Factoring polynomials and the knapsack problem.

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

July 11, 2000

Abstract

Although a polynomial time algorithm exists, the most commonly used algorithm for factoring a univariate polynomial f with integer coefficients is the Berlekamp-Zassenhaus algorithm which has a complexity that depends exponentially on n where n is the number of modular factors of f. This exponential time complexity is due to a combinatorial problem; the problem of choosing the right subset of these n factors. In this paper we reduce this combinatorial problem to a knapsack problem of a kind that can be solved with polynomial time algorithms such LLL or PSLQ. The result is a practical algorithm that can factor large polynomials even when n is large as well.

1 Introduction

Let f be a polynomial of degree N with integer coefficients,

$$f = \sum_{i=0}^{N} a_i x^i$$

where $a_i \in \mathbb{Z}$. Assume that f is *monic* (i.e. $a_N = 1$) and that f is square-free (no multiple roots), so the gcd of f and f' equals 1. Let p be a prime number and let $F_p = \mathbb{Z}/(p)$ be the field with p elements. Let $\mathbb{Z}_p$ denote the ring of p-adic integers. If we take a prime number p such that f mod p in $F_p[x]$ is still square-free then one can factor f in $\mathbb{Z}_p[x]$ by factoring f in $F_p[x]$ and applying Hensel lifting.

$$f = f_1 f_2 \cdots f_n$$

Here f_i are monic irreducible polynomials in $\mathbb{Z}_p[x]$. To distinguish between factors in $\mathbb{Z}_p[x]$ and (what we are aiming to find) factors in $\mathbb{Z}[x]$ we will call $f_1, \ldots, f_n$ the *p-adic factors* of f, and factors of f in $\mathbb{Q}[x]$ will be referred to as a *rational factors*. Note that by Gauss' lemma any monic rational factor automatically has integer coefficients. Of course on a finite computer we can only compute *approximations* $\overline{f_i}$ of the p-adic factors f_i with some finite accuracy a, meaning that $\overline{f_i}$ is congruent to f_i modulo p^a. These approximations $\overline{f_i}$ will be called *modular factors*. They are close to f_i in the p-adic valuation norm. The rational factors and p-adic factors (but not the modular factors) divide f in characteristic 0.

Now for every monic rational factor $g \in \mathbb{Z}[x]$ of f there exists a subset of the p-adic factors $S \subset \{f_1, f_2, \ldots, f_n\}$ such that

$$g = \prod_{f_i \in S} f_i.$$

Conversely, if S is a subset of $\{f_1, f_2, \ldots, f_n\}$ then $g = \prod_{f_i \in S} f_i$ is a rational factor of f if and only if $g \in \mathbb{Z}[x]$, so if and only if the coefficients of g (which a priori are p-adic numbers) are integers.

The *combinatorial problem* is now the following: how to find subsets S of

$$\{f_1, f_2, \ldots, f_n\} \tag{1}$$

for which the product of the elements of S has integer coefficients. The Berlekamp-Zassenhaus algorithm [7], implemented in most computer algebra systems, essentially tries all subsets, so the complexity is proportional to 2^n.

Given one such S, one may wonder how a computer can decide if the product g has integer coefficients, considering the fact that only approximations of $f_1, f_2, \ldots, f_n$ with finite accuracy a can be computed. This problem is handled as follows. One computes a bound B, the Landau-Mignotte bound [5], such that one can prove that the coefficients of any rational factor have absolute values $< B$. Then one takes the integer a such that $p^a > 2B$. Denote the *symmetric remainder* of g modulo p^a as the unique polynomial $\overline{g} \in \mathbb{Z}[x]$ such that $\overline{g}$ is congruent to g modulo p^a and the coefficients of $\overline{g}$ are in the interval $(-\frac{1}{2}p^a, \frac{1}{2}p^a]$. This $\overline{g}$ can be computed by multiplying the modular factors $\overline{f_i}$ whenever their accuracy is at least a. Then the following three are equivalent:

1) $g \in \mathbb{Z}[x]$,
2) $g = \overline{g}$
3) $\overline{g}$ divides f

Note that 3) implies that the coefficients of $\overline{g}$ are bounded by B which can be used as a quick test before checking if $\overline{g}$ divides f. For each of the 2^n subsets $S \subset \{f_1, f_2, \ldots, f_n\}$, or 2^{n-1} subsets if one skips complements of sets that have already been tried, a product $\overline{g}$ is computed modulo p^a, and for each of these $\overline{g}$ one must check if $\overline{g}$ divides f. Because there are 2^n subsets, it is clear that the Berlekamp-Zassenhaus algorithm has exponential time complexity, so at first sight one might expect it to be slow. However, in practice it works well because its complexity is not exponential in the degree N, it is only exponential in the number of p-adic factors n which is usually much smaller than N. In fact, in most cases it is much faster than other known algorithms, which is why computer algebra systems use this "exponential time" algorithm. But in some applications, such as resolvent polynomials for Galois group computations, n can indeed be large. For such polynomials this algorithm really does take exponential time.

