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

For several decades the standard algorithm for factoring polynomials f with rational coefficients has been the Berlekamp-Zassenhaus algorithm. The complexity of this algorithm 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 lattice reduction algorithms. The result is a practical algorithm that can factor polynomials that are far out of reach for previous algorithms.

The presented solution to the combinatorial problem is different from previous lattice based factorizers; these algorithms avoided the combinatorial problem by solving the entire factorization problem with lattice reduction. This led to lattices of large dimension and coefficients, and thus poor performance. This is why lattice based algorithms, despite their polynomial time complexity, did not replace Berlekamp-Zassenhaus as the standard method. That is now changing; new versions of computer algebra systems such as Maple, Magma, NTL and Pari have already switched to the algorithm presented here.

1. INTRODUCTION

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

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

where $c_i \in \mathbb{Z}$. 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. $c_N = 1$). 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

* Partially supported by NSF grant DMS-9805983.

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 mod p 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* $\mathcal{C}^a(f_i) \in \mathbb{Z}[x]$ of the p-adic factors f_i with some finite accuracy a. These approximations $\mathcal{C}^a(f_i)$ are called the *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.

Definition 1.1. Let $c \in \mathbb{Z}_p$ be a p-adic integer and let $0 \leq b \leq a$ be integers. The *symmetric remainder* $\mathcal{C}^a(c)$ of c modulo p^a is the unique integer $-p^a/2 < \mathcal{C}^a(c) \leq p^a/2$ that is congruent to c modulo p^a.
Now define $\mathcal{C}^a_b(c)$ as $\mathcal{C}^{a-b}((c - \mathcal{C}^b(c))/p^b)$. The definitions extend to polynomials by applying $\mathcal{C}^a$ or $\mathcal{C}^a_b$ to each coefficient. Now $\mathcal{C}^a(c)$ is also called an *approximation* of c with *accuracy* a, and $\mathcal{C}^a_b(c)$ is called a two-sided *cut* of c.

If one thinks of c as an infinite power series in p then $\mathcal{C}^a(c)$ is what remains after removing the i'th powers of p for all $i \geq a$. To find $\mathcal{C}^a_b(c)$, remove i'th powers of p for all $i \geq a$ as well as $i < b$, and divide by p^b.

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 [14], 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, which is enough to find $\mathcal{C}^a(g)$ but not enough to find g. This problem is handled as follows (see [14]). One computes a bound B_{lm}, the Landau-Mignotte bound [7], such that one can prove that the coefficients of any rational factor have absolute values $< B_{\mathrm{lm}}$. Then take the integer a such that $p^a > 2B_{\mathrm{lm}}$. Then the following three are equivalent:

1) $g \in \mathbb{Z}[x]$,
2) $g = \mathcal{C}^a(g)$
3) $\mathcal{C}^a(g)$ divides f in $\mathbb{Z}[x]$

Note that 3) implies that the coefficients of $\mathcal{C}^a(g)$ are bounded by B_{lm}. 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, one has to test if the product $\mathcal{C}^a(g)$ modulo p^a divides f in $\mathbb{Z}[x]$. Using ideas from [1, 4], in particular section 3.1.1 in [1], testing one such S can be done in a nearly constant, extremely small, amount of time. However, because the number of subsets to be tested is exponentially large, it is clear that the Berlekamp-Zassenhaus algorithm has exponential time complexity. So at first sight one might expect it to be slow. But in practice it works well because the complexity is not exponential in the degree N, it is only exponential in the number of p-adic factors n, a number that is usually much smaller than N. In fact, it is almost always much faster than other algorithms such as [6] and (see below) variations on [6], 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 [6]. 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 just 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

[1]this is slightly different from the original algorithm

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.

The method of [6] essentially computes the minimum polynomial over $\mathbb{Q}$ of a p-adic root (a root of f_1). There exist several variations, all of which are polynomial time. One may replace a p-adic root by a complex root [10], and/or replace LLL by PSLQ [2]. An interesting improvement is given in [8], instead of one root, all complex roots are used, to compute not a factor but an idempotent e. If e is a non-trivial idempotent then $\gcd(e, f)$ is a non-trivial factor of f in $\mathbb{Z}[x]$.

Multiplying modular factors is a very fast way to construct rational factors. But in (variations of) [6], this construction is replaced by something else, which explains why [14] performs much better in practice.

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 lattice reduction algorithms such as [5, 6, 10, 11, 12, 13]. So instead of (as in [6]) 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 same efficient way as in [14], by multiplying a set S of p-adic factors. Lattice reduction will only be used to find subsets of (1), and not to construct any other information about g (such as coefficients of g).

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 the coefficients of f.

