Lacunaryx: Computing bounded-degree factors of lacunary polynomials

Bruno Grenet*
LIRMM – UMR 5506 CNRS
Université de Montpellier
bruno.grenet@lirmm.fr

1 Introduction

In this paper, we report on an implementation in the free software Mathemagix [9] of lacunary factorization algorithms, distributed as a library called Lacunaryx. These algorithms take as input a polynomial in sparse representation, that is as a list of nonzero monomials, and a integer d, and compute its degree-d factors[1]. The complexity of these algorithms is polynomial in the size of the sparse representation of the input polynomial and d. The size of the sparse representation of a polynomial $f = \sum_{j=1}^{k} c_j X_1^{\alpha_{1,j}} \cdots X_n^{\alpha_{n,j}} \in \mathbb{Q}[X_1, \dots, X_n]$ is $\sum_j (h(c_j) + \sum_i \log(1 + \alpha_{i,j}))$ where the $h(\cdot)$ denotes the *height* of a rational number, that is the maximum of the absolute values of its numerator and denominator.

In our implementation, we focus on polynomials with rational or integer coefficients. Algorithms in the univariate case have been given by Cucker, Koiran and Smale [3] and extended by Lenstra [8]. We implement a variation of Lenstra's algorithm. In the multivariate case, the first algorithms were given by Kaltofen and Koiran [6, 7], and simpler algorithms with a different approach were then given by Chattopadhyay, Grenet, Koiran, Portier and Strozecki [2, 1] and extended by Grenet [5, 4]. We implement the latter. The algorithms of Koiran and Kaltofen are not easily implementable since they rely on the value on some non-explicit number-theoretic constant.

2 Algorithms

In this section, we briefly recall Lenstra's algorithm and our modification of it. This algorithm treats differently *cyclotomic factors*, that is factors that divide $X^r - 1$ for some r, and *non-cyclotomic factors*. Note that the factor X and its multiplicity are trivial to compute. Then we present the multivariate algorithm that we implement. We refer to the original papers for proofs of correctness and complexity estimates.

2.1 Lenstra's algorithm

Lenstra's algorithm has two completely disjoint steps for computing cyclotomic factors on the one hand, and non-cyclotomic factors on the other hand. Let us first describe the step for non-cyclotomic factors.

Theorem 1 (Lenstra's Gap Theorem) *Let d be a positive integer and $f = f_1 + f_2 \in \mathbb{Q}[X]$. There exists a constant $\gamma(f, d)$ such that if $\operatorname{val}(f_2) - \deg(f_1) > \gamma(f, d)$, every non-cyclotomic factor of degree at most d of f divides both f_1 and f_2.*

*Partially supported by a LIX–Qualcomm®–Carnot postdoctoral fellowship, and by the French Agence Nationale de la Recherche under grant CATREL #ANR-12-BS02-001.

[1]Here and henceforth, a *degree-d factor* is an irreducible factor of degree *at most d*.

The constant in the theorem is polynomial in the sparse representation of f and in d. Based on this theorem, Lenstra describes an algorithm that, given $f \in \mathbb{Q}[X]$ and $d > 0$, computes all non-cyclotomic factors of degree at most d of f in time polynomial in the sparse representation of f and in d. For, one first computes $\gamma(f, d)$ and computes f_1 of minimal degree such that $\operatorname{val}(f - f_1) - \deg(f_1) > \gamma(f, d)$. Then one can recursively compute $f_2, \dots, f_s$ such that $f = f_1 + \cdots + f_s$ and every degree-d non-cyclotomic factor of f is a common factor of $f_1, \dots, f_s$. These factors can be computed as factors of $\gcd(f_1, \dots, f_s)$. To obtain the multiplicities, the same algorithm is applied to the derivatives of f, or more precisely to its sparse derivatives: The sparse derivative $f^{[1]}$ of f is defined as $(f/X^{\operatorname{val}(f)})'$ where $'$ denotes the standard derivative.

The second step of Lenstra's algorithm computes the cyclotomic degree-d factors of f as follows. If g is cyclotomic, it divides by definition $X^r - 1$ for some r. Thus, to compute the cyclotomic factors of f that divide $X^r - 1$, it is enough to compute the factors of $\gcd(f, X^r - 1)$. Let us write $f = \sum_{j=1}^{k} c_j X^{\alpha_j}$ and define $f^{\bmod r} = \sum_{j=1}^{k} c_j X^{\alpha_j \bmod r}$. Then $f^{\bmod r}$ is a degree-$(r-1)$ polynomial and one can notice that $\gcd(f, X^r - 1) = \gcd(f^{\bmod r}, X^r - 1)$. This yields a polynomial time (in r and the sparse representation of f) algorithm to compute this gcd. Number-theoretic considerations show that considering all values of r up to $2d^2$ is enough to compute all the degree-d cyclotomic factors. Altogether, the degree-d cyclotomic factors of f can be computed in polynomial time in d and the size of the sparse representation of f. Again, the multiplicities are obtained using the same algorithm on the sparse derivatives of f.

