A polynomial time algorithm for diophantine equations in one variable.

Felipe Cucker, Pascal Koiran, Steve Smale

Laboratoire de l'Informatique du Parallélisme

Ecole Normale Supérieure de Lyon
Unité de recherche associée au CNRS n°1398

A Polynomial Time Algorithm for Diophantine Equations in One Variable

Felipe Cucker, Pascal Koiran
and Steve Smale

November 1997

Research Report N^{o} 97-45

Ecole Normale Supérieure de Lyon
46 Allée d'Italie, 69364 Lyon Cedex 07, France
Téléphone : (+33) (0)4.72.72.80.00   Télécopieur : (+33) (0)4.72.72.80.80
Adresse électronique : lip@lip.ens−lyon.fr

A Polynomial Time Algorithm for Diophantine Equations in One Variable

Felipe Cucker, Pascal Koiran and Steve Smale

November 1997

Abstract

We show that the integer roots of of a univariate polynomial with integer coefficients can be computed in polynomial time. This result holds for the classical (i.e. Turing) model of computation and a sparse representation of polynomials (i.e. coefficients and exponents are written in binary, and only nonzero monomials are represented).

Keywords: sparse polynomials, diophantine equations, computer algebra.

Résumé

On montre que les racines entières d'un polynôme en une variable à coefficients entiers peuvent être calculées en temps polynomial. Ce résultat est valable pour le modèle de calcul classique des machines de Turing et pour une représentation creuse des polynômes (coefficients et exposants sont écrits en binaire, et seuls les monômes non nul sont représentés).

Mots-clés: polynômes creux, équations diophantiennes, calcul formel.

A Polynomial Time Algorithm for Diophantine Equations in One Variable*

Felipe Cucker, Pascal Koiran and Steve Smale

November 25, 1997

Abstract

We show that the integer roots of of a univariate polynomial with integer coefficients can be computed in polynomial time. This result holds for the classical (i.e. Turing) model of computation and a sparse representation of polynomials (i.e. coefficients and exponents are written in binary, and only nonzero monomials are represented).

Keywords: sparse polynomials, diophantine equations, computer algebra.

1 Introduction

The goal of this paper is to prove the following.

Theorem 1 *There is a polynomial time algorithm which given input $f \in \mathbb{Z}[t]$ decides whether f has an integer root and, moreover, the algorithm outputs the set of integer roots of f.*

Here we are using sparse representation of polynomials and the classical (i.e. Turing) model of computation and complexity. That is, for $f \in \mathbb{Z}[t]$,

$$f = a_d t^d + \ldots + a_1 t + a_0,$$

we encode f by the list of pairs $\{(i, a_i) \mid 0 \le i \le d \text{ and } a_i \ne 0\}$. The size of the sparse representation of f is defined by

$$size(f) = \sum_{i|a_i \ne 0} (size(a_i) + size(i)) = \sum_{i|a_i \ne 0} (ht(a_i) + ht(i))$$

*Felipe Cucker and Steve Smale are with the Mathematics Departement of the City University of Hong Kong. Most of this work was done when Pascal Koiran visited them in June 1997.

where $ht(a) = \log(1 + |a|)$ is the (logarithmic) height of an integer $a \in \mathbb{Z}$. Thus, $size(f)$ is roughly the number of bits needed to write down the list representing f. Polynomial time means that the number of bit operations to output the answer is bounded by $c(size(f))^d$ for positive constants c, d.

Note that the degree of f is at most $2^{size(f)}$ and this exponential dependence is sharp in the sense that there is no $q \in \mathbb{N}$ such that the degree of f is bounded by $(size(f))^q$ for all f. In particular, evaluating f at a given integer x may be an expensive task since the size of $f(x)$ may be exponentially large as a function of $size(f)$ and $size(x)$.

Algorithms for sparsely encoded polynomials (or just *sparse polynomials* as they are usually called) are usually much less efficient than for the standard (dense) representation in which f is represented by the list $\{a_0, a_1, \ldots, a_d\}$. This is due to the fact that some polynomials of high degree can be represented in a very compact way.

