Factoring polynomials and the knapsack problem.

Mark van Hoeij

Department of Mathematics
Florida State University
Tallahassee, FL 32306-3027, USA
E-mail: hoeij@math.fsu.edu

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 subsets of these n factors. In this paper this combinatorial problem is reduced to a type of knapsack problem that can be solved with polynomial time algorithms such LLL or PSLQ. The result is a practical algorithm that can factor polynomials that are far out of reach for any previous algorithm.

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 square-free (no multiple roots), so the gcd of f and f' equals 1. Until section 2.3 we will also assume that f is *monic* (i.e. $a_N = 1$). Let p be a prime number and let $\mathbf{F}_p = \mathbb{Z}/(p)$ 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 $\mathbf{F}_p[x]$ is still square-free then one can factor f in $\mathbb{Z}_p[x]$ by factoring f in $\mathbf{F}_p[x]$ 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* $\overline{f_i} \in \mathbb{Z}[x]$ 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 $S \subseteq \{f_1, f_2, \ldots, f_n\}$ 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 the subsets S of

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

for which the product of the elements of S is a polynomial with integer coefficients. The Berlekamp-Zassenhaus algorithm [8], 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 [6], 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 $S \subseteq \{f_1, f_2, \ldots, f_n\}$, 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 such as [5], 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 [5]. 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 factorization problems as well, even for a non-commutative case such as differential operators, c.f. [3].

The method given by Lenstra, Lenstra and Lovász to construct 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 2.1), a type of knapsack problem that can be solved by LLL (or by PSLQ). So instead of (as in [5]) 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. A subset S of (1) can be encoded by a 0–1 vector, 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 the 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

[1] this is slightly different from the original algorithm

the coefficients of f. This results in a better complexity and much better computation timings.

I would like to thank Arne Storjohann for pointing out a problem in a previous version of this paper.

2. THE KNAPSACK FACTORIZATION ALGORITHM

DEFINITION 2.1. 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. 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 2.1. *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 2.2. *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 2.1. 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. ▮

Every monic factor g in $\mathbb{Z}_p[x]$ of f can be encoded by a *0–1 vector* $v = (v_1 \cdots v_n)$ with $v_i \in \{0, 1\}$ as follows

$$g = \prod_{i=1}^{n} f_i^{v_i}. \tag{3}$$

In lemma 2.2 the values of the v_i were restricted to 0 and 1, but a similar result can be given for any values of $v_1, \ldots, v_n \in \mathbb{Z}$:

LEMMA 2.3. *Let $V_i = \mathrm{Tr}_{1..N}(f_i) \in \mathbb{Q}_p^N$. These vectors $V_1, \ldots, V_n$ are linearly independent over $\mathbb{Q}_p$. Furthermore if $v_1, \ldots, v_n \in \mathbb{Z}$ and we take $g = \prod_{i=1}^{n} f_i^{v_i}$ and $V = \sum_{i=1}^{n} v_i V_i$ then*

$$g \in \mathbb{Q}(x) \Longleftrightarrow V \in \mathbb{Q}^N \Longleftrightarrow V \in \mathbb{Z}^N \tag{4}$$

Proof. Let $\alpha_1, \ldots, \alpha_N$ be the roots of $f = f_1 \cdots f_n$ in the algebraic closure. Then the $\mathrm{Tr}_{1..N}(x - \alpha_j) = (\alpha_j^1, \ldots, \alpha_j^N)^T$ are linearly independent because these vectors divided by α_j form a Vandermonde matrix. The vectors V_i are sums of disjoint sets of the $\mathrm{Tr}_{1..N}(x - \alpha_j)$ and are therefore linearly independent as well.

We can define Tr_i for a quotient of polynomials $g = g_1/g_2$ as $\mathrm{Tr}_i(g_1) - \mathrm{Tr}_i(g_2)$. Then $V = \mathrm{Tr}_{1..N}(g)$ so if $g \in \mathbb{Q}(x)$ then V must also be defined over $\mathbb{Q}$, in fact V must be in $\mathbb{Z}^N$ because the α_j are algebraic integers (we assumed $f \in \mathbb{Z}[x]$ to be monic). Conversely, if $V \in \mathbb{Q}^N$ then g must also be defined over $\mathbb{Q}$ because

$$\mathrm{Tr}_{1..N} : \{\prod_{i=1}^{n} f_i^{v_i} | v_1, \ldots, v_n \in \mathbb{Z}\} \to \mathbb{Z}V_1 + \cdots + \mathbb{Z}V_n$$

is one to one due to the linear independence of the V_i. ∎

In the same way one sees that $\mathrm{Tr}_{1..\infty}$ is a one to one homomorphism from $\mathbb{C}(x)^*/\mathbb{C}^*$ to $\mathbb{C}^{\mathbb{N}}$ that commutes with the action of the Galois group of $\mathbb{C}$ over $\mathbb{Q}$.

