CSE291-14: The Number Field Sieve

https://cseweb.ucsd.edu/classes/wi22/cse291-14

Emmanuel Thomé

February 24, 2022

Part 7

The square root step

Plan

Forcing squares

Recall where relations are coming from:

- the integer $a_i - b_i m$ is smooth. $\qquad a_i - b_i m = \pm \prod p_j^{e_j}$,
- the ideal $(a_i - b_i\alpha)$ is also smooth. $\qquad (a_i - b_i\alpha) = \prod \mathfrak{p}_k^{e'_k}$,

We consider a combination $S(x) = \prod_i (a_i - b_i x)$.

- How do we make $S(m)$ a square in $\mathbb{Q}$?
- How do we make $S(\alpha)$ a square in $\mathbb{Z}[\alpha]$?

(we mostly focus on the algebraic side.)

Computing the square root

Once we know that some combination $S(x)$ is such that $S(\alpha)$ is a square in $\mathbb{Z}[\alpha]$, how do we actually compute the square root?

If we succeed, we find:

- A rational number R such that $R^2 = S(m)$.
- An integer polynomial such that $T(\alpha)^2 = S(\alpha)$.

This means that $R^2 \equiv T(m)^2 \mod N$, which gives a non-trivial factor with probability $\geq 1/2$.

Factoring only!

All this square root business is only relevant for the integer factorization context.

Forcing squares: several steps

First part of the answer: linear algebra.

- By solving a linear system, we force all valuations to be even.

Is this enough ? Clearly not.

- We only guarantee that the principal ideal $(S(\alpha))\mathcal{O}_K$ is the square of an ideal.
- This is not the same as having $S(\alpha) = \square$ in $\mathbb{Z}[\alpha]$.

We can overcome these obstructions heuristically with characters.

Obstructions

Obstructions

Many reasons for $S(\alpha)$ to not be a square in $\mathbb{Z}[\alpha]$.

- If we were lazy with algebraic number theory, the ideal $S(\alpha)\mathcal{O}_K$ has even valuations at almost all ideals, but perhaps not all.
- The ideal $S(\alpha)\mathcal{O}_K$ could perhaps be the square of an ideal, but would that ideal be principal ?
- As a generator, $S(\alpha)$ would still have a unit contribution.
- Even if $S(\alpha)$ is a square, is it a square in $\mathbb{Z}[\alpha]$?

Also for $S(m)$: the sign ($=$ unit contribution in $\mathbb{Q}$) is an issue.

Characters

Bad news
- Deciciding principality is intractable.
- Units cannot be computed.

Our goal: find ways around these obstacles!

Characters

Formally for all row dependencies $S(x) = \prod_i (a_i - b_i x)$:

- $S(\alpha)$ belongs to a subgroup V of $K^\times$ (using $K = \mathbb{Q}(\alpha)$).
- squares of elements of V form a subgroup of V.
- The quotient $\Omega = V/V^2$ is a $\mathbb{F}_2$-vector space.
  (But computing a basis is not possible.)
- Algebraic number theory allows us to bound $\dim \Omega$.
  - Unit rank + Class group 2-rank + skipped ideals + 2-torsion units.
  - Only the class group 2-rank is not easy to know. But it's well under 20 anyway.
  - Thus $\dim \Omega$ is significantly below 64.

Def: character = $\mathbb{F}_2$-linear map $(\Omega, \times) \to (\mathbb{F}_2, +)$.

Characters

How do we use characters ?

Let χ be an algebraic character $\Omega = V/V^2 \to \mathbb{F}_2$.

- Let $S_1(x)$ be a row dependency with $\chi(S_1(\alpha)) = 1$.
- Let $S_2(x)$ be a row dependency with $\chi(S_2(\alpha)) = 1$.
- Then $S_1 S_2$ has $\chi(S_1(\alpha) S_2(\alpha)) = 0$.

If we can compute χ, we can fabricate combinations in $\operatorname{Ker} \chi$.

Idea: pick several characters, hoping that $\bigcap_\chi \operatorname{Ker} \chi = 0 \in \Omega$.

