CSE291-14: The Number Field Sieve

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

Emmanuel Thomé

January 4, 2022

Part 1a

Introduction

Problem statement and motivations

Hardness

Algorithms: theory and practice

Course trivia

Plan

Problem statement and motivations

Hardness

Algorithms: theory and practice

Course trivia

The challenge

Integer factorization problem (IF)

Let $N = pq$ be a product of 2 primes with $\log p \approx \log q$. Factor N.
The hardness of IF increases with the size of N.

Discrete logarithm problem (DLP) in a group G

G is assumed to be cyclic. Given $a = g^x$ in G, find x.
The DLP hardness depends on the group G.

(If not absolutely comfortable with DLP, we'll talk about it in greater detail anyway.)

Why do we want to look at these problems?

Several motivating arguments.

- Cryptography and cryptanalysis.
  - IF is the hard problem that RSA relies on.
  - DLP is the hard problem that Diffie-Hellman relies on.
- Computational mathematics.
  - Many number-theoretic calculations use IF as a subroutine.
- Recreation.

RSA encryption

- N integer, and p, q (prime) such that $N = pq$.
- $\phi(N) = (p-1)(q-1) = \#(\mathbb{Z}/N\mathbb{Z})^*$.
- e coprime to $\phi(N)$, therefore invertible modulo $\phi(N)$.
- d computed such that $de \equiv 1 \mod \phi(N)$. $\rightarrow$ the trapdoor!

Alice		Bob
public key: (N, e)		private key: (p, q, d)
plaintext: $m \mod N$		
$x = m^e \bmod N$	$\longrightarrow$	$m = x^d \bmod N$

$$x^d \equiv m^{ed} \equiv m^{1+k\phi(N)} \equiv m \mod N.$$

RSA assumption

Bob keeps p, q secret, and publishes N.

$\Rightarrow$ Bob knows p, q, and can compute $\phi(N) = (p-1)(q-1)$.

$\Rightarrow$ He can therefore compute d.

$\Rightarrow$ and recover the m from the ciphertext $x = m^e \mod N$.

If an attacker can factor N, they can do the same.

Note: we have implications above.

The RSA assumption

The RSA assumption is: it is hard to recover m from x.

Of course it gets harder as N grows.

No one knows if hardness of IF and the RSA assumption are equivalent. (The common wisdom is that they are probably not.)

Diffie-Hellman

Very important primitive: key exchange.

Public data: a finite cyclic group $\boxed{G = \langle g \rangle}$.

Alice		Bob
$\boxed{k_A}$ random		$\boxed{k_B}$ random
$x_A = g^{k_A}$	$\xrightarrow{\boxed{x_A}}$	$x_B = g^{k_B}$
$\boxed{y} = x_B^{k_A}$	$\xleftarrow{\boxed{x_B}}$	$\boxed{y} = x_A^{k_B}$

Alice and Bob can then use y as a symmetric encryption key for their communication (e.g. with AES).

DH assumption

The goal of the attacker is to do something hard that neither Alice nor Bob has to do:

Knowing $\boxed{x_A = g^{k_A}}$ and $\boxed{x_B = g^{k_B}}$, find $\boxed{y} = g^{k_A k_B}$.

As before, if an attacker can solve DLP, he can do this.

The DH assumption

The DH assumption is that this is hard to find $g^{k_A k_B}$.
Of course it depends on the group G.

No one knows if DH and hardness of DLP are equivalent.
There are results suggesting that they might be.

Mathematical motivation

In some mathematical contexts, the prime factors of some integers hide some interesting structure.

Example: in a number field K, the only way to be sure that we correctly compute the ring of integers $\mathcal{O}_K$ is by factoring the discriminant of the defining polynomial of K.

Recreational aspect

Because IF is so easy to state, it has been looked at for a long time, and also attracts many enthusiasts.

- The Cunningham project has been running for decades.
  - Goal: collect factorization of numbers of the form $b^n \pm 1$.
  - Mathematical motivation does exist, but is quite dim.
- There is a sizable community of factoring enthusiasts (and big prime hunters).

A constructive cryptographical motivation

Discrete logarithm is a way to make a map explicit.

- Mathematician viewpoint: a cyclic group G is like $\mathbb{Z}/N\mathbb{Z}$. That's not interesting.

$$G \cong \mathbb{Z}/N\mathbb{Z}.$$

- However, this equivalence is precisely what DLP is all about!

Example of CSIDH-512:

- efficient uniform random sampling in a cyclic group via discrete logarithm computations.
- a DLP computation has helped make a cryptosystem proposal more efficient.

Plan

Problem statement and motivations

Hardness

Algorithms: theory and practice

Course trivia

Case-sensitiveness

A word of caution

As we talk about integer factorization, there's often an integer N that is floating around.
The bit size of the input N is of course $n = \log_2 N$.

Qualitatively, all complexities are understood as functions of n: an exponential algorithm is one that is exponential in n.

- A $O(\sqrt{N})$ algorithm is exponential in n.
- A $O(N^{0.001})$ algorithm is also exponential.
- A $O(\exp(\sqrt{n}))$ algorithm is sub-exponential.

Upper bounds on asymptotic complexity

Complexity (spoiler alert)

The Number Field Sieve (this class!) is the algorithm which solves these problems with the best asymptotic complexity:

- For an n-bit input, typically: $\exp\left(cn^{1/3}(\log n)^{2/3} \cdot (1 + o(1))\right)$ with an explicit constant c.
- This is sub-exponential.
- $(1 + o(1))$ hides a lot.
- What this means in concrete terms is hard to say.

We'll discuss the complexity in more detail as we review the different algorithms later on.

Hardness: converse problems

Note that of course, the converse problems are easy.

This situation is good for cryptography, of course.

The converse of DLP is ...

- ...
- efficiently done with ...
- which costs ... group operations for an n-bit input.

The converse of IF is ...

- ...
- efficiently done with ...
  - in practice: ...
  - in theory: ...

Complexity-wise, IF and DLP are in NP, but that doesn't say much.

Hardness: converse problems

Note that of course, the converse problems are easy.

This situation is good for cryptography, of course.

The converse of DLP is ...

- exponentiation in a group,
- efficiently done with ...
- which costs ... group operations for an n-bit input.

The converse of IF is ...

- ...
- efficiently done with ...
  - in practice: ...
  - in theory: ...

Complexity-wise, IF and DLP are in NP, but that doesn't say much.

Hardness: converse problems

Note that of course, the converse problems are easy.

This situation is good for cryptography, of course.

The converse of DLP is ...

- exponentiation in a group,
- efficiently done with square and multiply,
- which costs ... group operations for an n-bit input.

The converse of IF is ...

- ...
- efficiently done with ...
  - in practice: ...
  - in theory: ...

Complexity-wise, IF and DLP are in NP, but that doesn't say much.

Hardness: converse problems

Note that of course, the converse problems are easy.

This situation is good for cryptography, of course.

The converse of DLP is ...

- exponentiation in a group,
- efficiently done with square and multiply,
- which costs $O(n)$ group operations for an n-bit input.

The converse of IF is ...

- ...
- efficiently done with ...
  - in practice: ...
  - in theory: ...

Complexity-wise, IF and DLP are in NP, but that doesn't say much.

Hardness: converse problems

Note that of course, the converse problems are easy.

This situation is good for cryptography, of course.

The converse of DLP is ...

- exponentiation in a group,
- efficiently done with square and multiply,
- which costs $O(n)$ group operations for an n-bit input.

The converse of IF is ...

- integer multiplication,
- efficiently done with ...
  - in practice: ...
  - in theory: ...

Complexity-wise, IF and DLP are in NP, but that doesn't say much.

Hardness: converse problems

Note that of course, the converse problems are easy.

This situation is good for cryptography, of course.

The converse of DLP is ...

- exponentiation in a group,
- efficiently done with square and multiply,
- which costs $O(n)$ group operations for an n-bit input.

The converse of IF is ...

- integer multiplication,
- efficiently done with FFT-based methods.
  - in practice: ...
  - in theory: ...

Complexity-wise, IF and DLP are in NP, but that doesn't say much.

Hardness: converse problems

Note that of course, the converse problems are easy.

This situation is good for cryptography, of course.

The converse of DLP is ...

- exponentiation in a group,
- efficiently done with square and multiply,
- which costs $O(n)$ group operations for an n-bit input.

The converse of IF is ...

- integer multiplication,
- efficiently done with FFT-based methods.
  - in practice: Schönhage-Strassen (among others);
  - in theory: ...

Complexity-wise, IF and DLP are in NP, but that doesn't say much.

Hardness: converse problems

Note that of course, the converse problems are easy.

This situation is good for cryptography, of course.

The converse of DLP is ...

- exponentiation in a group,
- efficiently done with square and multiply,
- which costs $O(n)$ group operations for an n-bit input.

The converse of IF is ...

- integer multiplication,
- efficiently done with FFT-based methods.
  - in practice: Schönhage-Strassen (among others);
  - in theory: Harvey-van der Hoeven, $O(n \log n)$.

Complexity-wise, IF and DLP are in NP, but that doesn't say much.

Lower bounds on complexity

I am not aware of any non-trivial lower bound on:

- IF
- or any concrete instance of DLP.

Only result in the black box model

The only result is Nechaev-Shoup's: if you know nothing about a group G, then DLP in G costs $\Omega(\sqrt{\#G})$.

it is also $O(\sqrt{\#G})$, so it's $\Theta()$.

Heuristics and Probabilistic algorithms

This slide is some sort of a disclaimer.

As far as the cryptanalysis motivation goes, we ultimately don't mind if we achieve a security breach "heuristically", or "probabilistically". If it's broken, it's broken.

NFS does many wild heuristics, and is also a probabilistic algorithm. The GRH is only one of the many heuristics. This won't disturb us.

The search for proven algorithms for factoring or discrete logarithms is a very different topic. At this point, proven deterministic algorithms have exponential complexity.

Plan

Problem statement and motivations

Hardness

Algorithms: theory and practice

Course trivia

A zoo of algorithms

Part of this course will briefly review the pre-NFS algorithms, in particular because NFS did build on something!

It is also interesting from a historical perspective.

Some recursion is going on: NFS uses factorizations of moderate-size numbers as a subroutine.

Less advanced algorithms are better for these moderate sizes, and thus can be used within NFS!

What makes NFS stand out

Most of the pre-NFS algorithms needed mostly arithmetic with:

- finite fields $\mathbb{F}_p$,
- polynomials over $\mathbb{Z}$ or over $\mathbb{F}_p$,
- linear algebra.

In a way, it's the same with NFS. However NFS is inscribed in a broader mathematical context which can be frightening:

- number fields and algebraic number theory,
- a few other things touched upon in passing.

The cryptanalytic motivation has fueled the constant interest of people over several years/decades. NFS folklore is rich of multiple things.

Computational aspect

This course intends to present the NFS algorithm AND address its more practical aspects.

- What does it take to program NFS?
- What are the existing implementations and their shortcomings?
- How hard are the different steps of the algorithm?
- What implementation choices are important?

Required background

NFS touches upon many aspects, but very often on a fairly elementary level.

- Algebraic number theory.
  But one does not need to master ANT to understand NFS!
- Computer algebra and computer arithmetic.
  Mostly finite fields, polynomials.
- Complexity and analysis of algorithms.
- Analytic number theory.
  Prime number theorem is mostly enough.
- Basic principles of computer architecture.
- Programming.

Relevant literature

The course web page lists several resources, books, and articles. The immense majority is available online.

A few textbooks have spot-on chapters that deal with NFS, namely:

- V. Shoup, A Computational Introduction to Number Theory and Algebra.
- S. Galbraith, Mathematics of Public Key Cryptography.

Both are available online (links on course web page).

Early history and high-level view of NFS are best documented in

- A.K. Lenstra, H.W. Lenstra, Jr, The Development of the Number Field Sieve.

Plan

Problem statement and motivations

Hardness

Algorithms: theory and practice

Course trivia

About me

PhD in France some decades ago. Spent a year in Chicago as a visitor while I was beginning my PhD (previous millenium).

Full-time researcher at Inria since 2003, where I've been leading a research group since 2015. (group = 8 permanent full-time researchers + 2 permanent faculty. Small number of students).

Contributed several high-level things about NFS-like algorithms. Also contributed several low-level things and lots of code. Main author of Cado-NFS. Participated in many computational records.

Fulbright grant. Visiting professor in the CSE department this year. Enjoying San Diego!

Participation in records

I participated in most large NFS computations since 2010.

- 2010: Factorization of RSA-768
- 2014: Discrete logarithms in $\mathbb{F}_{2^{809}}$.
- 2015: Logjam attack, real-time 512-bit discrete logarithm computations and MitM exploit.
- 2016: Discrete logarithms modulo "special" 1024-bit p.
- 2019: Discrete logarithms modulo 795-bit p.
- 2019: Factorization of 795-bit RSA-240.
- 2020: Factorization of 829-bit RSA-250.

The Cado-NFS software

Cado-NFS is a full implementation of the Number Field Sieve, started in 2007. Cado-NFS is

- standalone,
- LGPL-licensed,
- open source and open development (open git since 2009),

  https://cado-nfs.inria.fr/

- mostly C and C++, hundreds of kLoc.

Contributors: 3 (core) + random variable in $[0, 5]$.

Cado-NFS was used in all my recent record computations.

Course organization

I'll try to put in perspective the theory aspects and the practical ones.

By "practical", I mean:

- Things that are definitely important in an implementation
- Things that are potentially useful in an implementation
- Considerations of how a typical implementation performs.

The main computer illustration platform will be Cado-NFS. It's probably wise to install it right away, if not done already.

Cado-NFS runs on linux.

pick the git master branch, please. It does break every now and then, please report!

Grading

Grading is primarily based on course participation.

If you want to maximize credit for this course, I expect that you turn in some work.

- No formal homework assignments, but some open questions or suggestions for independent investigation.
- Or you can pick a claim that appears in the courses, and work to document how true (or wrong!) it is. This can (but does not have to) entail experimental work.

CSE291-14: The Number Field Sieve

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

Emmanuel Thomé

January 4, 2022

Part 1b

Prime numbers and basic sieving

Primes and the prime counting function

Compositeness testing

Finding primes with sieving

Goals for today

Factoring is about splitting into primes.

Today: focus on primes.

Some techniques that already apparent in this context are also relevant for NFS.

Smooth numbers

Definition: smooth numbers

An integer N is B-smooth if it has all its prime factors are $\leq B$.
Example: $1152 = 2^7 \times 3^2$ is 3-smooth.

In NFS, most of the time ($> 50\%$) is spent in testing if certain numbers are smooth. This is very close to factoring.

Plan

Primes and the prime counting function

Compositeness testing

Finding primes with sieving

Primes !

There are infinitely many primes. Euclide knew that.

A much harder result:

Theorem (Prime number theorem, 1896)

$\pi(x) = \#\{p \text{ prime}, p < x\} \sim \dfrac{x}{\log x}$.

$\pi(x)$ *is known as the* *prime counting function*.

$\pi(x)$ is however not explicitly known.

Handwavy statement: a random integer around x has a probability $1/\log x$ of being a prime.

Handwavy statements like this are ok, but if anything more precise is wanted, things become hard pretty quickly.

$\pi(x)$ and $\mathrm{li}(x)$

Better approximations of $\pi(x)$ are known. The generalized Riemann hypothesis (GRH) implies (and is equivalent to):

$$\pi(x) - \mathrm{li}(x) = O(\sqrt{x}\log x)$$

where $\mathrm{li}(x)$ is the logarithmic integral function $\int_0^x \mathrm{d}t/\log t$.

In practice, we're ok with assuming GRH, and $\mathrm{li}(x)$ is quite good: relative error $< 10^{-5}$ for $x \approx 2^{32}$.

Note that $\mathrm{li}(x)$ is (almost) among the standard math functions in C++17:

```
double nprimes_interval(double p0, double p1)
{
    using namespace std;
    return expint(log(p1)) - expint(log(p0));
}
```

Dividing a prime range into pieces

We commonly have tasks to do like:

For all primes in $[a, b]$, do something.

In a multi-core, very parallel world, we want to divide this into pieces with roughly equal work. This is made easy with access to a decent approximation of $\pi(x)$.

The isPrime problem

Algorithmic problem: isPrime. Tell whether $x \in \mathbb{N}$ is prime or not.

Proved to be solvable in deterministic polynomial time only in 2002 (Agrawal, Kayal and Saxena).

Here: not only we don't care about deterministic, but we don't need a certified answer either.

Consequence: Stick to last century algorithms for compositeness testing.

Plan

Primes and the prime counting function

Compositeness testing

Finding primes with sieving

Compositeness testing

General idea

Start from a statement that says:

If N is prime then for all $x \mod N$, something(x) holds.

If we find an x such that something(x) doesn't hold, we have a proof that N is composite.

Testing something(x) is generally quick.

It is hopeless to try to prove primality with such statements.

People try to give bounds on the number of false negatives among witnesses x.

Fermat

If N is prime then it verifies Fermat's little Theorem:
for all x coprime to N, $x^{N-1} \equiv 1 \bmod N$

False negatives: $2^{340} \equiv 1 \bmod 341$.
Def. We say that 341 is a Fermat pseudoprime in base 2.

Even worse, this can happen for any base:

Def. A Carmichael number is a composite that is a Fermat pseudoprime in any base.

First examples: 561, 1105, 1729, 2465, 2821, 6601, 8911, 10585.

See A002997.

Thm. (Alford, Granville, Pomerance, 1994; later: Harman, 2008)
There are at least $n^{1/3}$ Carmichael numbers between 1 and n.
(the true count might be as large as $n^{1-o(1)}$).

Solovay–Strassen

If N is prime then $\left(\frac{x}{N}\right) = x^{(N-1)/2} \bmod N$.
note: The Jacobi symbol $\left(\frac{x}{N}\right)$ can be computed with the reciprocity law, which is close to the Euclidean algorithm.

Bound on false negatives: For a fixed composite N, the probability that N is a pseudoprime in base x is less than $1/2$.

Rem. This is very pessimistic.

Miller-Rabin

Idea. Refine previous test, using properties of the 2-Sylow subgroup of the multiplicative group of $\mathbb{Z}/N\mathbb{Z}$.
Assume that N is an odd prime; $N - 1 = 2^s d$, where d is odd.
For any x coprime to N, one of the following holds:

- $x^d \equiv 1 \mod N$;
- $x^{d.2^r} \equiv -1 \mod N$, for an $0 \leq r \leq s - 1$.

Proof. If N is an odd prime, there are 2 square roots of 1 modulo N, namely -1 and 1. Otherwise, there are more.

Thm. For any fixed composite N, the probability that N is a strong pseudoprime in base x is less than $1/4$.

Rem. Again, very pessimistic. Testing with $x = 2$ and $x = 3$ proves primality of N's up to $1,373,653$.

isPrime works

The bottom line is that testing a number for probable primality can be done:

- in a way that is satisfactory for "most purposes" (at the very least for all our potential uses within NFS),
- and in polynomial time, with minimal storage overhead.

There are several more advanced algorithms in this realm, some of which actually prove primality and give certificates. It is not useful to explore these in our usage context.

Plan

Primes and the prime counting function

Compositeness testing

Finding primes with sieving

Eratosthenes' sieve

First goal: find all primes up to a certain bound B.

Memory requirement. Array of B bits with random access.

~~0~~	~~1~~	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24

- Initialize the array with 1's everywhere. Let $P = 2$.
- While P is less than $\sqrt{B}$, do:

Eratosthenes' sieve

First goal: find all primes up to a certain bound B.

Memory requirement. Array of B bits with random access.

~~0~~	~~1~~	2	3	~~4~~	5	~~6~~	7	~~8~~	9	~~10~~	11	~~12~~	13	~~14~~	15	~~16~~	17	~~18~~	19	~~20~~	21	~~22~~	23	~~24~~

- Initialize the array with 1's everywhere. Let $P = 2$.
- While P is less than $\sqrt{B}$, do:
  - Zero out all array values at positions that are non-trivial multiples of P.

Eratosthenes' sieve

First goal: find all primes up to a certain bound B.

Memory requirement. Array of B bits with random access.

~~0~~	~~1~~	2	3	~~4~~	5	~~6~~	7	~~8~~	9	~~10~~	11	~~12~~	13	~~14~~	15	~~16~~	17	~~18~~	19	~~20~~	21	~~22~~	23	~~24~~

- Initialize the array with 1's everywhere. Let $P = 2$.
- While P is less than $\sqrt{B}$, do:
  - Zero out all array values at positions that are non-trivial multiples of P.
  - Advance P until the array value at P is 1.

Eratosthenes' sieve

First goal: find all primes up to a certain bound B.

Memory requirement. Array of B bits with random access.

~~0~~	~~1~~	2	3	~~4~~	5	~~6~~	7	~~8~~	~~9~~	~~10~~	11	~~12~~	13	~~14~~	~~15~~	~~16~~	17	~~18~~	19	~~20~~	~~21~~	~~22~~	23	~~24~~

- Initialize the array with 1's everywhere. Let $P = 2$.
- While P is less than $\sqrt{B}$, do:
  - Zero out all array values at positions that are non-trivial multiples of P.
  - Advance P until the array value at P is 1.

Eratosthenes' sieve

First goal: find all primes up to a certain bound B.

Memory requirement. Array of B bits with random access.

~~0~~	~~1~~	2	3	~~4~~	5	~~6~~	7	~~8~~	~~9~~	~~10~~	11	~~12~~	13	~~14~~	~~15~~	~~16~~	17	~~18~~	19	~~20~~	~~21~~	~~22~~	23	~~24~~

- Initialize the array with 1's everywhere. Let $P = 2$.
- While P is less than $\sqrt{B}$, do:
  - Zero out all array values at positions that are non-trivial multiples of P.
  - Advance P until the array value at P is 1.

Eratosthenes' sieve

First goal: find all primes up to a certain bound B.

Memory requirement. Array of B bits with random access.

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24

- Initialize the array with 1's everywhere. Let $P = 2$.
- While P is less than $\sqrt{B}$, do:
  - Zero out all array values at positions that are non-trivial multiples of P.
  - Advance P until the array value at P is 1.
- Return the indices (≥ 2) of the 1's still in the array.

Eratosthenes' sieve

Time complexity.

The integer P in the loop takes all prime values up to $\sqrt{B}$.

For each P, we visit $\lfloor B/P \rfloor$ positions.

So the total number of operations is $\sum_{P<\sqrt{B} \text{ prime}} \lfloor B/P \rfloor$, which is essentially

$$B \sum_{P<\sqrt{B} \text{ prime}} \frac{1}{P}$$

By Mertens' theorem, this gives a cost of $O(B \log \log B)$ operations.

Caveat

The cost above is given in arithmetic operations (+ memory accesses). Arithmetic on integers below B has bit complexity $O(\log B)$ at least.

Wheel sieving

Starting idea. Most of the primes are odd. Sounds stupid to consider an even integer if we look for a prime (!)

This easily saves a factor of 2.

The Wheel sieve generalizes this. with all small primes up to k.

Let $M = 2 \cdot 3 \cdot 5 \cdot 7 \cdot 11 \cdots k$, and prepare an array of size M where the position i contains 1 iff i is coprime to M, and 0 otherwise.

Use it as a repeated mask to quickly kill all positions of the array that are divisible by a prime smaller than k.

Taking $k \approx \sqrt{B}$ yields

Thm. (Pritchard) All primes up to B can be computed in $O(B/\log\log B)$ operations.

Segmented sieve

It is not always necessary to allocate the full array beforehand.

- Simple sieves work well in a segmented way, requiring only $O(\sqrt{B})$ storage at a given time, with no significant penalty.
- Harder to do with more advanced version of the wheel sieve.

An interesting property is also random access.

- It is interesting to look for primes in $[a, b]$ and pay only $O(\sqrt{b})$ initialization cost.
- This is particularly useful when parallelizing.

(Cado-NFS: 4d61b4182, 12186c0ab; see also primegen)

CSE291-14: The Number Field Sieve

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

Emmanuel Thomé

January 6, 2022

Part 1c

Old factoring algorithms

Factoring with simple sieving

The product tree approach

Recap from last time

- We know how to test primality.
- We learned how to find primes in an interval with sieving.
- When we do so, the size of the sieve array matters.
- We may use a segmented version of the sieve of Eratosthenes in order to alleviate the memory concerns. With segmentation, random access is possible: we can search for primes in a range $[a, b]$ without enumerating all primes up to a. Enumeration up to $\sqrt{b}$ is enough.

Plan

Factoring with simple sieving

The product tree approach

More information with sieving

Instead of storing a zero in the array, one can keep further information.

Depending on the information stored, one can get more or less data on the factorization of the integers, at a cost of higher memory:

- Initialize with zero, and add one at each sieving step. Gives the number of distinct prime factors. Requires $B \log_2 \log_2 B$ bits of memory.
- With only 2 bits per position, one can get the numbers that contain exactly two distinct primes.

Sieving for factoring integers

Can we recover the full factorization of integers with sieving?

The exact goal needs to be stated.

Factorization with elementary sieving

Assume all primes below B are known.
In a sieve region $\mathcal{R} = [A, 2A]$ (for example):

- Goal 1: for each $N \in \mathcal{R}$, find its prime factors below B.
- Goal 2: find all $N \in \mathcal{R}$ whose prime factors are all below B. These are called smooth numbers.
- Goal 3: like goal 2, but list the prime factors of smooth numbers, too.

Whether our goal is 1, 2, or 3, we have an array with a value tied to each $n \in \mathcal{R}$.

Finding smooth parts with sieving

To achieve goal 1:

- Initialize the array cell indexed by n with the integer n itself.
- Whenever this index is identified as a multiple of a prime below B, divide it (perhaps several times), and store the information about the divided values.
- Eventually, information at index n gives the prime divisors of n below B, as well as the cofactor

This is expensive.

- Memory: $O(\#\mathcal{R} \cdot \log_2 n)$

  (which is also the size of the output).
- Arithmetic cost is large as well, with many divisions.

Identifying smooth integers with sieving

To achieve goal 2:

- Initialize the array cell indexed by n with an approximation of $\log n$.
- When sieving, subtract $\log p$ at the sieved position.
- In the end, positions with a small remaining value are likely to be smooth.
- Some caveats: rounding, prime powers.

This is a very simple, yet very important mechanism.

Cost: $O(\#\mathcal{R})$ approximated values, and $O(\#\mathcal{R} \cdot \log\log B)$ additions/subtractions.

Improvements discussed at length, especially when $\mathcal{R}$ is large.

Factoring smooth integers with sieving

To achieve goal 3, one option is to do as in goal 1, and filter the results.

Better:

- Do detection first (as in goal 2).
- In a second step, do re-sieving, but keep information only for the indices that we know are smooth.

This is very worthwhile when smooth numbers are rare.

This re-sieving technique will appear (much) later on in the NFS context as well.

Plan

Factoring with simple sieving

The product tree approach

Batch smoothness detection

Fact. The Sieve of Eratosthenes relies on two properties:

- The set of numbers to test for primality (or for smoothness) has a structure: *arithmetic progression*.
- The set of primes we consider has a structure: *all the primes up to a bound*.

