Approximate verification and enumeration problems

Sylvain Peyronnet[1], Michel De Rougemont[2], and Yann Strozecki[1]

[1] LRI, Université Paris-Sud XI, Orsay, F-91405
[2] Université Paris II & LIAFA, Université Paris 7, Paris, F-75005

Abstract. We study enumeration problems using probabilistic methods, with application 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. The first method allows to enumerate all words of a given size which distinguish two probabilistic automata with a polynomial delay. The second method enumerates words which ε-distinguish two nondeterministic finite automata.We also enumerate strategies which ε-distinguish two Markov Decision Processes in time polynomial in the dimension of their statistical representation.

1 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. Enumeration is better understood as a dynamic process which produces the solutions one at a time. One wants to bound the *delay* between two solutions. Enumeration problems can also be defined for large objects given as a black box. The number of solutions to enumerate can then be infinite and we either restrict the solutions set or sample them uniformly at random.

We study two enumeration problems with 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 polynomial in the number of variables and the degree [13]. Also, if the polynomial is multilinear, the polynomial can be interpolated with a polynomial number of calls to the black box between each produced monomial [20]. Second, the enumeration of points which separate two polytopes whose difference has a large enough volume. The *Polytope Separator* algorithm solves this problem, and is based on a random walk as the one used to compute the volume of a polytope [11] and is polynomial in the dimension of the space.

In model checking, we compare schemas, such as regular expressions or Büchi automata on words. One may ask to enumerate all the words which distinguish

two regular expressions or Büchi automata: they represent the counter-examples. Given formulas ψ_1 and ψ_2 in some logic, we want to enumerate the structures $\mathcal{U}$ such that ψ_1 and ψ_2 disagree on $\mathcal{U}$. This may be computationally hard, so 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 set a distance on the structures and define, for $\varepsilon \in [0,1]$, $\mathcal{U} \models_\varepsilon \psi$ 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).

In probabilistic model checking, given two probabilistic automata $\mathcal{A}_1$ and $\mathcal{A}_2$, we want to 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 [21] and a recent more efficient probabilistic algorithm [12] based on polynomials associated to the automata. We apply enumeration methods to these structured multilinear polynomials and obtain probabilistic algorithms to generate 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 [8]. We want to ε-distinguish (for the distance introduced in [5]) 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 are probabilistics methods for:
- Enumerating efficiently points in the difference of two polytopes (Theor. 4).
- Enumerating words that ε-distinguish nondeterministic automata (Theor. 5).
- Enumerating strategies which ε-distinguish two MDPs (Theor. 6).

In section 2, we show how to enumerate words which distinguish two probabilistic automata. Section 3 presents the Polytope Separator algorithm. We apply it in section 4, to enumerate words which ε-distinguish two regular expressions and in section 5 to enumerate strategies which ε-distinguish two MDPs.

2 Enumeration of monomials and separation of probabilistic automata

2.1 Equivalence testing

In this section, we compare two probabilistic automata denoted by A and B.

Definition 1. *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.

We associate a classical polynomial P_A to the automaton 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 and thus seems hard to evaluate. We give another form of P_A, which allows to evaluate it in polynomial time expression since it involves only polynomial size sums and products:

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

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. The deterministic algorithm of [21] decides this property with 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 (see [12]). Indeed, deciding whether P_A is equal to P_B is equivalent to deciding whether $A \equiv B$. To do that, it is enough to compute $P_A - P_B$ on random integer points by the Schwarz-Zippel Lemma.

Lemma 1. *[Schwarz-Zippel [18, 22]] Let P be a nonzero polynomial with n variables of total degree D, if $x_1, \dots, x_n$ are randomly chosen integers in a set S of size $\frac{D}{\varepsilon}$ then the probability that $P(x_1, \dots, x_n) = 0$ is bounded by ε.*

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. Since the transition probabilities are in $\mathbb{R}$ we count the number of arithmetic operations in the complexity of the following algorithms. It can be turned into roughly the same boolean complexity by considering transition matrices with small rational numbers as coefficients.

Theorem 1 (In [12]). *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 $O(nm\log(\varepsilon^{-1}))$ arithmetic operations. If A and B are not equivalent, a minimal counter-example can be produced in the same time.*

The algorithm which produces a counter-example, that is a word which has not the same probability to be accepted by A and B, is given in Section 2.4 of [12]. It can also be seen as a specialization of Alg. 1 given in [20] which produces a monomial of any multilinear polynomial.