We will now weaken lemma 2.2, instead of necessary and sufficient conditions for $g \in \mathbb{Z}[x]$ we will now consider just necessary conditions. Let s and d be positive integers and let A be an s by d matrix with integer entries. Then denote

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

It follows from lemma 2.2 that

LEMMA 2.4. *If $g \in \mathbb{Z}_p[x]$ is a monic factor of f then*

$$g \in \mathbb{Z}[x] \Longrightarrow T_A(g) \in \mathbb{Z}^s. \tag{5}$$

If the following condition holds: "the rowspace of A contains the first $\lfloor N/2 \rfloor$ standard basis elements $(1\ 0 \cdots 0)$, $(0\ 1\ 0 \cdots 0), \ldots$" *then the converse of statement (5) holds as well.*

See remark 2. in section 2.3 for comments on the choice of matrix A. 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_0 is a bound on the absolute value of the complex roots of f then dB_0^i is a bound for $|\mathrm{Tr}_i(g)|$ for any rational factor g of f of degree $\leq d$. 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 the condition in lemma 2.4 holds 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 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

Definition 2.2. 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 $\overline{r}$ be the symmetric remainder of r modulo p^{b_i}. Then $r - \overline{r}$ is divisible by p^{b_i}, so $u = (r - \overline{r})/p^{b_i}$ 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 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 2.5. *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 the condition in lemma 2.4 holds then the converse implication is true as well.

Proof. Assume the condition in lemma 2.4 holds. 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 $\mathrm{Tr}_i(g)$, $1 \leq i \leq N/2$ can be determined from $T_A(g)$ when the condition on A in lemma 2.4 holds. So these $\mathrm{Tr}_i(g)$ must be rational numbers. Then $g \in \mathbb{Z}[x]$ because of lemma 2.2. ▮

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 order to increase a. If we take the value a in the same as in the Berlekamp-Zassenhaus algorithm then additional Hensel lifting is rarely needed. In fact, for large irreducible polynomials, a smaller value for a than what is needed in Berlekamp-Zassenhaus often suffices to prove irreducibility. This way, to compute an irreducibility proof for a polynomial, one may reduce the amount of Hensel lifting which is worthwhile because Hensel lifting dominates the memory usage and can also dominate the computation time.

The main difference between T_A and T_A^b is the following. Recall that $T_A(g)$ gives some partial (or complete, if the condition in lemma 2.4 holds) 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 2.6. *Let S be a subset of $\{f_1, \ldots, f_n\}$ and let g be the product of the elements of S. Then*

$$T_A^b(g) = \epsilon + \sum_{f_i \in S} T_A^b(f_i) \tag{6}$$

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 2.2, $\overline{r}$ is the symmetric remainder of r mod p^{b_i} and $u = (r - \overline{r})/p^{b_i}$. 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 $\overline{r}/p^{b_i}$ 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 $|\overline{r}/p^{b_i}| \leq 1/2$ instead of $< 1/2$. However, the lemma still holds because $\epsilon_i = (|S|+1)/2$ is still not possible, because all the $\overline{r}/p^{b_i}$ 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 2.7. *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*

$$\epsilon + \sum_{f_i \in S} T_A^b(f_i) = 0 \tag{7}$$

for some vector $\epsilon \in \mathbb{Z}^s$ with entries bounded in absolute value by $|S|/2$.
If the condition in lemma 2.4 holds then the converse is true as well.