The first polynomial time algorithm was given by Lenstra, Lenstra and Lovász in [4]. Instead of taking all subsets of (1), they take only 1 element, say f_1, and then determine (if it exists, in other words if f is reducible) a polynomial $g \in \mathbb{Z}[x]$ of degree[1] $N-1$ such that f_1 divides g. Then $\gcd(f,g)$ is a non-trivial rational factor of f because f and g have a non-trivial p-adic factor in common. This way the combinatorial problem and thus the exponential complexity are avoided; instead of trying all combinations of all p-adic factors $f_1, \ldots, f_n$, the construction of a rational factor is based on just one p-adic factor. This idea of constructing a rational factor from one local factor can be used for other kinds of factorization problems as well, even for a non-commutative problem such as differential operators, c.f. [2].

The method given by Lenstra, Lenstra and Lovász to construct a rational factor g is as follows. First they construct a lattice which contains a vector of which the entries are the coefficients of g, and then they apply their *lattice reduction algorithm*, also called *LLL algorithm*. This strategy turned out to have many applications in a wide variety of topics; the LLL algorithm solves many problems that could not be handled before.

In this paper we will reduce the above mentioned combinatorial problem to a different combinatorial problem (see theorem 1), a type of knapsack problem that can be solved by LLL (or by PSLQ). So instead of (as in [4]) using LLL to avoid the combinatorial problem by constructing g in a completely different way, we will use LLL to solve the combinatorial problem, and construct g in the usual way by multiplying a set S of p-adic factors. Instead of using LLL to find the coefficients of g as entries of a short vector in a lattice, we will use LLL to find subsets of (1).

This approach has two advantages. Subsets of (1) can be encoded by a vector of zeros and ones, whereas a vector of coefficients of g can have much larger entries. So the vectors we construct by LLL are much shorter, making it easier to find them. A second advantage is that dimension in the lattice problem in our method is not proportional to N but to n which is usually much smaller. These advantages mean that cost of the lattice reduction in our method depends only on n, it does not depend on N, nor on the size of the coefficients of f. This results in a better complexity and much better computation timings.

2 Some experiments in Maple

Let $h \in \mathbb{Z}[x]$ be the monic irreducible polynomial

$$\begin{aligned} h = \; & x^{20} - 5x^{18} + 864x^{15} - 375x^{14} - 2160x^{13} + 1875x^{12} + 10800x^{11} \\ & +186624x^{10} - 54000x^9 + 46875x^8 + 270000x^7 - 234375x^6 \\ & -2700000x^5 - 1953125x^2 + 9765625. \end{aligned}$$

[1] this is slightly different from the original algorithm

Suppose α_i, $i = 1, 2, \ldots, 20$ are the roots of h. One can calculate the monic square-free polynomial $f \in \mathbb{Z}[x]$ that has the following roots: $\alpha_i + \alpha_j$, $1 \leq i < j \leq 20$. The degree of f is $20 \cdot 19/2 = 190$ and the coefficients have up to 89 digits. The action of the Galois group G of h on $\{\alpha_1, \ldots, \alpha_{20}\}$ induces an action on the set $A = \{\{\alpha_i, \alpha_j\} | i \neq j\}$. The number of orbits of G in A equals the number of irreducible factors of f, in particular f is irreducible if and only if the action of G on A is transitive (when G is a 2-transitive group). The lengths of these orbits are the degrees of the irreducible factors of f.

So the factorization of f yields information about the Galois group of h. The Galois group G of this polynomial h acts on the roots $\alpha_1, \ldots, \alpha_{20}$ as A_6 acts on the 20 subsets with 3 elements of $\{1, 2, 3, 4, 5, 6\}$. Knowing this, it can quickly be determined that f has 3 factors, one of degree 10 and two of degree 90. Of course we can not use this information in the factorization algorithm because we want to do the converse, factoring f in order to obtain information on Galois group of h.

