Approximate verification and enumeration problems

Sylvain Peyronnet

LRI, Université Paris-Sud XI, Orsay, F-91405

Michel De Rougemont

Université Paris II & LIAFA, Université Paris 7, Paris, F-75005

Yann Strozecki

LRI, Université Paris-Sud XI, Orsay, F-91405

Abstract

We study enumeration problems with probabilistic methods and apply them to verification problems. We consider the enumeration of monomials of a polynomial given as a black box, and the enumeration of discrete points which separate two polytopes in a space of dimension n, using a random walk which provides witnesses if the volume of the difference of the polytopes is large enough. We apply the first method to enumerate words which distinguish two probabilistic automata with n states and m transitions in time $O(n.m)$. We apply the second method to enumerate words which ε-distinguish two nondeterministic finite automata using an embedding in a vector space of dimension p, in time polynomial in p. We also enumerate strategies which ε-distinguish two Markov Decision Processes in time polynomial in the dimension of their statistical representation.

I. Introduction

An enumeration problem consists in generating all structures that satisfy a given property. It can be defined for any NP problem: instead of deciding if there is one correct solution among an exponential number of candidates, one should list all the solutions. In fact, enumeration is better understood as a dynamic process which produces the solutions one at a time. One wants to bound the time between two solutions, called the delay. Enumeration problems can also be defined for large objects given as a black box. On such objects, the number of solutions to enumerate is potentially infinite. In this case, we either arbitrarily restrict the solutions or sample them uniformly at random.

We study two enumeration problems, which have direct applications to verification. First, the enumeration of the monomials of a large multivariate polynomial given as a black box, *i.e.,* the polynomial can be evaluated on specific values for the variables, in one call. One of the monomials of a polynomial can be produced with a number of calls to the black box polynomial in the number of variables and the degree [1]. Moreover, if the polynomial is multilinear, the polynomial can be interpolated with a polynomial number of calls to the black box between each produced monomial [2]. Second, the enumeration of points which separate two polytopes whose difference has a large enough volume. The algorithm to solve this problem, the so-called Polytope Separator, is based on a random walk as the one used to compute the volume of a polytope [3] and is polynomial in the dimension of the space.

A fundamental problem in Model-Checking is to compare schemas, such as regular expressions, Büchi Automata on words, DTDs on trees. One may ask to enumerate all the words which distinguish two regular expressions or two Büchi Automata and similarly trees which distinguish two DTDs: they represent the counter-examples. More generally, given two formulas ψ_1 and ψ_2 in some Logic $\mathcal{L}$, we want to enumerate all the structures $\mathcal{U}$ such that ψ_1 and ψ_2 disagree on $\mathcal{U}$. As the separation may be computationally hard, we study if we can realize it with high probability. If it is still hard, we relax the exact enumeration to an approximate enumeration. We fix a distance on the structures $\mathcal{U}$ and for $0 \leq \varepsilon \leq 1$, define $\mathcal{U} \models_\varepsilon \psi$ if there exists a structure $\mathcal{U}'$, ε-close to $\mathcal{U}$ such that $\mathcal{U}' \models \psi$. The approximate ε-version is to enumerate $\mathcal{U}$ such that $\mathcal{U} \models \psi_1$ and $\mathcal{U} \not\models_\varepsilon \psi_2$ or symmetrically $\mathcal{U} \models \psi_2$ and $\mathcal{U} \not\models_\varepsilon \psi_1$.

In Probabilistic Model-Checking, a similar question can be asked. Given two probabilistic automata $\mathcal{A}_1$ and $\mathcal{A}_2$, enumerate the words w such that $Pr[w \in \mathcal{A}_1] \neq Pr[w \in \mathcal{A}_2]$. There is a deterministic polynomial algorithm to distinguish two probabilistic automata [4] and a recent more efficient probabilistic algorithm [5] based on polynomials associated to the automata. We apply our enumeration methods to these very structured multilinear polynomials and obtain probabilistic algorithms to generate one or all the words which distinguish the two automata.

It is computationally hard to separate nondeterministic automata or Markov Decision Processes (MDPs), even with probabilistic methods, unless $\mathsf{PSPACE} = \mathsf{BPP}$. We thus only solve approximate versions of these problems. In both cases, we represent the objects to compare by polytopes and apply our Polytope Separator algorithm to generate counter-examples. On nondeterministic automata, we use the word embedding introduced in [6]. We want to ε-distinguish (for the distance introduced in [7]) two MDPs with traces on the same alphabet Σ. We represent them by the k-gram (for $k = 1/\varepsilon$) of the stationary distributions of their traces, i.e. vectors of dimension $|\Sigma|^k$. We show how to find strategies which ε-distinguish the MDPs in polynomial time in the size of the MDPs and the dimension, whereas previous methods were exponential in the dimension.

Our main results on enumeration problems are:

• An improvement of the method which generates one monomial of a multilinear polynomial for some very structured polynomials (Theorem 3).

• A probabilistic algorithm to generate points in the difference of two polytopes in polynomial time in the dimension(Theorem 6).

Our main results on approximate verification are:

• A probabilistic method for enumerating words that distinguish two probabilistic automata (Theorem 4).

• A probabilistic method to enumerate words which ε-distinguish two nondeterministic automata (Theorem 7).

• A probabilistic method to enumerate strategies which ε-distinguish two MDPs (Theorem 8).

In section II, we review the enumeration of monomials of a polynomial given as a black box, and the statistical embeddings which we will use for the approximate enumeration. In section III, we show how to enumerate one or several words which distinguish two probabilistic automata. In section IV, we construct the Polytope Separator algorithm. In section V, we apply this separator to enumerate words which ε-distinguish two regular expressions and to enumerate strategies which ε-distinguish two MDPs.

II. Preliminaries

We first review the enumeration of monomials of polynomials, one of the basic technique of this paper. We then recall how certain polytopes can be associated with regular expressions in the context of approximate verification [6], and with MDPs in the context of (state, action) frequencies [7]. These representations will be used in section V.

A. Polynomials and Enumeration

A polynomial can be defined by a directed acyclic graph, whose nodes of indegree two are labeled by $+$ or $\times$ and whose nodes of indegree 1 are labeled by variables or constants. Such objects are called arithmetic circuits and one fundamental problem is to test in deterministic polynomial time if two circuits compute the same polynomial. This problem is called POLYNOMIAL IDENTITY TESTING (PIT) and is equivalent to testing if a polynomial is identically zero. PIT has been proved tractable only for restricted classes of circuits [8]. However, if we allow randomness to be used, PIT is tractable even in the more general model of polynomials given by a black box as shown in the next lemma.

Lemma 1. *[Schwarz-Zippel [9], [10]] Let P be a non zero polynomial with n variables of total degree D, if $x_1, \ldots, x_n$ are randomly chosen integers in a set S of size $\frac{D}{\epsilon}$ then the probability that $P(x_1, \ldots, x_n) = 0$ is bounded by ϵ.*

Thanks to this lemma, we can test whether two polynomials given by a black box are equal. Moreover, we can get more information from a polynomial: one of its monomial can be computed in time polynomial in the number of its variables and its degree [1], [2]. Using the algorithm computing one monomial, we can produce all monomials, a process called interpolation.

