Randomness Efficient Testing of Sparse Black Box Identities of Unbounded Degree over the Reals

Markus Bläser[1] **and Christian Engels**[2]

1 **Computer Science, Saarland University**
Postfach 151150, 66041 Saarbrücken, Germany
`mblaeser@cs.uni-saarland.de`

2 **Computer Science, Saarland University**
Postfach 151150, 66041 Saarbrücken, Germany
`engels@cs.uni-saarland.de`

Abstract

We construct a hitting set generator for sparse multivariate polynomials over the reals. The seed length of our generator is $O(\log^2(mn/\epsilon))$ where m is the number of monomials, n is number of variables, and $1-\epsilon$ is the hitting probability. The generator can be evaluated in time polynomial in $\log m$, n, and $\log 1/\epsilon$. This is the first hitting set generator whose seed length is independent of the degree of the polynomial. The seed length of the best generator so far by Klivans and Spielman [16] depends logarithmically on the degree.

From this, we get a randomized algorithm for testing sparse black box polynomial identities over the reals using $O(\log^2(mn/\epsilon))$ random bits with running time polynomial in $\log m$, n, and $\log \frac{1}{\epsilon}$.

We also design a deterministic test with running time $\tilde{O}(m^3n^3)$. Here, the $\tilde{O}$-notation suppresses polylogarithmic factors. The previously best deterministic test by Lipton and Vishnoi [18] has a running time that depends polynomially on $\log \delta$, where δ is the degree of the black box polynomial.

1998 ACM Subject Classification F. Theory of Computation

Keywords and phrases Descartes' rule of signs, polynomial identity testing, sparse polynomials, black box testing

Digital Object Identifier 10.4230/LIPIcs.STACS.2011.555

1 Introduction

Polynomial identity testing is the problem of testing if a polynomial P is equal to zero. Applications include primality testing [1] and testing if a graph has a perfect matching [21], just to mention a few. There are also results in complexity theory which use identity testing as an ingredient. This includes $\mathsf{IP} = \mathsf{PSPACE}$ [23] and the PCP theorem [5, 6].

Of course, if we are given the coefficients of the polynomial, this is an easy problem. The problem gets interesting if the polynomial is given in a compact form, either by a circuit or by a black box. Though these two variants look similar, they are of a very different nature. If we are given an arithmetic circuit C that computes the polynomial P, it is easy to *find* a point ξ with $P(\xi) \neq 0$ provided that $P \not\equiv 0$. For some constant c, the point $\xi = (2^{2^{c\cdot|C|}}, 2^{2^{2\cdot c\cdot|C|}}, \dots, 2^{2^{n\cdot c\cdot|C|}})$ is such a point: First note that the degree d of P and the size of the coefficients of P are bounded by $2^{|C|}$ and $2^{2^{|C|}}$, resp. The Kronecker substitution, which maps each variable X_i to Y^{d^i} for $1 \leq i \leq n$, is an injective mapping of the set of n-variate polynomials of degree $\leq d$ to univariate polynomials such that the degree of the univariate polynomial is bounded by d^{n+1} and the coefficients are preserved. It is easy to see that for a nonzero univariate monic polynomial, every real number larger than the absolute values of all coefficients cannot be a root. The point ξ is obtained by performing a Kronecker substitution and then plugging in a large enough number.

This point ξ is too large to be constructed explicitly in polynomial-time but it can be computed by a polynomial size circuit via repeated squaring. But we do not know how to *evaluate* the circuit at this point. This means that in the circuit model, the polynomial identity testing problem is equivalent to deciding whether a circuit that computes a number computes the value zero, see [3]. There is an efficient randomized algorithm for this problem which chooses a random prime with a polynomial number of bits and evaluates the circuit modulo this prime. In particular, if $\mathsf{RP} = \mathsf{P}$, then there is an efficient deterministic algorithm for this problem. On the other hand, derandomizing this algorithm implies circuit lower bounds [14].

In the black box model, evaluation is of course no problem at all; the black box does it for us. It is even sufficient that the black box only tells us whether the polynomial evaluated at the query point is zero or not. In the black box model, randomization is inherently needed for polynomial-time algorithms (see [16]). Why? First consider a deterministic algorithm. We claim that when the only information that we have is that the polynomial in the black box P has $\leq m$ monomials, then the algorithm has to query the black box at least m times. This is shown by an adversary argument: Every query at a point ξ is answered with zero. Each answer gives a linear equation $P(\xi) = \sum_{\mu=1}^{m} \alpha_\mu \xi_1^{e_{\mu,1}} \cdots \xi_n^{e_{\mu,n}} = 0$ on the coefficients $\alpha_1, \dots, \alpha_m$. As long as less than m queries are done, the system of equations has a nontrivial solution. So the answer to the queries can be produced by two polynomials, the zero polynomial and a nonzero polynomial given by the nontrivial solution above. If now a polynomial-time randomized algorithm would use less than $(1-\epsilon) \cdot \log m$ random bits, then derandomizing it trivially by running over all choices for the random bits will give a deterministic algorithm making less than m queries, which does not exist. (Note that our adversary argument did not make any assumption about the running time of the deterministic algorithm.)