Proof. If $g \in \mathbb{Z}[x]$ then $T_A^b(g) = 0$ by lemma 2.5, so the result follows from lemma 2.6. Conversely, if equation (7) holds, then by lemma 2.6 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 the condition in lemma 2.4 holds then the $\mathrm{Tr}_i(g)$ can be computed from $T_A(g)$ and must be in $\mathbb{Q}$. Then $g \in \mathbb{Z}[x]$ by lemma 2.2. ▌

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 2.7 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 2.1. **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 $S \subseteq \{f_1, \ldots, f_n\}$, if the product g of the elements of S is a rational factor of f then*

$$\sum_{i=1}^{s} \epsilon_i e_i + \sum_{i=1}^{s} \gamma_i E_i + \sum_{i=1}^{n} v_i T_A^{b,a}(f_i) = 0 \tag{8}$$

for some integers ϵ_i and γ_i with absolute value at most $|S|/2$, and where

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

The proof that the γ_i are bounded integers is similar to the proof that the ϵ_i are bounded integers in lemma 2.7. However, this bound is not needed when using LLL to solve the knapsack problem in equation (8).

Equation (8) in the theorem converges (in the p-adic valuation norm) to equation (7) in lemma 2.7 when a tends to infinity (and the b_i are kept constant). This implies that if the condition in lemma 2.4 holds then for sufficiently large a, just like in lemma 2.7, the converse of the theorem holds as well.

2.1. The knapsack lattice

Denote $L_{\mathbb{Q}}$ as the set of all $v = (v_1 \cdots v_n) \in \mathbb{Z}^n$ for which $\prod f_i^{v_i}$ is defined over $\mathbb{Q}$. Every 0–1 vector $(v_1 \cdots v_n) \in L_{\mathbb{Q}}$ satisfies equation (8) for every matrix A and every accuracy a. Note that if $g_1, \ldots, g_r$ are the irreducible monic factors of f in $\mathbb{Z}[x]$ then there is a basis $w_1, \ldots, w_r$ of $L_{\mathbb{Q}}$ where w_k is defined as the 0–1 vector $(v_1 \cdots v_n)$ for which $g_k = \prod f_i^{v_i}$. If any basis for $L_{\mathbb{Q}}$ is known, we can construct a matrix with the elements of that basis as rows and compute the reduced row echelon form R of that matrix. Then R has $w_1, \ldots, w_r$ as rows. In particular R has the following properties:

A. Each column contains precisely one 1, and all other entries are 0.
B. If $(v_1 \cdots v_n)$ is a row of this matrix then $g = \prod f_i^{v_i} \in \mathbb{Z}[x]$.

Suppose we have a lattice $L \subseteq \mathbb{Z}^n$ such that $L_{\mathbb{Q}} \subseteq L$. Note that we can check if L equals $L_{\mathbb{Q}}$ because if $L_{\mathbb{Q}} \subseteq L$ then $L_{\mathbb{Q}} = L$ if and only if L satisfies conditions A and B. Condition B can be checked like in the Berlekamp-Zassenhaus algorithm; by computing the symmetric remainder of the product modulo p^a and checking if the result divides f. Initially we take $L = \mathbb{Z}^n$ to be sure that L contains $L_{\mathbb{Q}}$. Suppose L is not equal to $L_{\mathbb{Q}}$. The goal is to calculate a lattice L' such that

$$L_{\mathbb{Q}} \subseteq L' \subseteq L.$$

Then we can replace L by L'. This is repeated until $L = L_{\mathbb{Q}}$, when all irreducible monic factors $g_1, \ldots, g_r \in \mathbb{Z}[x]$ of f can be computed like in condition B.

Choose an s by d matrix A and accuracy a. We want to compute a lattice $L' \subseteq L$, hopefully of smaller dimension than L, that contains all solutions $(v_1 \cdots v_n)$ of equation (8). If $\dim(L') = \dim(L)$ then we need to use a different matrix A and try again, or if that does not work we need a higher value for a (i.e. more Hensel lifting). Eventually this must be successful (see end of this section), and we find L' with smaller dimension. Then replace L by L', check if $L_{\mathbb{Q}} = L$, and after finitely many steps the algorithm is done.

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. Let B_L be a basis for L. Initially $L = \mathbb{Z}^n$ and B_L is the standard basis. To solve the knapsack-like problem given in equation (8) we will construct the *knapsack lattice*, a lattice Λ such that the vector

$$v_S = (Cv_1 \ \cdots Cv_n \quad \epsilon_1 \cdots \epsilon_s)$$

is an element of Λ for every solution S of equation (8). A vector v in Λ 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 (8),

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

The trivial factor f of f corresponds to the set $S = \{1, \ldots, n\}$, and so this is always a solution of (8). Hence Λ always contains at least one M-short vector.