Characters

We know some characters:

- Parity of the valuation at an ideal.
  - This should be zero for all ideals that we took into account when computing valuations.
- Sign at any real embedding.
- Square-ness of the value modulo a maximal ideal $\mathfrak{p}$.
  - Only works if non-zero modulo $\mathfrak{p}$.
  - Can also divide by uniformizing element.
- yet some other choices, but mostly useful in the DL context.

Really plenty to choose from. Stick to the most efficient ones.

Characters

Example of a fast character check

Let $\mathfrak{p} = (p, x - r)$ be prime ideal, with $p < 2^{64}$.

- Compute $S(r) \bmod p$.
- In the (unlikely) case $p \mid S(r)$, use $p^{-\nu_p(S(r))}S(r)$ instead.
- Define $\chi(S(\alpha)) = (\frac{S(r)}{p})$.
  This is 1 if $S(\alpha)$ maps to a square mod $\mathfrak{p}$, and -1 if not.

Implementation: never compute the $S(x)$ for real.

- Compute characters directly on the $(a - b\alpha)$.
- Map $+1$ to 0 mod 2, and -1 to 1 mod 2.
- Apply the merge matrix to the obtained characters.
- Deduce (linear algebra) the recombinations with $\{\chi\} \to 0$.

Characters = necessary conditions

Being a square at the reduction modulo a prime ideal is certainly a necessary condition.

Heuristically, we expect that the different characters that we pick will generate the full dual space Ω^*, so that something that evaluates trivially at all of them is zero in $\Omega = V/V^2$, and so is an element of $V^2 \subset (K^\times)^2$.

Some class field theory results can bring some rigor, e.g. the Grunwald-Wang theorem.

Input of the square root step

- Linear algebra $\Rightarrow$ all valuation even.
- Characters step $\Rightarrow \prod_i (a_i - b_i\alpha) = \square$ in $\mathbb{Q}(\alpha)$.
- Multiply by some power of f_d, and then by $f'(\alpha)^2 \Rightarrow \square$ in $\mathbb{Z}[\alpha]$.
  (by a textbook number theoretic trick)

Thus

$$(a_1 - b_1 m) \times \cdots \times (a_s - b_s m) \equiv \phi((a_1 - b_1\alpha) \times \cdots \times (a_s - b_s\alpha)) \mod N$$

is actually a congruence of squares

BUT computing the square root is non trivial, esp. on algebraic side.

Square root psychology

The sqrt step is
- mathematically interesting
- rarely an issue computationally speaking.

However it comes LAST in a long computation.

- The "user" is in front of his terminal.
- Having to wait is annoying, esp. if code improvements are easily obtained.

The algebraic square root:

- gathers most mathematical difficulties.
- is computationally harder.

$\Rightarrow$ Focus on alg. sqrt.

Approaches for sqrt

We investigate several approaches.

- All algorithms are linear or quasi-linear in the input size.
- However the input is large. 21GB for RSA-768.
  Input known as a product form $\prod(a_i - b_i\alpha) = S(\alpha)$.
- The output really is $T(m) \bmod N$ where $T(\alpha)^2 = S(\alpha)$.
  If possible, try to avoid computing $T(\alpha)$ itself.

The different approaches compute different things.

- Some compute $S(\alpha)$, some don't.
- Some compute $T(\alpha)$, some compute $T(m) \bmod N$ directly.
- Some exploit the known factorization of $(a - b\alpha)$.

NFS square root 30 years ago

May rephrase the problem as factoring $X^2 - S(\alpha)$ in $K = \mathbb{Q}(\alpha)$.

- Quasi-linear (we take $K = \mathbb{Q}(\alpha)$ constant).
- BUT unacceptably expensive 30 years ago.
  (by then, a Karatsuba implementation was all it took to have the edge in a bignum battle).
- Motivation to explore specially adapted algorithms:
  - Montgomery;
  - Couveignes.

NFS square root 30 years ago

May rephrase the problem as factoring $X^2 - S(\alpha)$ in $K = \mathbb{Q}(\alpha)$.

- Quasi-linear (we take $K = \mathbb{Q}(\alpha)$ constant).
- BUT unacceptably expensive 30 years ago.
  (by then, a Karatsuba implementation was all it took to have the edge in a bignum battle).
- Motivation to explore specially adapted algorithms:
  - Montgomery;
  - Couveignes.