What can we do with less or no structure? Such situations exist:

- Coppersmith's variant of NFS with several number fields: *tested numbers have no structure*
- Large-prime separation: *primes in an interval, and some prime are forbidden*

Shopping list

Main tool. Asymptotically fast integer arithmetic. Using FFT-based techniques, integers of n bits can be multiplied, divided in almost linear time. Same for GCD.

Notation. $\mathrm{M}(n)$ is the cost of multiplying two integers of n bits. Division costs $O(\mathrm{M}(n))$ and GCD costs $O(\mathrm{M}(n)\log n)$.

Note: asymptotically fast multiplication algorithms are readily available in software. The GNU multiprecision library (GMP), for example, has an implementation of the Schönhage-Strassen.

The multiplication of two integers of one billion bits each takes about. . . (your guess).

Batch'ed trial division

Input.

- A set of integers $x_1, \ldots, x_k$;
- A factor base $\mathcal{F}$, i.e. a set of primes $p_1, \ldots, p_\ell$;

Output. The x_i that are $\mathcal{F}$-smooth.

Idea. Compute the GCD of x_i with $\prod p_j$.

If these GCD are computed sequentially, we get a quadratic complexity.

Rationale. Try to do operations between integers of the same size.

First step: $\Pi\, p_j$

The approach is pretty bold.

Multiply all p_j's together!
Multiply all x_i's together!
Collect winnings.

$P = \prod p_j$ a big number. The product of all primes below 2^B is a $2^{B+0.53}$-bit integer.

Fortunately, fast integer arithmetic is not only useful in theory, it is also useful in practice!

Strategies to compute $P = \Pi\, p_j$

Naive. Even with fast multiplication, $\prod p_j$ costs a quadratic bit-complexity.

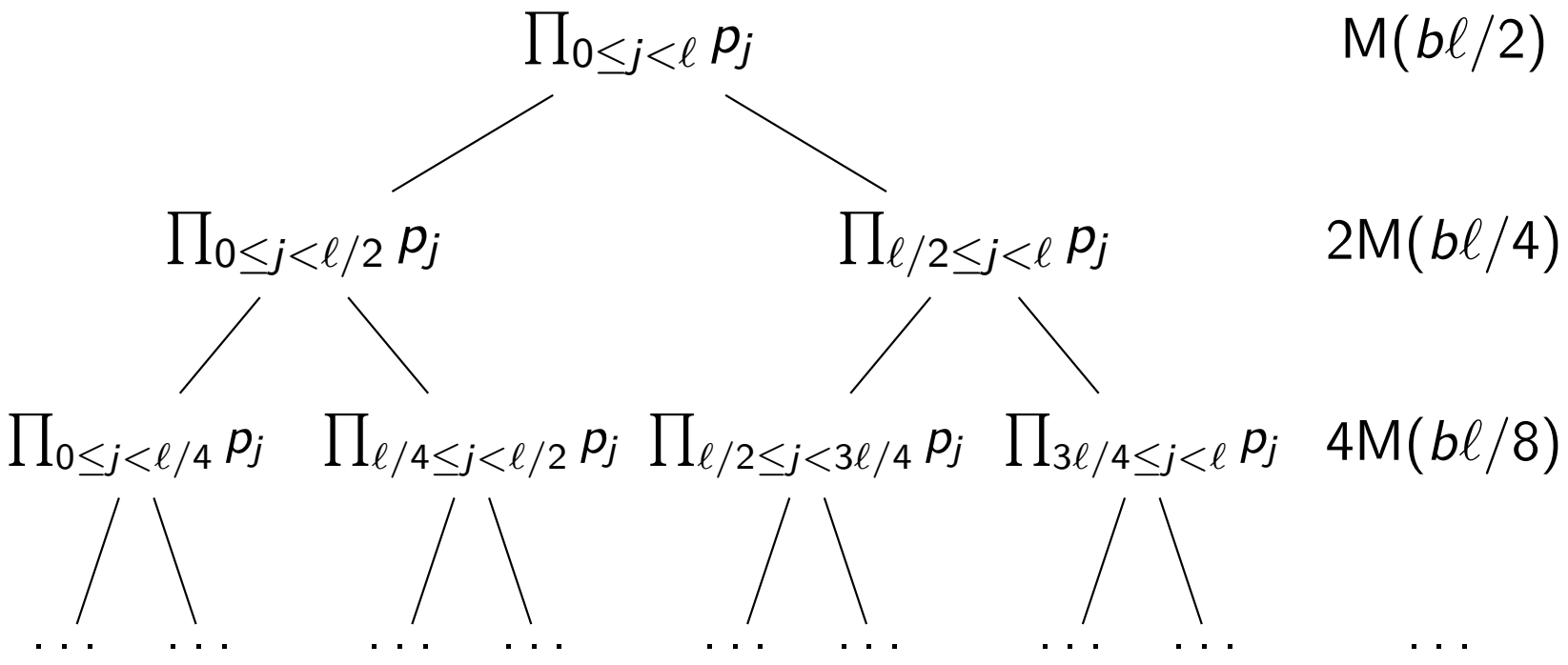
Subproduct-tree.

Assume the number ℓ of p_j is a power of 2, and build a binary tree, from leaves that are the primes. Do a multiplication at each node.

- Same number of multiplications;
- All of them are balanced (the two operands have the same size).

Complexity. If all primes have b bits, total cost is $O(\mathsf{M}(\ell b) \log \ell)$.

First step: $\prod p_j$

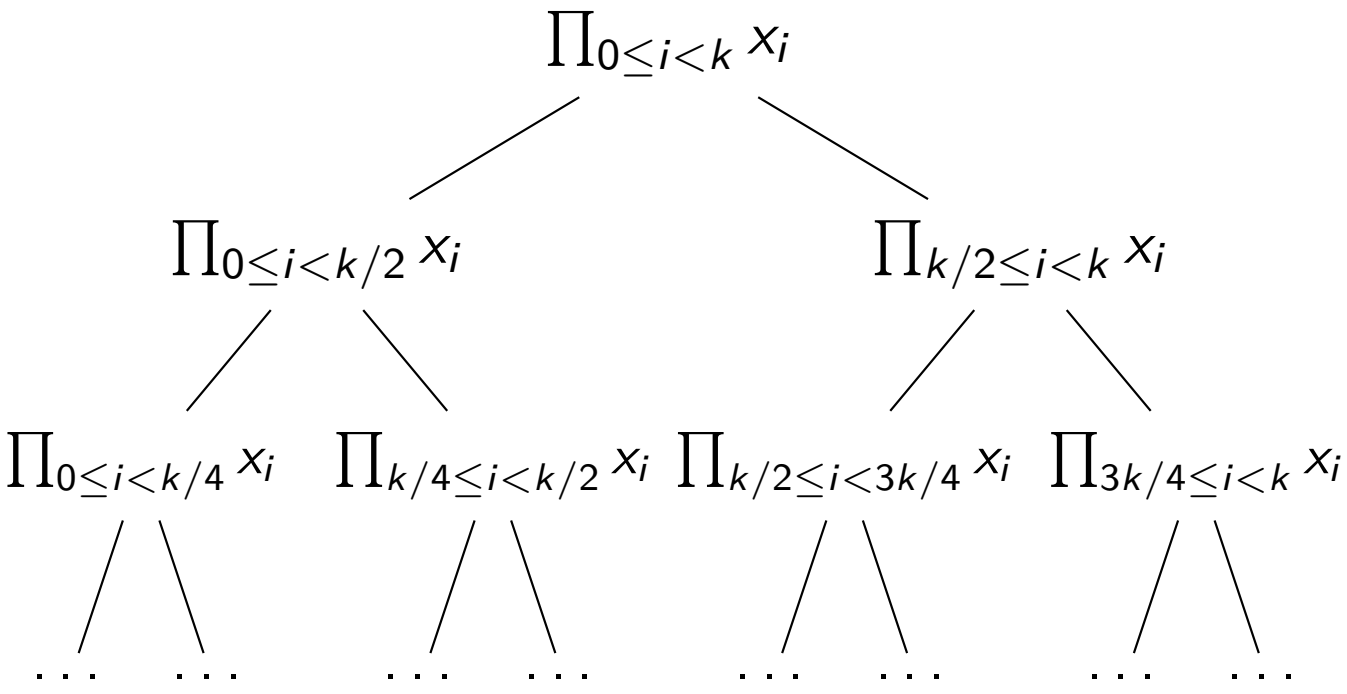


Although quasi-linear, multiplication is still supra-linear:

$4\mathsf{M}(b\ell/8) \le 2\mathsf{M}(b\ell/4) \le \mathsf{M}(b\ell/2)$.

Consequence. If M is close to linear, same cost at each level.

Do the same for x_i

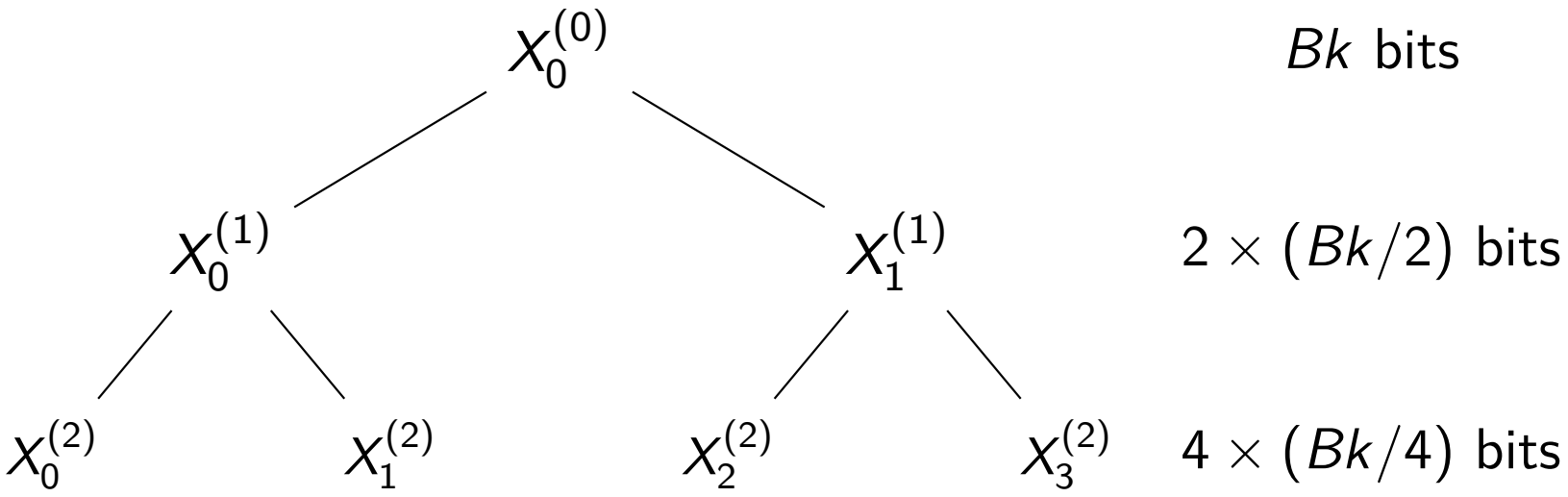


Rem. If the x_i have B bits, cost of this construction is $O(\mathrm{M}(Bk)\log k)$.

Important. Keep the whole tree in memory (and lose a log factor in space complexity).

The tree of the x_i stays in memory

Let $X_i^{(r)}$ denotes the i-th node at depth r from the root.



We have a single large integer on top, and a large collection of small integers at the bottom.

Descend the remainder tree

Let $P = \prod p_j$ that has been computed.

Remember that we want the GCD of P with the leaves x_i.

Idea. Compute $P \bmod x_i$ before taking these GCD. For that, we descend P along the remainder tree of the x_i's.

Let $R_i^{(r)} = P \bmod X_i^{(r)}$. Since $X_i^{(r-1)} = X_{2i}^{(r)} X_{2i+1}^{(r)}$, we have

$$R_{2i}^{(r)} = R_i^{(r-1)} \bmod X_{2i}^{(r)} \quad \text{and} \quad R_{2i+1}^{(r)} = R_i^{(r-1)} \bmod X_{2i+1}^{(r)}.$$

$$R_0^{(0)} = P \bmod X_0^{(0)}$$

$$R_0^{(1)} = R_0^{(0)} \bmod X_0^{(1)} \qquad R_0^{(1)} = R_1^{(0)} \bmod X_1^{(1)}$$

$$R_0^{(2)} \qquad R_1^{(2)} \qquad R_2^{(2)} \qquad R_3^{(2)}$$

Descend the remainder tree

At level r, this will cost 2^r divisions of an integer of size $Bk/2^{r-1}$ and one of size $Bk/2^r$. Close to $O(\mathrm{M}(Bk))$.

Again, the total cost is $O(\mathrm{M}(Bk)\log k)$ plus $O(\mathrm{M}(Bk, b\ell))$ for the first step.

Conclusion. We can compute all the GCD of $\prod p_j$ and the x_i in a time that is quasi-linear in the input size.

Is it practical? YES!!! Several fun stories related to that. (try to search "GCD all the keys" or something similar).

Left as exercise. Handle powers, deduce the full factorization. Everything follows more or less easily, in the same complexity.
Main reference: D. J. Bernstein.
Sage scripts avalaible at `https://facthacks.cr.yp.to/`

Wrap up

- Prime testing is not a difficulty for the usage range that we target.
- Sieving is unsurprisingly a very basic building block that will resurface. Remember resieving.
- Batch smoothness detection is a very interesting tool. It's not only fun, but also useful in NFS context.

CSE291-14: The Number Field Sieve

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

Emmanuel Thomé

January 6, 2022

Part 1d

Old factoring algorithms

Pollard ρ

$p - 1$ and $p + 1$

ECM

Mundane factoring needs within NFS

Enumerating primes gives a method to factor a number.
This is called trial division (TD).

TD works only to some extent

The required time to trial-divide N by all prime numbers below B with sieving is roughly $\widetilde{O}(B \log N)$

Many other integer factorization algorithms can be used, with the common characteristic:

- Runtime is polynomial in $\log N$.
- $\frac{\text{runtime}}{\text{success probability}}$ is (at most) exponential in $\log B$.

Within NFS, these algorithms are used to obtain auxiliary factorization of many intermediate numbers.

- Pollard rho.
- $p-1$, $p+1$, the Elliptic Curve Method (ECM);

Second goal: combinations of congruences

A second class of algorithms is given by those whose complexity depends only on N (albeit super-polynomially).

These algorithms are the precursors of the lineage that culminates with NFS.

- Fermat factoring.
- Dixon random squares method.
- CFRAC: the continued fraction method.
- QS: the quadratic sieve.

Plan

Pollard ρ

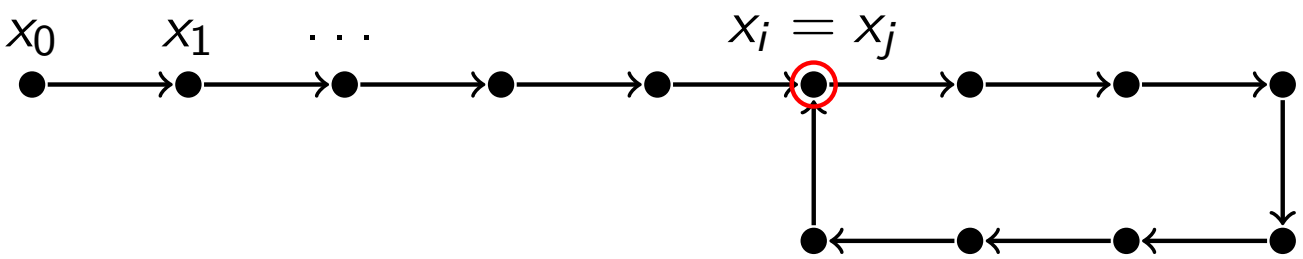
$p - 1$ and $p + 1$

ECM

Pollard ρ

Idea. Pick k random elements x_i modulo N. If two of them are equal modulo p, then $\gcd(x_i - x_j, N)$ is likely to give p.

Trick. The number of GCD to test is quadratic. To avoid that, use a pseudo-random sequence $x_{i+1} = f(x_i)$ and cycle detection.



Analysis. Birthday paradox says that the first collision occurs after $O(\sqrt{p})$ elements. More rigourous: use properties of the functional graph of f. AofA can find constant in $O()$.

Pollard ρ

Implementation.

- Use $f(x) = x^2 + c$. This gives enough randomness (quantified with expander graph theory).
- Use Floyd cycle detection: run two sequences in parallel, one going twice as fast as the other. Lose at most a factor of 2.

The overall complexity is $O(\sqrt{p})$ operation modulo N to extract a factor p.

In the worst case, where N is RSA, this gives $O(N^{1/4})$.

Pollard-Strassen

Pollard ρ is heuristic.

In a nearby (asymptotic) complexity ballpark, the Pollard-Strassen is slightly more expensive, but proven and not heuristic.

Pollard-Strassen

We are after a prime factor $p < B$ of N. Let $C = \left\lceil \sqrt{B} \right\rceil$.

- Compute $P = \prod_{i=0}^{C-1} (X - i) \in \mathbb{Z}/N\mathbb{Z}[X]$. Keep product tree.

  This costs $\mathrm{M}(C) \log C$ operations.

- Compute $Q = \prod_{i=0}^{C-1} (X + Ci) = (-C)^C P(-X/C)$.

  This costs $O(C)$ operations.

- Multi-evaluate Q at the roots of P.

  This costs $\mathrm{M}(C) \log C$ operations.

- Any evaluation that has a non-trivial gcd with N narrows down a potential factor to a range of size C.

PS is not useful in practice, but is a fun application of asymptotically fast algorithms!

Plan

Pollard ρ

$p - 1$ and $p + 1$

ECM

$p - 1$ and $p + 1$

Another family of algorithms.

- One aspect in common with ρ.

  "Something" happens $\mod p \longrightarrow$ detect it $\mod N$.
- The gist of it is how "something" is defined.
- Some algebra is involved.

Pollard $p-1$ method

Idea: assume $p \mid N$ and a is prime to p. Then

$$(p \mid a^{p-1}-1 \text{ and } p \mid N) \Rightarrow p \mid \gcd(a^{p-1}-1, N).$$

Same if some R is known s.t. $p-1 \mid R$ and we compute

$$\gcd((a^R \bmod N)-1, N).$$

How do we find R ? Only reasonable hope is that $p-1 \mid B!$ for some (small) B. In other words, $p-1$ is B-smooth.

Algorithm: $R = \prod_{p^\alpha \le B} p^\alpha = \operatorname{lcm}(2, \ldots, B)$.

Our "**something**" is the event $a^R \equiv 1 \mod p$.

$p-1$ is one-shot

$p-1$ succeeds if N is divisible by some p with $p-1$ smooth.

If, for a given N, $p-1$ failed to find a factor, you need to find another algorithm to factor it.

If we fix B and consider many integers N_i with one known factor $\approx 2^x$, $p-1$ will return a factor for a fixed fraction of the input† and will fail for the rest.

† $\frac{\psi(2^x,B)}{2^x}$ (see lecture about smoothness)

The $p+1$ method (Williams, Guy, ...)

Idea. Work in an extension of degree 2 of $\mathbb{F}_p$.

The multiplicative group is of order $p^2 - 1 = (p-1)(p+1)$, and the subgroup $\mathbb{T}_2(p)$ of elements of norm 1 is of order $p+1$.

Difficulties.

- We do not know p; can not be sure to work in a genuine field extension.
- How do we work with elements of norm 1 anyway?

$p+1$: working around difficulties

Implicit representation of $T_2(p)$

Fact: if θ is a root of $x^2 - Ax + 1 \mod p$, then:

- the other root is $1/\theta$.
- if $D = A^2 - 4$ is a non-square, θ is an element of norm 1 in $\mathbb{F}_p(\sqrt{D}) \approx \mathbb{F}_{p^2}$ (i.e., an element of $\mathbb{T}_2(p)$).
- if D is a square, θ is simply an element of $\mathbb{F}_p$. We'll be redoing $p-1$ inadvertently.

In effect, we're choosing (the shape of) θ first, and D afterwards.

- We don't know if $\sqrt{D}$ defines $\mathbb{F}_{p^2}$ or not.
- In either case, if $p - \left(\frac{D}{p}\right)$ is B-smooth, then $\theta^{B!} \equiv 1 \mod p$.
- How do we compute with θ^n?

$p+1$: computing θ^n

We only need to care about $v_n = \theta^n + \theta^{-n} \in \mathbb{F}_p$. We have:

$$v_0 = 2, \quad v_1 = A, \quad v_{m+n} = v_m v_n - v_{m-n}.$$

We use a Montgomery ladder to compute $\{v_n, v_{n+1}\}$.

$$\{v_n, v_{n+1}\} \to \{v_{2n}, v_{2n+1}\}.$$
$$v_{2n} = v_n^2 - v_0,$$
$$v_{2n+1} = v_n v_{n+1} - v_1.$$

$$\{v_n, v_{n+1}\} \to \{v_{2n+1}, v_{2n+2}\}.$$
$$v_{2n+1} = v_n v_{n+1} - v_1,$$
$$v_{2n+2} = v_{n+1}^2 - v_0.$$

Very similar to standard binary powering

Requires $\log_2 n$ mult and $\log_2 n$ squares modulo N. (compared to $4.5 \log_2 n$ operations with naive representation of $\mathbb{F}_p(\sqrt{D})$).

$p + 1$: summary

As the $p - 1$ method, this is a one-shot method.

It is not a lot more expensive than the $p - 1$ method.

However, it brings something new only 50% of the time.

Can this be generalized:

- Yes, but generalizations with field extensions don't work as well.
  - See Factoring with cyclotomic polynomials.
  - Work in algebraic tori which are varieties of dimension $\phi(d)$.
- The "good" generalization is ECM. Curves are varieties of dimension 1.

Handling close misses

The $p-1$ and $p+1$ algorithms succeed if
$p \pm 1 = \text{small} \times \text{small} \times \cdots \times \text{small}$.

What happens if $p \pm 1 = \text{small} \times \text{small} \times \cdots \times \text{medium}$?

Assuming there is only one extra factor of medium size, it can be caught with the Stage 2 algorithms.

Second phase: the classical one

Let $b = a^R \bmod N$ and $\gcd(b, N) = 1$.

Hyp. $p - 1 = Qs$ with $Q \mid R$ and s prime, $B_1 < s \leq B_2$.

Test: is $\gcd(b^s - 1, N) > 1$ for some s.

Let s_j denote the j-th prime. In practice all $s_{j+1} - s_j$ are small (Cramer's conjecture: $s_{j+1} - s_j \leq (\log B_2)^2$).

- Precompute $c_\delta \equiv b^\delta \bmod N$ for all possible δ (small);
- Compute next value with one multiplication:

$$b^{s_{j+1}} = b^{s_j} c_{s_{j+1} - s_j} \bmod N.$$

Cost: $O((\log B_2)^2) + O(\log s_1) + (\pi(B_2) - \pi(B_1))$ multiplications $+(\pi(B_2) - \pi(B_1))$ gcd's. When $B_2 \gg B_1$, $\pi(B_2)$ dominates.

Rem. We need to enumerate all primes $< B_2$; use a table of size $O(B_2)$ or a low-memory (segmented) Eratosthenes sieve.

Second phase: faster

Select $w \approx \sqrt{B_2}$, $v_1 = \lceil B_1/w \rceil$, $v_2 = \lceil B_2/w \rceil$.

Write our prime s as $s = vw - u$, with $0 \le u < w$, $v_1 \le v \le v_2$.
One has $\gcd(b^s - 1, N) > 1$ iff $\gcd(b^{wv} - b^u, N) > 1$.

1. Precompute $b^u \bmod N$ for all $0 \le u < w$.
2. Precompute all $(b^w)^v$ for all $v_1 \le v \le v_2$.
3. For all u and all v evaluate $\gcd(b^{vw} - b^u, N)$.

Number of multiplications is
$w + (v_2 - v_1) + O(\log_2 w) = O(\sqrt{B_2})$, memory is also $O(\sqrt{B_2})$.

Number of *gcd* is still $\pi(B_2) - \pi(B_1)$.

Second phase: faster

Algorithm:
- Compute $h(X) = \prod_{0 \le u < w}(X - b^u) \in \mathbb{Z}/N\mathbb{Z}[X]$
- Evaluate all $h((b^w)^v)$ for all $v_1 \le v \le v_2$.
- Evaluate all $\gcd(h(b^{wv}), N)$.

Analysis, using product trees:

Step 1: $O((\log w)\mathrm{M}(w))$ operations (product tree).

Step 2: $O((\log w)\mathrm{M}(\log N))$ for b^w; $v_2 - v_1$ for $(b^w)^v$; multi-point evaluation on w points takes $O((\log w)\mathrm{M}(w))$.

Rem. Evaluating $h(X)$ along a geometric progression of length w takes $O(w \log w)$ operations (see Montgomery-Silverman).

Total cost: $O((\log w)\mathrm{M}(w)) = O(B_2^{0.5+o(1)})$.

Plan

Pollard ρ

$p - 1$ and $p + 1$

ECM

The ECM algorithm

The starting observation is that Pollard $p-1$ is nice, but of limited use.

- Pollard $p-1$ implicitly uses $\mathbb{F}_p^*$. And there is only one $\mathbb{F}_p^*$ per p.
- The $p+1$ algorithm uses another group which therefore increases the factoring chances.
- But it basically stops here.

What ECM achieves is that it works works with a structure defined modulo p (and therefore computable implicitly with arithmetic modulo N), which has many different instances.

This is done with Elliptic curves.

The ECM algorithm

ECM: the Elliptic Curve factoring Method. ECM = a variant of $p \pm 1$ which is Not a one-shot algorithm.

An elliptic curve: set of solutions of certain algebraic systems.

- $y^2 = x^3 + ax + b$ (with constants a, b).
- $By^2 = x^3 + Ax^2 + x$ (with constants A, B).
- $x^2 + y^2 = 1 + dx^2y^2$ (with constant d).
- $x^2 + y^2 = 1$ and $ax^2 + z^2 = 1$ (with constant a).

All are different ways to define isomorphic mathematical objects.

Elliptic curves over $\mathbb{F}_p$

Given an equation that defines an elliptic curve E:
$E(\mathbb{F}_p) = \{\text{points with coordinates in } \mathbb{F}_p\} \cup \{\infty\}$.

- Elements of $E(\mathbb{F}_p)$ can be represented easily.
- $E(\mathbb{F}_p)$ forms a finite group with easily computable group law.
- $p + 1 - 2\sqrt{p} \leq \#E(\mathbb{F}_p) \leq p + 1 + 2\sqrt{p}$.
- If we work modulo N, we also work in $E(\mathbb{F}_p)$ implicitly.
- A match in $E(\mathbb{F}_p)$ can be detected with arithmetic modulo N by gcd.

The common notation is to write the group law on elliptic curves additively.

- ∞ is the neutral element.
- $P + Q$ is the group law.
- $[n]P$ is done by double-and-add (a.k.a. square-and-multiply, additively).

ECM: adapting $p-1$

ECM works in the same way as $p-1$ and $p+1$.