Since $G \simeq A_6$ has no elements of order > 5 there can not be irreducible p-adic factors of degree > 5. Therefore, at any prime p there are at least $190/5 = 38$ p-adic factors. When applying Maple's implementation of Berlekamp-Zassenhaus on f the following things happen. First a number of primes are tried to see which has the fewest (i.e. 38) p-adic factors. It can choose for example $p = 19$. Using various primes it also constructs a set of possible degrees of rational factors, although in this particular example this does not help because this set is $\{0, 5, 10, \ldots, 190\}$. Then, using Hensel lifting, it calculates the p-adic factors $f_1, \ldots, f_{38}$ up to accuracy a, where a is a positive integer that depends on the bound it computes for the coefficients of rational factors of f. Because f has large coefficients, this bound and hence p^a are large as well ($a = 2^7$ and p^a has 164 digits, so the $\overline{f_i}$ have up to 164 digits as well). So the Hensel lifting takes some time, about 50 seconds on a Pentium 266. After that it tries to find rational factors by computing products of s p-adic factors. First $s = 1$, then $s = 2$, $s = 3$, etc. At $s = 1$ there are 38 cases to check, and no rational factor is found. For $s = 2$ there are $38 \cdot 37/2$ cases to check. This is still not a large number of cases so $s = 1$ and $s = 2$ take little time. It takes less than 2 seconds (after Hensel lifting) to find the rational factor of degree 10 (which consists of 2 p-adic factors of degree 5). Then these two p-adic factors can be removed from the list, leaving 36 p-adic factors, which means there remain 2^{36-1} cases to check. On a Pentium 266, Maple would take years to do this (although this computation time can be reduced a lot by avoiding many unnecessary multiplications of modular factors, c.f. [1, 3]). This example illustrates the strength as well as the weakness of the Berlekamp-Zassenhaus factorization method, it can quickly find rational factors that consist of few p-adic factors, but it is exponentially slow in finding rational factors that consist of many p-adic factors. When n is large it is a good strategy to use Berlekamp-Zassenhaus to determine rational factors that consist of few p-adic factors, and (if the factorization is not yet completed) to use the knapsack factorization algorithm presented in this paper to determine the remaining rational factors. On this example our experimental implementation takes about 200 seconds to factor f, of which 50 seconds is spent on Hensel lifting, 20 seconds on searching small factors with Berlekamp-Zassenhaus (combinations of three p-adic factors were tried as well), and most of the remaining time is spent on lattice reduction. One of the factors of degree 90 has coefficients of up to 38 digits, the other has coefficients up to 46 digits. We note that in order to be able to find these coefficients by computing a short vector in a lattice as in [4], the remaining vectors would need to be even larger, making the lattice reduction extremely hard.

A preliminary implementation of the knapsack factorization algorithm can be found on `http://www.math.fsu.edu/~hoeij/knapsack.html` where some additional examples are given as well. One polynomial has degree $N = 192$, two rational factors and $n = 96$ p-adic factors. Factorization takes about 17000 seconds, Hensel lifting took 180 seconds. Another polynomial has degree $N = 351$ and $n = 61$. Factorization took 3000 seconds, half of which was spent on Hensel lifting. For the lattice reduction we used Storjohann's Maple implementation, called lattice(L,integer), which appears to be good because we were unable to improve timings by implementing PSLQ in Maple. Lattice reduction dominates the computation time unless N is much larger than n or f has very large coefficients, in which case Hensel lifting dominates the computation time. It appears experimentally that the time spent on lattice reduction is about $O(n^6)$ which is reasonably close to the complexity estimates of lattice reduction (note that the lengths of the input vectors must depend on n because of the factor $2^{n/2}$ between the length of the vector that LLL can find and the length of the shortest vector). The polynomial P7 on Paul Zimmerman's website, for which the factorization was not yet known, took about 8000 seconds and is irreducible according to our preliminary implementation.

3 The knapsack factorization algorithm

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 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 monic polynomials f_1, f_2. Consider the polynomial

$$g = (x - x_1)(x - x_2) \cdots (x - x_d).$$

We can write g as

$$g = x^d + (-1)^1 E_1 x^{d-1} + (-1)^2 E_2 x^{d-2} + \cdots + (-1)^d E_d x^0$$

where $E_i(x_1, \ldots, x_d)$ is the i'th *elementary symmetric polynomial* in the variables $x_1, \ldots, x_d$. The i'th *power polynomial* $P_i(x_1, \ldots, x_d)$ is defined as $x_1^i + x_2^i + \ldots x_d^i$. Note that

$$\mathrm{Tr}_i(g) = P_i.$$

It is a classical result in invariant theory that

$$\mathbb{Q}[E_1, \ldots, E_d] = \mathbb{Q}[P_1, \ldots, P_d]$$

in other words $E_1, \ldots, E_d$ can be expressed as polynomials of $P_1, \ldots, P_d$ and vice versa. For example, $E_1 = P_1$ and $E_2 = (P_1^2 - P_2)/2$. As a consequence

Lemma 1 *A monic polynomial g of degree d has rational numbers as coefficients if and only if* $\mathrm{Tr}_i(g) \in \mathbb{Q}$ *for all* $i \in \{1, \ldots, d\}$.

We can use the first of the following two relations (the Newton identities)

$$P_i = -iE_i - \sum_{k=1}^{i-1} P_k E_{i-k} \qquad iE_i = -P_i - \sum_{k=1}^{i-1} P_k E_{i-k} \tag{2}$$

to recursively compute $P_i = \mathrm{Tr}_i(g)$ for any monic polynomial g and any positive integer i. Note that the second relation can be used to calculate the coefficients $(-1)^i E_i$ of g from the $\mathrm{Tr}_i(g)$. However, we will not need this because once we find the right set S of p-adic factors we can also calculate g by multiplication. Both conversions, E_i to P_i and vice versa, can be done very quickly, especially since we are computing modulo p^a.