We will focus on the problem in the black box model. Here we are given a black box which we can query at specific points. This black box will evaluate our polynomial at the points and return the value in one time step. For identity testing, it is of course sufficient to know whether the polynomial evaluates at the query point to zero or not. Since we cannot inspect the polynomial other than by querying values, we need a hitting set, that is, a set H such that for every potential polynomial P in the black box, there is a point $x \in H$ with $P(x) \neq 0$ or, stronger, for some fraction of all $x \in H$, $P(x) \neq 0$. A hitting set generator even allows us to sample from a hitting set at random. We will call the number of random bits used by a hitting set generator its seed length. For a formal definition, see Definition 2.4. The so-called Schwartz-Zippel test [22, 24] was one of the first hitting set generators (see [20] for an alternative proof over finite fields). It is designed for dense polynomials and works over arbitrary (large enough) fields. If the polynomial has degrees $\delta_1, \dots, \delta_n$ in the variables $X_1, \dots, X_n$, then the seed length of the generator is $\sum_{i=1}^{n} \lceil \log(\delta_i + 1) \rceil + n \log n$. This seed length was improved by Chen and Kao [13] to $\sum_{i=1}^{n} \lceil \log(\delta_i + 1) \rceil$. However, their test works only for integer coefficients and makes some assumptions on the size of the coefficients. So strictly speaking, their generator is incomparable. Lewin and Vadhan [17] extended the work by Chen and Kao to fields of positive characteristic. Bläser, Hardt, and Steurer [12] constructed a hitting set generator with asymptotically optimal seed length $(1 + \mathrm{o}(1)) \sum_{i=1}^{n} \log(\delta_i + 1)$. Note that a polynomial with degrees $\delta_1, \ldots, \delta_n$ can have $(\delta_1 + 1) \cdots (\delta_n + 1)$ monomials, so we get a matching lower bound on the seed length by the argument outlined above. Prior to this, Agrawal and Biswas [1] achieved the same seed length, but only if the polynomial is given by a circuit.

Klivans and Spielman [16] considered the case of sparse polynomials. This means that the number of monomials is bounded by some parameter m. For polynomials with a bound of δ on the total degree, the running time of their algorithm is polynomial in $\log m$, n, $\log \delta$, and $\log \frac{1}{\epsilon}$. The algorithm needs $\mathrm{O}(\log(mn\delta/\epsilon))$ random bits and has error probability bounded by ϵ. In this paper, we focus on sparse polynomials over the reals. Here Descartes' rule of signs says that a univariate polynomial with at most m monomials has a most $m-1$ positive real roots. This gives an efficient hitting set generator ("plug in a random integer between 1 and $2m$") for univariate polynomials with optimal seed length, which is independent of the degree. This suggests that there should also be hitting set generators for sparse multivariate polynomials over the reals that are independent of the degree. There are multivariate versions of Descartes' rule of signs like Khovanskii's theorem [15] (see also [9, 10] for improvements). However, the size of the resulting hitting set is exponential in the number of variables, which is far too large.

In this work, we give an efficient hitting set generator for sparse polynomials over the reals with running time and seed length independent of the degree of the polynomial. This results in a faster algorithm for real polynomials with high degree but few monomials. In particular, we obtain a randomized algorithm that uses $\mathrm{O}(\log^2(mn/\epsilon)) = \mathrm{O}(\log^2 m + \log^2 n + \log^2 \frac{1}{\epsilon})$ random bits, has running time polynomial in $\log m$, n, and $\log 1/\epsilon$, and error probability $\leq 1/\epsilon$. So compared to the algorithm by Klivans and Spielman, the dependence on $\log m$ and $\log n$ is slightly worse but we are completely independent of the degree δ. We also construct a deterministic algorithm with running time $\tilde{\mathrm{O}}(m^3 n^3)$. Here, the $\tilde{\mathrm{O}}$-notation suppresses polylogarithmic factors. The previously best deterministic test by Lipton and Vishnoi [18] has a running time that depends polynomially on $\log \delta$ (see also [11]).

We conclude by pointing out that the situation over the reals is a special one. Over arbitrary fields, the runtime dependence on the degree is necessary. A short example will show this. Given a field $\mathbb{F}_{p^k}$ for some prime p and a polynomial $P(X) = X^{p^{ck}} - X$, $P(X) \in \mathbb{F}_{p^k}[X]$. The polynomial is not equal to $0 \in \mathbb{F}_{p^k}[X]$; however, it evaluates to zero at every $a \in \mathbb{F}_{p^k}$ and even at every $b \in \mathbb{F}_{p^{ck}}$. Hence, the running time and number random bits will depend on the degree of the polynomial. Otherwise, we could not distinguish $P(X)$ from zero.

2 Hitting Set Generators and Transformations

▶ **Definition 2.1.** We will denote by $\mathbb{R}_m[X_1, \ldots, X_n]$ the set of all polynomials with at most m monomials in the variables $X_1, \ldots, X_n$.

We consider the polynomial identity testing problem restricted to sparse polynomials.

▶ **Definition 2.2.** PIT (n, m) is the problem of deciding if a polynomial in $\mathbb{R}_m[X_1, \ldots, X_n]$ given by a black box is identically zero.

▶ **Definition 2.3.** A set $H \subseteq \mathbb{R}^n$ is a *hitting set* for PIT (n, m) with *hitting probability* $1 - \epsilon$, if for all nonzero $P \in \mathbb{R}_m[X_1, \ldots, X_n]$,

$$\Pr_{x \in H} [P(x) \neq 0] \geq 1 - \epsilon.$$

In the definition above, x is drawn uniformly at random from H. While a hitting set is nice, we also want to be able to efficiently generate elements from the hitting set. Furthermore, the elements in the hitting set should be integers (or from any other subset from $\mathbb{R}$ that can be efficiently represented. But for our purposes, $\mathbb{Z}$ is fine.)

▶ **Definition 2.4.** A function $\mathcal{H} : \{0,1\}^\rho \to \mathbb{Z}^n$ is called a *hitting set generator* for $\mathrm{PIT}\,(n, m)$ with *hitting probability* $1 - \epsilon$ and *seed length* ρ if the image of $\mathcal{H}$ is a hitting set with hitting probability $1 - \epsilon$. We denote the image of $\mathcal{H}$, that is, the hitting set generated by $\mathcal{H}$, by $\mathrm{im}(\mathcal{H})$.

Of course, we would like to have hitting set generators for $\mathrm{PIT}\,(n, m)$ for each choice of n and m. We call this a uniform hitting set generator. It gets three inputs: n, m, and a seed r of length $\rho(n, m)$. Instead of $\mathcal{H}(n, m, r)$ we will often write $\mathcal{H}_{n,m}(r)$. If we keep n and m fixed and run over all seeds r, we get a hitting set for $\mathrm{PIT}\,(n, m)$. The evaluation time of a hitting set generator is the maximal time needed to compute $\mathcal{H}_{n,m}(r)$ for all $r \in \{0,1\}^{\rho(n,m)}$. The evaluation time is a function of n and m.