2.2 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 separate 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 polynomial delay thanks to Theorem 2 of [20]. We describe here a specialization of this algorithm to the polynomial $P_A - P_B$, which is simpler and has a better complexity. 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 theorem.

Theorem 2. *Let A and B be two probabilistic automata with at most m transitions and n states. There is a probabilistic algorithm to enumerate with probability $1-\varepsilon$ all words of size less than l which separate A and B. The delay between the production of two counter-examples is in $O(ml^3 \log(|\Sigma|\varepsilon^{-1}))$ arithmetic operations and the time to produce all of them is linear in their number.*

Proof. Let P be the polynomial $P_A - P_B$ and let $w = w_1 w_2 \dots w_k$ be a word. We denote by P_w the polynomial P where for each $i \leq k$ we have substituted 1 to X_{i,w_i} and 0 to $X_{i,\sigma}$ for $\sigma \neq w_i$. The algorithm relies on the fact, that for each $i \leq l$, a monomial of P contains exactly one of the variables of $\{X_{i,\sigma}\}_{\sigma\in\Sigma}$. Therefore we have $P_w = \sum_{\sigma\in\Sigma} X_{k+1,\sigma} P_{w\sigma}$.

Let T be the tree whose nodes are labeled by a prefix of a word which separate A from B. The children of a node labeled by w are all nodes labeled by $w\sigma$ for some $\sigma \in \Sigma$. Therefore the leaves of this trees are labeled by all the separating words. A node of label w is in T if and only if P_w is not zero, which can be tested thanks to the Schwarz-Zippel lemma. Therefore a depth-first traversal of T generates all the separating words.

Now, let study the complexity of this procedure. First, the error in the Schwarz-Zippel Lemma can be bounded by ε', if we do $\log(\varepsilon'^{-1})$ independent random evaluations of the polynomial, so that we only use random integers less than $2l$. Note that we can test whether $P_{w\sigma}$ is identically zero for each $\sigma \in \Sigma$ at once. We substitute the same random integers to the variables $\{X_{i,\sigma}\}_{i>k+1,}$ in all $P_{w\sigma}$. With probability $1-|\Sigma|\varepsilon'$, all $P_{w\sigma}$ will evaluate to some non zero value if they are not identically zero. Thanks to the particular structure of P, we can compute all $P_{w\sigma}$ on these random values in time $O(ml\log(\varepsilon'^{-1}))$.

The probabilistic test is used in the algorithm at most $|\Sigma|^l$ times which is the maximal number of leaves in T. Therefore, we must choose $\varepsilon' = \varepsilon\Sigma^{-l-1}$ so that the whole algorithm succeeds with probability $1-\varepsilon$.

When we traverse a leaf, we find a counter-example, but we still have to compute its coefficient in P that is the difference of probability to be accepted by A and B. This can be done by a single evaluation of P in $O(ml)$ operations. Finally the delay between the production of two counter-examples is bounded

by the time to visit at most $2l$ nodes since l is the depth of T, hence it is in $O(ml^3 \log(|\Sigma|\varepsilon^{-1})$ arithmetic operations.

Producing all the words which separate two automata, can be helpful to compute a distance between automata, at least when it depends on all accepted words of a given size. We show that computing such a distance or an approximation of it is usually hard. Enumeration may thus 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 in turn allow to compute a better approximation of the distance.

The maximal distance is defined as the maximum of $|A(w) - B(w)|$ over all words w. The problem to decide whether a probabilistic automaton computes a word with a probability greater than some given positive rational is called the *Emptiness* problem. The emptiness problem is undecidable [17], and can be reduced to the computation of the maximal distance. Indeed, if one wants to decide whether an automaton A accepts a word with probability larger than $q \in \mathbb{Q}$, it is equivalent to test whether the maximal distance of A and B is larger than q, where B accepts all words with probability 0.

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. Note that the *n-Emptiness* problem can be reduced to the computation of the bounded maximal distance in the same way as the unbounded version.

Some relaxed version of the *n-Emptiness* problem is proved to be NP-hard in [5]. Hence the bounded maximal distance is hard to compute, in fact approximating this distance is still hard. The hardness of the bounded maximal distance together with the representation of a probabilistic automata by a multilinear polynomial 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 $\mathsf{P} = \mathsf{NP}$.*

In this section, we have seen that we can distinguish two probabilistic automata in polynomial time, while deciding if they are far for the bounded maximal distance is hard. In the next sections, we are interested with the separation of non-deterministic automa or MDPs which is hard, and to make these problems tractable, we choose to assume some properties on distances between these objects. Moreover, the algorithms for producing one or all counter-example to the equivalence of two automata are Bellagio algorithms: They are probabilistic but they always produce the same objects in the same order (see [9]). The randomness is useful only to make them polynomially faster. The algorithms presented in the next sections rely on a random walk and by their very nature they produce counter-examples which depend on the randomness.

