets $T \subseteq S$. We further define the support $\operatorname{supp} f \coloneqq \{s \in S | f(s) > 0\}$. *The number of elements in the support is called size of f and is denoted by $|f|$. We define $\mathcal{D}(S)$ to be the set of all distributions over S.*

Definition 2.2 (Probabilistic labeled transition systems)
A probabilistic labeled transition system (PLTS) with a set of actions Act is a tuple $\mathcal{A} = (S, \to)$ such that

- *S is a finite set, containing the states of the system*
- *$\to \subseteq S \times Act \times \mathcal{D}(S)$ is a finite relation, containing the transitions of the system*

Definition 2.3 (Notation)
We use the following terms and abbreviated notations for a PLTS A:

- *We will write $s \xrightarrow{a} f$ instad of $(s, a, f) \in \rightarrow$*
- *For each distribution f such that $\exists s, a$ with $s \xrightarrow{a} f$ we call u_f the associated probabilistic state*
- *The set of all probabilistic states is denoted by U*

Definition 2.4 (Probabilistic bisimulation)
Let $\mathcal{A} = (S, \rightarrow)$ be a PLTS. An equivalence relation $\sim$ on S is called a probabilistic bisimulation if and only if:

- *For all states $s, t \in S$ such that $s \sim t$ and $s \xrightarrow{a} f$ for some $a \in Act$ and $f \in \mathcal{D}(S)$ there is a $g \in \mathcal{D}(S)$ such that $t \xrightarrow{a} g$ and $f[B] = g[B] \quad \forall B \in S/\sim$*

Two states $s, t \in S$ are called probabilistically bisimilar if and only if:

- *There is a probabilistic bisimulation $\sim$ such that $s \sim t$*

Two distributions $f, g \in \mathcal{D}(S)$ resp. their probabilistic states u_f, u_g care called probabilistically bisimilar if and only if:

- *There is a probabilistic bisimulation $\sim$ such that for all sets $B \in S/\sim$ it holds that $f[B] = g[B] \quad \forall u, v \in B$. We write $u_f \sim u_g$ if f and g are bisimilar.*

This definition is best understood by a short example. Consider the following graphical description of two PLTS. For better visibility, action states are colored blue, while probabilistic states are colored orange.

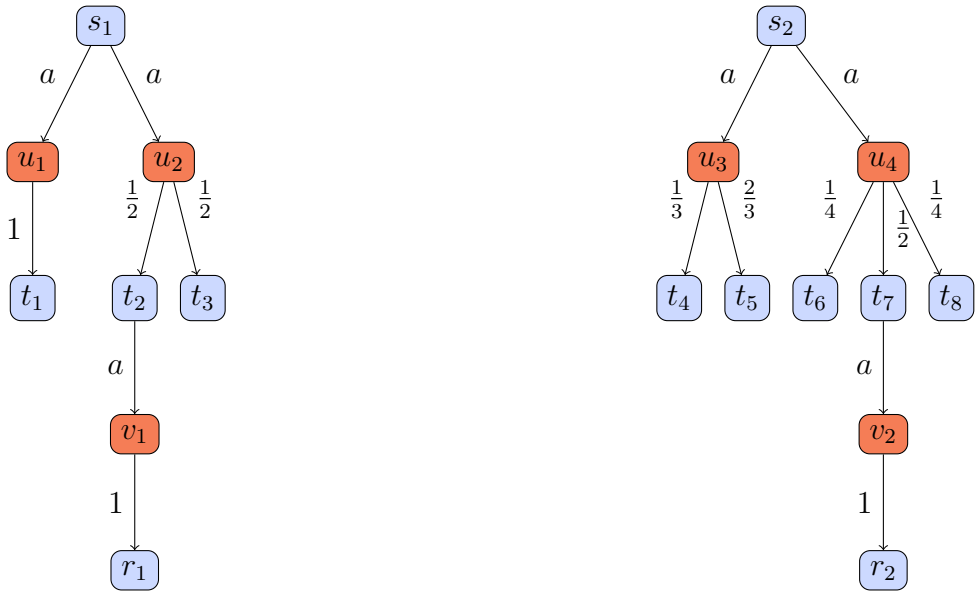

Figure 1: Two probabilistically bisimilar nondeterministic transition systems

As can be seen, these two PLTS behave 'in a similar way'. We assume s_1 and s_2 to be their initial states. Then, after an initial non-deterministic choice, one of two cases will occur.

1. The PLTS enters a probabilistic state, such that the following action state will be a terminal state with probability 1.

2. The PLTS enters a probabilistic state, such that with probability $\frac{1}{2}$ the following action state will be a terminal state and with probability $\frac{1}{2}$ the following action state is not a terminal state and instead provides an action a which will then lead to a terminal state with probability 1.

This behavior can be formally captured by the following probabilistic bisimulation:

$$\begin{gathered} s_1 \sim s_2 \\ u_1 \sim u_3 \\ u_2 \sim u_4 \\ t_1 \sim t_3 \sim t_4 \sim t_5 \sim t_6 \sim t_8 \\ t_2 \sim t_7 \\ v_1 \sim v_2 \\ r_1 \sim r_2 \end{gathered}$$

In order to understand the importance of probabilistic bisimulation for the overall subject of model checking, we cite the following theorem.

Theorem 2.1
Two probabilistic bisimilar PLTS are equivalent under Probabilistic Computation Tree Logic.

Proof 2.1
A proof by Segala and Lynch can be found in [5].

In other words, determining whether $A \vDash \phi$ holds for a PLTS A and a PCTL-formula ϕ is equivalent to determining whether $A_\sim \vDash \phi$ holds, given that A and $A_\sim$ are probabilistically bisimilar.

3 Algorithm

In the following, we will discuss an algorithm given in [3] that allows us to compute the largest possible bisimulation.

Theorem 3.1 (Bijection between equivalence relations and partitions)
Let X be a finite set, $P \subseteq \mathcal{P}(X)$ a partition of X and $\sim$ an equivalence relation on X.