▶ **Theorem 2.5.** *If there is a uniform hitting set generator for* $\mathrm{PIT}\,(n, m)$ *with hitting probability* $1 - \epsilon > 0$*, seed length* $\rho(n, m)$*, and evaluation time* $t(n, m)$*, then there is*
1. *a deterministic algorithm for* $\mathrm{PIT}\,(n, m)$ *with running time* $\mathrm{O}(2^{\rho(n,m)} \cdot t(n, m))$ *and*
2. *a randomized algorithm for* $\mathrm{PIT}\,(n, m)$ *using* $\rho(n, m)$ *random bits with running time* $\mathrm{O}(t(n, m))$ *and error probability* $1 - \epsilon$*.*

Proof. For the deterministic algorithm, we just run over all seeds r, compute $\mathcal{H}_{n,m}(r)$ for each r and check whether the black box polynomial evaluates to zero at $\mathcal{H}_{n,m}(r)$ for all r. If not, P is of course not identically zero. Otherwise, P is identically zero by the definition of hitting set. In the randomized case, we just take a random seed r and evaluate P at $\mathcal{H}_{n,m}(r)$. ◀

▶ **Definition 2.6.** A function $T : \{0,1\}^\rho \times \mathbb{Z}^{n'} \to \mathbb{Z}^n$ is called a *hitting set transformation* from $\mathrm{PIT}\,(n', m')$ to $\mathrm{PIT}\,(n, m)$ with success probability $1 - \beta$ if for every hitting set H for $\mathrm{PIT}\,(n', m')$ with hitting probability $1 - \epsilon$,

$$\{T(r, x) \mid r \in \{0,1\}^\rho,\ \ x \in H\}$$

is a hitting set with hitting probability $\geq (1 - \beta)(1 - \epsilon)$.

A hitting set transformation transforms a hitting set for $\mathrm{PIT}\,(n', m')$ into a hitting set for $\mathrm{PIT}\,(n, m)$. It gets an additional random string r of length ρ. The size of the hitting set is extended by a factor of at most 2^ρ and we need an additional ρ random bits to draw a sample from the transformed hitting set. The parameter β measures the loss of "quality".

Of course, we again want uniform transformations, that is, T gets n and m as additional inputs and $\rho(n, m)$, $n'(n, m)$ and $m'(n, m)$ are functions depending on n and m. There is one algorithm that computes T for all choices of n and m. Again, we will write $T_{n,m}(r, x)$ instead of $T(n, m, r, x)$. For fixed n and m, T is a hitting set transformation in the sense of the definition above. For fixed n and m, the time needed to evaluate a hitting set transformation depends on the size of the elements in the hitting set we apply the transformation to. For a hitting set $H \subseteq \mathbb{Z}^{n'}$, the size of an element $x \in H$ is the bit length of an encoding of x. (We use some standard encoding here, e.g., integers are encoded by a signed binary representation, thus the size is $\log |x| + \mathrm{O}(1)$. The size of tuples of integers is the sum of the sizes of the integers in it.)

▶ **Theorem 2.7.** *Let* $\mathcal{H}_{n',m'} : \{0,1\}^{\rho'} \to \mathbb{Z}^{n'}$ *be a uniform hitting set generator for* $\mathrm{PIT}\,(n', m')$ *with hitting probability* $1 - \epsilon$ *the evaluation time of which is bounded by* $t(n', m')$*.*

Let $T_{n,m} : \{0,1\}^\rho \times \mathbb{Z}^{n'} \to \mathbb{Z}^n$ be a uniform hitting set transformation from PIT (n', m') *to* PIT (n, m) *with success probability $1 - \beta$ which can be computed in time $g(n, m, s)$ where s is the maximal size of an element in the input hitting set. Then there are*

1. *a deterministic algorithm for* PIT (n, m) *the running time of which is* $\mathrm{O}(2^{\rho(n,m)+\rho'(n',m')} \cdot (g(n, m, t(n', m')) + t(n', m')))$ *and*
2. *a randomized algorithm for* PIT (n, m) *using $\rho(n, m)+\rho'(n', m')$ random bits with running time* $\mathrm{O}(g(n, m, t(n', m')) + t(n', m'))$ *and success probability $(1 - \beta)(1 - \epsilon)$.*

Above, n' and m' are functions of n and m.

Proof. We start with the randomized algorithm. By the definition of hitting set transformation,

$$\{T_{n,m}(r, x) \mid r \in \{0,1\}^\rho,\ \ x \in \mathrm{im}(\mathcal{H}_{n',m'})\}$$

will be a hitting set for PIT (n, m) with hitting probability $(1 - \beta)(1 - \epsilon)$. To sample from this set, we choose two seeds $r \in \{0,1\}^{\rho(n,m)}$ and $r' \in \{0,1\}^{\rho'(n',m')}$ uniformly at random. We evaluate P at $T_{n,m}(r, \mathcal{H}_{n',m'}(r'))$ and claim that P is identically zero if the result is zero. Otherwise, P is obviously not identically zero. By the definition of hitting probability, this algorithm succeeds with probability $(1 - \beta)(1 - \epsilon)$.

For the running time, note that we evaluate $\mathcal{H}_{n',m'}$ once (in time $\mathrm{O}(t(n', m'))$ and $T_{n,m}$ once (in time $\mathrm{O}(g(n, m, t(n', m'))$. Note that the size of $\mathcal{H}_{n',m'}(r')$ can be at most $t(n', m')$.