I would like to thank Victor Shoup, Allan Steel, Arne Storjohann, Paul Zimmermann and the referees for finding errors in a previous version of this paper and for providing valuable suggestions and examples.

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.

This is also called the i'th Newton sum (the name i'th trace refers to the fact that if g is irreducible then $\mathrm{Tr}_i(g)$ is the trace of α^i, taken over the field extension given by a root α 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

$$g = x^d + \tilde{E}_1 x^{d-1} + \tilde{E}_2 x^{d-2} + \cdots + \tilde{E}_d x^0$$

where $\tilde{E}_i = (-1)^i E_i$ and $E_i = E_i(x_1, \ldots, x_d)$ 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)

$$P_i = -i\tilde{E}_i - \sum_{k=1}^{i-1} P_k \tilde{E}_{i-k} \qquad i\tilde{E}_i = -P_i - \sum_{k=1}^{i-1} P_k \tilde{E}_{i-k} \tag{2}$$

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 $\tilde{E}_i = (-1)^i E_i$ 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, $\tilde{E}_i$ 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. *Assume that* 0 *is not a root of f. 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*

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

Proof. Let $\alpha_1, \ldots, \alpha_N$ be the roots of $f = f_1 \cdots f_n$ in the algebraic closure of $\mathbb{Q}_p$. 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 (the Galois group over $\mathbb{Q}$ leaves V invariant, hence it permutes the pre-images of V, so the fact that the map is 1-1 implies that g is invariant and hence defined 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

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

The purpose of matrix A is that in the algorithm, instead of using a vector $\mathrm{Tr}_{1..d}(g)$ with d entries (d could be very large) we will use a vector $T_A(g)$ with a small number of entries (s will be small in comparison to n).

If during the computation it turns out that the converse of statement (5) below does not hold then the algorithm will increase s, thereby using more Tr_i's in T_A. This way the algorithm will always reach a point where the converse of (5) will hold. For some polynomials this point will already be reached with the first matrix A, for other polynomials it will take more steps. In the implementation we start with a very small value of s and a sparse matrix A, hoping that more than one step is needed before the converse of (5) will hold. That turns out to be beneficial for the performance as it tends to lead to easier lattice reductions.

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 row space 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.*

Proof. If $\mathrm{Tr}_{1..d}(g) \in \mathbb{Z}^d$ then $A\mathrm{Tr}_{1..d}(g) \in \mathbb{Z}^s$. Conversely, if the condition on the row space of A holds, then $T_A(g) \in \mathbb{Z}^s$ implies $\mathrm{Tr}_{1..d}(g) \in \mathbb{Q}^d$ and so the lemma follows from lemma 2.2. ▮

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 question is similar to a problem in the Berlekamp-Zassenhaus algorithm (see section 1), and will be handled in a similar way. If B_{rt} is a bound on the absolute value of the complex roots of f then dB_{rt}^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. Given S, one can calculate $\mathcal{C}^a(T_A(g))$, 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 $\mathcal{C}^a(T_A(g))$ we could say something about the coefficients of g; if for example $T_A = \mathrm{Tr}_{1..s}$ then the first s coefficients of g can be computed 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 $B_i < \frac{1}{2}p^{b_i}$ (it is not necessary to take the smallest possible 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 A satisfies the condition in lemma 2.4 then the converse implication is true as well.

Proof. Assume that A satisfies the condition in lemma 2.4. 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 $\mathrm{Tr}_i(g)$, $1 \le i \le N/2$ can be determined from $T_A(g)$ when A satisfies the condition in lemma 2.4. 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 to be 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 often dominate the computation time as well.

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 on A 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,

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

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 $p \ne 2$. 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 A satisfies the condition in lemma 2.4 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 $T_A^b(g) = \epsilon' + \sum T_A^b(f_i) = \epsilon' - \epsilon \in \mathbb{Z}^s$. 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 A satisfies the condition in lemma 2.4 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. ▮

Choose integers a_i such that $b_i < a_i$. Let $\overline{c}_{j,i}$ be the i'th entry of $T_A(f_j)$ and let $\tilde{c}_{j,i}$ be the i'th entry of $T_A^b(f_j)$. Now let

$$c_{j,i} = \mathcal{C}_{b_i}^{a_i}(\overline{c}_{j,i}) = \mathcal{C}^{a_i - b_i}(\tilde{c}_{j,i}) \tag{8}$$

and let $\mathcal{C}_j \in \mathbb{Z}^s$ be defined as

$$\mathcal{C}_j = (c_{j,1} \cdots c_{j,s}).$$

So the i'th entry of $\mathcal{C}_j$ is an approximation of the i'th entry of $T_A^b(f_j)$ with accuracy $a_i - b_i$, and is a two-sided cut of the i-th entry of $T_A(f_j)$. It can be calculated from $\mathcal{C}^a(f_j)$ provided that $a \geq a_i$.

