Early Termination Factorization of Univariate Polynomials with Rational Coefficients

Andrew Novocin & Mark van Hoeij
Department of Mathematics, Florida State University

Summary of Zassenhaus' Algorithm

Let $f \in \mathbb{Z}[x]$ be separable and monic with degree N. **Goal:** the factors of f in $\mathbb{Z}[x]$.

Idea 1: If $g \in \mathbb{Z}[x]$ divides f then the coefficients of g are smaller than some computable **bound** L.

Idea 2: If $g \in \mathbb{Z}[x]$ divides f then g can be **reconstructed** when g mod p^a is known for some $p^a > 2L$.

Idea 3: Factor $f = f_1 \cdots f_r$ over $\mathbb{Z}_p$ (the p-adic integers). There are only **finitely many** monic factors of f in $\mathbb{Z}_p[x]$. Each is of the form

$$g_v := \prod f_i^{v_i}$$

for some 0–1 vector $v = (v_1, \ldots, v_r)$.

Idea 4: $f_1, \ldots, f_r$ (and hence g_v) are not known exactly, but are only known mod p^a. That's enough using idea 2. In practice we find $f_1, \ldots, f_r$ mod p and Hensel Lift until they are known mod p^a.

Two Problems with Zassenhaus' Algorithm

Problem 1: Let $g_1, \ldots, g_s$ be the true factors of f in $\mathbb{Z}[x]$ and let $f_1, \ldots, f_r$ be the local factors (over the p-adic integers). The Zassenhaus algorithm applies Hensel lifting to determine $f_1, \ldots, f_r$ with a p-adic accuracy a that is guaranteed to be high enough to recover any potential factor of f in $\mathbb{Z}[x]$.

However, this p-adic accuracy a is often much higher than what was actually necessary to recover all factors $g_1, \ldots, g_s$. In practice it frequently happens that f has one large factor, say g_1, and zero or more small factors, say $g_2, \ldots, g_s$. Then, to recover $g_1, \ldots, g_s$ we do not need p^a to be larger than twice the largest coefficient of g_1. All we need is that p^a is larger than twice the largest coefficient in $g_2, \ldots, g_s$. That suffices to reconstruct $g_2, \ldots, g_s \in \mathbb{Z}[x]$ from their modular images, after which the remaining factor g_1 can be determined by a division in $\mathbb{Z}[x]$.

It is easy to give examples where this latter a is ten times smaller than the a used in Zassenhaus' algorithm. Just multiply a small irreducible polynomial by a big one.
(Of course a needs to be large enough not only to find $g_2, \ldots, g_s$, but also large enough to prove that $g_1, \ldots, g_s$ are irreducible. More precisely, a needs to be large enough to solve the combinatorial problem mentioned in Problem 2 below. However, using [van Hoeij 2002] this can usually be done with much less Hensel lifting than what is used in Zassenhaus' algorithm.)

Problem 2: When there are many local factors $f_1, \ldots, f_r$ it can take an exponentially long time to decide which of the local factors combine to form a rational factor g_i. The [van Hoeij 2002] algorithm presented a practical solution to this problem (no complexity result was given at the time, however, we now have a preprint which presents new asymptotically sharp bounds).

Problem Statement

We want to compute the factors $g_1, \ldots, g_s$ as quickly as possible, by not Hensel lifting further than necessary. The easiest way to prevent lifting too far is to do these two steps after each Hensel lift:

1. Try to solve the combinatorial problem using [van Hoeij 2002]
2. and if this succeeds, try to reconstruct $g_2, \ldots, g_s$ from their modular images (and g_1 with a division).

Suppose that lifting to at least p^{100} was necessary to solve both steps 1 and 2. We use quadratic Hensel lifting, so a doubles each step. This means that we solve the problem once we lifted to p^{128}, which is close to optimal. So compared to Zassenhaus' algorithm we could save much CPU time on Hensel lifting (Hensel lifting often dominates the CPU time).

Key Problem: *But couldn't our approach also be slower in some cases?* After all: What about the time that was spent when step 1 or 2 failed when we lifted to p^{64}, or to p^{32}, etc.? Step 2 costs little CPU time, but if step 1 failed by not lifting far enough, couldn't we have wasted CPU time?

Our Results

The CPU time in step 1 is spent on lattice reduction (LLL). This CPU time is a function of the number of LLL switches that were performed by LLL. We set up our algorithm in such a way that every LLL switch, no matter when it was made, will make a measurable amount of progress toward solving the combinatorial problem. We measure progress by a quantity μ (more details later). We prove an a priori upper bound of $\mathcal{O}(Nr^2)$ for μ in our preprint, and aim to improve this to $\mathcal{O}(r^3)$.
Each LLL switch, regardless of when it was made, will necessarily increase μ by at least 1. So any LLL switch brings us measurably closer to solving the combinatorial problem, which means the CPU time spent on "unsuccessful" calls to step 1 is not wasted. This way, our algorithm should always be efficient, regardless of how much Hensel lifting turned out to be necessary at the end.