We get the deterministic algorithm by derandomizing this algorithm in a straight forward manner: Just run over all seeds. ◀

Let $X_1, \dots, X_n$ and $Y_1, \dots, Y_{n'}$ be two sets of variables. A *monomial substitution* σ is a mapping that maps each X_ν to a monomial in $Y_1, \dots, Y_{n'}$. Such a substitution naturally induces a ring homomorphism, which we also call σ, from $\mathbb{R}[X_1, \dots, X_n]$ to $\mathbb{R}[Y_1, \dots, Y_{n'}]$. Since a monomial substitution cannot increase the number of monomials of a polynomial, this ring homomorphism maps polynomials in $\mathbb{R}_m[X_1, \dots, X_n]$ to polynomials in $\mathbb{R}_m[Y_1, \dots, Y_{n'}]$, too. A randomized monomial substitution gets an additional seed $r \in \{0,1\}^\rho$ and maps each X_i to a monomial $\sigma_r(X_i)$ in $Y_1, \dots, Y_{n'}$ that depends on the chosen r.

▶ **Lemma 2.8.** *If there is a randomized monomial substitution as above such that for every nonzero polynomial $P \in \mathbb{R}_m[X_1, \dots, X_n]$,*

$$\Pr_{r\in\{0,1\}^\rho}[\sigma_r(P) \not\equiv 0] \geq 1 - \beta$$

then there is a hitting set transformation from PIT (n', m) *to* PIT (n, m) *with seed length ρ and success probability $1 - \beta$.*

Proof. Let H' be a hitting set for PIT (n', m) with hitting probability $1 - \epsilon$. We claim that

$$H := \{(\sigma_r(X_1)(y), \dots, \sigma_r(X_n)(y)) \mid r \in \{0,1\}^\rho,\ \ y \in H'\}$$

is a hitting set with hitting probability $(1 - \beta)(1 - \epsilon)$. Let $P \in \mathbb{R}_m[X_1, \dots, X_n]$ be nonzero. Note that evaluating $P(x)$ for some $x \in H$ is the same as first computing $\sigma_r(P)$ and then plugging in y, where r and y are chosen such that $x = \sigma_r(X_1)(y), \dots, \sigma_r(X_n)(y)$.

With probability $\geq 1 - \beta$, $\sigma_r(P)$ is nonzero. In this case, $\sigma_r(P)(y)$ is nonzero with probability $1 - \epsilon$. Therefore, H has hitting probability $(1 - \beta)(1 - \epsilon)$. ◀

The time needed to compute the transformation depends on the degree of the monomials generated by the substitution (and on the time needed to compute the monomials, but this will be negligible in the following). If we have a uniform monomial substitution, that is, a substitution that gets n as an additional parameter and n' depends on n, then we also get a uniform hitting set transformation with $m' = m$.

3 Deterministic Algorithm

We continue with presenting our deterministic algorithm. It will have a running time of $\tilde{\mathrm{O}}\left(m^3n^3\right)$, which is independent of the degree of the polynomial. We start with a simple hitting set generator for $\mathrm{PIT}(1,m)$ and build a hitting set transformation from $\mathrm{PIT}(1,m)$ to $\mathrm{PIT}(n,m)$. The running time of the transformation will depend polynomially on m. Therefore, we only get a good deterministic algorithm with this approach but not a randomized one (which should have a running time that is polynomial in $\log m$).

Algorithm 1 Descartes Generator for $\mathrm{PIT}(1,m)$

```
1: Input: Seed $r \in \{0,1\}^{\log(m/\epsilon)}$
2: Use the seed $r$ to choose $y \in \{1,\dots,\frac{m}{\epsilon}\}$ uniformly at random.
3: Output: y
```

▶ **Theorem 3.1.** *Algorithm 1 is a uniform hitting set generator for* $\mathrm{PIT}(1,m)$ *with hitting probability* $1-\epsilon$*. The seed length is* $\log\frac{m}{\epsilon}$*, the evaluation time is* $\mathrm{O}(\log\frac{m}{\epsilon})$ *and the output size is* $\log\frac{m}{\epsilon}+\mathrm{O}(1)$*.*

Proof. A real polynomial P with m monomials can have at most $m-1$ positive real roots by Descartes' rule of signs (see e.g., [8]). Hence, the hitting probability is $\geq \frac{m/\epsilon-m}{m/\epsilon} = 1-\epsilon$. The other statements are clear from the construction. ◀

3.1 Hitting Set Transformation

Next, we design a hitting set transformation from $\mathrm{PIT}(1,m)$ to $\mathrm{PIT}(n,m)$. The construction is based on results by Klivans and Spielman [16, Section 3]. However, the construction by Klivans and Spielman depends on the degree. We refine the construction in such a way that it becomes independent of the degree.

In the following, $x \cdot y$ denotes the standard inner product $\sum_{\nu=1}^{n} x_\nu y_\nu$ of two vectors x and y in an n-dimensional vector space. Let $N = \frac{mn}{\beta}$ and q be a prime with $N < q \leq 2N$. For $1 \leq i \leq N$, let a_i denote the vector $\left(1, i \bmod q, i^2 \bmod q, \dots, i^{n-1} \bmod q\right)^T \in \mathbb{Z}_q^n$. The entries of a_i are denoted by $a_{i,1},\dots,a_{i,n}$.

▶ **Lemma 3.2.** *Let* $N = \frac{mn}{\beta}$ *and* q *be a prime such that* $N \leq q \leq 2N$*. Let* $b = (b_1,\dots,b_n)^T \in \mathbb{Z}^n$ *be a vector not equal to zero. Then* $a_i \cdot b$ *is zero for at most* $n-1$ *indices* i*.*

Proof. Let e be the largest exponent such that $q^e | b_\nu$ for every $1 \leq \nu \leq n$. Let $c_\nu = \frac{b_\nu}{q^e}$ and c be the vector $(c_1,\dots,c_n)^T$. We now reduce every entry of c modulo q and call this vector $\hat{c}$. This is a nonzero vector by the choice of e.