3 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 extremal 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 points 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. The problem to separate two polytopes is easy for $\mathcal{H}$-polytopes. 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 $\bar{e_i}$ denote the negation of e_i. The set of inequalities $S \cup \{\bar{e_i}\}$ defines a polytope, from which a point can be found 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 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.

3.1 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. 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. 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. The two distances are related, as the following lemma states (proof in appendix).

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

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

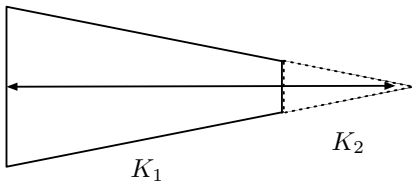

Fig. 1. The polytopes: K_1 in hard lines and K_2 in hard lines and dotted lines

The inequality is tight, see for instance Fig. 1 in dimension two, a situation easily generalizable to any dimension. Moreover, $d_{vol}(K_1, K_2)$ cannot be bound 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.

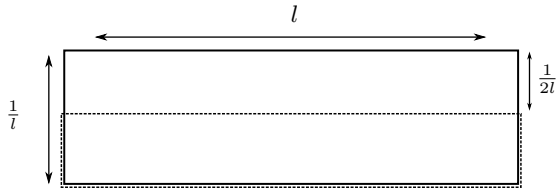

Fig. 2. The polytope K_1 in hard lines and the polytope K_2 in dotted lines

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 3.

Theorem 3 (Theorem 1.2 of [2]). *The diameter of a $\mathcal{H}$-polytope is* NP*-hard to approximate within a factor polynomial in the dimension.*

In Prop. 2 (proof in appendix), we show that the Hausdorff distance and its normalized version are hard to compute. This motivates 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.*

3.2 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 [14, 11, 19].

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 within polytopes, it is convenient to pre-process the polytopes by putting them into *quasi-isotropic position* [19]. 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 [14, 11]).

Definition 2. *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:*

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

Th. 6 of [19] states 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 $O(n^5 |\ln \gamma| \ln n)$. This algorithm to put a convex body K into γ-nearly isotropic position is called *QISO*. $QISO(K, \gamma)$ is a γ-nearly isotropic version of K, with prob. at least 2/3. We sketch the idea of this algorithm. First, pairwise "nearly" independent points are "nearly" uniformly drawn from K. Then, an affine transformation A brings 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 a difficult task, for which a bootstrapping step is required (thus the overall complexity).

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 the algorithm $P{-}B$: $P{-}B(x, \delta)$ is a random point, distributed uniformly in $\mathbb{B}(x, \delta)$ ($\delta \in \mathbb{R}$). Then, Alg. 1 picks at random a point of

Algorithm 1: B-W(S, x, k, δ)

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, δ);
  if $y \in S$ **then**
    B-W$(S, y, k-1, \delta)$
  B-W$(S, x, k-1, \delta)$
end

a set given by a strong membership oracle (SMO). The parameter k in Alg. 1 is set at the mixing time of a Ball Walk, thus Alg. 1 outputs a point almost uniformly distributed in S. The parameter δ is the step size of the Ball Walk. Finally, by using Alg. 2 twice (*i.e.,* on $(K_1, K_2, \gamma, \varepsilon)$ and $(K_2, K_1, \gamma, \varepsilon)$) we find with high probability a witness that $K_1 \neq K_2$ if $d_{vol}(K_1, K_2) \geq \varepsilon$. The use of Alg. 2 on both $(K_1, K_2, \gamma, \varepsilon)$ and $(K_2, K_1, \gamma, \varepsilon)$ is called the *Polytope Separator*.

Algorithm 2: 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, γ);
  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

Theorem 4. *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 [10] when $\delta = 1/\sqrt{n}$, 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 sampling m points is $(1 - \frac{\varepsilon}{2})^m$ (the Ball Walk is $\frac{\varepsilon}{2}$-nearly uniform with high probability). By taking $m = \frac{2}{\varepsilon}\ln(3)$ the probability of not finding a point $x \in K_1$ such that $x \notin K_2$ 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 [19]). The mixing time of the Ball Walk is essentially $O(n^3)$ when $\delta = 1/\sqrt{n}$ (see [11, 10]). 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 a call depends on the representation of the polytope, but is always polynomial. The running time is then polynomial in n and ε^{-1}.

