

Fast prime field Gröbner basis computation

Bjarke Hammersholt Rouné
Department of Mathematics
University of Kaiserslautern

Kaiserslautern, February 14, 2013

Part I Relevant background on F_4

The F_4 Algorithm

$ABCD$ decomposition

Martani's modification

Part II What I have been doing

The story of F_4 in Mathic

Matrix construction

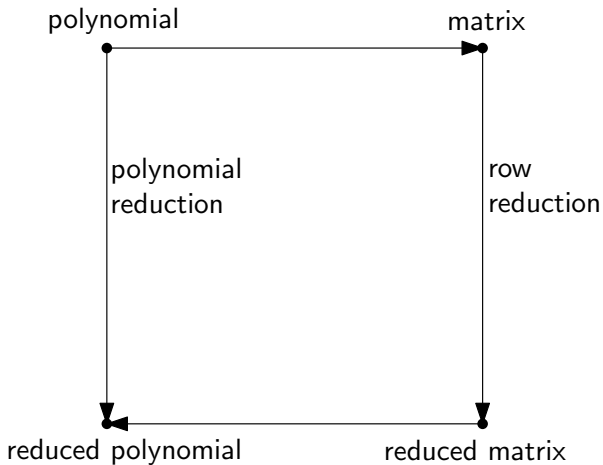
Part III The future

Part I

The F_4 algorithm

Based on “A New Efficient Algorithm for Computing Gröbner Bases (F_4)” by Jean-Charles Faugère

F4 is like classic reduction, but in a matrix.



A polynomial reduction

$$2x^2 - y : (x - 1, y + 2)$$

$$2x^2 - y \xrightarrow{2x(x - 1) = 2x^2 - 2x} 2x - y$$

$$2x - y \xrightarrow{2(x - 1) = 2x - 2} -y + 2$$

$$-y + 2 \xrightarrow{(-1)(y + 2) = -y - 2} 4$$

Let us do that in a matrix

$$2x^2 - y : (x - 1, y + 2)$$

$$2x^2 - y \xrightarrow{2x(x-1) = 2x^2 - 2x} 2x - y$$

$$2x - y \xrightarrow{2(x-1) = 2x - 2} -y + 2$$

$$-y + 2 \xrightarrow{(-1)(y+2) = -y - 2} 4$$

$$\begin{bmatrix} x^2 & x & y & 1 \\ 1 & -1 & 0 & 0 \\ 0 & 1 & 0 & -1 \\ 0 & 0 & 1 & 2 \\ \hline 2 & 0 & -1 & 0 \end{bmatrix} \begin{matrix} x^2 - x \\ x - 1 \\ y + 2 \\ 2x^2 - y \end{matrix}$$

$$2x^2 - y : (x - 1, y + 2)$$

$$\begin{array}{c}
 2x^2 - y \xrightarrow{2x(x-1) = 2x^2 - 2x} 2x - y \\
 \left[\begin{array}{cccc} x^2 & x & y & 1 \\ 1 & -1 & 0 & 0 \\ 0 & 1 & 0 & -1 \\ 0 & 0 & 1 & 2 \\ \hline 2 & 0 & -1 & 0 \end{array} \right] \begin{array}{l} x^2 - x \\ x - 1 \\ y + 2 \\ 2x^2 - y \end{array} \xrightarrow{\quad} \left[\begin{array}{cccc} x^2 & x & y & 1 \\ 1 & -1 & 0 & 0 \\ 0 & 1 & 0 & -1 \\ 0 & 0 & 1 & 2 \\ \hline 0 & 2 & -1 & 0 \end{array} \right] \begin{array}{l} x^2 - x \\ x - 1 \\ y + 2 \\ 2x - y \end{array}
 \end{array}$$

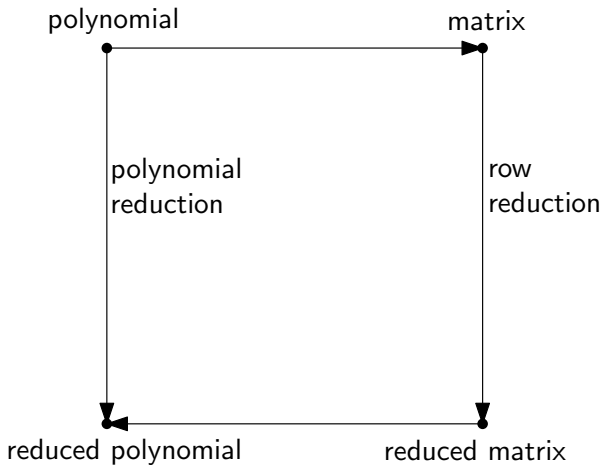
$$2x^2 - y : (x - 1, y + 2)$$

$$\begin{array}{ccc}
 2x - y & \xrightarrow{2(x - 1) = 2x - 2} & -y + 2 \\
 \left[\begin{array}{cccc|c} x^2 & x & y & 1 & \\ 1 & -1 & 0 & 0 & x^2 - x \\ 0 & 1 & 0 & -1 & x - 1 \\ 0 & 0 & 1 & 2 & y + 2 \\ 0 & 2 & -1 & 0 & 2x - y \end{array} \right] & \longrightarrow & \left[\begin{array}{cccc|c} x^2 & x & y & 1 & \\ 1 & -1 & 0 & 0 & x^2 - x \\ 0 & 1 & 0 & -1 & x - 1 \\ 0 & 0 & 1 & 2 & y + 2 \\ 0 & 0 & -1 & 2 & -y + 2 \end{array} \right]
 \end{array}$$

$$2x^2 - y : (x - 1, y + 2)$$

$$\begin{array}{c}
 \begin{array}{cccc|l}
 x^2 & x & y & 1 & \\
 1 & -1 & 0 & 0 & x^2 - x \\
 0 & 1 & 0 & -1 & x - 1 \\
 0 & 0 & 1 & 2 & y + 2 \\
 \hline
 0 & 0 & -1 & 2 & -y + 2
 \end{array}
 \end{array}
 \xrightarrow{(-1)(y+2) = -y-2}
 \begin{array}{c}
 \begin{array}{cccc|l}
 x^2 & x & y & 1 & \\
 1 & -1 & 0 & 0 & x^2 - x \\
 0 & 1 & 0 & -1 & x - 1 \\
 0 & 0 & 1 & 2 & y + 2 \\
 \hline
 0 & 0 & 0 & 4 & 4
 \end{array}
 \end{array}$$

What about that top arrow?