Let $i_1,\dots,i_n$ be n pairwise distinct indices. Consider the matrix

$$M := \begin{pmatrix} a_{i_1,1} & \cdots & a_{i_1,n} \\ \vdots & \ddots & \vdots \\ a_{i_n,1} & \cdots & a_{i_n,n} \end{pmatrix}.$$

This is a Vandermonde matrix over $\mathbb{Z}_q$. Since $i_1,\dots,i_n$ are pairwise distinct, M is invertible. Hence $M \cdot \hat{c}$ is nonzero as $\hat{c}$ is nonzero. This means that there exists an $\ell \in \{1,\dots,n\}$ such that $a_{i_\ell} \cdot \hat{c} \neq 0$ over $\mathbb{Z}_q$. Therefore $a_{i_\ell} \cdot c \neq 0$ over $\mathbb{Z}$. This implies $a_{i_\ell} \cdot b = a_{i_\ell} \cdot q^e c \neq 0$. Thus, we found at least one i, namely i_ℓ, for which $a_{i_\ell} \cdot b$ is not equal to zero. Since $i_1,\dots,i_n$ were arbitrary, the claim of the lemma follows. ◀

Algorithm 2 Transformation from PIT $(1, m)$ to PIT (n, m)

```
1: Input: Seed $r \in \{0,1\}^{\log(mn/\beta)}$, $y \in \mathbb{Z}$
2: Let $N = \frac{mn}{\beta}$ and $N < q \le 2N$ be a prime.
3: Use the seed $r$ to choose $i \in \{1, \dots, N\}$ uniformly at random.
4: Compute $y^{a_{i,\nu}}$ for every $1 \le \nu \le n$.
5: Output: $(y^{a_{i,1}}, y^{a_{i,2}}, \dots, y^{a_{i,n}})$
```

Algorithm 2 is our hitting set transformation. It implements a randomized monomial substitution: We pick one a_i uniformly at random and replace every variable X_ν in our polynomial by $Y^{a_{i,\nu}}$. Our substitution is similar to the one by Klivans and Spielman; however, our choice of the prime q is independent of δ.

▶ **Lemma 3.3** (Correctness). *Algorithm 2 is a uniform hitting set transformation from* PIT $(1, m)$ *to* PIT (n, m) *with success probability* $1 - \beta$.

Proof. Let $P = \sum_{\mu=1}^{m} \alpha_\mu X_1^{\delta_{\mu,1}} \cdots X_n^{\delta_{\mu,n}}$ be a nonzero polynomial in $\mathbb{R}_m[X_1, \dots, X_n]$. Let b_μ be the vector $(\delta_{\mu,1} - \delta_{1,1}, \dots, \delta_{\mu,n} - \delta_{1,1})^T$ for $2 \le \mu \le m$. These vectors are nonzero and there are at most $m - 1$ of these vectors.

Let us now look at $a_i \cdot b_\mu$. Lemma 3.2 tells us that for every μ at most $n - 1$ choices of i set this scalar product to zero. Therefore, for at most $(m-1)(n-1)$ indices i, there is a μ with $a_i \cdot b_\mu = 0$. There are N possible values for i.

For a randomly chosen i the probability that $a_i \cdot b_\mu \ne 0$ for all μ is at least

$$\frac{N - (m-1)(n-1)}{N} \ge 1 - \beta.$$

So with probability $\ge 1 - \beta$, $a_i \cdot b_\mu \ne 0$ for $2 \le \mu \le m$. Let us denote by δ_μ the vector $(\delta_{\mu,1}, \dots, \delta_{\mu,n})^T$. By the definition of b_μ, $a_i \cdot \delta_\mu \ne a_i \cdot \delta_1$ for $2 \le \mu \le m$. This means that our monomial $\alpha_1 X_1^{\delta_{1,1}} \cdots X_n^{\delta_{1,n}}$ is not canceled by $\alpha_\mu X_1^{\delta_{\mu,1}} \cdots X_n^{\delta_{\mu,n}}$ for every $2 \le \mu \le m$. This implies that the image $\hat{P}$ of P under the monomial substitution $X_\nu \mapsto Y^{a_{i,\nu}}$, $1 \le \nu \le n$, is nonzero with probability $1 - \beta$. Hence, the algorithm computes a randomized monomial substitution. Lemma 2.8 finishes the proof. ◀

▶ **Lemma 3.4** (Runtime and Randomness). *Algorithm 2 has a seed length of* $\log \frac{mn}{\beta}$ *and a running time of* $\mathrm{O}\left(\frac{mn}{\beta} \log^2 \frac{mn}{\beta} \log\log \frac{mn}{\beta} + \frac{mn^2}{\beta} \log y\right)$, *where y is the input.*

Proof. Let us start with the runtime. We need $\mathrm{O}(N \log^2 N \log\log N)$ bit operations to find all primes from 2 to $2N$ by the Sieve of Eratosthenes (see e.g., [7]). Hence, we need $\mathrm{O}(\frac{mn}{\beta} \log^2 \frac{mn}{\beta} \log\log \frac{mn}{\beta})$ steps for finding our prime number.

A prime q with $N < q \le 2N$ exists by Bertrand's postulate. The entries of one a_i have at most $\log(\frac{mn}{\beta}) + \mathrm{O}(1)$ bits. To compute the entries, we need to multiply two numbers of at most $\log(\frac{mn}{\beta}) + \mathrm{O}(1)$ bits because we compute modulo q. We do n multiplications of this form. Hence the computation of one vector a_i takes at most $\mathrm{O}(n \log^2 \frac{mn}{\beta})$ steps. The values $y^{a_{i,\nu}}$ are numbers with $\frac{mn}{\beta} \log y$ bits. By using repeated squaring, we can compute one of the entries in time $\mathrm{O}(\frac{mn}{\beta} \log y)$ and all values $y^{a_{i,\nu}}$, $1 \le \nu \le n$, in time $\mathrm{O}(n \cdot \frac{mn}{\beta} \log y)$.

The number of random bits used is clear from the construction. ◀

3.2 Final Algorithm

We now have all the necessary pieces to construct our deterministic algorithm.