The computational cost for determining $\mathcal{C}_j$ 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 $\mathbb{Z}^s$. Note that, although column notation was used for $T_A(f_j)$, we will use row notation for the vectors $\mathcal{C}_j$ and e_i.

THEOREM 2.1. **The factorization knapsack problem**. *Let f be a monic squarefree 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*

$$\sum_{i=1}^{s}(\epsilon_i + \gamma_i p^{a_i - b_i})e_i + \sum_{i=1}^{n} v_i \mathcal{C}_i = 0 \tag{9}$$

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 (9).

Equation (9) in the theorem converges (in the p-adic valuation norm) to equation (7) in lemma 2.7 when the a_i tend to infinity and the b_i are kept constant. This implies that if A satisfies the condition in lemma 2.4 then for sufficiently large a_i the converse of the theorem holds as well.

2.1. The knapsack lattice

Denote W 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 $\mathbb{Q}$. Note that if $g_1, \ldots, g_r$ are the irreducible monic factors of f in $\mathbb{Z}[x]$ then $\{w_1, \ldots, w_r\}$ is a basis of W in reduced echelon form, where w_k is defined as the 0–1 vector $(v_1 \cdots v_n)$ for which $g_k = \prod f_i^{v_i}$. Finding this reduced echelon basis $\{w_1, \ldots, w_r\}$ of W is the same as solving the combinatorial problem (equation (1) in section 1) in the Berlekamp-Zassenhaus algorithm.

We will use the following notations: If $L \subset \mathbb{Z}^n$ is a lattice, then B_L is a basis of L. The matrix whose rows are the elements of B_L is denoted by (B_L), and the reduced row echelon form of this matrix is denoted by $\mathrm{rref}(B_L)$. If any basis B_W of W is known, then the combinatorial problem is solved because $\{w_1, \ldots, w_r\}$ are the rows of $\mathrm{rref}(B_W)$.

LEMMA 2.8. *Let L be a lattice such that*

$$W \subseteq L \subseteq \mathbb{Z}^n. \tag{10}$$

Let $R = \mathrm{rref}(B_L)$. Then $L = W$ if and only if the following two conditions hold:

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

Proof. If $L = W$ then $\{w_1, \ldots, w_r\}$ must be the rows of R because of the uniqueness of the reduced row echelon form, and thus conditions A) and B) hold. Conversely, if A) and B) hold, and if $W \subseteq L$ then $\{w_1, \ldots, w_r\}$ must be linear combinations of the rows of R. But then $\{w_1, \ldots, w_r\}$ must be the rows of R because $\mathbb{Z}[x]$ has unique factorization. So L, the row space of R, must be equal to W, the span of $\{w_1, \ldots, w_r\}$. ∎

If we have a lattice L that satisfies equation (10) then we can test if $L = W$ by checking if conditions A) and B) hold. Condition B) can be checked in exactly the same way as in the Berlekamp-Zassenhaus algorithm; by computing the symmetric remainder of the product modulo p^a and checking if the result divides f in $\mathbb{Z}[x]$.

Initially we take $L = \mathbb{Z}^n$ in the algorithm, so that we can be certain that equation (10) holds. If we make sure that (10) continues to hold throughout the algorithm, then at each step, we can test if $L = W$ with lemma 2.8. Suppose $L \neq W$. The goal is to calculate a new lattice L' such that

$$W \subseteq L' \subseteq L$$

so that L' satisfies equation (10) as well. Then L is replaced by L'. The algorithm keeps repeating this until $L = W$, i.e. until conditions A) and B) hold. Once these conditions hold, testing that they hold gives as a byproduct all irreducible factors of f in $\mathbb{Z}[x]$ because that is how B) is tested. These irreducible factors are then the output of the algorithm.

1. **Algorithm terminates?** To prove that the algorithm terminates, we must prove that eventually conditions A) and B) hold.
2. **Output correct?** To prove that the output is correct (i.e. irreducible and complete) if there is an output at all (i.e. if it terminates), we do not need to prove that A) and B) will eventually hold, we only need to prove that equation (10) continues to hold during each step, because if $W \subseteq L$ then the algorithm can not terminate unless $L = W$. So it can not terminate unless the factors of f it produced are irreducible and complete.

Choose an s by d matrix A and choose integers a_i and b_i (recall that one must have $a \geq a_i > b_i > \log(2B_i)/\log(p)$). We will show how the algorithm can compute a new lattice $L' \subseteq L$, hopefully of smaller dimension than L, that contains all solutions $(v_1 \cdots v_n)$ of equation (9). This implies correctness of the algorithm because $w_1, \ldots, w_r$ satisfy equation (9) and hence $W \subseteq L'$.