(a) P induces an equivalence relation $\sim_P$ defined by $x \sim_P y \iff \exists A \in P : x \in A \wedge y \in A$

(b) $\sim$ induces a partition $P_\sim \coloneqq \{[x]_\sim \mid x \in X\}$ where $[x]_\sim$ denotes the equivalence class of x

(c) There is a bijection between the set of equivalence relations on X and the set of partitions of X

Proof 3.1
Known from linear algebra.

Because of this equivalence in the aforementioned theorem, we will from now on work with probabilistic bisimulation as a partition on the set of all action states and probabilistic states.

Definition 3.1 (Stable sets and partitions)

- *A set of action states B is called stable under a set of probabilistic states $C \subseteq U$ iff for all actions $a \in Act$ it holds that:*

$$s \xrightarrow{a} C \iff t \xrightarrow{a} C \quad \forall s, t \in B$$

- *A set of probabilistic states $B \subseteq U$ is called stable under a set of action states $C \subseteq S$ iff*

$$u[C] = v[C] \quad \forall u, v \in B$$

- *A partition $\mathcal{B}$ is called stable under another partition $\mathcal{C}$, if each block $B \in \mathcal{B}$ is stable under all sets of action states or probabilistic states, repsectively.*

Theorem 3.2 (Stability under itself)
Let $A = (S, \longrightarrow)$ be a PLTS and $\mathcal{B}$ a partition of S.
If $\mathcal{B}$ is stable under itself, then the corresponding equivalence relation $\sim_B$ is a probabilistic bisimulation.

Proof 3.2
We check the definition of probabilistic bisimilarity:
Suppose two action states $s, t \in S$ are in the same equivalence class $B \in \mathcal{B}$. Let there be a transition $s \xrightarrow{a} u_f$ with $f \in \mathcal{D}(S)$. Then there is a block $B' \in P$ which contains u_f, so that $s \xrightarrow{a} B'$. Since $\mathcal{B}$ is stable under itself, B is stable under B'. Consequently, there is $g \in B'$ such that $t \xrightarrow{a} u_g$.

Now let $B'' \in P$ be an arbitrary block. Since B' is stable under B'', it holds that

$$u_f[B''] = u_g[B'']$$

And thus, $\sim_B$ fulfills the definition of a probabilistic bisimulation.

3.1 Basic algorithm

Below we present the algorithm developed by Groote, Verduzco and de Vink to compute probabilistic bisimulation for a given PLTS.

Algorithm 1 Abstract partition refinement algorithm for probabilistic bisimulation

1: **function** PARTITION REFINEMENT
2: $\mathcal{C} := \{S, U\}$
3: $\mathcal{B} := \{U\} \cup \{S_A \mid A \subseteq Act\}$
4: where $S_A = \left\{ s \in S \mid \forall a \in Act \left(\exists u \in U : s \xrightarrow{a} u \iff a \in A \right) \right\}$
5: **while** $\mathcal{C}$ contains a non-trivial constellation C **do**
6: choose block B_C from $\mathcal{B}$ in C
7: replace in $\mathcal{C}$ constellation C by B_C and $C \backslash B_C$
8: **if** C contains probabilistic states **then**
9: **for all** blocks B of action states in $\mathcal{B}$ unstable under B_C or $C \backslash B_C$ **do**
10: refine $\mathcal{B}$ by splitting B
11: into blocks of states with the same actions into B_C and $C \backslash B_C$
12: **else**
13: **for all** blocks B of probabilistic states in $\mathcal{B}$ unstable under B_C **do**
14: refine $\mathcal{B}$ by splitting B
15: into blocks of states with equal probabilities into B_C
16: **return** $\mathcal{B}$

Definition 3.2 (Nomenclature)
For a more precise discussion of the above algorithm, the following terms are used

- $\mathcal{B}$ *is called the set of blocks. Each* $B \in \mathcal{B}$ *is called a block.*
- $\mathcal{C}$ *is called the set of constellations. Each* $C \in \mathcal{C}$ *is called a constellation.*

3.2 Correctness

We will proof correctness of the above algorithm by establishing three invariants that are given below. Even though their proofs are quite technical, they can provide useful insights into the overall structure of the algorithm.
Afterwards, we will discuss an example on how the algorithm works given a specific PLTS.

Theorem 3.3 (Invariant 1)
Probabilistic bisimilarity $\simeq_p$ *is a refinement of B.*

Proof 3.3
We have to show that for all pairs of bisiimilar states there is a block $B \in \mathcal{B}$ *that cotains both states. Let* $s, t \in S$ *be action states with* $s \simeq_p t$*. Let* $u_f, u_g \in U$ *be probabilistic states with* $u_g \simeq_p u_g$*.*

We start with the initialization in line 3: If $s \simeq_p t$*, then according to definition, there is a probabilistic bisimulation* $\sim$ *such that* $s \sim t$*. That means, that for every* f *with* $s \xrightarrow{a} f$ *there is a state* g *with* $t \xrightarrow{a} g$*. In other words, every action that is enabled in* s *is also enabled in* t*. As* $\sim$ *is an equivalence relation, we also have* $t \sim s$ *and thus an action* a *is enabled in* s *iff it is enabled in* t*. If* A *now denotes the set of enabled actions in* s*, we have found that* $s, t \in S_A$ *are placed in the same block. If* $u_f \simeq_p u_g$*, then obviously* $u_f, u_g \in U$ *are placed within the same block as well.*

Now we show that this invariant is maintained within each iteration of the while-loop. In the first part, we only consider action states $s \simeq_p t$. As $s \simeq_p t$ at the start of the while-loop, they are placed in a common block $B_{st} \in \mathcal{B}$. This block B_{st} will only be altered within the while-loop, if there is a non-trivial constellation C consisting of only probabilistic states and a block $B_C \in B$ such that B_{st} is unstable under either B_C or $C \backslash B_C$. In this case, we split B_{st} in parts, so that the resulting refined sets $B_1, ..., B_n$ fulfill $\forall i, a$:

$$x, y \in B_i \Longrightarrow \left(x \xrightarrow{a} B_C \Longleftrightarrow y \xrightarrow{a} B_C \right)$$
$$x, y \in B_i \Longrightarrow \left(x \xrightarrow{a} C \backslash B_C \Longleftrightarrow y \xrightarrow{a} C \backslash B_C \right)$$

Since s and t are bisimilar and $\simeq_p$ is a refinement of $\mathcal{B}$, they already fulfill $\forall a$ that

$$\left(s \xrightarrow{a} B_C \Longleftrightarrow t \xrightarrow{a} B_C \right)$$

because $B_C \in \mathcal{B}$. As a consequence, they also have the same actions into $C \backslash B_C$:

$$\left(s \xrightarrow{a} C \backslash B_C \Longleftrightarrow t \xrightarrow{a} C \backslash B_C \right)$$

Thus, s and t will be placed in the same block B_i, so Invariant 1 will not be violated.