When the polynomial is multilinear, there is a faster interpolation method. We use an appropriate variables substitution, a univariate interpolation and Lemma 1 to decide if there is a monomial which contains all variables in a set L_1 and no variables in a set L_2. Then we vary the value of L_1 and L_2 to discover all monomials with a good delay as stated in the next theorem.

Theorem 1. *[Theorem 2 of [2]] Let P be a multilinear polynomial with n variables, there is a randomized algorithm with a delay polynomial in n which lists all the monomials of P.*

B. Statistical embeddings on words

For a word w, let $\mathsf{ustat}_k(w)$ be the density vector of all the $n-k+1$ subwords of length k of the word w, also called the k-gram of w or the shingles's density vector in [11]. As an example, for binary words and $k = 2$ there are 4 possible subwords of length 2, which we take in lexicographic order. For the binary word $w = 000111$, $\mathsf{ustat}(w) = (2/5, 1/5, 0, 2/5)$ as there are 2 subwords 00, 1 subword 01, no 10 subword and 2 subword 11 among the possible 5 subwords. We extend the definition of ustat to loops of length $n > k$ by considering all the n subwords of length k. This representation is useful to design Property

Testers [6] which approximately decide if two words are close or far, or if a word is close or far to a regular expression.

The *edit distance* between two words is the minimal number of insertions, deletions and substitutions of a letter required to transform one word into the other. The *edit distance with moves* (EDM) allows one additional operation: Moving one arbitrary substring to another position in a single step. Two words are ε-close if $\mathsf{dist}(w, w') = \frac{EDM(w,w')}{\max\{|w|,|w'|\}} \leq \varepsilon$. They are ε-far if they are not ε-close. The distance of a word w to a regular expression r is $\min_{w' \in r}\{\mathsf{dist}(w, w')\}$.

Note that for the 2^n binary words of length n, there are only a polynomial number of possible ustat_k vectors. An ε-Tester to decide if $w \in r$ or if w is ε-far from r uses this property as it constructs $H_r = \{\mathsf{ustat}_k(w) : \ w \in r\}$ for $k = 1/\varepsilon$, which is a union of polytopes. Consider the nondeterministic automaton A associated with the regular expression r, and A^k the automaton where a transition is made of k transitions in A. Each polytope is the convex hull of $\mathsf{ustat}_k(l)$ vectors of compatible loops l of A^m for $m \geq k$, i.e. loops which occur on the same accepting run.

As an example, let $r = (0110)^*(11)^*$, A an automaton for r, and $k = 2$. $(0110)^l$ and $(11)^l$ are the A^k-loops of r, for any l. These loops are A^k-compatible. Let $\mathsf{ustat}((0110)^l) = (\frac{l-1}{4l-1}, \frac{l}{4l-1}, \frac{l}{4l-1}, \frac{l}{4l-1})$ which converges to $s_1 = (1/4, 1/4, 1/4, 1/4)$ when $l \to \infty$ and $s_2 = \mathsf{ustat}((11)^l) = (0, 0, 0, 1)$. Then by definition, $H_r = \mathsf{Convex-Hull}(s_1, s_2)$, which is a segment. Notice that although the dimension of the s_i is large (2^k), each vector is sparse and has at most $|A|$ non zero entries.

The distance of a word w to r is approximately the L_1-distance between $\mathsf{ustat}_k(w)$ and H_r (see [6]). To compare two regular expressions r_1 and r_2, we construct the polytopes H_{r_1} and H_{r_2}. We provide in this article a way to compare these polytopes in time polynomial in the dimension, while previous techniques were exponential in the dimension. This approach also generalizes to infinite words, to context-free properties and also to unranked ordered trees.

C. Statistical analysis of MDPs

Let Σ be a finite alphabet (set of actions) and S the set of states. If S is finite, $\Delta(S)$ denotes the set of distributions over S.

Definition 1. *A Markov Decision Process (MDP) is a tuple $\mathcal{S} = (S, \Sigma, P, \Delta_0(S))$. S is a finite set of states, Σ is a set of actions, and $P : S \times \Sigma \times S \to [0; 1]$ is the transition relation. The probability to go from a state s to a state a, when an action $a \in \Sigma$ is chosen, is denoted by $P(s, a, t)$ or $P(t|s, a)$. The initial distribution is denoted by $\Delta_0(S)$.*

If there is no action a from s, $P(t|s, a) = 0$ for all $t \in S$. A *run*, on $\mathcal{S}$ is a finite or infinite alternating sequence of states and actions, which begins with a state and ends with a state. We write Ω^* for the set of finite runs, Ω for the set of infinite runs on $\mathcal{S}$. If $n \in \mathbb{N}$ and $r \in \Omega$, we write $r_{|n}$ for the sequence of the first $n-1$ state action couples in r and the n-th state in r. The *trace* $Tr(r)$ of a run r is the sequence of actions. If $n \in \mathbb{N}$, X_n and Y_n are the random variables which associate to a run r its n-th state and its n-th action. A *policy* on $\mathcal{S}$ is a function $\sigma : \Omega^* \to \Delta(\Sigma)$. A policy resolves the non determinism of the system by choosing a distribution on the set of available actions from the last state of the given run. We write HR for the set of History dependent and Randomized policies. A policy is *deterministic* when for all runs $r = (s_1, a_1, \ldots, a_{i-1}, s_i)$ on $\mathcal{S}$, $\sigma(r) \in \Sigma$. The simplest policies are *positional*, *i.e.,* only depend on the states, and $\sigma(r) = \sigma(s_i)$.

Let σ be a policy on $\mathcal{S}$, $k \in \mathbb{N}$ and $T \geq 0$. Let $\hat{y}_k^T$ be the random variable which associates to all $r \in \Omega$ the k-gram of its prefix of length T, i.e. $\hat{y}_k^T = \mathsf{ustat}_k(r_{|T}) \in [0; 1]^{(S \times \Sigma)^k}$. Given an initial distribution α, the Expected state action frequency vector $y_{\sigma,\alpha,k}^T$ is $\mathbb{E}_{\sigma,\alpha}[\hat{y}_k^T]$, i.e. the expectation of $\hat{y}_k^T$. It may converge as $T \to +\infty$, to the limit point $y_{\sigma,\alpha,k}$. The analysis of MDPs with this state action frequency vector was initiated in [12] and [13] for $k = 1$ and generalized in [7] for an arbitrary k, by introducing the new MDP $\mathcal{S}^k = (S', \Sigma, P', \alpha)$ which iterates k transitions in $\mathcal{S}$, i.e. $S' = (\prod_{i=1}^{k-1} S \times \Sigma) \times S$ and with probabilities adjusted to k transitions. Consider the set of possible $y_{\sigma,\alpha,k}$ over all the strategies in HR,

$$H_k(\alpha) = \bigcup_{\sigma \in HR} y_{\sigma,\alpha,k}$$

This set H_k is a polytope independent of the initial distribution α which has an efficient $\mathcal{H}$-representation: for each $s' \in S'$:

$$\sum_{s \in S'} \sum_{a \in \Sigma} P'(s'|s, a) \cdot y(s, a) = \sum_{a' \in \Sigma} y(s', a') \qquad (1)$$

Each hyperplane corresponds to the conservation of densities in state s', and we have $|S'|$ such equations.

