Factoring polynomials and 0–1 vectors

Mark van Hoeij
Department of Mathematics
Florida State University
Tallahassee, FL 32306-3027
hoeij@math.fsu.edu

Abstract

A summary is given of an algorithm, published in [4], that uses lattice reduction to handle the combinatorial problem in the factoring algorithm of Zassenhaus. Contrary to Lenstra, Lenstra and Lovász, the lattice reduction is not used to calculate coefficients of a factor but is only used to solve the combinatorial problem, which is a problem with much smaller coefficients and dimension. The factors are then constructed efficiently in the same way as in Zassenhaus' algorithm.

1 Comparison of three factoring algorithms

Let $f \in \mathbb{Q}[x]$ be a polynomial of degree N. Write $f = \sum_{i=0}^{N} a_i x^i$ with $a_i \in \mathbb{Q}$. Suppose that f is monic (i.e. $a_N = 1$) and square-free (i.e. $\gcd(f, f') = 1$). Assume that the coefficients a_i do not have more than D digits.

Let p be a prime number and let $\mathbb{Q}_p$ be the field of p-adic numbers. Let $f = \prod_{i=1}^{n} f_i$ where $f_i \in \mathbb{Q}_p[x]$ are the monic irreducible factors of f in $\mathbb{Q}_p[x]$. Let v be a 0–1 vector, i.e. $v = (v_1 \dots v_n) \in \{0,1\}^n$. Let $g_v = \prod f_i^{v_i}$. Then

$$\{g_v | v \in \{0,1\}^n\}$$

is the set of all monic factors of f in $\mathbb{Q}_p[x]$. Let

$$V = \{v \in \{0,1\}^n | g_v \in \mathbb{Q}[x]\}$$

so $\{g_v | v \in V\}$ is the set of all monic factors of f in $\mathbb{Q}[x]$. Let

$$B = \{v \in V | g_v \text{ irreducible in } \mathbb{Q}[x]\}.$$

Because $\mathbb{Q}[x]$ has unique factorization it follows that V is the set of all $\{0,1\}$-linear combinations of B.

The algorithm of Zassenhaus [1] to factor f in $\mathbb{Q}[x]$ works as follows. First the f_i are computed modulo a power of p using Hensel lifting. The computation time is bounded by $P_Z(N, D)$ which is a polynomial in N and D. Then, for $v \in \{0,1\}^n$, a method is given to decide if $g_v \in \mathbb{Q}[x]$ or not, and if so, to calculate $g_v \in \mathbb{Q}[x]$. If g_v is not in $\mathbb{Q}[x]$, then the time c to verity that is very small, and is almost independent of N, D. For all practical purposes we can assume that this cost c is constant. Calculating g_v for all $v \in B$ gives the set of all irreducible factors of f. But no method is given to calculate the set B, other than to try all v. Denote $|v| = \sum v_i$, so g_v is a product of $|v|$ p-adic factors. First one tries all v with $|v| = 1$, then $|v| = 2$, etc. Whenever a $g_v \in \mathbb{Q}[x]$ is found, the corresponding f_i are removed, and n decreases. The complexity depends on $M = \max\{|v|, v \in B\}$. The worst case is $M = n$, i.e. f is irreducible, because then all 2^n

vectors v will be be tried (or 2^{n-1} vectors, by skipping the complements of the v's that were already tried). So the total cost is at most: $P_Z(N,D) + c2^{n-1}$ where c is a very small constant. If $M = n/2$ then the cost is essentially the same. If $M < n/2$ then the cost is lower, we can bound the cost by $P_Z(N,D) + cE_Z(n,M)$ where $E_Z(n,M) \leq 2^{n-1}$ depends exponentially on M. After trying all $|v| \leq 3$, which can be done quickly, we may assume that $M > 3$ (if there are still any f_i left).