For dense polynomials, the existence of a real root can de decided efficiently (by Sturm's algorithm). It seems to be an open problem whether this can also be done in polynomial time with the sparse representation. Theorem 1 says that the existence of an *integer* root for sparse polynomials can be decided in polynomial time. In fact, all integer roots can be computed within that time bound. Our algorithm relies in particular on an efficient procedure for evaluating the sign of f at a given integer x. The (efficient) sign evaluation problem seems to be open for rational values of x.

We note here that a version of Theorem 1 is well-known for dense polynomials. For a general overview on computer algebra for one variable polynomials see [Akritas 1989; Mignotte 1992].

2 Computing signs of sparse polynomials

The main result of this section is the proof that one can evaluate the sign of a polynomial f at $x \in \mathbb{Z}$ in polynomial time. That is, given $f \in \mathbb{Z}[t]$ and $x \in \mathbb{Z}$, we can compute the quantity

$$sign\,(f(x)) = \begin{cases} -1 & \text{if } f(x) < 0 \\ 0 & \text{if } f(x) = 0 \\ 1 & \text{if } f(x) > 0 \end{cases}$$

in time polynomial in $size(x)$ and $size(f)$.

Theorem 2 *There exists an algorithm which given input $x \in \mathbb{Z}$ and $f \in \mathbb{Z}[t]$ computes the sign of $f(x)$. The halting time of this algorithm is bounded by a polynomial in $size(x)$ and $size(f)$.*

Recall that a *straight-line* program with one variable is a sequence $\mathcal{P} = \{c_1, \ldots, c_k, t, u_1, \ldots, u_\ell\}$ where $c_1, \ldots, c_k \in \mathbb{Z}$, and for $i \leq \ell$, $u_i = a * b$ with $* \in \{+, -, \times\}$ and a, b two elements in the sequence preceding u_i.

Clearly, u_ℓ may be considered as a polynomial $f(t)$; we say that $\mathcal{P}$ *computes* $f(t)$. For every polynomial $f(t)$ there exist straight-line programs computing $f(t)$. Thus, straight-line programs are regarded as yet another way to encode polynomials which turns out to be even more compact than the sparse encoding. We define the size of $\mathcal{P}$ to be

$$size(\mathcal{P}) = \ell + \sum_{i=1}^{k} size(c_i).$$

Lemma 1 *Let $\mathcal{P}$ be a straight-line program in one variable of size s computing $f(t)$ and $x \in \mathbb{Z}$ such that $0 \leq f(x) < T$ for some $T > 0$. Then $f(x)$ can be computed in time polynomially bounded in s and $size(T)$.*

PROOF. One performs the arithmetic operations (there are at most s of them) in the ring $\mathbb{Z}_T$ of integers modulo T. Each operation in this ring is done with a number of bit steps polynomial in $size(T)$.

The result, $\overline{f(x)}$, is the value of $f(x)$ modulo T and therefore, by hypothesis, the value of $f(x)$. □

Remark 1 A similar result holds if we have $-T < f(x) < 0$.

Lemma 2 *There is an algorithm which given input $(x, \alpha) \in \mathbb{Z}^2$, $x > 0$, $\alpha \geq 0$ outputs $\ell \in \mathbb{Z}$, $\ell > 0$, such that $2^{\ell-1} \leq x^\alpha \leq 2^{\ell+1}$. The halting time is bounded by a polynomial in $size(x)$ and $size(\alpha)$.*

SKETCH OF THE PROOF. Compute $\ell \in \mathbb{Z}$, $\ell > 0$ satisfying $\ell - 1 \leq \alpha \log x \leq \ell + 1$. To do so, one computes an approximation y of $\log x$ such that $|y - \log x| \leq 1/(2\alpha)$. □

PROOF OF THEOREM 2. We can assume that $x > 0$ since if $x < 0$ then $f(x)$ is $g(-x)$ where g is obtained from f by changing the sign of the coefficients of the monomials with odd degree. Also, if $x = 0$ the problem can be solved by looking at the constant term of f. Thus, suppose $x > 0$.