Let us now consider probabilistic states $u_f \simeq_p u_g$. As $\simeq_p$ is a refinement of $\mathcal{B}$ at the start of the while-loop, the blocks B_C and $C \backslash B_C$, there are disjunct blocks $B_i, C_i \in P_{\simeq_p}$ such that

$$B_C = \bigcup_{i=1}^{n} B_i$$
$$C \backslash B_C = \bigcup_{i=1}^{m} C_i$$

And as u_f and u_g are bisimilar, they probabilities on the sets B_i, C_i are equal:

$$u_f(B_i) = u_g(B_i)$$
$$u_f(C_i) = u_g(C_i)$$

Now we utilize, that the blocks B_i, C_i are parwise disjunct to calculate

$$u_f(B_C) = \sum_{i=1}^{n} u_f(B_i) = \sum_{i=1}^{n} u_g(B_i) = u_g(B_C)$$
$$u_f(C \backslash B_C) = \sum_{i=1}^{n} u_f(C_i) = \sum_{i=1}^{n} u_g(C_i) = u_g(C \backslash B_C)$$

We have shown that u_f and u_g have equal probabilities into B_C, so that in line 15, they will both be placed in the same newly refined block.
This concludes our proof of Invariant 1.

Theorem 3.4 (Invariant 2)
Partition B is a refinement of partition C.

Proof 3.4
Invariant 2 clearly holds at the time of initialization, since $S_A \in S \ \forall A \subseteq Act$.

Now directly to the while-loop: We only consider the constellation $C \in \mathcal{C}$, since everything in $\mathcal{C}$ else is left untouched. By induction hypothesis, we know that C is the union of disjunct blocks within B:

$$C = \bigcup_{i=1}^{n} B_i$$

By choice of $B_C \in \mathcal{B}$, the resulting smaller blocks $B_C \in \mathcal{C}$ and $C \backslash B_C \in \mathcal{C}$ are already disjunct unions of blocks within B:

$$B_C = B_C$$
$$C \backslash B_C = \bigcup_{i=1, B_i \neq B_C}^{n} B_i$$

In the remaining lines of the while-loop, we only further refine $\mathcal{B}$ and thus do not violate Invariant 2.

Theorem 3.5 (Invariant 3)
Partition $\mathcal{B}$ is stable under the set of constellations $\mathcal{C}$.

Proof 3.5
Let us first consider the initialization in lines $2-3$. Then, $\mathcal{B}$ is stable under $\mathcal{C}$ because:

- *U is stable under S as we have*

$$u[S] = 1 = v[S] \quad \forall u, v, \in U$$

- *S_A is stable under S for all $A \subseteq Act$ because by definition of S_A it holds that*

$$s \xrightarrow{a} U \iff t \xrightarrow{a} U \quad \forall s, t \in S_A$$

Now we proof that Invariant 3 is not violated within the while-loop. Therefore, we have to show that after splitting C into B_C and $C \backslash B_C$, $\mathcal{B}$ will be stable under the new blocks B_C and $C \backslash B_C$. That is exactly what we achieve through the refinements in lines $8-15$. By splitting each unstable block into new blocks with the same actions or equal probabilities into B_C we directly fulfill the definition of stability again.

Theorem 3.6 (Correctness of Algorithm 1)
Algorithm 1 correctly computes probabilistic bisimilarity for a given PLTS. On termination, the partition $\mathcal{B}$ is equal to the partition $P_{\simeq_p}$ of bisimilar states.

Proof 3.6
"$\subseteq$": After termination of Algorithm 1, the partition $\mathcal{C}$ only contains trivial constellations, which is guaranteed by the while-loop. This means, that every block $C \in \mathcal{C}$ consists of exactly one block $B \in \mathcal{B}$. Due to Invariant 2, $\mathcal{B}$ and $\mathcal{C}$ coincide.
By invariant 3, $\mathcal{B}$ is then stable under itself. And as such, $\mathcal{B}$ is a probabilistic bisimulation according to Theorem 3.2.
Thus, two states in the same block within $\mathcal{B}$ are probabilistically bisimilar.

"$\supseteq$": Due to Invariant 1, two states that are probabilistically bisimilar are placed in the same block within $\mathcal{B}$. This concludes the proof of correctness.

4 Example

In the following, we will provide an example of how algorithm 1 works. Therefore, consider the PLTS in Figure 2.

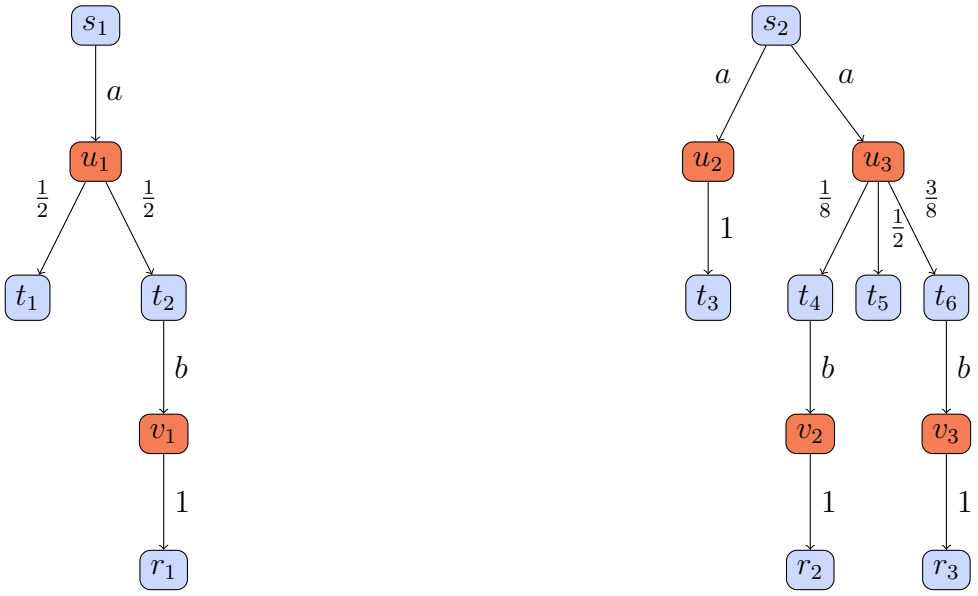

Figure 2: Example PLTS for algorithm 1

The partitions $\mathcal{B}$ and $\mathcal{C}$ are initialized as follows:

$$\mathcal{C} = \{U, S\} = \{\{u_{1-3}, v_{1-3}\}, \{s_1, s_2, t_{1-6}, r_{1-3}\}\}$$
$$\mathcal{B} = \{U, S_{\{a\}}, S_{\{b\}}, S_{\varnothing}\} = \{\{u_{1-3}, v_{1-3}\}, \{s_1, s_2\}, \{t_2, t_4, t_6\}, \{t_1, t_3, t_5, r_{1-3}\}\}$$

Now, we will explain each iteration of the while-loop.

1. $\mathcal{C}$ clearly contains a non-trivial constellation, as for example $S_{\{a\}} \subsetneq S$. We now chose $C = S$ and $B_C = S_{\{a\}}$. As C only contains action states, lines $13 - 15$ are executed and each block of probabilistic states is refined if necessary. Therefore, we have to compute the different probabilities into B_C. These are trivial:

$$u_i[B_C] = u_i[\{s_1, s_2\}] = 0 \quad \forall i$$
$$v_i[B_C] = v_i[\{s_1, s_2\}] = 0 \quad \forall i$$

   Therefore, no block of probabilistic states is unstable under B_C and no splitting has to be done.

   Thus, we result in

$$\mathcal{C} = \{U, S_{\{a\}}, S \backslash S_{\{a\}}\} = \{U, \{s_1, s_2\}, \{t_{1-6}, r_{1-3}\}\}$$
$$\mathcal{B} = \{\{u_{1-3}, v_{1-3}\}, \{s_1, s_2\}, \{t_2, t_4, t_6\}, \{t_1, t_3, t_5, r_{1-3}\}\}$$

2. We choose $C = \{t_{1-6}, r_{1-3}\}$ and $B_C = \{t_2, t_4, t_6\}$. As C only contains action states, each block of probabilistic states in B is refined if necessary. We compute

$$u_1[B_C] = u_3[B_C] = \frac{1}{2}$$
$$u_2[B_C] = v_i[B_C] = 0$$

Thus, U is split into $\{u_1, u_3\}$ and $\{u_2\}$, resulting in

$$\mathcal{C} = \{U, \{s_1, s_2\}, \{t_2, t_4, t_6\}, \{t_1, t_3, t_5, r_{1-3}\}\}$$
$$\mathcal{B} = \{\{u_1, u_3\}, \{u_2, v_{1-3}\}, \{s_1, s_2\}, \{t_2, t_4, t_6\}, \{t_1, t_3, t_5, r_{1-3}\}\}$$

3. We choose $C = U$ and $B_C = \{u_2, v_{1-3}\}$. As B_C only contains probabilistic states, we now have to refine all blocks of action states in B. Therefore, we notice that

$$s_1 \nrightarrow B_C \quad \text{but} \quad s_2 \rightarrow B_C$$

This means, the block $\{s_1, s_2\} \in \mathcal{B}$ is unstable under B_C and has to be split. All other blocks in $\mathcal{B}$ are stable under B_C. Therefore, we end up with

$$\mathcal{C} = \{\{u_1, u_3\}, \{u_2, v_{1-3}\}, \{s_1, s_2\}, \{t_2, t_4, t_6\}, \{t_1, t_3, t_5, r_{1-3}\}\}$$
$$\mathcal{B} = \{\{u_1, u_3\}\{u_2, v_{1-3}\}, \{s_1\}, \{s_2\}, \{t_2, t_4, t_6\}, \{t_1, t_3, t_5, r_{1-3}\}\}$$

4. We choose $C = \{s_1, s_2\}$ and $B_C = \{s_1\}$. As B_C only contains action states, we have to refine all blocks of probabilistic states that are unstable under B_C. To that end, we compute

$$u_i[B_C] = v_i[B_C] = 0 \quad \forall i$$

Thus, we end up with

$$\mathcal{C} = \{\{u_1, u_3\}\{u_2, v_{1-3}\}, \{s_1\}, \{s_2\}, \{t_2, t_4, t_6\}, \{t_1, t_3, t_5, r_{1-3}\}\}$$
$$\mathcal{B} = \{\{u_1, u_3\}\{u_2, v_{1-3}\}, \{s_1\}, \{s_2\}, \{t_2, t_4, t_6\}, \{t_1, t_3, t_5, r_{1-3}\}\}$$

We can see,that now $\mathcal{B} = \mathcal{C}$ and the algorithm terminates.

As a conclusion, the algorithm found the following probabilistically bisimilar states.

$$u_1 \simeq_p u_3$$
$$u_2 \simeq_p v_1 \simeq_p v_2 \simeq_p v_3$$
$$t_2 \simeq_p t_4 \simeq_p t_5$$
$$t_1 \simeq_p t_3 \simeq_p t_6 \simeq_p r_1 \simeq_p r_2 \simeq_p r_3$$

And by correntess of algorithm 1, these are all pairs of bisimilar states that can be found.

5 Complexity

5.1 Data structures

In this section, we will focus on how to effectively implement algorithm 1 and discuss its complexity. Therefore, we start by describing the basic data structures that are used.
All of these data structures can be used in one of two ways. Either related to action states or related to probabilistic states. We will mention differences among these two types where necessary.

