Algorithms for sparse and black box matrices over finite fields

Erich Kaltofen
North Carolina State University
`google->kaltofen`

Factorization of an integer N (continued fraction, quadratic sieves, number field sieves)

Compute a solution to the congruence equation

$$X^2 \equiv Y^2 \pmod{N}$$

via r relations on b basis primes

$$X_1^2 \cdot X_2^2 \cdots X_r^2 \equiv (p_1^{e_1})^2 \cdot (p_2^{e_2})^2 \cdots (p_b^{e_b})^2 \pmod{N}$$

Then N divides $(X+Y)(X-Y)$, hence

$\mathrm{GCD}(X+Y, N)$ divides N

Relation computation

Step 1: Compute $s > r$ relations on b basis primes

$$\forall 1 \le i \le s\colon Y_i^2 \equiv p_1^{c_{i,1}} \cdot p_2^{c_{i,2}} \cdots p_b^{c_{i,b}} \pmod{N}$$

Step 2: select r relations $X_1 = Y_{i_1}, \ldots, X_r = Y_{i_r}$ such that

$$\forall 1 \le j \le b\colon c_{i_1,j} + c_{i_2,j} + \cdots + c_{i_r,j} \equiv 0 \pmod{2}$$

One must compute non-zero solutions to the
sparse homogeneous linear system modulo 2

$$\begin{bmatrix} x_1 & \ldots & x_s \end{bmatrix} \begin{bmatrix} c_{1,1} \bmod 2 & \ldots & c_{1,b} \bmod 2 \\ c_{2,1} \bmod 2 & \ldots & c_{2,b} \bmod 2 \\ \vdots & & \vdots \\ c_{2,1} \bmod 2 & \ldots & c_{2,b} \bmod 2 \end{bmatrix} \equiv \begin{bmatrix} 0 & \ldots & 0 \end{bmatrix} \pmod{2}$$

LDDMLtR’s RSA-120 matrix modulo 2

Row nr.	Columns with non-zero entries
1	0 1 481 1355 3b42 5cf6 c461 eda1 f0e7 15d19 199e0 2c317 33a50
2	0 1 9b4 f26 3214 7f99 a146 bc7e 10087 175c5 1953a 320b5 39425
⋮	⋮
245 811	0 1 2 3 4 6 8 9 b c d f 10 12 13 14 16 17 18 19 1d 1e 1f 20 25 26 . . . 3624a 36473 36905 37727 395eb

There are 10 – 217 non-zero entries/column, with 252 222 columns and 11 037 745 non-zero entries total; in the above format the matrix occupies 48 Mbytes of disc space.

```
challenge-rsa-honor-roll@rsa.com
RSA-155
Factors:
102639592829741105772054196573991675900716567808038066803341933521790711307779
*
106603488380168454820927220360012878679207958575989291522270608237193062808643
Date:    August 22, 1999
Method:  the General Number Field Sieve,
         with a polynomial selection method of Brian Murphy
         and Peter L. Montgomery,
         with lattice sieving (71%) and with line sieving (29%),
         and with  Peter L. Montgomery's blocked Lanczos and
         square root algorithms;
Time:    * Polynomial selection:
         The polynomial selection took approximately 100 MIPS years,
         equivalent to 0.40 CPU years on a 250 MHz processor.
         ...
         * Sieving: 35.7 CPU-years in total,
         ...
           124 722 179 relations were collected by eleven different sites,
         ...
         * Filtering the data and building the matrix took about a month
   * Matrix: 224 hours on one CPU of the Cray-C916 at SARA, Amsterdam;
             the matrix had  6 699 191 rows and  6 711 336 columns,
             and weight  417 132 631 ( 62.27 nonzeros per row);
             calendar time: ten days
   * Square root:  Four jobs assigned one dependency each were run
                   in parallel on separate 300 MHz R12000 processors
                   within a 24-processor SGI Origin 2000 at CWI.
                   One job found the factorisation after 39.4 CPU-hours,
   ...
   * The total calendar time for factoring RSA-155 was 5.2 months
     (March 17 - August 22)
     (excluding polynomial generation time)
     We could reduce this to one month sieving time and
     one month processing time if we had more sievers and
     had more experience with matrix-generation strategies.
Address: (of contact person)
Email:   Herman.te.Riele@cwi.nl
```