In most examples (even with large N) the number M will be small. Then P_Z dominates the computation time and the algorithm works fast. However, in some examples M can be large, in which case cE_Z can dominate the computation time. This happens when f has few factors in $\mathbb{Q}[x]$ but many factors in $\mathbb{Q}_p[x]$ for every p. Such polynomials have a very special Galois group; order$(\sigma) << N$ for every σ in the Galois group. Extreme examples are the Swinnerton-Dyer polynomials, where order$(\sigma) \leq 2$ for all σ. Other examples are resolvent polynomials, which tend to be polynomials of high degree with small Galois groups. For these polynomials, the computation time is dominated by cE_Z, and the algorithm of Zassenhaus will be exponentially slow.

The first polynomial time algorithm was given by Lenstra, Lenstra and Lovász. In their paper [2] they give a lattice reduction algorithm (the LLL algorithm). Many combinatorial problems can be solved in polynomial time with LLL by encoding the solutions of the problem as short vectors in a lattice. The LLL algorithm can find the set S of short vectors, provided that all vectors outside of span(S) are sufficiently long in comparison.

In [2] the LLL algorithm is used in the following way. Take one factor $f_1 \in \mathbb{Q}_p[x]$ of f. The problem to be solved with LLL is: Find, if it exists, a non-zero $g \in \mathbb{Q}[x]$ of degree $< N$ such that f_1 divides g. If such g exists, then $\gcd(f,g) \in \mathbb{Q}[x]$ is a non-trivial factor of f. This problem is reduced to lattice reduction as follows. First calculate f_1 modulo a sufficiently high power of p by Hensel lifting. The cost of Hensel lifting can be bounded by $P_{L_1}(N,D)$ which is a polynomial in N, D. From that, a lattice can be constructed that (if f is reducible) contains a vector U_g whose entries are the coefficients of g. Then the LLL algorithm can find this vector in a time bounded by $P_{L_2}(N,D)$ which is also a polynomial in N and D. So the total computation time is bounded by $P_L = P_{L_1} + P_{L_2}$, which is a polynomial in N, D. However, $P_{L_1} > P_Z$ because one must Hensel lift up to a substantially higher power of p. Furthermore, $P_{L_2} >> P_Z$, in other words, the algorithm of Zassenhaus is much faster, except when $cE_Z >> P_Z$ which only happens for polynomials with special Galois groups.

So, there exists a polynomial time algorithm [2], and an exponential time algorithm [1] that is faster most of the time (except when M is large). How can the advantages of both algorithms be combined?

Suppose that $g_v \in \mathbb{Q}[x]$. The algorithm in [2] would find such $g = g_v$ by computing a vector U_g in an N-dimensional lattice whose entries are the coefficients of g. Suppose that g is large (say degree five hundred and coefficients with thousands of digits). Lattice reduction is a very general method that can be applied to solve many combinatorial problems. So it is to be expected that if it is used to construct a large expression such as U_g that it will take a long time. To have a faster computation, we must use LLL to construct smaller vectors instead.

The vectors in B are much smaller than the vector U_g, in two ways. The entries have only 1 digit whereas the entries of U_g can have many digits. And the number of entries is only n, usually n is much smaller than N (except for Swinnerton-Dyer polynomials where $n = N/2$, which is smaller but not much smaller than N).

Because of the much smaller size, LLL can calculate the elements of B much faster than the vector U_g. We need to design an input lattice for LLL in such a way, that span(B) and hence B can be obtained from the short vectors found by LLL. To keep the LLL cost to a minimum, we must make sure that the short vectors found by LLL do not contain any information (other than the set B) about the coefficients of a factor g, so that the LLL cost will not depend on the size of g. The LLL cost will then be bounded by a polynomial $P(n)$ that depends only on n, and not on N or D. The cost $cE_Z(n,M)$ of finding the set B in Zassenhaus' algorithm is now replaced by $P(n)$, and the total cost of factoring is now: $P_Z(N,D) + P(n)$. So the resulting algorithm

is faster than Zassenhaus' algorithm whenever $P(n) < cE_Z(n, M)$. It turns out in experiments that the cut-off point is low. That means that when $P(n)$ is not smaller than $cE_Z(n, M)$, then $P(n)$ and $cE_Z(n, M)$ are both small, so then the computation time is close to $P_Z(N, D)$ for both algorithms. However, when n is larger, then $P(n)$ can be much smaller than $cE_Z(n, M)$. Experiments show that polynomials with $n > 400$, $N, D > 2000$ can be handled, which is far beyond the reach of [1, 2].

