Interpolation of polynomials given by straight-line programs

Sanchit Garg[a] Éric Schost[b]

[a]*India Institute of Technology, New Dehli, India*

[b]*Computer Science Department, The University of Western Ontario, London, Ontario, Canada*

Abstract

We give an algorithm for the interpolation of a polynomial A given by a straight-line program. Its complexity is polynomial in $\tau, \log(d), L, n$, where τ is an input bound on the number of terms in A, d is a bound on its partial degree in all variables, L is the length of the given straight-line program and n is the number of variables.

Key words: Interpolation, straight-line program

1 Introduction

There is a growing literature regarding the interpolation of sparse polynomials. In many studies, given a *black box* that computes a polynomial A, one seeks to output the sets of coefficients and monomials of A. Ideally, we would wish for a complexity polynomial in the number of terms in A and in the logarithm of its degree (since this is the bit-length of its largest exponent). However, no such result is known; we give references to previous work below.

We are interested here in interpolating polynomials given not by black-boxes but by *straight-line programs*, that perform only additions, subtractions and multiplications and use constants from a ring $\mathbb{S}$ (see e.g. [4, Chapter 4]). As is to be expected, this restricted model will enable us to devise better algorithms. On the other hand, this model is powerful enough to cover many applications.

Email addresses: `sanchiit_g@yahoo.com` (Sanchit Garg), `eschost@uwo.ca` (Éric Schost).

Our motivation comes from polynomial system solving algorithms using Hensel lifting techniques, such as those initiated by [15,13,14]. For systems of positive dimension, the output of these algorithms is given as a straight-line program in the "parameters" of the problem. Since these methods are used in conjunction with modular techniques, typical base fields for such applications are prime fields of cardinality up to 2^{80}. When the monomial representation of the output is required, dense interpolation techniques are used (such as in the implementation in the Maple RegularChains package of the algorithm of [6]). Obviously, sparse interpolation of straight-line programs will be useful to handle the case when the output is sparse (see e.g. [27] for considerations on the sparseness of polynomials arising in elimination processes).

We start by a result for univariate polynomials. To state it, we use the $O\tilde{}(\,)$ notation, so as to omit logarithmic factors: f is in $O\tilde{}(g)$ if f is in $O(g\log(g)^{\alpha})$, for some constant α [8, Chapter 25.7]. Logarithms are in base 2.

Theorem 1 *Let A be in $\mathbb{S}[X]$, where $\mathbb{S}$ is a ring. Given a straight-line program of size L that computes A, as well as upper bounds τ and d on the number of terms and degree of A, one can find all coefficients and exponents of A using $O\tilde{}(L\tau^4\log(d)^2)$ operations in $\mathbb{S}$ and a similar number of bit operations.*

From this, one easily deduces the following corollary for the multivariate case.

Corollary 1 *Let A be in $\mathbb{S}[X_1,\ldots,X_n]$, where $\mathbb{S}$ is a ring. Given a straight-line program of size L that computes A, as well as upper bounds τ and d on the number of terms and maximum partial degree of A, one can find all coefficients and exponents of A using $O\tilde{}(Ln^2\tau^4\log(d)^2+n^4\tau^4\log(d)^3)$ operations in $\mathbb{S}$ and a similar number of bit operations.*

In case the base ring is actually the integer ring, it is not hard to modify our results to obtain overall bit complexity estimates. In addition to the bound d on the degree of A, one would also require a bound h on the bit-length of its coefficients. Then, using the previous algorithms over $\mathbb{S}=\mathbb{Z}/2^h\mathbb{Z}$ yields complexity results of order $O\tilde{}(L\tau^4\log(d)^2h)$ for univariate polynomials and $O\tilde{}(Ln^2\tau^4\log(d)^2h+n^4\tau^4\log(d)^3h)$ for multivariate ones. For straight-line programs involving *rational* constants, extra care should be taken to avoid cancelling denominators through modular reductions; we refer to [18, Section 6] for an example of a probabilistic workaround to such issues.