Let 0_n be the n-dimensional zero row vector, let E_i as in theorem 2.1, E_i^T is its transpose, and let the s-dimensional row vector T_{f_i} be the transpose of $-T_A^{b,a}(f_i)$. We take the following basis $B_E \bigcup B_T$ for lattice $\Lambda \subseteq \mathbb{Z}^{n+s}$ where

$$B_E = \{(0_n \ E_i^T) | 1 \leq i \leq s\}$$

$$B_T = \{vm | v \in B_L\} \ \text{ where } m = \begin{pmatrix} C & 0 & \cdots & 0 & T_{f_1} \\ 0 & C & \cdots & 0 & T_{f_2} \\ \vdots & \vdots & & \vdots & \vdots \\ 0 & 0 & \cdots & C & T_{f_n} \end{pmatrix}$$

and we reduce the elements of B_T modulo the elements of B_E. When $L = \mathbb{Z}^n$ then B_L has n elements and $B_E \bigcup B_T$ has $n + s$ elements. These elements are linearly independent, they form a triangular system with determinant

$$D = C^n p^{(a-b_1)+\cdots+(a-b_s)}.$$

In general the number l of basis elements for Λ equals s plus the number of elements of B_L. We can use LLL to compute a reduced basis $V_1, \ldots, V_l$ of Λ. Let $W_1, \ldots, W_l$ be the Gram-Schmidt basis for $V_1, \ldots, V_l$. We can compute approximations $\overline{W}_i$ of W_i by floating point arithmetic. Let r be the smallest integer such that $|\overline{W}_k| > M'$ for all $r < k \leq l$, where M' is M plus a bound on the round-off errors in the $\overline{W}_i$. So $|W_k| > M$ for all $k > r$. It follows from the proof of (1.11) in [5] that all M-short vectors in Λ must then be in the span of $V_1, \ldots, V_r$. So we can throw away the vectors $V_{r+1}, \ldots, V_l$. If we did not reduce the basis of Λ with the LLL algorithm then it is nearly certain that $r = l$ so we can not throw away any vectors. If the the number $p^{(a-b_1)+\cdots+(a-b_s)}$ is too small then one can expect $r = l$ as well, see remark 3. in section 2.3. If $r < l$ then we can divide these $V_1, \ldots, V_r$ by C, take the projection on the first n coordinates and take L' as the lattice spanned by that. It follows from theorem 2.1 that L' contains $L_{\mathbb{Q}}$ so

$$L_{\mathbb{Q}} \subseteq L' \subseteq L.$$

Suppose $L' = L$ for arbitrarily high values of a. This is only possible when $T_A(\prod f_i^{v_i}) \in \mathbb{Z}^s$ for every $(v_1 \cdots v_n) \in L$, in other words the elements of L already satisfy the conditions on the Tr_i specified by A. This can happen for example in the following situation: Suppose that in the first step we use only Tr_1 in A, i.e. $A = (1)$, we compute $L' \subseteq \mathbb{Z}^n$, and we replace L by L'. Suppose that in the second step the only new Tr_i we use is Tr_2, for example $A = (0\ 1)$ or A is the 2 by 2 identity matrix. It could be possible to construct a polynomial f such that for all g's for which $\mathrm{Tr}_1(g) \in \mathbb{Z}$ one also has $\mathrm{Tr}_2(g) \in \mathbb{Z}$. If this happens then L' will not be a proper subset of L no matter how large a is because all g's obtained from L already satisfy the new condition given in A, i.e. $\mathrm{Tr}_2(g) \in \mathbb{Z}$. In this situation the basis vectors B_T, reduced modulo B_E, are already short without using any lattice reduction so this situation can be detected quickly. In such a situation little computation time has been lost and we can simply proceed with the next matrix A, using the next Tr_i's.

If $\dim(L') = \dim(L)$ for arbitrarily high a for a collection of matrices A that cover all Tr_i, $i \leq N$ then $L_{\mathbb{Q}} = L$ because of lemma 2.3. So whenever $L_{\mathbb{Q}} \neq L$ there must exist matrices A and positive integers a such that $\dim(L') < \dim(L)$. If we keep using more Tr_i's and increase a we must find eventually find such L'. This implies that the algorithm terminates in finitely many steps.

