Sparse polynomial interpolation in practice

Joris van Der Hoeven, Grégoire Lecerf

Sparse polynomial interpolation in practice

by Joris van der Hoeven and Grégoire Lecerf

Laboratoire d'informatique de l'École polytechnique (LIX, UMR 7161 CNRS)
Campus de l'École polytechnique, Route de Saclay, 91128 Palaiseau Cedex, France

Email: {vdhoeven,lecerf}@lix.polytechnique.fr

1 Introduction

Sparse polynomial interpolation consists in recovering of a sparse representation of a polynomial P given by a blackbox program which computes values of P at as many points as necessary. In practice P is typically represented by DAGs (Directed Acyclic Graphs) or SPLs (Straight Line Programs). For polynomials with at most t terms over a field $\mathbb{K}$, this task typically involves $O(t)$ evaluations of P, operations on integers for discovering the exponents of the nonzero terms of P, and an additional number of operations in $\mathbb{K}$ that can be bounded by $\tilde{O}(t)$ for recovering the coefficients [3].

However, the latter complexity analysis does not take into account the sizes of coefficients in $\mathbb{K}$ and the expression swell that might occur when evaluating at points with high height [9]. For practical implementations, it is important to perform most of the computations over a prime finite field $\mathbb{F}_p$, where p fits into a single machine register. On modern architectures, this typically means that $p < 2^{64}$. There mainly are two approaches which can be used over finite fields: the "prime number" approach [9] and the "Kronecker substitution" approach [8]. For more references and practical point of views on known algorithms we refer the reader to [1, 2, 7] (see also [5] in a more restricted context). We recall these approaches in Section 2. In Section 3, we present new techniques which may be used to further enhance and combine these classical approaches.

We report on our implementations in the multimix C++ library of the Mathemagix system [6]. For the sake of conciseness some discussions on the probabilistic aspects will be informal. The interested reader might consult [2, 10] on this topic. Of course, once a candidate for the sparse representation is found, then the Schwartz-Zippel lemma can be used to verify it with a low level of failure [4].

2 Previously known approaches

Throughout this paper, we will use vector notation for n-tuples: given variables $x_1, ..., x_n$, we write $\mathbb{K}[x] := \mathbb{K}[x_1, ..., x_n]$. If α is a vector of n elements in $\mathbb{K}$ or the sequence of the variables $x_1, ..., x_n$, and if $a \in \mathbb{N}^n$, then we write $\alpha^a := \alpha_1^{a_1} \cdots \alpha_n^{a_n}$ and $|a| := a_1 + \cdots + a_n$. If $P = \sum_{i=1}^{t} c_i\, x^{e_i}$ with all the exponents $e_i \in \mathbb{N}^n$ pairwise distinct and $c_i \neq 0$, then for all $\alpha \in \mathbb{K}^n$ we have

$$f(z) := \sum_{i \geqslant 0} P(\alpha_1^i, ..., \alpha_n^i)\, z^i \;=\; \sum_{1 \leqslant i \leqslant t} \frac{c_i}{1 - \alpha^{e_i} z}. \tag{1}$$

If the values α^{e_i} are pairwise distinct, then it is classical that $c_1, \alpha^{e_1}, ..., c_t, \alpha^{e_t} \in \mathbb{K}$ can be computed efficiently from the first $2\,t$ terms of the power series $f(z)$ [3]. One may further recover the e_i from the α^{e_i} with suitable values of α. If no bound for t is known in advance, then we try to interpolate for successive upper bounds t in geometric progression.

Over many common fields such as $\mathbb{K} = \mathbb{Q}$, the numbers α^{e_i} can become quite large. For efficiency reasons, it is better to work over a finite field $\mathbb{F}_p$ as much as possible. Moreover, we wish to take a p which fits into a single machine register, so that computations in $\mathbb{F}_p$ can be done very efficiently. When using modular reduction, the problems of computing the e_i and computing the c_i are actually quite distinct, and are best treated separately. The e_i do not mostly depend on p (except for a finite number of bad values) and only need to be computed for one p. The c_i are potentially large numbers in $\mathbb{Q}$, so we may need to compute their reductions modulo several p and then recover them using rational number reconstruction [4]. From now on, we focus on the computation of the *support* of P written $\operatorname{supp} P := \{e_i \,|\, 1 \leqslant i \leqslant t\}$. Let d_i represent the partial degree of P in x_i, and let $D_i := d_i + 1$.