▶ **Theorem 3.5.** *There exists a deterministic algorithm which tests if a polynomial given as a black box is equivalent to the zero polynomial in* $\tilde{\mathrm{O}}\left(m^3n^3\right)$ *steps.*

Proof. We take Algorithm 1 and Algorithm 2 to construct a deterministic algorithm for $\mathrm{PIT}\left(n,m\right)$ using Theorem 2.7. Choose arbitrary ϵ, β, both smaller than $\frac{1}{4}$. Plugging theses values into Theorem 2.7 yields an algorithm with a running time of

$$\mathrm{O}\left(2^{\log mn+\log m}\cdot\left(mn\log^2 mn\log\log mn+mn^2\log m+\log m\right)\right)=\tilde{\mathrm{O}}\left(m^3n^3\right).$$

Note that $t\left(1,m\right)$ is $\log m$. ◀

This construction gives us an efficient deterministic algorithm which has a runtime independent of the degree.

4 Randomized Algorithm

Let us continue with our randomized algorithm. Instead of starting with univariate polynomials, as in the deterministic case, we start with multivariate polynomials that have a significantly smaller number of variables, namely $\lceil\log q\rceil+1$. Note that $\log q$ is roughly $\log m+\log n$.

Our transformation from $\mathrm{PIT}\left(\lceil\log q\rceil+1,m\right)$ to $\mathrm{PIT}\left(n,m\right)$ will be very helpful as the number of random bits that the hitting set generator uses is linear in the number of variables. With the transformation mentioned above, we can efficiently reduce the number of variables our underlying hitting set generator has to work on.

Again, we divide the construction into three parts. First, we present a hitting set generator for multivariate polynomials. We continue with our transformation which uses the vectors a_i defined in the previous section. Finally, we combine the generator and the transformation yielding a hitting set generator with runtime polynomial in n, $\log m$ and $\log\frac{1}{\epsilon}$ and seed length $\mathrm{O}(\log^2\frac{mn}{\epsilon})$.

4.1 Hitting Set Generator for Multiple Variables

Our hitting set generator is a modified version of the Schwartz-Zippel generator. We adapt it to sparse polynomials over the reals in such a way that the runtime and the seed length become independent of the degree.

▶ **Lemma 4.1.** *Let* $P\in\mathbb{R}_m[X_1,\dots,X_n]$ *be a nonzero polynomial. Let* z_ν, $1\le\nu\le n$, *be drawn independently and uniformly at random from* $Z\subseteq\mathbb{Z}$ *and let* $z=(z_1,\dots,z_n)$. *Then*

$$\Pr_{z\in Z^n}\left[P\left(z\right)=0\right]\le\frac{mn}{|Z|}.$$

Proof. We prove the lemma by induction in n, in a similar fashion to the proof of the original lemma. If $n=1$ then the claim follows from Descartes' rule of signs.

If $n>1$, we can write P as $\sum_{\mu=1}^{m'}X_1^{\delta_{\mu,1}}P_\mu\left(X_2,\dots,X_n\right)$ where $m'\le m$. We look at the polynomial as a polynomial in $\mathbb{R}[X_2,\dots,X_n][X_1]$, a polynomial in X_1 with coefficients from $\mathbb{R}[X_2,\dots,X_n]$.

We have

$$\begin{aligned}\Pr_z\left[P(z)=0\right] &= \Pr_z\left[P(z_1,\dots,z_n)=0 | P_1(z_2,\dots,z_n)=0\right]\cdot\Pr_z\left[P_1(z_2,\dots,z_n)=0\right]\\ &\qquad+\Pr_z\left[P(z_1,\dots,z_n)=0 | P_1(z_2,\dots,z_n)\neq 0\right]\cdot\Pr_z\left[P_1(z_2,\dots,z_n)\neq 0\right]\\ &\leq \Pr_z\left[P_1(z_2,\dots,z_n)=0\right] \qquad (1)\\ &\qquad+\Pr_z\left[P(z_1,\dots,z_n)=0 | P_1(z_2,\dots,z_n)\neq 0\right]. \qquad (2)\end{aligned}$$

We can bound (1) using the induction hypothesis and (2) by Descartes' rule of signs, as P, after plugging in $z_2,\dots,z_n$, is a nonzero univariate polynomial with at most m monomials. This yields the bound

$$\Pr_z\left[P(z)=0\right]\leq\frac{(n-1)m}{|Z|}+\frac{m}{|Z|}=\frac{nm}{|Z|},$$

which proves the lemma. ◀

Algorithm 3 Schwartz-Zippel Hitting Set Generator

```
1: Input: Seed $r \in \{0,1\}^{n\log(mn/\epsilon)}$
2: Use the seed $r$ to choose $z \in \{1,\dots,\frac{mn}{\epsilon}\}^n$
3: Output: $z$
```

▶ **Theorem 4.2.** *Algorithm 3 is a uniform hitting set generator for* PIT (n,m) *with hitting probability* $1-\epsilon$*. The seed length is* $n\log\frac{mn}{\epsilon}$*, the evaluation time is* $\mathrm{O}\left(n\log\frac{mn}{\epsilon}\right)$ *and the output size is* $\mathrm{O}\left(n\log\frac{mn}{\epsilon}\right)$.

Proof. The running time and randomness used are clear from the construction. We can use Lemma 4.1 for bounding the error probability: For nonzero P,

$$\Pr_z\left[P(z)=0\right]\leq\frac{mn}{|Z|}$$

which in our case can be bounded by $\frac{mn}{mn/\epsilon}=\epsilon$. ◀

As it is, the Schwartz-Zippel hitting set generator is very inefficient. The number of random bits used depends linearly on the number of variables. The reason for this is that in the proof, we assume that P_1 has m monomials but also P as a univariate polynomial in X_1 has m monomials. If we knew tighter bounds then we would be able to reduce the number of random bits used. However, we cannot assume that we know such bounds in the black box model. Therefore, we will try to reduce the number of variables instead by a suitable monomial substitution.