We are interested in the set of possible traces of an MDP, as we want to compare two MDPs with entirely different states but with the same action set. Hence, we consider the similar vector on the traces $\hat{x}_{\sigma,\alpha,k}^T = \mathsf{ustat}_k(Tr(r_{|T})) \in [0; 1]^{(\Sigma)^k}$ and its limit $x_{\sigma,\alpha,k}$ when $T \to +\infty$. Notice that:

$$x_{\sigma,\alpha,k}[v] = \sum_{u \in (S \times \Sigma)^k \ s.t. \ Tr(u)=v} y_{\sigma,\alpha,k}[u] \qquad (2)$$

i.e., the projection vector on the actions. We are mainly interested in the projection of the polytope H_k, also independent of α, that we denote by $\pi(H_k)$ and which is defined as follows: $\pi(H_k) = \{x_{\sigma,\alpha,k}\}$

The ε-distance between two weakly communicating MDPs $\mathcal{S}_1, \mathcal{S}_2$, introduced in [7], is the the Haussdorf distance between $\pi(H_{1,k})$ and $\pi(H_{2,k})$ for $k = 1/\varepsilon$. A vector x ε-distinguishes two MDPs if it is inside one polytope and ε-far from the other one. It corresponds to strategies which separate the most the traces of the MDPs for the edit distance with moves between traces. Precisely, let $\mathsf{dist}_k(x, \mathcal{S}) = \mathrm{Inf}_{z \in \pi(H_k)} ||x - z||_1$. Then

$$\mathsf{dist}_k(\mathcal{S}_1, \mathcal{S}_2) = \max_{\substack{x_1 \in \pi(H_{1,k}) \\ x_2 \in \pi(H_{2,k})}} \{\mathsf{dist}_k(x_1, \mathcal{S}_2), \mathsf{dist}_k(x_2, \mathcal{S}_1)\}$$

We can easily compute $\mathsf{dist}_k(x, \mathcal{S})$ with a linear program while $\mathsf{dist}_k(\mathcal{S}_1, \mathcal{S}_2)$ is hard (in the dimension), even to approximate, as we could otherwise approximate the diameter which is hard [14]. Other metrics to compare probabilistic systems are related to bisimulation [15], [16], [17], the L_1 or L_2 distances between distributions, the relative entropy of two distributions (Kullback–Leibler divergence), and the $\bar{D}$ distance [18].

III. Enumeration of monomials and separation of probabilistic automata

Here, we compare two automata denoted by A and B.

Definition 2. *A probabilistic automaton is a tuple $A = (S, \Sigma, M, \alpha, \eta)$ where S is a set of n states, Σ a finite alphabet, M is a collection of transition matrices M for each letter $\sigma \in \Sigma$: $M : \Sigma \to \mathbb{R}^{n \times n}$ where each $M(\sigma)$ is a probabilistic transition matrix, α is an initial probabilistic distribution of states, η is the final vector in $\mathbb{R}^n$.*

Let $w = w_1 w_2 \dots w_k$ be a word, we let $A(w) = \alpha(\prod_{i=1\dots k} M(w_i))\eta$ denote the probability that w is accepted by A. The number of states of A and B is bounded by n and their number of transitions is bounded by m.

A. Polynomial of an automaton

In this subsection, we present a polynomial introduced in [5] and a way to evaluate it efficiently. The polynomial is associated with an automaton A and is denoted by P_A. The set of its $|\Sigma|n$ variables is $\{X_{\sigma,i}\}_{\sigma \in \Sigma, i \leq n}$. It encodes the words of size less or equal to n and their probability to be accepted by A:

$$P_A(x) = \sum_{k=0}^{n} \sum_{w \in \Sigma^k} A(w) X_{w_1,1} X_{w_2,2} \dots X_{w_k,k}.$$

The polynomial P_A has an exponential number of monomials exponential and thus seems hard to evaluate. We give an alternate form of P_A, which enables us to evaluate it in polynomial time. First, remark that:

$$\sum_{w \in \Sigma^i} A(w) X_{w,1} X_{w_2,2} \dots X_{w_i,i} = \alpha \prod_{j=1}^{i} \sum_{\sigma \in \Sigma} X_{\sigma,j} M(\sigma) \eta.$$

Indeed, expanding the product of the right expression leads to $\alpha \left(\sum_{w \in \Sigma^i} \prod_{j=1}^{i} X_{w_j,j} M(w_j) \right) \eta$. By definition $A(w) = \alpha(\prod_{i=1\dots k} M(w_i))\eta$, thus factoring the product of matrices in front of each monomial lead to the stated equality.

Therefore, P_A is equal to the following expression:

$$\alpha \left(\sum_{k=0}^{n} \prod_{j=1}^{k} \sum_{\sigma \in \Sigma} X_{\sigma,j} M(\sigma) \right) \eta$$

Since this expression involves only polynomial size sums and products it is possible to evaluate P_A efficiently.

B. Equivalence checking

Given two probabilistic automata A and B, we wish to decide if $A \equiv B$, i.e. if all words are accepted with the same probability. There is a deterministic polynomial time algorithm [4] to decide this property with a complexity in $O(n^3|\Sigma|)$. It is also possible to use the polynomial representation of A and B, to design a probabilistic algorithm to test if A and B have the same language as it has been proved in [5].

To decide whether P_A is equal to P_B is equivalent to decide $A \equiv B$. By Schwarz-Zippel Lemma 1, to decide if $P_A = P_B$, one should compare the evaluation of P_A and P_B on random integer points. The complexity of this testing procedure is equal to the one of the evaluation of P_A and P_B, which can be done very efficiently by a succession of products of a vector by the matrices representing the transitions of A and B.

Theorem 2 (In [5]). *Let A and B be two automata with at most n states and m transitions. We can decide with probability $1 - \varepsilon$ whether $A \equiv B$ in $O(nm \log(\epsilon^{-1}))$ arithmetic operations.*

If the inputs A and B are sparse, that is $m = o(|\Sigma|n^2)$, which is very common in equivalence testing, the complexity is better than the one of the best deterministic method. In fact, this probabilistic algorithm has been benchmarked on typical instances and performs better than the known deterministic methods [5].

C. Producing one counter-example

Here, we give a probabilistic algorithm which produces one word such that its probability to be accepted by A and

B is different. We call this word a counter-example and we say it separates A from B. Our algorithm is efficient: it has the same complexity as the equivalence testing and is faster than the algorithm of the next subsection which produces all counter-examples. This algorithm is related to one given in Section 2.4 of [5], but also to a procedure given in [2] which produces a monomial of any multilinear polynomial. Our aim is to find one monomial of $P_A - P_B$, since it represents a word which separates A from B.