Let $\mathrm{Tr}_{1..d} = (\mathrm{Tr}_1, \ldots, \mathrm{Tr}_d)^T$, so

$$\mathrm{Tr}_{1..d}(g) = \begin{pmatrix} \mathrm{Tr}_1(g) \\ \mathrm{Tr}_2(g) \\ \vdots \\ \mathrm{Tr}_d(g) \end{pmatrix}.$$

Lemma 2 *Let f be a monic polynomial of degree N in $\mathbb{Z}[x]$ and let $d = \lfloor N/2 \rfloor$, the largest integer $\leq N/2$. Let F be a field of characteristic 0, such as the field of p-adic numbers. Then for any monic factor $g \in F[x]$ of f the following are equivalent*

1. $g \in \mathbb{Z}[x]$
2. $\mathrm{Tr}_{1..d}(g) \in \mathbb{Z}^d$
3. $\mathrm{Tr}_{1..d}(g) \in \mathbb{Q}^d$

Proof: The Newton identities show that 1) implies 2). It is clear that 2) implies 3). Now assume 3). If the degree of g is $\leq d$ then g must be in $\mathbb{Q}[x]$ because of lemma 1. Then 1) follows from Gauss' lemma. Now assume that the degree of g is larger than d and assume 3). Take $h = f/g$, so $f = gh$. Since $f \in \mathbb{Z}[x]$ we have $\mathrm{Tr}_{1..d}(f) \in \mathbb{Z}^d$ and by 3) we have $\mathrm{Tr}_{1..d}(g) \in \mathbb{Q}^d$. Then $\mathrm{Tr}_{1..d}(h)$ must also be in $\mathbb{Q}^d$ because $\mathrm{Tr}_{1..d}(f) = \mathrm{Tr}_{1..d}(g) + \mathrm{Tr}_{1..d}(h)$. Now we can apply the lemma on h, so $h \in \mathbb{Z}[x]$ and hence $g = f/h$ is in $\mathbb{Z}[x]$ as well.

We will now weaken the lemma, instead of necessary and sufficient conditions for $g \in \mathbb{Z}[x]$ we will now consider just necessary conditions. Let s be a positive integer and let A be an s by d matrix with integer entries. Then denote

$$T_A = A\mathrm{Tr}_{1..d} \qquad T_A(g) = A \begin{pmatrix} \mathrm{Tr}_1(g) \\ \mathrm{Tr}_2(g) \\ \vdots \\ \mathrm{Tr}_d(g) \end{pmatrix}$$

where $d = \lfloor N/2 \rfloor$. From lemma 2 it follows that

$$g \in \mathbb{Z}[x] \Longrightarrow T_A(g) \in \mathbb{Z}^s \tag{3}$$

and we have "if and only if" in equation (3) when A is the identity matrix or more generally, when A has rank $d = \lfloor N/2 \rfloor$.

For efficiency reasons we will use matrices A with fewer rows, and, as long as this is necessary, gradually increase s, the number of rows of A. Note that for a random dense matrix A, even when $s = 1$, it is very probable to have "if and only if" in equation (3) because f has only finitely many monic factors $g \in \mathbb{Z}_p[x]$, namely 2^n.

Let S be a subset of the p-adic factors and let g be the product of S. Then

$$T_A(g) = \sum_{f_i \in S} T_A(f_i).$$

So a necessary condition for g to be a rational factor is that the sum of the $T_A(f_i)$, $f_i \in S$ has integer entries. However, how can this be decided considering that the $T_A(f_i)$ can only be determined up to some finite accuracy a? (this a is the same as the accuracy of $\overline{f_i}$ because no divisions are needed to compute $T_A(f_i)$ from f_i). This question is similar to a problem in the Berlekamp-Zassenhaus algorithm (see section 1), and will be handled in a similar way. If B is a bound on the absolute value of the complex roots of f then dB^i is a bound for $|\mathrm{Tr}_i(g)|$ for any rational factor g of f of degree d. Bounds for the entries of $T_A(g)$ can be computed from this. One can calculate the symmetric remainder modulo p^a of $\sum_{f_i \in S} T_A(f_i)$, and a necessary condition for $g \in \mathbb{Z}[x]$ is that this symmetric remainder satisfies the bound in each row (different rows may have different bounds).

If $g \in \mathbb{Z}[x]$, then from the symmetric remainder of $T_A(g)$ we could say something about the coefficients of g. In particular, if A is the identity matrix then all coefficients of g can be computed quickly from $T_A(g)$ using the Newton identities (2). However, this will not be needed because g can also be computed by multiplying the f_i in S. Because of that, there is no good reason to compute the precise[2] value (modulo a power of p) of $T_A(g)$, all we need to know is if it satisfies the bound or not. For this purpose T_A^b will be defined below.