- Pick a curve and a point P on it.
  Important: do it in one go, e.g.

$$x, y, a \in_R \mathbb{Z}/N\mathbb{Z}, \text{ then let } b = y^2 - x^3 - ax.$$

- Hope for $\#E(\mathbb{F}_p)$ to be B_1-smooth for a divisor p of N.
- Compute $[B_1!]P$. Test (gcd) if something happened mod p.
- Stage 2 works, too.
- If no factor found, start over with a new curve.

Arithmetic

ECM is de facto the home of most improvements on these $(p-1)$-like factoring methods.

- Arithmetic: a lot of time is spent in computations modulo N. It is worthwhile to use trade-offs (such as Montgomery multiplication), and size-specific code.
- Some ways to choose an elliptic curves and a point on them are better than others (we can force some small factors in $\#E(\mathbb{F}_p)$).
- For fixed B_1, there are ways to compute $[B_1!]P$ slightly more efficiently than by square-and-multiply (Lucas chains, PRAC).

ECM performance

ECM finds a factor $\approx p$ of an n-bit integer N in time:

$$\exp\left(\sqrt{2}(\log p)^{1/2}(\log\log p)^{1/2}(1+o(1))\right) \times \mathrm{M}(n).$$

This is called a sub-exponential complexity (w.r.t $\log p$).

ECM is very efficient for $p \approx 10^{30\ldots40}$, record of $p \approx 10^{83}$.

ECM can be distributed massively.

Reference implementation: GMP-ECM (Paul Zimmermann).

Note: we'll see this funny kind of complexity again!

ECM: fun and profit

Factoring enthusiasts like big ECM hits.

Yearly top ten table here.

It's the beginning of January, the best time of the year to enter that table (at least temporarily).

- A few core-years and/or luck to make it to the table (60 digits).
- A few dozen core-years to find a 70 digit factor.

CSE291-14: The Number Field Sieve

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

Emmanuel Thomé

January 11, 2022

Part 2

Combinations of congruences

Combining congruences

CFRAC and QS

QS-era improvements

The golden age of QS and MPQS

Analysis of QS

Plan

Combining congruences

CFRAC and QS

QS-era improvements

The golden age of QS and MPQS

Analysis of QS

Combining congruences

An early idea due to Fermat. We try to factor N. Set $r = \lceil \sqrt{N} \rceil$.

- For $i = 0, \ldots,$ compute $f(i) = (r+i)^2 - N$.
- If $f(i)$ is a square, then we have:

$$\begin{aligned} (r+i)^2 - N &= x^2, \\ (r+i-x)(r+i+x) &= N. \end{aligned}$$

Let $N = pq$. This method factors N in time $O(\frac{|p-q|}{2})$.
This suceeds if p, q are too close to $\sqrt{N}$. Otherwise hopeless.

Note: tricks to make this work without long integer arithmetic:

- $f(i+1) - f(i) = 2(r+i) + 1$.
- To test whether $f(i) = \square$, look at $\left(\frac{f(i)}{p}\right)$ for many small p's.

This saves some work, but does not change the outcome much.

Remaining idea: search for squares.

Combining congruences

Given a composite N, what does $x^2 \equiv y^2 \mod N$ give ?

$$\begin{aligned} x^2 &\equiv y^2, \\ (x-y)(x+y) &\equiv 0, \\ (\frac{x}{y}-1)(\frac{x}{y}+1) &\equiv 0 \quad \text{(we may assume } \gcd(y,N)=1\text{)}. \end{aligned}$$

- If N has k distinct prime factors, there are 2^k different square roots
- A "random" congruence $x^2 \equiv y^2$ reveals a factor with prob $1 - \frac{1}{2^{k-1}}$.
- Note that this cannot work for prime powers.

Kraitchik

From the 1930's:

- Looking at congruences is enough.
- If $r^2 \bmod N$ and $s^2 \bmod N$ are not squares, but their product is, then we succeed.

This is the principle of combination of congruences.

Combination of congruences

$$46^2 \bmod 2041 = 75 = 3 \times 5^2,$$
$$47^2 \bmod 2041 = 168 = 2^3 \times 3 \times 7,$$
$$48^2 \bmod 2041 = 263 = \text{I am lazy, too hard}\ldots$$
$$49^2 \bmod 2041 = 360 = 2^3 \times 3^2 \times 5,$$
$$50^2 \bmod 2041 = 459 = 3^3 \times 17,$$
$$51^2 \bmod 2041 = 560 = 2^4 \times 5 \times 7, \qquad \ldots$$

This leads to

$$(\underbrace{46 \times 47 \times 49 \times 51}_{x})^2 \equiv 2^{10}3^4 5^4 7^2 \equiv (\underbrace{2^5 3^2 5^2 7}_{y})^2 \mod N.$$

And $\gcd(x - y, N) = 13$.

Dixon random squares algorithm

This was formalized by Dixon in the 1970s. Proven probabilistic.

Smoothness is the important thing!

We are interested in the factorization of $r^2 \bmod N$ only if it is **smooth**.

- We fix a smoothness bound B.
- The set of primes $\mathcal{P}_B$ is called the factor base.

Algorithm:

- Pick r at random. Test divisibility by all primes below B. If $r^2 \bmod N$ is B-smooth, keep the relation:

$$r_i^2 \bmod N \equiv p_1^{e_{i,1}} \times \cdots \times p_k^{e_{i,k}}.$$

- Try to match these. This is a linear algebra problem over $\mathbb{F}_2$.

Combination by linear algebra

We have a set $\mathcal{R}$ of relations $r_i^2 \equiv p_1^{e_{i,1}} \times \cdots \times p_k^{e_{i,k}}$.

- Consider the matrix $M \in \mathcal{M}_{\#\mathcal{R}\times\#\mathcal{P}}(\mathbb{Z})$, $M = (e_{i,j})$.

$$\begin{aligned} 46^2 \bmod 2041 &= 75 = 3 \times 5^2, \\ 47^2 \bmod 2041 &= 168 = 2^3 \times 3 \times 7, \\ 49^2 \bmod 2041 &= 360 = 2^3 \times 3^2 \times 5, \\ 51^2 \bmod 2041 &= 560 = 2^4 \times 5 \times 7, \end{aligned} \implies \begin{pmatrix} 0 & 1 & 2 & 0 \\ 3 & 1 & 0 & 1 \\ 3 & 2 & 1 & 0 \\ 4 & 0 & 1 & 1 \end{pmatrix}$$

- A vector $V = (v_i)_{1\le i\le \#\mathcal{R}}$ yields $VM = (\sum_i v_i e_{i,j})_j$, and:

$$(\prod r_i^{v_i})^2 \equiv \prod_j p_j^{\sum_i v_i e_{i,j}}.$$

- We want V such that coordinates of VM are even: it suffices to search for nullspace elements over the field $\mathbb{F}_2$.

Pitfalls

The random squares method has one main pitfall

$$r^2 \mod N \text{ is big: as large as } N.$$

Further improvements focused on making the numbers to test for smoothness somewhat smaller.

Plan

Combining congruences

CFRAC and QS

QS-era improvements

The golden age of QS and MPQS

Analysis of QS

CFRAC

The CFRAC algorithm (Lehmer&Powers, 1931, and Morrison&Brillhart, 1975) was the first practical algorithm to factor large numbers, starting with $F_7 = 2^{128} + 1$.

Idea: use the continued fraction approximation of $\sqrt{N}$.

How continued fractions work is barely relevant, but a by-product is a host of identities of the form:

$$U_n^2 - NV_n^2 = Q_n,$$
$$U_n^2 \equiv Q_n \mod N,$$

with the very sweet property that $|Q_n| < 2\sqrt{N}$.

Test the Q_n's for smoothness. They are only as large as $2\sqrt{N}$.

The quadratic sieve (Pomerance)

In fact, continued fractions are not achieving anything magical.

A simpler supply of small square residues:

- Let $f(i) = \left(\left\lceil \sqrt{N} \right\rceil + i\right)^2 - N$.
- As long as i remains small, $f(i)$ is of the order of $\sqrt{N}$.

Fermat was using the same f, hoping for $f(i)$ to be a square.
Here, we are combining this with the Dixon-like approach.

QS: enter sieving

QS turn the Dixon approach on its head when it comes to factoring the residues.

Decide beforehand on a sieving space: interval $i \in [-A, A]$. We will try to factor all residues $f(i)$.

- Dixon: for each i, for each p, see if p divides $f(i)$.
- QS: for each p, for each i, see if p divides $f(i)$.
- QS: for each p, determine indices i such that p divides $f(i)$.

Computing all valuations at once

Fix p. Let r_0, r_1 be the two roots of the quadratic equation $f(i) \equiv 0 \mod p$.

$$\{i \in [-A, A],\ f(i) \equiv 0 \mod p\} = \{r_0, r_0 \pm p, \ldots\} \cup \{r_1, r_1 \pm p, \ldots\}.$$

Algorithm: We maintain an array $T[i]$ indexed by $i \in [-A, A]$.

- For each $p \leq B$, do:
  - Compute r_0, r_1
  - $r := r_0$. While $r \leq A$ do:
    - $T[r] \leftarrow T[r] + \log p$,
    - $r \leftarrow r + p$.
  - idem for r_1 as well as $\{r_i - kp\}$.
- Do this also for prime powers
- For all i such that $T[i] = \log |f(i)|$, we know that $f(i)$ is smooth.

$T[i] = \log|f(i)| \Leftrightarrow f(i)$ smooth

Ignore powers for the moment. For each p:

- we have characterized the set $\mathcal{S}_{p,i} = \{i,\ \nu_p(f(i)) > 0\}$.
- we have added $\log_2 p$ to $T[i]$ for each i in this set.

Once we have sieved for all $p \in \mathcal{P}_B$:

$$\begin{aligned} T[i] &= \sum_{p \in \mathcal{P}_B} \begin{cases} \log_2 p \text{ if } p \mid f(i), \\ 0 \text{ otherwise,} \end{cases} \\ &= \sum_{p \in \mathcal{P}_B,\ p \mid f(i)} \log p, \\ &= \log\left(B\text{-smooth part of } f(i) \text{ without powers}\right). \end{aligned}$$

If $T[i] = \log|f(i)|$ after sieving, then $f(i)$ is B-smooth and square-free.

Sieving with powers

Assume that $f(i) \equiv 0$ has 2 distinct roots mod p (so $p \nmid \mathrm{disc}(f)$).

- How many roots mod p^2 ?
- How many roots mod p^k ?
- Which log contribution should we add ?

Sieving with powers

Assume that $f(i) \equiv 0$ has 2 distinct roots mod p (so $p \nmid \mathrm{disc}(f)$).

- How many roots mod p^2 ?
- How many roots mod p^k ?
- Which log contribution should we add ?

We want to have an accumulated contribution of $k \log_2 p$ when $p^k \mid f(i)$, but since $f(i)$ is hit when sieving for $p, p^2, \ldots, p^k$, we need only add $\log_2 p$ each time.

$T[i] = \log|f(i)| \Leftrightarrow f(i)$ smooth

For each p^k (assuming we consider k up to ∞. In fact we don't):

- we have characterized the set $\mathcal{S}_{p^k,i} = \{i,\ \nu_p(f(i)) \geq k\}$.
- we have added $\log_2 p$ to $T[i]$ for each i in this set.

Thus eventually:

$$\begin{aligned} T[i] &= \sum_{p \in \mathcal{P}_B} \left(\sum_{k \text{ s.t. } i \in \mathcal{S}_{p^k,i}} \log p \right), \\ &= \sum_{p \in \mathcal{P}_B} \left(\sum_{k,\ \nu_p(f(i)) \geq k} \log p \right), \\ &= \sum_{p \in \mathcal{P}_B} \nu_p(f(i)) \log p, \\ &= \log\left(B\text{-smooth part of } f(i)\right). \end{aligned}$$

QS: summary

We want to factor N. Let $f(x) = (\lceil\sqrt{N}\rceil + x)^2 - N$.

- Choose bounds A and B. Analysis will tell how.
- Create array of low-precision values T, indices $i \in [-A, A]$.
- For all primes below B:
  - Find roots of f mod p.
  - Add $\log_2 p$ to the appropriate locations.
  - Do this also for powers.
- For all i, compare $\log_2 |f(i)|$ and $T[i]$.
  Deduce the i's such that $f(i)$ is B-smooth.
- Factor the smooth $f(i)$, possibly with resieving (lecture 1).
- Form a big linear system. Find a nullspace element.
- Determine a congruence of squares. Attempt to factor N.

QS: the road ahead

In the previous summary of QS:

- The decor is basically here so that we can explain many of the following improvements.
- As it turns out, almost every line of the previous description has been a topic of study.
  These lines of study have all led to improvements that, most often are still part of relevant know-how for NFS.
  - Some improvements are very local.
  - Some have a broader consequence.

Plan

Combining congruences

CFRAC and QS

QS-era improvements

The golden age of QS and MPQS

Analysis of QS

QS: all steps deserve a look

We want to factor N. Let $f(x) = (\lceil\sqrt{N}\rceil + x)^2 - N$.

- Choose bounds A and B. Analysis will tell how.
- Create array of low-precision values T, indices $i \in [-A, A]$.
- For all primes below B:
  - Find roots of f mod p.
  - Add $\log_2 p$ to the appropriate locations.
  - Do this also for powers.
- For all i, compare $\log_2 |f(i)|$ and $T[i]$.
  Deduce the i's such that $f(i)$ is B-smooth.
- Factor the smooth $f(i)$, possibly with resieving (lecture 1).
- Form a big linear system. Find a nullspace element.
- Determine a congruence of squares. Attempt to factor N.

QS: all steps deserve a look

We want to factor N. Let $f(x) = (\lceil\sqrt{N}\rceil + x)^2 - N$.

- Choose bounds A and B. Analysis will tell how.
- Create array of low-precision values T, indices $i \in [-A, A]$.
- For all primes below B:
  - Find roots of f mod p.
  - Add $\log_2 p$ to the appropriate locations.
  - Do this also for powers.
- For all i, compare $\log_2 |f(i)|$ and $T[i]$.
  Deduce the i's such that $f(i)$ is B-smooth.
- Factor the smooth $f(i)$, possibly with resieving (lecture 1).
- Form a big linear system. Find a nullspace element.
- Determine a congruence of squares. Attempt to factor N.

Data type for array cells $T[i]$

$\log_2 |f(i)|$ and $\log_2 p$ are real numbers. What do we store in $T[i]$?

- Straightforward choice: store the integer $\lfloor \log_2 |f(i)| \rceil$, which certainly fits in 8 bits.
  This introduces some fuzziness, because of rounding. We can probably live with it.
- Better choice: adjust the log base so that the full 8-bit range is used. This is better for accuracy.
  - Determine the max value of $|f(i)|$ beforehand.
  - Provide for some tolerance for rounding.

QS: all steps deserve a look

We want to factor N. Let $f(x) = (\lceil\sqrt{N}\rceil + x)^2 - N$.

- Choose bounds A and B. Analysis will tell how.
- Create array of low-precision values T, indices $i \in [-A, A]$.
- For all primes below B:
  - Find roots of f mod p.
  - Add $\log_\beta p$ to the appropriate locations.
  - Do this also for powers.
- For all i, compare $\log_\beta |f(i)|$ and $T[i]$.
  Deduce the i's such that $f(i)$ is B-smooth.
- Factor the smooth $f(i)$, possibly with resieving (lecture 1).
- Form a big linear system. Find a nullspace element.
- Determine a congruence of squares. Attempt to factor N.

log-norm computations and comparisons

When we compare $T[i]$ with $\log_\beta |f(i)|$, do we actually compute $f(i)$ and its log? Of course not.

- Keeping track of $f(i)$ alone would not be too hard.
- Once we take the log, the (rounded) value changes rarely.

In the QS context, this is rather easy to solve. This is an instance of the log-norm computations issue, that appears also in NFS.

Note that in practice, we prefer to do log-norm computations first:

- Fill $T[i]$ with rounded values of $\log_\beta |f(i)|$.
- Subtract the $\log_\beta p$ from array cell.

The benefit is that comparisons are done versus a constant bound.

QS: all steps deserve a look

We want to factor N. Let $f(x) = (\lceil\sqrt{N}\rceil + x)^2 - N$.

- Choose bounds A and B. Analysis will tell how.
- Create array of low-precision values T, indices $i \in [-A, A]$.
- For all primes below B:
  - Find roots of f mod p.
  - Add $\log_\beta p$ to the appropriate locations.
  - Do this also for powers.
- For all i, compare $\log_\beta |f(i)|$ and $T[i]$.
  Deduce the i's such that $f(i)$ is B-smooth.
- Factor the smooth $f(i)$, possibly with resieving (lecture 1).
- Form a big linear system. Find a nullspace element.
- Determine a congruence of squares. Attempt to factor N.

Not all primes are the same

As presented, QS is one very large sieving effort.

The sieving work varies a lot with p.

- p small: an immense number of narrowly-spaced table updates.
- p large: rare, far apart table updates.

The proper way to deal with that has been a key topic for decades, and is still a very current topic with NFS today.

QS: all steps deserve a look

We want to factor N. Let $f(x) = (\lceil\sqrt{N}\rceil + x)^2 - N$.

- Choose bounds A and B. Analysis will tell how.
- Create array of low-precision values T, indices $i \in [-A, A]$.
- For all primes below B:
  - Find roots of f mod p.
  - Add $\log_\beta p$ to the appropriate locations.
  - Do this also for powers.
- For all i, compare $\log_\beta |f(i)|$ and $T[i]$.
  Deduce the i's such that $f(i)$ is B-smooth.
- Factor the smooth $f(i)$, possibly with resieving (lecture 1).
- Form a big linear system. Find a nullspace element.
- Determine a congruence of squares. Attempt to factor N.

Linear algebra

How to deal with the linear systems that appear in factoring computations is a topic of its own.

Remember that as we only want to make all valuations even, linear algebra only needs to be done modulo 2.

Tentatively, we'll cover that mid-February.

QS: all steps deserve a look

We want to factor N. Let $f(x) = (\lceil\sqrt{N}\rceil + x)^2 - N$.

- Choose bounds A and B. Analysis will tell how.
- Create array of low-precision values T, indices $i \in [-A, A]$.
- For all primes below B:
  - Find roots of f mod p.
  - Add $\log_\beta p$ to the appropriate locations.
  - Do this also for powers.
- For all i, compare $\log_\beta |f(i)|$ and $T[i]$.
  Deduce the i's such that $f(i)$ is B-smooth.
- Factor the smooth $f(i)$, possibly with resieving (lecture 1).
- Form a big linear system. Find a nullspace element.
- Determine a congruence of squares. Attempt to factor N.

Congruence of squares

To form a congruence of squares, the task is pretty much trivial in the QS context.

- Multiply the $f(i)$ that participate in the winning combination.
- Compute the corresponding factored form $\prod_{p\in\mathcal{F}} p^{\nu_p}$, which has all valuation ν_p even. (We need the ν_p's in $\mathbb{Z}$.)
- Then we have:

$$\left(\prod(\lceil\sqrt{N}\rceil + i)\right)^2 = \pm\left(\prod_{p\in\mathcal{F}} p^{\nu_p/2}\right)^2.$$

  (it is ok to compute all that modulo N, of course.)
- Dealing with the sign is not too hard.
  - A sign is attached to each relation given by a smooth $f(i)$.
  - We can insert a sign column in our matrix ($+1 \to 0$; $-1 \to 1$) so that a winning combination is always positive.

In the NFS context, this step becomes somewhat more complicated (but remains negligible overall in terms of computation time).

Some improvements of NFS have broader consequences.

QS: all steps deserve a look

We want to factor N. Let $f(x) = (\lceil\sqrt{N}\rceil + x)^2 - N$.

- Choose bounds A and B. Analysis will tell how.
- Create array of low-precision values T, indices $i \in [-A, A]$.
- For all primes below B:
  - Find roots of f mod p.
  - Add $\log_\beta p$ to the appropriate locations.
  - Do this also for powers.
- For all i, compare $\log_\beta |f(i)|$ and $T[i]$.
  Deduce the i's such that $f(i)$ is B-smooth.
- Factor the smooth $f(i)$, possibly with resieving (lecture 1).
- Form a big linear system. Find a nullspace element.
- Determine a congruence of squares. Attempt to factor N.

Special-q

As described, QS uses a huge sieving table.

There are innumerable downsides to this.

- Diminishing returns;
- Need to have good aim;
- Splitting the computation into pieces is possible, but very irregular;
- Addressing a huge array in memory is not very efficient.

Special-q

The special-q idea was introduced in the early 1980s by Davis and Holridge, in the QS context.

- Pick a q that is slightly above B.
- Work only with i's that are such that $q|f(i)$.
  i.e., if $f(r) \equiv 0 \mod q$, work with $g(i) = f(qi + r)$.
- Do this for many different q's.

In effect, this divides the sieving work into many subtasks:

- One subtask per q. There are many primes around B.
- Each subtask only needs to address A/q indices.
- All subtasks cost roughly the same.
- There are a few downsides. Possible duplicates, and abundance of q's in the relation matrix.

This is a significant overhaul of how the whole algorithm works, but it pays off significantly. We will see this with NFS.

QS: all steps deserve a look

We want to factor N. Let $f(x) = (\lceil\sqrt{N}\rceil + x)^2 - N$.

- Choose bounds A and B. Analysis will tell how.
- Create array of low-precision values T, indices $i \in [-A, A]$.
- For all primes below B:
  - Find roots of f mod p.
  - Add $\log_\beta p$ to the appropriate locations.
  - Do this also for powers.
- For all i, compare $\log_\beta |f(i)|$ and $T[i]$.
  Deduce the i's such that $f(i)$ is B-smooth.
- Factor the smooth $f(i)$, possibly with resieving (lecture 1).
- Form a big linear system. Find a nullspace element.
- Determine a congruence of squares. Attempt to factor N.

Be tolerant

The comparison of $\log_\beta |f(i)|$ and $T[i]$ leads to an important choice: do we keep this i or not?

Note that $\left| T[i] - \log_\beta |f(i)| \right|$ measures the size of the non-smooth cofactor part in $f(i)$.

- We know that this cofactor is free of prime factors below B (except maybe powers).
- As a consequence, if the cofactor is less than B^2, it has to be a (large) prime number.

So this is "almost" a relation, and the extra prime is found at no extra cost.

This was the starting point of the large prime idea.

Large primes

Historical point of view. When the cofactor is a large prime, we have a "partial" relation.

- Set a bound $L \leq B^2$ on the large primes.
- When k partial relations are found, the birthday paradox tells us that we can form $\frac{k^2}{2L}$ combinations with large primes canceled in pairs.

Note that this point of view is vastly outdated. What remains is that the cofactor needs further inspection.

QS: all steps deserve a look

We want to factor N. Let $f(x) = (\lceil\sqrt{N}\rceil + x)^2 - N$.

- Choose bounds A and B. Analysis will tell how.
- Create array of low-precision values T, indices $i \in [-A, A]$.
- For all primes below B:
  - Find roots of f mod p.
  - Add $\log_\beta p$ to the appropriate locations.
  - Do this also for powers.
- For all i, compare $\log_\beta |f(i)|$ and $T[i]$.
  Deduce the i's such that $f(i)$ is B-smooth.
- Factor the smooth $f(i)$, possibly with resieving (lecture 1).
- Form a big linear system. Find a nullspace element.
- Determine a congruence of squares. Attempt to factor N.

MPQS (Montgomery)

Probably the most important (and most specific) improvement of QS was its multiple polynomial variant.

Annoying feature of QS: $|f(x)|$ gets bigger as x grows.

$$\max_{-A \le i \le A} |f(i)| = 2A\sqrt{N}.$$

We want to find other functions playing the role of $f(x)$.

MPQS (Montgomery)

What happens if we look at $(ux+v)^2$ for some u, v ?

$$(ux+v)^2 = u^2x^2 + 2uvx + v^2 - uw + uw \text{ for any } w,$$

If we have $v^2 - uw = N$:

$$\frac{1}{u}(ux+v)^2 \equiv ux^2 + 2vx + w \mod N.$$

Fix u s.t. $\left(\frac{N}{u}\right) = 1$. Choose $v \leq \frac{u}{2}$ s.t. $v^2 \equiv N \mod u$.
Set w accordingly. We have $w \approx -N/u$.

$$-\frac{N}{y} \overset{\sim}{<} ux^2 + 2vx + w \overset{\sim}{<} A^2u - \frac{N}{u}].$$

For a given A, smallest values for $u \approx \frac{2\sqrt{N}}{A}$. $\Rightarrow$ Bound $\frac{1}{\sqrt{2}}A\sqrt{N}$.

MPQS

- For a given sieve interval size, we have found a better polynomial.
- More important, we have many such polynomials.
- If u is a product of factor base primes, a large number of polynomials can be used (other option: $u = \square$).
- Shorter intervals per polynomial $\Rightarrow$ smaller residues.
- Initialization cost per polynomial: solving $v^2 \equiv N \mod u$. See SIQS for a way to amortize this (e.g. in Crandall-Pomerance).

MPQS/SIQS (with previous improvements) is the leading algorithm today for p below 100-120 decimal digits.

Software: msieve and yafu probably have the best QS code around.

Plan

Combining congruences

CFRAC and QS

QS-era improvements

The golden age of QS and MPQS

Analysis of QS

MPQS is efficient

MPQS is a great leap forward compared to CFRAC, for two reasons.

- The numbers that are checked for smoothness are considerably smaller.
- Massive distribution is possible. (although special-q's were a way to achieve that as well, it was not used that way).

Starting around 1986, and especially around the turn of the 1990s, many factoring records were broken with QS.

Factoring by electronic mail

Very influential paper by Lenstra and Manasse, 1989.

> "*how big are the integers we can factor within one month of elapsed time, if we only want to use computing time that we can get for free?*"

This pretty much defined the way to establish the state of the art in academic cryptanalysis for the following decades.

Of course, there is a built-in gap with the cryptanalytic power of nation-state adversaries.

Plan

Combining congruences

CFRAC and QS

QS-era improvements

The golden age of QS and MPQS

Analysis of QS

Analysis of QS

The analysis of QS employs techniques from various domains.

The most important things that we try to estimate are:

- The size of the numbers $f(i)$ that we try to factor.
- The probability that these are B-smooth.
- The time it takes to collect enough relations.
- The time it takes to solve the linear system.

Most of the analysis techniques (for smoothness probabilities, in particular) were only nascent in the beginning of the 1980s.

Smoothness: Estimating $\Psi(x, y)$

There's one main theorem known as:

- Canfield-Erdős-Pomerance (1983),
- Construction kit lemma,
- whatever credit people give... (Odlyzko / Balasubramanian)

It's also valid in various contexts.

Canfield-Erdős-Pomerance (CEP) Theorem

Let $x, y \to +\infty$ and $\epsilon > 0$ s.t. $(\log x)^\epsilon < \log y < (\log x)^{1-\epsilon}$.
Let $\Psi(x, y) = \#\{n,\ 1 \le n \le x,\ n \text{ is } y\text{-smooth}\}$.

$$\frac{1}{x}\Psi(x, y) \sim \rho(u) = u^{-u(1+o(1))}$$

where $u = \frac{\log x}{\log y}$, and ρ is the Dickman-de Bruijn function.