Factorization of polynomial f over finite field $\mathbb{F}_p$ (Berlekamp 1967 algorithm)

Note that since $a^p \equiv a \pmod{p}$ for all $a \in \mathbb{F}_p$ we have

$$x^p - x \equiv x \cdot (x-1) \cdot (x-2) \cdots (x-p+1) \pmod{p}$$

Compute a polynomial solution to the congruence equation

$$w(x)^p \equiv w(x) \pmod{f(x)}$$

Then f divides $w \cdot (w-1) \cdot (w-2) \cdots (w-p+1)$, hence

$\mathrm{GCD}(w(x)-a, f(x))$ divides $f(x)$ for some $a \in \mathbb{F}_p$

Solving $w^p \equiv w \pmod{f}$ by linear algebra

For $w(x) \in \mathbb{F}_p[x]$, $\deg(w) < n = \deg(f)$:

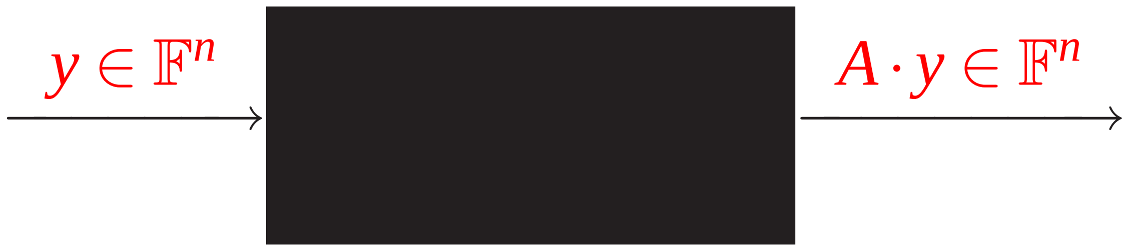
$$w(x)^p = w(x^p) \equiv w(x) \pmod{f(x)}$$

$$\Updownarrow$$

$$\overrightarrow{w(x^p) \bmod f(x)} = \underbrace{[w_0 \ldots w_{n-1}]}_{\vec{w}} \cdot \underbrace{\begin{bmatrix} \vdots \\ \overrightarrow{x^{ip} \bmod f(x)} \\ \vdots \end{bmatrix}_{0 \le i < n}}_{Q} = \vec{w}$$

(Petr's 1937 matrix)

Black box matrix concept



$A \in \mathbb{F}^{n \times n}$ singular
$\mathbb{F}$ an arbitrary, e.g., finite field

Perform linear algebra operations, e.g., $A^{-1}b$ [Wiedemann 86, Kaltofen & Saunders 91] with

$O(n)$	black box calls and
$n^2(\log n)^{O(1)}$	arithmetic operations in $\mathbb{F}$ and
$O(n)$	intermediate storage for field elements

Black box model is useful for dense, structured matrices

$$\begin{bmatrix} 1 & \dots & & \dots & \frac{1}{n} \\ & & \vdots & & \\ & & \frac{1}{i+j-1} & & \\ & & \vdots & & \\ \frac{1}{n} & \dots & & \dots & \frac{1}{2n-1} \end{bmatrix} \begin{bmatrix} x_1 \\ \vdots \\ x_n \end{bmatrix} = \begin{bmatrix} b_1 \\ \vdots \\ b_n \end{bmatrix}$$

(Hilbert matrix)

Savings is in space, not time: $O(1)$ vs. $O(n^2)$.

