CADO-NFS: An Implementation of The Number Field Sieve

Paul Zimmermann

INRIA Nancy - Grand Est

History of the CADO project

CADO is sponsored by the French Research Agency (ANR).
It started in 2008 for 3 years.
It involves 3 teams: CACAO (INRIA Nancy), TANC (INRIA Saclay), and Gérald Tenenbaum's team (IECN Nancy).
Objectives:

- better understand how the Number Field Sieve works
- publish a state-of-the art implementation, not to break new records, but to routinely factor numbers of 155 digits
- use that code base to try new ideas and/or new algorithms

Download page

Snapshots of the code are available from:

`http://cado.gforge.inria.fr/software.en.html`

Distributed under the GNU Lesser General Public License (LGPL), v2.1 or later

Current main contributors to the code: Jérémie Detrey, Pierrick Gaudry, Alexander Kruppa, François Morain, Emmanuel Thomé, Paul Zimmermann.

Running example (c158)

158-digit cofactor of $2^{1105} - 1$ from Cunningham's project (with GNFS):

$$22154364133980674336470181109040831394717683475796744$$
$$21181885667014866964155323990066535093200031258473964$$
$$92745614364575964912676054513751759154942453418248 41$$

Factorization started May 8, 2009.

Notations

N: number to factor

$f(x)$: algebraic polynomial of degree d

$g(x)$: linear polynomial

$F(x, y) = y^d f(x/y)$: homogeneous algebraic polynomial

$G(x, y) = yg(x/y)$: homogeneous linear polynomial

Polynomial Selection

Implements Kleinjung's algorithm: *On polynomial selection for the general number field sieve*, Math. of Computation, 2006.

Two steps: (1) find polynomials with a small norm; (2) root sieve to improve root properties.

Sketch: choose $f(x) = a_d x^d + \cdots + a_0$, $g(x) = px - m$.
We want $N = a_d m^d + a_{d-1} m^{d-1} p + \cdots + a_0 p^d$, thus

$$N = a_d m^d \bmod p.$$

Fix d, a_d, ℓ. Choose $p = p_1 p_2 \ldots p_\ell$ with $p_j \equiv 1 \bmod d$. This ensures that $N = a_d x^d \bmod p_j$ has d solutions, and thus $N = a_d m^d \bmod p$ has d^ℓ solutions.

Take the d^ℓ solutions which are closer to $\tilde{m} = (N/a_d)^{1/d}$. Each solution leads to a polynomial $f(x)$ such that $p^d f(m/p) = N$ with

$$|a_{d-1}| < p + d a_d (m - \tilde{m})/p$$

and $|a_i| < p + m$ for $0 \leq i \leq d - 2$.

Similarly, Kleinjung shows that one can bound a_{d-2} in terms of a sum of linear terms depending on the roots of $N = a_d x^d \bmod p_j$. Small values of this sum can be found in $O(d^{\ell/2})$ using a meet-in-the-middle algorithm.

Root Properties

A polynomial $f(x)$ has good root properties when it has many roots modulo small primes. This means that we can expect the values taken by $F(a, b)$ to have many small prime factors. The root property is quantified by Murphy's α (Murphy's PhD, ANU, 1999).

Rotation

Once a polynomial pair $f(x), g(x) = px - m$ with small norm is found, improve the *root properties* with *rotation* by a multiple of $g(x)$:

$$f(x) + (\lambda x + \mu)g(x).$$

If λ, μ are not too large, this will not change much the norm of $f(x)$, but might improve the number of roots of $f(x)$ for small primes (say up to 2000).

The CADO-NFS implementation of the rotation is still very naive. We are currently implementing a improved version (*rootsieve*, Emmanuel Thomé and Antonio Vera).

Running example ($2^{1105} - 1$, c158)

We searched for a polynomial of degree $d = 5$.
p was chosen as the product of $\ell = 8$ primes < 1024, plus one extra prime $p_0 < 100000$.

a_d was forced to be a multiple of 60, in the range $a_d < 10^{11}$.

The norm bound was $2 \cdot 10^{23}$ (as defined by Kleinjung).

The rotation was bounded by $|\mu| < 2^{18}$.