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 higher values for $a_i - b_i$ (if at some point $a_i > a$ then this means that more Hensel lifting needs to be done to increase a). Eventually this must be successful (lemma 2.10) and we find L' with smaller dimension. Then replace L by L', check conditions A) and B) to see if $W = L$, 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 (everything is now in row notation). To solve the knapsack-like problem given in equation (9) we will construct the *knapsack lattice*, a lattice Λ such that the vector

$$v_S = (Cv_1 \ \cdots Cv_n \quad \epsilon_1 \cdots \epsilon_s) \tag{11}$$

is an element of Λ for every solution S of equation (9). The γ_i from (9) are not used. A vector v in Λ will be called M-short if the length $|v|$ of v is $\leq M$. Note that v_S is M-short for every solution S of the knapsack problem (9),

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

Let $\{e_1, \ldots, e_s\}$ be the standard basis of $\mathbb{Z}^s$. The 0 element of $\mathbb{Z}^n$ will be denoted by 0^n. All of these vectors are in row notation. The notation $(v,\ w) \in \mathbb{Z}^{n+s}$ refers to the vector obtained by concatenating v and w. The knapsack lattice $\Lambda \subseteq \mathbb{Z}^{n+s}$ is defined by the following basis: $B_\Lambda = B_{\mathcal{C}} \bigcup B_{p^*}$ where

$$B_{p^*} = \{(0^n,\ p^{a_i - b_i} e_i) | i \leq s\},\ B_{\mathcal{C}} = \{(Cv,\ vm) | v \in B_L\}, m = \begin{pmatrix} \mathcal{C}_1 \\ \vdots \\ \mathcal{C}_n \end{pmatrix}.$$

In the first step, $L = \mathbb{Z}^n$. Then B_L has n elements, B_Λ has $n+s$ elements, and the matrix of the basis of Λ is

$$(B_\Lambda) = \begin{pmatrix} C & 0 & \cdots & 0 & c_{1,1} & \cdots & c_{1,s} \\ 0 & C & \cdots & 0 & c_{2,1} & \cdots & c_{2,s} \\ \vdots & \vdots & & \vdots & \vdots & & \vdots \\ 0 & 0 & \cdots & C & c_{n,1} & \cdots & c_{n,s} \\ 0 & 0 & \cdots & 0 & p^{a_1 - b_1} & \cdots & 0 \\ \vdots & \vdots & & \vdots & \vdots & & \vdots \\ 0 & 0 & \cdots & 0 & 0 & \cdots & p^{a_s - b_s} \end{pmatrix},$$

where $c_{j,i}$ was defined in equation (8). In this case the matrix is square, so the determinant of the lattice Λ is then

$$D = C^n p^{(a_1-b_1)+\cdots+(a_s-b_s)}.$$

The number l of elements of B_Λ equals s plus the number of elements of B_L. We can use LLL to compute an LLL-reduced basis $V_1, \ldots, V_l$ of Λ. Let $V_1^*, \ldots, V_l^*$ be the Gram-Schmidt basis for $V_1, \ldots, V_l$. We can compute approximations $\overline{V}_k^*$ of V_k^* by floating point arithmetic. Let $r \leq l$ be the smallest integer such that $|\overline{V}_k^*| > M'$ for all $r < k \leq l$, where M' is M plus a bound on the round-off errors in the $\overline{V}_k^*$. So $|V_k^*| > M$ for all $k > r$. Now define $\Lambda' \subseteq \Lambda$ as the span of $\{V_k | k \leq r\}$, and let $L' \subseteq L$ be the projection of $\frac{1}{C}\Lambda'$ on the first n coordinates.

Lemma 2.9. *$W \subseteq L'$, so if the algorithm terminates, then the output is correct.*

Proof. It follows from the proof of (1.11) in [6] that if $|V_k^*| > M$ for all $k > r$ then all M-short vectors are in the span of $\{V_k | k \leq r\}$. If S is a solution of the knapsack problem (9), then the vector v_S from equation (11) is M-short, and thus in Λ'. If S corresponds to an irreducible factor g_k and 0–1 vector w_k, then w_k is $\frac{1}{C}$ times the projection of v_S on the first n coordinates, and $v_S \in \Lambda'$ hence $w_k \in L'$, and the lemma follows. ∎

If $r < \dim(L)$ then the algorithm makes progress because then $\dim(L') \leq r < \dim(L)$. The LLL-reduction of B_Λ is essential, because without it, it would be nearly certain that $r \geq \dim(L)$. If the number $p^{(a_1-b_1)+\cdots+(a_s-b_s)}$ is too small then one can expect $r \geq \dim(L)$ as well, see remark 3. in section 2.3.