Since then: practicality of asymptotically fast methods everywhere.

- May explore again the direct approach.
  - Easy to program.
  - Can leverage fast implementations in e.g. Gnu MP.
- We are also interested in parallelization.

The best way

The most efficient algorithm for the square root is Montgomery's.

Montgomery's algorithm is much different from the others.

- Requires some number-theoretic primitives not there in typical NFS code.
  - Zassenhaus round-2.
  - Complete ideal factorization.
  - Arbitrary ideal arithmetic.
  - Lattice reduction.
- Cado-NFS has only had this functionality for a short time. This led us to postpone (indefinitely?) the implementation of Montgomery's square root algorithm.

It makes more sense to describe Montgomery's algorithm last in this talk.

Plan

The direct (lifting) approach

The direct/lifting/naive/brute-force way:

- Compute $S(\alpha)$ from the set of $a_i - b_i\alpha$.
- Compute $T(\alpha)$ as a square root in $\mathbb{Z}[\alpha]$.
- Deduce $T(m)$ mod N.

Key: take an inert prime p: such that $p\mathcal{O}_K$ is prime.

- The field $\mathcal{O}_K/p\mathcal{O}_K$ is isomorphic to $\mathbb{F}_{p^d}$.
- The field $K_p = K \otimes \mathbb{Z}_p$ is a degree d extension of $\mathbb{Q}_p$.
- Elements of $\mathbb{Z}[\alpha]$ easy to recognize in K_p: expansion terminates.
- p-adic structure allows for a lifting approach.

I love diagrams

$$\begin{tikzcd}
\mathcal{O}_K \arrow[rr] \arrow[dr, "\operatorname{mod} p"'] & & K \otimes_{\mathbb{Z}} \mathbb{Z}_p \cong \mathbb{Q}_{p^d} \arrow[dl, "\operatorname{mod} p"] \\
& \mathbb{F}_{p^d} &
\end{tikzcd}$$

Lifting approach: strategy

- Projection map $\pi : \mathbb{Z}_p[\alpha] \to \mathbb{F}_{p^d}$ to the residue field.
- Consider $t \in K_p$ an arbitrary lift of $\pm\sqrt{\pi(S(\alpha))}$.
  We thus have $t - T(\alpha) \equiv 0 \bmod p$.
- $T(\alpha)$ is a fixed point of $x \to x + \frac{S(\alpha)-x^2}{2x}$.
- lift, lift, lift.

Requirements
- Arithmetic in K_p (fixed precision will do).
- Fast integer multiplication.
- How high should we lift ?
- Inert primes...

$T(\alpha)$ determined only up to sign !

There is no such thing as “the” square root.
The choice of t determines the $T(\alpha)$ obtained by lifting.

How high is the lift ?

Rough estimate:
- Coefficients of $S(\alpha)$ have, say, n bits.
- So coefficients of $T(\alpha)$ should have $\approx n/2$ bits.

Easy to make it slightly more serious.

- Compute floating-point approximations to $\log |a_i - b_i z|$.
  - z runs over complex roots of f (recall $f(\alpha) = 0$).
  - Round (take ceil)
- We obtain an upper bound for $\log |T(z)|$.
- Derive upper bound on coefficients of T.
- Can restrict the lift to sufficiently large p^λ.

Inert primes ?

- (Chebotarev) The density of primes inert in a degree d number field is proportional to the ratio

$$\frac{\#\{\sigma \in G,\ \mathrm{ord}(\sigma) = d\}}{\# G}$$

  where G is the Galois group.
- But that number could be zero ! E.g. for $G \cong \mathbb{Z}/2\mathbb{Z} \times \mathbb{Z}/2\mathbb{Z}$.
- Fortunately for GNFS, polynomials have the generic $G = \mathfrak{S}_d$. Not so for SNFS.
- The lifting approach does not work as is for non-generic Galois group.

Lifting approach implementations

For RSA-768, the lifting approach could not handle the sqrt step.

- Memory requirement probably $\approx$ 100GB. We did not have that much RAM by then (2010).
- No longer an issue now. RSA-250 square root: 900GB RAM, 10 hours.