Let k be the number of monomials of f so that

$$f = a_1 t^{\beta_1} + \ldots + a_k t^{\beta_k} \qquad \text{with } \beta_1 > \beta_2 > \ldots > \beta_k \geq 0.$$

Then, $f(x)$ can be evaluated using Horner's rule as follows. Let $\alpha_k = \beta_k$ and $\alpha_j = \beta_j - \beta_{j+1}$ for $j = 1, \ldots, k-1$. Then, $\beta_j = \alpha_j + \alpha_{j+1} + \ldots + \alpha_k$ for $j = 1, \ldots, k$.

Now we inductively define $p_0 = 0$ and

$$s_i = p_{i-1} + a_i \quad \text{and} \quad p_i = s_i x^{\alpha_i}$$

for $i = 1, \ldots, k$. We then have $p_k = f(x)$.

The precise evaluation of $f(x)$ using the sequence of operations given by Horner's rule is not achieved in polynomial time since the intermediate results can be too large. Instead, we will inductively compute a sequence of rough approximations of s_i and p_i, with the right sign and of small (i.e. polynomially bounded) size.

More precisely, we will produce a sequence of pairs $(m_i, M_i) \in I\!N^2$ and $(v_i, V_i) \in I\!N^2$ and a sequence of integers σ_i, with $i = 1, \ldots, k$ with the following properties.

For $i = 1, \ldots, k$, $\sigma_i \in \{-1, 0, 1\}$ and

$$\begin{cases} p_i \in [2^{m_i}, 2^{M_i}] & \text{if } \sigma_i = 1 \\ p_i \in [-2^{M_i}, -2^{m_i}] & \text{if } \sigma_i = -1 \\ p_i = 0 & \text{if } \sigma_i = 0. \end{cases} \tag{1}$$

Moreover,

$$0 \leq M_i - m_i \leq 3i. \tag{2}$$

Note that, since $m_i \leq \log |p_i|$, we can write m_i with a number of bits which is polynomial in $S = \max\{size(x), size(f)\}$. The same holds for M_i since $M_i \leq m_i + 3i$.

The same properties hold for s_i and (v_i, V_i). That is, for $i = 1, \ldots, k$,

$$\begin{cases} s_i \in [2^{v_i}, 2^{V_i}] & \text{if } \sigma_i = 1 \\ s_i \in [-2^{V_i}, -2^{v_i}] & \text{if } \sigma_i = -1 \\ s_i = 0 & \text{if } \sigma_i = 0 \end{cases} \tag{3}$$

and

$$V_i - v_i \leq 3i - 2. \tag{4}$$

The general appearance of the algorithm is the following.

> For input (x, f),
> compute $\alpha_1, \ldots, \alpha_k$ as above and let $\sigma_0 = 0$.
> Then, inductively, for $i = 1, \ldots, k$
>   (a) compute v_i, V_i and σ_i from m_{i-1}, M_{i-1} and σ_{i-1}
>   (b) compute m_i and M_i from v_i, V_i and σ_i.
> Output σ_k

We will show now how steps (a) and (b) are done.

For (a), suppose that m_{i-1}, M_{i-1} and σ_{i-1} are known. Then, we compute v_i, V_i and σ_i as follows.

If $\sigma_{i-1} = 0$ then compute ℓ such that $2^\ell \leq |a_i| < 2^{\ell+1}$ and let $v_i = \ell$, $V_i = \ell + 1$ and $\sigma_i = sign\,(a_i)$.
If $\sigma_{i-1} \neq 0$ proceed as follows.
If $2^{m_{i-1}} \geq 2|a_i|$ we have two cases:
if $\sigma_{i-1}a_i > 0$ then let $v_i = m_{i-1}$ and $V_i = M_{i-1} + 1$
else, if $\sigma_{i-1}a_i < 0$, let $v_i = m_{i-1} - 1$ and $V_i = M_{i-1}$.
On the other hand, if $2^{m_{i-1}} < 2|a_i|$,
compute the exact value of p_{i-1} using Lemma 1 with $T = 2^{M_{i-1}} + 1$ and let $s_i = p_{i-1} + a_i$.
If $s_i = 0$, let $\sigma_i = 0$.
If $s_i \neq 0$ then
compute ℓ such that $2^\ell \leq |s_i| < 2^{\ell+1}$ and let $v_i = \ell$, $V_i = \ell + 1$ and $\sigma_i = sign\,(s_i)$.