Definition 5.1 (Global variables)
Within the algorithm, we use the following global variables:

- *A list $\mathcal{B}$ of blocks*
- *A list $\mathcal{C}$ of constellations*
- *A list of transitions*
- *A list of states*
- *A stack of non-trivial constellations for constant time read-access*
- *An array state_to_constellation_cnt with the following purpose: For each transition $s \xrightarrow{a} u$ we store the number of action transitions that go from s to the constellation C with $u \in C$ using the action a. This number is then stored in the array.*

Definition 5.2 (Transition)
A transition has the following attributes:

- *from - referring to a state*
- *to - referring to a state*
- *label - We assume action transitions to be consecutively numbered. Then the label attribute is given by the appropriate action number. In case of probabilistic transitions, label is given by the fraction describing the appropriate probability for this transition.*
- *state_to_constellation_cnt_ptr - A pointer to the appropriate value within the global array state_to_constellation_cnt*

Definition 5.3 (State)
A state has the following attributes

- *list of incoming transitions*
- *reference to the block $B \in \mathcal{B}$, that contains the state*
- *mark_state - A boolean used for computations explained later in this section*
- *residual_transition_count - An integer used for further computations*
- *transition_cnt_ptr - A pointer used for further computations*

- *In probabilistic states, there is the variable cumulative_prob used for further computations*

Definition 5.4 (Block)
A block has the following attributes

- *A reference to the constellation $C \in \mathcal{C}$ in which the block is contained*
- *A list of states within the block*
- *The number of states within the block*
- *A list of incoming transitions. In the case, where a block only contains probabilistic states, the list of incoming action transitions is ordered by action label.*
- *A variable indicating if this block is marked.*

Definition 5.5 (Constellation)
A constellation has the following attributes:

- *A list of blocks in this constellation*
- *Number of states in this constellation*

5.2 Efficient refinement

In Algorithm 1, we left open the question on how to efficiently calculate refinement. This is done in two steps: First by marking blocks that need to be split and second by carrying out the splitting.
We first concentrate on marking the blocks. For blocks of action states, we will use the function aMark, for blocks of probabilistic states we will instead use pMark.

Definition 5.6 (aMark)
Let $C \in \mathcal{C}$ be a non-trivial constellation and $B_C \in \mathcal{B}$ a block contained in C. Let $a \in Act$ be an action.
The function $\mathrm{aMark}(\mathcal{B}, C, B_C, a)$ *returns a tuple* $(B_a, \mathrm{left_a}, \mathrm{mid_a}, \mathrm{right_a}, \mathrm{large_a})$ *defined by*

$$
\begin{aligned}
B_a &= \{B \in \mathcal{B} \mid \exists s \in B : s \xrightarrow{a} B_C\} \\
\mathrm{left_a}(B) &= \{s \in B \mid s \xrightarrow{a} B_C \wedge s \not\xrightarrow{a} C \backslash B_C\} \\
\mathrm{mid_a}(B) &= \{s \in B \mid s \xrightarrow{a} B_C \wedge s \xrightarrow{a} C \backslash B_C\} \\
\mathrm{right_a}(B) &= \{s \in B \mid s \not\xrightarrow{a} B_C \wedge s \xrightarrow{a} C \backslash B_C\}
\end{aligned}
$$

Here, $\mathrm{left_a}$, $\mathrm{mid_a}$ *and* $\mathrm{right_a}$ *are themselves functions that return a set of states for each $B \in B_a$*
The computation of this tuple as carried out as follows:

1. *In order to compute B_a we can traverse the list of all incoming action transitions into B_C. Then for each such transition $s \xrightarrow{a} B_C$, we store the block of s in B_a.*

2. *While traversing this list of incoming action transitions and given a block $B \in \mathcal{B}$, we store all such states $s \in B$ with a transition into B_C in the set* $\text{left}_a(B)$. *For the first state that we move in* left_a *we set residual_transition_count to state_to_constellation_cnt -1. For every other action transition, where the source state is already contained in* left_a*, we decrease residual_transition_count of the source state by one.*

3. *All remaining states $s \in B$ form the set* $\text{right}_a(B)$.

4. *We iterate over all states in* $\text{left}_a(B)$ *and move those to* $\text{mid}_a(B)$ *where the residual_transition_count is not equal to zero.*

5. *During all the movement operations, we can keep track of all the sets' sizes and thus keep track of the maximum set among* $\text{left}_a(B)$, $\text{mid}_a(B)$, $\text{right}_a(B)$. *This maximum set is then returned as* $\text{large}_a(B)$

Theorem 5.1 (Correctness and complexity of aMark)
The function aMark *correctly computes the sets* $\text{left}_a(B)$, $\text{mid}_a(B)$, $\text{right}_a(B)$ *and does so in O(number of incoming action transitions into B_C)*

Proof 5.1
We start by observing, that for $B \in B_a$ the set $A \coloneqq \{s \in B \mid s \overset{a}{\not\to} B_C \wedge s \overset{a}{\not\to} C \backslash B_C\}$ must be empty. If this was not the case, then there would be a state $s \in B$ with no outgoing transition into C. But according to the definition of B_a there already is a state $t \in B$ with $s \overset{a}{\to} B_C$. In other words, t has an outgoing transition into C. But since s and t are within the same block of $\mathcal{B}$, this is a contradiction to Invariant 3, stating $\mathcal{B}$ is stable under $\mathcal{C}$.

Now that we know that A must be empty, we can already argue that the computation of $\text{right}_a(B)$ *is correct: As we remove all states with outgoing transitions into B_C from B, the remaining set is equal to*

$$\{s \in B \mid s \overset{a}{\not\to} B_C\} = \text{right}_a(B) \cup A = \text{right}_a(B)$$

Now we proof that the computation of $\text{left}_a(B)$ *is correct. This is mainly guaranteed by the definition of state_to_constellation_cnt . If the variable residual_transition_count reaches zero, it means that during step 2 of the above algorithm, we have visited all transitions that go from state s to the constellation C. But as we have only iterated over all incoming transitions into B_C, the number of action transitions $s \overset{a}{\to} B_C$ is equal to the number of action transitions $s \overset{a}{\to} C$. Thus, there cannot be another action transition $s \overset{a}{\to} C \backslash B_C$ which proves that the computation of* $\text{left}_a(B)$ *is correct.*

The same argumentation shows that the computation of $\text{mid}_a(B)$ *is correct. If the amount of action transitions from s into B_C is not equal to the amount of action transitions from s into C, there must necessarily be at least one transition from s into $C \backslash B_C$.*