2.2. The algorithm

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

Step 1 Apply the Berlekamp-Zassenhaus algorithm but search only for those rational factors that consist of at most three 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 then use Berlekamp-Zassenhaus to do the rest of the work 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 4 Let $B_L = \{e_1, \ldots, e_n\}$, the standard basis for $\mathbb{Z}^n$.

Step 5 Choose a matrix A, see also remark 2. in section 2.3.
Compute an upper bound B_i for the i'th entry of $T_A(g)$ for any rational factor g. Choose the integers $b_i > \log(2B_i)/\log(p)$, see also remark 3. for this choice. If $a \leq b_i$ then do additional Hensel lifting to increase a.

Step 6 Compute the basis $B_E \bigcup B_T$ for lattice Λ like in section 2.1. If the elements of B_T, reduced modulo the elements of B_E, are already short then go back to step 5 and choose a different matrix A.

Step 7 Apply the LLL algorithm to compute a reduced basis $V_1, \ldots, V_l$ for Λ. Do a floating point Gram-Schmidt computation (see section 2.1) to determine an as small as possible integer r such that all M-short vectors in Λ are in $\mathbb{Z}V_1 + \cdots + \mathbb{Z}V_r$. Then compute the new basis B_L as in section 2.1.

Step 8 Let r be the number of elements of the new B_L. Note that $\dim(L_{\mathbb{Q}}) \leq n/4$ because all rational factors consisting of < 4 p-adic factors have been removed in step 1.
If $r = 1$ then f must be irreducible and the computation ends.
If $r > n/4$ then go to step 5, otherwise proceed to step 9.

Step 9 Take a matrix with the elements of B_L as rows and compute the reduced row echelon form R. If R does not satisfy condition A) in section 2.1 then go to step 5, otherwise proceed to step 10.

Step 10 Check condition B) in section 2.1 as follows. Do step 10a for $k \in \{1, \ldots, r\}$.

Step 10a Let $(v_1 \cdots v_n)$ be the k'th row of R. Let g_k be the symmetric remainder of $\prod f_i^{v_i}$ modulo p^a. If g_k does not divide f then go back to step 5.

Step 11 At this point all g_k, $k \in \{1, \ldots, r\}$ divide f. Then $g_1, \ldots, g_r$ are the monic irreducible factors of f in $\mathbb{Z}[x]$ and $f = g_1 \cdots g_r$.

2.3. Remarks on the algorithm

1. In **step 1** we should try all combinations of 1, 2 and 3 p-adic factors because that takes little time. Four is on the edge, it may or may not be worthwhile to check combinations of four p-adic factors. Combining five p-adic factors is too much, at that point it is most likely better to proceed to step 3 unless n is small (say $n < 15$) because then 2^{n-1} is so small that it is best to use only Berlekamp-Zassenhaus.

2. There are different strategies for choosing matrix A in **step 5**. The first is to take $d = s$ and to take A as the s by s identity matrix. Then $T_A = \mathrm{Tr}_{1..d}$. A second strategy is to take $d > s$ and to use small random integers as entries. This way several Tr_i's can be combined in one entry of $T_A(g)$. If we take a small integer s, take $d = \lfloor N/2 \rfloor$ (usually a much smaller d will do) and take a random s by d matrix A then it is very likely that the converse of statement (5) in lemma 2.4 will hold because f has only finitely many monic factors $g \in \mathbb{Z}_p[x]$, namely 2^n. The advantage of the second strategy is that the number of rows that we need is small because several Tr_i's are combined in one row of A.

We use a variation on the first strategy, starting with a very small value for $s = d$, so using only few Tr_i's. Then for each next lattice reduction we use only few new Tr_i's at the same time. This way we keep decreasing the set of g's with $T_A(g) \in \mathbb{Z}^s$. The advantage of this approach is that it tends to lead to faster lattice reductions because we do not try to solve everything in one lattice reduction.

3. We measure the *amount of information* that is put into the lattice Λ by the number

$$I = ((a - b_1) + \cdots + (a - b_s))\log(p)$$

This number is roughly the logarithm of the determinant D in section 2.1 (the factor C^n is ignored). The computation time of one lattice reduction depends mainly on the following two quantities: the number of vectors (in the first step this is $n + s$ where s is much smaller than n) and the number I. We found experimentally that increasing I a factor 2 causes the lattice reduction to take roughly 6 times longer.