For the determination of suitable $\alpha \in \mathbb{F}_p^n$ such that the e_i can be reconstructed from the α^{e_i}, the "prime number" approach is to take $\alpha = q \bmod p$, where $q = (q_1, ..., q_n)$ and q_i is the i-th prime number. As long as $q^{e_i} < p$, we may then recover e_i from q^{e_i} by repeated divisions. If $d := \max_{i \leqslant t} |e_i|$ is the total degree of P, then it suffices that $q_n^d < p$. We recall that q_n asymptotically grows as $\tilde{O}(n)$, so that p can be taken of bit-size in $\tilde{O}(d \log n)$. This approach is therefore efficient when d is sufficiently small.

If $\mathbb{K} = \mathbb{F}_p$, the "Kronecker substitution" approach is to take $\alpha = (\omega, \omega^{D_1}, \omega^{D_1 D_2}, ..., \omega^{D_1 \cdots D_{n-1}})$, where ω is a primitive root of the multiplicative group $\mathbb{F}_p^*$. Assuming that $D_1 \cdots D_n \leqslant p - 1$, we can compute all the value $\alpha^{e_i} = \omega^{e_{i,1} + e_{i,2} D_1 + \cdots + e_{i,n} D_1 \cdots D_{n-1}}$. Moreover, assuming that the prime number p is "smooth", meaning that $p - 1$ has only small prime factors, the discrete logarithm problem can be solved efficiently in $\mathbb{F}_p^*$: given $a \in \mathbb{F}_p^*$ there exists a relatively efficient algorithm to compute $l \in \{0, ..., p-2\}$ with $\omega^l = a$ [11]. In this way we deduce all the e_i. This approach is efficient when the partial degrees are small. More specifically, we prefer it over the prime number approach if $D_1 \cdots D_n < q_n^d$.

We notice that bounds for $d, d_1, ..., d_n$ are sometimes known in advance (e.g. from a syntactic expression for P) and sometimes have to be guessed themselves. The total degree can be guessed efficiently by using sparse interpolation for the univariate polynomial $P(\alpha_1 t, ..., \alpha_n t)$, where $\alpha = (\alpha_1, ..., \alpha_n)$ is a random point in $(\mathbb{F}_p^*)^n$. Similarly, the partial degrees d_i are obtained by interpolating $P(\alpha_1, ..., \alpha_{i-1}, t, \alpha_{i+1}, ..., \alpha_n)$.

Remark. It often occurs that the expression for P involves division even though the end result is a polynomial; this happens for instance when computing a symbolic determinant using Gaussian elimination. When division is allowed in the prime number approach, then some care is required in order to avoid systematic divisions by zero. For instance, $1/(x_1 - 1)$ cannot be evaluated at α^0 and $1/(3\,x_1 - 2\,x_2)$ cannot be evaluated at α^1 when $\alpha = q = (2, 3, ..., q_n)$. The first problem can be avoided by taking $f := \sum_{i \geqslant 1} P(\alpha^i)\, z^i$. The nuisance of the second problem can be reduced by replacing the q_i by small random and pairwise distinct primes.

3 New incremental approaches

In this section we carry on with the same notation, and $\mathbb{K}$ is the finite field $\mathbb{F}_p$. We aim at decomposing the interpolation problem into several ones with less variables in order to interpolate more polynomials than with the previously known techniques. For all our algorithms, we assume that the support $\operatorname{supp} Q = \{f_1, ..., f_s\} \subseteq \mathbb{N}^{n-m}$ of $Q(x_{m+1}, ..., x_n) := P(\alpha_1, ..., \alpha_m, x_{m+1}, ..., x_n)$ is known for some $1 < m < n$ and random $\alpha_1, ..., \alpha_m \in \mathbb{F}_p^*$. We will write $d_{\leqslant m}$ and $d_{>m}$ for the total degrees of P in $x_1, ..., x_m$ resp. $x_{m+1}, ..., x_n$. Let $\eta > 0$ and assume that $p \geqslant \eta\, d\, t$. By the Schwartz-Zippel lemma, taking random α_i in $\{0, ..., \eta\, s\, d_{\leqslant m} - 1\}$ ensures that $\operatorname{supp} Q$ coincides with the projection of $\operatorname{supp} P$ on $\mathbb{N}^{n-m}$ with probability at least $1 - 1/\eta$.