The Dickman-De Bruijn function

$\rho(u)$ is the solution of a delay differential equation.

$$u\rho'(u) + \rho(u-1) = 0.$$

Note that computing ρ is totally possible.

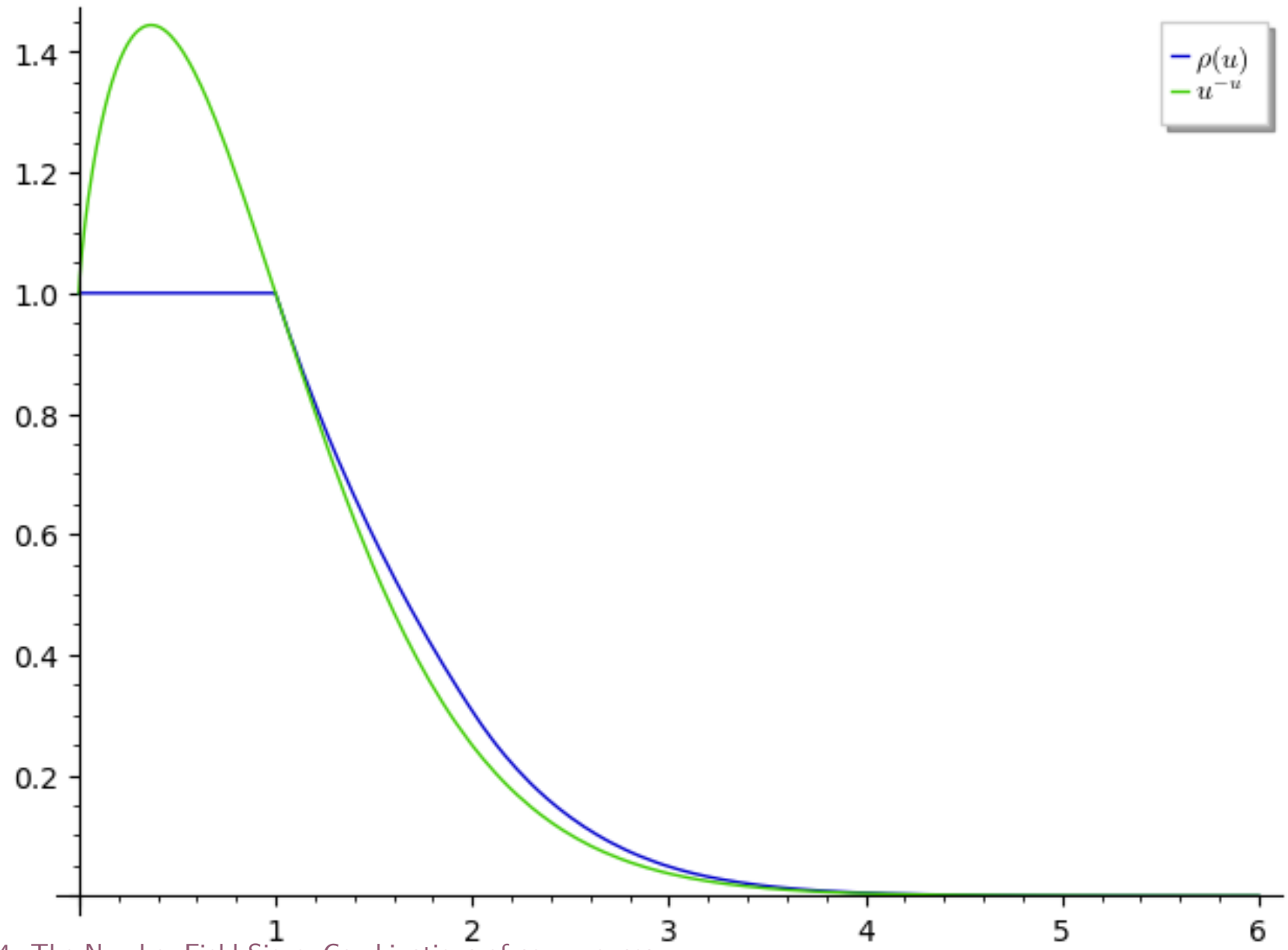
```
sage: plot(dickman_rho,x,0,10)
sage: plot(dickman_rho,x,0,10,scale='semilogy')
```

The asymptotic estimation of ρ is a pain, however, and this is what we use for the analysis of sieving algorithms:

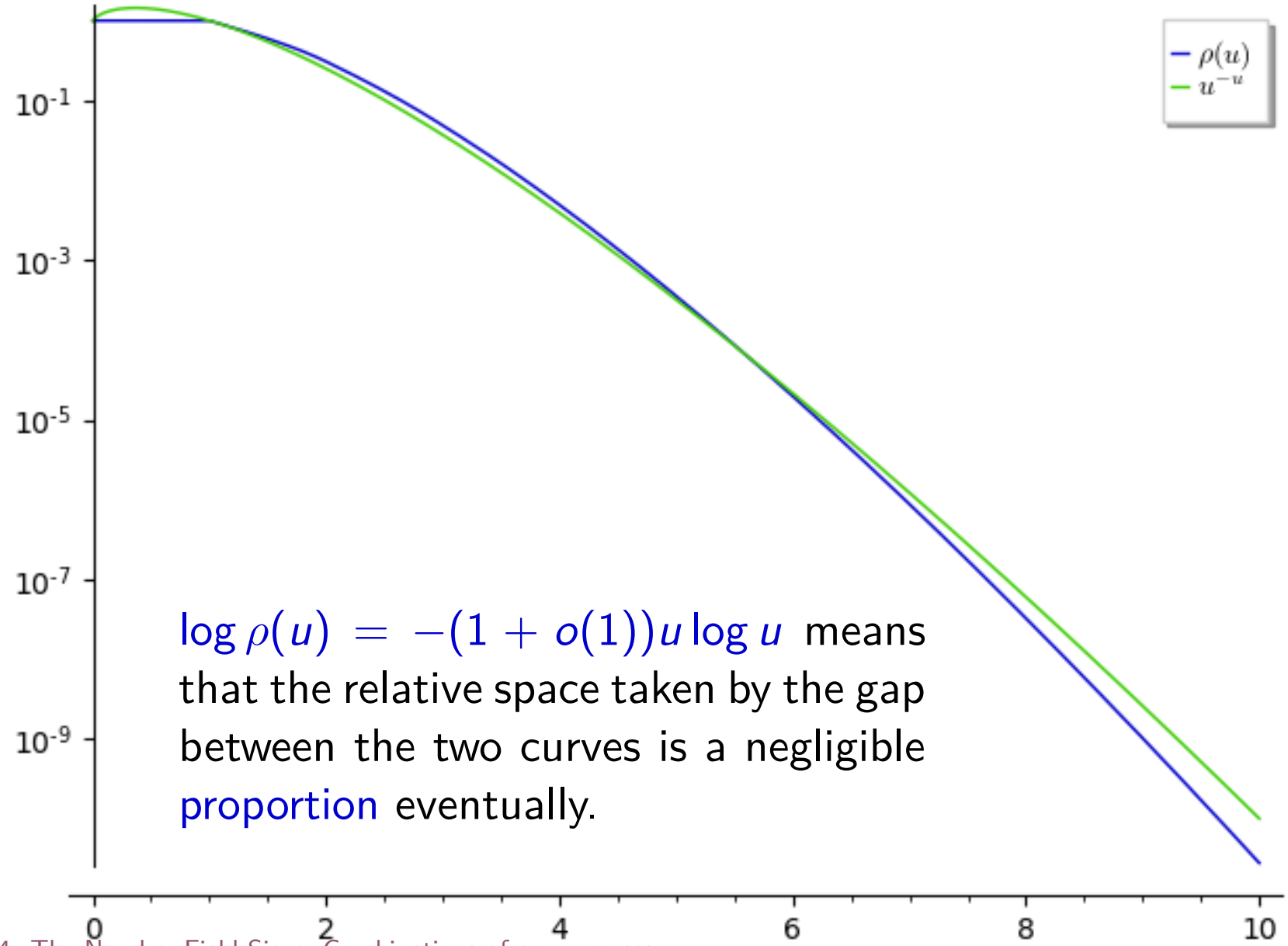
$$\rho(u) = u^{-u(1+o(1))}.$$

Note that $(1 + o(1))$ is quite inaccurate.

$\rho(u)$



$\rho(u)$ (log scale in y)



$\log \rho(u) = -(1 + o(1))u \log u$ means that the relative space taken by the gap between the two curves is a negligible proportion eventually.

The L function

The following notation is attributed to R. Schroeppel.

$$L_x[a, \alpha] = \exp\left(\alpha(\log x)^a(\log\log x)^{1-a}\right).$$

CEP with the L function

A random integer $n \leq L_x[a, \alpha]$ is $L_x[b, \beta]$-smooth with probability:

$$L_x\left[a - b, -\frac{\alpha}{\beta}(a - b)(1 + o(1))\right].$$

This formulation is very important for analyzing sieve algorithms.

Calculus with $L_x[a, \alpha] = \exp\left(\alpha(\log x)^a(\log\log x)^{1-a}\right).$

Basic formulae with L

$$L_x[a,\alpha] \times L_x[b,\beta] = \begin{cases} L_x[a, \alpha + o(1)] & \text{if } a > b, \\ L_x[b, \beta + o(1)] & \text{if } b > a, \\ L_x[a, \alpha + \beta] & \text{if } a = b. \end{cases}$$

$$L_x[a,\alpha] + L_x[b,\beta] = \begin{cases} L_x[a, \alpha + o(1)] & \text{if } a > b, \\ L_x[b, \beta + o(1)] & \text{if } b > a, \\ L_x[a, \max(\alpha, \beta)] & \text{if } a = b. \end{cases}$$

$$L_x[b,\beta]^{\log_{\log x} L_x[a,\alpha]} = L_x[a+b, \alpha\beta].$$

$$L_{L_x[b,\beta]}[a,\alpha] = L_x[ab, \alpha\beta^a b^{1-a} + o(1)].$$

Computation model

Analysis is done in the RAM model: memory access is for free.

Of course, given the leeway in the final asymptotic complexity estimate, this does not matter much.

Vanilla QS

Let $A = L_N[a, \alpha]$ be the bound on i.
Let $B = L_N[b, \beta]$ be the smoothness bound.

The number of collected relations is:

$$\underbrace{L_N[a, \alpha]}_{\text{exhaust } i} \times (\text{smoothness prob.}).$$

The cost of collecting relations (with detection by sieving) is:

$$L_N[a, \alpha] \times (\log\log B + (\text{smoothness prob.}) \times (\text{factoring time})).$$

The cost of linear algebra ($\pi(B)$ equations and unknowns) is:

$$(L_N[b, \beta + o(1)])^\omega = L_N[b, \omega\beta + o(1)].$$

Collecting relations

Assume $a < 1$. We have

$$\max_{-A \le i \le A} |f(i)| = 2A\sqrt{N} = L_N[1, 1/2 + o(1)].$$

Smoothness probability is $L_N[1-b, -\frac{1}{2\beta}(1-b)(1+o(1))]$.

We want enough relations:

$$L_N[a, \alpha] \times L_N[1-b, -\frac{1}{2\beta}(1-b)(1+o(1))] = L_N[b, \beta + o(1)].$$

In particular:

- we need $\beta > 0$, therefore we must have $a \ge 1 - b$.
- we cannot have $b < 1/2$: an LHS-RHS match would be impossible.

So let's try $a = b = 1/2$.

With $a = b = 1/2$

We want enough relations:

$$L_N[1/2, \alpha] \times L_N[1/2, -\frac{1}{4\beta}(1 + o(1))] = L_N[1/2, \beta + o(1)],$$
$$\alpha - \frac{1}{4\beta}(1 + o(1)) = \beta + o(1),$$
$$\alpha = (\beta + \frac{1}{4\beta})(1 + o(1)).$$

Note that this implies in particular $\alpha > \beta$.

Still some work to do

There are still a few undecided things.

- How much does it cost to factor the smooth values?
  Options: resieving, TD, ECM, Batch smoothness detection.
- How much does the linear algebra cost?
  Options: Gauss, or (later developed) sparse linear algebra.

TD $+ \omega \geq 2$

Assume we do TD for each smooth $f(i)$.

- Relation collection: $A \log\log B + \pi(B)^2$.
- Linear algebra $\pi(B)^\omega$.

Total cost $L_N[1/2, \max(\alpha, 2\beta, \omega\beta)(1 + o(1))]$. Minimize as follows:

$$\begin{aligned} \alpha &= \omega\beta, \\ \beta + 1/(4\beta) &= \omega\beta, \\ 4(\omega - 1)\beta^2 &= 1, \\ \beta &= \frac{1}{2\sqrt{\omega - 1}}. \end{aligned}$$

So that the complexity (with TD) is $L_N[1/2, \frac{\omega}{2\sqrt{\omega-1}}(1 + o(1))$.

Factoring does not matter

TD is not a very smart mechanism.

Yet, as long as we use sieving to detect the smooth values $f(i)$, what algorithm we use to actually factor them does not matter.

However, if even sieving were to be replaced by some exponential algorithm, the complexity would be different.

The complexity of QS is

$$L_N[1/2, \frac{\omega}{2\sqrt{\omega - 1}}(1 + o(1))].$$

If $\omega = 2$, this becomes $L_N[1/2, 1 + o(1)]$.

Impact of improvements

Many improvements were made to MPQS. Some are rather minor, and some had a major practical impact.

What is their impact on the complexity?

Take MPQS. $|f(i)|$ drops from $2A\sqrt{N}$ to $\frac{1}{\sqrt{2}}A\sqrt{N}$.

In both cases, this is $L_N[1, 1/2 + o(1)]$.

Impact of improvements

Many improvements were made to MPQS. Some are rather minor, and some had a major practical impact.

What is their impact on the complexity?

Take MPQS. $|f(i)|$ drops from $2A\sqrt{N}$ to $\frac{1}{\sqrt{2}}A\sqrt{N}$.

In both cases, this is $L_N[1, 1/2 + o(1)]$.

Disappointment

None of the practical improvements to QS (even MPQS) has the slightest impact on its asymptotic complexity.

Wrap-up

- Combination of congruences.
- QS successfully introduces sieving.
- Many improvements, some minor, some major.
  MPQS, special-Q.
- Massive distribution becomes possible (MPQS).
- Analysis yields $L_N[1/2, 1 + o(1)]$, but unfortunately most improvements are invisible.

CSE291-14: The Number Field Sieve

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

Emmanuel Thomé

January 13, 2022

Part 3a

NFS: using higher degree

Factoring with cubic integers

A rosy example

Can we go further?

MPQS is great. Can we do better?

Yes: NFS.

NFS is a compicated algorithm, and we will approach it from several angles.

- Earliest example: cubic integers.
- A simple example where everything goes well.
- We need some mathematical background.
- (next week) sometimes, things are more complicated.

Plan

Factoring with cubic integers

A rosy example

Factoring with cubic integers

John Pollard (who had invented the ρ and $p-1$ methods decades earlier) came up in 1988 with a nice idea to factor integers of a special form using cubic integers.

The Number Field Sieve (NFS) was born. Note that NFS is not an extension of QS. It is more related on an algorithm by Coppersmith, Odlyzko and Schroeppel (1986) called the linear sieve, to compute discrete logarithms. (not discussed here).

As it turns out, it took a few exciting years to go from Pollard's nifty idea to a full-fledged factoring algorithm.

Factoring with cubic integers

Pollard's method is well suited to numbers of a special form.

Target: $N = 2F_7 = 2(2^{128} + 1) = m^3 + 2$ for $m = 2^{43}$.

A mathematical object that is poised to take a key role is the number field $\mathbb{Q}(\sqrt[3]{-2})$.

A number field

A number field is field that contains $\mathbb{Q}$, and is defined by a defining polynomial with integer coefficients.

Analogy:

$\mathbb{Z}/p\mathbb{Z}$	$K = \mathbb{Q}[x]/f(x)$
quotient of $\mathbb{Z}$ by modulus p.	quotient of $\mathbb{Q}[x]$ by f.
• Work with integers.	• polynomials in $\mathbb{Q}[x]$.
• mod p when needed.	• mod f when needed.
• field $\Leftrightarrow$ p prime.	• field $\Leftrightarrow$ f irreducible.

Number fields are the main topic of algebraic number theory.

Basic operations work without surprises: $+$, $\times$, inversion with extended Euclidean algorithm (on polynomials).

Notation

In $K = \mathbb{Q}[x]/f(x)$, it is common to use a greek letter, say α, to denote $x \mod f(x)$.

- x is the indeterminate in the polynomial ring $\mathbb{Q}[x]$.
- α is an element of K.

By construction:

- α is a root of $f(x)$ in K.
- α is a generator of K: we have $K = \mathbb{Q}(\alpha)$.

At times, we may also write $\mathbb{Q}(\sqrt[3]{-2})$.

Number fields in software

Nowadays, readily available software can deal with number fields: SageMath, Magma, . . .

```
~ $ sage
+--------------------------------------------------------------------+
| SageMath version 9.4, Release Date: 2021-08-22                     |
| Using Python 3.9.5. Type "help()" for help.                        |
+--------------------------------------------------------------------+
sage: ZP.<x> = ZZ['x']
sage: K.<alpha> = NumberField(x^3+2)
sage: foo = 1 + alpha
sage: foo^2
alpha^2 + 2*alpha + 1
sage: foo^3
3*alpha^2 + 3*alpha - 1
sage: foo^17
-3160*alpha^2 - 44999*alpha - 51679
```

Common traits with $\mathbb{Q}$

$K = \mathbb{Q}[x]/f(x)$ shares many properties with $\mathbb{Q}$.

- ring of integers: the most $\mathbb{Z}$-like ring in K.
  - Usually noted $\mathcal{O}_K$ (my preferred one) or $\mathbb{Z}_K$.
  - The ring of integers of $\mathbb{Q}(\sqrt[3]{-2})$ is $\mathbb{Z}[\sqrt[3]{-2}]$.
  - Unfortunately it's not that easy in general.
- There is a notion that relates to prime numbers and unique factorization.

This is pretty handwavy, but it's enough for us at this point.

Note: a number field can be embedded into a subfield of $\mathbb{C}$.

Two paths to $\mathbb{Z}/N\mathbb{Z}$

We work in $K = \mathbb{Q}[x]/f(x)$, and assume that $f(m) \equiv 0 \mod N$.
Take $\phi(x) \in \mathbb{Z}[x]$. We map it to $\mathbb{Z}/N\mathbb{Z}$ in two ways.

- $\phi(m) \in \mathbb{Z}$, once reduced $\bmod N$, is in $\mathbb{Z}/N\mathbb{Z}$.
- $\phi(\alpha) \in K = \mathbb{Q}(\alpha)$.
  There is a ring morphism:

$$\begin{cases} \mathbb{Z}[\alpha] & \to \mathbb{Z}/N\mathbb{Z} \\ \alpha & \mapsto m. \end{cases}$$

Proof: if two polynomials in $\mathbb{Z}[x]$ differ by a multiple of f, their evaluations at m differ by a multiple of $f(m) \equiv 0 \mod N$.

These are two ways to reach $\phi(m) \bmod N \in \mathbb{Z}/N\mathbb{Z}$.

Example

$N = 2F_7 = 2(2^{128} + 1)$, $m = 2^{43}$, $f(x) = x^3 + 2$,
$K = \mathbb{Q}(\alpha) = \mathbb{Q}[x]/f(x)$.

```
sage: N=2^129+2; m=2^43
sage: a=56; b=89
sage: a-b*m
-782852278976456
sage: a-b*alpha
-89*alpha + 56
sage: Integers(N)(a-b*m)
680564733841876926926748432011257446458
sage: Integers(N)((a-b*alpha).polynomial()(m))
680564733841876926926748432011257446458
```

The diagram

$$
\begin{tikzcd}
 & \mathbb{Z}[x] \arrow[dl, "x \to m"'] \arrow[dr, "x \to \alpha"] & \\
\mathbb{Z}[m] \arrow[dr, "t \to t \bmod N"'] & & \mathbb{Z}[\alpha] \arrow[dl, "\alpha \to m \bmod N"] \\
 & \mathbb{Z}/N\mathbb{Z} &
\end{tikzcd}
$$

This diagram commutes.

We will come back to it later on.

Units

In $\mathbb{Z}$ some elements are invertible: ± 1.

In the ring of integers of a number field, some elements are invertible. These are called units.
There are many units in $\mathbb{Z}[\sqrt[3]{-2}]$.

```
sage: foo = 1 + alpha
sage: foo^-1
-alpha^2 + alpha - 1
sage: foo^17
-3160*alpha^2 - 44999*alpha - 51679
sage: foo^-17
-1861604361*alpha^2 + 2345474521*alpha - 2955112721
```

Find many ϕ

Let B be a smoothness bound.

Consider many polynomials $a - bx$ such that:

- The integer $a - bm$ is B-smooth.
- The element $a - b\alpha$ is smooth in $\mathbb{Q}(\alpha = \sqrt[3]{-2})$: it "factors" into "things" ("primes").

Then maybe we can do something with that.

```
sage: (a-b*m).factor()
-1 * 2^3 * 13 * 23 * 41 * 109 * 211 * 449 * 773
sage: (a-b*alpha).factor()
alpha * (-alpha + 1) * (alpha^2 - 6*alpha + 1) * (7*alpha + 3)
```

Combine many of these so as to get squares on both sides?

Factoring with number fields

TL;DR: It works.

However, in order to make it work, one needs to:

- Describe precisely the "primes" in $\mathbb{Z}[\alpha]$.
  And are we sure that it makes sense at all?
- Describe precisely the units in this ring.

This is entirely doable and we will do it, but we are first going to work with a simpler (made up) example.

Plan

Factoring with cubic integers

A rosy example

Create something absurdly easy

Our goal is to create an example that is even simpler than Pollard's example.

- We're not going to factor anything of computational interest.
- One of my goals is to have all relevant data fit on my slides.

Some number fields are simpler than others, so let's pick a very simple one:

- The degree of a number field is the degree of its definition polynomial. The field in Pollard's example has degree 3. Let's pick one of degree 2: a quadratic field.
- We want to keep control on units.

Units in quadratic fields

When it comes to units, quadratic fields are particularly easy.

In a quadratic field defined by a degree 2 polynomial $f(x) \in \mathbb{Z}[x]$ of discriminant Δ:

- if $\Delta > 1$ units are ± 1, and one unit of infinite order.
- if $\Delta < 0$, all units are of finite order.
  - in most cases, it's only ± 1.
  - special case $\Delta = -\mu^2$ has 4-th roots of unity.
  - special case $\Delta = -3\mu^2$ has 6-th roots of unity.

Quadratic fields are often classified as real quadratic fields and imaginary quadratic fields (they embed in $\mathbb{R}$ or $\mathbb{C}$).

A simple imaginary quadratic field

Let us pick $f(x) = x^2 - x + 3$.

Nice facts about $f(x)$

The number field K defined by f is an imaginary quadratic field.

- K is generated by $\alpha = \frac{1}{2}(1 + \sqrt{-11}))$, which is a root of f.
- The of integers $\mathcal{O}_K$ of K is $\mathbb{Z}[\alpha]$.
- There are no units in $\mathcal{O}_K$ beyond ± 1.
- $\mathcal{O}_K$ happens to be a unique factorization domain.
- Primes in $\mathcal{O}_K$:
  - Integer primes p are still prime in $\mathcal{O}_K$ if $\left(\frac{-11}{p}\right) = -1$.
  - Otherwise, p splits into two prime factors.

Primes in $\mathcal{O}_K$

The computer will tell us the following.

$2,$	$17,$
$3 = \alpha \times (1-\alpha),$	$19,$
$5 = (1+\alpha) \times (2-\alpha),$	$23 = (4+\alpha) \times (5-\alpha),$
$7,$	$29,$
$11 = -(1-2\alpha)^2,$	$31 = (4-3\alpha) \times (1+3\alpha),$
$13,$	$37 = (2+3\alpha) \times (5-3\alpha),$ $\quad \ldots$

It is possible to obtain this by hand, but somewhat tedious.

Can we factor a number?

Let us fix $N = 16259 = 16384 - 128 + 3$, and $m = 128$.

Given that $f(x) = x^2 - x + 3$, we have $f(m) \equiv 0 \mod N$.

We will do exactly as hinted at in the description of Pollard's algorithm.

- Enumerate many polynomials $\phi(x) = a - bx$.
- Look for those such that:
  - The integer $\phi(m)$ is smooth.
  - The element $\phi(\alpha)$ in K is smooth as well.

We fix a smoothness bound $B = 40$ (for factors of $f(m)$).

We will soon get to what this may mean on the number field side.

Relations

Try to factor values $f(m)$.

$1 - 1m = -127 = -127$,
$1 - 2m = -255 = -3 \times 5 \times 17$,
$1 - 3m = -383 = -383$,
$1 - 4m = -511 = -7 \times 73$,
$1 - 5m = -639 = -3^2 \times 71$,
$2 - 1m = -126 = -2 \times 3^2 \times 7$,
$2 - 3m = -382 = -2 \times 191$,
$2 - 5m = -638 = -2 \times 11 \times 29$,
$3 - 1m = -125 = -5^3$,
$3 - 2m = -253 = -11 \times 23$,
$3 - 4m = -509 = -509$,
$3 - 5m = -637 = -7^2 \times 13$,
$4 - 1m = -124 = -2^2 \times 31$,
$4 - 3m = -380 = -2^2 \times 5 \times 19$,
$4 - 5m = -636 = -2^2 \times 3 \times 53$,
$5 - 1m = -123 = -3 \times 41$,
$5 - 2m = -251 = -251$,
$5 - 3m = -379 = -379$,
$5 - 4m = -507 = -3 \times 13^2$,
$6 - 1m = -122 = -2 \times 61$,
$6 - 5m = -634 = -2 \times 317$,
$7 - 1m = -121 = -11^2$,
$7 - 2m = -249 = -3 \times 83$,
$7 - 3m = -377 = -13 \times 29$,
$7 - 4m = -505 = -5 \times 101$,
$7 - 5m = -633 = -3 \times 211$,
$8 - 1m = -120 = -2^3 \times 3 \times 5$,
$8 - 3m = -376 = -2^3 \times 47$,

Keep only the smooth ones!

Here are all the first few smooth $a - bm$ values for small a, b.

