The Complexity of Factoring Univariate Polynomials over the Rationals

Tutorial Abstract

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

ABSTRACT

This tutorial will explain the algorithm behind the currently fastest implementations for univariate factorization over the rationals. The complexity will be analyzed; it turns out that modifications were needed in order to prove a polynomial time complexity while preserving the best practical performance.

The complexity analysis leads to two results: (1) it shows that the practical performance on common inputs can be improved without harming the worst case performance, and (2) it leads to an improved complexity, not only for factoring, but for LLL reduction as well.

Categories and Subject Descriptors

I.1.2 [**Symbolic and Algebraic Manipulation**]: Algorithms; G.4 [**Mathematics of Computing**]: Mathematical Software

General Terms

Algorithms

Keywords

Symbolic Computation, Polynomial Factorization

Most practical factoring algorithms in $\mathbb{Q}[x]$ use a structure similar to Zassenhaus [10]: factor f modulo a small prime p, Hensel lift this factorization, and recombine these local factors into factors in $\mathbb{Q}[x]$. To reduce the combinatorial part, one first tries several primes, and then selects a p for which the number (denoted r) of factors mod p is smallest.

Polynomial time algorithms, based on lattice reduction, were given in [7, 9]. For a polynomial f of degree N, and entries bounded in absolute value by 2^h, these algorithms perform $\mathcal{O}(N^2(N+h))$ LLL switches. Schönhage [9] gave a modification to LLL reduction, and obtained the following complexity for factoring: $\tilde{\mathcal{O}}(N^4(N+h)^2)$, where $\sim$ indicates that logarithmic factors are ignored.

ISSAC'13, June 26–29, 2013, Boston, Massachusetts, USA.
ACM 978-1-4503-2059-7/13/06.

When comparing an algorithm with other algorithms, it is useful to know

Q.1 What is the worst-case asymptotic behavior?

Q.2 What is the best proven upper bound?

Q.3 Is the bound a good predictor for the running time on worst-case inputs?

Q.4 What about the performance on typical inputs?

Most attention is usually spent on Q.2 even though the other questions are more interesting.

Despite its exponential complexity, Zassenhaus [10] is usually faster than the polynomial time algorithms in [7, 9]. To explain this, note that [10] is not exponential in terms of N, it is exponential in r, which is usually $\ll N$. However, it is possible for r and N to be comparable in size. The best-known examples are Swinnerton-Dyer polynomials. They have $r = N/2$, which is worst-case ($r > N/2$ indicates the presence of factors of low degree).

The algorithm with the best complexity is described in [4]. It is based on the strategy in [5, 8]. It is efficient in practice and improves Schönhage's complexity, except if one makes the highly restrictive assumption that $N + h = \mathcal{O}(r)$, in which case [4] is only faster by a large constant.

We will denote the complexity of the algorithm in [4] as $C_H + C_L + C_O$ where C_H is the time spent on Hensel lifting, C_L is the time spent inside a variation of the LLL algorithm, and C_O are other costs (factoring f modulo several primes, preparing the input for LLL, reconstructing factors in $\mathbb{Z}[x]$ by multiplying p-adic factors, etc.).

For the LLL cost C_L, the answer to question Q.2 is $\tilde{\mathcal{O}}(r^6)$. The main ingredient is to prove that the number of LLL switches is bounded by $\mathcal{O}(r^3)$ (no logarithmic factors). Note that the bound for C_L is independent of both N and h! Regarding question Q.1, experiments suggest that C_L grows less than r^6, but this does not imply that the asymptotic complexity has degree < 6 (perhaps r^6 still manifests itself for impractically large inputs?). Regarding question Q.3 for C_L, it seems that r^6 is roughly in the right ball-park.

C_H has lower degree than C_L, but this does not imply that C_H is asymptotically smaller than C_L. After all, C_H depends on both N and h which are generally much larger than r. Indeed, for typical inputs, C_H dominates the CPU time. Ideally, one would have a bound for C_H that (a) can be proven, and (b) reflects the actual behavior of the algorithm.

Hensel lifting is highly optimized [3]. In order to know how much time [4] will spend on Hensel lifting, we need to

know the p-adic precision that [4] will lift to. That turns out to be difficult to predict, it can vary considerably, and our bound appears to be a factor N higher than what can actually occur.