The time complexity is easy to prove, since there are only two iterations needed, i.e. in step 2 and step 4. In step 2, we directly iterate over all incoming action transitions. In step 4, we iterate over all states in $\text{left}_a(B)$. *But the amount of states in this set is by definition limited above by the amount of incoming action transitions into B_C. Thus, the overall time complexity is O(number of incoming action transitions into B_C).*

Definition 5.7 (pMark)
Let $C \in \mathcal{C}$ be a non-trivial constellation and $B_C \in \mathcal{B}$ a block contained in C.
The function $\mathrm{pMark}(\mathcal{B}, C, B_C)$ *returns a tuple* $(B_p, \mathrm{left_p}, \mathrm{mid_p}, \mathrm{right_p}, \mathrm{large_p})$ *defined by*

$$\begin{aligned} B_p &= \{B \in \mathcal{B} \mid \exists u \in B : u[B_C] > 0\} \\ \mathrm{left_p}(B) &= \{u \in B \mid u[B_C] = 1\} \\ \mathrm{mid_p}(B) &= \{\{u \in B \mid u[B_C] = q\} \mid q \in (0,1)\} \\ \mathrm{right_p}(B) &= \{u \in B \mid u[B_C] = 0\} \\ \mathrm{large_p}(B) &= \textit{the largest set among the above} \end{aligned}$$

The computation of this tuple is carried out as follows:

1. *Traverse through the list of incoming probabilistic transitions into B_C. For each such transition $u \to_p B_C$, look up the block B which contains u.*
   - *If $B \notin B_p$, add B to B_p. Set cumulative_prob of state u to p and move u temporarily to* $\mathrm{left_p}(B)$.
   - *If $B \in B_p$, increase cumulative_prob of state u by p.*
2. *Add all remaining states in B, that have not been moved yet to* $\mathrm{right_p}(B)$.
3. *Traverse the list of states s in* $\mathrm{left_p}(B)$ *and move all states with cumulative_prob < 1 to a new set M. The remaining states form* $\mathrm{left_p}(B)$.
4. *Sort M with respect to cumulative_prob in increasing order.*
5. *By traversing M, split M into the single subsets* $\mathrm{mid_p}(B)$.
6. *While moving single states to new sets, we can keep track of the corresponding sizes and thus determine* $\mathrm{large_p}(B)$.

Theorem 5.2 (Correctness and complexity of pMark)
The function pMark *correctly computes the sets* $\mathrm{left_p}(B)$, $\mathrm{mid_p}(B)$, $\mathrm{right_p}(B)$ *and does so in O(number of incoming probability transitions + sorting penalty)*

Proof 5.2
By construction of step 1 in above definition, cumulative_prob of a state u contains $u[B_C]$ after having traversed the complete list of incoming transitions. This already shows that the above computation is correct.
The sorting in step 4 will be useful later on, when we calculate the time complexity of the complete algorithm. If we discard it for now, what remains is an iteration over all incoming probabilistic transitions in step 1 and another iteration over $\mathrm{left_p}(B)$ *in step 3. But the size of* $\mathrm{left_p}(B)$ *is bounded above by the number of incoming probabilistic transitions into B_C due to construction in step 1.*
So the overall time complexity of pMark is O(number of incoming probability transitions + sorting penalty).

The marking functions aMark and pMark will be used in the detailed description of the algorithm which is provided in the next section.
Before we can accurately describe the complete algorithm, we still have to explain how the splitting of blocks can be achieved. This is explained below.

Definition 5.8 (Splitting)
Splitting a block B into a smaller block $B' \subseteq B$ and its complement $B \backslash B'$ is done in the following way:

1. *Iterate over all states in B' and assign them to the block B'. Assuming B' is already stored in a separate list, moving a single state can be done in constant time.*

2. *For each moved state, iterate over its incoming transitions. Remove each incoming transition from the list of incoming transitions of B and add it to the list of incoming transitions of B'.*

3. *After having moved all states, group the incoming action transitions of B' per action label.*

4. *Iterate over all incoming transitions $s \xrightarrow{a} u$ with $u \in B_C$*

   - *If residual_transition_count of s is positive, create a new value in the array of state_to_constellation_cnt values. Set this value to state_to_constellation_cnt - residual_transition_count and store a pointer of that new entry in transition_cnt_ptr .*

     At the same time, replace the old value in state_to_constellation_cnt for this transition to residual_transition_count of state s.

The fourth step in splitting is necessary, because we have to maintain the ordering of incoming action transitions as mentioned in 5.4. This ordering will later be useful in the detailed algorithm.

5.3 Detailed algorithm

With all the above definitions, we can now present the final algorithm in detail and analyze its complexity.

Algorithm 2 Partition refinement algorithm for probabilistic bisimulation

1:	**function** Partition Refinement $(S, U, \to)$	
2:	$\mathcal{C} := \{S, U\}$	$\} O(n_a + n_p)$
3:	$\mathcal{B} := \{U\} \cup \{S_A \mid A \subseteq Act\}$ where $S_A = \left\{ s \in S \mid \forall a \in Act \left(\exists u \in U : s \xrightarrow{a} u \iff a \in A \right) \right\}$	$\} O(n_p + n_a + m_a)$
4:	group the incoming action transitions in each block per label	$\} O(m_a)$
5:	initialise `state_to_constellation_cht` for each transition	$\} O(m_a)$
6:	**while** $\mathcal{C}$ contains a non-trivial constellation C **do**	$\} \leq n$ iterations
7:	choose a block B_C from $\mathcal{B}$ in C such that $\lvert B_C \rvert \leq \frac{1}{2}\lvert C \rvert$	
8:	split constellation C into B_C and $C \backslash B_C$ in C	$\} O(1)$
9:	**if** C contains probabilistic states **then**	
10:	**for all** incoming actions a of states in B_C **do**	$\} \leq \lvert Act \rvert$ iterations
11:	$\langle \boldsymbol{B}_a, \text{left}_a, \text{mid}_a, \text{right}_a, \text{large}_a \rangle :=$aMark$(\mathcal{B}, C, B_C, a)$	$\} O($ nr of incoming a transitions in $B_C)$
12:	**for all** blocks $B \in \mathbf{B}_a$ **do**	
13:	**for all** non-empty subsets $B' \subseteq B$, different from $\text{large}_a(B)$ in $\{\text{left}(B)_a, \text{mid}_a(B), \text{right}_a(B)\}$ **do**	$\} O($ nr of incoming a transitions in $B_C)$
14:	move B' out of B and add B' as new block to $\mathcal{B}$	$\} O($ nr of incoming transitions in $B')$
15:	**else**	
16:	$\langle \boldsymbol{B}_p, \text{left}_p, \text{mid}_p, \text{right}_p, \text{large}_p \rangle :=$ pMark $(\mathcal{B}, C, B_C)$	$\} O($nr of incoming prob. transitions in $B_C)$ plus a sorting penalty
17:	**for all** blocks $B \in \mathbf{B}_p$ **do**	
18:	**for all** non-empty sets of states $B' \subseteq B$ not equal to $\text{large}_p(B)$ in $\{\text{left}_p(B)\} \cup \text{mid}_p(B) \cup \{\text{right}(B)_p\}$ **do**	$\} O($ nr of incoming prob. transitions in $B_C)$
19:	move B' out of B and add B' as a new block to $\mathcal{B}$	$\} O($ nr of incoming transitions in $B')$
20:	**return** $\mathcal{B}$	