Idea for Wiedemann's algorithm

$A \in \mathbb{F}^{n \times n}$, $\mathbb{F}$ a (possibly finite) field

$\phi^A(\lambda) = c'_0 + \cdots + c'_m \lambda^m \in \mathbb{F}[\lambda]$ **minimum polynomial** of A

$\forall u, v \in \mathbb{F}^n\colon\ \forall j \geq 0\colon$

$$u^{Tr} A^j \phi^A(A) v = 0$$

$$\Updownarrow$$

$$c'_0 \cdot \underbrace{u^{Tr} A^j v}_{a_j} + c'_1 \cdot \underbrace{u^{Tr} A^{j+1} v}_{a_{j+1}} + \cdots + c'_m \cdot \underbrace{u^{Tr} A^{j+m} v}_{a_{j+m}} = 0$$

$$\Updownarrow$$

$\{a_0, a_1, a_2, \ldots\}$ is generated by a linear recursion

Theorem [Wiedemann 1986]: *For* ***random*** $u, v \in \mathbb{F}^n$, *a linear generator for* $\{a_0, a_1, a_2, \ldots\}$ *is one for* $\{I, A, A^2, \ldots\}$.

$$\forall\, j \geq 0: \; c_0 a_j + c_1 a_{j+1} + \cdots + c_d a_{j+d} = 0$$

$$\Big\Downarrow \text{(with high probability)}$$

$$c_0 A^j v + c_1 A^{j+1} v + \cdots + c_d A^{j+d} v = \mathbf{0}$$

$$\Big\Downarrow \text{(with high probability)}$$

$$c_0 A^j + c_1 A^{j+1} + \cdots + c_d A^{j+d} = \mathbf{0}$$

that is, with high probability $\phi^A(\lambda)$ divides $c_0 + c_1\lambda + \cdots + c_d\lambda^d$

Algorithm homogeneous Wiedemann

Input: $A \in \mathbb{F}^{n\times n}$ singular
Output: $w \neq \mathbf{0}$ such that $Aw = \mathbf{0}$

Step W1: Pick random $u, v \in \mathbb{F}^n$; $\quad b \leftarrow Av$;
for $i \leftarrow 0$ **to** $2n-1$ **do** $a_i \leftarrow u^{Tr}A^i b$.
(Requires $2n$ black box calls.)

Step W2: Compute a linear recurrence generator for $\{a_i\}$,
$c_\ell \lambda^\ell + c_{\ell+1}\lambda^{\ell+1} + \cdots + c_d \lambda^d, \quad \ell \geq 0, d \leq n, c_\ell \neq 0.$

Step W3: $\widehat{w} \leftarrow c_\ell v + c_{\ell+1}Av + \cdots + c_d A^{d-\ell} v$;
(With high probability $\widehat{w} \neq 0$ and $A^{\ell+1}\widehat{w} = 0$.)
Compute first k with $A^k \widehat{w} = 0$; **return** $w \leftarrow A^{k-1}\widehat{w}$.
(Requires $\leq n$ black box calls.)

Step W2 detail

Coefficients $c_0, \dots, c_n$ can be found by computing a non-trivial solution to the Toeplitz system

$$\begin{bmatrix} a_n & a_{n-1} & \dots & & a_1 & a_0 \\ a_{n+1} & a_n & & & a_2 & a_1 \\ \vdots & a_{n+1} & \ddots & & \vdots & a_2 \\ & \vdots & & & & \vdots \\ a_{2n-2} & & & \ddots & a_{n-1} & \\ a_{2n-1} & a_{2n-2} & \dots & & a_n & a_{n-1} \end{bmatrix} \cdot \begin{bmatrix} c_n \\ c_{n-1} \\ c_{n-2} \\ \vdots \\ \\ c_0 \end{bmatrix} = \mathbf{0}$$

or by the Berlekamp/Massey algorithm.

Cost: $O(n(\log n)^2 \log\log n)$ arithmetic ops.

Many recent results

Lambert [96], Teitelbaum [98], Eberly & Kaltofen [97]	relationship of Wiedemann and Lanczos approach
Villard [97]	analysis of **block** Wiedemann algorithm
Giesbrecht [97] and Mulders & Storjohann [99]	computation of **diophantine** solutions
Chen, Eberly, Kaltofen, Saunders, Villard & Turner [2K]	butterfly network, sparse and diagonal preconditioners
Villard [2K] & Storjohann [01]	characteristic polynomial
Kaltofen & Villard [04] Storjohann [05]	fast algorithm for determinant of a **dense** integer matrix
Villard & Jeannerod [04]	optimal algorithm for inverse of a **dense polyn.** matrix
Eberly, Giesbrecht, Giorgi Storjohann, Villard [06]	faster rational solution of sparse systems

LINSOLVE0: *Given blackbox A, compute $w \neq 0$ such that $Aw = 0$.*

NONSINGULAR$\leq$LINSOLVE0: For $Ax = b$ solve $\left[A \mid -b\right] w = 0$ and compute $x = \frac{1}{w_{n+1}} \begin{bmatrix} w_1 \\ \dots \\ w_n \end{bmatrix}$.