It is immediate to check that, if m_{i-1}, M_{i-1} and σ_{i-1} satisfy conditions (1) and (2) then v_i, V_i and σ_i satisfy conditions (3) and (4). All lines in the above algorithm are executed in polynomial time. This is immediate except for the computation of the exact value of p_i. But the algorithm in Lemma 1 has a halting time bounded by a polynomial in $size(\mathcal{P})$ and $size(T)$ for any $\mathcal{P}$ computing $p_i(x)$. In our case one can take any straight-line program computing p_i of size polynomial in the size of f (Horner's rule as exposed above provides one with $2i-1$ operations) and we note that the size of T, is about M_{i-1}, and

$$M_{i-1} \leq m_i + 3(i-1) < \log(2|a_i|) + 3(i-1)$$

which is also polynomial in $size(f)$.

For (b), we proceed as follows.

Compute ℓ such that $2^{\ell-1} \leq x^{\alpha_i} \leq 2^{\ell+1}$ as in Lemma 2.
If $\sigma_i \neq 0$ then let $m_i = v_i + \ell - 1$ and $M_i = V_i + \ell + 1$.

Notice that in (a) we do not use the values of m_{i-1} and M_{i-1} if $\sigma_i = 0$. Consequently, we do not compute them in (b) if this is the case. $\square$

Remark 2 It is an open problem whether one can compute the sign of $f(x)$ in polynomial time if f is given as a straight-line program. This is so even allowing the use of randomization, in which case the state of the art is an algorithm for deciding whether $f(x) = 0$ in randomized (one-side error) polynomial time (see [Schwartz 1980]).

3 Proof of Theorem 1

First we give a preliminary lemma. In the sequel we count roots without multiplicity, that is, the expression "k roots" means k different roots.

Lemma 3 *Let $f \in \mathbb{R}[t]$ have k monomials. Then f has at most $2k$ real roots.*

PROOF. If $k = 1$ the statement is true. If $k > 1$ write $f = x^\alpha p$ with $p(0) \neq 0$. Then p', the derivative of p, has $k - 1$ monomials and, by induction hypothesis, at most $2(k-1)$ roots. From this we deduce that p has at most $2k - 1$ real roots and hence f has at most $2k$. □

Definition 1 Let $p \in \mathbb{Z}[t]$ and $M \in \mathbb{Z}$, $M > 0$. Let $\mathcal{C} = \{[u_i, v_i]\}_{i=1,\dots,N}$ be a list of closed intervals with integer endpoints satisfying $u_i < u_{i+1}$ and $v_i = u_i$ or $v_i = u_i + 1$ for all i. We say that $\mathcal{C}$ *locates the roots of p in* $[-M, M]$ if for each root ζ of p in $[-M, M]$ there is $i \leq N$ such that $\zeta \in [u_i, v_i]$. Note that in this case p has no roots in (v_i, u_{i+1}) for all i.

Let $g \in \mathbb{Z}[t]$ and $M \in \mathbb{Z}$, $M > 0$. Write $g = t^\alpha p$ with $p(0) \neq 0$ and suppose that $\mathcal{C}' = \{[u_i, v_i]\}_{i=1,\dots,N}$ locates the roots of p' in $[-M, M]$. Then, for each $i < N$, p has at most one root in the interval (v_i, u_{i+1}) since, by Rolle's theorem, if p has two roots in (v_i, u_{i+1}) the p' must have a root in this interval as well.

Moreover, p has a root in this interval if and only if $p(v_i)p(u_{i+1}) < 0$. This is so since if $p(v_i)p(u_{i+1}) \geq 0$ and p has some root in (v_i, u_{i+1}) then, either p has (at least) two roots in $[v_i, u_{i+1}]$ or it has a double root in (v_i, u_{i+1}). In both cases p' has a root in (v_i, u_{i+1}) contradicting the choice of $\mathcal{C}'$.