Theorem 5.3 (Correctness of algorithm 2)
Algorithm 2 correctly computes probabilistic bisimilarity for a given PLTS. On termination, the partition $\mathcal{B}$ is equal to the partition $P_{\simeq_p}$ of bisimilar states.

Proof 5.3
As correctness of algorithm 1 is already known, we need to show equivalence between algorithm 1 and 2. As only minor changes have been made to most of the lines, we only need to consider lines 10-14 and 16-19 of algorithm 2.

- *Lines 10-14 of algorithm 2 are equivalent to lines 9-10 of algorithm 1.*

  First, we have to show that all blocks B of action states, that are unstable under B_C or $C \backslash B_C$ are visited in algorithm 2. Assume, that such a block is not contained in any B_a, the by definition of B_a, there is no outgoing transition from B to B_C. And thus, B must be stable under B_C and $C \backslash B_C$ which is a contradition.

  Second, we have to show that an unstable block B is split in blocks of states with the same actions into B_C and $C \backslash B_C$. This is guaranteed by definition of the sets $\mathrm{left_a}(B)$, $\mathrm{mid_a}(B)$ *and* $\mathrm{right_a}(B)$. *The fourth set*

  $$\{s \in B \mid s \overset{a}{\not\to} B_C \wedge s \overset{a}{\not\to} C \backslash B_C\}$$

  must already be empty, because otherwise there is a state t with $t \overset{a}{\not\to} C$. But as there also is a state s with $s \overset{a}{\to} B_C \subseteq C$, B is not stable under C. This is a violation of Invariant 3 which also holds for algorithm 2.

- *Lines 16-19 of algorithm 2 are equivalent to lines 12-13 of algorithm 2.*

  First, we have to show that all blocks B of probabilistic states, that are unstable under B_C are visited in algorithm 2. Let B be a block with $B \notin B_p$. Then $u[B_C] = 0 \quad \forall u \in B$ and consequently B is stable under B_C. Thus, all blocks that might be unstable under B_C are contained in B_p.

  Second, we have to show that an unstable block B is split in blocks of states with equal probabilities in B_C. But this once again follows from definition of the sets $\mathrm{left_p}(B)$, $\mathrm{mid_p}(B)$ *and* $\mathrm{right_p}(B)$.

5.4 Complexity analysis

For ease of notation, we make the following definition:

$$\begin{aligned} n_a &\coloneqq \text{ number of action states} \\ n_p &\coloneqq \text{ number of probabilistic states} \\ m_a &\coloneqq \text{ number of action transitions} \\ m_p &\coloneqq \text{ number of probabilistic transitions} \end{aligned}$$

With these abbreviations, we can determine the time complexity of above algorithm.

Theorem 5.4 (Time and space compelxity)
The total time complexity of algorithm 2 is $O\left((m_a + m_p) \log n_p + (m_p + n_a) \log n_a\right)$ and the space complexity is $O\left(m_a + m_p + n_a\right)$.

Proof 5.4
We first consider time complexity and turn to space complexity afterwards.
In line 2*, the set* $\mathcal{C}$ *of constellations can be constructed by iterating over all states, thus giving a time complexity of* $O(n_a + n_p)$*.*

In line 3*, the sets* S_A *can be constructed by performing a partition refinement on* S*. Starting of with the obvious partition* $\{S\}$*, we then perform an iteration over all actions* $a \in Act$*. Assume the current partition to be* $\{S_i \mid i \in I\}$ *and let* $A = \{s \in S \mid \exists u : s \xrightarrow{a} u\}$*. Then each subset* S_i *is split into* $S \cap A$ *and* $S \backslash A$*. After having traversed all actions, the final partition will consist of the sets* S_A*. In case of the above algorithm, this iteration over* $a \in Act$ *and the construction of* A *can be done efficiently, as the list of outgoing transitions from* S *is ordered by action label. Thus, this partition refinement operation can be implemented in* $O(n_a + m_a)$*. Thus, line* 3 *has a total time complexity of* $O(n_a + n_p + m_a)$*.*

In line 4*, the incoming action transistion of each block are grouped per action label. For each block, this can be done using a bucket sort, as we assume the action labels to be consecutively numbered. Thus, time complexity for line* 4 *is* $O(m_a)$*.*

In line 5*, the variable state_to_constellation_cnt is initialized as follows: For each action label* a*, we create a new entry in the array of state_to_constellation_cnt values and assign to it the number of action transitions with that label. This can be done by iterating over all action transitions* $s \xrightarrow{a} u$ *grouped by action label and incrementing the appropriate value in the array. At the same time, we can initialize state_to_constellation_cnt_ptr to point to that array entry. Thus, time complexity for line* 5 *is* $O(m_a)$*.*