There are two steps in the procedure. The first one is a run of the *probabilistic* equivalence testing algorithm on the automata A and B described in Section III-B. Let denote by $M(\vec{X_i})$ the matrix $\sum_{\sigma\in\Sigma} X_{\sigma,i}M(\sigma)$ whose coefficients are linear forms over $|\Sigma|$ variables. We have $P_A = \alpha_1 \left(\prod_{i=1}^{s} M_1(\vec{X_i})\right)\eta_1$ and $P_B = \alpha_2 \left(\prod_{i=1}^{s} M_2(\vec{X_i})\right)\eta_2$. From this run, we compute:

- the size s of the smallest word which separates A from B
- the set $\{v_{\sigma,i}\}_{\sigma\in\Sigma, i\leq s}$ such that $P_A(\vec{v}) \neq P_B(\vec{v})$
- the vectors $u_l = \left(\prod_{i=1}^{l} M_1(\vec{v_i})\right)\eta_1$
- the vectors $w_l = \left(\prod_{i=1}^{l} M_2(\vec{v_i})\right)\eta_2$

The second step is *deterministic*: we try to set as many variables of $P_A - P_B$ to zero, so that the remaining polynomial will be only one monomial. The algorithm begins with $P_A(\vec{v}) - P_B(\vec{v}) \neq 0$ and replaces most $v_{\sigma,i}$ by 0 without changing this property.

By definition of P_A and P_B and linearity, we can decompose them in the following sums:

$$\begin{cases} P_A(\vec{v}) = \sum_{\sigma\in\Sigma} \alpha_1 v_{\sigma,1} M_1(\sigma) u_{s-1} \\ P_B(\vec{v}) = \sum_{\sigma\in\Sigma} \alpha_2 v_{\sigma,1} M_2(\sigma) w_{s-1} \end{cases}$$

Since $P_A(\vec{v}) - P_B(\vec{v}) \neq 0$ there is a $\sigma \in \Sigma$ such that

$$\alpha_1 v_{\sigma,1} M_1(\sigma) u_{s-1} \neq \alpha_2 v_{\sigma,1} M_2(\sigma) w_{s-1}.$$

We now set $v_{\sigma',1} = 0$ for all $\sigma' \neq \sigma$. We have $P_A(\vec{v}) = \alpha_1 v_{\sigma,1} M_1(\sigma) u_{s-1}$ and $Q(\vec{v}) = \alpha_2 v_{\sigma,1} M_2(\sigma) w_{s-1}$ thus $P_A(\vec{v}) - P_B(\vec{v}) \neq 0$. Remark that if we set $X_{\sigma',1} = 0$ for all $\sigma' \neq \sigma$ in the symbolic polynomial $P_A - P_B$, all its monomials contain the variable $X_{\sigma,1}$. We apply this transformation for $i = s$ to 1: we select one σ and set to zero all $v_{\sigma',i}$ with $\sigma \neq \sigma'$, while keeping $P_A(\vec{v}) - P_B(\vec{v}) \neq 0$. It is easy to see that if we set all the corresponding variables to zero in the polynomial $P_A - P_B$, a single monomial remains and the associated word of size s separates A from B.

The results of this section are summarized as follows.

Theorem 3. *Let A and B be two probabilistic automata over the same alphabet with at most n states and m transitions. There is an algorithm which decides if A and B define the same language and if not, outputs a minimal counter-example with probability $1 - \epsilon$ in time $O(nm\log(\epsilon^{-1}))$ and in space $O(n^2)$.*

Algorithm 1: Separation of two automata

```
Input: $A$, $B$: automata; $s$: integer; $v_{\sigma,i} \in \mathbb{Q}$:
        evaluation points; $u_i$, $w_i$: vectors
Output: A monomial of $P_A - P_B$
begin
    $a = \alpha_1$; $b = \alpha_2$; $r = 1$;
    for $i = s - 1$ to $0$ do
        for $\sigma \in \Sigma$ do
            if $av_{\sigma,i}M_1(\sigma)u_i \neq bv_{\sigma,i}M_2(\sigma)w_i$ then
                $a \longleftarrow av_{\sigma,i}M_1(\sigma)$;
                $b \longleftarrow bv_{\sigma,i}M_2(\sigma)$;
                $r \longleftarrow \frac{rX_{\sigma,i}}{v_{\sigma,i}}$;
                Break ;
    return $(a\eta_1 - b\eta_2)r$
end
```

Proof: During one inner loop of Alg. 1 we do several products between a matrix and a vector in $O(m)$ arithmetic operations. There are s iterations of the inner loop with $s \leq n$. Thus the complexity of finding a separating word is dominated by the complexity of the first step: the equivalence testing. Since we have to keep in memory the vectors u_l and w_l to speed up the computation, the space used is $O(n^2)$. ∎

D. Producing all counter-examples

In some practical context it is interesting to produce many counter-examples which will be used as a test bed to separate a program from its specification.

We leverage the polynomial representation to design an algorithm which enumerates all counter-examples: the monomials of $P_A - P_B$ are the words which separates A from B and their coefficient is the difference of accepting probability for A and B. Since $P_A - P_B$ is multilinear, all its monomials can be enumerated with a delay polynomial thanks to Theorem 1. It is easy to change the definition of P_A and P_B so that they represent the words of size l accepted by A and B for any given integer l. As a consequence, we can state the following proposition.

Theorem 4. *Let A and B be two probabilistic automata with at most nm transitions. There is a probabilistic algorithm which enumerates with probability $1 - \varepsilon$ all words of size less than l which separate A and B. Moreover, the delay between the production of two counter-examples is in $O(ml^3\varepsilon^{-1})$ and the time to produce all counter-examples is linear.*

Producing all the words which separate two automata, can be seen as a way to compute the distance between them. We show below that computing such a distance or an approximation is usually hard. The enumeration may be the best way to approach this problem. Indeed the delay of the algorithm is polynomially bounded, thus any increase in computing time enables to produce more counter-examples which allow to compute a better approximation of the distance.

E. Distance between two automata

We want to compute the distance between two probabilistic automata A and B to obtain a more qualitative information than their equivalence. The maximal distance is defined as the maximum of $|A(w) - B(w)|$ over all words w. The problem to decide whether a probabilistic automata computes a word with a probability greater than some given positive rational is called the *Emptiness* problem. The emptiness problem is undecidable [19], and can be reduced to the computation of the maximal distance.

To overcome the undecidability, we have to change the distance: we consider the bounded maximal distance that is the maximum of $|A(w) - B(w)|$ over all words w of size n. The n-*Emptiness* problem is the Emptiness problem restricted to words of size n, where n is part of the instance and given in unary. Remark that the n-*Emptiness* problem can be reduced to the computation of the bounded maximal distance. Some relaxed version of the n-*Emptiness* problem is proved to be NP-hard in [7]. Hence the computation of the bounded maximal distance is hard to compute, in fact approximating this distance is still hard. It is one of the reason why we choose to relax the problem, for instance with the edit distance with moves.

The hardness result can be used to show that enumeration in some order may be hard, a result of self interest.

Proposition 1. *Let P be a multilinear polynomial given by a black box. There is no polynomial delay algorithm to produce the monomials in decreasing order of coefficient unless* $\mathsf{P} = \mathsf{NP}$.

IV. Separation of two polytopes

This section considers polytopes and their geometric difference. A polytope can be represented by a set of points, of which it is the convex hull, it is then called a $\mathcal{V}$-polytope. It can also be represented by a set of linear inequalities, it is then called a $\mathcal{H}$-polytope. In general, the number of vertices can be exponential in the number of inequalities and vice-versa. One way to abstract away the representation is to represent a polytope by a so-called strong membership oracle: the oracle is given a point and answers whether it belongs to the polytope.