4.2 Hitting Set Transformation

We will use our a_i as previously defined. Let $a_i=(a_{i,1},\dots,a_{i,n})$, N, and q be as in Lemma 3.2. We define $a_{i,\nu,\kappa}$ by

$$a_{i,\nu}=\sum_{\kappa=0}^{\lceil\log q\rceil}a_{i,\nu,\kappa}2^{\kappa},$$

that is, $a_{i,\nu,\kappa}$, $0\leq\kappa\leq s$ is the binary expansion of $a_{i,\nu}$. For the rest of the paper let $s=\lceil\log q\rceil$.

Algorithm 4 Transformation from PIT $(s+1, m)$ to PIT (n, m)

```
1: Input: Seed $r \in \{0,1\}^{\log(mn/\beta)}$, $y_0, \dots, y_s$
2: Let $N = \frac{mn}{\beta}$ and $N < q \leq 2N$ be a prime.
3: Use the seed $r$ to choose $i \in \{1, \dots, N\}$ uniformly at random.
4: Set $z_\nu$ to $y_0^{a_{i,\nu,0}} \cdots y_s^{a_{i,\nu,s}}$ for every $1 \leq \nu \leq n$.
5: Output: $(z_1, \dots, z_n)$
```

▶ **Lemma 4.3** (Correctness). *Algorithm 4 is a uniform hitting set transformation from* PIT $(s+1, m)$ *to* PIT (n, m) *with success probability* $1 - \beta$.

Proof. Let T be our transformation and let P be a nonzero polynomial. T essentially implements the following monomial substitution: $X_\nu \mapsto Y_0^{a_{i,\nu,0}} \cdots Y_s^{a_{i,\nu,s}}$, $1 \leq \nu \leq n$. If we now replace each Y_j by Y^{2^j}, then $X_\nu \mapsto Y^{\sum_{j=1}^{s} a_{i,\nu,j} 2^j} = Y^{a_{i,\nu}}$, $1 \leq \nu \leq n$ and we get the same substitution as used in the transformation for the deterministic algorithm in Section 4.1. Since for this combined substitution, the probability that P is mapped to zero is at most β, this has to be true for the substitution $X_\nu \mapsto Y_0^{a_{i,\nu,0}} \cdots Y_s^{a_{i,\nu,s}}$, too. Now the claim follows from Lemma 2.8. ◀

Before we prove the bounds on our transformation we need to take a short excursion on finding prime numbers. We state a proof of this well-known result for the sake of completeness.

▶ **Lemma 4.4** (Finding primes with few random bits). *We can find a prime q of size $N < q \leq 2N$ with success probability $1 - \epsilon$ in time* poly $\left(\log N, \log \frac{1}{\epsilon}\right)$ *steps using* $\mathrm{O}\left(\log N + \log \frac{1}{\epsilon}\right)$ *random bits.*

Proof. We construct $\log^2 N$ pairwise independent bit strings of length $\log N$. We can do this using only $2 \log N$ random bits as stated by Luby and Wigderson [19]. We deterministically test whether each number is prime by using the algorithm developed by Agrawal, Kayal and Saxena [2] using $\mathrm{O}(\log^6 N)$ steps. By the Chebyshev bound, one of these number is prime with probability $1 - \mathrm{o}(1)$.

We can increase the probability to $1 - \epsilon$ (think of ϵ being a small function) by doing a random walk on an expander graph which costs us $\mathrm{O}\left(\log \frac{1}{\epsilon}\right)$ extra random bits [4]. ◀

▶ **Lemma 4.5** (Runtime and Randomness). *Our hitting set transformation has a seed length of* $\mathrm{O}(\log \frac{mn}{\beta})$ *bits and a runtime polynomial in* $\log m$, n, $\log \frac{1}{\beta}$, *and* $\log y$, *where* $y = \max\{|y_0|, \dots, |y_s|\}$

Proof. If we want to have an overall error probability of β we have to set the error for the prime finding algorithm to $\beta/2$ and adjust our choice of β in the transformation to $\beta/2$.

Let us take a closer look at the runtime bounds. We need

$$\mathrm{O}\left(\log^8 \frac{2mn}{\beta} + \log \frac{2}{\beta}\right) = \mathrm{O}\left(\log^8 \frac{mn}{\beta}\right)$$

steps for finding the prime number. Computing the a_i takes us, as seen in the previous transformation, $\mathrm{O}(n \log^2 \frac{2mn}{\beta})$ steps. To calculate the z_ν, we multiply at most $s+1$ numbers with $\log y$ bits. This needs total time of $\mathrm{O}(n \cdot s \cdot \log y)$. Since $s = \mathrm{O}(\log m + \log n + \log \frac{1}{\beta})$, the total running time is $\mathrm{poly}(\log m, n, \log \frac{1}{\beta}, \log y)$.

The randomness used is clear. We need $\mathrm{O}(\log \frac{mn}{\beta})$ random bits for generating the a_i as in Lemma 3.4. Our prime finding needs $\mathrm{O}(\log \frac{mn}{\beta})$ random bits as well. ◀

4.3 Final Algorithm

Again, we combine our hitting set generator with our hitting set transformation to get our randomized algorithm.

▶ **Theorem 4.6.** *There exists a probabilistic algorithm for* PIT (n, m)*. It has success probability* $\geq 1 - \epsilon$*, uses* $\mathrm{O}(\log^2 \frac{mn}{\epsilon})$ *random bits, and has runtime polynomial in* n*,* $\log m$ *and* $\log \frac{1}{\epsilon}$*.*

Proof. We combine Algorithm 3 with Algorithm 4 and use again Theorem 2.7. We set both error parameters to be $\frac{\epsilon}{2}$. Let us look at the running time. Note that Algorithm 3 now works on $s+1$ variables which is $\lceil \log q \rceil + 1$. This is of course in $\mathrm{O}\left(\log \frac{mn}{\epsilon}\right)$. The total running time is

$$\mathrm{poly}\left(\log m, n, \log \frac{1}{\epsilon}, s, \log y\right) + \mathrm{O}(s \log \frac{ms}{\epsilon}).$$

In our case $\log y$ is at most $\log N$. This gives us a runtime bound of $\mathrm{poly}\left(\log m, n, \log \frac{1}{\epsilon}\right)$.