2 How to construct the lattice to find B?

To find linear conditions on v we can not use the coefficients of the polynomial g_v, because they do not depend linearly on v, $g_{u+v} = g_u g_v$. In order to find linear conditions, a vector $T_A(g)$ with s entries (s will be small compared to n) will be defined, that has the following property: $T_A(g_1 g_2) = T_A(g_1) + T_A(g_2) \in \mathbb{Q}_p^s$ for all non-zero $g_1, g_2 \in \mathbb{Q}_p[x]$. This $T_A(g)$ will be constructed in such a way that the entries of $T_A(g_v)$ are p-adic integers for all 0–1 vectors v, and if furthermore $g_v \in \mathbb{Q}[x]$ then the entries are integers, bounded in absolute value by $\frac{1}{2}p^b$ for some integer b.

Now $T_A(g_v) = \sum v_i T_A(f_i)$ is a linear combination of the $T_A(f_i)$, and when $g_v \in \mathbb{Q}[x]$ then the entries of this linear combination are integers. However, the entries of $T_A(f_i)$ are not yet suitable for use in the LLL input vectors for two reasons. First, these entries of $T_A(f_i)$ are p-adic integers, which are not finite expressions. Second, if $g_v \in \mathbb{Q}[x]$, then the entries of $T_A(g_v)$ are integers bounded by $\frac{1}{2}p^b$, and these integers give some partial information about the coefficients of g_v. It would be inefficient to have this information in the lattice. Both problems are solved by cutting each entry of $T_A(f_i)$ on two sides. If t is such an entry, then the p-adic integer t can be written as $t = \sum_{i=0}^{\infty} t_i p^i$ with t_i integers and $-\frac{p}{2} < t_i \leq \frac{p}{2}$. Choose $a > b$, then cut such t by replacing it with $\sum_{i=b}^{a-1} t_i p^{i-b}$. So the powers $\geq a$ and the powers $< b$ of p in t are removed. The first causes the expression to be finite, and the second removes unnecessary information about the coefficients of g from the lattice. Denote the result of this cutting by $T_A^{b,a}(f_i)$. Let $e_1, \ldots, e_n$ be the standard basis vectors, so $g_{e_i} = f_i$. Denote $V_i = (e_i, T_A^{b,a}(f_i))$ as the concatenation of the vectors e_i and $T_A^{b,a}(f_i)$. The number of entries is $n + s$ which is a little more than n. Denote E_j, $j \in \{1, \ldots, s\}$ as the vector with $n + s$ entries, all 0 except for the $n + j$'th entry which is p^{a-b}.

The $n + s$ vectors V_i and E_j are now the input vectors for LLL. To find the set B, calculate the short vectors in the lattice spanned by the V_i and E_j, then take the projection on the first n entries, and then reduce those vectors to echelon form. The resulting vectors form the set B, provided that a, b and the other parameters in the algorithm were set properly. This can be verified; the polynomials g_v for v in the calculated set B are automatically irreducible if they are in $\mathbb{Q}[x]$, so to verify the correctness of B one only needs to check that $g_v \in \mathbb{Q}[x]$ for all $v \in B$.

The reason that the LLL algorithm lets us find the vectors v for which $g_v \in \mathbb{Q}[x]$ is because for those v, the entries of $T_A(g_v)$ are integers bounded by $\frac{1}{2}p^b$, hence $T_A^{b,a}(g_v) = 0$. Now $\sum v_i T_A^{b,a}(f_i)$ is almost the same as $T_A^{b,a}(g_v) = 0$, except for some round-off errors (caused by cutting the p-adic numbers) which must be of the form $\epsilon_1 + \epsilon_2 p^{a-b}$, where ϵ_1, ϵ_2 are small. The vector $\sum v_i V_i = (v, \sum v_i T_A^{b,a}(f_i))$ is in the lattice. After reducing with the vectors E_i, all entries are small; the first n entries are all 0 or 1, and the last s entries are small as well. So $\sum v_i V_i$ is a short vector. When this vector is found by LLL, v can be read off by taking the first n entries.