From a $\mathcal{H}$-polytope or a $\mathcal{V}$-polytope, we can simulate a strong membership oracle. For a $\mathcal{V}$-polytope, defined by a set of point S, we check if the point given to the oracle is in the convex hull of S, that is the point is a convex combination of points in S. This problem can be reduced to solving a system of linear inequalities in a time polynomial in S. For a $\mathcal{H}$-polytope, defined by a system of linear inequalities, we only have to test if the input point satisfies the inequalities in time linear in the size of the system.

From an algorithmic point of view, the representation is crucial. In particular, the problem to separate two polytopes is easy for $\mathcal{H}$-polytope. Let K_1 and K_2 be two $\mathcal{H}$-polytopes represented respectively by the sets of inequalities S and $\{e_1, \dots e_m\}$. Let $\bar{e_i}$ denotes the negation of e_i. The set of inequalities $S \cup \{\bar{e_i}\}$ defines a polytope, from which a point can be find in polynomial time. Since $K_1 \setminus K_2$ is equal to the union of the points satisfying $S \cup \{\bar{e_i}\}$ for all i, we have a polynomial time algorithm to decide whether $K_1 \setminus K_2 = \emptyset$ and to produce one of its elements.

However, we need another method when the representation is different. This is the reason why we design a complex algorithm to find a point in the difference of two polytopes through a random walk. Moreover, the random walk method enables us to sample almost uniformly the difference of two polytopes. This should be seen as the best approximation to the enumeration of all points, an unfeasible task since the difference of two polytopes has an infinite number of points.

A. Hardness and relation between distances

Let $K \in \mathbb{R}^n$ be a polytope, we denote by $\mathcal{V}(K)$ the volume of the polytope. Let $d(x, y)$ be the L_1 distance on $\mathbb{R}^n$. The distance of a point x to a compact K is $d(x, K) = \min_{y \in K} d(x, y)$, and this minimum is realized by some point of K. We denote by $diam(K)$ the diameter of K, that is the largest distance between two points of K. Let K_1 and K_2 be two convex polytopes in $\mathbb{R}^n$, we consider the two following distances between these objects:

1) The Hausdorff pseudo-distance:

$$d_H(K_1, K_2) = \max_{x \in K_1} d(x, K_2)$$

We symmetrize and normalize this distance:

$$d_h(K_1, K_2) = \max\left\{\frac{d_H(K_1, K_2)}{diam(K_1)}, \frac{d_H(K_2, K_1)}{diam(K_2)}\right\}$$

2) The volume of the difference as a pseudo-distance:

$$d_{\mathrm{VOL}}(K_1, K_2) = \mathcal{V}(K_1 \setminus K_2)$$

We symmetrize and normalize this distance:

$$d_{vol}(K_1, K_2) = \max\left\{\frac{d_{\mathrm{VOL}}(K_1, K_2)}{\mathcal{V}(K_1)}, \frac{d_{\mathrm{VOL}}(K_2, K_1)}{\mathcal{V}(K_2)}\right\}$$

Remark that d_{vol} is not defined when K_1 and K_2 are of volume 0 which happens if their dimensions are lower than the dimension of the space in which they are embedded. It is always possible to assume that at least one of the polytope is of positive volume (if it is not a singleton): we compute an affine subspace generated by the points of the polytope and restrict the whole space to this subspace.

It is worth noting that these two distances are related, as the following lemma states (see the proof in appendix).

Lemma 2. *For all polytopes K_1 and K_2 such that $K_1 \cap K_2 \neq \emptyset$ we have:*

$$d_h(K_1, K_2)^n < d_{vol}(K_1, K_2)$$

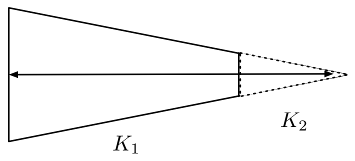

Fig. 1. The polytope K_1 in hard lines and the polytope K_2 in hard lines and dotted lines

The inequality of lemma 2 is tight, see for instance Fig. 1 in dimension two, a situation easily generalizable to any dimension. Moreover, it is not possible to bound $d_{vol}(K_1, K_2)$ by some continuous increasing function of $d_h(K_1, K_2)$. Indeed in Fig. 2, $d_{vol}(K_1, K_2) = \frac{1}{2}$ while $d_h(K_1, K_2) = \frac{1}{2l^2}$ where l can be made arbitrarily large.

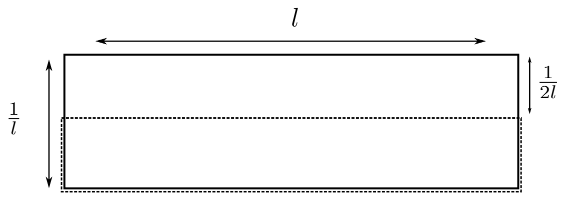

Fig. 2. The polytope K_1 in hard lines and the polytope K_2 in dotted lines

In fact, the Hausdorff distance is hard to compute, while it is possible to approximate the volume distance (if it is not too small). The proof that the Hausdorff distance is hard to approximate relies on the hardness to approximate the diameter of a polytope, as stated in theorem 5.

Theorem 5 (Theorem 1.2 of [14])**.** *The diameter of a $\mathcal{H}$-polytope is* NP*-hard to approximate within a factor polynomial in the dimension.*

In prop. 2(see the proof in appendix), we show that the Hausdorff distance and its normalized version are hard to compute. This strengthen our choice of the volume distance to design an algorithm to separate two sufficiently different polytopes.

Proposition 2. *The approximation of the functions d_H or d_h within a factor polynomial in the dimension over $\mathcal{H}$-polytopes is* NP*-hard.*

B. Sampling the difference

Our goal is to design an algorithm that provides a witness to the fact that two polytopes K_1 and K_2 are different. To do so, our algorithm is sampling points in each polytope. If $d_{vol}(K_1, K_2)$ is large enough, the algorithm samples points in the difference between K_1 and K_2 with high probability. Sampling uniformly points of a convex body is a well-studied algorithmic problem. Our algorithm is based on known results [20], [3], [21].

To sample points, we use the *Ball Walk*. The idea of this walk is to pick a uniform random point y from the ball of a given radius centered at the current point x. If y is in the polytope, we proceed from y, otherwise from x.

In order to speed up the convergence of random walks into polytopes, it is convenient to pre-process the polytopes by putting them into *quasi-isotropic position* [21]. Once this is done, sampling becomes easier. Finding the exact isotropic position is hard, so we consider algorithms for putting a convex body into nearly isotropic position (see for instance [20], [3]).