Harder (?) problem
LINSOLVE1: *Given blackbox A (possibly singular) and b, compute x such that $Ax = b$.*

Random sampling in the nullspace is equivalent to LINSOLVE1: select a random vector y and solve $Ax = b$ for $b = Ay$.

LinSolve1 via preconditioning

Suppose the minpoly of A is $1 \cdot \lambda + \cdots + c_m \lambda^m$
(the canonical form of A has "no nil-potent blocks.")

If $Ax = b$ is consistent then $b = A \cdot y$

hence $1 \cdot b + \cdots + c_m A^{m-1} b = 1 \cdot Ay + \cdots + c_m A^m y = 0$

so $b = A \cdot (\underbrace{-c_2 b - \cdots - c_m A^{m-2} b}_{x})$.

In [Chen *et al.* 2000] it is shown that Wiedemann's random sparse matrix multipliers give $\widetilde{A}$ the above property:

$\widetilde{A} = LAR$ where L, R are certain sparse 0-1 matrices

Note: L, R have $O(n(\log n)^2)$ non-zero entries.

Diophantine solutions
by Giesbrecht:
Find several rational solutions.

$$A(\tfrac{1}{2}x^{[1]}) = b, \quad x^{[1]} \in \mathbb{Z}^n$$
$$A(\tfrac{1}{3}x^{[2]}) = b, \quad x^{[2]} \in \mathbb{Z}^n$$

$$\gcd(2,3) = 1 = 2 \cdot 2 - 1 \cdot 3$$
$$A(2x^{[1]} - x^{[2]}) = 4b - 3b = b$$

Hensel lifting [Moenck and Carter 1979, Dixon 1982]:

1: For $j = 0, 1, \ldots, k$ and a prime p Do

Compute $\overline{x}^{[j]} = x^{[0]} + px^{[1]} + \cdots + p^j x^{[j]} \equiv x \pmod{p^{j+1}}$

1.a. $\widehat{b}^{[j]} = \dfrac{b - A\overline{x}^{[j-1]}}{p^j} = \dfrac{\widehat{b}^{[j-1]} - Ax^{[j-1]}}{p}$

1.b. Solve $Ax^{[j]} \equiv \widehat{b}^{[j]} \pmod{p}$ reusing the minpoly of $A \bmod p$

2: Recover denominators of x_i by continued fractions of $\overline{x}_i^{[k]}/p^k$.

Coppersmith's 1992 blocking

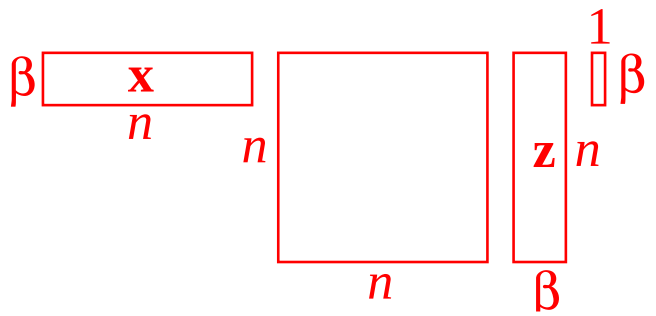
Use of the block vectors $\mathbf{x} \in \mathbb{F}^{n\times\beta}$ in place of u

$\mathbf{z} \in \mathbb{F}^{n\times\beta}$ in place of v

$$\mathbf{a}_i = \mathbf{x}^{Tr} A^{i+1} \mathbf{z} \in \mathbb{F}^{\beta\times\beta}, \quad 0 \le i < 2n/\beta + 2.$$