Separating forms. Assume that $1 < m < n - 1$ and let $u \in \mathbb{N}^{n-m}$ be such that $u \cdot f_1, ..., u \cdot f_s$ are pairwise distinct. Such a vector u, seen as a linear form acting on the exponents, is said to separate the exponents of Q. A random vector in $\{0, ..., \eta\, s^2/2\}^{n-m}$ separates exponents with probability at least $1 - 1/\eta$. Since one can quickly verify that u is a separating form, it is worth spending time to search for u with small entries.

Now consider a new variable y and $R(x_1, ..., x_m, y) := P(x_1, ..., x_m, y^{u_1}, ..., y^{u_{n-m}})$. The partial degree of R in y is $O(\eta\, s^2\, d_{>m})$, hence its total degree is $O(\eta\, d\, t^2)$. If h is in the support of R then there exists a unique exponent e of P such that $e = (h_1, ..., h_m, u \cdot (e_{m+1}, ..., e_n))$. We can compute e efficiently after sorting the f_i accordingly to the values $u \cdot f_i$. We have thus shown how to reduce the sparse interpolation problem for P to the "smaller" sparse interpolation problem for Q. If $p \gg d\, t^2$, then the method succeeds with high probability.

Coefficient ratios. Let $\alpha = (q_1, ..., q_n)$ and let $f(z)$ be the generating series associated to P and α as in (1). In a similar way, we may associate a generating series $\tilde{f}(z)$ to $\tilde{P}(x) = P(q_1\, x_1, ..., q_m\, x_m, x_{m+1}, ..., x_n)$. Using sparse interpolation, we find c_i and α^{e_i} such that (1) holds, as well as the analogous numbers $\tilde{c}_i = c_i\, q_1^{e_{i,1}} \cdots q_m^{e_{i,m}}$ and α^{e_i} for $\tilde{f}(z)$. If $q_m^{d_{\leqslant m}} < p$, then these data allow us to recover the vectors $e_{i,\leqslant m} = (e_{i,1}, ..., e_{i,m})$ from the ratios $\tilde{c}_i / c_i$. We may next compute the complete e_i from $\alpha^{e_i - e_{i,\leqslant m}}$, since we assumed $\operatorname{supp} Q$ to be known. This strategy also applies to the "Kronecker substitution" approach. In that case, we take $\tilde{P}(x) = P(\omega\, x_1, ..., \omega^{D_1 \cdots D_{m-1}}\, x_m, x_{m+1}, ..., x_n)$, and we require that $D_1 \cdots D_m \leqslant p - 1$. If none of the conditions $q_m^{d_{\leqslant m}} < p$ or $D_1 \cdots D_m < p - 1$ holds, then we let $\tilde{m}$ be maximal such that either $q_{\tilde{m}}^{d_{\leqslant \tilde{m}}} < p$ or $D_1 \cdots D_{\tilde{m}} \leqslant p - 1$, and we recursively apply the strategy to $P(\alpha_1, ..., \alpha_{\tilde{m}}, x_{\tilde{m}+1}, ..., x_n)$ and then to P itself.

Closest points. Let $\alpha = (\omega, \omega^{D_1}, ..., \omega^{D_1 \cdots D_{n-1}})$ be as in the "Kronecker substitution" approach. For each f_i in the support of Q, let $\varphi_i \in \{0, ..., p-2\}$ be such that $\omega^{\varphi_i} = \alpha^{f_i}$, and let δ be the smallest distance between any two distinct points of $\bigcup_{i=0}^{s} (\varphi_i + (p-1)\,\mathbb{Z})$. If $d_{>m}$ is large and the exponents of P are random, then δ is expected to be of the order p/s^2 (this follows from the birthday problem: taking s random points out of p/δ, the probability of a collision is $\geqslant 1/2$ as soon as $s \asymp \sqrt{p/\delta}$). Now assume that $D_1 \cdots D_m \leqslant \delta$. Performing sparse interpolation on P, we obtain numbers $\omega^{e_1}, ..., \omega^{e_t}$, as well as the corresponding $\varepsilon_i \in \{0, ..., p-2\}$ with $\omega^{\varepsilon_i} = \alpha^{e_i}$. For each $i \in \{1, ..., t\}$, there is unique $j \in \{1, ..., s\}$ and $k \in \{0, 1\}$ with $0 \leqslant \varepsilon_i - \varphi_j + k\,(p-1) < D_1 \cdots D_m \leqslant \delta$. We may thus recover e_i from f_j and $\varepsilon_i - \varphi_j + k\,(p-1)$. In order to improve the randomness of the numbers φ_i modulo $p-1$, it is recommended to replace D_m by a random larger number.