Found polynomial pair

$$p = 7685161733689883 = 13 \cdot (11 \cdot 31 \cdot 101 \cdot 131 \cdot 151 \cdot 181 \cdot 191 \cdot 251)$$

$$\begin{aligned}
f(x) &= 58091538480x^5 \\
&- 1315689856261290x^4 \\
&+ 156765946044170484443x^3 \\
&+ 1042081179621456253068086x^2 \\
&- 2183136495145828996001649925x \\
&- 1111980717564699032500216442812 5 \\
g(x) &= 7685161733689883x - 20714253316791917649369553887 4
\end{aligned}$$

The norm of $f(x)$ is $\approx 6.6 \cdot 10^{20}$, skewness 11372, $\alpha \approx -3.83$.

One poly. with smaller norm found ($5.6 \cdot 10^{20}$), but larger α.

Many poly. with smaller α found (e.g., -6.27), but larger norm.

Total polynomial selection time about 30 cpu days (2.4Ghz Opteron).

Wall-clock time was about 34 hours.

Sieving

Implements *Continued Fractions and Lattice Sieving*, Jens Franke and Thorsten Kleinjung, Proceedings of SHARCS 2005.

(We first had a line siever implemented by Alexander Kruppa.)

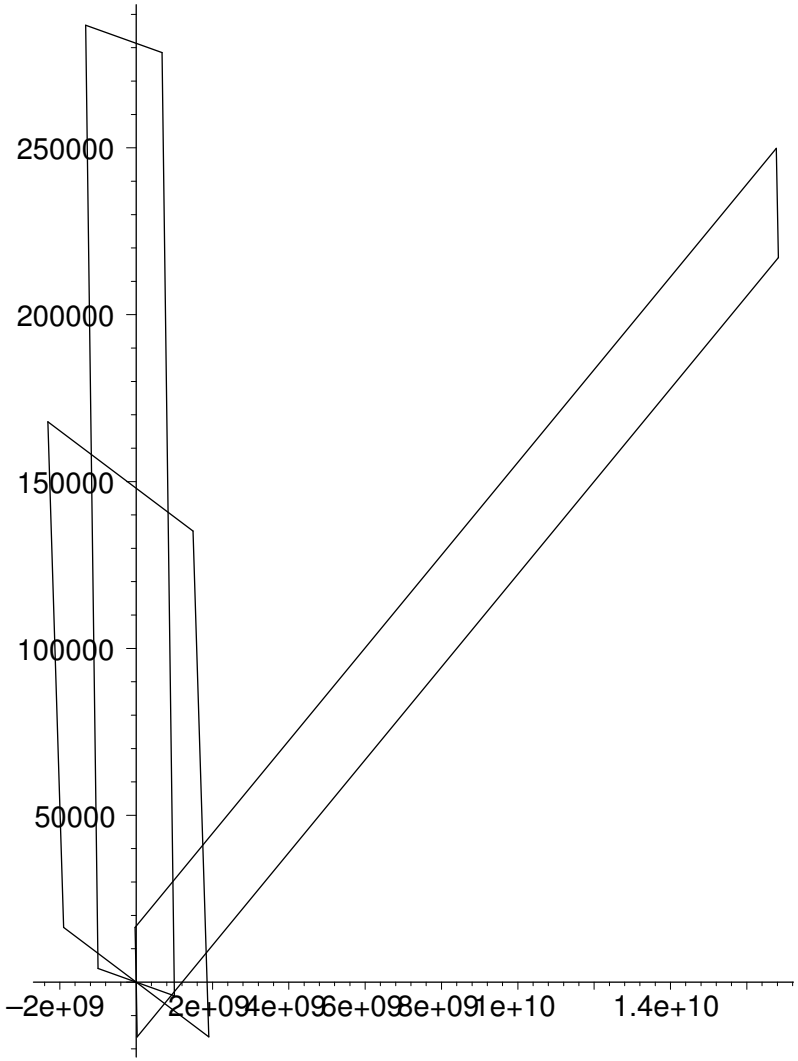
Take a prime q (special-q), and a root ρ of $f(x) \mod q$. The pairs (a, b) where (q, ρ) divides $F(a, b)$ form a lattice

$$a = iq + j\rho, b = j.$$

Find a "reduced basis" of that lattice (taking into account the skewness):

$$a = ia_0 + ja_1, b = ib_0 + jb_1.$$

Sieving Region for 3 special-q's for RSA-155



The sieving region is now a rectangle $-I/2 \leq i < I/2$, $0 \leq j < I/2$.