Sketch of LLL algorithm

Let $b_1, \ldots, b_r \in L$ be a basis of a lattice $L \subset \mathbb{R}^m$ and denote $b_1^*, \ldots, b_r^* \in \mathbb{R}^m$ as the Gram-Schmidt orthogonalization over $\mathbb{R}$ of $b_1, \ldots, b_r$. Let $l_i = \log_{4/3}(\| b_i^* \|^2)$ and $\mu_{i,j} = \frac{b_i \cdot b_j^*}{b_j^* \cdot b_j^*}$.

Input: A basis $b_1, \ldots, b_r$ of a lattice L.
Output: A LLL-reduced basis of L.

1. *(Gram-Schmidt over $\mathbb{Z}$).* By subtracting suitable $\mathbb{Z}$-linear combinations of $b_1, \ldots, b_{i-1}$ from b_i make sure that $|\mu_{i,j}| \leq 1/2$ for all $j < i$.
2. *(LLL Switch).* If there is a k such that interchanging b_{k-1} and b_k will decrease l_{k-1} by at least 1 then do so.
3. *(Repeat).* If there was no such k in Step 2, then the algorithm stops. Otherwise go back to Step 1.

What each LLL switch does is to move some of the G-S length from b_i's to later vectors in our sequence of vectors.

$$\begin{array}{ccccccccc} l_1 & \Longrightarrow & l_2 & \Longrightarrow & l_3 & \Longrightarrow & \cdots & \Longrightarrow & l_r \\ b_1 & \leftrightarrow & b_2 & \leftrightarrow & b_3 & \leftrightarrow & \cdots & \leftrightarrow & b_r \end{array}$$

Each LLL switch brings us closer to a good basis (small l_1 and big l_r).

Solving the Combinatorial Problem

Take this matrix A which is similar to [BHKS], but with the last $N-1$ columns scaled down a factor c_j. Here c_j is the number $\sqrt{N}c_j'$ rounded up to the next power of 2, where c_j' is an upper bound for coefficient$(f \cdot g'/g,\, x^{N-1-j})$ for any factor g of f in $\mathbb{Z}[x]$.

$$A = \begin{pmatrix} & & & & & p^a/c_{N-1} \\ & & & & \cdot^{\cdot^{\cdot}} & \\ & & & p^a/c_1 & & \\ 1 & & & * & \cdots & * \\ & \ddots & & \vdots & \ddots & \vdots \\ & & 1 & * & \cdots & * \end{pmatrix} \quad \text{where } *_{i,j} = \frac{1}{c_j}\text{coefficient}(f \cdot \frac{f_i'}{f_i} \bmod p^a,\, x^{N-1-j}).$$

- Let $b_1, \ldots, b_k$ be an LLL reduced basis for the *rows* of this matrix.
- As long as the G.S. length of the last one is greater than $B := \sqrt{r+1}$, remove it. Let $b_1, \ldots, b_s$ = remaining vectors. If a is larger than the bound given in [BHKS] Theorem 4.3 then rref$(b_1, \ldots, b_s)$ yields the solution to the combinatorial problem. However, that bound does not appear to be sharp, in all examples we tried, a much smaller value of a was sufficient to solve the combinatorial problem (which includes the task of proving that $g_1, \ldots, g_s$ are irreducible).

Earlier we mentioned a switch-complexity (bound for the number of LLL switches) of $\mathcal{O}(Nr^2)$, or even $\mathcal{O}(r^3)$. The number a (and hence the lengths of the row-vectors given by matrix A) can depend on the coefficient size of f. So how can the switch-complexity be independent of coefficient size? That is what we will explain below.

Lets start with a basis for $\mathbb{Z}^r$. That's the left lower corner of our matrix. Now add one row and column:

$$\begin{pmatrix} & & & p^a/c_1 \\ 1 & & & * \\ & \ddots & & \vdots \\ & & 1 & * \end{pmatrix}$$

We could LLL this matrix and see if that solved the combinatorial problem (if not then add the next row/column, etc.). However, entries in the last column could be huge, so instead of LLL, we use the algorithm below, whose switch-complexity is independent of the coefficient size of f as we shall see.

Gradual B-Reduction

1. Scale down last column a factor 2^{rd} where d is large enough to make the last column of size $\mathcal{O}(1)$.
2. Repeat d times:
   - Scale up last column a factor 2^r.
   - LLL (the vectors are the rows of the matrix)
   - Remove (if any) last vector(s) with G.S. length $> B = \sqrt{r+1}$.