Several existing implementations (msieve, cado-nfs).

Without inert primes

It is possible to do without inert primes.
(e.g. Pari does that to factor polynomials in number fields.)

- Pick a prime which is "as inert as it can be": splits in the smallest possible number of factors.
  (This may still mean up to $d/2$ factors.)
- Compute desired roots modulo each of the several factors of $p\mathcal{O}_K$, and lift appropriately.
- Find the correct combination.

This roadmap will be explored later in this talk.

Plan

Working modulo many primes ?

1990 era:
- no widely available FFT implementation for multiplying integers (GMP has one only since 1999).
- Explore ways to avoid it.

Can't we work with several primes ?

Suppose we have 100 inert primes.

- Compute square root modulo $p_0, \ldots, p_{99}$.
- Try to recombine via CRT.
- Problem: only 2 correct square roots among 2^{100} possible combinations !

We need a way to tell apart the two square roots consistently $\bmod p$.

Couveignes' algorithm

Couveignes' trick for **odd-degree number fields**

We have $\mathrm{Norm}_{K/\mathbb{Q}}(-T(\alpha)) = -\mathrm{Norm}_{K/\mathbb{Q}}(T(\alpha))$.
Thus for $t \equiv \pm\pi(T(\alpha))$, we have:

$$\mathrm{Norm}_{\mathbb{F}_{p^d}/\mathbb{F}_p}(t) = \pm\,\mathrm{Norm}_{K/\mathbb{Q}}(T(\alpha)) \bmod p.$$

- $\left|\mathrm{Norm}_{K/\mathbb{Q}}(T(\alpha))\right|$ is something we can compute.
- Define the square root $T(\alpha)$ as the one with positive norm.
- This leads to a well-defined choice of root modulo each p.

Notations
- Let $\{p_i\}$ be a collection of sufficiently many primes.
- Let $T_i(x)$ be such that $T_i(\alpha) \equiv T(\alpha) \mod p_i$.
- Let $q_i \in [0, \prod_i p_i[$ satisfy $q_i \bmod p_j = \delta_{i,j}$.

Couveignes' algorithm

We have $T(\alpha) = \sum_i q_i T_i(\alpha)$.

- We need not compute $T(\alpha)$.
  Directly computing $T(m) \bmod N$ is simpler.
  $\Rightarrow$ accumulate the contributions modulo each p_i.

Complexity:

- dominated by the computation of $\{S(\alpha) \bmod p_i\}$.
- if $S(\alpha)$ is read for each p_i, quadratic complexity.

Key characteristics:

- Limited to odd degree.
- Requires inert primes.

Plan

Another CRT method

Goal: have something which works in the CRT way, but:

- without requiring inert primes.
- with good complexity.

We only target a small number of primes.

Mixing existing ideas is enough to achieve this.

- Borrow from the structure of number field root finding.
- Use subproduct trees very often.

Setup

Let $\{p_i\}$ be a collection of primes totally split in K.

Facts with CRT modulo prime ideals

We let $(p_i) = \mathfrak{p}_{i,1} \cdots \mathfrak{p}_{i,d}$, and $P = \prod p_i$.

- CRT: Knowing $\{T \bmod \mathfrak{p}_{i,j}\}$, we can recover $T \bmod P$.
- Each of the coefficients of T above is expressed linearly.

Indeed we may write polynomials $Q_{i,j}(x)$ such that:

$$T(x) \equiv \sum_{i,j} Q_{i,j}(x) T_{i,j} \mod P.$$

It is also posssible to do the same with the p_i-adic lifts of $T_{i,j}$.

CRT and lifting together

Let p be totally split in K. We have:

$$\mathbb{Z}[\alpha] \hookrightarrow K_p \cong \mathbb{Q}_p \times \cdots \times \mathbb{Q}_p.$$

For computing $\pm\sqrt{S(\alpha)}$ in all the $\mathbb{Z}_p$ parts, one has to:

- Compute the roots of f modulo p: $r_1, \ldots, r_d$.
- Lift each p-adically to some precision: $\tilde{r}_1, \ldots, \tilde{r}_d$.
- Deduce the image of $S(\alpha)$ in each part.
- Compute the p-adic square root (by lifting).