$1 - 2m = -255 = -3 \times 5 \times 17,$
$2 - 1m = -126 = -2 \times 3^2 \times 7,$
$2 - 5m = -638 = -2 \times 11 \times 29,$
$3 - 1m = -125 = -5^3,$
$3 - 2m = -253 = -11 \times 23,$
$3 - 5m = -637 = -7^2 \times 13,$
$4 - 1m = -124 = -2^2 \times 31,$
$4 - 3m = -380 = -2^2 \times 5 \times 19,$
$5 - 4m = -507 = -3 \times 13^2,$
$7 - 1m = -121 = -11^2,$
$7 - 3m = -377 = -13 \times 29,$
$8 - 1m = -120 = -2^3 \times 3 \times 5,$
$9 - 1m = -119 = -7 \times 17,$
$9 - 2m = -247 = -13 \times 19,$
$10 - 3m = -374 = -2 \times 11 \times 17,$
$11 - 1m = -117 = -3^2 \times 13,$
$11 - 2m = -245 = -5 \times 7^2,$
$11 - 5m = -629 = -17 \times 37,$
$12 - 1m = -116 = -2^2 \times 29,$
$13 - 1m = -115 = -5 \times 23,$
$13 - 2m = -243 = -3^5,$
$13 - 5m = -627 = -3 \times 11 \times 19,$
$14 - 1m = -114 = -2 \times 3 \times 19,$
$14 - 3m = -370 = -2 \times 5 \times 37,$
$16 - 1m = -112 = -2^4 \times 7,$
$16 - 3m = -368 = -2^4 \times 23,$
$16 - 5m = -624 = -2^4 \times 3 \times 13,$
$17 - 1m = -111 = -3 \times 37,$

Number field side

Same deal.
I haven't said how we can factor in K yet.

$$
\begin{aligned}
1-\alpha &= (1-\alpha),\\
1-2\alpha &= (1-2\alpha),\\
1-3\alpha &= (2-\alpha)^2,\\
1-4\alpha &= (1-\alpha)^2 \times (1+\alpha),\\
1-5\alpha &= (1-5\alpha),\\
2-\alpha &= (2-\alpha),\\
2-3\alpha &= -(1+\alpha)^2,\\
2-5\alpha &= (1-\alpha) \times (5-\alpha),\\
3-\alpha &= -(\alpha)^2,
\end{aligned}
$$

$$
\begin{aligned}
3-2\alpha &= -(\alpha) \times (1+\alpha),\\
3-4\alpha &= -(\alpha)^2 \times (2-\alpha),\\
3-5\alpha &= -(\alpha) \times (4+\alpha),\\
4-\alpha &= (1-\alpha) \times (1+\alpha),\\
4-3\alpha &= (4-3\alpha),\\
4-5\alpha &= (4-5\alpha),\\
5-\alpha &= (5-\alpha),\\
5-2\alpha &= -(1-\alpha)^3,\\
5-3\alpha &= (5-3\alpha),
\end{aligned}
$$

Side note: there are no "integer primes" in these factorizations.
There is a reason for that.

When are both sides smooth?

$$
\begin{aligned}
1+3m &= 5\times 7\times 11 & 1+3\alpha &= (3\alpha+1),\\
1-2m &= -3\times 5\times 17 & 1-2\alpha &= -(2\alpha-1),\\
2+1m &= 2\times 5\times 13 & 2+1\alpha &= -(-\alpha+1)^2,\\
2-1m &= -2\times 3^2\times 7 & 2-1\alpha &= (-\alpha+2),\\
2-5m &= -2\times 11\times 29 & 2-5\alpha &= (-\alpha+1)\times(-\alpha+5),\\
3+2m &= 7\times 37 & 3+2\alpha &= -(\alpha)^3,\\
3-1m &= -5^3 & 3-1\alpha &= -(\alpha)^2,\\
3-2m &= -11\times 23 & 3-2\alpha &= -(\alpha)\times(\alpha+1),\\
3-5m &= -7^2\times 13 & 3-5\alpha &= -(\alpha)\times(\alpha+4),\\
4+5m &= 2^2\times 7\times 23 & 4+5\alpha &= -(-\alpha+1)\times(-3\alpha+5),\\
4+1m &= 2^2\times 3\times 11 & 4+1\alpha &= (\alpha+4),\\
4-1m &= -2^2\times 31 & 4-1\alpha &= (-\alpha+1)\times(\alpha+1),\\
4-3m &= -2^2\times 5\times 19 & 4-3\alpha &= (-3\alpha+4),\\
5+1m &= 7\times 19 & 5+1\alpha &= (-\alpha+1)\times(2\alpha-1),\\
7-1m &= -11^2 & 7-1\alpha &= -(-\alpha+1)^2\times(-\alpha+2),
\end{aligned}
$$

$$\ldots$$

Put these in a matrix

	-1	2	3	5	7	11	13	17	19	23	29	31	37	-1	$1-\alpha$	α	$2-\alpha$	$1+\alpha$	$2\alpha-1$	$\alpha-5$	$\alpha+4$	$-3\alpha+4$	$3\alpha+1$	$-3\alpha-2$	$3\alpha-5$
$(1,-3)$				1	1	1																	1		
$(1,2)$	1		1	1				1						1					1						
$(2,-1)$		1		1			1							1	2										
$(2,1)$	1	1	2		1												1								
$(2,5)$	1	1				1					1				1					1					
$(3,-2)$					1								1	1		3									
$(3,1)$	1			3										1		2									
$(3,2)$	1					1				1				1		1		1							
$(3,5)$	1				2		1							1		1					1				
$(4,-5)$		2			1					1				1	1										1
$(4,-1)$		2	1			1															1				
$(4,1)$	1	2										1			1			1							
$(4,3)$	1	2		1					1													1			
$(5,-1)$					1				1						1				1						
$(7,1)$	1					2								1	2		1								

(a few more rows below!)

Put these in a matrix

	-1	2	3	5	7	11	13	17	19	23	29	31	37	-1	$1-\alpha$	α	$2-\alpha$	$1+\alpha$	$2\alpha-1$	$\alpha-5$	$\alpha+4$	$-3\alpha+4$	$3\alpha+1$	$-3\alpha-2$	$3\alpha-5$
$(1,-3)$				1	1	1																	1		
$(1,2)$	1		1	1				1						1					1						
$(2,-1)$		1		1			1							1											
$(2,1)$	1	1			1												1								
$(2,5)$	1	1				1					1				1					1					
$(3,-2)$					1								1	1		1									
$(3,1)$	1			1										1											
$(3,2)$	1					1				1				1		1		1							
$(3,5)$	1						1							1		1					1				
$(4,-5)$					1					1				1	1										1
$(4,-1)$			1			1															1				
$(4,1)$	1											1			1			1							
$(4,3)$	1			1					1													1			
$(5,-1)$					1				1						1				1						
$(7,1)$	1													1			1								

(a few more rows below!)

Linear algebra

We must find a nullspace element.

This will guarantee an even valuation for all primes that appear, and also an even number of -1's (on both sides).

Here is what the knowledge of a nullspace element tells us:

$$\begin{aligned} R(x) = {} & (2x+3) \times (-3x+7) \times (x+8) \times (-2x+9) \\ & \times (-x+14) \times (-x+16) \times (-x+17) \times (-4x+19). \end{aligned}$$

This gives:

$$\begin{aligned} R(m) &= 2^8 \times 3^2 \times 7^2 \times 13^2 \times 17^2 \times 19^2 \times 29^2 \times 37^2, \\ R(\alpha) &= (\alpha)^4 \times (-\alpha+1)^8 \times (\alpha+1)^2 \times (-\alpha+2)^6 \\ & \quad \times (2\alpha-1)^2 \times (-3\alpha+5)^2. \end{aligned}$$

Done!

At this point we are pretty much done.

$$\begin{aligned}
\sqrt{R(128)} &= 2^4 \times 3 \times 7 \times 13 \times 17 \times 19 \times 29 \times 37 \\
&= 1513857072 \equiv 14100 \mod N. \\
\sqrt{R(\alpha)} &= (\alpha)^2 \times (-\alpha+1)^4 \times (\alpha+1) \times (-\alpha+2)^3 \\
&\times (2\alpha-1) \times (-3\alpha+5), \\
\sqrt{R(\alpha)} &= -3735\alpha + 13995. \\
\sqrt{R(\alpha)} &\mapsto -464085 \equiv 7426 \mod N.
\end{aligned}$$

And

$$\gcd(14100 - 7426, 16259) = 71.$$

Yes, this is all cheating

Some hurdles were deliberately sidestepped in the previous example.

- No real use case for number fields of degree 2.
- The ring of integers (which we haven't properly defined) is rarely as simple as $\mathbb{Z}[\alpha]$.
- Units are never as simple as $\{\pm 1\}$.
- We don't even have unique factorization in general! However, we do have something interesting with ideals in the ring of integers $\mathcal{O}_K$.

Next: algebraic number theory background.

CSE291-14: The Number Field Sieve

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

Emmanuel Thomé

January 18, 2022

Part 3b

Algebraic Number Theory background

Number fields, algebraic numbers

Algebraic integers, ring of integers

Ideals

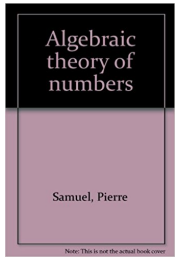
Factoring into prime ideals

Units and the class group

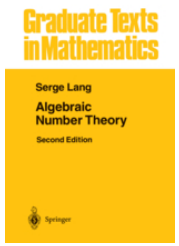
Textbooks

Numerous textbooks available on algebraic number theory.

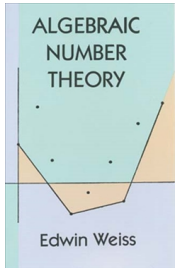
- A good read:
  P. Samuel, Algebraic theory of numbers, Hermann, 1970 (multiple editions).



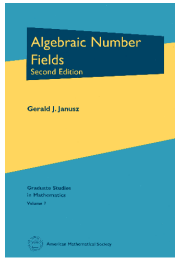
- Not advisable for a first read:
  S. Lang, Algebraic Number Theory, Springer, GTM 110, 1994.



- E. Weiss, Algebraic Number Theory, Dover, Mc-Graw Hill, 1963 (multiple editions).



- G. Janusz, algebraic number fields, AMS, GSM 7, 1996.



Plan

Number fields, algebraic numbers

Algebraic integers, ring of integers

Ideals

Factoring into prime ideals

Units and the class group

Goals

Our goals here:

- define the basic vocabulary: algebraic numbers, number fields.
- give a few examples.
- introduce the very few bits of Galois theory that we need in order to define the norm of an element.

Note: we deliberately don't give proofs. Those can be found in textbooks.

Algebraic numbers

Def. Let $K \subset L$ be two fields. "$x \in L$ is algebraic over K" means:

$$\exists P \in K[X], \quad P(x) = 0.$$

- if all $x \in L$ are algebraic, L/K is an algebraic extension ;
- a finite extension is algebraic ;
- an algebraic extension is not necessarily finite ($\bar{\mathbb{Q}}$).
- Common terminology:
  - Algebraic number = something algebraic over (a finite extension of) $\mathbb{Q}$.
  - Number field = a finite algebraic extension of $\mathbb{Q}$.

Roots of the defining polynomial

Let f be irreducible over $\mathbb{Q}$.

By construction, f has a root in $K = \mathbb{Q}[x]/f$.

Where do the other roots of f lie ?

- In some cases, they are also in K. Some examples:
  - If f has degree 2,
  - If f is a cyclotomic polynomial (e.g. $x^4 + 1 = \Phi_8$).
- Most often they are not. Most typical example: $\mathbb{Q}(\sqrt[3]{2})$.

It is sometimes convenient to think of the roots of f in an algebraic closure of K. For example in $\mathbb{C}$.

This links to the Galois group.

Example

```
sage: K.<h> = NumberField(x^4+1)
sage: h.minpoly()
x^4 + 1
sage: h.minpoly().roots(K)
[(h, 1), (-h, 1), (h^3, 1), (-h^3, 1)]
sage: h.minpoly().change_ring(K).factor()
(x - h) * (x + h) * (x - h^3) * (x + h^3)
```

Example

```
sage: K.<alpha> = NumberField(x^3-2)
sage: alpha.minpoly()
x^3 - 2
sage: alpha.minpoly().roots(K)
[(alpha, 1)]
sage: alpha.minpoly().change_ring(K).factor()
(x - alpha) * (x^2 + alpha*x + alpha^2)
```

On top of K, the field where the other roots of f live is an extension of degree 2.

Splitting field

Let f be irreducible over $\mathbb{Q}$.

- $K = \mathbb{Q}(\alpha) = \mathbb{Q}[x]/f$ brings one root to f.
  - there may be more.
  - But α may also be the only root: f may factor in K as

$$f = (x - \alpha) \times (\text{irreducible factor of degree } n - 1).$$

- We may then build another extension, of degree at most $n - 1$.
- And so on and so forth.

The splitting field (normal closure) of f has degree at most $n!$.
This is what happens generically, for f having no magical property.

Galois groups

Normal extension

A field extension L/K is normal if and only if, given $g \in K[x]$ irreducible:

$$g \text{ has a root in } L \Leftrightarrow g \text{ splits completely.}$$

Def. Normal+Separable=Galois (see textbooks, e.g. Stewart).
In the NFS world, we're always separable.

$\mathrm{Gal}(L/K)$: group of automorphisms of L leaving K fixed.

In the NFS context, L is never computed, and we are not really interested in $\mathrm{Gal}(L/\mathbb{Q})$ either. However:

- $\mathrm{Gal}(L/\mathbb{Q})$ is the Galois-related thing which is a group.
- We are interested in its action on K.

Galois group (2)

$$
\begin{tikzcd}
 & L, \text{ normal closure of } K \arrow[d, dash, bend left, "G^K = \mathrm{Gal}(L/K)"] \arrow[dd, dash, bend right=60, "G = \mathrm{Gal}(L/\mathbb{Q})"'] \\
 & K \arrow[d, dash, bend left, "G/G^K"] \\
 & \mathbb{Q}
\end{tikzcd}
$$

- When we speak of "the Galois group of f", or of K, we're implying G.
- But G can be partitioned into cosets, each acting in a unique way on K (elements of G do not leave K fixed!).

A "random" polynomial of degree n has Galois group $\mathfrak{S}_n$.

Embeddings into $\mathbb{C}$

Take for example $K = \mathbb{Q}(\alpha) = \mathbb{Q}[x]/x^3 - 2$. We have three embeddings of K into $\mathbb{C}$.

$$\phi_1 : \left\{ \begin{array}{ccc} K & \to & \mathbb{C}, \\ \alpha & \mapsto & \sqrt[3]{2}, \end{array} \right. \quad \phi_2 : \left\{ \begin{array}{ccc} K & \to & \mathbb{C}, \\ \alpha & \mapsto & j\sqrt[3]{2}, \end{array} \right. \quad \phi_3 : \left\{ \begin{array}{ccc} K & \to & \mathbb{C}, \\ \alpha & \mapsto & j^2\sqrt[3]{2}. \end{array} \right.$$

The Galois group of $x^3 + 2$ is $\mathfrak{S}_3$, of order 6.

Given $K = \mathbb{Q}(\alpha)$, the set of roots in a splitting field is:
$(\alpha_1, \ldots, \alpha_n) = (\alpha^\sigma)_{\sigma \in G/G_K}$. (notation: $\alpha^\sigma = \sigma(\alpha)$)

The Galois group thus controls the various existing embeddings into $\mathbb{C}$.

Norm, trace, etc

Symmetric functions of the roots are defined over $\mathbb{Q}$ (because by Galois theory, they are fixed by G).

Two important examples. Let $\zeta \in K$.

$$\mathrm{Tr}_{K/\mathbb{Q}}(\zeta) = \sum_{\sigma \in G/G_K} \zeta^\sigma,$$
$$\mathrm{Norm}_{K/\mathbb{Q}}(\zeta) = \prod_{\sigma \in G/G_K} \zeta^\sigma.$$

In particular the norm can be turned into something very algorithmic, computable, and useful.

Computing the norm

Let $A(\alpha) = \sum_i a_i \alpha^i$ denote an element of K.

- A denotes a polynomial with coefficients in $\mathbb{Q}$.
- The Galois conjugates are $A(\alpha)^\sigma = A(\alpha^\sigma)$.
- But note also that $\{\alpha^\sigma\}_{\sigma \in G/G_K}$ are exactly the roots of f.

Thus the computation of the norm is achieved by the Resultant of f and A.

The resultant is the product of the evaluations of a polynomial at all the roots of another.

- it is an eminently computable thing!
  Only arithmetic in the coefficient ring is needed.
- and we will deal with simple cases only.

The norm and the resultant

Definition of $\mathrm{Res}(u(x), v(x))$

$$\mathrm{Res}(u(x), v(x)) = \mathrm{lc}(u)^{\deg v} \prod_{u(\mu)=0} v(\mu) = \mathrm{lc}(v)^{\deg u} \prod_{v(\nu)=0} u(\nu),$$

$$= \text{(also) determinant of the Sylvester matrix.}$$

Repeat: the roots of f are $\{\alpha^\sigma\}_{\sigma \in G/G_K}$.
IOW: $f = \mathrm{lc}(f) \prod_{\sigma \in G/G_K} (x - \alpha^\sigma)$
Therefore

$$\mathrm{Norm}_{K/\mathbb{Q}}(A(\alpha)) = \prod_{\sigma \in G/G_K} A(\alpha^\sigma) = \prod_{r \in \text{roots of } f} A(r)$$

$$= (1/f_n)^{\deg A} \mathrm{Res}(f, A).$$

Notice that we do not need to compute L or $\mathrm{Gal}(L/K)$.

Common case in NFS

In the NFS context, we often consider algebraic numbers like $a - b\alpha$. Their norm can be computed easily.

$$\begin{aligned}\mathrm{Norm}_{K/\mathbb{Q}}(a - b\alpha) &= \frac{1}{f_n}\,\mathrm{Res}(f, a - bx) = \frac{b^n}{f_n} f(\frac{a}{b}),\\ &= \frac{1}{f_n}\left(f_n a^n + f_{n-1} a^{n-1} b + \cdots + f_0 b^n\right).\end{aligned}$$

If one introduces the homogeneous polynomial

$$F(X, Y) = Y^n f(X/Y) = f_n X^n + f_{n-1} X^{n-1} Y + \cdots + f_0 Y^n,$$

then $\mathrm{Norm}_{K/\mathbb{Q}}(a - b\alpha) = \frac{1}{f_n} F(a, b)$.

Note: F is more than a computational hack. It means something.

Working in K

More generally, one may compute in number fields using polynomials in a generating element.

Trace, norm, etc of an element ζ correspond to trace, determinant of the multiplication-by-ζ matrix in any basis. We even have:

Definition: Characteristic polynomial of an algebraic number

The char. poly. of an algebraic number ζ is the char. poly. of the multiplication-by-ζ matrix in any basis.

Definition: Minimal polynomial of an algebraic number

The minimal polynomial of an algebraic number ζ is the min. poly. of the multiplication-by-ζ matrix in any basis.

Software

Software for working with number fields:

- Pari/gp (GPL). Most advanced. Interface is very bad.
- Sage. Includes pari, but lots of glue code missing.
- Magma. Includes a severely outdated version of pari. But interface is very complete. Good enough for our purposes.

Keep in mind: norm, resultant, Galois group

- The norm of any algebraic number can be computed.
- It is obviously a multiplicative thing.
- To compute it, the Resultant can be used.
- $\mathrm{Norm}(a - b\alpha) = \frac{1}{f_n}\,\mathrm{Res}(a - bx, f) = \frac{1}{f_n} F(a, b)$.
- The Galois group dwells somewhere around. It's often the full symmetric group. We don't have to bother much with it, except maybe know that it exists.
- All of this is readily available in computer software.

Plan

Number fields, algebraic numbers

Algebraic integers, ring of integers

Ideals

Factoring into prime ideals

Units and the class group

Goals

Goal here:

- Give a proper definition of the ring of integers of a number field.

Integrality

Definition: integral element

Let $A \subset B$ be two rings. "$x \in L$ is integral over A" means:

$$\exists P \in A[X], \quad P \text{ monic and } P(x) = 0.$$

Prop. $x \in L$ is integral over A iff $\exists M$ f.g. A-module with $xM \subset M$.

Def. Elements of B which are integral over A form the integral closure of A in B (which is an A-algebra).

Def. A ring is integrally closed if it is its own integral closure in its field of fractions.

Examples:
- $\mathbb{Z}$ is integrally closed.
- An integral closure is integrally closed.

Algebraic integers

In the number field case:

Definition: algebraic integer

Let K be a number field. An algebraic number $\zeta \in K$ is an algebraic integer iff it is integral over $\mathbb{Z}$.

Criterion: an algebraic number is integral iff its characteristic polynomial has coefficients in $\mathbb{Z}$.

Example

```
sage: K.<z>=NumberField(x^2+11)
sage: z.charpoly()
x^2 + 11
sage: ((z+1)/2).charpoly()
x^2 - x + 3
```

Sometimes, there are surprising algebraic integers!

Ring of integers

Definition: ring of integers

Def. Let $K/\mathbb{Q}$ be a number field. The ring of integers $\mathcal{O}_K$ of K is the integral closure of $\mathbb{Z}$ in K.

Prop. $\mathcal{O}_K$ is a finitely generated torsion-free $\mathbb{Z}$-module.

- Finitely generated: there is a basis over $\mathbb{Z}$.
- Torsion-free: there is no way to multiply something by an integer and get zero.

Ring of integers

Properties we expect and appreciate:

- all algebraic integers are in the ring of integers.
- the ring of integers is a ring.

$\mathcal{O}_K$ is the most reasonable $\mathbb{Z}$-like ring to work with within K.

Unfortunately, computing $\mathcal{O}_K$ is difficult.

Example

```
sage: K.<alpha>=NumberField(x^3+7)
sage: OK=K.ring_of_integers()
sage: OK.basis()
[1, alpha, alpha^2]
sage: K.<alpha>=NumberField(x^4 - 2*x^3 - 2*x^2 - 2*x + 1)
sage: OK=K.ring_of_integers()
sage: OK.basis()
[1/2*alpha^2 + 1/2, 1/2*alpha^3 + 1/2*alpha, alpha^2, alpha^3]
```

Examples of algebraic integers

Textbook case: $f \in \mathbb{Z}[x]$ monic and irreducible.

Let $K = \mathbb{Q}(\alpha) = \mathbb{Q}[x]/f$.

- Then α is an algebraic integer.
- So are all $a - b\alpha$ with $a, b \in \mathbb{Z}$,
- or $A(\alpha)$ with $A \in \mathbb{Z}[x]$. But $\mathcal{O}_K$ may be larger than $\mathbb{Z}[\alpha]$!

Real-life case: f not monic

Say $f = f_n x^n + \cdots$. Let $\hat{\alpha} = f_n \alpha$. We have:

$$\begin{aligned} 0 = f_n^{n-1} f(\alpha) &= f_n^n \alpha^n + f_n^{n-1} f_{n-1} \alpha^{n-1} + \cdots + f_n^{n-1} f_0, \\ &= \hat{\alpha}^n + f_{n-1} \hat{\alpha}^{n-1} + f_n f_{n-2} \hat{\alpha}^{n-2} + \cdots + f_n^{n-1} f_0. \end{aligned}$$

So $\hat{\alpha}$ is an algebraic integer. But $\mathcal{O}_K$ may be larger than $\mathbb{Z}[\hat{\alpha}]$!

Integral basis

We can always fabricate subrings of $\mathcal{O}_K$ of the form $\mathbb{Z}[\alpha]$.

But in general $\mathcal{O}_K$ needs not be of that form.

Which best form can we expect in full generality ?

- $\mathcal{O}_K$ can be written as: $\mathcal{O}_K = \mathbb{Z}\omega_1 + \cdots \mathbb{Z}\omega_n$,
- where ω_i are algebraic integers of the form $\frac{1}{d}A_i(\alpha)$ for some common denominator d (hard task: find the ω_i).
- $(\omega_i)_i$ is a $\mathbb{Q}$-basis of K.
- The matrix whose rows are coefficients of A_i may be put into Hermite normal form. Internally this is what is done in software.

Keep in mind

- The ring of integers $\mathcal{O}_K$ is cool.
- The minimal polynomials of its elements are in $\mathbb{Z}[x]$ and monic.
- $\mathcal{O}_K$ is a ring, with a basis.
- It is unfortunately rarely as simple as $\mathbb{Z}[\alpha]$.
- When we start from a non-monic definition polynomial, its root is not an algebraic integer, and $\mathbb{Z}[f_n\alpha]$ is typically much smaller than $\mathcal{O}_K$.

Further topic: orders

Orders (= certain types of subrings) in number fields are useful. These must be introduced in order to explain how to compute $\mathcal{O}_K$.

A picture

$$\begin{array}{ccc} K & & \\ | & & \\ \mathbb{Q} & \supset & \mathbb{Z} \end{array}$$

We are chiefly interested in:

A picture

$$\begin{array}{ccc} K & \supset & \mathcal{O}_K \\ \Big| & & \Big| \\ \mathbb{Q} & \supset & \mathbb{Z} \end{array} \quad \text{integral closure}$$

We are chiefly interested in:

- The ring of integers $\mathcal{O}_K$, as a first-class citizen in this big picture. Not necessarily that we must compute it.

A picture

$$
\begin{array}{ccccccc}
K & \supset & \mathcal{O}_K & \supset & \mathfrak{p}_1 \cdots \mathfrak{p}_m \\
| & & | & & \diagdown \,|\, \diagup \\
\mathbb{Q} & \supset & \mathbb{Z} & \supset & p\mathbb{Z}
\end{array}
$$

We are chiefly interested in:

- The ring of integers $\mathcal{O}_K$, as a first-class citizen in this big picture. Not necessarily that we must compute it.
- The decomposition (factorization) of prime (ideals) of $\mathbb{Z}$ in $\mathcal{O}_K$.

A picture

$$
\begin{array}{ccccccc}
K & \supset & \mathcal{O}_K & \supset & \mathfrak{p}_1 \cdots \mathfrak{p}_m & & \mathbb{F}_{p^{k_1}} \cdots \mathbb{F}_{p^{k_m}} \\
| & & | & & \diagdown \,|\, \diagup & & \diagdown \,|\, \diagup \\
\mathbb{Q} & \supset & \mathbb{Z} & \supset & p\mathbb{Z} & & \mathbb{F}_p
\end{array}
$$

We are chiefly interested in:

- The ring of integers $\mathcal{O}_K$, as a first-class citizen in this big picture. Not necessarily that we must compute it.
- The decomposition (factorization) of prime (ideals) of $\mathbb{Z}$ in $\mathcal{O}_K$, and the residue fields.

A picture

$$\begin{array}{ccccc} K & \supset & \mathcal{O}_K & \supset & \mathcal{O}_K^*, \text{ the unit group} \\ | & & | & & | \\ \mathbb{Q} & \supset & \mathbb{Z} & \supset & \{\pm 1\} \end{array}$$

We are chiefly interested in:

- The ring of integers $\mathcal{O}_K$, as a first-class citizen in this big picture. Not necessarily that we must compute it.
- The decomposition (factorization) of prime (ideals) of $\mathbb{Z}$ in $\mathcal{O}_K$, and the residue fields.
- Other multiplicative structure, e.g. units.

Plan

Number fields, algebraic numbers

Algebraic integers, ring of integers

Ideals

Factoring into prime ideals

Units and the class group

Goals

Our goals here:

- define ideals, operations on ideals, and some vocabulary.
- give a few examples.
- show how it can work algorithmically.

Primes ?

The ring of integers is nice, but lacks one thing: unique factorization.

Example: in $\mathbb{Q}(\sqrt{-5})$, one has $6 = 2 \times 3 = (1+\sqrt{-5})(1-\sqrt{-5})$, and all "look prime".