2.2 Modifications to Lenstra's algorithm

To describe our modifications to Lenstra's algorithm, let us reformulate it slightly. In the first step, it computes a polynomial g that *contains* the non-cyclotomic factors of f as factors. One can easily compute $h = f/g$. Then the cyclotomic factors of f contained in both g and h. Lenstra's algorithm can be seen as follows: First compute g and h, then factor g, compute the cyclotomic factors of h and return the union of these factors (viewed as multiset, to take multiplicities into account).

We build on this view in our algorithm. Instead of computing a single g and a single h, we actually compute two sets G and H such that for any degree-d polynomial ℓ,

$$\operatorname{mult}_\ell(f) = \begin{cases} \sum_{g \in G} \operatorname{mult}_\ell(g) + \sum_{h \in H} \operatorname{mult}_\ell(h) & \text{if } \ell \text{ is cyclotomic, and} \\ \sum_{g \in G} \operatorname{mult}_\ell(g) & \text{otherwise.} \end{cases} \tag{1}$$

Let us remark that the most expensive part of Lenstra's algorithm is the factorization of the polynomial $\gcd(f_1, \dots, f_s)$. Our modification allows us to reduce as much as possible the degree of the polynomials that have to be finally factored.

To compute the sets G and H, we reverse the logic of Lenstra's algorithm: While Lenstra computes an a priori bound $\gamma(f, d)$ to split f as a sum, we split f as much as possible (say as a sum of monomials) and then merge together the summands, beginning with the closest ones. At each step, if the summands have a non trivial gcd g, we apply recursively our algorithm to f/g to get the two sets G and H, and return $(G \cup \{g\}, H)$. Else, we merge the two closest summands (that is we replace both of them by their sum) and compute again the gcd of the new summands, until there remains a unique summand. As described, the algorithm would always return an empty set H. This is because we always compute the gcd of the summands. Actually, this is useless is the following case: If two summands f_i and f_{i+1} have been merged but $\operatorname{val}(f_{i+1}) - \deg(f_i) > \gamma(f_i + f_{i+1}, d)$, we know that the non-cyclotomic degree-d factors of $f_i + f_{i+1}$ are common to f_i and f_{i+1}. The gcd does not need to be computed, and we can directly merge the next summands. Similarly, when it remains a unique summand at the end of the algorithm, we put it into H instead of G. The formal description of this algorithm is given in Algorithm 1.

After G and H have been computed, it remains to compute all the factors of the polynomials in G using any classical algorithm, and the cyclotomic ones for polynomials in H using Lenstra's algorithm for

Algorithm 1 Separation of f

Input: $f = \sum_{j=1}^{k} c_j X^{\alpha_j}$ with $\alpha_1 \leq \cdots \leq \alpha_k$, and $d > 0$;
Output: The sets G and H as defined in Eq. (1).

```
1: procedure SEPARATION($f, d$)
2:     $S \leftarrow [c_1 X^{\alpha_1}, \ldots, c_k X^{\alpha_k}]$;
3:     $b \leftarrow$ false;                                    ▷ Only monomials
4:     while $|S| > 1$ do
5:         if $b$ then
6:             $g \leftarrow \gcd(S)$;
7:             if $\deg(g) > 0$ then
8:                 $(G, H) \leftarrow$ SEPARATION($f/g, d$);
9:                 return $(G \cup \{g\}, H)$;
10:        $t \leftarrow$ index which minimizes $\delta = \mathrm{val}(S[t+1]) - \deg(S[t])$;
11:        $b \leftarrow (\delta > \gamma(S[t] + S[t+1], d))$;          ▷ true or false
12:        $S[t] \leftarrow S[t] + S[t+1]$;
13:        $S \leftarrow S \setminus \{S[t+1]\}$;
14:    if $b$ then return $(\{S[0]\}, \emptyset)$;
15:    else return $(\emptyset, \{S[0]\})$;
```

cyclotomic factors. Finally, the union of these factors is returned.

Actually, we also propose a modification of Lenstra's algorithm for cyclotomic factors. Let $h \in H$ be a polynomial, and suppose we aim to compute its degree-d cyclotomic factors. As explained in the previous section, one has to compute the relevant values of r to consider for computing all degree-d cyclotomic factors. Lenstra proposes to use the set $\{1, \ldots, 2d^2\}$ and proves that this set is sufficient. Actually, one can compute a much smaller set $R \subsetneq \{1, \ldots, 2d^2\}$ such if g is a degree-d cyclotomic polynomial, g divides $X^r - 1$ for some $r \in R$. By definition, the r-th cyclotomic polynomial, denoted by ϕ_r, is the only irreducible polynomial that divides $X^r - 1$ and does not divide any $X^s - 1$ for $s < r$. Furthermore, it is known that $\deg(\phi_r) = \varphi(r)$ where φ is Euler's totient function. This implies that to compute all cyclotomic degree-d factors of h, it is sufficient to consider the values of r such that $\varphi(r) \leq d$. In our implementation, we compute the set R of these values in a naive way, by considering all $r \in \{1, \ldots, 2d^2\}$ and evaluating φ on each of these integers. Note that a rough lower bound of $\varphi(r)$ is $\sqrt{r/2}$, whence the bound $2d^2$ in Lenstra's algorithm and in our implementation. Next for $r \in R$, we can actually compute exactly the r-th cyclotomic polynomial as $\phi_r = (X^r - 1)/\gcd(X^r - 1, \prod_{s<r} \phi_s)$. Thus, instead of computing the factors of $\gcd(f, X^r - 1)$, we compute ϕ_r and simply test whether it divides f, using again the fact that ϕ_r divides f if and only if it divides $f^{\bmod r}$.