Lemma 2.10. *The algorithm terminates.*

Proof. We have to show that if $L \neq W$ then eventually $\dim(L') < \dim(L)$, so that after finitely many steps the algorithm reaches $L = W$. It will be convenient for the notation (although the proof also works in general) to assume that matrix A has only one row, with the i'th entry equal to 1 and other entries 0. So then $s = 1$, $T_A(f_j) = \mathrm{Tr}_i(f_j)$, and $\mathcal{C}_j = \mathcal{C}_{b_1}^{a_1}(\mathrm{Tr}_i(f_j))$ which is an integer whose value depends on a_1 (in the proof a_1 will vary and b_1 will be constant). Denote

$$U(v) = \sum_{j=1}^{n} v_j \mathrm{Tr}_i(f_j) \in \mathbb{Z}_p \quad \text{for} \quad v = (v_1 \cdots v_n) \in L$$

and let $U(v, a_1)$ be the symmetric remainder mod $p^{a_1-b_1}$ of the integer $v_1\mathcal{C}_1 + \cdots + v_n\mathcal{C}_n$. Denote $\tilde{M} = n2^{n/2}M$, $\mathrm{Sol}_i(L) = \{v \in L | U(v) \in \mathbb{Z}\}$, and $B(L, a_1)$ as the set of all $v \in L$ for which $|Cv|^2 + U(v, a_1)^2 \leq \tilde{M}^2$. Since $\tilde{M} \geq M$ it follows that W is contained in the span of $B(L, a_1)$.

If $a_1 \geq a_1'$ then $B(L, a_1) \subseteq B(L, a_1')$, and since this is a finite set there exists a positive integer a_1' such that $B(L, a_1) = B(L, a_1')$ for all $a_1 \geq a_1'$. If a vector v is an element of $B(L, a_1)$ for all $a_1 \geq a_1'$, then $U(v, a_1)$ is bounded when $a_1 \to \infty$, hence $U(v) \in \mathbb{Z}$ and so $v \in \mathrm{Sol}_i(L)$. Therefore, $B(L, a_1) \subseteq \mathrm{Sol}_i(L)$ whenever $a_1 \geq a_1'$.

It is possible that $L = \mathrm{Sol}_i(L)$, even if the i'th trace had not been used before in the algorithm. For example, one could think of a polynomial f for which $\mathrm{Tr}_1(g) \in \mathbb{Z} \implies \mathrm{Tr}_2(g) \in \mathbb{Z}$ happens to be true for all monic factors $g \in \mathbb{Z}_p[x]$ of f. For such f, if $L = \mathrm{Sol}_1(\mathbb{Z}^n)$ then $L = \mathrm{Sol}_2(L)$. Such cases, although they are of course not very common, must be dealt with. Fortunately, these cases can be detected at negligible computational cost, because if $L = \mathrm{Sol}_i(L)$ then $U(v, a_1)$ is already small (without doing any lattice reduction) for every $v \in B_L$. And when this occurs, the remedy is trivial, just take the next value for i (in step 6 in section 2.2 the remedy is to go back to step 5). After a finite number of times we may assume that $L \neq \mathrm{Sol}_i(L)$. This implies that $\dim(\mathrm{Sol}_i(L)) < \dim(L)$ because $L/\mathrm{Sol}_i(L)$ can not have torsion elements: If $k > 1$ and $U(kv) \in \mathbb{Z}$ then $U(v) \in \mathbb{Q}$, but then $U(v) \in \mathbb{Z}$ because the $\mathrm{Tr}_i(f_j)$ are algebraic integers.

Now, for more notational convenience, assume that $L = \mathbb{Z}^n$. Then $l = n + 1$ and the matrix (B_Λ) is then the l by l matrix from section 2.1, with determinant $D = C^n p^{a_1-b_1}$. Let $V_1, \ldots, V_l$ be the LLL-basis and $V_1^*, \ldots, V_l^*$ the Gram-Schmidt basis, and let $r \leq l$ be minimal such that $|V_k^*| > M$ for all $k > r$. Now $|D| = \prod_k |V_k^*|$ so there exists k' such that $|V_{k'}^*| \geq |D|^{1/l}$. If a_1 is sufficiently large, then $|V_{k'}^*| > 2^{n/2}M$. It follows from the properties of an LLL-basis that then $|V_k^*| > M$ for all $k \geq k'$, and hence $r < l = n + 1$. So we make progress, i.e. $\dim(L') < \dim(L)$, except when $r = n = l - 1$ and the projection of $V_1, \ldots, V_n$ on the first n coordinates is linearly independent. To complete the proof we will show that this case is not possible whenever $a_1 \geq a_1'$.