However, $\mathcal{O}_K$-ideals do enjoy unique factorization.

Here

$$6\mathcal{O}_K = \left\langle 2, 1+\sqrt{-5} \right\rangle^2 \times \left\langle 3, 1+\sqrt{-5} \right\rangle \times \left\langle 3, 1-\sqrt{-5} \right\rangle,$$
$$\left\langle 1+\sqrt{-5} \right\rangle = \left\langle 2, 1+\sqrt{-5} \right\rangle \times \left\langle 3, 1+\sqrt{-5} \right\rangle,$$
$$\left\langle 1-\sqrt{-5} \right\rangle = \left\langle 2, 1+\sqrt{-5} \right\rangle \times \left\langle 3, 1-\sqrt{-5} \right\rangle.$$

Ideals in $\mathcal{O}_K$

Ideals are very important objects in number fields.

Definition

An ideal I of $\mathcal{O}_K$ is such that:

- I forms an additive group.
- I is stable under multiplication by elements of $\mathcal{O}_K$.

An ideal may be specified by giving a set of generators.

Notation

All sets below are $\mathcal{O}_K$-ideals by construction.

$$\langle x \rangle = x\mathcal{O}_K = \{xa,\ a \in \mathcal{O}_K\}.$$
$$\langle x, y \rangle = \{xa + yb,\ a, b \in \mathcal{O}_K\}.$$
$$\langle x_1, \ldots, x_k \rangle = \{\sum_i x_i a_i,\ a_i \in \mathcal{O}_K\}.$$

Ideals

We can add ideals:

$$I + J = \{\text{ideal generated by sums of elements of } I \text{ and } J\}.$$

We can multiply ideals:

$$I \times J = \{\text{ideal generated by products of elements of } I \text{ and } J\}.$$

We can intersect ideals: $I \cap J =$ set-wise intersection, really!

Note that since an ideal is made of elements of $\mathcal{O}_K$, we have:

- $I \times J \subset I \times \mathcal{O}_K = I$: «to contain is to divide».
- $I \cap J$ really works as the lcm of ideals.
- $I + J$ contains I and J: this is a gcd.
  Ideals such that $I + J = \mathcal{O}_K$ are coprime.
  E.g. two ideals that contain coprime integers are coprime.

Ideals

Definition: prime ideals

An ideal I is prime if $ab \in I$ implies $a \in I$ or $b \in I$.
Fact: if I is prime, then $\mathcal{O}_K/I$ is an integral domain.

Definition: maximal ideals

An ideal I is maximal if it is maximal for inclusion (nobody between I and $\mathcal{O}_K$).
Fact: if I is prime, then $\mathcal{O}_K/I$ is a field.

Fact: in a number field, all prime ideals are maximal. So these two concepts are identical as far as we are concerned.

Fractional ideals

Ideals in $\mathcal{O}_K$ form a multiplicative semigroup. Extension desired !

Def $I \subset K$ is a fractional ideal (of $\mathcal{O}$), or a (fractional) $\mathcal{O}$-ideal iff I is a non-zero $\mathcal{O}$-module and $\exists a \in \mathcal{O},\ aI \subset \mathcal{O}$.

Terminology:
- Integral ideal: ideal of $\mathcal{O}$.
- Fractional ideal: more general.

Informally: fractional ideal = ideal with denominator.

Definition of ideal division

$$I^{-1} = \{a \in K, aI \subset \mathcal{O}_K\}.$$

Fractional ideals

Fantastic properties of $\mathcal{O}_K$

$\mathcal{O}_K$ is a Dedekind domain (integrally closed, Noetherian, all prime ideals maximal).
This implies that the fractional $\mathcal{O}_K$-ideals form a group with unique factorization.

Representing ideals

Note: $\mathcal{O}_K$ is not in general a principal ideal domain.

- Ideals can be represented by a set of generators.
  Two are always enough.
- Fractional ideals: integer denominator, + generators.
- Principal ideals: one generator is possible, but often not worthwhile (or too large)

Algorithmically, it is sometimes useful to represent ideals more generally as $\mathbb{Z}$-modules within K, with generators in HNF form.

(HNF = Hermite Normal Form = like Gauss, but on integer matrices)

Example

```
sage: K.<alpha>=NumberField(x^3+7)
sage: OK=K.ring_of_integers()
sage: [K(c) for c in OK.basis()]
[1, alpha, alpha^2]
sage: OK.ideal(11).factor()
(Fractional ideal (11, alpha^2 + 5*alpha + 3))
 * (Fractional ideal (11, alpha - 5))
sage: I11a=OK.ideal(11).factor()[0][0]
sage: I11b=OK.ideal(11).factor()[1][0]
sage: I11a.basis()
[11, 11*alpha, alpha^2 + 5*alpha + 3]
sage: I11b.basis()
[11, alpha + 6, alpha^2 + 8]
sage: OK.ideal(29).factor()
(Fractional ideal (-2*alpha^2 + 3*alpha + 10))
 * (Fractional ideal (-alpha^2 + 2*alpha - 2))
```

HNF means algorithms

```
sage: L=[u*v for u in I11a.basis() for v in I11b.basis()]
sage: L
[121,
 11*alpha + 66,
 11*alpha^2 + 88,
 121*alpha,
 11*alpha^2 + 66*alpha,
 88*alpha - 77,
 11*alpha^2 + 55*alpha + 33,
 11*alpha^2 + 33*alpha + 11,
 11*alpha^2 + 33*alpha - 11]
sage: m=matrix(ZZ,[uv.vector() for uv in L])
sage: m1=m.hermite_form(include_zero_rows=False)
sage: m1
[11  0  0]
[ 0 11  0]
[ 0  0 11]
sage: ideal([OK(v) for v in m1.rows()])
Fractional ideal (11)
```

Plan

Number fields, algebraic numbers

Algebraic integers, ring of integers

Ideals

Factoring into prime ideals

Units and the class group

Ideals above ideals

For I an $\mathcal{O}_K$-ideal, $I \cap \mathbb{Z}$ is a $\mathbb{Z}$-ideal.
$I \cap \mathbb{Z} = p\mathbb{Z} \Leftrightarrow$ "I lies above p".

What are the prime ideals that lie above p.

Surely, $\langle p \rangle = p\mathcal{O}_K$ is one such ideal, but are there ideals that contain (divide) $\langle p \rangle = p\mathcal{O}_K$?

Obvious mathematical breakthrough

We are attempting to factor the prime number p in the number field K.

Number fields must be Bill Gates' delight!

> *The obvious mathematical breakthrough would be development of an easy way to factor large prime numbers.*

Norm of ideals

The quotient ring $\mathcal{O}_K/I$ is always finite.

- Norm $I \stackrel{\text{def}}{=} \#(\mathcal{O}_K/I)$. If K is Galois, $\prod_\sigma I^\sigma = \langle \text{Norm}\, I \rangle$.
- If I is principal, $\text{Norm}\, \langle \gamma \rangle = |\text{Norm}\, \gamma|$.
  (beware: this is only for (fractional) $\mathcal{O}_K$-ideals).
- The norm is multiplicative: $\text{Norm}\, IJ = \text{Norm}\, I \cdot \text{Norm}\, J$.

For example, in a number field of degree n, the norm of $\langle p \rangle$ is p^n.

We look for the largest ideals that contain (divide) $\langle p \rangle$.

- Their norm has to be a p-th power.
- There are generally several such prime ideals above p.

Prime ideals

Important case when I is maximal (same as prime, for us):

- then $\mathcal{O}_K/I$ is a field.
- If I lies above p, then $\mathcal{O}_K/I$ is an extension of $\mathbb{F}_p = \mathbb{Z}/(\mathbb{Z} \cap I)$.
- The degree $[\mathcal{O}_K/I : \mathbb{Z}/(\mathbb{Z} \cap I)]$ is called the residue class degree or inertia degree of I.
- The inertia degree is commonly denoted f, but we also have f lying around...

Factorization of $p\mathcal{O}_K$

Guiding principle

Try to «read» the factorization of $\langle p \rangle$ from that of f mod p.

Caveat: This does not always work!

Condition (Dedekind criterion):

- if we have defined orders and indices of orders:
  p coprime to $[\mathcal{O}_K : \mathbb{Z}[\alpha]]$ (IOW, $\mathbb{Z}[\alpha]$ is p-maximal).
  In particular, if $\nu_p(\operatorname{disc} f) \leq 1$, then our condition is satisfied.
- if not, the only thing we can do is to write sufficient conditions that guarantee that we are in the easy case.

Sufficient conditions for the Dedekind crit.

In we are in any of the following situations:

- $\mathcal{O}_K = \mathbb{Z}[\alpha]$
- or $p \nmid f_n \operatorname{disc} f$ "coarse Dedekind criterion"
- or, informally, if $\mathcal{O}_K$ is not very different from $\mathbb{Z}[\alpha]$, as far as p is concerned

then the Dedekind criterion holds and we are in the easy case: the factorization of $\langle p \rangle$ is directly linked to that of f mod p.

Factorization of $\langle p \rangle = p\mathcal{O}_K$

Nice situation, when $\mathbb{Z}[\alpha]$ is p-maximal.

- Factors of $p\mathcal{O}_K$ correspond to factors of f mod p.
- Inertia degrees are degrees of irreducible factors.
- Ideal multiplicities are multiplicities of irr. factors.

Example. Let $K = \mathbb{Q}(\alpha)$ with $\alpha^3 = 2$.

- $\langle 2 \rangle = \mathfrak{a}_2^3$, with $\mathfrak{a}_2 = \langle 2, \alpha \rangle$. $\mathcal{O}_K/2\mathcal{O}_K \cong (\mathbb{F}_2)^3$.
- $\langle 3 \rangle = \mathfrak{a}_3^3$, with $\mathfrak{a}_3 = \langle 3, \alpha + 1 \rangle$. $\mathcal{O}_K/3\mathcal{O}_K \cong (\mathbb{F}_3)^3$.
- $X^3 - 2 \equiv (X+2)(X^2 + 3X - 1) \mod 5$, thus $\langle 5 \rangle = \mathfrak{a}_5\mathfrak{b}_5$, with $\mathfrak{a}_5 = \langle 5, \alpha + 2 \rangle$ and $\mathfrak{b}_5 = \langle 5, \alpha^2 + 3\alpha - 1 \rangle$. $\mathcal{O}_K/5\mathcal{O}_K \cong \mathbb{F}_5 \times \mathbb{F}_{5^2}$.

More taxonomy

Definitions

- p is inert in K if $\langle p \rangle$ is a prime ideal (hence $\mathcal{O}_K/p\mathcal{O}_K \equiv \mathbb{F}_{p^d}$).
- p ramifies in K if $\langle p \rangle$ has a repeated factor ($\Rightarrow p \mid \operatorname{disc} K$).
- p splits completely in K if $\langle p \rangle$ factorizes only into prime ideals of inertia degree 1.

Prime ideals of $\mathcal{O}_K$ also inherit this terminology: inert, ramified.

Unramified ideals have multiplicity 1 in the factorization of $(I \cap \mathbb{Z})\mathcal{O}_K$.

Examples on previous slide: $\mathfrak{a}_2, \mathfrak{a}_3$ ramified. $\mathfrak{a}_5, \mathfrak{b}_5$ unramified.

Important, for f defining a p-maximal $\mathbb{Z}[\alpha]$:

- p ramifies iff f has a repeated factor (i.e. $p \mid \operatorname{disc} f$).
- Also holds more generally: p ramifies iff $p \mid \operatorname{disc} K$.

Factoring ideals into prime ideals

Given a (possibly fractional) $\mathcal{O}_K$-ideal I, how do we factor it into prime ideals?

$$I = I_1 \cdot I_2 \cdot \cdots \cdot I_k.$$

This is a two-step process:

- Factor Norm I.
- For each p^k that appears in the factorization, find which of the prime ideals above p have a non-zero valuation at I.
- If I is fractional, one simple way to go is to factor the integral ideal dI first, and then divide by the prime ideals that divide $d\mathcal{O}_K$.

Prime ideals above primes

$$
\begin{array}{ccccccc}
K & \supset & \mathcal{O}_K & \supset & \mathfrak{p}_1 \cdots \mathfrak{p}_m & & \mathbb{F}_{p^{k_1}} \cdots \mathbb{F}_{p^{k_m}} \\
| & & | & & \diagdown \,|\, \diagup & & \diagdown \,|\, \diagup \\
\mathbb{Q} & \supset & \mathbb{Z} & \supset & p\mathbb{Z} & & \mathbb{F}_p
\end{array}
$$

Breathe

Things to keep in mind:

Ideals, in general, are things that we can deal with:

- they have bases (as $\mathbb{Z}$-modules) or generators (as $\mathcal{O}_K$ modules).
- operations: $+$, $\times$ (also: $\cap$).
- we can do operations on ideals using linear algebra.

Prime numbers in $\mathbb{Z}$ factor into prime ideals in $\mathcal{O}_K$.

Prime ideals in $\mathcal{O}_K$:

- are always above some rational prime p in $\mathbb{Z}$.
- lead to finite fields of the form $\mathcal{O}_K/I$ (finite field extending $\mathbb{F}_p$).

Easy ideals

Some ideals are very easy to work with.
When I is unramified and has residue class degree 1, then $I = (p, \alpha - r)$ for some $r \in \mathbb{F}_p$. This corresponds to the field isomorphism:

$$\begin{cases} \mathcal{O}_K/I & \to & \mathbb{F}_p, \\ \alpha & \mapsto & r \end{cases}$$

Note: these ideals are the most common ones!

- There are only finitely many prime ideals whose norm is not coprime to $\operatorname{disc} K$.
- Among the unramified prime ideals, those of residue class degree > 1 are less frequent.

Factorization of $\langle a - b\alpha \rangle = (a - b\alpha)\mathcal{O}_K$

Important case for NFS: factorization of $I = \langle a - b\alpha \rangle$.

It's actually easy to find the easy prime ideals that divide I.

See next lecture.

Further topics

Non-easy ideals

While non-easy ideals are exceedingly rare in the NFS context, there are a few situations where we want to deal with the mildly complicated process of finding their valuations in factorizations. This is covered in books (e.g. Cohen). I probably won't cover it.

Distribution of prime factoring patterns

When factoring $\langle p \rangle$, factoring patterns are not random at all. They are prescribed by a very important theorem called Chebotarev's density theorem, which ties these patterns to the Galois group.
Again, I probably won't cover it.

Plan

Number fields, algebraic numbers

Algebraic integers, ring of integers

Ideals

Factoring into prime ideals

Units and the class group

Units

Which elements of $\mathcal{O}_K$ are invertible ?

Theorem

An algebraic integer $x \in \mathcal{O}_K$ is invertible iff $\mathrm{Norm}_{K/\mathbb{Q}}(x) = \pm 1$.

Caveat: $x \in K$ with Norm $= 1$ has no reason to be a unit in $\mathcal{O}_K$.

As an abelian group, U_K has:

- A (finite!) torsion subgroup U_{tors} (roots of unity) ;
- a rank, so that $U_K \cong U_{\mathrm{tors}} \times \mathbb{Z}^{\mathrm{rank}}$.

Units

Finding torsion units is essentially trivial.

Finding the rank of the torsion-free part is also trivial (Dirichlet Unit Theorem).

It is very difficult to find the generators of the torsion-free part.

The class group

Principal ideals form a subgroup of the group of (fractional) ideals.

Class group, class number

The quotient $I(\mathcal{O}_K)/K^\times$ is called the class group $\mathrm{Cl}(\mathcal{O}_K)$.
Its order is called the class number of $\mathcal{O}_K$, often denoted h.

Fact: the class group is a finite abelian group.

Various consequences of the definition:

- An ideal is principal iff it maps to zero in the class group.
- If $h = 1$ (the class group is trivial) then any ideal is principal.
- If the exponent of the class group is λ, then for any ideal, I^λ is principal.

Computing the class group

Computing the class number (and structure of $\mathrm{Cl}(\mathcal{O}_K)$) is hard.

It is linked to the computation of a system of generators for units.

The number field sieve does in fact include the statement of a method for tackling the problem.

Generally, the complexity for computing h is subexponential.

Further topics

There is a lot more to say about the unit group and the class group (which are intimately related).

CSE291-14: The Number Field Sieve

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

Emmanuel Thomé

January 20, 2022

Part 3c

NFS in the not-so-easy case

A roadmap for NFS

Stumbling blocks

Prime ideals and factorization of $\langle a - b\alpha \rangle$

Making sense of a relation

The main steps of NFS and the NFS diagram

Recap from last time

We learned a lot from the algebraic number theory background.

How do we get back on our feet, and think about a factoring algorithm?

- The roadmap of the too-easy algorithm seemed very simple.
- We learned about multiple roadblocks that we have to circumvent to make this work:
  - Beyond the entirely-trivial cases (how do we factor F_7?)
  - and also in greater generality (how do we factor general numbers?)
- And then, assuming all this can be overcome, can we really make this a sieving algorithm?

Plan

A roadmap for NFS

Stumbling blocks

Prime ideals and factorization of $\langle a - b\alpha \rangle$

Making sense of a relation

The main steps of NFS and the NFS diagram

How would we factor N?

- Find $f \in \mathbb{Z}[x]$ and $m \in \mathbb{Z}$ such that $f(m) \equiv 0 \mod N$. Neither m, nor $\deg f$, nor the coefficients of f should be too large.
  - The analysis will help us see that in greater detail.
  - For some numbers, some very nice values exist.
- Fix a smoothness bound B.
- Find many pairs (a, b) such that:
  - $a - bm$ factors into primes below B.
  - $\langle a - b\alpha \rangle$ factors into prime ideals of norm below B.
- Using linear algebra, find a subset of the $(a - bx)$ such that:
  - $\prod_i (a_i - b_i m)$ is a square in $\mathbb{Z}$.
  - $\prod_i (a_i - b_i \alpha)$ is a square in $\mathbb{Z}[\alpha]$.
- Write down both square roots in $\mathbb{Z}$ and $\mathbb{Z}[\alpha]$, map them to $\mathbb{Z}/N\mathbb{Z}$, and hopefully get a factor.

How would we factor N?

- Find $f \in \mathbb{Z}[x]$ and $m \in \mathbb{Z}$ such that $f(m) \equiv 0 \mod N$. Neither m, nor $\deg f$, nor the coefficients of f should be too large.
  - The analysis will help us see that in greater detail.
  - For some numbers, some very nice values exist.
- Fix a smoothness bound B.
- Find many pairs (a, b) such that:
  - $a - bm$ factors into primes below B.
  - $\langle a - b\alpha \rangle$ factors into prime ideals of norm below B.
- Using linear algebra, find a subset of the $(a - bx)$ such that:
  - $\prod_i (a_i - b_i m)$ is a square in $\mathbb{Z}$.
  - $\prod_i (a_i - b_i \alpha)$ is a square in $\mathbb{Z}[\alpha]$.
- Write down both square roots in $\mathbb{Z}$ and $\mathbb{Z}[\alpha]$, map them to $\mathbb{Z}/N\mathbb{Z}$, and hopefully get a factor.

How would we factor N?

- Find $f \in \mathbb{Z}[x]$ and $m \in \mathbb{Z}$ such that $f(m) \equiv 0 \mod N$. Neither m, nor $\deg f$, nor the coefficients of f should be too large.
  - The analysis will help us see that in greater detail.
  - For some numbers, some very nice values exist.
- Fix a smoothness bound B.
- Find many pairs (a, b) such that:
  - $a - bm$ factors into primes below B.
  - $\langle a - b\alpha \rangle$ factors into prime ideals of norm below B.
- Using linear algebra, find a subset of the $(a - bx)$ such that:
  - $\prod_i (a_i - b_i m)$ is a square in $\mathbb{Z}$.
  - $\prod_i (a_i - b_i \alpha)$ is a square in $\mathbb{Z}[\alpha]$.
- Write down both square roots in $\mathbb{Z}$ and $\mathbb{Z}[\alpha]$, map them to $\mathbb{Z}/N\mathbb{Z}$, and hopefully get a factor.

How would we factor N?

- Find $f \in \mathbb{Z}[x]$ and $m \in \mathbb{Z}$ such that $f(m) \equiv 0 \mod N$. Neither m, nor $\deg f$, nor the coefficients of f should be too large.
  - The analysis will help us see that in greater detail.
  - For some numbers, some very nice values exist.
- Fix a smoothness bound B.
- Find many pairs (a, b) such that:
  - $a - bm$ factors into primes below B.
  - $\langle a - b\alpha \rangle$ factors into prime ideals of norm below B.
- Using linear algebra, find a subset of the $(a - bx)$ such that:
  - $\prod_i (a_i - b_i m)$ is a square in $\mathbb{Z}$.
  - $\prod_i (a_i - b_i \alpha)$ is a square in $\mathbb{Z}[\alpha]$. **This is tricky!**
- Write down both square roots in $\mathbb{Z}$ and $\mathbb{Z}[\alpha]$, map them to $\mathbb{Z}/N\mathbb{Z}$, and hopefully get a factor.

How would we factor N?

- Find $f \in \mathbb{Z}[x]$ and $m \in \mathbb{Z}$ such that $f(m) \equiv 0 \mod N$. Neither m, nor $\deg f$, nor the coefficients of f should be too large.
  - The analysis will help us see that in greater detail.
  - For some numbers, some very nice values exist.
- Fix a smoothness bound B.
- Find many pairs (a, b) such that:
  - $a - bm$ factors into primes below B.
  - $\langle a - b\alpha \rangle$ factors into prime ideals of norm below B.
- Using linear algebra, find a subset of the $(a - bx)$ such that:
  - $\prod_i (a_i - b_i m)$ is a square in $\mathbb{Z}$.
  - $\prod_i (a_i - b_i \alpha)$ is a square in $\mathbb{Z}[\alpha]$.
- Write down both square roots in $\mathbb{Z}$ and $\mathbb{Z}[\alpha]$, map them to $\mathbb{Z}/N\mathbb{Z}$, and hopefully get a factor.

Plan

A roadmap for NFS

Stumbling blocks

Prime ideals and factorization of $\langle a - b\alpha \rangle$

Making sense of a relation

The main steps of NFS and the NFS diagram

NFS

Not all fields are as cool as $\mathbb{Q}(\sqrt{-11})$ (see lecture 4).

- The ring of integers is not always obvious.
  Sometimes, it is even extremely hard to compute $\mathcal{O}_K$!
- In general, we do not have unique factorization of elements.
- We're not certain that we'll always like to restrict ourselves to a monic definition polynomial. (Spoiler alert: indeed, we won't!)
- The units can be much more complicated than ± 1.

We expect some difficulties!

Pollard's F_7 example

In the cubic integers example, Pollard only had the units issue to deal with.

- The field $\mathbb{Q}(\alpha) = \mathbb{Q}[x]/(x^3 + 2)$ does have a unit of infinite order.
- Fortunately, this generator is easy to find: $1 + \alpha$.
  This is easy to see: $\mathrm{Res}(1 + x, x^3 + 2) = (-1)^3 + 2 = 1$.

So there's no really annoying difficulty here.

We can simply add a column with the valuation in $(1 + \alpha)$.

What is a real pain, however, is how to factor algebraic numbers into elements. We'll leave that aside.

Pollard's F_7 example

In the case of $\mathbb{Q}(\sqrt[3]{-2})$, we would need the following preparation work.

- Choose a smoothness bound B.
- List all primes below B.
- List all primes in $\mathbb{Q}(\alpha)$ whose norm is below B.
- List the known units (-1 and $1+\alpha$)

Then we would need to find pairs (a, b) such that we have simultaneous smoothness.

- Can we do that with sieving? Yes.
- Will this end up giving us a factorization? Yes.

Sieving for smooth (a, b)

We are interested in many possible polynomials $\phi = a - bx$.

Note: it is useless to consider the case $\gcd(a, b) > 1$, since it brings no useful new information compared to the coprime case.

Pollard used sieving in a simple way:

- For each b from 1 to 2000, sieve the range $-4800 \leq a < 4800$ in order to detect the smooth values of $a - bm$.
  See file pollard.sage on Canvas.
- For each apparently smooth $a - bm$, compute and try to factor $\mathrm{Norm}(a - b\alpha)$.
- In cases where $\mathrm{Norm}(a - b\alpha)$ is smooth, factor it, and record this information.

F_7

F_7 was first factored with CFRAC in 1970.

Pollard's method: 1988.

- Is it significantly faster? Not really.
- Is it a general factoring method? Not at all.
- But it does bring something new.

First, we'll see how it can work with a number fields where not all ideals are principal.

Plan

A roadmap for NFS

Stumbling blocks

Prime ideals and factorization of $\langle a - b\alpha \rangle$

Making sense of a relation

The main steps of NFS and the NFS diagram

Sieving: later

We're definitely going to describe a sieving algorithm.

But: for the moment (next few slides), our description will be trial-division-based.

Remember that conceptually, sieving can be introduced after the fact by swapping two loops.

Two sides

Whenever we want to create a relation, there are clearly two sides to consider. Similarities are very strong.

- On the rational side, we compute $a - bm$.
  - For each prime number p, see if $p \mid a - bm$. If yes, record the valuation.
  - If $a - bm$ is fully factored, we're happy.
  - Pay attention to ± 1.
- On the algebraic side, we "compute" $a - b\alpha$.
  - For each prime ideal $\mathfrak{p}$, see if $\mathfrak{p} \mid \langle a - b\alpha \rangle$. If yes, record the valuation.
  - If $\langle a - b\alpha \rangle$ is fully factored this way, we're happy.
  - Pay attention to units.

Note: this does not mean that we factor $a - b\alpha$.

Note2: we need to think a bit about the interpretation of the relation that we obtain.

Factoring on the algebraic side

In order to be able to factor things on the algebraic side:

- we need to determine all "small" prime ideals that will define our factor base.
  "small": their norm must be below some bound B.
- we need be able to check if an ideal divides another.

We're also aware of the gap between factoring an element (which is not well-defined), and factoring an ideal into prime ideals. Units are part of this gap.

Plan

Prime ideals and factorization of $\langle a - b\alpha \rangle$

- Hard things vs doable things
- Describing prime ideals
- Factoring into ideals

Bad news, first

Real-life example (from DLP-240):

$$
\begin{aligned}
f &= 286512172700675411986966846394359924874576536408786368056\, x^3 \\
&\quad + 249088203007157661364751159824397355165818886038172555398 90\, x^2 \\
&\quad - 187636975600130165644039539283271210355804094599448546527 37\, x \\
&\quad - 236610408827000256250190838220824122997878994595785432202599
\end{aligned}
$$

$\operatorname{disc} f =$ A 236-digit integer (not an RSA modulus!).

Computing $\mathcal{O}_K$ is very hard

It is very hard to be absolutely sure that we have computed $\mathcal{O}_K$.

Computing $\mathcal{O}_K^*$ is infeasible

The computation of a system of generators for $\mathcal{O}_K^*$ is completely out of reach.

Good news

While the global objects (such as $\mathcal{O}_K$ and $\mathcal{O}_K^*$) are hard to compute, everything that is local (attached to a prime p) is much more tractable (polynomial in $\log p$ and $\deg f$).