How to construct the matrix before doing the reduction?

$$2x^2 - y : (x - 1, y + 2)$$

$$\begin{array}{lcl}
 2x^2 - y & \xrightarrow{2x(x-1) = 2x^2 - 2x} & 2x - y \\
 2x - y & \xrightarrow{2(x-1) = 2x - 2} & -y + 2 \\
 -y + 2 & \xrightarrow{(-1)(y+2) = -y - 2} & 4
 \end{array}
 \quad
 \begin{array}{c}
 \begin{matrix} x^2 & x & y & 1 \end{matrix} \\
 \left[\begin{array}{cccc|c}
 1 & -1 & 0 & 0 & x^2 - x \\
 0 & 1 & 0 & -1 & x - 1 \\
 0 & 0 & 1 & 2 & y + 2 \\
 2 & 0 & -1 & 0 & 2x^2 - y
 \end{array} \right]
 \end{array}$$

$$2x^2 - y : (x - 1, y + 2)$$

Start with $2x^2 - y$

$$\left[\begin{array}{cc} x^2 & y \\ \hline 2 & -1 \end{array} \right] 2x^2 - y$$

$$2x^2 - y : (x - 1, y + 2)$$

$$x(x - 1) = x^2 - x \text{ reduces } x^2$$

$$\begin{array}{c} \downarrow \\ \begin{array}{cc} x^2 & y \\ \hline 2 & -1 \end{array} \end{array} 2x^2 - y \longrightarrow \begin{array}{ccc} x^2 & x & y \\ \hline 1 & -1 & 0 \end{array} \begin{array}{c} x^2 - x \\ 2x^2 - y \end{array}$$

$$2x^2 - y : (x - 1, y + 2)$$

$x - 1$ reduces x

$$\left[\begin{array}{ccc|c} x^2 & \downarrow x & y & \\ 1 & -1 & 0 & x^2 - x \\ \hline 2 & 0 & -1 & 2x^2 - y \end{array} \right] \longrightarrow \left[\begin{array}{cccc|c} x^2 & x & y & 1 & \\ 1 & -1 & 0 & 0 & x^2 - x \\ 0 & 1 & 0 & -1 & x - 1 \\ \hline 2 & 0 & -1 & 0 & 2x^2 - y \end{array} \right]$$

$$2x^2 - y : (x - 1, y + 2)$$

$y + 2$ reduces y

$$\left[\begin{array}{cccc} x^2 & x & y & 1 \\ 1 & -1 & 0 & 0 \\ 0 & 1 & 0 & -1 \\ \hline 2 & 0 & -1 & 0 \end{array} \right] \begin{array}{l} x^2 - 1x \\ x - 1 \\ 2x^2 - y \end{array} \longrightarrow \left[\begin{array}{cccc} x^2 & x & y & 1 \\ 1 & -1 & 0 & 0 \\ 0 & 1 & 0 & -1 \\ \hline 0 & 0 & 1 & 2 \\ 2 & 0 & -1 & 0 \end{array} \right] \begin{array}{l} x^2 - x \\ x - 1 \\ y + 2 \\ 2x^2 - y \end{array}$$

$$2x^2 - y : (x - 1, y + 2)$$

We have no reducer for 1

$$\begin{array}{c}
 \begin{array}{cccc}
 x^2 & x & y & \overset{\text{red}}{\downarrow} 1 \\
 1 & -1 & 0 & 0 \\
 0 & 1 & 0 & -1 \\
 0 & 0 & 1 & 2 \\
 \hline
 2 & 0 & -1 & 0
 \end{array}
 \begin{array}{l}
 x^2 - x \\
 x - 1 \\
 y + 2 \\
 2x^2 - y
 \end{array}
 \end{array}
 \longrightarrow
 \begin{array}{c}
 \begin{array}{cccc}
 x^2 & x & y & 1 \\
 1 & -1 & 0 & 0 \\
 0 & 1 & 0 & -1 \\
 0 & 0 & 1 & 2 \\
 \hline
 2 & 0 & -1 & 0
 \end{array}
 \begin{array}{l}
 x^2 - x \\
 x - 1 \\
 y + 2 \\
 2x^2 - y
 \end{array}
 \end{array}$$

$$2x^2 - 2x : x - 1$$

F4 unnecessarily adds the row $x - 1$.

$$2x^2 - 2x \xrightarrow{2x(x-1) = 2x^2 - 2x} 0 \quad \left[\begin{array}{ccc|c} x^2 & x & 1 & \\ \hline 1 & -1 & 0 & x^2 - x \\ 0 & 1 & -1 & x - 1 \\ \hline 2 & -2 & 0 & 2x^2 - 2x \end{array} \right]$$

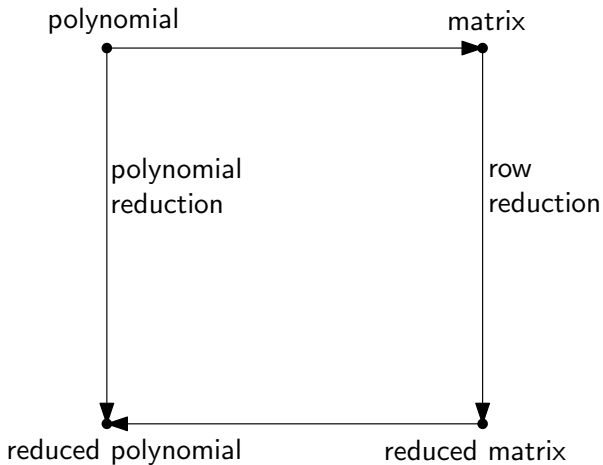
$$2x^2 - 2x : x - 1$$

F4 unnecessarily adds the row $x - 1$.

$$2x^2 - 2x \xrightarrow{2x(x-1) = 2x^2 - 2x} 0 \quad \left[\begin{array}{ccc|c} x^2 & x & 1 & \\ \hline 1 & -1 & 0 & x^2 - x \\ 0 & 1 & -1 & x - 1 \\ \hline 2 & -2 & 0 & 2x^2 - 2x \end{array} \right]$$

There will be n unnecessary rows for $x^{n+1} - x^n : x - 1$.

You now understand F4



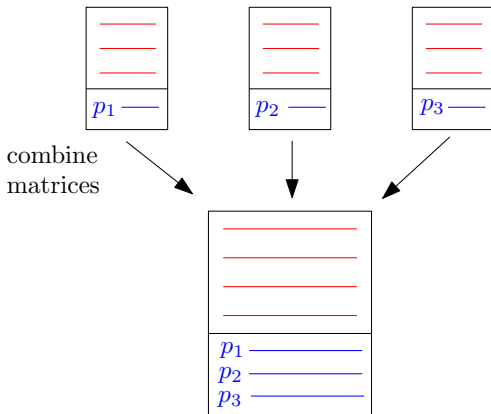
Benchmark for homogeneous cyclic-8 mod 101

reducer	time/s	\sum classic reducers or
		\sum matrix top rows
classic	104.1	6,828,742
matrix-based	85.0	7,000,845

Main point: About same speed for full Gröbner basis computation.