Compute a bound B_i for the i'th entry of T_A, i.e. the absolute value of the i'th entry of $T_A(g)$ must be $< B_i$ for any rational factor g of f. Choose a list of positive integers $b = (b_1 \cdots b_s)$ such that $B_i < \frac{1}{2}p^{b_i}$. Then define

[2] in the sense of the usual absolute value norm, not the p-adic valuation norm

Definition 2 *For any monic polynomial $g \in \mathbb{Z}_p[x]$ define $T_A^b(g) \in \mathbb{Z}_p^s$ as follows. Let r be the i'th entry of $T_A(g)$. Let t be the symmetric remainder of r modulo p^{b_i}. Then $r - t$ is divisible by p^{b_i}, so $u = (r-t)/p^{b_i}$ is a p-adic integer. Then the i'th entry of $T_A^b(g)$ is defined as u.*

If g is a rational factor of f then the i'th entry of $T_A(g)$ is bounded by B_i, hence smaller in absolute value than $\frac{1}{2}p^{b_i}$, and so it equals to its symmetric remainder modulo p^{b_i}. Then $T_A^b(g)$ will be zero, which proves the first part of the following lemma.

Lemma 3 *Let $g \in \mathbb{Z}_p[x]$ be a monic factor of f. Then*

$$g \in \mathbb{Z}[x] \Longrightarrow T_A^b(g) = 0.$$

Furthermore, if $\mathrm{rank}(A) = \lfloor N/2 \rfloor$ *then the converse implication holds as well.*

Proof: Assume $\mathrm{rank}(A) = \lfloor N/2 \rfloor$. The entries of $T_A^b(g)$ can only be zero if the entries of $T_A(g)$ (which a priori are p-adic integers) are integers. The $\mathrm{Tr}_i(g)$ can be determined from $T_A(g)$ when A has full rank. So they must be rational numbers. Then $g \in \mathbb{Z}[x]$ because of lemma 2.

We note that if f_j was approximated with accuracy a then the i'th entry of $T_A^b(f_j)$ can be computed modulo p^{a-b_i}. So a should be greater than b_i, in particular a needs to be larger than $\log(2B_i)/\log(p)$. If not, more Hensel lifting is required. In practice the value of a (as determined by the bound used in the Berlekamp-Zassenhaus algorithm) is much larger than $\log(2B_i)/\log(p)$. Only when Tr_i's are used with i close to $N/2$, which seems unlikely to be necessary, could there be a small chance that additional Hensel lifting is needed.

When a is large, which is usually the case, we will normally take the b_i larger than necessary for the condition $B_i < \frac{1}{2}p^{b_i}$ to hold. The reason is that the input vectors for lattice reduction have length proportional to p^{a-b_i}. For good performance of the algorithm we want these $a - b_i$ to be not too big (also not too small), see also the comments in section 3.3.

The main difference between T_A and T_A^b is the following. Recall that $T_A(g)$ gives some partial (or complete, if A has full rank) information on the coefficients of a rational factor g. That information has been cut away in the definition of T_A^b; everything that is smaller than the bound B_i has been rounded off to 0. Additivity is lost due to this round off, $T_A^b(f_1 f_2)$ need not be equal to $T_A^b(f_1) + T_A^b(f_2)$. But T_A^b is still almost additive.

Lemma 4 *Let S be a subset of $\{f_1, \ldots, f_n\}$ and let g be the product of the elements of S. Then*

$$T_A^b(g) = \epsilon + \sum_{f_i \in S} T_A^b(f_i) \tag{4}$$

where $\epsilon = (\epsilon_1 \cdots \epsilon_s)^T \in \mathbb{Z}^s$. Furthermore,

$$|\epsilon_i| \leq \frac{|S|}{2}$$

where $|\epsilon_i|$ denotes the absolute value of ϵ_i and $|S|$ denotes the number of elements of S.

Proof: Notations as in definition 2, t is the symmetric remainder of r mod p^{b_i} and $u = (r-t)/p^{b_i}$. It is the difference between r/p^{b_i} and u that causes T_A^b to be no longer additive, and this difference is t/p^{b_i} which is a rational number with absolute value $< 1/2$, if $p \neq 2$. And the only possible denominator is a power of p. With $|S|+1$ polynomials (the f_i and g), these differences add up to a rational number $\epsilon_i \in \mathbb{Q}$ with absolute value $< (|S|+1)/2$. Now ϵ is the difference of $T_A^b(g)$ and the sum of the $T_A^b(f_i)$ with f_i in S, and all of these have p-adic integers as entries. So ϵ_i must also be a p-adic integer, as well as a bounded rational number with only a power p as a possible denominator, so it must be an integer. Its absolute value is $< (|S|+1)/2$, hence $\leq |S|/2$ and the lemma follows.
If $p = 2$ we have $|t/p^{b_i}| \leq 1/2$ instead of $< 1/2$. However, the lemma still holds because $\epsilon_i = (|S|+1)/2$ is still not possible, because all the t/p^{b_i} would need to be equal to $1/2$. But then we would have $-1/2 = \epsilon_i + |S| \cdot (-1/2)$ so $\epsilon_i = (|S|-1)/2$ and the lemma still holds.