For the applications we have in mind to polynomial system solving, degree bounds (such as Bézout's) are available. The determination of bounds on the number of terms is the subject of more recent work such as [27]; without such bounds, one should use probabilistic early termination as in [24].

Acknowledgements. Erich Kaltofen greatly contributed to this paper through numerous discussions; we also wish to thank him for sending us a copy of the manuscripts [21,19]. Our thanks also go to Mark Giesbrecht and Dan Roche for many discussions on this topic, and to the reviewers for their comments. Financial support from NSERC and the Canada Research Chair program is acknowledged.

2 Previous work

Avendaño, Krick and Pacetti [2, Theorem 3.1] obtained an algorithm for interpolating a polynomial $A \in \mathbb{Z}[x]$ given by a straight-line program, with a bit complexity polynomial in L, τ, $\log(d)$, h and h', where h is an upper bound on the height of A and h' an upper bound on the height of the values of A (and some of its derivatives) at some sample points. As far as we can tell, this algorithm does not work over arbitrary rings, e.g., not in characteristic less than τ. Previously, Kaltofen [18] gave an algorithm with a complexity polynomial in the degree bound d.

Most previous results address the related question of black-box interpolation over a field. We review here some of these results, pointing out in particular that for this question, no algorithm is known that would have a complexity polynomial in $\tau, \log(d)$ and apply over arbitrary fields.

A first work (that did not use black boxes) was Grigoriev and Karpinski's [16], for a particular polynomial (the determinant of a Tutte matrix). Other early results are due to Zippel [28,29] and Ben Or and Tiwari [3], with improvements by Kaltofen and Lakshman [22]. Zippel's algorithm requires root finding in degree τ and computing τ discrete logarithms, assuming that these logarithms are bounded by d; no bound polynomial in $\log(d)$ is known for this task for an arbitrary base field.

For polynomials with integer coefficients, Mansour [26] and Alon and Mansour [1] obtain a deterministic bit complexity polynomial in $h, n, \log(d), \tau$, where h is an upper bound on the bit-length of the output coefficients, but their approach does not seem to extend to other rings or fields. Avendaño, Krick and Pacetti [2] obtain a heuristic algorithm for the interpolation of polynomials in $\mathbb{Z}[X]$, with a complexity polynomial in $h, \log(d), \tilde{h}, p$, where $\tilde{h}$ is a bound on the heights of $A(y_i)$, for some suitable evaluation points $y_i \in \mathbb{Z}$, and where p is a "lucky" prime greater than τ. However, the probabilistic aspects are not fully understood yet.

Kaltofen and Lakshman [22,21] and Kaltofen, Lakshman and Wiley [23] present modular versions of Ben Or and Tiwari's algorithm, with a cost quasi-linear in τ in [21]. Kaltofen [19] also suggested to compute modulo primes p for which

$p - 1$ is smooth, to facilitate the discrete logarithm computations.

Over finite fields, Grigoriev, Karpinski and Singer [17] show that sparse interpolation is possible, up to computing in extensions of the base field; their algorithm has complexity $O\tilde{}(n^2\tau^6 + q^{2.5})$, where q is the size of the base field. In floating-point arithmetic, a numerically robust extension of Ben Or and Tiwari's algorithm is described in [11].

A question close to sparse interpolation is the recovery of the sparsest shift of a polynomial [10]. Giesbrecht and Roche [12] showed recently how to recover the sparsest shift using modular methods; even if the question and their method are distinct from ours, it is worth mentioning that similar techniques are used (generating families of primes provably containing suitable ones).