Next, we can find an upper bound for the number of iterations in the while loop in line 6*. As we restrict the choice of* B_C *by* $|B_C| \leq \frac{1}{2}|C|$ *and make* B_C *a new constellation itself, each state can only be contained in such a* B_C $\log_2(n_a)$ *times (or* $\log_2(n_p)$ *times respectively).*

As argued in the theorems above, the time complexity of lines 10-13 and 16-18 is $O($*number of incoming action transitions*$) \subseteq O(m_a)$ *and* $O($*number of incoming probabilistic transitions*$) \subseteq O(m_p)$ *if we discard the sorting penalty in* $pMark$*. So far, the total time complexity is* $O(m_a \log(n_p) + m_p \log(n_a))$*, but we still need to consider line 14, line 19 as well as the sorting penalty.*

Next, consider the moving of blocks in lines 14 and 19. According to definition 5.8 a single splitting operation requires to move each state and to group the incoming action transitions of the new block. Using bucket sort once again, grouping can be done in linear time. As we only move small blocks B' *with* $|B'| \leq \frac{1}{2}|B|$*, each state can only be involved a maximum of* $\log_2(n_a)$ *or* $\log_2(n_p)$ *times. Thus, the total time complexity for all moving operations together is* $O((m_a + n_p)\log(n_p) + (m_p + n_a)\log(n_a))$*.*

Finally, we consider the total time needed for the sorting penalty of $pMark$*. We follow the argumentation of Valmari and Franceschinis [7] to show that toal sorting time complexity is* $O(m_p \log(n_p))$*. Therefore, let* K *be the total number of sorting steps. Furthermore, let* k_i *be the number of probabilistic states in* $|_p$ *in the* $i-th$ *sorting step. Obviously,* $k_i \leq n_p$*. Now we also observe, that for a probabilistic state* $u \in \mathrm{mid_p}$ *the amount of reachable constellations increases by one. Before splitting takes place,* u *can reach the constellation* C*. But after splitting,* u

can reach B_C as well as $C \backslash B_C$ with non-zero probability. This is according to the definition of $\mathrm{mid_p}$*, stating that $0 < u[B_C] < 1$. Thus, the total number of times a probabilistic state u can be involved in splitting is bounded above by the number of probabilistic transisitons. In other words: $\sum_{i=1}^{K} k_i \leq m_p$. Using this inequality, we can find a boundary for the total time needed for sorting:*

$$O\left(\sum_{i=1}^{K} k_i \log(k_i)\right) \subseteq O\left(\sum_{i=1}^{K} k_i \log(n_p)\right) \subseteq O\left(m_p \log(n_p)\right)$$

Summing up, we find the total time complexity of algorithm 2 to be

$$O((m_a + m_p + n_p)\log(n_p) + (m_p + n_a)\log(n_a))$$

As obviously $n_p \leq m_p$, the time complexity stated in the theorem follows:

$$O((m_a + m_p)\log(n_p) + (m_p + n_a)\log(n_a))$$

The space complexity is not hard to prove. All data structures, including state_to_constellation_cnt are linear in the number of transitions or states. Thus, we end up with a space complexity of

$$O(m_a + m_p + n_a)$$

Theorem 5.5 (Time and space complexity in useful special case)
In case all action states are reachable, the time complexity of the algorithm is $O((m_a + m_p)\log n_p + m_p \log n_a))$ and the space complexity is $O\left(m_a + m_p\right)$

Proof 5.5
If all action states are reachable, then the number of probabilistic transistions m_p cannot be smaller than $n_a - 1$. Otherwise, there would be two action states without any incoming probabilistic transitions. But since only one action state can be the initial state, the other one would be unreachable.
Using inequality $m_p \geq n_a - 1$ as explained above, we can simplify the complexity boundaries from theorem 5.4 which concludes the proof.

6 Conclusion

The presented algorithm from Groote, Verduzco and de Vink (GRV) is an efficient method for calculating probabilistic bisimulation. In their paper [3] the authors compare their algorithm to earlier results from Baier, Engelen and Majster-Cederbaum (BEM) in [1]. For the purpose of this seminar report, we leave out the details of this comparison and only mention the key findings. Under certain conditions the GRV algorithm outperforms the BEM algorithm by far as can be seen in the table below. A central assumption necessary to obtain these result is, that

	GRV	BEM
Space complexity	$O(n_p)$	$O(n_a n_p \lvert Act \rvert)$
Time complexity	$O(n_p \log n_a)$	$O(n_a n_p \log n_a + n_a^2 n_p)$

the support of each distribution is limited by some constant c. This translates to the inequality

$m_p \leq cn_p$. In cases where this assumption is not fulfilled, the performance gap between both algorithms is slightly smaller. Using different benchmarks, Groote, Verduzco and de Vink were able to verify the performance difference between both algorithms in practice. See [3] for details on this discussion.

Nevertheless, there is still room for future work. As of now, the choice of the non-trivial constellation C and block B_C is non-determinstic. The authors already noticed in [3] that it is not clear yet how an optimal choice can be made with regard to time complexity. Another open question is, whether and how this algorithm can be adjusted to fit different notions of equivalence, such as probabilistic branching simulation which is presented in [6].

References

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

[2] C. Baier and J.-P. Katoen. *Principles of Model Checking*. The MIT Press, 2008.

[3] J. Groote, H. Rivera Verduzco, and E. de Vink. An efficient algorithm to determine probabilistic bisimulation. *Algorithms*, 11(9):131, Sep 2018.

[4] R. Paige and R. E. Tarjan. Three partition refinement algorithms. *SIAM Journal on Computing*, 16(6):973–989, 1987.

[5] R. Segala and N. Lynch. Probabilistic simulations for probabilistic processes. In B. Jonsson and J. Parrow, editors, *CONCUR '94: Concurrency Theory*, pages 481–496, Berlin, Heidelberg, 1994. Springer Berlin Heidelberg.

[6] A. Turrini and H. Hermanns. Polynomial time decision algorithms for probabilistic automata. *Information and Computation*, 244:134 – 171, 2015.

[7] A. Valmari and G. Franceschinis. Simple o(m logn) time markov chain lumping. In J. Esparza and R. Majumdar, editors, *Tools and Algorithms for the Construction and Analysis of Systems*, pages 38–52, Berlin, Heidelberg, 2010. Springer Berlin Heidelberg.