Lemma 5 *Let S be a subset of $\{f_1, \ldots, f_n\}$ and let g be the product of S. If $g \in \mathbb{Z}[x]$ then*

$$\epsilon + \sum_{f_i \in S} T_A^b(f_i) = 0 \tag{5}$$

*for some vector $\epsilon \in \mathbb{Z}^s$ with entries bounded in absolute value by $|S|/2$.
If $\mathrm{rank}(A) = \lfloor N/2 \rfloor$ then the converse holds as well.*

Proof: If $g \in \mathbb{Z}[x]$ then $T_A^b(g) = 0$ by lemma 3, so the result follows from lemma 4. Conversely, if equation (5) holds, then by lemma 4 it follows that there is a vector $\epsilon' \in \mathbb{Z}^s$ such that $T_A^b(g) = \epsilon' + \sum T_A^b(f_i)$, which equals $\epsilon' - \epsilon \in \mathbb{Z}^s$. So the entries of $T_A^b(g)$, which are a priori p-adic integers, are integers. Then $T_A(g)$ must be in $\mathbb{Z}^s$ as well. If $\mathrm{rank}(A) = \lfloor N/2 \rfloor$ then the $\mathrm{Tr}_i(g)$ can be computed by from $T_A(g)$ and must be in $\mathbb{Q}$. Then $g \in \mathbb{Z}[x]$ by lemma 2.

For each p-adic factor f_j we can compute

$$T_A^{b,a}(f_j) \in \mathbb{Z}^s$$

which is defined as the approximation of $T_A^b(f_j)$; the i'th entry must be in the interval $(\frac{1}{2}p^{a-b_i}, \frac{1}{2}p^{a-b_i}]$ and must have accuracy $a - b_i$. So the i'th entry of $T_A^{b,a}(f_j) - T_A^b(f_j)$ is congruent to 0 modulo p^{a-b_i}. The computational cost for determining $T_A^{b,a}(f_j)$ is small compared to the cost of Hensel lifting up to accuracy a. We can now reformulate lemma 5 as follows. Let $e_1, \ldots, e_s$ be the standard basis of $\mathbb{Z}^s$ so we can write $\epsilon = \sum_i \epsilon_i e_i$. Let E_i be $p^{a-b_i} e_i$.

Theorem 1 The factorization knapsack problem. *Let f be a monic polynomial in $\mathbb{Z}[x]$ and $f_1, \ldots, f_n$ the irreducible p-adic factors. For every $S \subset \{f_1, \ldots, f_n\}$, if the product g of the elements of S is a rational factor of f then*

$$\sum_{i=1}^{s} \epsilon_i e_i + \sum_{i=1}^{s} \gamma_i E_i + \sum_{i=1}^{n} \delta_i^S T_A^{b,a}(f_i) = 0 \tag{6}$$

where the ϵ_i and γ_i are integers with absolute value at most $|S|/2$ and where

$$\delta_i^S = \begin{cases} 1 & \text{if } f_i \in S \\ 0 & \text{otherwise.} \end{cases}$$

The proof that the γ_i are bounded integers is similar to the proof that the ϵ_i are bounded integers in lemma 5. However, this bound is not needed when using LLL to solve the knapsack problem in equation (6).

Equation (6) in the theorem converges (in the p-adic valuation norm) to equation (5) in lemma 5 when a tends to infinity (and the b_i are kept constant). This implies that if $\text{rank}(A) = \lfloor N/2 \rfloor$ then for sufficiently large a, just like in lemma 5, the converse of the theorem holds as well.

3.1 The knapsack lattice

Let

$$M = \sqrt{C^2 N + s(N/2)^2}$$

where C is a positive integer chosen in such a way that neither one of the two terms under the square root is much larger than the other one. To solve the knapsack-like problem given in equation (6) we will construct a lattice L such that the vector

$$v_S = (C\delta_1^S \ \cdots C\delta_n^S \quad \epsilon_1 \cdots \epsilon_s)$$

is an element of L for every solution S of equation (6). A vector v in L will be called M-short if $v \neq 0$ and the length $|v|$ of v is $\leq M$. Note that v_S is M-short for every solution S of the knapsack problem (6),

$$|v_S|^2 \leq C^2|S| + s(|S|/2)^2 \leq M^2.$$

Since $S = \{1, \ldots, n\}$, which corresponds to the trivial factor $g = f$, is always a solution of (6), it follows that L always contains at least one M-short vector.

Denote W as the span of all vectors v_S for which S is a solution of equation (6). Our objective is a lattice L such that

1. All M-short vectors in L are in W.

2. All vectors in L that are not M-short are large enough such that the reduced basis, computed by the LLL algorithm, has the following properties

   (a) At least one M-short vector appears in the reduced basis.

(b) Proposition 1.12 combined with remark 1.14 in [4] can be used to show that there are no M-short vectors outside the span of the M-short vectors found in the reduced basis.

If all vectors outside of W have length greater than $M2^{(n+s)/2}$ then these conditions hold and LLL is guaranteed to find a basis of W. If the shortest vector outside of W is shorter than $M2^{(n+s)/2}$ then we still have a chance that conditions 2a and 2b hold, but if that vector is much shorter (say around $M2^{(n+s)/8}$) then it is unlikely that LLL finds any M-short vectors (even though we know that at least one exists).