3 Preliminaries

Our basic idea (already used in many references above) consists in evaluating the unknown polynomial A at roots of unity; here, we will construct these roots by working in extensions of the base ring $\mathbb{S}$ of the form $\mathbb{S}[X]/\langle X^{p_i} - 1\rangle$, for suitable values of p_i. To recover A, we will use an approach similar to Lemma 3 in Grigoriev and Karpinski's work [16], combining Chinese remaindering, applied to symmetric functions, and integer root finding (in [16], the application is different: this idea is used to solve systems of the form $\sum_i x_i^m - \sum_j y_j^m = A_m$ in unknowns x_i and y_j).

Formally, our computational model is the algebraic RAM, as defined for instance by Kaltofen in [20]. The cost estimates count two kinds of operations: algebraic operations in the base ring and bit operations corresponding to integer manipulations, in particular flow control of the algorithm. Concretely, the cost of flow control will be negligible, so we will not mention it explicitly.

The algorithm relies on polynomial and integer multiplication. We thus denote by M a function such that polynomials of degree less than d can be multiplied in $\mathsf{M}(d)$ operations in the base ring $\mathbb{S}$ and $\mathsf{M}(d)$ bit operations, and by $\mathsf{M}_{\mathbb{Z}}$ a function such that integers of bit-length d can be multiplied in $\mathsf{M}_{\mathbb{Z}}(d)$ bit operations. We make the usual super-linearity assumptions of [8, Chapter 8]; by the results of Cantor-Kaltofen [5] and Fürer [7], one can take $\mathsf{M}(d)$ in $O(d\log(d)\log\log(d))$ and $\mathsf{M}_{\mathbb{Z}}(d)$ in $O(d\log(d)\,2^{O(\log^*(d))})$, that is, both are in $O\tilde{}(d)$.

We also need to find roots of a squarefree polynomial (written χ below) of degree n in $\mathbb{Z}[T]$, knowing that χ splits into linear factors in $\mathbb{Z}[T]$. In our context, we can slightly simplify the p-adic method of [25], as we will know a prime p (actually, several) for which χ remains squarefree modulo p. As pointed out in [22], one can find all roots of χ modulo p by fast evaluation, using $O(\mathsf{M}(p)\log(p))$ operations in $\mathbb{F}_p$, which is $O\tilde{~}(p)$ bit operations. Then, by [8, Theorem 15.18], lifting these roots to integer ones takes $O(\mathsf{M}(n)\mathsf{M}_{\mathbb{Z}}(h)\log(n)) \subset O\tilde{~}(nh)$ bit operations, where h is a bound on the bit-length of the coefficients and roots of χ, and on the bit-length of p.

4 Proof of the main results

As usual, we reduce the multivariate case to the univariate one by Kronecker's substitution. To $A \in \mathbb{S}[X_1, \ldots, X_n]$ of partial degree bounded by d in all variables, we associate $A' \in \mathbb{S}[X]$ given by

$$A' = A(Y_1, \ldots, Y_n),$$

with $Y_i = X^{(d+1)^{i-1}}$. The degree of A' is then less than $(d+1)^n$; besides, given a straight-line program of size L for A, one can deduce a straight-line program of size $L + O(n^2 \log(d))$ for A', as each variable is raised to a power at most $(d+1)^n$ to form A'. Conversely, given the monomials and exponents of A', one can recover the exponents of A by writing those of A' in base $d+1$; this takes $O\tilde{~}(n\log(d))$ bit operations per coefficient. Taking these overheads into account, one readily deduces Corollary 1 from Theorem 1.

We can thus focus on the univariate case of Theorem 1. Recall that A is given by a straight-line program that performs only additions, subtractions and multiplications; we let L be the number of these operations. We further let τ and d be the given bounds on the number of terms and degree of A, so that we can write

$$A = \sum_{i=1}^{\sigma} a_i X^{e_i},$$

with $\sigma \le \tau$, $a_i \ne 0$ and $e_i \le d$ for all i, and $e_i < e_j$ for $i < j$.

Let $(p_i)_{i\ge 1} = (2, 3, 5, \ldots)$ be the set of primes and let χ be the (unknown) polynomial $\prod_{i=1}^{\sigma}(T - e_i) \in \mathbb{Z}[T]$. We now describe all steps of the algorithm and give their cost; summing these costs proves Theorem 1.