Optimization: bunch the S-pair reductions together

many smaller matrices/reductions ...



... turn into one larger matrix

Optimization: bunch the S-pair reductions together

$$x^2 - xy^2 : xy - 1 \quad \text{and} \quad x^3 - 2xy^2 : xy - 1$$

The two reductions can share the $y(xy - 1) = xy^2 - y$ row.

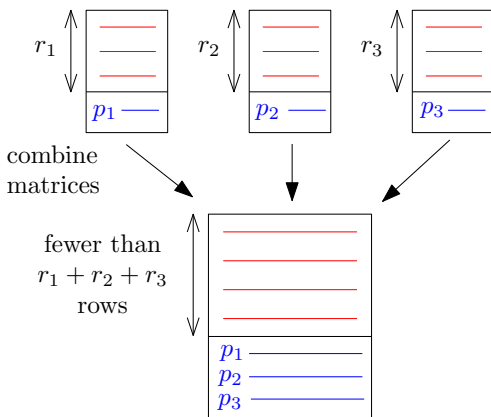
$$\begin{array}{ccc} x^2 & xy^2 & y \\ \left[\begin{array}{ccc|l} 0 & 1 & -1 & xy^2 - y \\ 1 & -1 & 0 & x^2 - xy^2 \end{array} \right] & & \end{array}$$
$$\begin{array}{ccc} x^3 & xy^2 & y \\ \left[\begin{array}{ccc|l} 0 & 1 & -1 & xy^2 - y \\ 1 & -2 & 0 & x^3 - 2xy^2 \end{array} \right] & & \end{array}$$

take union of rows
and of columns

$$\begin{array}{cccc} x^3 & x^2 & xy^2 & y \\ \left[\begin{array}{cccc|l} 0 & 0 & 1 & -1 & xy^2 - y \\ 0 & 1 & -1 & 0 & x^2 - xy^2 \\ 1 & 0 & -2 & 0 & x^3 - 2xy^2 \end{array} \right] & & \end{array}$$

Optimization: bunch the S-pair reductions together

many smaller matrices/reductions ...



... turn into one larger matrix

Benchmark for homogeneous cyclic-8 mod 101

reducer	time/s	\sum classic reducers or \sum matrix top rows
classic	104.1	6,828,742
many smaller matrices	85.0	7,000,845

Benchmark for homogeneous cyclic-8 mod 101

reducer	time/s	\sum classic reducers or \sum matrix top rows
classic	104.1	6,828,742
many smaller matrices	85.0	7,000,845
fewer bigger matrices	4.3	76,149

Part I Relevant background on F_4

The F_4 Algorithm

$ABCD$ decomposition

Martani's modification

Part II What I have been doing

The story of F_4 in Mathic

Matrix construction

Part III The future

Part I

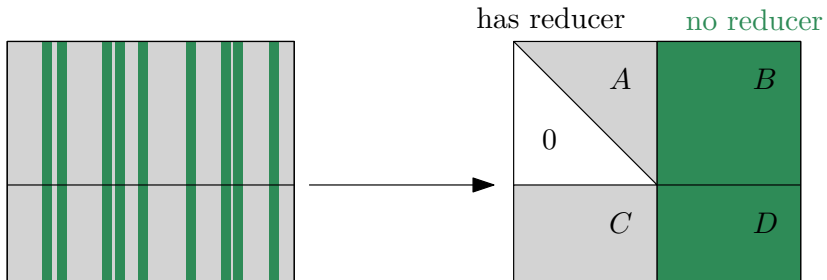
ABCD decomposition

Based on “Parallel Gaussian Elimination for Gröbner bases computations in finite fields” by Jean-Charles Faugère and Sylvain Lachartre.

Move columns without a reducer/pivot to the right

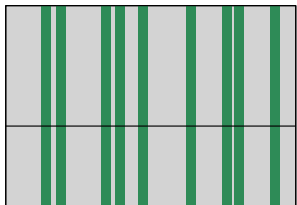
$$\begin{array}{c}
 \begin{bmatrix} \underline{1} & 7 & 9 & 0 & 0 & 6 \\ 0 & 0 & \underline{1} & 4 & 0 & 0 \\ 0 & 0 & 0 & 0 & \underline{1} & 0 \\ \hline 3 & 1 & 0 & 3 & 2 & 0 \\ 0 & 5 & 0 & 0 & 4 & 0 \end{bmatrix}
 \end{array}
 \longrightarrow
 \begin{array}{c}
 \begin{array}{cc} \text{has reducer} & \text{no reducer} \end{array} \\
 \begin{bmatrix} \underline{1} & 9 & 0 & | & 7 & 0 & 6 \\ 0 & \underline{1} & 0 & | & 0 & 4 & 0 \\ 0 & 0 & \underline{1} & | & 0 & 0 & 0 \\ \hline 3 & 0 & 2 & | & 1 & 3 & 0 \\ 0 & 0 & 4 & | & 5 & 0 & 0 \end{bmatrix}
 \end{array}$$

ABCD decomposition

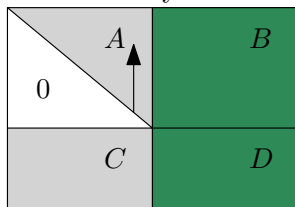


Faugère-Lachartre algorithm

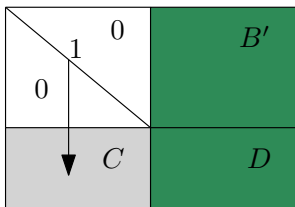
decompose into A, B, C, D



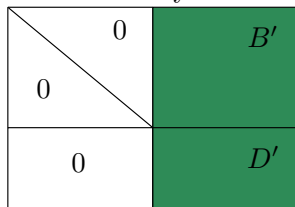
reduce A by itself



reduce C



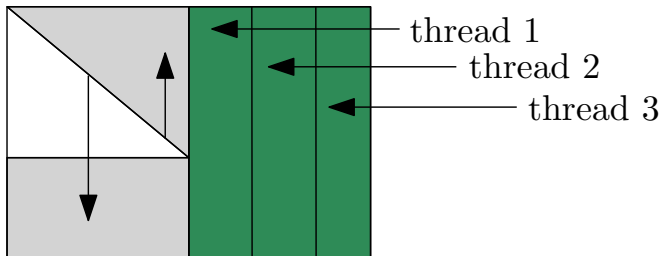
reduce D' by itself



Compute the right columns in parallel

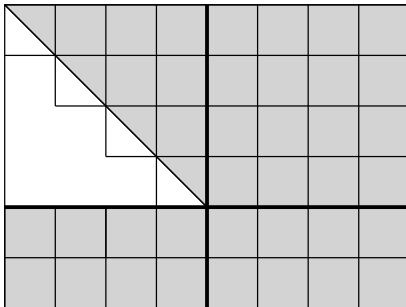
We do not need to modify the left columns.