We take the following basis for lattice L

$$B_L = \{(Ce_i, -T_A^{b,a}(f_i))|1 \leq i \leq n\} \bigcup \{(0, E_i)|1 \leq i \leq s\}$$

where (v_1, v_2) with $v_1 \in \mathbb{Z}^n$, $v_2 \in \mathbb{Z}^s$ stands for the row vector obtained by converting (taking the transpose) the column vectors v_1 and v_2 to row vectors and them concatenating them. These $n+s$ vectors are linearly independent in $\mathbb{Z}^{n+s}$, they form a triangular system with determinant $C^n p^{(a-b_1)+\cdots+(a-b_s)}$. If the determinant is too small then condition 2 above will not hold and we can not expect to find any M-short vectors with lattice reduction.

When condition 2a holds, i.e. when an M-short vector was found, then it is very often true that all M-short vectors were found. But condition 2b still needs to be checked, which involves a sufficiently accurate floating point Gram-Schmidt computation. This is necessary to prove that the factors we find (or f itself) are irreducible; if 2b shows that there can not be more M-short vectors then there can not be more solutions S of the knapsack problem than the ones that were found, and so there can not be more rational factors of f.

3.2 The algorithm

Input: A monic square-free polynomial $F \in \mathbb{Z}[x]$.
Output: All irreducible factors in $\mathbb{Z}[x]$.

Step 1 Apply the Berlekamp-Zassenhaus algorithm but search only for those rational factors that consist of few p-adic factors. Whenever a rational factor is found, remove the corresponding p-adic factors from the list.

Step 2 Let $f_1, \ldots, f_n$ be the remaining p-adic factors and let f be the polynomial F divided by the rational factors that were found in step 1. If n is small, say $n < 15$, then use Berlekamp-Zassenhaus to do the rest of the factorization as well.

Step 3 At this point n is not small, the p-adic factors $f_1, \ldots, f_n$ have been computed modulo p^a where the prime p was chosen to minimize n, and a was chosen using the Landau-Mignotte bound. Now the knapsack factorization algorithm begins.

Step 4a Let B_L be a list of row vectors $e_1, \ldots, e_n$.

Step 4b Pick a positive integer C as in section 3.1 so that neither of the two terms in $M = \sqrt{C^2 N + s(N/2)^2}$ dominates. Then multiply all elements of B_L by C. This step is optional, one may also take $C = 1$.

Step 5 Let N be the degree of f.
Let $d = \lfloor N/2 \rfloor$
Let $s = 1$ and let $s_{\text{old}} = 0$.
Let A be the s by d matrix $A = (1\ 0\ 0 \cdots 0)$.
Compute the bound B_1 (see section 3).
Choose the integer $b_1 > \log(2B)/\log(p)$ in such a way that $a - b_1$ is a positive integer close to $0.022n^2/\log(p)$. If this is not possible (a is too small) then do additional Hensel lifting to increase a.

Step 6 (Extending B_L) Do steps 6a, 6b and 6c for k from $s_{\text{old}} + 1$ to s

Step 6a For each f_i let t_i be the k'th entry of $T_A^{b,a}(f_i)$.

Step 6b For each row vector v in the basis B_L, add one extra entry at the end of v, namely the symmetric remainder of $v_1 t_1 + \cdots v_n t_n$ modulo p^{a-b_k}. This increases the number of entries in each row from $n + k - 1$ to $n + k$

Step 6c Add the vector $(0 \cdots 0\ p^{a-b_k})$ to the list B_L. This increases the number of row vectors in B_L to $n + k$. The determinant is now $C^n p^{(a-b_1)+\cdots+(a-b_k)}$.

Step 7 Apply lattice reduction on B_L.

Step 8 Take all M-short vectors $v_1, \ldots, v_r$ of B_L. If $r = 0$ proceed to step 12. Remove from the v_i the entries in columns $n + 1, \ldots, n + s$. Then divide the v_i by C and then compute the reduced row echelon form R of the matrix formed by these row vectors v_i. If R does not have precisely one 1 in each column, with the rest 0, then proceed to step 12. So there must be precisely n non-zero entries in matrix R, all of which must be 1, and there must be one non-zero entry in each column. If $r = 1$ then proceed with step 9, if $r > 1$ then go to step 10.

Step 9 ($r = 1$) Check if condition 2b in section 3.1 holds. If so, then f is irreducible and the algorithm terminates. If not, proceed to step 12.

Step 10 ($r > 1$) Do step 10a for k from 1 to r.

Step 10a Let v be the r'th row of R. Let S be the set of those f_i for which the i'th entry of v is 1. Let g_r be the symmetric remainder modulo p^a of the product of S. If g_r does not divide f then proceed to step 12.

Step 11 Check if condition 2b in section 3.1 holds. If so, then $g_1, \ldots, g_r$ are the irreducible factors of f and the algorithm terminates. If the condition does not hold, then irreducibility can not be guaranteed so we need to apply recursion on $g_1, \ldots, g_r$.