If we do so modulo many primes p_i, we can recover the result as in the CRT setup.
Lift up to precision λ modulo each p_i, then:

$$T(x) \equiv \sum_{i,j} \pm Q_{i,j}(x) T_{i,j} \mod P^{\lambda}.$$

Finding the correct combination

Issue already encountered:

many CRT shares $\Rightarrow$ intractable recombination problem

- This is the reason why we focus on relatively few primes (number of shares $k = d \times \#\{p_i\} \lessapprox 60$).
- Improved reconstruction allows more parallelism.

Streamlining reconstruction

Let M be a bound on the coefficients of $T(x)$, and let $P^\lambda > \frac{M}{\epsilon}$.

$$T(x) \equiv \sum_{i,j} \pm Q_{i,j}(x) T_{i,j} \mod P^\lambda,$$

$$\underbrace{\frac{1}{P^\lambda} T(x)}_{\text{coeffs}<\epsilon} \equiv \sum_{i,j} \pm \frac{Q_{i,j}(x) T_{i,j}}{P^\lambda} \mod 1.$$

Problem reduced to (e.g.) finding a sum of floating-point values close to an integer.

- Trivially implementable in $O(2^{k/2})$ for k shares.
- Can go up to $k \approx 60$ easily.
- Best complexity $O(2^{0.313k})$.

Overcoming obstacles

For computing $\{S(\alpha) \bmod \mathfrak{p}_{i,j}\}$, use subproduct trees.

- Compute $S(\alpha)$.
- Compute P^{λ} as a subproduct tree.
- Make $S(\alpha)$ descend along the tree.

The method achieves the same complexity as the lifting approach.

- Waived the inert prime assumption.
- Possible to parallelize: on t^2 nodes, t-fold reduction in both time and space complexity on each node.

Plan

Montgomery's algorithm

The specificity of Montgomery's algorithm is that it computes a square root of $S(\alpha) \in \mathbb{Z}[\alpha]$ by exploiting

$$S(\alpha)\mathcal{O}_K = \prod_i \mathfrak{p}_i^{2e_i}$$

where all $\mathfrak{p}_i$ and e_i are known.

Note that Montgomery's algorithm is also valid for computing arbitrary λ-th roots.

A bit of history look back

When is it described ? Known three versions by Montgomery.

- 1994; proceedings of a conference celebrating 50th anniversary of Math. Comp. Published as AMS PSAM #48, 1994, p. 567-571. Text says: “the final version of this paper will be submitted for publication elsewhere”.
- 1995; draft dated May.
- 1997; draft dated May 16, 24 pages. Used to be available on the web. Lots of "HELP", "TBD" and so on. Most advanced version.
- 1998; P. Nguyen version in ANTS-III. A few extra remarks.
- 2012; E. Thomé. Survey article on these algorithms and the otherwise existing folklore.

A short note

The characters step gives a vector v s.t. $vM = 0$ over $\mathbb{F}_2$.

Montgomery's algorithm really finds a square root in the field $\mathbb{Q}(\alpha)$, not in $\mathbb{Z}[\alpha]$.

- We don't have to bother with $f_d'(\alpha)$.
- We're free to create our starting $S(\alpha)$ as a fraction.

A short note

Deciding on the starting $S(\alpha)$ from the coefficients of v:

The naive approach is to lift v to $\mathbb{Z}$ as
$\bar{0} \in \mathbb{F}_2 \to 0 \in \mathbb{Z}$
$\bar{1} \in \mathbb{F}_2 \to 1 \in \mathbb{Z}$

but we could also lift $\bar{1} \in \mathbb{F}_2$ to any odd integer, in particular -1.

Optimization trick

Instead of computing a square root for $\prod_i (a_i - b_i\alpha)$, depending on the algorithm, it might be more suitable to address $\prod_i (a_i - b_i\alpha)^{\epsilon_i}$ with $\epsilon_i = \pm 1$ a random coin flip.
For s terms, $|\nu_{\mathfrak{p}}|$ drops from expected s/p to $2\sqrt{s/p}$.

Cf random walks, etc.