2.3 Multivariate case

The multivariate case has the same high-level structure as the univariate case. Namely, the computation of the so-called *unidimensional factors* reduces to the factorization of some univariate polynomials, while the computation of the *multidimensional factors* is based on a *Gap Theorem*. The

A *unidimensional polynomial* is a polynomial $f \in \mathbb{Q}[X_1, \ldots, X_n]$ that can be written as $f(\boldsymbol{X}) = \boldsymbol{X}^{\boldsymbol{\alpha}} f_\pi(\boldsymbol{X}^{\boldsymbol{\delta}})$ where $f_\pi \in \mathbb{Q}[X]$ has valuation 0. A polynomial that is not unidimensional is said *multidimensional.*

The algorithms we implement for these two steps are almost identical to the one described in our previous work. Simply, for the multidimensional factors, we use the same modification as for Lenstra's algorithm in the univariate case, that is we first write f as a sum of monomials and then iteratively merge the summands. This way, we actually build a so-called *single-linkage clustering* of the set of exponents.

3 Examples

We include an example of timings that is pretty representative of the computation times we obtain. It is not comparable to other softwares since in the range of degrees that we consider, other software are unable to give any answer. (Often, they even do not accept large enough exponents for polynomials.) Also, the nature of the algorithms we use make the computation times very versatile: A very small change in the input polynomial may force the algorithm to factor a much larger polynomial, or to compute a gcd with much larger inputs. This is a important question to be able to (automatically) provide an estimation of the size of the polynomials that will appear in the gcd computations.

We consider a random polynomial of degree $> 1\,000\,000$ and $> 10\,000$ nonzero terms constructed as a product of three parts: a product of 5 degree-10 polynomials, a product of 3 polynomials of the form $X^r - 1$ with r of order 100 000 and a sparse polynomial of degree 1 000 000 with 40 monomials each multiplied by polynomials of degree 20. Next table gives the timings to compute degree-5, 10, 15, and 20 factors. We observe that the main part of the computation rapidly becomes the gcd computations.

This example can be found in `lacunaryx/bench/bounded_degree_univariate_bench.cpp`. The timings were obtained on an *Intel© Core^{TM} CPU @ 2.60GHz* platform with *7.7GB RAM.*

Degree of the factors	5	10	15	20
Total time (ms)	1994	2924	12190	26165
Non-cyclotomic (ms)	1948	2856	12140	26061
Cyclotomic (ms)	46	68	50	104
Gcd computations (ms)	91	749	9223	23151

References

[1] A. Chattopadhyay, B. Grenet, P. Koiran, N. Portier, and Y. Strozecki. Computing the multilinear factors of lacunary polynomials without heights. Manuscript (submitted), 2013. arXiv:1311.5694.

[2] A. Chattopadhyay, B. Grenet, P. Koiran, N. Portier, and Y. Strozecki. Factoring bivariate lacunary polynomials without heights. In *Proc. ISSAC'13*, pages 141–158, 2013. arXiv:1206.4224.

[3] F. Cucker, P. Koiran, and S. Smale. A polynomial time algorithm for Diophantine equations in one variable. *J. Symb. Comput.*, 27(1):21–30, 1999.

[4] B. Grenet. Bounded-degree factors of lacunary multivariate polynomials, 2014. Manuscript (submitted).

[5] B. Grenet. Computing low-degree factors of lacunary polynomials: a Newton-Puiseux approach. In *Proc. ISSAC'14*, pages 224–231. ACM, 2014.

[6] E. Kaltofen and P. Koiran. On the complexity of factoring bivariate supersparse (lacunary) polynomials. In *Proc. ISSAC'05*, pages 208–215. ACM, 2005.

[7] E. Kaltofen and P. Koiran. Finding small degree factors of multivariate supersparse (lacunary) polynomials over algebraic number fields. In *Proc. ISSAC'06*, pages 162–168. ACM, 2006.

[8] H. Lenstra Jr. On the factorization of lacunary polynomials. In *Number theory in progress*, pages 277–291. De Gruyter, 1999.

[9] J. van der Hoeven et al. Mathemagix, from 2002. `http://www.mathemagix.org`.