There is a lot of freedom in the choice of the numbers p^{a-b} and s so one can choose the size of the LLL input vectors. For efficiency, the size should be not too large, but the size should also not be too small because LLL can only find the short vectors if the remaining vectors are sufficiently long in comparison. The number $s(a - b)\log(p)$ should be $O(n^2)$, and should be independent of N and D.

3 The i'th trace

Definition 1 *The i'th trace $\mathrm{Tr}_i(g)$ of a polynomial g is defined as the sum of the i'th powers of the roots (counted with multiplicity) of g.*

It is clear that

$$\mathrm{Tr}_i(f_1) + \mathrm{Tr}_i(f_2) = \mathrm{Tr}_i(f_1 f_2)$$

for any two polynomials f_1, f_2. Suppose $g = \sum c_i x^i$ is monic of degree d. Then $\mathrm{Tr}_i(g)$ for $i = 1, \ldots, k$ can be determined from c_{d-i} for $i = 1, \ldots, k$ with the Newton relations, and vice versa.

Now choose some $i_1, i_2, \ldots, i_s$, and for $g \in \mathbb{Q}_p[x]$ let $T_A(g)$ be a vector whose entries are the p-adic numbers $\mathrm{Tr}_{i_1}(g), \ldots, \mathrm{Tr}_{i_s}(g)$ multiplied by some integer m. The integer m is chosen in such a way that if g is a factor of f in $\mathbb{Q}[x]$, then the entries of $T_A(g)$ are integers. If f is monic and has integer coefficients then $m = 1$.

The number s is normally chosen much smaller than n or degree(g), so $T_A(g)$ will only contain partial information about the coefficients of g. By computing a bound for the absolute value of the complex roots of f, it is easy to bound $\mathrm{Tr}_i(g)$ for any factor $g \in \mathbb{Q}[x]$ of f, so it is easy to calculate a number b such that the absolute values of the entries of $T_A(g)$ are bounded by $\frac{1}{2}p^b$.

We use $T_A(g)$ to fix the problem that g_v does not depend linearly on v. There is also another way to fix this problem: In [3] Victor Miller uses idempotents to factor in $\mathbb{Q}[x]$. The main difference with our algorithm is that our algorithm is closer to Zassenhaus' algorithm, because only the 0–1 vectors are computed with integer-relation-finding or LLL; the factors g themselves are constructed like in [1]. Millers algorithm is less similar to [1] but is closer to [2] in the sense that everything is computed with integer-relation-finding, more precisely: it calculates the 0–1 vector and the idempotent e simultaneously, and if e is a non-trivial idempotent then $\gcd(e, f) \in \mathbb{Q}[x]$ is a non-trivial factor of f. In our algorithm, besides the 0–1 vector, no information about g is calculated with LLL, because the value of $T_A(g)$ which contains information about g is precisely what is being cut away when all powers $< b$ of p were removed.

References

[1] H. Zassenhaus, *On Hensel Factorization, I.*, Journal of Number Theory **1**, 291–311 (1969).

[2] A.K. Lenstra, H.W. Lenstra and L. Lovász, *Factoring polynomials with rational coefficients*, Math. Ann. **261**, 515–534 (1982).

[3] V. Miller, *Factoring polynomials via Relation-Finding.* ISTCS '92, Springer Lecture Notes in Computer Science **601**, 115–121 (1992).

[4] M. van Hoeij, *Factoring polynomials and the knapsack problem*, preprint available from `http://www.math.fsu.edu/~hoeij/` accepted for Journal of Number Theory (2000).