Step 0: computing bounds.

- We first choose q such that all coefficients of χ are bounded (in absolute value) by q. A suitable bound is $q = (d+1)^\tau$; a marginally better one is

$$(d+2-\tau)\cdots d(d+1).$$

Yet better results are obtained by taking the maximum of the coefficients of $(T+d+1-\tau)\cdots(T+d-1)(T+d)$, but this brings little improvement.

- We next choose r such that for any primes $p_{\sigma_1},\dots,p_{\sigma_r}$, $\prod p_{\sigma_i} \geq q$. To do so, remark that
$$p_{\sigma_1}\cdots p_{\sigma_r} \geq 2^r.$$
Hence, one can take $r = \lceil \log(q) \rceil \in O(\tau \log(d))$. After generating primes up to $p_{\lceil \log(q)\rceil}$, one can find a sharper bound if wanted.
- We next choose s such that $\Delta = \prod_{i<j}(e_j - e_i)$ has at most s prime factors. Remark that $\Delta \leq d^{\tau^2}$, so one can take $s = \lfloor \tau^2 \log(d) \rfloor$. By generating primes up to $p_{\lfloor \tau^2 \log(d)\rfloor}$, one could get a sharper bound if wanted.
- Finally, let $N = r + s \in O(\tau^2 \log(d))$.

Cost: using binary powering, one can compute q and r in $O(\mathsf{M}_{\mathbb{Z}}(\tau \log(d)))$ bit operations and s in $O(\mathsf{M}_{\mathbb{Z}}(\tau^2 \log(d)))$ bit operations; the cost of computing N is polylogarithmic in τ and $\log(d)$. Hence, the total time is $O\tilde{}(\tau^2 \log(d))$ bit operations.

Step 1: finding primes. By sieving, compute the first N primes $p_1,\dots,p_N$. All p_i are bounded by $2N\log(N)$; their sum is thus in $O(N^2 \log(N))$.

Cost: by [8, Theorem 18.10], the cost is $O(N \log(N)^2 \log\log(N))$ bit operations, which is in $O\tilde{}(\tau^2 \log(d))$.

Step 2: modular evaluations. For $i = 1,\dots,N$, compute $A_i = A \bmod (X^{p_i} - 1)$, as a dense polynomial, and let S_i be the set of exponents of A_i. Remark that for all i, $A_i = \sum_{j=1}^{\sigma} a_j X^{e_j \bmod p_i}$, so that S_i is contained in the set $\{e_j \bmod p_i\}$; however, it may happen that for some i, S_i has size less than σ if two exponents have the same residue modulo p_i.

Cost: for any i, we compute A_i by evaluating the given straight-line program for A modulo $X^{p_i} - 1$; all intermediate results are thus dense polynomials of degree less than p_i as well (an example of a similar computation is in [9]).

An addition modulo $X^{p_i} - 1$ takes $O(p_i)$ operations in $\mathbb{S}$; a multiplication modulo $X^{p_i} - 1$ takes $\mathsf{M}(p_i) + p_i$ operations, since Euclidean division by $X^{p_i} - 1$ uses only p_i additions. Hence, the cost of a single operation $(+, -, \times)$ in $\mathbb{S}[X]/(X^{p_i} - 1)$ is $O(\mathsf{M}(p_i))$. Computing A_i takes L times more, that is, $O(L\mathsf{M}(p_i))$ operations in $\mathbb{S}$. Summing on all i gives $O(LN\mathsf{M}(N\log(N)))$ operations in $\mathbb{S}$, which is in $O\tilde{}(L\,\tau^4 \log(d)^2)$; the bit complexity is similar.