Proposition 1 *There is an algorithm which, given input $g, p \in \mathbb{Z}[t]$, M, N and $\mathcal{C}'$ as above computes a list $\mathcal{C}$ locating the roots of p in $[-M, M]$. The list $\mathcal{C}$ has at most $N + 2k$ intervals where k is the number of monomials of g. The halting time of the algorithm is polynomially bounded in size(M), size(g) and N.*

PROOF. Using the algorithm of Theorem 2 compute the sign of p at the points $-M, u_1, v_1, \dots, u_N, v_N, M$.

Let $[x, y]$ be any of the $N + 1$ intervals $[-M, u_1]$, $[v_1, u_2], \dots, [v_{N-1}, u_N]$, $[v_N, M]$. If $p(x)p(y) > 0$ we know that there are no real roots of p in $[x, y]$. Otherwise, there is only one root which can be located in an interval of the form $[u, u+1]$ by applying the classical bisection algorithm with integer mid-points (the interval has the form $[u, u]$ if we find a mid-point u such that $p(u) = 0$). We form $\mathcal{C}$ by adding to $\mathcal{C}'$ these intervals.

Since the total number of roots of p is bounded by $2k$ it follows that the number of intervals in $\mathcal{C}$ is at most $N + 2k$.

The bound for the halting time is proved as follows. Bisection is applied to $N + 1$ intervals at most. Each of these intervals has length at most $2M$ an therefore, the number of sign evaluations is of the order of $\log M$, that is, it is linear in $size(M)$. Finally, all the sign evaluations (the $2(N+1)$ first ones and the ones performed during the bisection process) are done in polynomial time in $size(M)$ and $size(g)$ by Theorem 2. $\square$

Proof of Theorem 1. Let

$$f = a_1 t^{\beta_1} + \ldots + a_k t^{\beta_k}$$

with $\beta_1 > \beta_2 > \ldots > \beta_k \geq 0$. Then, we can define polynomials p_i inductively by

$$\begin{array}{ll}
f = t^{\gamma_k} p_1 & p_1(0) \neq 0 \text{ and } p_1 \text{ has } k \text{ monomials} \\
p_1' = t^{\gamma_{k-1}} p_2 & p_2(0) \neq 0 \text{ and } p_2 \text{ has } k-1 \text{ monomials} \\
\vdots & \\
p_{k-1}' = t^{\gamma_1} p_k & p_k \in \mathbb{Z},\ p_k \neq 0
\end{array}$$

where $\gamma_k = \beta_k$ and $\gamma_1, \ldots, \gamma_{k-1}$ only depend on $\beta_1, \ldots, \beta_k$.

If L is a bound for the absolute value of the coefficients of f, the coefficients of p_j are bounded by $L\beta_1^{j-1}$ for $j = 1, \ldots, k$. Therefore, since p_j has exactly $k - j + 1$ coefficients, we deduce that

$$size(p_j) \leq (k-j+1)(j-1)size(\beta_1) + size(f)$$

which is bounded by $2(size(f))^3$ for all $j = 1, \ldots, k$.

Now we note that if ζ is an integer root of f, then either $\zeta = 0$ or ζ divides a_k. To prove this, suppose that $f(\zeta) = 0$ and $\zeta \neq 0$. Then $\beta_k = 0$, that is, $f = a_1 t^{\beta_1} + \ldots + a_{k-1} t^{\beta_{k-1}} + a_k$ and we have

$$a_1 \zeta^{\beta_1} + \ldots + a_{k-1} \zeta^{\beta_{k-1}} = -a_k.$$

Since ζ divides the left-hand side, it must divide a_k.

Thus, all integer roots of f are in the interval $[-|a_k|, |a_k|]$ and we can restrict our search to this interval.