- For any prime p, we can describe the prime ideals of $\mathcal{O}_K$ that are above p, even if we do not know $\mathcal{O}_K$.
- For any prime ideal $\mathfrak{p}$, finding the $\mathfrak{p}$-valuation of an ideal such as $\langle a - b\alpha \rangle$ is doable, even if we do not know $\mathcal{O}_K$.
- For most primes p, these tasks are actually very easy.

The other bit of good news is that we can work around the fact that computing $\mathcal{O}_K^*$ is out of reach.

Plan

Prime ideals and factorization of $\langle a - b\alpha \rangle$

- Hard things vs doable things
- Describing prime ideals
- Factoring into ideals

What are the prime ideals above p?

Preliminary question: does p divide f_n or $\mathrm{disc}(f)$?
If yes, you'll have to ask an expert (they won't charge much).

If not, then $\mathbb{Z}[\alpha]$ (or $\mathbb{Z}[\hat{\alpha}]$ if f not monic) can be used in lieu of $\mathcal{O}_K$. We can really do as if they were the same.

- If f factors modulo p into irreducible factors of degrees $d_1 + \cdots + d_k = n$, then there are k prime ideals above p, of residue class degrees d_1 to d_k.
- Repeated factors cannot appear (because $p \nmid \mathrm{disc}\, f$).

Example

$f = x^3 + 2$, $p = 31$: f splits completely $\bmod p$.
There are three prime ideals of degree 1 above p.

$f = x^3 + 2$, $p = 41$: f splits $\bmod p$ into $(\deg = 1) \times (\deg = 2)$.
There are two prime ideals, of degrees 1 and 2, above p.

What are the prime ideals above p?

Identifying most prime ideals

In the easy case ($p \nmid f_n \operatorname{disc} f$), a prime ideal above p is uniquely determined by

- The prime number p
- One of the irreducible factors of f mod p.

The most typical case is when the residue class degree is 1.
Such a prime ideal can be identified as $(p, x - r)$, or $(p, \alpha - r)$, or (p, r) depending on notations.

$(p, x - r)$ is the prime ideal above p that contains all algebraic integers that are $\mathcal{O}_K$-multiples of $(\alpha - r)$.

This is an implicit description, but it is sufficient for NFS.
Caveat: when $f_n \neq 1$, $(p, x - r) \neq \langle p, \alpha - r \rangle$.

Identifying most prime ideals

```
ideals=[]
f=K.defining_polynomial()
Disc=f.discriminant()
for p in prime_range(10000):
    if gcd(p,Disc) != 1:
        continue
    fp=f.change_ring(GF(p)).factor()
    for g,m in fp:
        assert m == 1
        if p^(g.degree()) < 10000:
            ideals.append((p,g))
```

Cado-NFS has a program called `makefb` which does just this.

What are the ideals that we miss?

There are prime ideals above the prime divisors of $f_n \operatorname{disc} f$.
Cado-NFS calls them "bad ideals".

- Whenever we look at what happens above a given p, everything is doable with a bit of code.
- We are only interested in prime ideals of small norm, and finding the prime numbers p in this range that divide $f_n \operatorname{disc} f$ is easy because they're small.

Note: in some cases, the simple mechanism can be extended.

There are a few "bad ideals" in $\mathcal{O}_K$.
With some effort, we can find and describe them.

Plan

Prime ideals and factorization of $\langle a - b\alpha \rangle$

- Hard things vs doable things
- Describing prime ideals
- Factoring into ideals

Divisibility by easy ideals

Question: is some ideal above p a divisor of the ideal $\langle a - b\alpha \rangle$?

Preliminary question: does p divide f_n or $\mathrm{disc}(f)$?
If yes, you'll have to ask an expert (they won't charge much).

If not, we are in the easy case, and it is quite simple.

Divisibility by easy ideals

Assume that
- $p \nmid f_n \operatorname{disc} f$ (easy case).
- p is coprime to $\gcd(a, b)$.
- $\mathfrak{p}$ is identified by $(p, g(x))$.
- We want to check if $\mathfrak{p} \mid \langle a - b\alpha \rangle$.

$$\begin{aligned} \mathfrak{p} \mid \langle a - b\alpha \rangle &\Leftrightarrow g(a/b) \equiv 0 \mod p \\ &\Leftrightarrow \operatorname{Res}(a - bx, g(x)) \equiv 0 \mod p \end{aligned}$$

Side-effect: at most one matching $\mathfrak{p}$ above a given p, and $\nu_{\mathfrak{p}}(\langle a - b\alpha \rangle) = \nu_p(\operatorname{Res}(a - bx, f(x)))$.

Only ideals of degree 1 matter

This can happen only if $\deg g = 1$.
As long as we are factoring $\langle a - b\alpha \rangle$, only ideals of the form $(p, x - r)$ can appear.

Data format

To represent the factorization of $\langle a - b\alpha \rangle$, we typically store this information:

- The integers a and b.
- All the prime factors of $\mathrm{Res}(a - bx, f(x))$.

This is concise, and sufficient to precisely identify all prime ideals in the factorization (when we need to do so).

- For most primes, this boils down to computing $a/b \bmod p$.
- For "bad primes", this is doable as well.

All this identification work can be done basically as fast as `printf`.

Plan

A roadmap for NFS

Stumbling blocks

Prime ideals and factorization of $\langle a - b\alpha \rangle$

Making sense of a relation

The main steps of NFS and the NFS diagram

Example from Cado-NFS

To do an F_7 factorization with Cado-NFS:

```
git clone https://gitlab.inria.fr/cado-nfs/cado-nfs
cd cado-nfs
make -j4
[download f7.params]
[download f7.poly]
./cado-nfs.py --wdir /tmp/F7 f7.params slaves.hostnames=localhost
```

We find in one of the `/tmp/F7/F7.upload/F7.*.gz` files:

```
-1044,509:2,2,d,13,10f,119,fa7,3a03:2,b,1f,161,e2f
```

Example from Cado-NFS

A relation: `-1044,509:2,2,d,13,10f,119,fa7,3a03:2,b,1f,161,e2f`

- `-1044,509`: These are $a = -1044$ and $b = 509$ (in decimal).
- `2,2,d,13,10f,119,fa7,3a03`: The prime factors of $a - b \times 2^{43}$.
- `2,b,1f,161,e2f`: The prime factors of $\mathrm{Res}(a - bx, x^3 + 2)$.

This says that (blue and red are hex above, decimal below):

$$
\begin{aligned}
-1044 - 509 \cdot 2^{43} &= \pm 2^2 \times 13 \times 19 \times \cdots \\
\langle -1044 - 509\alpha \rangle &= \text{a “bad ideal” of norm 2} \\
&\quad \times (11, \alpha - 4) \\
&\quad \times (31, \alpha - 27) \\
&\quad \times (353, \alpha - 292) \\
&\quad \times (3631, \alpha - 1389).
\end{aligned}
$$

Violet numbers such as 1389 are implicit: $a/b \bmod 3631 = 1389$.

Things to pay attention to

- Repeated factors are rare and when a prime divides multiple times, it is printed multiple times (see the 2 in the example).
- The unit on the rational side does not appear in the relation. It's easy enough to find out the sign!
- There is some information about "bad ideals".
  We might provide it to our expert so that they can identify these ideals properly.
- On the algebraic side, we only have a factorization into ideals.

Important caveat for non-monic f

Reminder:

$$\mathrm{Norm}\langle a - b\alpha\rangle = \mathrm{Norm}(a - b\alpha) = \frac{1}{f_n}\,\mathrm{Res}(a - bx, f(x)).$$

- We claim that we are writing down the factorization of $\langle a - b\alpha\rangle$.
- But the prime factors that we list are those of $\mathrm{Res}(a - bx, f(x))$.
- There's got to be something missing.

The ideal J is here to square things up

When $f_n \neq 1$, we are actually writing down the factorization of $J \times \langle a - b\alpha\rangle$, with $J = \langle 1, \alpha\rangle^{-1} = \{x,\ x \in \mathcal{O}_K \text{ and } x\alpha \in \mathcal{O}_K\}$.

- $J = \langle 1, \alpha\rangle^{-1}$ is an integral ideal of norm f_n.
  J has no reason to be prime (e.g., if f_n isn't, J isn't either).
- This is hardly ever mentioned in the literature.

Example with non-monic f

The number $2^{199} + 3^{109}$ is a nice 60-digit number to play with.

```
./cado-nfs.py --wdir /tmp/c60 $(bc<<<2^199+3^109)
```

Summary of the information we have

On the algebraic side, we have:

- in a straightforward manner, the ideals and valuations in the factorization of $\langle a - b\alpha \rangle \times J$, when $p \nmid f_n \operatorname{disc}(f)$ (all p but finitely many).
- with some extra work, the full factorization of $\langle a - b\alpha \rangle$ can be obtained, but we'll have to ask our expert for that.

What remains to be done

If we follow our basic workplan, we can see how linear algebra will produce a subset of the $(a - bx)$ such that

- $\prod_i (a_i - b_i m)$ is a square in $\mathbb{Z}$ (we will add a column with the sign for that).
- $\prod_i \langle a_i - b_i \alpha \rangle$ has even valuations at
  - all easy prime ideals if we only look at these.
  - all prime ideals with some extra effort.

Therefore $\langle \prod_i (a_i - b_i \alpha) \rangle$ is the square of an ideal, but we do not know if $\prod_i (a_i - b_i \alpha)$ is the square of an element!

We will see how to work around this difficulty when we address the square root computation.

Plan

A roadmap for NFS

Stumbling blocks

Prime ideals and factorization of $\langle a - b\alpha \rangle$

Making sense of a relation

The main steps of NFS and the NFS diagram

What we have done so far

We have a few ideas of how an NFS algorithm could look like.

- So far, we mentioned ad hoc numbers, but our demo gives away the fact that it also works in greater generality.
- Factoring into prime ideals is doable.
- We mentioned some possibilities down the road, but I claim that these can be circumvented.

Now: list (and name) all the different steps of the General Number Field Sieve (GNFS).

We're going to repeat blocks of our sketch slide "How would we factor N?"

How would we factor N?

- Find $f \in \mathbb{Z}[x]$ and $m \in \mathbb{Z}$ such that $f(m) \equiv 0 \mod N$. Neither m, nor $\deg f$, nor the coefficients of f should be too large.
  - The analysis will help us see that in greater detail.
  - For some numbers, some very nice values exist.
- Fix a smoothness bound B.
- Find many pairs (a, b) such that:
  - $a - bm$ factors into primes below B.
  - $\langle a - b\alpha \rangle$ factors into prime ideals of norm below B.
- Using linear algebra, find a subset of the $(a - bx)$ such that:
  - $\prod_i (a_i - b_i m)$ is a square in $\mathbb{Z}$.
  - $\prod_i (a_i - b_i \alpha)$ is a square in $\mathbb{Z}[\alpha]$. **This is tricky!**
- Write down both square roots in $\mathbb{Z}$ and $\mathbb{Z}[\alpha]$, map them to $\mathbb{Z}/N\mathbb{Z}$, and hopefully get a factor.

Finding f and m

Find $f \in \mathbb{Z}[x]$ and $m \in \mathbb{Z}$ such that $f(m) \equiv 0 \mod N$.
Neither m, nor $\deg f$, nor the coefficients of f should be too large.

- The analysis will help us see that in greater detail.
- For some numbers, some very nice values exist.

This is called **Polynomial Selection**: next lecture.

Here's a simple method called base-m to do it for arbitrary N:

- Choose the degree d of f s.t. $N > 2^{d^2}$.
- Set $m = \lceil N^{1/(d+1)} \rceil$.
- Write N in base m: $N = \sum_{i=0}^{d} f_i m^i$ where $0 \le f_i < m$.
- Set $f = \sum_{i=0}^{d} f_i x^i$. (not monic!).
- Notation-wise, we sometimes write "the rational polynomial" as $g = x - m$.

Parameters

Remark that d is a free parameter in the previous slide.

So is, for example, the bound B.

As well as many, many other parameters!

This is called parameter selection

Parameter selection is among the black arts in NFS!

- Asymptotic analysis gives asymptotic guidelines.
- In practice, it's a complicated matter which requires a lot of global understanding of how NFS works.

We'll tentatively cover a bit of the practical side of this by the end of the quarter.

Finding pairs a, b

Find many pairs (a, b) such that:

- $a - bm$ factors into primes below B.
- $\langle a - b\alpha \rangle$ factors into prime ideals of norm below B.

This is called **Relation Collection**: beginning of February.

One of the ways to do relation collection is sieving.

- It is actually possible to sieve for rational primes $p \in \mathbb{Z}$ but also for prime ideals $\mathfrak{p} \subset \mathcal{O}_K$.
- There are many, many, many parameters.
- Most of the old knowledge of sieving from the QS era is relevant.
- This is the most expensive part, computationally speaking.

Combining pairs

Using linear algebra, find a subset of the $(a - bx)$ such that:

- $\prod_i (a_i - b_i m)$ is a square in $\mathbb{Z}$.
- $\prod_i (a_i - b_i \alpha)$ is a square in $\mathbb{Z}[\alpha]$. **This is tricky!**

Combining pairs

Using linear algebra, find a subset of the $(a - bx)$ such that:

- $\prod_i (a_i - b_i m)$ is a square in $\mathbb{Z}$.
- $\prod_i (a_i - b_i \alpha)$ is (almost) a square in $\mathbb{Z}[\alpha]$.

This comprises two steps: We will see both mid-February.

- The **Filtering** step is a pre-processing step.
- Then we have **Linear Algebra** proper.

Linear algebra is the second most expensive step, and requires expensive hardware, too.

Factoring N, at last

Arrange so that $\prod_i (a_i - b_i\alpha)$ really is a square in $\mathbb{Z}[\alpha]$.
Write down both square roots in $\mathbb{Z}$ and $\mathbb{Z}[\alpha]$, map them to $\mathbb{Z}/N\mathbb{Z}$, and hopefully get a factor.

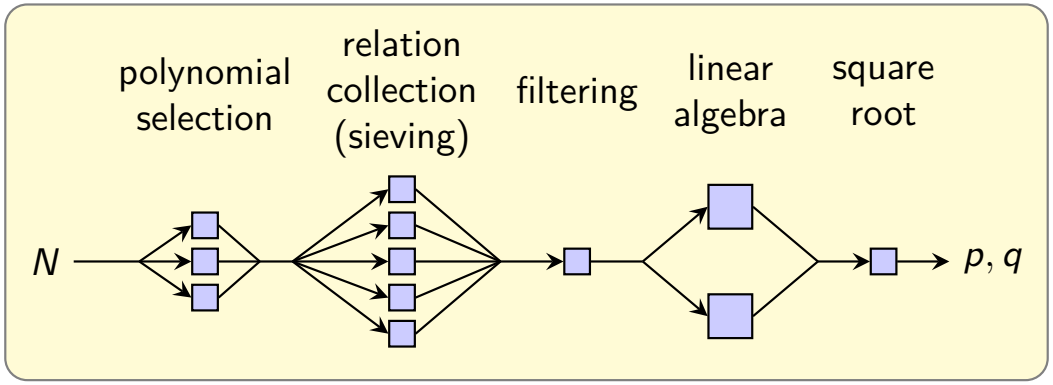
Again, two steps here. End of February.

- A pre-processing step called **the characters step**.
- Then **the square root step**.

This step will entail some more algebraic number theory, as well asymptotically fast algorithms.

As each square root only has probability $1/2$ to factor N, this step is designed to produce several independent square roots.

The different steps of NFS



Note: there is also a version of NFS that computes discrete logarithms in $\mathbb{F}_p^*$. The main outline is similar. End of February.

Some handwaving

- We find f with a known root m modulo N.
- Let $\mathbb{Q}(\alpha)$ be the number field defined by f.
- For any polynomial $P(x)$, we have:
  - the integer $P(m)$;
  - the number field element $P(\alpha)$;

  These are compatible: both map to $P(m)$ mod N in $\mathbb{Z}/N\mathbb{Z}$.

$$
\begin{tikzcd}
 & \mathbb{Z}[x] \arrow[dl, "x \to m"'] \arrow[dr, "x \to \alpha"] & \\
\text{subring of } \mathbb{Q} \quad \mathbb{Z}[m] = \mathbb{Z} \arrow[dr, "\bmod N"'] & & \mathbb{Z}[\alpha] \quad \text{subring of } \mathbb{Q}(\alpha) \arrow[dl, "\alpha \to m"] \\
 & \mathbb{Z}/N\mathbb{Z} &
\end{tikzcd}
$$

Some handwaving

- We find f with a known root m modulo N.
- Let $\mathbb{Q}(\alpha)$ be the number field defined by f.
- For any polynomial $a - bx$, we have:
  - the integer $a - bm$;
  - the number field element $a - b\alpha$;

These are compatible: both map to $P(m)$ mod N in $\mathbb{Z}/N\mathbb{Z}$.

$$
\begin{tikzcd}
 & \mathbb{Z}[x] \arrow[dl, "x \to m"'] \arrow[dr, "x \to \alpha"] & \\
\text{subring of } \mathbb{Q} \quad \mathbb{Z}[m] = \mathbb{Z} \arrow[dr, "\bmod N"'] & & \mathbb{Z}[\alpha] \quad \text{subring of } \mathbb{Q}(\alpha) \arrow[dl, "\alpha \to m"] \\
 & \mathbb{Z}/N\mathbb{Z} &
\end{tikzcd}
$$

Some handwaving

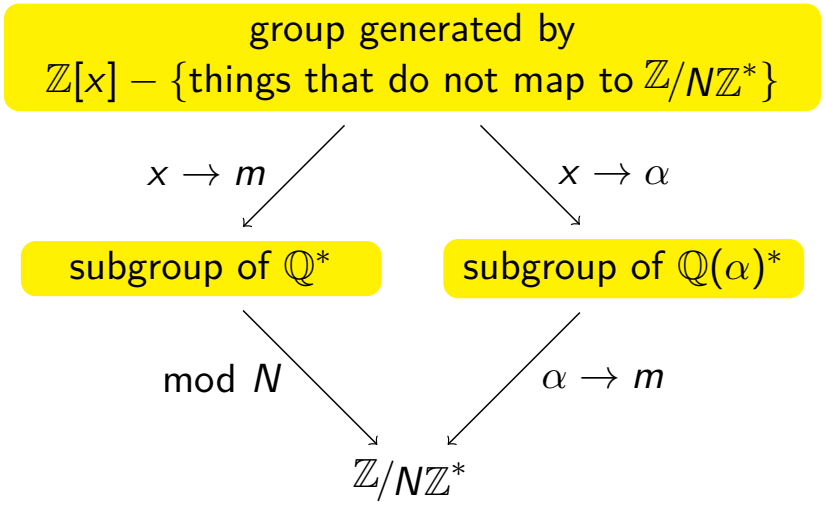
- We find f with a known root m modulo N.
- Let $\mathbb{Q}(\alpha)$ be the number field defined by f.
- For any polynomial $\prod_i (a_i - b_i x)$, we have:
  - the integer $\prod_i (a_i - b_i m)$;
  - the number field element $\prod_i (a_i - b_i \alpha)$;

  These are compatible: both map to $P(m)$ mod N in $\mathbb{Z}/N\mathbb{Z}$.

$$
\begin{tikzcd}
 & \mathbb{Z}[x] \arrow[dl, "x \to m"'] \arrow[dr, "x \to \alpha"] & \\
\text{subring of } \mathbb{Q} \quad \mathbb{Z}[m] = \mathbb{Z} \arrow[dr, "\bmod N"'] & & \mathbb{Z}[\alpha] \quad \text{subring of } \mathbb{Q}(\alpha) \arrow[dl, "\alpha \to m"] \\
 & \mathbb{Z}/N\mathbb{Z} &
\end{tikzcd}
$$

Write something multiplicative

The NFS diagram can also be written as a multiplicative diagram, even though it is a bit awkward to write it as such.



No difference in practice between the two diagrams.

- The multiplicative one just says that we won't stumble on factors of N accidentally. There is no practical difference between $\mathbb{Z}[x]$ and the structure on top.
- The multiplicative diagram does have an interest in the discrete logarithm context.

Rundown of an NFS computation

A more detailed look at the factorization of $2^{199} + 3^{109}$.

```
./cado-nfs.py --wdir /tmp/c60 $(bc<<<2^199+3^109)
```

CSE291-14: The Number Field Sieve

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

Emmanuel Thomé

January 25, 2022

Part 3d

NFS: a quick analysis

Tools

Parameters

First order

Second order

Summary

Motivation

- NFS has many parameters.
- The asymptotic analysis can be a rough guide... asymptotically.
  It is not wise to take these values as granted for a practical computation.

We will (probably) come back to the complexity analysis of NFS in March.

Goal today: do the analysis, just to fix ideas.

Plan

Tools

Parameters

First order

Second order

Summary

The L function

The following notation is attributed to R. Schroeppel.

$$L_x[a, \alpha] = \exp\left(\alpha(\log x)^a(\log\log x)^{1-a}\right).$$

CEP with the L function

A random integer $n \leq L_x[a, \alpha]$ is $L_x[b, \beta]$-smooth with probability:

$$L_x\left[a - b, -\frac{\alpha}{\beta}(a - b)(1 + o(1))\right].$$

This formulation is very important for analyzing sieve algorithms.

Calculus with $L_x[a, \alpha] = \exp\left(\alpha(\log x)^a(\log\log x)^{1-a}\right).$

Basic formulae with L

$$L_x[a,\alpha] \times L_x[b,\beta] = \begin{cases} L_x[a, \alpha + o(1)] & \text{if } a > b, \\ L_x[b, \beta + o(1)] & \text{if } b > a, \\ L_x[a, \alpha + \beta] & \text{if } a = b. \end{cases}$$

$$L_x[a,\alpha] + L_x[b,\beta] = \begin{cases} L_x[a, \alpha + o(1)] & \text{if } a > b, \\ L_x[b, \beta + o(1)] & \text{if } b > a, \\ L_x[a, \max(\alpha, \beta)] & \text{if } a = b. \end{cases}$$

$$L_x[b,\beta]^{\log_{\log x} L_x[a,\alpha]} = L_x[a+b, \alpha\beta].$$

$$L_{L_x[b,\beta]}[a,\alpha] = L_x[ab, \alpha\beta^a b^{1-a} + o(1)].$$

$$\log_{\log x} L_x[a,\alpha] \cdot \log_{\log x} L_x[-a, 1/\alpha] = 1.$$

Plan

Tools

Parameters

First order

Second order

Summary

Many parameters

Three main parameters.

- The degree d of the polynomial f.
- The smoothness bound: $B = L_N[b, \beta]$.
- The bound on a and b in $a - b\alpha$: $A = L_N[a, \alpha]$.

Caveat: obvious notation clashes!

Many parameters

Three main parameters.

- The degree d of the polynomial f.
- The smoothness bound: $B = L_N[b, \beta]$.
- The bound on the coefficients of $\phi(x)$: $A = L_N[a, \alpha]$.

Convenient form for d

The simplistic base-m polynomial selection works for arbitrary N.

- Set $m = \lceil N^{1/(d+1)} \rceil$.
- Write N in base m: $N = \sum_{i=0}^{d} f_i m^i$ where $0 \le f_i < m$.
- Set $f = \sum_{i=0}^{d} f_i x^i$. (not monic!)

It will be convenient to choose d so that $N^{1/(d+1)}$ has a nice expression.

Asymptotically, we expect that d grows to ∞ as $N \to \infty$, so $N^{1/(d+1)} = \left(N^{1/d}\right)^{1+o(1)}$.

Use L notation

We have $N = L_N[1, 1]$, so let us take $d = \log_{\log N} L_N[D, \delta]$.
This yields $m = L_N[1 - D, 1/\delta \cdot (1 + o(1))]$.

Checklist

- $d = \deg f = \log_{\log N} L_N[D, \delta]$.
- The smoothness bound: $B = L_N[b, \beta]$.
- The bound on the coefficients of $\phi(x)$: $A = L_N[a, \alpha]$.
- This yields $m = L_N[1 - D, 1/\delta \cdot (1 + o(1)]$.

Next step: how large are $a - bm$ and $\mathrm{Norm}(a - b\alpha)$?

Checklist

- $d = \deg f = \log_{\log N} L_N[D, \delta]$.
- The smoothness bound: $B = L_N[b, \beta]$.
- The bound on the coefficients of $\phi(x)$: $A = L_N[a, \alpha]$.
- This yields $m = L_N[1 - D, 1/\delta \cdot (1 + o(1)]$.

~~Next step: how large are $a - bm$ and $\mathrm{Norm}(a - b\alpha)$?~~
Next step: how large are $\mathrm{Res}(\phi(x), x - m)$ and $\mathrm{Res}(\phi(x), f(x))$?

How large is $|a - bm| = |\operatorname{Res}(\phi(x), x - m)|$?

The coefficients of $\phi(x)$ are at most $A = L_N[a, \alpha]$.

We have $|\operatorname{Res}(\phi(x), x - m)| = O(Am)$.

$|\operatorname{Res}(\phi(x), x - m)|$

condition	$\lvert\operatorname{Res}(\phi(x), x - m)\rvert$
$a < 1 - D$	$m^{1+o(1)} = L_N[1 - D, 1/\delta \cdot (1 + o(1))]$
$a = 1 - D$	$L_N[1 - D, (\alpha + 1/\delta) \cdot (1 + o(1))]$.
$a > 1 - D$	$L_N[a, \alpha \cdot (1 + o(1))]$.

How large is $|\mathrm{Res}(\phi(x), f(x))|$?

The coefficients of $\phi(x)$ are at most $A = L_N[a, \alpha]$.

We have:

$$\mathrm{Res}(u - vx, f(x)) = f_d u^d + f_{d-1} u^{d-1} v + \cdots + f_0 v^d.$$

- All summands have the same size: $\approx m \cdot A^d$.
  Note: $A^d = L_N[a + D, \alpha\delta]$.
- The degree-dependent multiplication has negligible impact.
- FYI, more general formula: $\approx C \times \|\phi\|^{\deg f} \|f\|^{\deg \phi}$
  with C a combinatorial term that depends on $\deg f$ and $\deg \phi$.

How large is $|\mathrm{Res}(\phi(x), f(x))|$?

The coefficients of $\phi(x)$ are at most $A = L_N[a, \alpha]$.

We have:

$$\mathrm{Res}(u - vx, f(x)) = f_d u^d + f_{d-1} u^{d-1} v + \cdots + f_0 v^d.$$

- All summands have the same size: $\approx m \cdot A^d$.
  Note: $A^d = L_N[a + D, \alpha\delta]$.

$|\mathrm{Res}(\phi(x), f(x))|$; which one of m and A^d wins?

condition	$\lvert\mathrm{Res}(\phi(x), f(x))\rvert$
$a + D < 1 - D$	$m^{1+o(1)} = L_N[1 - D, 1/\delta \cdot (1 + o(1))]$
$a + D = 1 - D$	$L_N[1 - D, (\alpha\delta + 1/\delta) \cdot (1 + o(1))]$.
$a + D > 1 - D$	$L_N[a + D, \alpha\delta \cdot (1 + o(1))]$.