Step 3: filtering. Find $\sigma' = \max_{i \leq N} |S_i|$, and discard all p_i for which S_i does not have cardinality σ'. Let $R \subset \{1,\dots,N\}$ be the set of remaining indices. We claim that $|R| \geq r$ and $\sigma' = \sigma$ (the number of terms in A). Indeed, we remark first that $|S_i| \leq \sigma$ for all i. Besides, $|S_i| < \sigma$ if and only if p_i divides

$\Delta = \prod_{i<j}(e_i - e_j)$. Since Δ has at most s prime factors, there are at least $N - s = r$ elements i in $\{1, \ldots, N\}$ for which $|S_i| = \sigma$.

Cost: this step only requires to scan the sequence of exponents of the polynomials A_i. Since all degrees are at most $2N\log(N)$, the cost is $O\tilde{}(N^2)$ bit operations, which is in $O\tilde{}(\tau^4 \log(d)^2)$.

Step 4: recovering χ. At this stage, we know the sets S_i, but do not know how to pair their elements to do Chinese remaindering. However, the symmetric functions of the elements of the sets S_i can be matched unambiguously. Thus, for all $i \in R$, starting from S_i, we compute $\chi_i = \prod_{e \in S_i}(T - e) \in \mathbb{F}_{p_i}[T]$; we deduce χ by applying the Chinese remainder theorem to the (χ_i, p_i). Correctness follows from the fact that $|R| \geq r$, so that the product of the primes p_i for i in R is an upper bound on the coefficients in χ.

Cost: for any i, χ_i can be computed from its roots in $O(\mathsf{M}(\tau)\log(\tau))$ operations in $\mathbb{F}_{p_i}$ using subproduct tree techniques [8, Chapter 10], whence a total cost of $O\tilde{}(\tau N)$ bit operations. Then, since $\prod_{i \in R} p_i \leq (2N\log(N))^N$, Chinese remaindering has a bit cost of $O\tilde{}(\tau N)$ as well [8, Chapter 10]. Hence, the total cost is $O\tilde{}(\tau^3 \log(d))$ bit operations.

Step 5: recovering the e_i. Find the roots of χ to recover the exponents e_i, by lifting the roots known modulo p_i, for an i in R.

Cost: by the discussion in the previous section, the cost is $O\tilde{}(\tau^2 \log(d))$ bit operations.

Step 6: recovering the coefficients. Pick an element j in R. For $i = 1, \ldots, \tau$, compute $f_i = e_i \bmod p_j$ and let a_i be the coefficient of f_i in A_j.

Cost: since we perform τ modular reductions on integers bounded by d, the cost is $O\tilde{}(\tau \log(d))$ bit operations.

References

[1] N. Alon and Y. Mansour. epsilon-discrepancy sets and their application for interpolation of sparse polynomials. *Inf. Process. Lett.*, 54(6):337–342, 1995.

[2] M. Avendaño, T. Krick, and A. Pacetti. Newton-Hensel interpolation lifting. *Foundations of Computational Mathematics*, 6(1):82–120, 2006.

[3] M. Ben-Or and P. Tiwari. A deterministic algorithm for sparse multivariate polynomial interpolation. In *20th Annual ACM Symp. Theory Comp.*, pages 301–309. ACM, 1988.

[4] P. Bürgisser, M. Clausen, and M. A. Shokrollahi. *Algebraic Complexity Theory*, volume 315 of *Grundlehren der mathematischen Wissenschaften.* Springer-Verlag, 1997.

[5] D. G. Cantor and E. Kaltofen. On fast multiplication of polynomials over arbitrary algebras. *Acta Informatica*, 28(7):693–701, 1991.

[6] X. Dahan, X. Jin, M. Moreno Maza, and É. Schost. Change of order for regular chains in positive dimension. *Theoretical Computer Science*, to appear.

[7] M. Fürer. Faster integer multiplication. In *39th Annual ACM Symp. Theory Comp.*, pages 57–66. ACM, 2007.

[8] J. von zur Gathen and J. Gerhard. *Modern computer algebra.* Cambridge University Press, 1999.