Preliminaries

How does one force an algebraic number ζ to be small ?

- Answer 1: minimize the factorization of the ideal $\zeta\mathcal{O}_K$.

Preliminaries

How does one force an algebraic number ζ to be small ?

- Answer 1: minimize the factorization of the ideal $\zeta\mathcal{O}_K$.
- Answer 2: minimize the complex embeddings of ζ.

Let K have r real embeddings, and s pairs of complex embeddings.

$$\Phi : \begin{cases} K^\times \to \mathbb{R}^{r+s}, \\ \quad \zeta \mapsto (\log|\rho_1(\zeta)|, \dots, \log|\rho_r(\zeta)|, 2\log|\sigma_1(\zeta)|, \dots, 2\log|\sigma_s(\zeta)|). \end{cases}$$

We have $|\operatorname{Norm}(\zeta)| = \exp(\sum_i \Phi_i(\zeta))$.

This is as a crucial ingredient in the proof of the Unit theorem.

Logarithmic embeddings and finiteness

Consider an algebraic integer ζ.

- the coefficients of its characteristic polynomial χ are integers.
- if $\Phi(\zeta)$ is bounded, then so are all the symmetric functions on the complex embeddings.
- finite number of possible $\chi \rightsquigarrow$ finite number of possible ζ.
- the stricter the bound on $\Phi(\zeta)$, the fewer possible ζ.

Immediate corollary

The unit rank is $\leq r + s - 1$ (hyperplane $\sum = 0$ in $\mathbb{R}^{r+s}$).
The equality is moderately more expensive to obtain.

Bottom line: something which has trivial factorization and trivial embeddings is a torsion unit.

Do not compute the big thing!

In Montgomery's algorithm, we never compute $S(\alpha)$. We only keep track of its product form, and of valuations.

Montgomery's algorithm: key idea

This is an iterative algorithm, where at each step $k \geq 0$ we have:

$$S_k(\alpha) = S(\alpha)(\gamma_0^{\epsilon_0} \dots \gamma_{k-1}^{\epsilon_{k-1}})^{-\lambda}\mathcal{O} = \prod_{\mathfrak{p} \in \mathcal{F}} \mathfrak{p}^{\lambda \cdot e_{\mathfrak{p}}^{(k)}}.$$

- $\lambda = 2$ for square roots
- The notation ϵ_i denotes a sign ± 1.
- At each step k, we know the tuple $(e_{\mathfrak{p}}^{(k)})_{\mathfrak{p}}$.
  Positive exponents = numerator, negative = denominator.

Goal of step k: find a new multiplier γ_k.

- Objective 1: reduce the numerator or the denominator;
- Objective 2: reduce the complex embeddings.

Steps – meeting objective 1

How do we choose the new multiplier ?

- Target num. or den. depending on which has largest norm. WLOG, assume numerator, whence we set $\epsilon = 1$.
- Pick a subset $\mathcal{I}_k$ of the numerator ideals (pick $\mathfrak{p}$ at most $e_{\mathfrak{p}}^{(k)}$ times).
- $\mathcal{I}_k$ is an ideal, i.e. a $\mathbb{Z}$-lattice. Find a "nice" element in $\mathcal{I}_k$.

Lattice reduction (e.g. LLL) on a basis of $\mathcal{I}_k$ provides small generating elements. Basis elements v_i satisfy:

$$\mathrm{Norm}_{K/\mathbb{Q}}(\mathcal{I}_k) \mid \mathrm{Norm}_{K/\mathbb{Q}}(v_i),$$
$$m(v_i) \overset{\mathrm{def}}{=} \left| \frac{\mathrm{Norm}_{K/\mathbb{Q}}(v_i)}{\mathrm{Norm}_{K/\mathbb{Q}}(\mathcal{I}_k)} \right| \leq C_K,$$

with C_K an effectively computable constant (depends only on K).

Steps – meeting objective 1

For the new multiplier γ_k, we may choose one of the v_i.

- This kills the valuation from the ideals in $\mathcal{I}_k$ in the numerator.
- New ideals pop up in the denominator, with norm at most C_K.
- If we can cope with $\mathrm{Norm}(\mathcal{I}_k)$ suitably larger than C_K, we're on the right track.