The right column computations are independent of each other.



Subdivide the matrices into blocks

The idea is that computations on blocks fit in the CPU's cache.



Benchmark for Katsura- n mod 65521

	Katsura-11	Katsura-12	Katsura-13
Magma	19.5s	151.2s	1091.4s
F-L	2.9s	19.5s	149.6s

— Faugère-Lachartre

The F-L time is only matrix reduction — e.g. not construction.
The Magma time includes everything.

Part I Relevant background on F_4

The F_4 Algorithm

$ABCD$ decomposition

Martani's modification

Part II What I have been doing

The story of F_4 in Mathic

Matrix construction

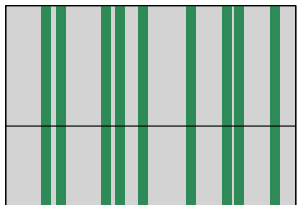
Part III The future

Part I

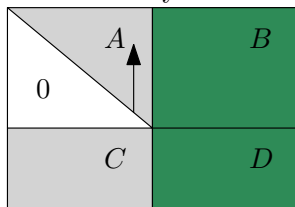
Martani's modification

Reminder: Faugère-Lachartre algorithm

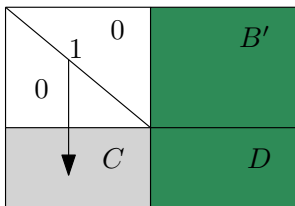
decompose into A, B, C, D



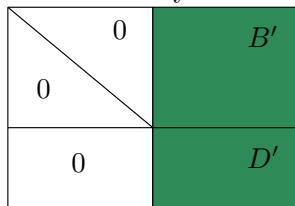
reduce A by itself



reduce C

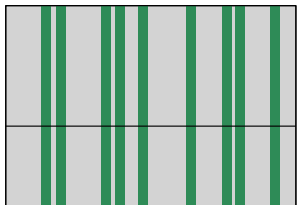


reduce D' by itself

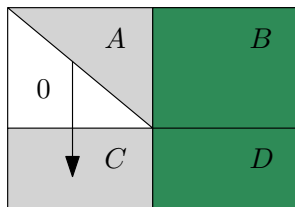


Fayssal Martani's modifications

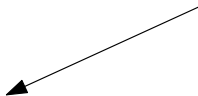
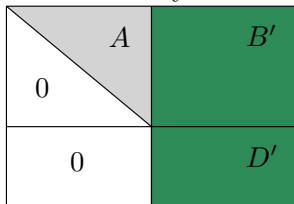
decompose into A, B, C, D



reduce C



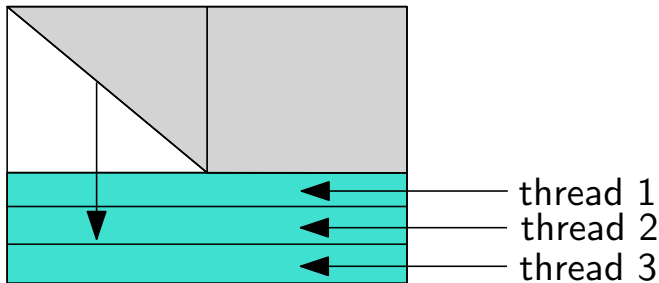
reduce D' by itself



Compute the bottom rows in parallel

We do not need to modify the **top** rows.

The **bottom row** computations are independent of each other.

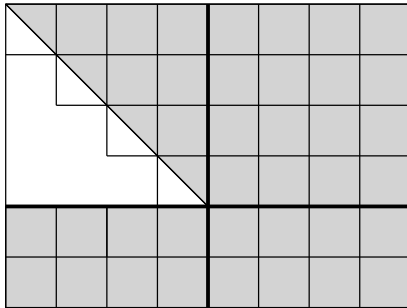


Benchmark for reducing matrices produced by F4

	Martani/s	Faugère-Lachartre/s
Katsura-13, matrix 6	12.6	31.6
Katsura-13, matrix 11	1.5	38.0
minrank-996, matrix 7	548	1354

— Martani

Martani uses: Matrix blocking



Modular arithmetic

Let $\mathbb{N}_p \stackrel{\text{def}}{=} \{0, 1, \dots, p-1\}$ and $a, b \in \mathbb{N}_p$. Then

$$a \odot b \in \mathbb{N}_p \text{ such that } a \odot b \equiv ab \pmod{p}$$

$$a \oplus b \in \mathbb{N}_p \text{ such that } a \oplus b \equiv a + b \pmod{p}$$

Modular arithmetic

Let $\mathbb{N}_p \stackrel{\text{def}}{=} \{0, 1, \dots, p-1\}$ and $a, b \in \mathbb{N}_p$. Then

$$a \odot b \in \mathbb{N}_p \text{ such that } a \odot b \equiv ab \pmod{p}$$

$$a \oplus b \in \mathbb{N}_p \text{ such that } a \oplus b \equiv a + b \pmod{p}$$

Unfortunately:

- \odot is much slower than multiplication.
- \oplus is much slower than addition.

Martani uses: Delayed modulus

Let $\alpha, \beta \in \mathbb{N}_p^n$. We want to compute $\alpha \cdot \beta \bmod p$.

Slow: $(\alpha_1 \odot \beta_1) \oplus \cdots \oplus (\alpha_n \odot \beta_n)$

Fast: $(\alpha_1\beta_1 + \cdots + \alpha_n\beta_n) \bmod p$

If $p < 2^{16}$ and $n < 2^{32}$ we can do the fast way with 64 bit integers.

Martani uses: Mixed dense/sparse vectors

Suppose two vectors have lots of zero entries.

Slow: Add two sparse format vectors (poorly predicted branch).

Martani uses: Mixed dense/sparse vectors

Suppose two vectors have lots of zero entries.

Slow: Add two sparse format vectors (poorly predicted branch).