4 Approximate separation of regular languages

We apply the Polytope Separator to a verification problem: the approximate separation of regular languages given by non deterministic regular automata.

4.1 Statistical embeddings on words

We recall how certain polytopes can be associated with regular expressions in the context of approximate verification [8]. 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 [3]. For 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 $w = 000111$, $\mathsf{ustat}_2(w) = (2/5, 1/5, 0, 2/5)$ 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 cyclic words of length $n > k$ by considering all the n subwords of length k. This representation is useful to design property testers [8] 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. More information on these distances can be found in [4]. 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 finite 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. A finite set of A^k loops is A^k compatible if all the loops can occur one after the other (in any order) in one accepting path of A^k. Each polytope is the convex hull of $\mathsf{ustat}_k(l)$ vectors of compatible loops l of A^m for $m \geq k$. The distance of a word w to r is approximately the L_1-distance between $\mathsf{ustat}_k(w)$ and H_r (see [8] for the proofs of what is stated in this paragraph).

As an example, let $r = (0110)^*(11)^*$, A an automaton for r, and $k = 2$. The A^k-loops of r are $(0110)^l$ and $(11)^l$, for any l. These loops are A^k-compatible. Let $\mathsf{ustat}_2((0110)^l) = (\frac{l-1}{4l-1}, \frac{l}{4l-1}, \frac{l}{4l-1}, \frac{l}{4l-1})$ which converges to

$s_1 = (1/4, 1/4, 1/4, 1/4)$ when $l \to \infty$ and $s_2 = \mathsf{ustat}_2((11)^l) = (0, 0, 0, 1)$. Then we have $H_r = \mathsf{Convex-Hull}(s_1, s_2)$, which is a segment. Although the dimension of the s_i's is large (2^k), each vector is sparse and has at most $|A|$ nonzero entries.

4.2 Construction and separation of the statistical polytopes

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.

To compare two regular expressions r_1 and r_2, we construct the polytopes H_{r_1} and H_{r_2}. We show how to use the Polytope Separator of section 3 to generate ustat vectors which separate r_1 from r_2. In particular the Polytope Separator has a complexity polynomial in the dimension, while previous techniques introduced in [8], were exponential in the dimension. This approach generalizes to infinite words, to context-free properties and also to unranked ordered trees.

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 components 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 $M^k(i, j) = \{u_1, \ldots, u_p\}$ 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 $M^{k+1}, \ldots, M^n$ since their coefficients are sets that may grow exponentially large. Instead, we replace the words by their ustat_k. Let $U^1 = \mathsf{ustat}_k[M^k]$, *i.e.*, $U^1(i, j) = \{\mathsf{ustat}_k(u_1), \ldots, \mathsf{ustat}_k(u_p)\}$. 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: $U_e^1(i, j) = \{(\mathsf{ustat}_k(u_1), w_1, v_1), \ldots (\mathsf{ustat}_k(u_p), w_p, v_p)\}$.

For $m = 1, \ldots, n - k + 1$, 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 seen as cyclic words: 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 $m \leq n$. H contains all the ustat of the loops of length less than n.

Algorithm 3: 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} and 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. Being of length less than n, basic loops 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 5. *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 time polynomial 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 [8], 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 from x.

5 Approximate separation of MDPs

In this section, we give a second application of the Polytope Separator algorithm to verification: the approximate separation of MDPs.

5.1 Statistical analysis of MDPs

We recall how certain polytopes can be associated with MDPs in the context of (state, action) frequencies [5]. 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 3. *A Markov Decision Process 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 state s to state t, when action $a \in \Sigma$ is chosen, is denoted $P(s,a,t)$ or $P(t|s,a)$. $\Delta_0(S)$ is the initial distribution.*

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 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.

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 $y_{\sigma,\alpha,k}$. Consider the set of possible $y_{\sigma,\alpha,k}$ over all the strategies in HR,

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

The analysis of MDPs with this state action frequency vector was initiated in [6] and [16] for $k = 1$ and generalized in [5] 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. The polytope H_k associated to $\mathcal{S}$ is equal to the polytope H_1 associated to $\mathcal{S}^k$ and they are independent of the initial distribution α. The polytope H_1 has an efficient representation by the following system of linear equalities and inequalities, for each $s' \in S'$:

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

Each equation 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 $T \to +\infty$. For all $v \in \Sigma^k$ we have:

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

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 [5], 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 [2]. Other metrics to compare probabilistic systems are related to bisimulation [7, 1], L_1 or L_2 distances between distributions, Kullback–Leibler divergence, and $\bar{D}$ distance [15].