Step 12 Let $s_{\text{old}} = s$. Increase s, and add $s - s_{\text{old}}$ rows to A. If $s < d$ then determine a value for b_i for each of these new rows in the same way as in step 5. However, if $s \geq d$ then keep increasing the values used for $a - b_i$ (may require more Hensel lifting) every time step 12 is reached. If $s \leq d$ then choose the new rows linearly independent of the previous rows, so that the rank of A is s.

3.3 Remarks on the algorithm

1. If the polynomial f is not monic, $a_N \neq 1$, we need to make two changes in the algorithm. If $g \in \mathbb{Q}[x]$ is a monic rational factor of f then the coefficients of g are no longer automatically integers. As a consequence, $\mathrm{Tr}_i(g)$ is no longer in $\mathbb{Z}$, which is something that the algorithm uses. However, $a_N^i \mathrm{Tr}_i(g) \in \mathbb{Z}$, so in the non-monic case $\mathrm{Tr}_i(g)$ needs to be replaced by $a_N^i \mathrm{Tr}_i(g)$. The second change is in step 10a. When we compute the product of S, it first needs to be multiplied by a_N before we compute the symmetric remainder g_r, just like the non-monic case in the Berlekamp-Zassenhaus algorithm. And we also need to divide g_r by the gcd of its coefficients as well.

2. In step 8, if condition 2b in section 3.1 holds, i.e. if it can be proven by a Gram-Schmidt computation that no additional M-short vectors exist outside the span of $v_1, \ldots, v_r$, then we do not necessarily need to go straight to step 12 whenever anything goes wrong. Instead, we can simply throw away all vectors from B_L except for $v_1, \ldots, v_r$ because we do not lose any solutions of the knapsack problem. The entries in column numbers $n+1, \ldots, n+s$ of $v_1, \ldots, v_r$ are replaced by zeros because these have already served their purpose. After that we proceed to step 12, but with a much smaller set B_L, which should improve efficiency.

   This case typically happens when matrix A has few non-zero columns, for example $A = (1\ 0 \cdots 0)$, because then the converse of equation (3) in section 3 no longer holds. Then the converse of theorem 1 does not hold either. In other words the knapsack problem will have too many solutions, i.e. not all solutions correspond to rational factors. However, theorem 1 is still true, the 0–1-vector of every rational factor g will still be in the span of $v_1, \ldots, v_r$. This way something is gained every time that M-short vectors were found and condition 2b holds. If r is small then we may also stop doing additional lattice reductions and just try out all 0–1-vectors that are in the span of the rows of matrix R.

3. If we start with matrix $A = (1\ 0\ 0 \cdots 0)$ then it is unlikely that a factorization will be obtained at $s = 1$, in other words we will end up in step 12. But this "wasted" lattice reduction costs little time because we used a small determinant and if we use remark 2 it may in the end speed up the overall computation time. Remark 2 is not yet implemented, so perhaps the timings mentioned in section 2 can be improved.

After $s = 1$ fails and step 12 is reached, we jump the value of s up to 5 and use non-zero entries in the first 10 columns of A. This way, with the choice of the number 0.022 in the algorithm, there appeared to be a fairly good chance of success in experiments. If $s = 5$ fails as well (one example so far) then s is increased by small amounts.

4. In all our tests we never needed a value of s anywhere near d. And the accuracy a that Maple chose for Hensel lifting in the Berlekamp-Zassenhaus algorithm was considerably higher than what our implementation needed to determine the right sets S. But we probably do need that value for a for computing g_r in step 10a.

5. It is not clear yet what the best choices for s and $a - b_i$ are. We tried starting with larger s and smaller $a - b_i$ (replacing the 0.022 with a smaller number), or smaller s and larger $a - b_i$. This did not appear to have much impact on the computation time. Even when we take $s = 1$, start with A having random integer entries in the first 10 columns, and replace 0.022 by 0.11 to keep the same determinant, we still get similar timings, even though the lengths of the input vectors are much higher this way.

6. We could also consider computing the factors f_j not in $\mathbb{Z}_p[x]$ but in $\mathbb{R}[x]$. Then compute the $Tr_i(f_j)$, cut away the integer part, and construct a knapsack problem in the same way. The disadvantage is that in $\mathbb{R}[x]$ there are at least $N/2$ factors f_j.

References

[1] J. Abbott, V. Shoup, P. Zimmermann, *Factorization in $\mathbb{Z}[x]$: The Searching Phase*, to appear in ISSAC'2000 Proceedings, available from `http://www.shoup.net/papers/`

[2] M. van Hoeij, *Factorization of Differential Operators with Rational Functions Coefficients*, J. Symb. Comput., 24, 537–561 (1997).

[3] E. Kaltofen, *Polynomial factorization.* In Computer Algebra, B. Buchber et all, Ed., 2 ed. Springer Verlag, pp. 95–113 (1982).

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

[5] M. Mignotte, *An inequality about factors of polynomials*, Math. of Computation, **28**, 128, 1153–1157 (1974).

[6] A. Storjohann, *Faster Algorithms for Integer Lattice Basis Reduction.* Technical Report 249, Departement Informatik, ETH Zurich, (1996).

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