Repeating this procedure will lead to something with trivial (or very small) ideal factorization.

BUT we have not taken care of the complex embeddings.
$\Rightarrow$ the resulting element will be an unacceptably large unit.

Steps – meeting objective 2

At step k, pick instead a combination $\sum c_i v_i$ so that:

- coefficients c_i remain small;
- cplx. emb. of the updated-unknown-square-root-to-discover are reasonably small, too.
  Note: uusrtd $= v_i \cdot S_k(\alpha)^{-\epsilon_k/2}$ if we are to take $\gamma_k = v_i$.

IOW, a second lattice reduction step in dimension $2d$:

$$w_i = \Big((\text{coeffs of } v_i), \Big(M \cdot |\pi(v_i \cdot S_k(\alpha)^{-\epsilon_k/2})|\Big)_\pi\Big)$$

where π runs through $\rho_1, \ldots, \rho_r, \sigma_1, \overline{\sigma_1}, \ldots, \sigma_s, \overline{\sigma_s}$ (d embeddings in total), and M is a scaling constant so that the lattice coefficients are overall balanced.

That’s it

This is about all there is to it. Nothing terrible.

Known implementations:

- Original Pari-GP implementation probably buried in the CWI sources.
- Textbook-accurate implementation by Chris Monico in Ggnfs.
- Magma toy implementation (from 2007) buried in the cado-nfs history.
- C implementation by F. Bahr, J. Franke and T. Kleinjung, hardly available.

Only the last one is parallelized.

Parallelizing Montgomery's sqrt

It is possible to treat $S(\alpha) = \prod_i (a_i - b_i\alpha)$ in batches.

Core operation:

$\{(a_i, b_i)\}$ + factorizations $\rightsquigarrow$ multipliers + factorization of quotient.

The first pass on the input data is inevitably I/O bound.

- list of relation-sets;
- list relevant (a, b) pairs;
- for each (a, b) pair, we need to access/compute the factorization;
- of course the whole relation dataset won't fit in memory.

Stream the matrix (complete relation dataset) in memory. One pass should be enough, provided we dispatch on several nodes.

Using Montgomery's algorithm for λ-th roots

When d is constant:

- $\lambda = 2$; $S(\alpha) = \prod_{(a,b)\in\mathcal{S}}(a - b\alpha)$: $\text{size}(S) = \Theta(\text{size}(\mathcal{S}))$.
- general case: $S(\alpha) = \prod_{(a,b)\in\mathcal{S}}(a - b\alpha)^{k_i}$ with $\overline{(k_i)} = \Theta(\lambda)$: $\text{size}(S) = \Theta(\lambda\text{size}(\mathcal{S}))$.

Complexity keypoint

Montgomery's algorithm is sensible to the size of the set $\mathcal{S}$, not of the product $S(\alpha)$ (which is never computed).

Possible to successfully run Montgomery-like e-th roots for e in the 10^{15} range.

Plan

CRT versus lifting

The CRT algorithm has been implemented in Cado-NFS.

- Has been used to (re-)do the sqrt for RSA-768.
  - Computation time 6 hours on 144 Intel cores.
  - Montgomery: 2 hours on 12 nodes (one of 12 cores on each, I/O + memory bound), + 2 hours on 1 node (12 cores), total 4 hours wall-clock mobilizing similar resources as above.
- Has also been used for a 190-digit SNFS.

Beyond that, the lifting approach remains the preferred one.

Crazy optimizations

Waiting for the globally insignificant sqrt computation to complete is annoying.

Improvements in November 2019, before the RSA-240 square root:

- parallel I/O all over the place!
- more parallelization in various pieces of the code.

Conclusion

Asymptotically fast algorithms are fast.

- Lifting approach has some limitations:
    - Needs inert primes.
    - Memory-hungry (probably soon no longer a problem).
- New CRT approach.
    - Nice parallelizable version of the lifting approach.
    - Waives the number field assumptions.
    - Almost competitive with Montgomery's algorithm.

On the other hand, code size in the end almost invalidates the "lazy programmer" assumption that this is more pragmatic than coding all the background for Montgomery's algorithm.