If $r = n$ then $|V_n^*| \leq M$, and the properties of an LLL-reduced basis then imply that $|V_k^*| \leq 2^{n/2}M$ for all $k \leq n$. This in turn implies that $|V_k| \leq \tilde{M} = n2^{n/2}M$ for all $k \leq n$. Then $\frac{1}{C}$ times the projection of $V_1, \ldots, V_n$ on the first n coordinates is in $B(L, a_1)$, but if $a_1 \geq a_1'$ then $B(L, a_1)$ is contained in $\mathrm{Sol}_i(L)$ which has dimension $< n$, which implies that the projection of $V_1, \ldots, V_n$ on the first n coordinates is dependent. ∎

Remark: Since no bound for a_1 has been determined, the proof of the lemma provides no bound for the running time. An estimate for a_1 (not a rigorous bound) is given in section 3.3.

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 the 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 $a_i > b_i > \log(2B_i)/\log(p)$, see also remark 3. for this choice. If $a_i > a$ then do additional Hensel lifting to increase a.

Step 6 Compute the basis $B_{p^*} \bigcup B_{\mathcal{C}}$ for lattice Λ like in section 2.1. If the last s entries of the elements of $B_{\mathcal{C}}$ reduced modulo B_{p^*} are already small 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 $\Lambda' = \mathbb{Z}V_1 + \cdots + \mathbb{Z}V_r$. Let L' be $\frac{1}{C}$ times the projection of Λ' on the first n coordinates. Then replace B_L by a basis of L'. If $|B_L|$ did not decrease then return to step 5 and use larger values for $a_i - b_i$.

Step 8 Let $r = |B_L|$ be the number of elements of the new B_L. Note that $\dim(W) \leq n/4$ 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 Compute $R = \mathrm{rref}(B_L)$. If R does not satisfy condition A) in lemma 2.8 then go to step 5, otherwise proceed to step 10.

Step 10 Check condition B) in lemma 2.8 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 $g_k = \mathcal{C}^a(\prod f_i^{v_i})$. If g_k does not divide f in $\mathbb{Z}[x]$ then go back to step 5.

Step 11 Now $f = g_1 \cdots g_r$, the irreducibility of the monic polynomials $g_1, \ldots, g_r$ in $\mathbb{Z}[x]$ has been proven, so the computation is done.

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 $T_A = \mathrm{Tr}_{1..s}$. 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 is needed is small because several Tr_i's are combined into 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 decrease L more gradually. 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

$$I = ((a_1 - b_1) + \cdots + (a_s - b_s))\log(p)$$

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 by 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 a_i, increase 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 use larger a_i (but still $a_i \leq a$). In very rare cases, we need to increase a, meaning more Hensel lifting, in order to be able to make I large enough. If we use strategy 2 for the choice of matrix A, where we aim to find W 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 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 1 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 often (but not always) 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 2, but only from 100 down to 65 with strategy 1 using only $\mathrm{Tr}_{1..2}$ 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 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 it is likely that no progress is made (L remains the same). The lattice always contains at least one M-short vector (corresponding to the trivial factor $g = f$). If LLL does not find any short vectors, then that is a clear signal that the value of I that was used was too small.

It can make sense, when increasing s (i.e. when adding rows to matrix A) or when replacing rows of A, to add a row that already appeared in A, in the following way. When row i equals a new row i' that is being added, then take $a_{i'}, b_{i'}$ in such a way that the intervals $(b_{i'}, a_{i'})$ and (b_i, a_i) do not overlap. This could also be used as a test to see if all vectors in B_L actually meet the condition coming from row i like they are supposed to, and if they do not, then the same row can be added to A but using a different (without overlap) two-sided cut of the p-adic numbers.

It is possible to have 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 $a_i, \tilde{a}_i$ instead of a_i, b_i and one using $\tilde{a}_i, b_i$ instead of a_i, b_i, where $\tilde{a}_i$ is an integer between a_i 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, so $c_N \neq 1$, we need to make two changes to 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, $c_N^i \mathrm{Tr}_i(g) \in \mathbb{Z}$, so in the non-monic case Tr_i needs to be replaced by $c_N^i \mathrm{Tr}_i$.

The second change is in step 10a. This change is identical to the difference between the monic and non-monic case in the Berlekamp-Zassenhaus algorithm. To find g_k, take $\mathcal{C}^a(c_N \prod f_i^{v_i})$ instead of $\mathcal{C}^a(\prod f_i^{v_i})$, and divide it by the gcd of its coefficients.