Consider the algorithm

input f
Compute $p_1, \ldots, p_k$.
Let $\mathcal{C}_k = [0, 0]$.
For $i = k-1, \ldots, 1$, inductively
  compute $\mathcal{C}_i$ locating the roots of p_i in $[-|a_k|, |a_k|]$
  using Proposition 1 with input $\mathcal{C}_{i+1}$.
Let $\mathcal{S} = \emptyset$.
For each endpoint x of an interval in $\mathcal{C}_1$,
  if $f(x) = 0$ then let $\mathcal{S} = \mathcal{S} \cup \{x\}$.
Output $\mathcal{S}$

The list $\mathcal{C}_k$ isolates the roots of p_k. Then, by $k-1$ applications of Proposition 1, the list $\mathcal{C}_1$ isolates the roots of p_1 and since it contains the interval $[0,0]$, the roots of f. This ensures the correctness of the algorithm.

The polynomial bound for the halting time follows from Proposition 1 plus the fact that $size(p_j) \leq 2(size(f))^3$ for all $j = 1, \ldots, k$. □

4 A Refinement

Let $f = \sum_{i=0}^n a_i t^{\alpha_i}$ be an integer polynomial with $\alpha_0 < \alpha_1 < \cdots < \alpha_n$ and all a_i's nonzero. Given $k \in \{1, \ldots, n-1\}$, one can write uniquely f as $f = r_k + x^{\alpha_{k+1}} q_k$ where r_k and q_k are integer polynomials, and $\deg(r_k) = \alpha_k$ (of course, $r_k = \sum_{i=0}^k a_i t^{\alpha_i}$ and $q_k = \sum_{i=k+1}^n a_i t^{\alpha_i - \alpha_k}$). With these notations, we have the following simple fact.

Proposition 2 *Let $M_k = \sup_{0 \leq i \leq k} |a_i|$. If x is an integer root of f and $|x| \geq 2$, x must also be a root of q_k and r_k provided that $\alpha_{k+1} - \alpha_k > 1 + \log M_k$.*

PROOF. Since x is a root of f, $|r_k(x)| = |q_k(x)| \cdot |x|^{\alpha_{k+1}}$. Moreover,

$$|r_k(x)| \leq M_k(1 + |x| + \cdots + |x|^{\alpha_k}) = M_k \frac{|x|^{1+\alpha_k} - 1}{|x| - 1}.$$

¿From these two relations we obtain

$$|q_k(x)| \cdot |x|^{\alpha_{k+1} - \alpha_k} \leq M_k |x| / (|x| - 1) \leq 2M_k$$

since $|x| \geq 2$. Finally, $q_k(x) \neq 0$ implies $(\alpha_{k+1} - \alpha_k) \log |x| \leq 1 + \log M_k$ since $|q_k(x)| \geq 1$ in this case. This is in contradiction with the hypothesis $\alpha_{k+1} - \alpha_k > 1 + \log M_k$. We conclude that $q_k(x) = 0$, and $r_k(x) = 0$ follows immediately. □

This proposition applies in particular to polynomials that have a small number of terms compared to their degree (of course these are precisely the polynomials for which the sparse representation is interesting). Specifically, if f is a polynomial of degree $d = \alpha_n$ with a nonzero constant coefficient (i.e. $\alpha_0 \neq 0$) and $M = M_n = \sup_{0 \leq i \leq n} |a_i|$, there must exist a gap of at least d/n between two consecutive powers of f. Therefore one can always apply this proposition when $\frac{d}{n} > 1 + \log M$.

In any case, if the proposition applies we can first compute the integer roots of r_k (or q_k) and then check whether any of these roots is also a root of q_k (or r_k). This can sometimes speed up the algorithm described in the previous sections, in particular when either q_k or r_k is of small size compared to f. For instance, if f is of the form $f(x) = x^2 - 3 + x^5 q(x)$, only -1 and 1 can possibly be integer roots of f. And if f is of the form $f(x) = x^2 - 9 + x^7 q(x)$, all integer roots are in $\{-3, -1, 1, 3\}$.

5 Final Remarks

Natural extensions of Theorem 1 would consider the existence of rational or real roots of f. For rational roots, the arguments in Section 3 can be extended. If a rational p/q is a root of f then p divides the constant term and q divides the leading coefficient. Thus, the number of possible roots is again exponential in $size(f)$ and the bisection method applies. However, it is an open question whether one can compute the sign of $f(p/q)$ in polynomial time. For real roots the situation seems even more difficult since bisection only may not detect multiple roots.