$F(a, b)$ becomes $F'(i, j)$, where for each i, j, q divides $F'(i, j)$ (thus we can consider $F'(i, j)/q$). Note that F' is not skewed any more.

We split factor base primes:

- small primes ($p < I$): they are sieved with classical (line) sieving
- medium primes ($p > I$): they are sieved with a special algorithm.

Sieving Medium Factor Base Primes $p > I$

In the (i, j) sieving rectangle $-I/2 \leq i < I/2$, $0 \leq j < I/2$, the (i, j) pairs divisible by an ideal (p, r) form themselves a lattice.

$p < I$: there is at least one match per line $j = j_0 \Rightarrow$ line sieving

$p > I$: only 0 or 1 sieved point per line $j = j_0$. Franke and Kleinjung show that one can compute an unique basis $\{(\alpha, \beta), (\gamma, \delta)\}$ of the p-sublattice with:

$$\beta, \delta > 0, \quad -I < \alpha \leq 0 \leq \gamma < I, \quad \gamma - \alpha \geq I$$

such that one jumps from (i, j) to (i', j'):

$$(i', j') = (i, j) + \begin{cases} (\alpha, \beta) & \text{if } i + \alpha \geq -I/2 \\ (\gamma, \delta) & \text{if } i + \gamma < I/2 \\ (\alpha, \beta) + (\gamma, \delta) & \text{if } i + \alpha < -I/2 \text{ and } I/2 \leq i + \gamma \end{cases}$$

Franke-Kleinjung's Lattice Sieving

Especially interesting for $p \gg I$

Counterpart: the initialization (computing $\alpha, \beta, \gamma, \delta$) is quite expensive. It amounts to an additive Euclidean sequence starting from $-p$ and r, stopping as soon as both numbers are smaller than I

Sieving: the CADO-NFS implementation

We use buckets (corresponding to the L1 cache) to store the sieve locations:

- the sieving itself fills the buckets
- then we apply the buckets to the sieve array, one bucket (local region) at a time

We use a quite precise estimation of the norm of $F(a, b)$ and $G(a, b)$, which is quite expensive.

Once the sieving is done, the division by factor base elements and the cofactorization (finding large primes) are done using algorithms and code developed by Alexander Kruppa (will be described in his PhD thesis).

Running Example (c158): Sieving Parameters

Factor base bound: 20M on the rational side, 40M on the algebraic side.

Large prime bound: 2^{30} on both sides.

Cofactor bound: 2^{60} on rational side (two large primes), 2^{90} on algebraic side (three large primes).

Sieving region $I = 2^{14}$ (sieve array of 2^{27} points).

Special-q range: 40M to about 105M.

Running Example (c158): Sample of Sieving Results

Sieving for $42,900,000 \leq q < 43,000,000$ on a 2.833Ghz Core 2.

Total 5638 special-q's sieved.

Average J (height of sieving region) is 6599.

Total sieving time 43791.7 seconds for 188860 reports (0.232s/r, 33.5r/sq).

Norm computation 7%, sieving 54% (medium primes 45%), cofactorization 39%.

Running Example (c158): Sieving Time

(Timings on 2.833Ghz Core 2.)

Sieving rate is 0.232s/r around 43M, goes down to 0.277s/r around 103M.

Consider “average” of 0.255s/r.

Found about 100M relations: about 10 cpu months.

Sieving started May 9, ended May 29: wall clock time 20 days.

Duplicate Removal

We use the approximate algorithm already used for RSA-155:
On the Number Field Sieve Integer Factorisation Algorithm,
Stefania Cavallar, PhD thesis, University of Leiden, 2002.

$$H(a,b) = 314159265358979323a + 271828182845904523b$$

$$h(a,b) = H(a,b) \bmod 2^{64}$$

Cavallar shows that a collision cannot occur in H, thus only in h.