Output = LLL reduced $b_1, \ldots, b_s$.

We now have to show that the switch-complexity is independent of d (the number of LLL calls). Note that d depends on a, so d depends indirectly on the coefficient size of f.

Value & Bounding the Number of LLL Switches

We start with $r+1$ vectors, and at any given time we have $b_1, \ldots, b_s$ remaining vectors and $r+1-s$ removed vectors. Again $l_1, \ldots, l_s$ are the logarithmic Gram-Schmidt lengths. We now assign a value to the current configuration as:

$$\mu(b_1, \ldots, b_s) = 0 \cdot l_1 + 1 \cdot l_2 + \cdots + (s-1) \cdot l_s + (r+1-s) \cdot r \cdot \log_{4/3}(2^{3r}B^2)$$

The key to the proof is now that

- $\mu = 0$ at the beginning.
- No step in the algorithm decreases μ.
- Each LLL switch increases μ by at least 1, regardless of which LLL call made that switch.
- μ can never become more than $(r+1-0) \cdot r \cdot 10r = 10(r+1)r^2$

Early Termination Factorization

Algorithm: Let $f \in \mathbb{Z}[x]$ be monic and separable. Let c_j be as defined earlier. Also let $b_1, \ldots, b_r$ be the rows of the $r \times r$ identity matrix.

1. Factor f mod p and call the local factors $f_1, \ldots, f_r$.
2. Quadratic Hensel Lift until $\prod p^a/c_j \geq (B^r 2^{(r(r-1)/2})$ only including the j with $p^a/c_j > 2^{3r+1}$.
3. Repeat the following until there are no more logarithmic coefficients with $p^a/c_j > 2^{3r+1}$ to add:
   (a) Add a logarithmic coefficient (scaled down by c_j and only if $p^a/c_j > 2^{3r+1}$) to each b_i and add the row vector $(0, \ldots, 0, p^a/c_j)$ as seen before.
   (b) Gradual B-Reduce our vectors.
   (c) Check to see if the remaining $b_1, \ldots, b_s$ solve the combinatorial problem.
4. If we are not done, then make one more Hensel Lift and go to Step 3.

Why This is Interesting

There used to be a gap between the best factoring algorithm in theory, and the best algorithm in practice. Before 2001, the Zassenhaus algorithm performed best in practice, while LLL/Schönhage had the best theoretical complexity. This gap between theory and practice grew even wider with [van Hoeij 2002] because this algorithm was even faster in practice, while even worse in theory ([van Hoeij 2002] contains no complexity bound).

Then in [BHKS] the gap was made smaller; polynomial time complexity bounds for two versions of the [van Hoeij 2002] algorithm were given. However, the gap between theory and practice remained very large because the version that was faster in practice received the worse theoretical bound!

We can now resolve this unfortunate situation. In our preprint (still in progress) we will make the (in practice) fastest version even faster, in practice, by saving time on Hensel lifting. For this (in practice) fastest version we will use the same strategy (using μ as above) to prove a bound for the switch-complexity that is *asymptotically sharp*. This bound perfectly captures the actual behavior of the algorithm, in fact, the value μ could even be used to give a realistic progress bar!

We do not merely reduce the gap between theory and practice; we eliminate the gap altogether. *The algorithm that is best in practice, and the algorithm that is best in theory, will now be the same algorithm.*

Our theoretical work resulted in more than just a better bound for the complexity of factoring. It also allowed us to solve the key problem (mentioned earlier) in designing an efficient early termination algorithm. Recall that the key problem was wasting time on attempts that were "unsuccessful" because we had not lifted far enough. We can now solve this problem by designing the algorithm with the value μ in mind, we just have to ensure that μ increases with every LLL switch, and that we take no steps that could decrease μ.

References

K. Belabas *A relative van Hoeij algorithm over number fields*, J. Symbolic Computation, **37** (2004), pp. 641–668.

K. Belabas, M. van Hoeij, J. Klüners, A. Steel, *Factoring polynomials over global fields*, arXiv:math/0409510v1 (2004).

M. van Hoeij, *Factoring polynomials and the knapsack problem*, J. Number Theory, **95** (2002).

M. van Hoeij and A. Novocin, *Complexity results for factoring univariate polynomials over the rationals*, preprint (2007).

A. K. Lenstra, H. W. Lenstra, Jr., and L. Lovász, *Factoring polynomials with rational coefficients*, Math. Ann. **261** (1982).

H. Zassenhaus, *On Hensel factorization I*, Journal of Number Theory (1969).