5. We could also consider computing the factors f_j in $\mathbb{R}[x]$ or $\mathbb{C}[x]$ 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. Perhaps this is the algorithm one was looking for in section 6 in [9]. The disadvantage is that over $\mathbb{R}$ there are always many (at least $N/2$) factors f_j, which is generally (except for Swinnerton-Dyer polynomials) much more than over $\mathbb{Z}_p$.

6. One could try to use a higher value for y in the LLL algorithm [6], so that LLL performs a stronger lattice reduction, and then use a smaller value for I. One can also use PSLQ [2] instead of LLL to solve knapsack problems. We chose for LLL (with integer arithmetic) because it worked faster in our Maple experiments, however, no careful analysis has been done to compare the two.

3. EXPERIMENTS AND COMPLEXITY

3.1. Implementations

The algorithm was first implemented by the author in Maple. The source code and a number of examples are available from [15]. The factorization of $x^{128} - x^{112} + x^{80} - x^{64} + x^{48} - x^{16} + 1$ in Maple7 (which contains this implementation) is > 500 times faster than in Maple6. An implementation in Magma [16] was done by Allan Steel. An implementation for NTL was done by Guillaume Hanrot and Paul Zimmermann [17]. There is also an implementation in Pari-GP [18], and a newer implementation for NTL has now been written by Victor Shoup [19].

3.2. Some experiments

Experiments done with the Maple implementation on a Pentium 266 (the polynomials can be obtained from the website in section 3.1) show that the algorithm can factor polynomials that could not be factored before. This is experimental proof that the algorithm is a significant improvement, especially if one considers that the implementations in Magma, NTL and Pari-GP are even faster (due to faster Hensel lifting and lattice reduction, it is clear that compiled C-code runs faster than interpreted Maple code).

Examples of polynomials that previously could not be factored by any available system are the polynomial P7 from Paul Zimmermann's website, and the 6-set resolvent polynomial (degree $N = 924$ with $n = 84$ modular factors) for a polynomial with Galois group M_{12}. This means that computationally testing and proving that the Galois group of a polynomial of degree 12 is M_{12} can now be done with the most trivial method (factoring k-set resolvents), a method that was previously considered not to be feasible for this group.

Allan Steel reported that his Magma implementation factored polynomials with more than 400 modular factors, and this number may grow even higher because of recent improvements in lattice reduction [5]. Belabas, Hanrot, and Zimmermann reported the following progress on their implementations: The polynomial P8 from Zimmermann's website, two years ago it was impossible to factor P8, with [1] the CPU time for the searching phase was reduced to one hour, and with the method presented here it is reduced to less than one second; the searching phase now takes less time than Hensel lifting.

In the remainder of this section a relatively small example will be given with degree $N = 190$ and $n = 38$ modular factors. The timings are given on a relatively slow machine (Pentium 266 laptop). On the same machine, the Pari-GP implementation runs this example about 15 times faster. 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 the 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 modular factors $\mathcal{C}^a(f_i)$ 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 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 [6], the remaining vectors would need to be even larger, so the input vectors would need to be very large making the lattice reduction very costly.

3.3. A heuristic complexity estimate

If for each lattice reduction, the number I is always bounded by a polynomial in n, then each lattice reduction costs polynomial time.

If for each lattice reduction, the number P, which is the probability that progress (i.e. $\dim(L') < \dim(L)$) is made, is greater than some positive constant ϵ, then the expected number of lattice reductions is $< n/\epsilon$ so it is $O(n)$.

If both are true then the expected cost of all lattice reductions combined is bounded by a polynomial in n. This polynomial is independent of N as well as the size of the coefficients of f, which is a twofold improvement over all previous lattice based factorizers.

Remark: the probability P depends on I. A higher value of I means a higher value of P (thus fewer lattice reductions). For practical performance one should choose values of I aimed at having P large (say $0.7 < P < 0.95$) but not very large ($P > 0.99$), because if P is very large it implies that in most cases, the same lattice result could have been obtained with a cheaper (i.e. smaller I) lattice reduction. In the proof of lemma 2.10 the probability of progress is $P = 1$, however, the proof did not provide any bound for I because no bound for a_1' was determined.

First if: With the above remark in mind, the implementation for strategy 2 starts with $I = cn^2$ with $c = 0.12$. With strategy 1 a smaller value for I is chosen. In case there is little or no progress, the value of I is increased but it remains $O(n^2)$.

Second if: It is experimentally true that if $I = cn^2$ for the right constant c, then P is large and thus the expected number of lattice reductions is at most $O(n)$. In practise the number of lattice reductions will be much smaller than $O(n)$, especially if we would switch to strategy 2 where the expected number is just above 1. To explain why cn^2 works, we will give a heuristical argument (not a proof) that for some $c_1 > 0$, $c_2 > 0$ and $\epsilon > 0$, $P > \epsilon$ when $I = cn^2$ where $c = \max(c_1, c_2)$.