Slow: Add two dense format vectors (time wasted on $0 + a = a$).

Martani uses: Mixed dense/sparse vectors

Suppose two vectors have lots of zero entries.

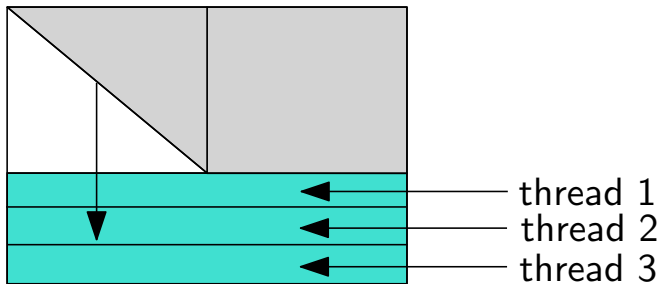
Slow: Add two sparse format vectors (poorly predicted branch).

Slow: Add two dense format vectors (time wasted on $0 + a = a$).

Fast: Add a dense and a sparse vector (if dense fits in cache).

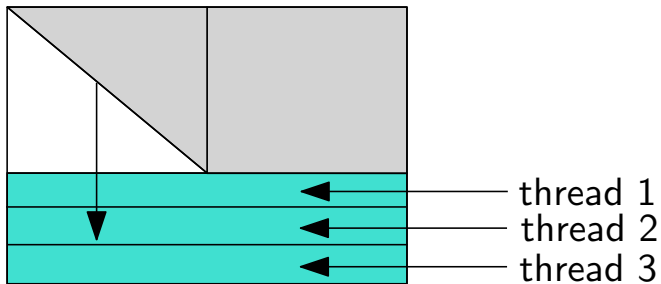
Martani's implementation

- Store top rows sparse and active **bottom rows** dense.



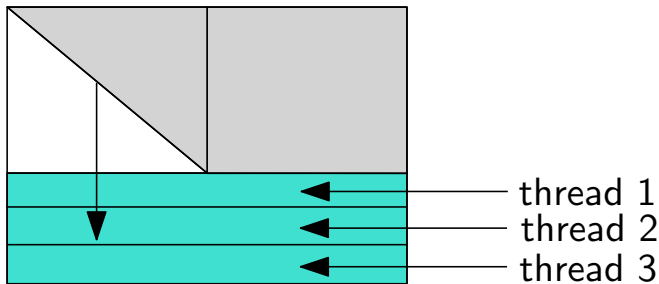
Martani's implementation

- Store top rows sparse and active bottom rows dense.
- Use delayed modulus for active bottom rows.



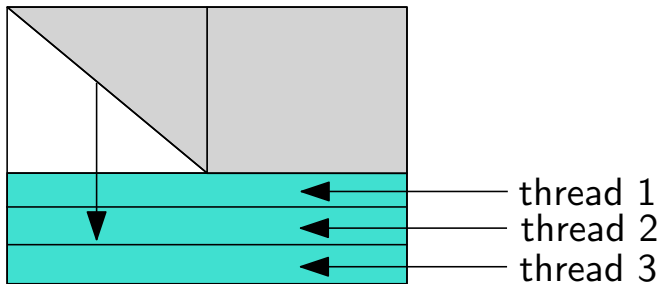
Martani's implementation

- Store top rows sparse and active bottom rows dense.
- Use delayed modulus for active bottom rows.
- Subdivide the matrix into cache-sized blocks.



Martani's implementation

- Store top rows sparse and active bottom rows dense.
- Use delayed modulus for active bottom rows.
- Subdivide the matrix into cache-sized blocks.
- Use the multiline data structure



Part I Relevant background on F_4

The F_4 Algorithm

$ABCD$ decomposition

Martani's modification

Part II What I have been doing

The story of F_4 in Mathic

Matrix construction

Part III The future

Part II

The story of F4 in Mathic

Things experts have told me about F4

*It is very difficult to implement F4 and get a good result.
I tried but never managed to get F4 to work as well as I
had hoped.*

Things experts have told me about F4

However implementing F4 usually fails in terms of performance (I wasn't successful until now). We don't think anymore, that it is an good algorithm, but it has very decent implementations. Allan Steel (magma) told us, he needed six years for his implementation.

Things experts have told me about F_4

So, you want to write a fast implementation of F_4 ? Many people have tried that and failed. What makes you think that you can do it?

So what's the secret?

A conversation I had with Faugère

Me: What is the great unpublished secret of F_4 ?

A conversation I had with Faugère

Me: What is the great unpublished secret of F_4 ?

Faugère: There is no secret.

A conversation I had with Faugère

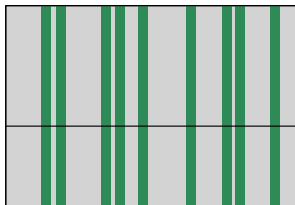
Me: What is the great unpublished secret of F_4 ?

Faugère: There is no secret.

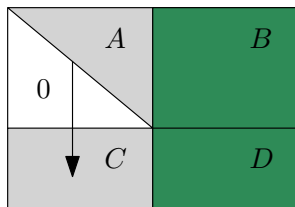
Me: Thanks.

What's another name for this?

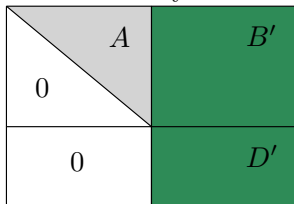
decompose into A, B, C, D



reduce C



reduce D' by itself



The great secret of F_4

The great secret of F_4

- Gaussian elimination with pivoting
- Delayed modulus
- Mixed sparse/dense vector addition

Turns out Faugère was right.

I could not believe it!

I whipped up a prototype for matrix reduction using

- Gaussian elimination with pivoting
- Delayed modulus
- Mixed sparse/dense vector addition

I could not believe it!

I whipped up a prototype for matrix reduction using

- Gaussian elimination with pivoting
- Delayed modulus
- Mixed sparse/dense vector addition

It was faster than Faugère-Lachartre's implementation!

Comparable to Martani's — slower for few cores, faster for many.

I still did not believe it

I extended my prototype to a simple F_4 implementation in Mathic.

I still did not believe it

I extended my prototype to a simple F_4 implementation in Mathic.

Benchmark on 2 cores for homogeneous cyclic-n mod 101

	threads	cyclic-8/s
old Mathic	1	11.1
old Mathic	2	8.7s
old Magma 2.17-9	1	8.9s

That was encouraging, so I continued the work.

Magma 2.17-9 is from 2011. Old Mathic is from October 2012.
Both Mathic and Magma are faster today.