4 Implementation and timings

The MATHEMAGIX platform divides into three main components: low level structuring and efficient C++ libraries for classical mathematical objects, a compiler and an interpreter of the eponym language, and interfaces to external libraries. MATHEMAGIX is an open source academic project hosted at http://gforge.inria.fr/projects/mmx/. The Internet home page concerning installation and documentation is http://www.mathemagix.org.

Multivariate polynomials, power series, jets, and related algorithms are integrated in the MULTIMIX C++ library of MATHEMAGIX. Algorithms for sparse and dense representations have been previously reported in our article [5], in which we focused on fast polynomial products assuming given an overset of the support. This short note reports on the implementation of DAGs, SLPs, and sparse interpolation. At the C++ level, a DAG with coefficients in `C` has type `dag_polynomial<C>`. Arithmetic operations, evaluation, substitution, and also construction from a polynomial in sparse representation are available. For efficiency purposes, the evaluation of one or several DAGs on several points can be speeded up by building a SLP from it, whose type is `slp_polynomial<C>`. These classes are available from `multimix/dag_polynomial.hpp` and `multimix/slp_polynomial.hpp`. An overview of the main features in MULTIMIX together with examples and useful links to browsable C++ source code is available from http://www.mathemagix.org/www/multimix/doc/html/index.en.html.

Sparse interpolation is implemented in `multimix/sparse_interpolation.hpp`. Complete examples of use can be found in `multimix/test/sparse_interpolation_test.cpp`. If $f : \mathbb{Q}^n \to \mathbb{Q}$ represents the polynomial function that we want to interpolate as a polynomial in $\mathbb{Q}[x_1, ..., x_n]$, then our main routines take as input a function $\bar{f} : \mathbb{Q}^n \times \mathbb{N} \to \mathbb{N}$ such that $\bar{f}\,((a_1, ..., a_n), p)$ is the preimage of $f(a_1, ..., a_n)$ computed modulo a prime number p. If computing f modulo p involves a division by zero, then an error is raised. For efficiency reasons, such a function $\bar{f}$ can be essentially a pointer to a compiled function. Of course DAGs over $\mathbb{Z}$ or $\mathbb{Q}$ can be converted to such functions *via* SLPs.

In the next examples, which can be found in `multimix/bench/sparse_interpolation_bench.cpp`, we illustrate the behaviours of the aforementioned algorithms on an *Intel(R) Core(TM) i7-3720QM CPU @ 2.60GHz* platform with 8 GB of *1600 MHz DDR3*.

Example 1. Our first family of examples concerns products of random polynomials given in sparse representation: we pick up m polynomials with t terms, in n variables, and with partial degrees at most d, and compute the support of their product. With $m = 8$, $n = 20$, $d = 40$, $t = 3$, then a direct use of the "Kronecker substitution" involves computing with GMP based modular integers of bit size about 160: the total time amounts to 171 s (91 s are spent in the Cantor-Zassenhaus stage for root finding). With the "coefficient ratios" approach, the set of variables is split into 3 blocks and all the computations can be done with 64-bit integers for a total time of 31 s (most of the time is still spent in the Cantor-Zassenhaus stages, but relatively less in the discrete logarithm computations). With the "closest points" technique the set of variables is split into 5 blocks and all the computations are also done with 64-bit integers for a total time of 47 s. The "separating form" approach is not competitive here, since it does not manage to keep the computations on 64 bits integers only. Nevertheless, in a "lacunary polynomials" context, namely when d becomes very large, the computation of a very large smooth prime number becomes very expensive, and the "separating form" point of view becomes useful.