In another direction, one could consider diophantine equations in several variables. For sparse polynomials in several variables, sign determination seems to be a difficult question, and it is not clear whether Theorem 2 can be generalized. Actually, right now it is not known whether any algorithm exists to decide diophantine equations in two variables.

Recall that the (logarithmic) height of an integer x is defined by $ht(x) = \log(1 + |x|)$.

Let $f \in \mathbb{Z}[t_1, \ldots, t_n]$, $f = \sum a_\alpha t^\alpha$. Here $a_\alpha \neq 0$, $\alpha \in \mathbb{N}^n$ is a multiindex and the sum is over a finite set $A \subset \mathbb{N}^n$. The *sparse representation* of f is the sequence of pairs (α, a_α), and the size of f for this representation is defined by

$$size(f) = \sum_{\alpha \in A} (ht(\alpha) + ht(a_\alpha))$$

where $ht(\alpha) = ht(\alpha_1) + \ldots + ht(\alpha_n)$.

It is well known that f can be evaluated at a point $x \in \mathbb{Z}^n$ in time polynomial in $size(f)$ and $size(x)$ if f is considered with the dense representation.

Problem 1 *Given $f \in \mathbb{Z}[t_1, \ldots, t_n]$ and $x \in \mathbb{Z}^n$, is it possible to compute sign$(f(x))$ in polynomial time in size(x) and size(f) for the sparse representation of f?*

Theorem 2 solves this problem for the case $n = 1$. For any fixed n, Shub [1993] solves it using Baker's theorem [Baker 1975] in case f has only two monomials (but the halting time depends exponentially in n). Moreover he poses a question akin to Problem 1.

Worse, the problem of deciding feasibility of diophantine equations in many variables is well-known to be undecidable (cf. [Matiyasevich 1993]). Thus we consider the 2-variable case. Since this problem looks much harder than in one variable, we would be happy with a single exponential algorithm for dense polynomials. If $f \in \mathbb{Z}[t_1, \ldots, t_n]$ has degree $d \in \mathbb{N}$, the *dense representation* of f is the sequence of coefficients $\{a_\alpha\}$ for all $\alpha \in \mathbb{N}^n$ with $|\alpha| = \alpha_1 + \ldots + \alpha_n \leq d$. The sequence is ordered by lexicographic ordering in $\mathbb{N}^n$. Then, the size of the dense representation of f is

$$size(f) = \sum_{|\alpha| \le d} size(a_\alpha).$$

Here $size(a) = ht(a)$ if $a \neq 0$ and $size(0) = 1$.

We propose the following conjecture.

Conjecture 1 *The feasibility of any diophantine equation $P(x, y) = 0$ can be decided in time 2^{Cs} where C is a universal constant and s is the size of P for the dense representation.*

This would follow from certain height estimates. Height bounds are a topic of current interest in number theory, but there are more conjectures than theorems. For instance, the Lang-Stark conjecture [Lang 1991] proposes the upper bound $|x| \le C \max(|a|^3, |b|^2)^k$ (C and k are universal constants) on the height of *all* solutions of equations of the form $y^2 = x^3 + ax + b$ with $4a^3 + 27b^2 \neq 0$. Here we only need a bound on the smallest height of a solution, though.

References

Akritas, A. [1989]. *Elements of Computer Algebra with Applications.* John Wiley & Sons.

Baker, A. [1975]. *Transcendental Number Theory.* Cambridge University Press.

Lang, S. [1991]. *Number Theory III*, Volume 60 of *Encyclopaedia of Mathematical Sciences.* Springer-Verlag.

Matiyasevich, Y. [1993]. *Hilbert's Tenth Problem.* The MIT Press.

Mignotte, M. [1992]. *Mathematics for Computer Algebra.* Springer-Verlag.

Schwartz, J. [1980]. Fast probabilistic algorithms for verification of polynomial identities. *Journal of the ACM 27*, 701–717.

Shub, M. [1993]. Some remarks on Bezout's theorem and complexity theory. In M. Hirsch, J. Marsden, and M. Shub (Eds.), *From Topology to Computation: Proceedings of the Smalefest*, pp. 443–455. Springer-Verlag.