Magma 2.17-9 is from 2011. Old Mathic is from October 2012.
Both Mathic and Magma are faster today.

Benchmark on 4 cores for homogeneous cyclic-n mod 101

	threads	cyclic-8/s	cyclic-9/s
Mathic	1	4.3	683
Magma 2.19-3	1	3.4	726
Singular 3-1-4	1	36.6	11,072
Mathic	8	1.4	208
Magma 2.19-3	8	2.6	531

Benchmark on 48 cores for homogeneous cyclic-n mod 101

	threads	cyclic-8/s	cyclic-9/s
Singular 3-1-5	1	70.6	21,126
Mathic	1	9.4	3,206
Mathic	8	3.6	593
Mathic	32	4.0	300

Part I Relevant background on F_4

The F_4 Algorithm

$ABCD$ decomposition

Martani's modification

Part II What I have been doing

The story of F_4 in Mathic

Matrix construction

Part III The future

Part II

Matrix construction

Where the times goes for Mathic

	cyclic-9, 1 thread	cyclic-9, 32 threads
Matrix construction	34%	36%
Matrix reduction	64%	57%
Everything else	< 1%	7%

It was hard to get matrix construction down to about 35%.

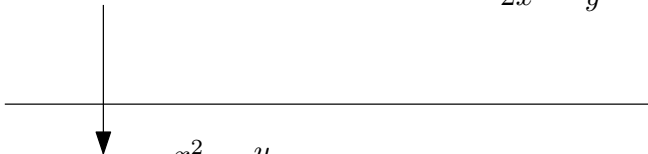
Matrix construction in Mathic

$$2x^2 - y : (x - 1, y + 2)$$

$$\begin{bmatrix} & \end{bmatrix}$$

Pending:

$$2x^2 - y$$



$$\begin{bmatrix} x^2 & y \\ 2 & -1 \end{bmatrix}$$

Pending:

$$x \cdot (x - 1)$$

$$y + 2$$

Matrix construction in Mathic

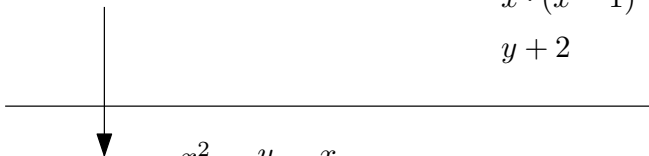
$$2x^2 - y : (x - 1, y + 2)$$

$$\begin{array}{cc} x^2 & y \\ [2 & -1] \end{array}$$

Pending:

$$x \cdot (x - 1)$$

$$y + 2$$



$$\begin{array}{ccc} x^2 & y & x \\ [2 & -1 & 0 \\ 1 & 0 & -1] \end{array}$$

Pending:

$$y + 2$$

$$x - 1$$

Matrix construction in Mathic

$$2x^2 - y : (x - 1, y + 2)$$

$$\begin{array}{ccc} x^2 & y & x \\ \left[\begin{array}{ccc} 2 & -1 & 0 \\ 1 & 0 & -1 \end{array} \right] \end{array}$$

Pending:

$$y + 2$$

$$x - 1$$



$$\begin{array}{cccc} x^2 & y & x & 1 \\ \left[\begin{array}{cccc} 2 & -1 & 0 & 0 \\ 1 & 0 & -1 & 0 \\ 0 & 1 & 0 & 2 \end{array} \right] \end{array}$$

Pending:

$$x - 1$$

Matrix construction in Mathic

$$2x^2 - y : (x - 1, y + 2)$$

$$\begin{array}{cccc} x^2 & y & x & 1 \\ \left[\begin{array}{cccc} 2 & -1 & 0 & 0 \\ 1 & 0 & -1 & 0 \\ 0 & 1 & 0 & 2 \end{array} \right] & \text{Pending:} & & \\ & x - 1 & & \end{array}$$



$$\begin{array}{cccc} x^2 & y & x & 1 \\ \left[\begin{array}{cccc} 2 & -1 & 0 & 0 \\ 1 & 0 & -1 & 0 \\ 0 & 1 & 0 & 2 \\ 0 & 0 & 1 & -1 \end{array} \right] & \text{Pending:} & & \end{array}$$

Matrix construction in Mathic

$$2x^2 - y : (x - 1, y + 2)$$

$$\begin{array}{cccc} x^2 & y & x & 1 \\ \left[\begin{array}{cccc} 2 & -1 & 0 & 0 \\ 1 & 0 & -1 & 0 \\ 0 & 1 & 0 & 2 \\ 0 & 0 & 1 & -1 \end{array} \right] \end{array}$$

Now sort column monomials and
permute rows and columns for *ABCD* decomposition.

Most time-consuming step

Given monomials a, b , what is the column index of ab ?

Most time-consuming step

Given monomials a, b , what is the column index of ab ?

or equivalently

Given $a, b \in \mathbb{N}^n$ and $f: \subseteq \mathbb{N}^n \rightarrow \mathbb{N}$, compute $f(a + b)$.

Mathic: Store f as a hash table.

Problem 1: We need a hash function $h: \mathbb{N}^n \rightarrow \mathbb{N}$.

Mathic: Store f as a hash table.

Problem 1: We need a hash function $h: \mathbb{N}^n \rightarrow \mathbb{N}$.

Allan Steel once told Michael Stillman about a neat way to map monomials to hash values, and Michael Stillman told me.

Mathic: Store f as a hash table.

Problem 1: We need a hash function $h: \mathbb{N}^n \rightarrow \mathbb{N}$.

Allan Steel once told Michael Stillman about a neat way to map monomials to hash values, and Michael Stillman told me.

Allan Steel: Use $h(v) \stackrel{\text{def}}{=} v \cdot p \pmod{2^{32}}$ for some vector p .

Mathic: Store f as a hash table.

Problem 1: We need a hash function $h: \mathbb{N}^n \rightarrow \mathbb{N}$.

Allan Steel once told Michael Stillman about a neat way to map monomials to hash values, and Michael Stillman told me.

Allan Steel: Use $h(v) \stackrel{\text{def}}{=} v \cdot p \pmod{2^{32}}$ for some vector p .

Problem 2: Now lookup is slow because computing $h(v)$ is slow.

Mathic: Store f as a hash table.

Problem 1: We need a hash function $h: \mathbb{N}^n \rightarrow \mathbb{N}$.

Allan Steel once told Michael Stillman about a neat way to map monomials to hash values, and Michael Stillman told me.

Allan Steel: Use $h(v) \stackrel{\text{def}}{=} v \cdot p \bmod 2^{32}$ for some vector p .