There exists a constant c_1 such that if $I \geq c_1 n^2$ then $|D|^{1/l} > 2^{n/2}M$ (notations as in the proof of lemma 2.10) which implied $r < l$. The proof of the lemma then showed that if

$$B(L, a_1) \subseteq \mathrm{Sol}_i(L) \tag{12}$$

then progress will be made, and that this holds whenever $a_1 \geq a_1'$. Let $D_{\tilde{M}} = \{v \in L | v \notin \mathrm{Sol}_i(L),\ |Cv| \leq \tilde{M}\}$. The number of elements is $|D_{\tilde{M}}| = \exp(O(n^2))$. If $|U(v, a_1)| > \tilde{M}$ for all $v \in D_{\tilde{M}}$ then (12) follows, and we will argue that this is likely whenever $I \geq c_2 n^2$ for some constant c_2. Now $U(v, a_1)$ is an element of the interval $(-q/2, q/2]$ where $q = p^{a_1 - b_1} = \exp(I)$.

If we assume (this is the part that makes our estimate heuristic, i.e. unproven) that there is a constant $c' \geq 1$ such that for each integer z in this interval, the probability that $U(v, a_1) = z$ is $\leq c'/q$, then the probability that $|U(v, a_1)| \leq \tilde{M}$ is at most $c'(2\tilde{M} + 1)/q$. The probability that $|U(v, a_1)| \leq \tilde{M}$ for any $v \in D_{\tilde{M}}$ is then $\leq P'$ where P' is defined as $|D_{\tilde{M}}| c'(2\tilde{M} + 1)/\exp(I)$. Choose $\delta > 0$. Then there exists a constant

c_2 such that $P' < \delta$ whenever $I \geq c_2 n^2$. If we take $\delta < 1 - \epsilon$ then $P \geq 1 - P' > \epsilon > 0$. The denominator of P' indicates that a choice of I, aimed at having P far away from 1 (say $P = 0.5$) is not optimal because then P could be increased significantly with a relatively small increase of I.

REFERENCES

1. J. Abbott, V. Shoup and P. Zimmermann, *Factorization in $\mathbb{Z}[x]$: The Searching Phase*, ISSAC'2000 Proceedings, 1–7 (2000).
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, 2nd ed,, editors B. Buchberger et all, Springer Verlag, 95–113 (1982).
5. H. Koy and C.P. Schnorr, *Segment LLL-Reduction of Lattice Bases*, Cryptography and Lattices Conference (CaLC) (2001).
6. A.K. Lenstra, H.W. Lenstra and L. Lovász, *Factoring polynomials with rational coefficients*, Math. Ann. **261**, 515–534 (1982).
7. M. Mignotte, *An inequality about factors of polynomials*, Math. of Computation, **28**, 1153–1157 (1974).
8. V. Miller, *Factoring Polynomials via Relation-Finding*, ISTCS '92, Springer Lecture Notes in Computer Science **601**, 115–121 (1992).
9. T. Sasaki, T. Saito and T. Hilano, *A unified method for multivariate polynomial factorization*, Japan J. Industrial and Applied Math 10, **1**, 21–39 (1993).
10. A. Schönhage, *Factorization of univariate integer polynomials by diophantine approximation and an improved basis reduction algorithm.* Lecture Notes in Computer Science **172**, 436–447, Springer-Verlag (1984).
11. C.P. Schnorr and M. Euchner, *Lattice Basis Reduction: Improved Practical Algorithms and Solving Subset Sum Problems*, Mathematical Programming, **66**, 181–191 (1994).
12. A. Storjohann, *Faster Algorithms for Integer Lattice Basis Reduction.* Technical Report 249, Departement Informatik, ETH Zurich, (1996).
13. B. de Weger, *Solving exponential diophantine equations using lattice basis reduction algorithms*, Journal of Number Theory **26**, 325–367 (1987).
14. H. Zassenhaus, *On Hensel Factorization, I.*, Journal of Number Theory **1**, 291–311 (1969).
15. M. van Hoeij, implementation, `http://www.math.fsu.edu/~hoeij/knapsack.html`
16. W. Bosma, J. Cannon and C. Playoust, *The Magma algebra system I: The user language*, J. Symb. Comput., **24**, 235–265 (1997).
17. G. Hanrot, P. Zimmermann, implementation, `http://www.loria.fr/~zimmerma/free/`
18. H. Cohen, Pari-GP, now maintained by K. Belabas, `http://www.parigp-home.de`
19. V. Shoup, NTL, A Library for doing Number Theory, `http://www.shoup.net/`