The value for I can be chosen. To decrease I, we can do one of the following: decrease[2] a, increase the b_i, or decrease s. To increase I we can: choose b_i as small as the bound on the i'th entry of $T_A(g)$ allows, use larger s, or in some rare cases we need to increase a, meaning more Hensel lifting.

If we use strategy 1 for the choice of matrix A, where we aim to find $L_{\mathbb{Q}}$ in a single lattice reduction using a matrix that combines several Tr_i's in each row, then a reasonable value for I is about $0.12n^2$. This gives a good

[2]it is not necessary to use the same value for a for each row of A.

chance to be ready in one step, and in the cases that one is not ready after one lattice reduction it is likely that not much information needs to be added in the second step.

If we use strategy 2 for the choice of matrix A, where we try to reduce the number of elements of B_L more gradually, processing only part of the information (few Tr_i's) at the same time, then we can make do with a smaller value for I, leading to faster lattice reductions. Suppose we can reduce the number of elements of B_L from 100 down to 5 in one step with strategy 1, but only from 100 down to 65 with strategy 2 using only $\mathrm{Tr}_{1..3}$ and say a quarter of the computation time due to the smaller value for I. Then the latter is better because in the next lattice reduction we only have $65+s$ vectors which takes much less time than lattice reducing 100 vectors. However, if of the original 100 vectors there are still > 90 left in step 7 then that is not good progress and we should use higher values for I. In fact, if I is too small then we may even make no progress (B_L remains the same) and LLL may not find any M-short vectors even though we know that at least one must exist (from the trivial factor $g = f$).

Note that when using different values for a and the b_i it is possible to put different information in Λ using the same Tr_i's, i.e. using the same matrix A. It may be useful to use the same matrix A twice in this way, the first time to try to reduce the number of elements of B_L and the second time as a quick test to make sure that the new B_L satisfies the conditions given by these Tr_i's. Another option is to add rows to A in each step (keeping the rows that were already there) instead of replacing A in each step.

It is possible to put in the same amount of information I while using shorter input vectors in the following way: replace the i'th entry of each $T_A(f_j)$ by two entries, one using a, c instead of a, b_i and one using c, b_i instead of a, b_i, where c is an integer between a and b_i. Judging from the complexity estimates for lattice reduction this seems better because the input vectors are shorter, but our implementation does not make use of it because it appears to make no noticeable difference in practice; it seems as if all that counts is n and I.

The initial choice for I in our algorithm depends just on n. So the cost of the first lattice reduction does not depend on N nor on the size of the coefficients of f. The cost of the second lattice reduction depends on the number of vectors remaining after the first lattice reduction, which depends on f.

4. 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. After computing the product, it first needs to be multiplied by a_N before taking the symmetric remainder g_k, just like the non-monic case in the Berlekamp-Zassenhaus algorithm. And we need to divide g_k by the gcd of its coefficients as well.

5. We could also consider computing the factors f_j in $\mathbb{R}[x]$ instead of $\mathbb{Z}_p[x]$. Then compute the $\mathrm{Tr}_i(f_j)$, cut away the integer part, and construct a knapsack problem in a similar way. The disadvantage is that in $\mathbb{R}[x]$ there are many (at least $N/2$) factors f_j.

6. One could try to use a higher value for y in the LLL algorithm [5], so that LLL performs a stronger lattice reduction, and then use a smaller value for I. Or one could use PSLQ instead of LLL. We were unable to improve timings by implementing PSLQ in Maple, although a C implementation of PSLQ or LLL will most likely give better results.

3. SOME EXPERIMENTS

3.1. A polynomial with Galois group A_6

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

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

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 Galois group G of h acts on the roots of f as well. The number of orbits equals the number of irreducible factors of f and the lengths of these orbits are the degrees of the irreducible factors of f. So the factorization of f yields some information on G.

The Galois group G acts on the roots $\alpha_1, \ldots, \alpha_{20}$ of h 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 when 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. Maple's implementation of Berlekamp-Zassenhaus does the following with f. 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, by 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, 4]). 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 (which is the reason for doing step 1 in section 2.2), but it is exponentially slow in finding rational factors that consist of many p-adic factors.