Problem 2: Now lookup is slow because computing $h(v)$ is slow.

Allan Steel: Store $h(v)$ with v .

Mathic: Store f as a hash table.

Problem 1: We need a hash function $h: \mathbb{N}^n \rightarrow \mathbb{N}$.

Allan Steel once told Michael Stillman about a neat way to map monomials to hash values, and Michael Stillman told me.

Allan Steel: Use $h(v) \stackrel{\text{def}}{=} v \cdot p \bmod 2^{32}$ for some vector p .

Problem 2: Now lookup is slow because computing $h(v)$ is slow.

Allan Steel: Store $h(v)$ with v .

Problem 3: Now $a + b$ is slow as we need to compute $h(a + b)$.

Mathic: Store f as a hash table.

Problem 1: We need a hash function $h: \mathbb{N}^n \rightarrow \mathbb{N}$.

Allan Steel once told Michael Stillman about a neat way to map monomials to hash values, and Michael Stillman told me.

Allan Steel: Use $h(v) \stackrel{\text{def}}{=} v \cdot p \bmod 2^{32}$ for some vector p .

Problem 2: Now lookup is slow because computing $h(v)$ is slow.

Allan Steel: Store $h(v)$ with v .

Problem 3: Now $a + b$ is slow as we need to compute $h(a + b)$.

Allan Steel: Not so, because $h(a + b) = h(a) + h(b)$.

$f(a + b)$ still most time-consuming step

A way to compute $f(a + b)$

- 1 Compute $c \leftarrow a + b$ and $h' \leftarrow h(a) + h(b)$.
- 2 For each entry s of f with $h(s) = h'$, check if $s = c$.

$f(a + b)$ still most time-consuming step

A way to compute $f(a + b)$

- 1 Compute $c \leftarrow a + b$ and $h' \leftarrow h(a) + h(b)$.
- 2 For each entry s of f with $h(s) = h'$, check if $s = c$.

A faster way to compute $f(a + b)$

- 1 Compute $h' \leftarrow h(a) + h(b)$
- 2 For each entry s of f with $h(s) = h'$, check if $s = a + b$.

$f(a + b)$ still most time-consuming step

A way to compute $f(a + b)$

- 1 Compute $c \leftarrow a + b$ and $h' \leftarrow h(a) + h(b)$.
- 2 For each entry s of f with $h(s) = h'$, check if $s = c$.

A faster way to compute $f(a + b)$

- 1 Compute $h' \leftarrow h(a) + h(b)$
- 2 For each entry s of f with $h(s) = h'$, check if $s = a + b$.

Isn't that just the same thing?

It's not the same thing!

Don't do this to check if $s = a + b$

- 1 for $i = 1, \dots, n$: $c_i \leftarrow a_i + b_i$.
- 2 for $i = 1, \dots, n$: if $s_i \neq c_i$, return false

It's not the same thing!

Don't do this to check if $s = a + b$

- 1 for $i = 1, \dots, n$: $c_i \leftarrow a_i + b_i$.
- 2 for $i = 1, \dots, n$: if $s_i \neq c_i$, return false

OK (c is never written to memory)

- 1 for $i = 1, \dots, n$: if $s_i \neq a_i + b_i$, return false

It's not the same thing!

Don't do this to check if $s = a + b$

- 1 for $i = 1, \dots, n$: $c_i \leftarrow a_i + b_i$.
- 2 for $i = 1, \dots, n$: if $s_i \neq c_i$, return false

OK (c is never written to memory)

- 1 for $i = 1, \dots, n$: if $s_i \neq a_i + b_i$, return false

Good (fewer if's, unrollable, optimized for true-answer)

- 1 $d \leftarrow 0$
- 2 for $i = 1, \dots, n$: $d \leftarrow d \mid (s_i \wedge (a_i + b_i))$.
- 3 If $d \neq 0$, return false

It's not the same thing!

Don't do this to check if $s = a + b$

- 1 for $i = 1, \dots, n$: $c_i \leftarrow a_i + b_i$.
- 2 for $i = 1, \dots, n$: if $s_i \neq c_i$, return false

OK (c is never written to memory)

- 1 for $i = 1, \dots, n$: if $s_i \neq a_i + b_i$, return false

Good (fewer if's, unrollable, optimized for true-answer)

- 1 $d \leftarrow 0$
- 2 for $i = 1, \dots, n$: $d \leftarrow d \mid (s_i \wedge (a_i + b_i))$.
- 3 If $d \neq 0$, return false

Better: 64 bit integers or SSE can check several entries per step.

It's not the same thing!

Don't do this to check if $s = a + b$

- 1 for $i = 1, \dots, n$: $c_i \leftarrow a_i + b_i$.
- 2 for $i = 1, \dots, n$: if $s_i \neq c_i$, return false

OK (c is never written to memory)

- 1 for $i = 1, \dots, n$: if $s_i \neq a_i + b_i$, return false

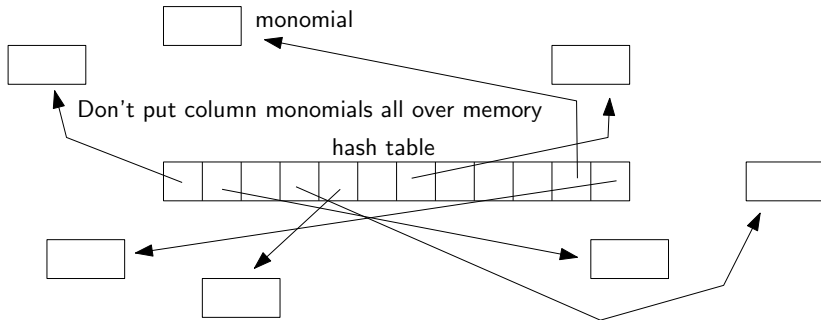
Good (fewer if's, unrollable, optimized for true-answer)

- 1 $d \leftarrow 0$
- 2 for $i = 1, \dots, n$: $d \leftarrow d \mid (s_i \wedge (a_i + b_i))$.
- 3 If $d \neq 0$, return false

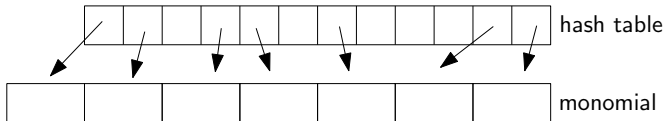
Better: 64 bit integers or SSE can check several entries per step.

Best: Also pack small exponents into fewer bits (for future).