Example 2. As a second family of examples, we consider the determinant of a $n \times n$ matrix whose entries are independent variables. This determinant is a polynomial in n^2 variables with $n!$ terms, whose total degree is n and whose partial degrees are all 1. The following table displays timings for interpolating this polynomial using the "Kronecker substitution" approach:

n	5	6	7	8
Kronecker substitution	110 ms	915 ms	9.8 s	162 s

Table 1. Timings for interpolating the determinant polynomial of a $n \times n$ matrix.

For $n=8$ computations are performed *via* 128-bits modular arithmetic. Here we used a fast compiled determinant function, so that most of the time is spent in the Cantor-Zassenhaus stage. We noticed that the "closest points" strategy is not well suited to this problem. The "coefficient ratios" approach allows to perform all the computations on 64-bits for $n=8$ within 223 s, but it is penalized by two calls to Cantor-Zassenhaus.

For a convenient use, our sparse interpolation routine is available inside the MMX-LIGHT interpreter. In the following session, we interpolate the determinant polynomial of a matrix of size $n \times n$ for $n=5$ in 1.3 s, and for $n=6$ in 9.6 s. As a comparison, the function `sinterp`, modulo $2^{31}-1$, of MAPLE 16 (trademark of WATERLOO MAPLE INC. http://www.maplesoft.com/products/maple) takes 18 s for $n=5$, and 1 hour for $n=6$.

```
Mmx] use "multimix"; n:= 5;
```

```
Mmx] det_mod (v: Vector Integer, p: Integer): Integer == {
       w == [ z mod modulus p | z in v ];
       M == [ @(w[i * n, (i+1) * n]) || i in 0..n ];
       return preimage det M; };
```

```
Mmx] coords == coordinates (coordinate ('x[i,j]) | i in 0..n | j in 0..n);
```

```
Mmx] #as_mvpolynomial% (det_mod, coords, 1000)
```

```
120
```

Bibliography

[1] A. Arnold, M. Giesbrecht, and D. S. Roche. Faster sparse polynomial interpolation of straight-line programs over finite fields. Technical report, arXiv:1401.4744, 2014.

[2] A. Arnold and D. S. Roche. Multivariate sparse interpolation using randomized Kronecker substitutions. Technical report, arXiv:1401.6694, 2014.

[3] M. Ben-Or and P. Tiwari. A deterministic algorithm for sparse multivariate polynomial interpolation. In *STOC '88: Proceedings of the twentieth annual ACM symposium on Theory of computing*, pages 301–309, New York, NY, USA, 1988. ACM Press.

[4] J. von zur Gathen and J. Gerhard. *Modern Computer Algebra*. Cambridge University Press, 2-nd edition, 2002.

[5] J. van der Hoeven and G. Lecerf. On the bit-complexity of sparse polynomial multiplication. *J. Symbolic Comput.*, 50:227–254, 2013.

[6] J. van der Hoeven, G. Lecerf, B. Mourrain, et al. Mathemagix, 2002. http://www.mathemagix.org.

[7] M. Javadi and M. Monagan. Parallel sparse polynomial interpolation over finite fields. In *Proceedings of the 4th International Workshop on Parallel and Symbolic Computation*, PASCO '10, pages 160–168. ACM Press, 2010.

[8] E. Kaltofen. Fifteen years after DSC and WLSS2, what parallel computations I do today. Invited lecture at PASCO 2010. In *Proceedings of the 4th International Workshop on Parallel and Symbolic Computation*, PASCO '10, pages 10–17, New York, NY, USA, 2010. ACM Press.

[9] E. Kaltofen, Y. N. Lakshman, and J.-M. Wiley. Modular rational sparse multivariate polynomial interpolation. In *ISSAC '90: Proceedings of the international symposium on Symbolic and algebraic computation*, pages 135–139, New York, NY, USA, 1990. ACM Press.

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

[11] S. C. Pohlig and M. E. Hellman. An improved algorithm for computing logarithms over GF(p) and its cryptographic significance. *IEEE Trans. Inf. Theory*, 24(1):106–110, 1978.