Definition 3. *Let K be a polytope with center of gravity $b(K)$. Let $0 < \gamma < 1$. K is in γ-nearly isotropic position if $||b(K)|| \leq \gamma$, and if $\forall v \in \mathbb{R}^n$, we have:*

$$(1-\gamma)||v||^2 \leq \frac{1}{vol(K)} \int_{K-b(K)} (v^\top x)^2 dx \leq (1+\gamma)||v||^2.$$

In Th. 6 of [21], it is stated that for $0 < \gamma < 1$, there is a randomized algorithm that finds an affine transformation A such that AK is γ-nearly isotropic, with probability at least $2/3$. The number of oracle calls of the algorithm is $O(n^5 |\ln \gamma| \ln n)$.

```
Algorithm 2: QISO($K, \gamma$)
  Input: $K$ : convex body ; $\gamma \in ]0, 1[$
  Output: $\gamma$-nearly isotropic version of $K$, with prob.
          at least $2/3$
```

Here, we only sketch the idea of the algorithm to put K in nearly isotropic position (we call it QISO, see Alg. 2). First, pairwise "nearly" independent points are "nearly" uniformly drawn from K. Then, by applying an affine transformation A, we bring K into nearly isotropic position. A depends mainly on the barycenter of the sampled points. The idea is to center the polytope around the origin by translating the center of gravity, but also to "round" it. If the sampling is close to the uniform distribution, then with high probability we obtain the nearly isotropic position. However, sampling pairwise "nearly" independent points is by itself a difficult task, for which a bootstrapping step is required (thus the overall complexity).

```
Algorithm 3: P-B($x, \delta$)
  Input: $x$ : point ; $\delta$ : radius
  Output: $y \in \mathbb{B}(x, \delta)$, distributed uniformly at
          random
```

Once the nearly isotropic position has been computed, we can use the Ball Walk in order to efficiently sample points uniformly at random from our polytope. To do so, we use Alg. 3 which samples a point in a ball of a given radius. Then, Alg. 4 picks at random, a point of a set given by a strong membership oracle (SMO). The parameter k in Alg. 4 is set at the mixing time of a Ball Walk, thus Alg. 4 outputs a point almost uniformly distributed in S. The parameter δ is the step size of the Ball Walk. $(K_1, K_2, \gamma, \varepsilon)$ and $(K_2, K_1, \gamma, \varepsilon)$) we find with high probability a witness that $K_1 \neq K_2$ if $d_{vol}(K_1, K_2) \geq \varepsilon$. We denote the use of Alg. 5 on $(K_1, K_2, \gamma, \varepsilon)$ and $(K_2, K_1, \gamma, \varepsilon)$, the *Polytope Separator.*

```
Algorithm 4: B-W($S, x, k, \delta$)
  Input: $S$ : set (as a strong membership oracle) ; $x$ :
         point ; $k$ : int
  Output: a point of $S$
  begin
      if $k = 0$ then
          return $x$
      $y$ =P-B($x, \delta$);
      if $y \in S$ then
          B-W($S, y, k-1, \delta$)
      B-W($S, x, k-1, \delta$)
  end
```

Theorem 6. *Let K_1 and K_2 be two polytopes, given as SMOs. For all $\varepsilon > 0$, if $d_{vol}(K_1, K_2) \geq \varepsilon$, then the* Polytope Separator *outputs a point x such that $x \in K_1 \wedge x \notin K_2$ or $x \in K_2 \wedge x \notin K_1$ with probability greater than $2/3$. Moreover, the running time of this algorithm is polynomial in n and ε^{-1}.*

Proof: Without loss of generality, we consider only the case $(K_1, K_2, \gamma, \varepsilon)$ and not the symmetric case $(K_2, K_1, \gamma, \varepsilon)$. First, we prove the correctness. Since the parameter $k = n^3 \cdot (2 + \ln(2n)) \cdot \ln(2/\varepsilon)$ is the mixing time for a Ball Walk, the algorithm outputs a point $\frac{\varepsilon}{2}$-nearly uniform in K_1. Since $d_{vol}(K_1, K_2) \geq \varepsilon$, there is a fraction ε of K_2 which is not in K_1. Thus the probability of not finding a point $x \in K_1$ such that $x \notin K_2$ after

```
Algorithm 5: E-C($J, K, \gamma, \varepsilon$)
  Input: $J$, $K$ : polytopes ; $\gamma, \varepsilon \in ]0, 1[$
  Output: $x$ such that $x \in J$ and $x \notin K$
  begin
      $k = n^3 \cdot (2 + \ln(2n)) \cdot \ln(2/\varepsilon)$;
      $J_{iso}$ =QISO($J, \gamma$);
      for $m = 1$ to $\frac{2}{\varepsilon}\ln(3)$ do
          y=B-W($J_{iso}$,P-B($0, 1$), $k$, $1/\sqrt{n}$);
          Compute $x \in J$ corresponding to $y \in J_{iso}$;
          if $x \in J$ and $x \notin K$ then
              return $x$
      return FAIL
  end
```

sampling m points is $(1 - \frac{\varepsilon}{2})^m$ (remind that the Ball Walk is $\frac{\varepsilon}{2}$-nearly uniform). By taking $p = \frac{2}{\varepsilon}\ln(3)$ the probability is upper bounded by $1/3$. So the algorithm is correct. The complexity can be decomposed as follows. To put a polytope in quasi-isotropic position, we need $O(n^5 |\ln \gamma| \ln n)$ oracle calls (see [21]). The mixing time of the Ball Walk is essentially $O(n^3)$ when $\delta = 1/\sqrt{n}$ (see [3], [22]). The Ball Walk is repeated $\frac{4}{\varepsilon}\ln(3)$ at most. Thus the complexity in terms of oracles calls is polynomial in n and ε^{-1}. The cost of an oracle call depends on the representation of the polytope, but is always polynomial. The running time is then polynomial in n and ε^{-1}. ■

V. Approximate Verification

We consider two applications of the Polytope Separator algorithm to verification.

A. Approximate separation of regular languages

Given two regular languages r_1, r_2, we want to enumerate the words which are in r_1 but not in r_2, or in r_2 but not in r_1. Since the equivalence of two regular languages is PSPACE-complete, the enumeration of one word is hard. We relax the problem: we wish to enumerate words in r_1 but ε-far from r_2, or in r_2 but ε-far from r_1 using the relative *edit distance with moves* between words.

The Equivalence testers, introduced in [6], build the polytopes associated with each regular expression and then test the equivalence by checking for all the points on the grid of step ε if they are in one of the polytopes. This procedure is exponential in the dimension. We show how to use the Polytope Separator of section IV to generate ustat vectors which separate r_1 from r_2.

Let A be an automaton with n states and M its transition matrix, *i.e.*, $M(i, j) = a$ if there is an a-transition between state i and state j. For simplicity let us assume that A is strongly connected. If it is not, we have to construct the graph of strongly connected component and to associate a polytope to each components. Let $k = 1/\varepsilon$, we consider A^k the automaton with k transitions in A. The transition matrix M^k of A^k is defined by $M^k(i,j) = \{u_1, \dots, u_p\}$ where each u_l is a word of length k such that j can be reached from i following u_l in A. We do not iterate $M^{k+1}, \dots, M^n$ since their coefficients are sets which may grow exponentially large.