The algorithm uses

$$\mathrm{O}\left((s+1)\log \frac{2m\,(s+1)}{\epsilon} + \log \frac{mn}{\epsilon}\right)$$

random bits. It is easy to see that this $\mathrm{O}(\log^2 \frac{mn}{\epsilon})$, as $s = \mathrm{O}(\log \frac{mn}{\epsilon})$. ◀

Note that our algorithm also allows for a so-called time-randomness tradeoff like some of the previous tests mentioned in the introduction do. When $\epsilon > \frac{1}{mn}$, then we just run the algorithm with some constant error probability and decrease the error probability to $\frac{1}{mn}$ by doing a random walk on an expander with an additional $\mathrm{O}(\log mn)$ random bits. When $\epsilon \leq \frac{1}{mn}$ then increase the success probability by just spending more time instead of more random bits by just running over all choices for the extra $\mathrm{O}(\log^2 \frac{1}{\epsilon})$ random bits and doing a majority vote. The running time in the second case is only quasipolynomial in $\frac{1}{\epsilon}$, however.

It remains an open problem whether we can bring down the number of random bits to $\mathrm{O}(\log m)$, which would match the lower bound of $(1-\epsilon) \cdot \log m$. One approach could be to decrease the number of random bits used in the Schwartz-Zippel Hitting Set Generator. While it is clear that we do an overapproximation on the number of monomials for many polynomials in the proof, it is not clear how we can use this fact.

References

1 Manindra Agrawal and Somenath Biswas. Primality and identity testing via chinese remaindering. In *FOCS*, pages 202–209, 1999.

2 Manindra Agrawal, Neeraj Kayal, and Nitin Saxena. Primes is in P. *Ann. of Math*, 2:781–793, 2002.

3 Eric Allender, Peter Bürgisser, Johan Kjeldgaard-Pedersen, and Peter Bro Miltersen. On the complexity of numerical analysis. *SIAM J. Comput.*, 39(3):1987–2006, 2009.

4 Sanjeev Arora and Boaz Barak. *Computational Complexity: A Modern Approach.* Cambridge University Press, New York, NY, USA, 2009.

5 Sanjeev Arora, Carsten Lund, Rajeev Motwani, Madhu Sudan, and Mario Szegedy. Proof verification and the hardness of approximation problems. *J. ACM*, 45(3):501–555, 1998.

6 Sanjeev Arora and Shmuel Safra. Probabilistic checking of proofs: A new characterization of NP. *J. ACM*, 45(1):70–122, 1998.

7 Eric Bach and Jeffrey Shallit. *Algorithmic number theory. Vol. 1.* Foundations of Computing Series. MIT Press, Cambridge, MA, 1996. Efficient algorithms.

8 Saugata Basu, Richard Pollack, and Marie-Françoise Roy. *Algorithms in real algebraic geometry*, volume 10 of *Algorithms and computation in mathematics.* Springer, 2003.

9 Frédéric Bihan, J. Maurice Rojas, and Frank Sottile. On the sharpness of fewnomial bounds and the number of components of fewnomial hypersurfaces. In *Algorithms in algebraic geometry*, volume 146 of *IMA Vol. Math. Appl.*, pages 15–20. Springer, New York, 2008.

10 Frédéric Bihan and Frank Sottile. New fewnomial upper bounds from Gale dual polynomial systems. *Mosc. Math. J.*, 7(3):387–407, 573, 2007.

11 Markus Bläser, Moritz Hardt, Richard J. Lipton, and Nisheeth K. Vishnoi. Deterministically testing sparse polynomial identities of unbounded degree. *Inf. Process. Lett.*, 109(3):187–192, 2009.

12 Markus Bläser, Moritz Hardt, and David Steurer. Asymptotically optimal hitting sets against polynomials. In Luca Aceto, Ivan Damgård, Leslie Ann Goldberg, Magnús M. Halldórsson, Anna Ingólfsdóttir, and Igor Walukiewicz, editors, *ICALP (1)*, volume 5125 of *Lecture Notes in Computer Science*, pages 345–356. Springer, 2008.

13 Zhi-Zhong Chen and Ming-Yang Kao. Reducing randomness via irrational numbers. In *STOC*, pages 200–209, 1997.

14 Valentine Kabanets and Russell Impagliazzo. Derandomizing polynomial identity tests means proving circuit lower bounds. In *STOC*, pages 355–364. ACM, 2003.

15 Askold Khovanskii. *Fewnomials.* AMS press, 1991.

16 Adam Klivans and Daniel A. Spielman. Randomness efficient identity testing of multivariate polynomials. In *STOC*, pages 216–223, 2001.

17 Daniel Lewin and Salil P. Vadhan. Checking polynomial identities over any field: Towards a derandomization? In *STOC*, pages 438–447, 1998.

18 Richard J. Lipton and Nisheeth K. Vishnoi. Deterministic identity testing for multivariate polynomials. In *SODA*, pages 756–760, 2003.

19 Michael Luby and Avi Wigderson. Pairwise independence and derandomization. *Foundations and Trends in Theoretical Computer Science*, 1(4), 2005.

20 Dana Moshkovitz. An alternative proof of the Schwartz-Zippel lemma. Technical report, The Electronic Colloquium on Computational Complexity, 2010.

21 Ketan Mulmuley, Umesh V. Vazirani, and Vijay V. Vazirani. Matching is as easy as matrix inversion. *Combinatorica*, 7(1):105–113, 1987.

22 Jacob T. Schwartz. Fast probabilistic algorithms for verification of polynomial identities. *J. ACM*, 27(4):701–717, 1980.

23 Adi Shamir. IP = PSPACE. *J. ACM*, 39(4):869–877, 1992.

24 Richard Zippel. Probabilistic algorithms for sparse polynomials. In Edward W. Ng, editor, *EUROSAM*, volume 72 of *Lecture Notes in Computer Science*, pages 216–226. Springer, 1979.