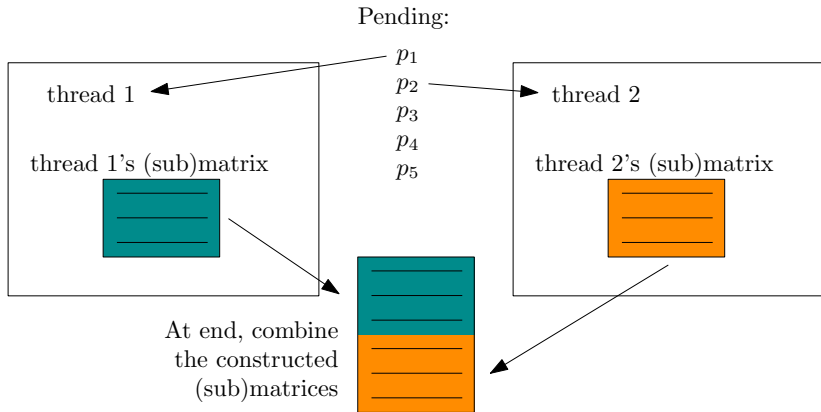
Make better use of the cache



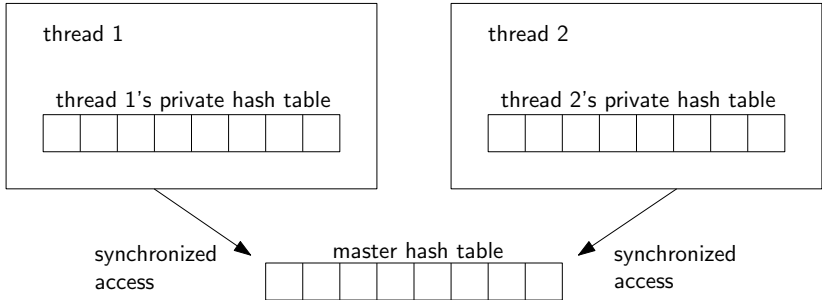
Put column monomials together for better cache performance



Parallel matrix construction in Mathic

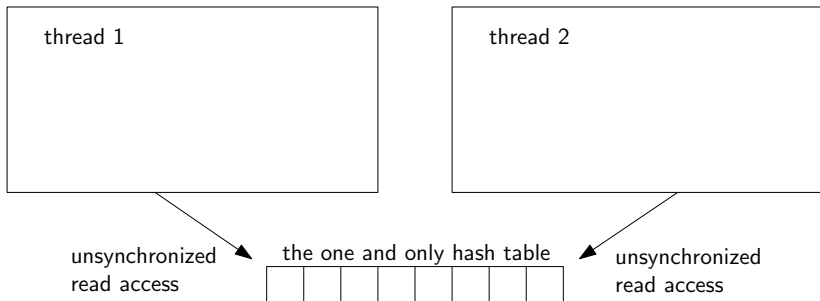


First attempt at parallel hash table scheme



Current parallel hash table scheme

Use C++11's `std::atomic<T>` for unsynchronized read access.
Lookups are no slower than for serial code on x86 and x64.



Part I Relevant background on F_4

The F_4 Algorithm

$ABCD$ decomposition

Martani's modification

Part II What I have been doing

The story of F_4 in Mathic

Matrix construction

Part III The future

Part III

The future

A few near-future improvements for Mathic

Use SSE for matrix construction and reduction

SSE instructions do 4 things simultaneously. Up to 4x the speed.

A few near-future improvements for Mathic

Use SSE for matrix construction and reduction

SSE instructions do 4 things simultaneously. Up to 4x the speed.

Subdivide the matrix into blocks

Standard technique. Should be a significant improvement.

A few near-future improvements for Mathic

Use SSE for matrix construction and reduction

SSE instructions do 4 things simultaneously. Up to 4x the speed.

Subdivide the matrix into blocks

Standard technique. Should be a significant improvement.

Pack small exponents into fewer bits

Standard technique. Should speed up matrix construction.

A few near-future improvements for Mathic

Use SSE for matrix construction and reduction

SSE instructions do 4 things simultaneously. Up to 4x the speed.

Subdivide the matrix into blocks

Standard technique. Should be a significant improvement.

Pack small exponents into fewer bits

Standard technique. Should speed up matrix construction.

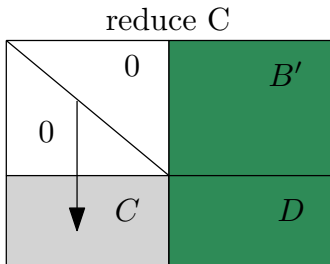
Do not double memory use while preparing matrix

Permuting rows and columns currently copies for double memory.

GPU computation

GPU's are great at linear algebra. Often 10-100x CPU speed.

Reducing C here is equivalent to $D \leftarrow D + CB'$.



The problem of choice

Gröbner basis performance depends on a great many choices:

- Which algorithm (classic, F4, F5, slimgb, ...)
- Which implementation (Magma, Mathic, Singular, 4ti2, ...)
- Which monomial order (grevlex, lex, glex, block, ...)
- What S-pair order (sugar, F5-like, normal, sparse-first, ...)
- What choice of reducers (oldest, sparsest)
- ...

There is no always-best set of choices.

Make the wrong choices and you'll be waiting forever.

The problem of choice

Gröbner basis performance depends on a great many choices:

- Which algorithm (classic, F4, F5, slimgb, ...)
- Which implementation (Magma, Mathic, Singular, 4ti2, ...)
- Which monomial order (grevlex, lex, glex, block, ...)
- What S-pair order (sugar, F5-like, normal, sparse-first, ...)
- What choice of reducers (oldest, sparsest)
- ...

There is no always-best set of choices.

Make the wrong choices and you'll be waiting forever.

Common practice for fitting choices to input

The user does it. Usually, that means it does not happen.

It's the same thing for Satisfiability (SAT)

Old practice for SAT

Experts come up with and implement heuristics.

It's the same thing for Satisfiability (SAT)

Old practice for SAT

Experts come up with and implement heuristics.

Then SATzilla happened. It won competitions by using a better idea. The paper has 198 citations since 2008.

It's the same thing for Satisfiability (SAT)

Old practice for SAT

Experts come up with and implement heuristics.

Then SATzilla happened. It won competitions by using a better idea. The paper has 198 citations since 2008.

The better practice for SAT

Machine learning algorithms make the choices.

Let's try that for Gröbner bases.

Mathic is open source

`https://github.com/broune/mathicgb`

The end

Part I Relevant background on F_4

The F_4 Algorithm

$ABCD$ decomposition

Martani's modification

Part II What I have been doing

The story of F_4 in Mathic

Matrix construction

Part III The future