Instead, we replace the words by their ustat_k. Let $U^1 = \mathsf{ustat}_k[M^k]$, *i.e.*, $U^1(i,j) = \{\mathsf{ustat}_k(u_1), \dots, \mathsf{ustat}_k(u_p)\}$. In addition to the ustat vector of a word, we need to remember its prefix w_i and suffix v_i of length $k-1$. Let p denotes the function prefix (resp. suffix s) which remove the last (resp. first) letter of a word, *i.e.*, $w_i = p(u_i)$ and $v_i = s(u_i)$. Let us define the extended U as:

$$U_e^1(i,j) = \{(\mathsf{ustat}_k(u_1), w_1, v_1), \dots (\mathsf{ustat}_k(u_p), w_p, v_p)\}$$

For $m = 1, \dots, n-k+1$, let U_e^{m+1} be the matrix such that for each pair of states (i,j), $U_e^{m+1}(i,j)$ contains the ustat_k vectors of words of length $k+m+1$ linking state i to state j. We build U_e^{m+1} from U_e^m: in each coefficient, we remove the first letter of the suffix v and add the new letter a, that is the new suffix is $v' = s(v).a$. We also modify the ustat to take into account the addition of a letter. Formally U_e^{m+1} is defined as follows:

$U_e^{m+1}(i,j) = \{(\frac{m\cdot \mathsf{ustat}_k}{m+1} + \frac{\mathsf{ustat}_k(v.a)}{m+1},\ w, v') \mid \exists l\ (\mathsf{ustat}_k, w, v) \in U_e^m(i,l),\ A(l,j) = a\}$.

When $i = j$, we reached a loop. We define H^{m+1} as the ustat_k of the loops: we adjust the ustat_k in U_e^m with the ustat_k of the k extra words. It is possible, as we kept the prefix w and the suffix v of length $k-1$.

$H^{m+1} = \{\mathsf{ustat}_k.\frac{m}{k+m+1} + \mathsf{ustat}_k(v.a).\frac{1}{k+m+1} + \mathsf{ustat}_k(a.w).\frac{1}{k+m+1} + \cdots + \mathsf{ustat}_k(s(v).a.w[1]).\frac{1}{k+m+1}$
$\mid \exists i,l\ (\mathsf{ustat}_k, w, v) \in U_e^m(i,l),\ A(l,i) = a\}$

We stop at U_e^n, and build the polytope H which is the convex hull of all the H^m for all $m \leq n$. This polytope contains all the ustat of the loops of length less than n.

Algorithm 6: Construction of the ustat polytope

```
Input: $A$: automata; $k$: integer
Output: The polytope $H$ associated with $A$
begin
    Compute $A^k$; $U_e^1$; $H^1$;
    for $m = 1$ to $n-k+1$ do
        Compute $U_e^{m+1}$
        Compute $H^m$
    return $H = Hull\{\cup_m H^m\}$
end
```

Lemma 3. *We can construct a $\mathcal{V}$-representation of H of size poly(n,k) in time poly(n,k).*

Proof: Basic loops are of length less than n, and thus appear at some stage m in U_e^m. Each entry of the matrix is a set of polynomial size, since the set of possible ustat vectors is polynomially bounded. The time to build H is polynomially bounded. ■

Theorem 7. *Given two regular expressions r_1, r_2 on words and ε, if $d_{vol}(H_{r_1}, H_{r_2}) \geq \lambda$ we can generate ε-separating words in polynomial time in the dimension and $1/\lambda$.*

Proof: Construct H_{r_1} and H_{r_2} as explained in the previous lemma. From each polytope, we build a membership oracle which takes a ustat vector x and answers YES if the L_1 distance of x to the polytope is greater than ε and NO otherwise. We apply the Polytope Separator on these two oracles. It outputs a separating ustat vector with high probability if it exists.

Given a separating ustat vector x in H_{r_1}, which is not in H_{r_2}, we can generate a large word w from x as follows: pick a starting word u of length k according to the x distribution, and let v be its suffix of length $k-1$. Then pick the next letter according to the conditional distribution $x(u|v)$, *i.e.*, the distribution of words which have v as a prefix. We repeat this process to obtain a word w of size n, for a large enough n, such that $\mathsf{ustat}_k(w)$ is ε-close to x By the completeness of the edit distance with moves [6], the word w is ε'-far from r_2. ■

Notice that the process has two probabilistic components: the random walk in the polytopes to find a separator x and then the generator to find w_n from x.

B. Approximate separation of MDPs

We want to apply the separator algorithm to ε-distinguish two MDPs on the same action set. We construct the polytopes $\pi(H_{k,1})$ and $\pi(H_{k,2})$ as defined in Section II-C. We then define an oracle which takes x, $\pi(H_{k,1})$ and ϵ and answers YES if $\mathsf{dist}(x, \pi(H_{k,1})) \leq \varepsilon$.

Let us recall how to efficiently compute $\mathsf{dist}_k(x, \mathcal{S}) = \text{Min}_{z\in\pi(H_k)}||x - z||_1$ with a linear program. Let $y \in [0;1]^{(S\times\Sigma)^k}$ and its projection and $x \in [0;1]^{(\Sigma)^k}$. Let us write $A.y = b$ for the equations (1) of section II-C and the equality $\sum_u y[u] = 1$. Let $x = C.y$ the equations (2) of section II-C and let us assume that all variables are ≥ 0 and ≤ 1.

We want to compute $\text{Min}_{z\in\pi(H_k)}||x - z||_1$ such that:

$$z = C.y$$

$$A.y = b$$

We have to consider the sum of the absolute values of $x[u] - z[u]$, so let $t[u] = |x[u] - z[u]|$ where $t \in [0;1]^{(\Sigma)^k}$ is a new vector. Then $t[u] \geq x[u] - z[u]$ and $t[u] \geq -x[u] +$

$z[u]$. If e is the vector in $[0;1]^{(\Sigma)^k}$ with all components equal to 1, we can write:

$$\mathrm{Min}_{t\in R^{(\Sigma)^k}} \quad e^t \cdot t$$
$$t \geq x - C.y$$
$$t \geq -x + C.y$$
$$A.y = b$$

The Oracle necessary for the separator algorithm takes $x, \pi(H_{k,1}), \varepsilon$ as inputs and answers YES if $\mathsf{dist}_k(x, \pi(H_{k,1}))$ computed by the above linear program is larger than ε and NO otherwise.

Theorem 8. *Given two communicating MDPs on the same action set Σ, $\pi(H_{k,1}), \pi(H_{k,2})$ and ε, if $d_{vol}(\pi(H_{k,1}), \pi(H_{k,2})) \geq \lambda$ we can generate ε-separating x vectors in polynomial time in the dimension and $1/\lambda$.*

Notice that the separator algorithm outputs a separating x, the statistics of the stationary distribution of a strategy in one of the MDP which is outside of the polytope of the other MDP. The set of saturating constraints in the linear system gives some information of the strategies whose statistics is close to x.

References

[1] A. Klivans and D. Spielman, "Randomness efficient identity testing of multivariate polynomials," in *Proceedings of the 33rd annual ACM symposium on Theory of computing*. ACM New York, NY, USA, 2001, pp. 216–223.

[2] Y. Strozecki, "Enumeration of the monomials of a polynomial and related complexity classes," *Mathematical Foundations of Computer Science*, pp. 629–640, 2010.

[3] R. Kannan, L. Lovász, and M. Simonovits, "Random walks and an o*(n5) volume algorithm for convex bodies," *Random structures and algorithms*, vol. 11, no. 1, pp. 1–50, 1997.