In practice, [4] lifts far less than [7, 9]. Regarding question Q.1, we conjecture that (just like [10]) it never lifts further than p^a with $\log(p^a) = \tilde{\mathcal{O}}(N + h)$. This can be translated into a number theoretical problem. This problem has some resemblance to the abc-conjecture and looks plausible, but unfortunately, we could not prove it. Regarding question Q.2, our bound from [2] is $\log(p^a) = \mathcal{O}(N(N + h))$, which is the same as in [7, 9].

Even though we could not prove an upper bound with $\log(p^a) = \tilde{\mathcal{O}}(N + h)$, the actual amount of Hensel lifting in [4] is often smaller still, through a technique called early-termination. Suppose for example that the input f is irreducible (that does not mean that factoring is an empty task; one still has to compute an irreducibility proof). Though counter intuitive, computing mod p^a often suffices to prove irreducibility of a polynomial with coefficients $\gg p^a$.

It would not harm the asymptotic worst-case complexity if the algorithm starts by Hensel lifting the p-adic factors mod p^a with $\log(p^a) = \mathcal{O}(N + h)$. But it would not be a good design choice. For instance, suppose f has two factors, $f = f_1 f_2 \in \mathbb{Z}[x]$, and that f_1 has 10-digit while f_2 has 1000-digit coefficients. Suppose also that the combinatorial problem is solved once $\log_{10}(p^a)$ reaches 50. Then f_1 (but not f_2) can be reconstructed from its image mod p^a. But f_2 can be constructed as f/f_1. So even though f had coefficients with ≈ 1000 digits, a far smaller p-adic precision could be sufficient to factor it.

Inputs where where one of the irreducible factors is large, and all the others are small, are common (e.g. irreducible inputs). For questions Q.3 and Q.4, this means that for polynomials with large coefficients, a good worst-case predictor for C_H can overestimate the CPU time on typical inputs.

The goal in [4] was to set up the algorithm in such a way that we make no compromises on the practical performance (typical inputs and worst-case inputs) while simultaneously designing it in such a way that it satisfies the best upper bound that we are able to prove for a factoring algorithm.

The bound for C_L is quite good, but the bound for C_H is not. It is highly implausible that $\log(p^a) = \mathcal{O}(N(N + h))$ is close to sharp in the worst case. But it is also difficult to prove a better bound. Settling this issue would lead to a much better (Q.1 instead of Q.2) description of the complexity of factoring.

I used to think that complexity analysis is not useful in general, that it was useful if and only if it gives a good description for the actual CPU time. If known bounds are far from the actual performance, then, in order to avoid confusion, I thought it is better to give no bound (i.e. complexity = unknown). My view has become more nuanced since then. The complexity analysis of C_L, the LLL cost during factoring, has lead to improvements ("gradual feeding", discussed at [1] and analyzed in [8, 6]) for other LLL applications as well. Moreover, the complexity analysis also showed how a strategy called "early termination" (to improve CPU timings on typical inputs) could be set up in such a way that the CPU time on worst-case inputs will not be adversely affected. So even if only CPU timings matter, as I used to think, complexity analysis still turned out to be useful.

1. REFERENCES

[1] *Open Questions From AIM Workshop*, 2006. www.aimath.org/WWN/polyfactor/polyfactor.pdf

[2] K. Belabas, M. van Hoeij, J. Klüners, and A. Steel. *Factoring polynomials over global fields. J. de Théorie des Nombres de Bordeaux*, **21**, 15–39, 2009.

[3] A. Bostan, G. Lecerf, B. Salvy, É. Schost, and B. Wiebelt. *Complexity issues in bivariate polynomial factorization*, Proceedings of ISSAC, 42–49, 2004.

[4] W. Hart, M. van Hoeij, A. Novocin, *Practical Polynomial Factoring in Polynomial Time*, ISSAC'2011 Proceedings, 163–170, 2011.

[5] M. van Hoeij, *Factoring polynomials and the knapsack problem*, *J. of Number Theory*, **95**, 167–189, 2002.

[6] Mark van Hoeij and Andrew Novocin. *Gradual sub-lattice reduction and a new complexity for factoring polynomials.* LATIN, 539–553, 2010.

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

[8] A. Novocin. *Factoring Univariate Polynomials over the Rationals.* PhD thesis, Florida State University, 2008.

[9] A. Schönhage. *Factorization of univariate integer polynomials by Diophantine approximation and improved basis reduction algorithm.* Proceedings of the 1984 International Colloquium on Automata, Languages and Programming (ICALP 1984), **172** LNCS, 436–447, 1984.

[10] H. Zassenhaus. *On Hensel Factorization I.* J. Number Theory, **1, 291–311, 1969.**