Find a **vector** polynomial $c_\ell \lambda^\ell + \cdots + c_d \lambda^d \in \mathbb{F}^\beta[\lambda]$, $d = \lceil n/\beta \rceil$:

$$\forall\, j \ge 0: \ \sum_{i=\ell}^{d} \mathbf{a}_{j+i} c_i = \sum_{i=\ell}^{d} \mathbf{x}^{Tr} A^{i+j} A \mathbf{z} c_i = \mathbf{0} \in \mathbb{F}^{\beta\times\beta}$$

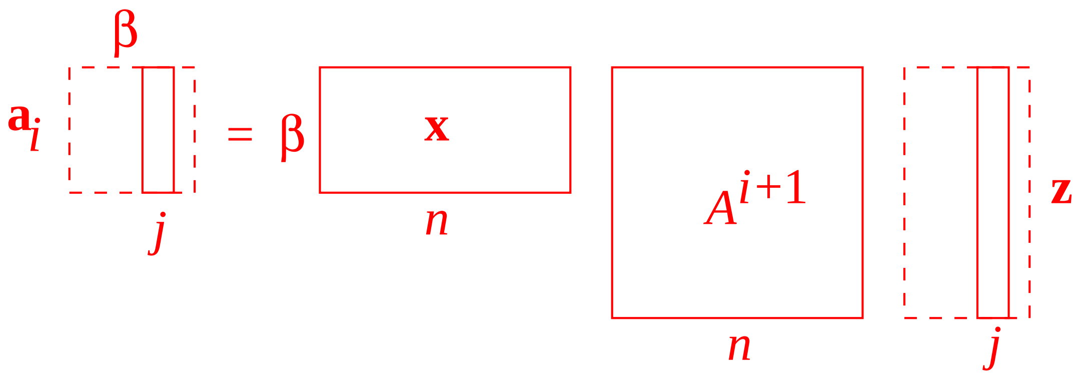


Then, analogously to before, with high probability

$$\widehat{w} = \sum_{i=\ell}^{d} A^{i-\ell} \mathbf{z} c_i \neq \mathbf{0}, \quad A^{\ell+1} \widehat{w} = \sum_{i=\ell}^{d} A^i A \mathbf{z} c_i = \mathbf{0} \in \mathbb{F}^n$$

Advantages of blocking

1. Parallel coarse- and fine-grain implementation



The j^{th} processor computes the j^{th} column of the sequence of (small) matrices.

2. Faster sequential running time:
   multiple solutions [Coppersmith; Montgomery 1994];
   $1+\varepsilon$ matrix times vector ops [Kaltofen 1995];
   determinant, charpoly, Smith form [Kaltofen & Villard 2004];
   charpoly of sparse matrix [Villard & Storjohann 2001]

3. Better probability of success [Villard 1997]

Computation of the matrix linear generator

Explicitly in Popov form by block Berlekamp/Massey algorithm [Rissanen 1972, Dickinson et al. 1974, Coppersmith 1994, Thomé 2001, Kaltofen & Yuhasz 2006] or implicitly in a block Lanczos version

Explicitly by a power Hermite-Padé approximation [Beckermann & Labahn 1994]

By a block Toeplitz solver [Kaltofen 1995]

Implementations

By Coppersmith, Kaltofen & Lobo, Montgomery, Dumas, Brent,...

The LinBox project [Canada: UWO, Calgary; France: ENS Lyon, IMAG Grenoble; USA: Delaware, NCSU, Washington Coll. MD]: A generic C++ library for black box linear algebra, including integer problems (“Symbolic MatLab” [`www.linalg.org`])

New abstraction mechanism black box matrix

Programming languages C++, Maple, GAP, C (Saclib)

Design principle
genericity through template parameter types (matrix entries)
and black box matrix model (sparseness and structuredness)

Open Problems

Large fields: Compute the characteristic polynomial
Certify the minimal polynomial
$\textsc{LinSolve}1 \le \textsc{LinSolve}0$

Small fields: Compute the determinant, rank of a sparse/blackbox
matrix without $O(\log n)$ slowdown