5.2 Construction and separation of the statistical polytopes

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 previously. 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 $\mathsf{dist}_k(x, \mathcal{S}) = \mathrm{Min}_{z \in \pi(H_k)} ||x - z||_1$ with a linear program. Let $y \in [0;1]^{(S \times \Sigma)^k}$ and its projection $x \in [0;1]^{(\Sigma)^k}$. Let us write $A.y = b$ for the equations (1) of section 5.1 and the equality $\sum_u y[u] = 1$. Let $x = C.y$ the equations (2) of section 5.1 and let us assume that all variables are ≥ 0 and ≤ 1.

We want to compute $\mathrm{Min}_{z \in \pi(H_k)} ||x - z||_1$ such that $z = C.y$ and $A.y = b$.

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: $\mathrm{Min}_{t \in R^{(\Sigma)^k}} \ e^t \cdot t$ s.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 6. *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 on the strategies whose statistics are close to x.

References

1. Christel Baier. Polynomial time algorithms for testing probabilistic bisimulation and simulation. In *Proc. CAV'96, LNCS 1102*, pages 38–49. Springer Verlag, 1996.
2. A. Brieden. Geometric optimization problems likely not contained in apx. *Discrete and Computational Geometry*, 28(2):201–209, 2002.
3. A. Broder. On the resemblance and containment of documents. In *SEQUENCES '97: Proceedings of the Compression and Complexity of Sequences 1997*, 1997.
4. G. Cormode and S. Muthukrishnan. The string edit distance matching problem with moves. In *Proceedings of the ACM-SIAM symposium on Discrete algorithms*, pages 667–676. Society for Industrial and Applied Mathematics, 2002.
5. 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)*, pages 299–308. IEEE Computer Society, 2009.
6. C. Derman. *Finite State Markovian Decision Processes*. Academic Press, Inc. Orlando, FL, USA, 1970.
7. J. Desharnais, V. Gupta, R. Jagadeesan, and P. Panangaden. Metrics for labelled Markov processes. *Theoretical Computer Science*, 318(3):323–354, 2004.
8. E. Fischer, F. Magniez, and M. de Rougemont. Approximate satisfiability and equivalence. *SIAM J. Comput.*, 39(6):2251–2281, 2010.
9. Eran Gat and Shafi Goldwasser. Probabilistic search algorithms with unique answers and their cryptographic applications. *Electronic Colloquium on Computational Complexity (ECCC)*, 18:136, 2011.
10. R. Kannan, L. Lovász, and R. Montenegro. Blocking conductance and mixing in random walks. *Comb. Probab. Comput.*, 15:541–570, July 2006.
11. R. Kannan, L. Lovász, and M. Simonovits. Random walks and an o*(n5) volume algorithm for convex bodies. *Random structures and algorithms*, 11(1):1–50, 1997.
12. S. Kiefer, A. Murawski, J. Ouaknine, B. Wachter, and J. Worrell. Language equivalence for probabilistic automata. In *Computer Aided Verification*, pages 526–540. Springer, 2011.
13. A.R. Klivans and D. Spielman. Randomness efficient identity testing of multivariate polynomials. In *Proceedings of the 33rd annual ACM symposium on Theory of computing*, pages 216–223. ACM New York, NY, USA, 2001.
14. L. Lovász and M. Simonovits. Random walks in a convex body and an improved volume algorithm. *Random structures & algorithms*, 4(4):359–412, 1993.
15. D. Ornstein and B. Weiss. How sampling reveals a process. *Annals of Probability*, 18:905–930, 1990.
16. M.L. Puterman. *Markov Decision Processes: Discrete Stochastic Dynamic Programming*. John Wiley & Sons, 1994.
17. M.O. Rabin. Probabilistic automata. *Information and control*, 6(3):230–245, 1963.
18. JT Schwartz. Fast probabilistic algorithms for verification of polynomial identities. *Journal of the ACM (JACM)*, 27(4):717, 1980.
19. M. Simonovits. How to compute the volume in high dimension? *Mathematical programming*, 97(1):337–374, 2003.
20. Y. Strozecki. Enumeration of the monomials of a polynomial and related complexity classes. *Mathematical Foundations of Computer Science*, pages 629–640, 2010.
21. W.G. Tzeng. A polynomial-time algorithm for the equivalence of probabilistic automata. *SIAM Journal on Computing*, 21:216, 1992.
22. R. Zippel. Probabilistic algorithms for sparse polynomials. *Symbolic and algebraic computation*, pages 216–226, 1979.