[9] P. Gaudry, É. Schost, and N. Thiéry. Evaluation properties of symmetric polynomials. *International Journal of Algebra and Computation*, 16(3):505–523, 2006.

[10] M. Giesbrecht, E. Kaltofen, and W.-S. Lee. Algorithms for computing sparsest shifts of polynomials in power, Chebyshev, and Pochhammer bases. *J. Symbolic Comput.*, 36(3–4):401–424, 2003.

[11] M. Giesbrecht, G. Labahn, and W.-S. Lee. Symbolic-numeric sparse interpolation of multivariate polynomials [extended abstract]. In *ISSAC'06*, pages 116–123. ACM Press, 2006.

[12] M. Giesbrecht and D. Roche. Interpolation of shifted-lacunary polynomials. In *MACIS'07*, to appear.

[13] M. Giusti, K. Hägele, J. Heintz, J.-E. Morais, J.-L. Montaña, and L.-M. Pardo. Lower bounds for Diophantine approximation. *Journal of Pure and Applied Algebra*, 117/118:277–317, 1997.

[14] M. Giusti, J. Heintz, J.-E. Morais, J. Morgenstern, and L.-M. Pardo. Straight-line programs in geometric elimination theory. *Journal of Pure and Applied Algebra*, 124:101–146, 1998.

[15] M. Giusti, J. Heintz, J.-E. Morais, and L.-M. Pardo. When polynomial equation systems can be solved fast? In *AAECC'11*, volume 948 of *LNCS*, pages 205–231. Springer-Verlag, 1995.

[16] D. Y. Grigoriev and M. Karpinski. The matching problem for bipartite graphs with polynomially bounded permanents is in NC (extended abstract). In *FOCS*, pages 166–172. IEEE, 1987.

[17] D. Y. Grigoriev, M. Karpinski, and M. F. Singer. Fast parallel algorithms for sparse multivariate polynomial interpolation over finite fields. *SIAM J. Comput.*, 19(6):1059–1063, 1990.

[18] E. Kaltofen. Computing with polynomials given by straight-line programs I: greatest common divisors. In *Proceedings of the seventeenth annual ACM symposium on Theory of computing*, pages 131–142. ACM, 1985.

[19] E. Kaltofen, 1988. Unpublished manuscript.

[20] E. Kaltofen. Greatest common divisors of polynomials given by straight-line programs. *J. ACM*, 35(1):231–264, 1988.

[21] E. Kaltofen and Y. Lakshman. Improved sparse multivariate polynomial interpolation algorithms, 1988. Unpublished manuscript.

[22] E. Kaltofen and Y. Lakshman. Improved sparse multivariate polynomial interpolation algorithms. In *ISSAC-88*, volume 358 of *LNCS*, pages 467–474. Springer Verlag, 1989.

[23] E. Kaltofen, Y. Lakshman, and J.-M. Wiley. Modular rational sparse multivariate polynomial interpolation. In *ISSAC'90*, pages 135–139. ACM, 1990.

[24] E. Kaltofen and W.-S. Lee. Early termination in sparse interpolation algorithms. *J. Symbolic Comput.*, 36(3–4):365–400, 2003.

[25] R. Loos. Computing rational zeros of integral polynomials by p-adic expansion. *SIAM J. Comput.*, 12(2):286–293, 1983.

[26] Y. Mansour. Randomized interpolation and approximation of sparse polynomials. *SIAM J. Comput.*, 24(2):357–368, 1995.

[27] B. Sturmfels, J. Tevelev, and J. Yu. The Newton polytope of the implicit equation. *Mosc. Math. J.*, 7(2):327–346, 351, 2007.

[28] R. Zippel. Probabilistic algorithms for sparse polynomials. In *EUROSAM' 79*, volume 72 of *LNCS*. Springer Verlag, 1979.

[29] R. Zippel. Interpolating polynomials from their values. *J. Symb. Comp.*, 9, 1990.