On this example our implementation takes 152 seconds to factor f, of which 50 seconds is spent on Hensel lifting, 8 seconds on searching factors consisting of < 4 p-adic factors with Berlekamp-Zassenhaus, and most of the remaining time is spent on lattice reductions. The first lattice reduction uses $\mathrm{Tr}_{1..2}$, takes about 70 seconds, and after that 16 vectors remain. The second lattice reduction uses $\mathrm{Tr}_{3..4}$, takes about 22 seconds, and 2 vectors remain. To construct the factors corresponding to these two vectors takes about 2 seconds. One of the factors of degree 90 has coefficients of up to 38 digits, the other has coefficients up to 46 digits. Note that to find these coefficients by computing a short vector in a lattice as in [5], the remaining vectors would need to be even larger, so the input vectors would need to be very large making the lattice reduction very hard.

3.2. A polynomial with Galois group M_{12}

Let

$$\begin{aligned} f \;=\; & 3x^{12} + 100x^{11} + 1350x^{10} + 9300x^9 + 32925x^8 + 45000x^7 - 43500x^6 \\ & -147000x^5 + 46125x^4 + 172500x^3 + 22500x + 1875. \end{aligned}$$

Let F_k be the polynomial of minimal degree such that the product of every set of k roots of f is a root of F_k. The polynomials F_5 and F_6 are square-free and have degrees $12!/(5!7!) = 792$ and $12!/(6!6!) = 924$. The polynomial f has Galois group M_{12} if and only if F_5 is irreducible and F_6 is reducible. For F_5 there are at least $n = 792/11 = 72$ p-adic factors and for F_6 there are at least $n = 924/11 = 84$ p-adic factors, because M_{12} does not have elements of order > 11. This makes factorization of F_5 or F_6 far out of reach of any previous algorithm.

Due to the high degree and very large size of the coefficients of F_5 and F_6 the computation time is dominated by Hensel lifting, which takes 81% of the 15 hours to factor F_6 on a Sun workstation (corresponds to around 30 hours on a Pentium 266). Considerable time and memory usage could be saved, however, because F_5 can still be proven to be irreducible even if we use a smaller value for a than prescribed by the Landau-Mignotte bound. And the smallest of the two factors of F_6 could also be computed with smaller a.

The polynomial P7 on Paul Zimmerman's web-site, which was apparently also out of reach for previous algorithms because the factorization was unknown, took only one hour on a Pentium 266 and is irreducible. Another polynomial with $n = 128$, degree $N = 256$, and 2 irreducible factors of degree 128 took about 7 hours to factor, most of which was spent on lattice reductions. Memory usage was dominated by Hensel lifting in all examples we tried. An implementation of the knapsack factorization algorithm and additional examples can be found on

`http://www.math.fsu.edu/~hoeij/knapsack.html`

REFERENCES

1. J. Abbott, V. Shoup and P. Zimmermann, *Factorization in* $\mathbb{Z}[x]$*: The Searching Phase*, to appear in ISSAC'2000 Proceedings. `http://www.shoup.net/papers/`
2. H. R. P. Ferguson, D. H. Bailey and S. Arno, *Analysis of PSLQ, An Integer Relation Finding Algorithm*, manuscript, (1995). `http://www.cecm.sfu.ca/organics/papers/bailey/`
3. M. van Hoeij, *Factorization of Differential Operators with Rational Functions Coefficients*, J. Symb. Comput., 24, 537–561 (1997).
4. E. Kaltofen, *Polynomial factorization.* In Computer Algebra, B. Buchber et all, Ed., 2 ed. Springer Verlag, pp. 95–113 (1982).
5. A.K. Lenstra, H.W. Lenstra and L. Lovász, *Factoring polynomials with rational coefficients*, Math. Ann. 261 (1982), 515-534
6. M. Mignotte, *An inequality about factors of polynomials*, Math. of Computation, **28**, 128, 1153–1157 (1974).
7. A. Storjohann, *Faster Algorithms for Integer Lattice Basis Reduction.* Technical Report 249, Departement Informatik, ETH Zurich, (1996).
8. H. Zassenhaus, *On Hensel Factorization, I.*, Journal of Number Theory **1**, 291–311 (1969).