Running Example ($2^{1105} - 1$, c158)

(Timings on 2.833Ghz Core 2.)

Input relations: $98,994,238$ (almost at end of sieving).

Hash table of $148,491,367$ cells (open addressing with linear probing).

Remains $82,622,897$ non-duplicates (17% duplicates).

Duplicate time is about 75 seconds (cpu).

Singleton Removal

We use an exact scheme, where all unique ideals are removed.

Use hash-table with open addressing and linear probing for storing ideals.

Hash-table contains p, root r mod p, and count (12 bytes per ideal for $p < 2^{32}$, 20 bytes otherwise!).

Pass 1 considers only ideals above the factor base bounds.

Pass 2 considers all ideals (including very small ones!)

If at the end of Pass 2, the excess is below a given value E_2, continue sieving.

Otherwise, remove relations until the excess is $E_1 < E_2$.

Running Example ($2^{1105} - 1$, c158)

(Timings on 2.833Ghz Core 2.)

We use $E_1 = 160$ and $E_2 = 2,000,000$.

Input: $82,622,897$ unique relations (almost at end of sieving)

Hash-table of $127,241,591$ entries (1456Mb).

Pass 1 takes 700 seconds and 3811Mb: $82,622,897$ relations with initial excess -297666.

End of pass 1: remains $41,748,672$ relations with excess $5,120,161$ (large primes only)

Pass 2 takes another 720 seconds and 5363Mb: remains $41,748,672$ relations with excess $1,416,714 < E_2$: continue sieving.

Pruning

Idea: generate more relations than needed, and remove some excess, hoping for a smaller (almost square) matrix.

A different strategy is to keep some excess for merging; in CADO-NFS we consider all ideals and keep only a very small excess, typically 160, that takes into account characters and the number of wanted dependencies.

The algorithm follows Cavallar's thesis ("clique removal").

A "clique" is a connected set of n relations linked by $n - 1$ ideals appearing exactly 2 times each.

If we remove one of those relations, an ideal becomes single, thus we can remove the whole connected set.

In CADO-NFS, we remove the sets with the largest n (and recompute the connected components from time to time).

Different weights are possible to define "heavy" connected components (see Cavallar's thesis).

Merging

If an ideal appears in k relations, we can replace them with $k-1$ relations without this ideal. This is a "k-merge".

Necessarily $k \geq 2$, otherwise we have missed a singleton.

For $k = 2$, we simply combine the two relations into a new one: the total weight decreases.

For $k > 2$, there are several ways to combine the relations. For $k = 3$ with relations A, B, C, one might output A+B, A+C, or B+A, B+C, or C+A, C+B.

In general, there are k^{k-2} ways to perform a k-merge (number of trees with k elements, Cayley numbers).

CADO-NFS implements two variants:

- SWAR: first perform all 2-merges, then all 3-merges, ...
- Markowitz: choose the best merge according to Markowitz pivoting

Linear Algebra

CADO-NFS implements block Wiedemann, with multi-thread and multi-processor code.

So far we have mainly used the multi-thread code with 4 cores.

This part of CADO-NFS was developed by Emmanuel Thomé and is (almost?) state-of-the-art.

Characters

We compute characters only *after* the linear algebra.

This is why we must use a slightly larger excess (say 160).

The input of the `characters` binary is a set of raw dependencies, the output is a set of (hopefully) true dependencies.

Square Root

CADO-NFS currenly implements a naive square root algorithm.

On the rational side we expand using a product-tree the product of all $G(a, b)$ for a given dependency, and take its integer square root.

On the algebraic side we do the same modulo an inert prime p, while reducing modulo the polynomial $f(x)$, then lift to some large enough power p^k, and map to integers.

This is more expensive than state-of-the-art algorithms, but was enough for us so far.

Choice of Parameters

CADO-NFS contains:

- several binaries obtained by compiling the corresponding source files: `polyselect`, `las`, `duplicates`, `purge`, `merge`, `bwc`, `characters`, `allsqrt`, `algsqrt`
- a `cadofactor.pl` Perl script, which takes as input a parameter file (say `c158.params`), a number N, a working directory, and performs everything automagically

Several sample parameter files are included in the CADO-NFS distribution.