[4] W. Tzeng, "A polynomial-time algorithm for the equivalence of probabilistic automata," *SIAM Journal on Computing*, vol. 21, p. 216, 1992.

[5] S. Kiefer, A. Murawski, J. Ouaknine, B. Wachter, and J. Worrell, "Language equivalence for probabilistic automata," in *Computer Aided Verification*. Springer, 2011, pp. 526–540.

[6] E. Fischer, F. Magniez, and M. de Rougemont, "Approximate satisfiability and equivalence," *SIAM J. Comput.*, vol. 39, no. 6, pp. 2251–2281, 2010.

[7] M. de Rougemont and M. Tracol, "Statistic analysis for probabilistic processes," in *Proc. of the 24th Annual IEEE Symposium on Logic in Computer Science (LICS)*. IEEE Computer Society, 2009, pp. 299–308.

[8] M. Agrawal, C. Saha, R. Saptharishi, and N. Saxena, "Jacobian hits circuits: Hitting-sets, lower bounds for depth-d occur-k formulas & depth-3 transcendence degree-k circuits," *Arxiv preprint arXiv:1111.0582*, 2011.

[9] J. Schwartz, "Fast probabilistic algorithms for verification of polynomial identities," *Journal of the ACM (JACM)*, vol. 27, no. 4, p. 717, 1980.

[10] R. Zippel, "Probabilistic algorithms for sparse polynomials," *Symbolic and algebraic computation*, pp. 216–226, 1979.

[11] A. Broder, "On the resemblance and containment of documents," in *SEQUENCES '97: Proceedings of the Compression and Complexity of Sequences 1997*, 1997.

[12] C. Derman, *Finite State Markovian Decision Processes*. Academic Press, Inc. Orlando, FL, USA, 1970.

[13] M. Puterman, *Markov Decision Processes: Discrete Stochastic Dynamic Programming*. John Wiley & Sons, 1994.

[14] A. Brieden, "Geometric optimization problems likely not contained in apx," *Discrete and Computational Geometry*, vol. 28, no. 2, pp. 201–209, 2002.

[15] J. Desharnais, V. Gupta, R. Jagadeesan, and P. Panangaden, "Metrics for labelled Markov processes," *Theoretical Computer Science*, vol. 318, no. 3, pp. 323–354, 2004.

[16] F. van Breugel and J. Worrell, "Approximating and computing behavioural distances in probabilistic transition systems," *Theor. Comput. Sci.*, vol. 360, no. 1-3, pp. 373–385, 2006.

[17] C. Baier, "Polynomial time algorithms for testing probabilistic bisimulation and simulation," in *Proc. CAV'96, LNCS 1102*. Springer Verlag, 1996, pp. 38–49.

[18] D. Ornstein and B. Weiss, "How sampling reveals a process," *Annals of Probability*, vol. 18, pp. 905–930, 1990.

[19] M. Rabin, "Probabilistic automata*," *Information and control*, vol. 6, no. 3, pp. 230–245, 1963.

[20] L. Lovász and M. Simonovits, "Random walks in a convex body and an improved volume algorithm," *Random structures & algorithms*, vol. 4, no. 4, pp. 359–412, 1993.

[21] M. Simonovits, "How to compute the volume in high dimension?" *Mathematical programming*, vol. 97, no. 1, pp. 337–374, 2003.

[22] R. Kannan, L. Lovász, and R. Montenegro, "Blocking conductance and mixing in random walks," *Comb. Probab. Comput.*, vol. 15, pp. 541–570, July 2006.

Appendix

For completness, we give the proofs of Section IV that we have ommited.

Lemma. *For all polytopes K_1 and K_2 such that $K_1 \cap K_2 \neq \emptyset$ we have:*

$$d_h(K_1, K_2)^n < d_{vol}(K_1, K_2)$$

Proof: Let x be a point of K_1 such that $d(x, K_2)$ is maximal. Let K be the intersection of K_1 with the ball of radius $d_H(K_1, K_2) = d(x, K_2)$ centered in x. By construction $K \cap K_2 = \emptyset$ and $K \subseteq K_1 \setminus K_2$. Since $K_1 \cap K_2 \neq \emptyset$, the ratio $\frac{diam(K_1)}{d_H(K_1,K_2)}$ is greater or equal to 1. Moreover K_1 is a convex polytope: we can cover it by an homothetic transformation of K of center x and ratio $\frac{diam(K_1)}{d_H(K_1,K_2)}$. Therefore, we have $K_1 \subseteq \frac{diam(K_1)}{d_H(K_1,K_2)} K$ and $\mathcal{V}(K_1) \leq \left(\frac{diam(K_1)}{d_H(K_1,K_2)}\right)^n \mathcal{V}(K)$. Moreover $K \subseteq K_1 \setminus K_2$, thus we obtain $\left(\frac{d_H(K_1,K_2)}{diam(K_1)}\right)^n \leq \frac{\mathcal{V}(K_1 \setminus K_2)}{\mathcal{V}(K_1)}$. The role of K_1 and K_2 can be exchanged in the argument, thus we have proved that $d_h(K_1, K_2)^n \leq d_{vol}(K_1, K_2)$. ■

Proposition. *The approximation of the functions d_H or d_h within a factor polynomial in the dimension over $\mathcal{H}$-polytopes is* NP*-hard.*

Proof: $(\mathbf{d_H})$ Let K be a polytope and x be any point of K, which can be found in polynomial time from the linear inequalities defining K. $d_H(K, x) = \max_{y\in K} d(y, x)$ thus $d_H(K, x) \leq diam(K)$. By triangular inequality, we have that $2 \times diam(K) \leq d_H(K, x)$. Thus, computing

$d_H(K,x)$ gives a 2-approximation of the diameter, which is NP-hard to compute according to Theorem 5.

($\mathbf{d_h}$) Let K_1 be a $\mathcal{H}$-polytope defined by a set of inequalities. A linear inequality defines a halfspace and thus K_1 is the intersection of halfspaces. Modify each of the inequalities of K_1 so that the halfspaces they define are translated by ε along their normal vectors. Let K_2 be the $\mathcal{H}$-polytope defined by the new inequalities. Remark that $K_1 \subseteq K_2$ and that each point of K_2 is at distance at most εn of a point in K_1, where n is the dimension of the space. Therefore, $diam(K_1) \leq diam(K_2) \leq diam(K_1) + \varepsilon 2n$. It is possible to obtain an approximation of the diameter of ratio exponential in n. From this approximation, we can set ε to be small enough, so that $diam(K_1) \leq diam(K_2) \leq 2diam(K_1)$.

Moreover we have $\varepsilon \leq d_H(K_2, K_1) \leq n\varepsilon$. From these two inequalities we obtain: $\frac{\varepsilon}{2diam(K_1)} \leq d_h(K_1, K_2) \leq \frac{n\varepsilon}{diam(K_1)}$. And finally we have $\frac{\varepsilon}{2} d_h(K_1, K_2)^{-1} \leq diam(K_1) \leq \varepsilon n d_h(K_1, K_2)^{-1}$. If $d_h(K_1, K_2)$ could be computed within a factor polynomial in n, we would obtain an approximation of the diameter of K_1, which is NP-hard to compute according to Theorem 5. ■