Checklist

- $d = \deg f = \log_{\log N} L_N[D, \delta]$.
- This yields $m = L_N[1 - D, 1/\delta \cdot (1 + o(1))]$.
- The smoothness bound: $B = L_N[b, \beta]$.
- The bound on the coefficients of $\phi(x)$: $A = L_N[a, \alpha]$.
- $|\operatorname{Res}(\phi, x - m)| \leq |\operatorname{Res}(\phi, f)| = L_N[\underbrace{\max(1 - D, a + D)}_{\nu}, \cdot]$.
- The smoothness probability is $L_N[\nu - b, \cdot]$, by CEP.

The total cost is:

$$\underbrace{(\text{finding smooth pairs}) + (\text{factoring into relations})}_{L_N[\max(a,b),\cdot] \text{ is a safe upper bound}} + \underbrace{(\text{linear algebra})}_{L_N[b,\cdot]}.$$

Plan

Tools

Parameters

First order

Second order

Summary

We need enough relations

$$L_N[a, \mathbf{+}] \times L_N[\max(1 - D, a + D) - b, \mathbf{-}] \geq L[b, \mathbf{+}].$$

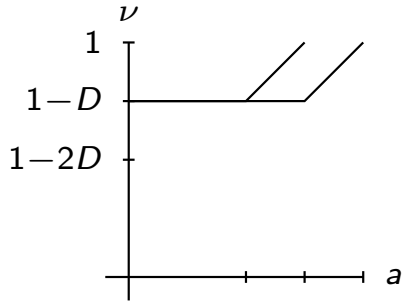
Several consequences (as we had in the QS case):

- $a \geq \max(1 - D, a + D) - b$.
- Furthermore, $a > b$ or $a > \max(1 - D, a + D) - b$ cannot be optimum choices, as we can improve the overall cost if it happens to be the case.
  - If $a > b$ and $1 - D \leq a + D$: decrease a to $\max(b, 1 - 2D)$.
  - If $a > b$ and $1 - D \geq a + D$: increase b, decrease a.
  - If $a = b$ and $a > \max(1 - D, a + D) - b$: decrease a and b.

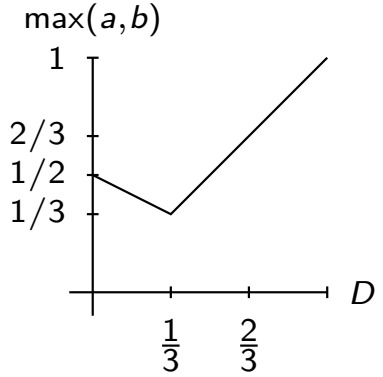
We can thus assume $a = b = \max(1 - D, a + D) - b$ (possibly with a $o(1)$ shift).

Optimum choice for D

- We don't know what D is, plot the size of both resultants as a function of a. (using $|\mathrm{Res}(\phi, f)| = L_N[\nu, \cdot]$.)



- Given our reasoning, here's how the optimum $\max(a, b)$ looks like as a function of D.



We do the analysis with $D = a = b = 1/3$, and see what we get.
In particular: pay attention to whether **+** and **−** compensate well!

Checklist

- $d = \deg f = \log_{\log N} L_N[1/3, \delta]$.
- $m = L_N[2/3, 1/\delta \cdot (1 + o(1))]$.
- The smoothness bound: $B = L_N[1/3, \beta]$.
- The bound on the coefficients of $\phi(x)$: $A = L_N[1/3, \alpha]$.
- Note: this makes $A^2 = L_N[1/3, 2\alpha + o(1)]$ polynomials ϕ to choose from.
- $|\operatorname{Res}(\phi, x - m)| = L_N[2/3, 1/\delta + o(1)]$.
- $|\operatorname{Res}(\phi, f)| = L_N[2/3, \alpha\delta + 1/\delta + o(1)]$.
- The smoothness probability is $L_N[1/3, \cdot]$, by CEP.

And the total cost would be $L_N[1/3, \cdot]$ if we find a solution.

Plan

Tools

Parameters

First order

Second order

Summary

Smoothness

Heuristic

We have to assume that values such as $\mathrm{Res}(\phi, x - m)$ or $\mathrm{Res}(\phi, f)$ behave like random integers of the same size.
This is mandatory if we want to apply CEP.
This heuristic is also present in QS, but not in Dixon's random squares.

Assuming that, the probability that both $\mathrm{Res}(\phi, x - m)$ and $\mathrm{Res}(\phi, f)$ are smooth is:

$$L_N\left[1/3, -\frac{1}{3\beta}\cdot\frac{1}{\delta}(1+o(1))\right] \times L_N\left[1/3, -\frac{1}{3\beta}\cdot(\alpha\delta+\frac{1}{\delta})(1+o(1))\right].$$
$$= L_N\left[1/3, -\frac{1}{3\beta}\cdot(\alpha\delta+\frac{1}{\delta}+\frac{1}{\delta})(1+o(1))\right].$$

More characterization of the optimum

The probability of smoothness:

$$L_N\left[1/3, -\frac{1}{3\beta}\cdot(\alpha\delta+\frac{1}{\delta}+\frac{1}{\delta})(1+o(1))\right].$$

Notice that δ no longer appears anywhere else.

Pick the best δ

The smaller the Res values, the better the smoothness probability. We minimize $\alpha\delta+\frac{2}{\delta}$ by with $\delta=\sqrt{2/\alpha}$.

"Having enough relations" translates to:

$$2\alpha-\frac{1}{3\beta}\cdot 2\sqrt{2\alpha}\geq\beta.$$

Factoring into relations

Spoilers:

- sieving will crush the cost of factoring relations to something asymptotically negligible,
- linear algebra will cost $(B^2)^{1+o(1)}$,

... so that the total cost is $L_N[1/3, 2\max(\alpha, \beta) + o(1)]$.

Given this total cost, it makes sense to search for a solution with $\alpha = \beta$. Can we find one?

Complexity of NFS

We have a solution with $\alpha = \beta$ if we find a solution to:

$$2\alpha - \frac{1}{3\beta} \cdot 2\sqrt{2\alpha} \geq \beta \text{ with } \alpha = \beta.$$

$$3\beta^2 \geq 2\sqrt{2\beta}.$$

$$\beta^{3/2} \geq \sqrt{8/9}$$

$$\alpha = \beta \geq \sqrt[3]{8/9}.$$

$$2\beta \geq \sqrt[3]{64/9}.$$

Complexity of NFS

Asymptotically, and heuristically, NFS has a complexity of:

$$L_N\left[1/3, (64/9)^{1/3} + o(1)\right].$$

Plan

Tools

Parameters

First order

Second order

Summary

Complexity of NFS: key equations

$$B = L_N\left[1/3, (8/9)^{1/3} + o(1)\right].$$

$$A = L_N\left[1/3, (8/9)^{1/3} + o(1)\right].$$

$$d = \log_{\log N}(L_N\left[1/3, 3^{1/3} + o(1)\right])$$

$$= (3^{1/3} + o(1)) \cdot \left(\frac{\log N}{\log\log N}\right)^{1/3}.$$

$$\mathrm{Res}(\phi, f) = L_N\left[1/3, \frac{3}{2} \cdot \sqrt{2\alpha} + o(1)\right] = L_N\left[1/3, 3 \cdot 3^{-1/3} + o(1)\right].$$

$$\mathrm{Res}(\phi, x - m) = L_N\left[1/3, \frac{1}{2} \cdot \sqrt{2\alpha} + o(1)\right] = L_N\left[1/3, 3^{-1/3} + o(1)\right].$$

Do not over-interpret this!

"In theory", algebraic norm is $3\times$ rational norm. Not in practice.

Huge difference with QS

$$\text{QS: } \exp\left(1 \cdot (\log N)^{1/2} (\log\log N)^{1/2} \cdot (1 + o(1))\right).$$

$$\text{NFS: } \exp\left((64/9)^{1/3} \cdot (\log N)^{1/3} (\log\log N)^{2/3} \cdot (1 + o(1))\right).$$

(note: $(64/9)^{1/3} = 1.923\ldots$)

Asymptotics can be tricky, but the complexity difference is really a major one.

Wrap up

- NFS complexity for arbitrary N is:

$$L_N[1/3, (64/9)^{1/3} + o(1)].$$

- It is for arbitrary N, thus General NFS (GNFS).
- We left much aside, including:
  - How do the inner algorithms work?
  - In particular, is it true that sieving can eliminate the cost of factoring into relations?
  - And is it true that we solve the linear system in time $B^{2+o(1)}$?

CSE291-14: The Number Field Sieve

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

Emmanuel Thomé

January 27, 2022

Part 4a

Polynomial selection in NFS

Introduction

Size of the coefficients: possible and impossible things

Searching for good polynomials with base-m

Skewness

Non-monic linear polynomials: Kleinjung's 2005 algorithm

Plan

Introduction

Size of the coefficients: possible and impossible things

Searching for good polynomials with base-m

Skewness

Non-monic linear polynomials: Kleinjung's 2005 algorithm

Goal

Our goal: review the different methods for polynomial selection.

- Why is it important?
- What kind of game is it?
- What are the different methods?
- How do we measure the quality of the output?

What is polynomial selection

The polynomial selection phase is when we choose the pair of polynomials that define both sides of the NFS diagram.

- The algebraic polynomial f defines the number field.
- The rational polynomial (thus far, $x - m$) completes the picture.

In practice, this is more general

In fact, not even $x - m$ is monic. Several methods do relax this condition (but not the first ones).
In some cases, both polynomials are of degree > 1.

Importance of polynomial selection

Polynomial selection is important because it determines the size of the "norms" (actually, of the integers being checked for smoothness).

- Asymptotic analysis crudely reduced polynomial selection to the choice of the pair (D, δ).
- We eventually found out that δ was controlling the compromise between the size of $\mathrm{Res}(\phi, x - m)$ and of $\mathrm{Res}(\phi, f)$.

This general role is also true in practice

A good polynomial selection makes these "norms" small as $\phi(x) = a - bx$ ranges over the values we explore.

- Certainly, some things can be achieved, and some can't.
- Can we force these values to be smooth more often than on average?

A general workplan

Starting point: a method that can yield good polynomial pairs.

- Arrange so that the method has many degrees of freedom.
- Explore a huge search space to find exceptional situations.
- Find reasonable assessment criteria that make it possible to identify which are the "exceptionally good" polynomial pairs.

An extension: non-linear rational polynomial

What if we replace $x - m$ by $m_1 x - m_0$?

- $\mathrm{Res}(a - bx, x - m) = a - bm$ becomes $\mathrm{Res}(a - bx, m_1 x - m_0) = am_1 - bm_0$, which looks nicer.
- If we write $m = m_0/m_1 \in \mathbb{Q}$ and that $f(m) \equiv 0 \mod N$, everything works as before.
- The condition to meet is the existence of a common root:

$$\mathrm{Res}(f(x), m_1 x - m_0) \equiv 0 \mod N.$$

This extra degree of freedom has been part of all polynomial selection algorithms since the early 2000s.

An extension: higher degree polynomials

If a polynomial selection can find a pair of nonlinear polynomials:

- whose resultant is divisible by N with multiplicity 1
- and with a known common root in $\mathbb{Z}/N\mathbb{Z}$

Then we can work exactly along the lines of NFS.

Caveat: no such thing is known in general, EXCEPT for DLP.

- NFS for DLP (discrete logs in $\mathbb{Z}/p\mathbb{Z}^\times$): p replaces N.
- The existence of root finding $\bmod p$ is the key.
- In some cases (but not always), this wins.

Notations

Traditionally, notations are as follows:

- f is the algebraic polynomial.
  The coefficients are named $f_0, \ldots, f_d$, or $a_0, \ldots, a_d$.
- g is the linear polynomial.

Often, to highlight the symmetric roles played by the two sides:

- f_0 is "the polynomial on side 0" (typically $\deg f_0 = 1$).
- f_1 is "the polynomial on side 1".
- But this messes with the per-coefficient notations.
  Notations $a_0, \ldots, a_d$ are preferred for coefficients of the nonlinear polynomial in this case.

Implementations such as Cado-NFS are mostly agnostic w.r.t side numbering.

Plan

Introduction

Size of the coefficients: possible and impossible things

Searching for good polynomials with base-m

Skewness

Non-monic linear polynomials: Kleinjung's 2005 algorithm

Size matters

Notations: $(f_0, f_1), \deg f_1 = d, f_1 = \sum a_i x^i$.

Our first approach consists in searching for "small" polynomial pairs.

- Eventually, one of our guides will be the size of the integers we will try to factor.
- Given the power dependency in the degrees of the polynomials, we have only a few possible choices for the degree.
- Given a choice for $d = \deg f_1$, can we obtain polynomials f_0 and f_1 with small coefficients?

Two equations (Kleinjung, 2016)

A good question to ask

In order to reach all integers in a range $[M, 2M]$, how large do we have to choose the coefficients of f_0 and f_1?
Let M^{c_0} be a bound on the coefficients of f_0 (likewise M^{c_1} for f_1).

- First constraint: to reach M different values with the degrees of freedom that we have:

$$c_0 \cdot (d_0 + 1) + c_1 \cdot (d_1 + 1) \geq 1.$$

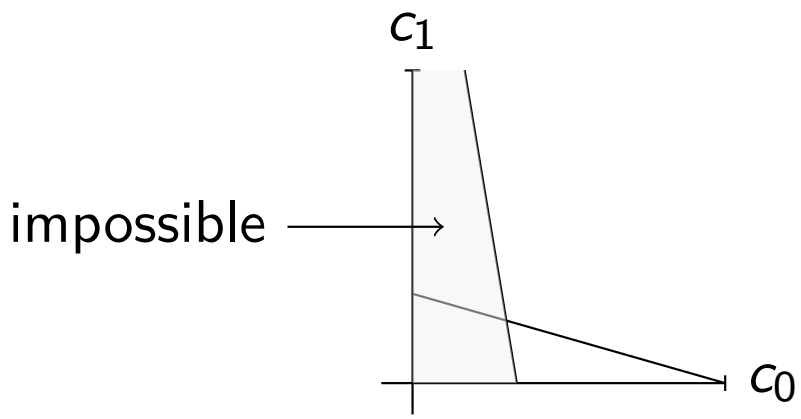
- Second constraint: $\mathrm{Res}(f_0, f_1)$ must be at least M.
  Since $\mathrm{Res}(f_0, f_1) = M^{o(1)} \|f_0\|^{\deg f_1} \|f_1\|^{\deg f_0}$, we must have:

$$c_0 \cdot d_1 + c_1 \cdot d_0 \geq 1.$$

Two different constraints

Note that the constraints are of different nature.

- $c_0 \cdot (d_0 + 1) + c_1 \cdot (d_1 + 1) \geq 1$.
  Pairs not meeting this constraint may exist, but such a family cannot reach all integers.
- $c_0 \cdot d_1 + c_1 \cdot d_0 \geq 1$.
  It is outright impossible for pairs to not meet this constraint, and be useful for NFS.



Important example #1

Take what the naive polynomial selection method gives: $d_0 = 1$, $d_1 = d$, $c_0 = c_1 = \frac{1}{d+1}$.

- $c_0 \cdot (d_0 + 1) + c_1 \cdot (d_1 + 1) = \frac{2}{d+1} + \frac{d+1}{d+1} \geq 1$.
- $c_0 \cdot d_1 + c_1 \cdot d_0 = \frac{d}{d+1} + \frac{1}{d+1} = 1$.

Put otherwise, the resultant bound is tight, but there is immense legroom in the choice of f_0.

Important example #2

What can we obtain with $c_1 = 0$?
i.e., a family of algebraic polynomials with coefficients bounded by a constant.

The remaining constraint rewrites simply as

$$c_0 d_1 = 1,$$

which does not say much.

Does this do anything?

If we have access to a fictitious oracle that outputs such a polynomial f_1, what does it give?

SNFS: polynomial selection with an oracle

If we have access to a fictitious oracle that outputs such a polynomial f_1, what does it give?

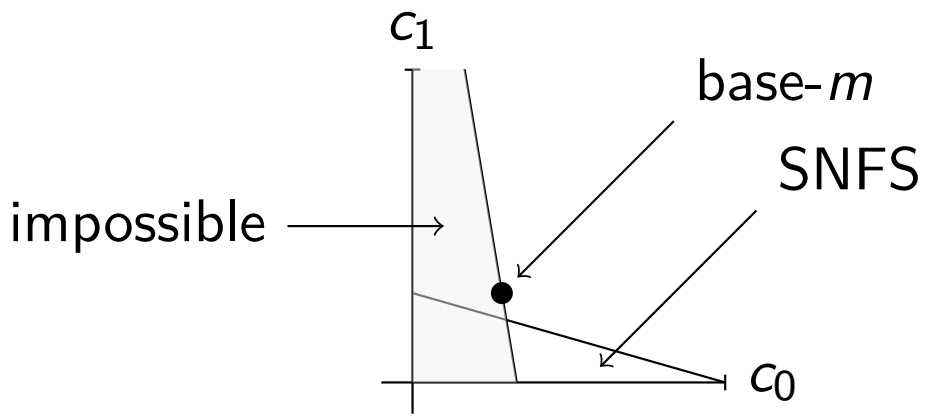
- We can do the entire NFS analysis based on that.
- The algebraic norm can be rewritten as $L_N[1/3, \cancel{\frac{1}{\delta}} + \alpha\delta]$.
- This changes the optimum δ from $\sqrt{2/\alpha}$ to $\sqrt{1/\alpha}$.
- Eventually, we end up with $L_N[1/3, (32/9)^{1/3} + o(1)]$.

This is called SNFS.

- The "special" integers are those that are precisely reached by this "ideal" choice.
- By extension, the SNFS term is also used for anything that is reached by a non-general polynomial selection.

The constraint space

Example for $d_0 = 1$ and $d_1 = 6$.



Note that $c_0 + c_1$ appears in the smoothness probability.

- $c_0 + c_1 = \log(\|f_0\| \cdot \|f_1\|)/\log N$.
- $c_0 + c_1$ measures the polynomial-dependent part of the maximum size of the integers which are checked for smoothness.

Thus the intersection point P is "ideal". Alas, moving towards P is expensive.

Where are we?

- Base-m polynomial selection is a starting point.
- We have an argument that explains that it is not "optimal".
- SNFS numbers are really special.

Plan

Introduction

Size of the coefficients: possible and impossible things

Searching for good polynomials with base-m

Skewness

Non-monic linear polynomials: Kleinjung's 2005 algorithm

What can base-m do?

Notations: $(f_0, f_1), \deg f_1 = d, f_1 = \sum a_i x^i$.

Recall that the simplistic base-m method choses $m \approx N^{1/(d+1)}$.

- There is an immense degree of freedom in the choice of m.
- Can we do many trials and hope for something nice to happen?

Opportunities for improvement:

- It is not a very big deal if $\|f_0\|$ (max coefficient of the linear polynomial) increases by a tiny bit.
- Can this be compensated by a larger decrease of $\|f_1\|$?

Base-m, revisited

Instead of picking m first, and then the coefficients of f:

- Choose a_d first, slightly smaller than $N^{1/(d+1)}$.
- Then choose m, and deduce the rest of the coefficients.

Our game: correlation between effort and yield

Ultimately, we want to answer the question:

> *"If we generate C polynomial pairs, what is the best we can obtain, as a function of C?"*

Can also be phrased as: if $a_d \approx N^{1/(d+1)}/c$, how many trials does it take to have all coefficients of f close to a_d?

Base-m, revisited

Let c be an arbitrary number.

- Choose $a_d \approx N^{1/(d+1)}/c$ (many possible choices!)
- Let $m = \lfloor (N/a_d)^{1/d} \rceil = (N/a_d)^{1/d} + \mu$ with $|\mu| \leq 1$.

Lemma: $a_{d-1} \approx a_d$

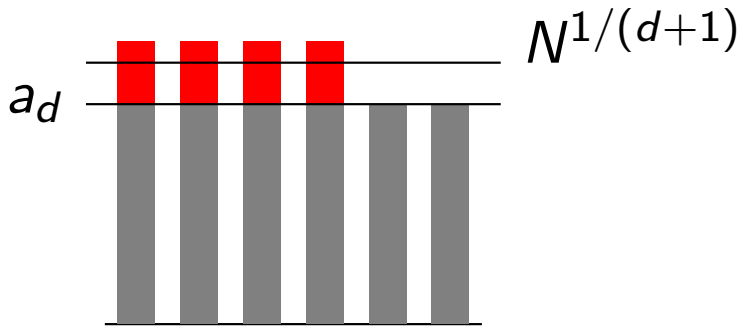
$$
\begin{aligned}
|a_{d-1}| = \frac{\left|N - a_d m^d\right|}{m^{d-1}} &= a_d \frac{\left|(m-\mu)^d - m^d\right|}{m^{d-1}} \\
&\leq a_d m \left|(1-\mu/m)^d - 1\right| \\
&\leq d a_d \times \text{small constant bound.}
\end{aligned}
$$

And d is small as well, so we expect a_{d-1} to have roughly as many bits as a_d.

Other coefficients

The $d-1$ coefficients a_0 to a_{d-2} are a priori close to m, with:

$$m \approx (N/a_d)^{1/d} \approx (N^{1-1/(d+1)}c)^{1/d} \approx N^{1/(d+1)}c^{1/d}.$$



Heuristic: a_0 to a_{d-2} behave like random integers. With probability $(a_d/m)^{d-1} \approx \left(c^{-(1/d+1)}\right)^{d-1} = c^{-(d^2-1)/d}$, all are $\leq a_d$.

Conclusion

By trying $c^{(d^2-1)/d}$ values a_d, we expect to:

- change $\|f_1\|$ to $N^{1/(d+1)}/c$.
- change $\|f_0\|$ to $N^{1/(d+1)} \times c^{1/d}$.

Rewrite

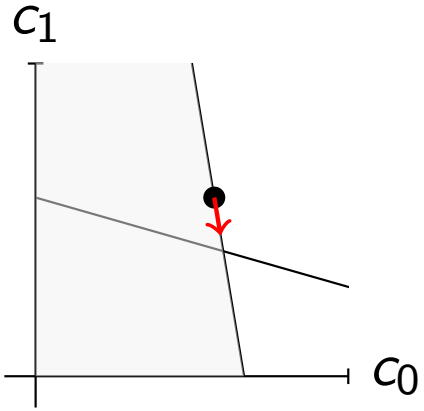
Conclusion (rewrite)

By trying $c^{(d^2-1)/d}$ values a_d, we expect to:

- change $\|f_1\|$ to $N^{1/(d+1)}/c$.
- change $\|f_0\|$ to $N^{1/(d+1)} \times c^{1/d}$.

We can also write: by trying C values a_d, we expect to:

- change $\|f_1\|$ to $N^{1/(d+1)}/C^{d/(d^2-1)}$.
- change $\|f_0\|$ to $N^{1/(d+1)} \times C^{1/(d^2-1)}$.
- change $\|f_0\|\|f_1\|$ to $N^{2/(d+1)}/C^{1/(d+1)}$.



This moves in the right direction!

- More work leads to smaller polynomials.
- This is woefully exponential, of course.

Plan

Introduction

Size of the coefficients: possible and impossible things

Searching for good polynomials with base-m

Skewness

Non-monic linear polynomials: Kleinjung's 2005 algorithm

Skewness

Notations: $(f_0, f_1), \deg f_1 = d, f_1 = \sum a_i x^i$, $\phi(x) = u - vx$.

Skewness is a way to add more flexibility to the polynomial selection process.

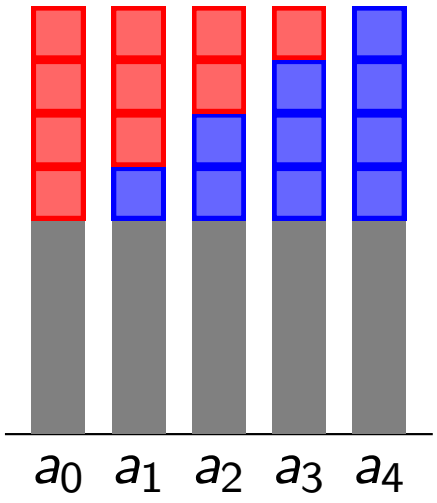
Observation: the polynomial $x - m$ is unbalanced. So is the expression $\mathrm{Res}(u - vx, x - m) = u - vm$.

- Can we work with larger u and smaller v?

Skewed polynomials

$\mathrm{Res}(u - vx, f_1) = F(u, v) = (u^d a_d + \cdots + u^i v^{d-i} a_i + \cdots + v^d a_0)$.

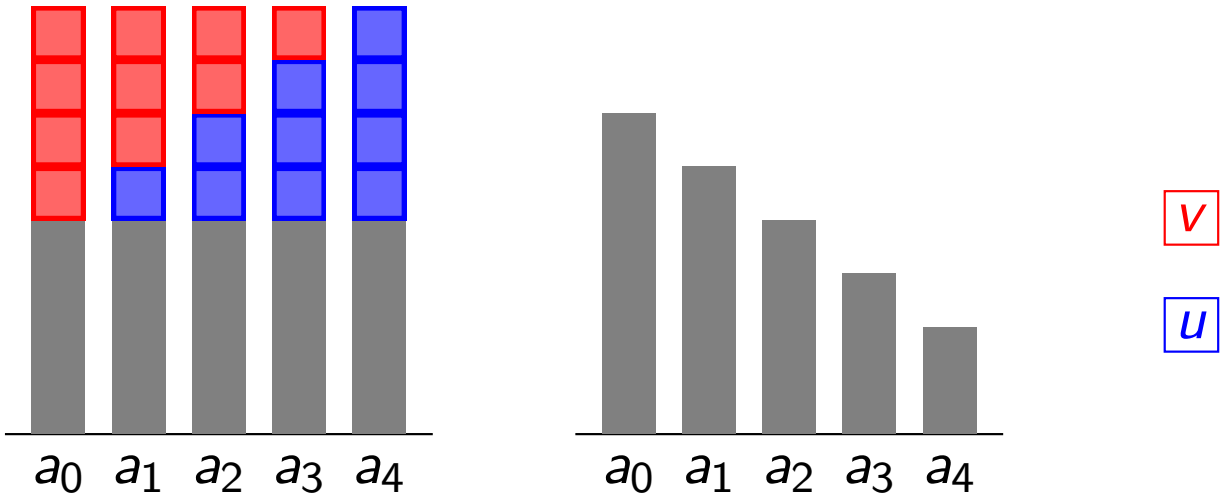
- If coefficients of f_1 have roughly the same size and both coefficients of $\phi(x)$ are bounded by A, then all $a_i u^i v^{d-i}$ have the same size.



Skewed polynomials

$\mathrm{Res}(u - vx, f_1) = F(u, v) = (u^d a_d + \cdots + u^i v^{d-i} a_i + \cdots + v^d a_0)$.

- If coefficients of f_1 have roughly the same size and both coefficients of $\phi(x)$ are bounded by A, then all $a_i u^i v^{d-i}$ have the same size.
- If the a_i are unbalanced, say $\frac{a_i}{a_{i+1}} \approx S > 1$, then with $|u| < A\sqrt{S}$ and $|v| < A/\sqrt{S}$, all $a_i u^i v^{d-i}$ have the same size.

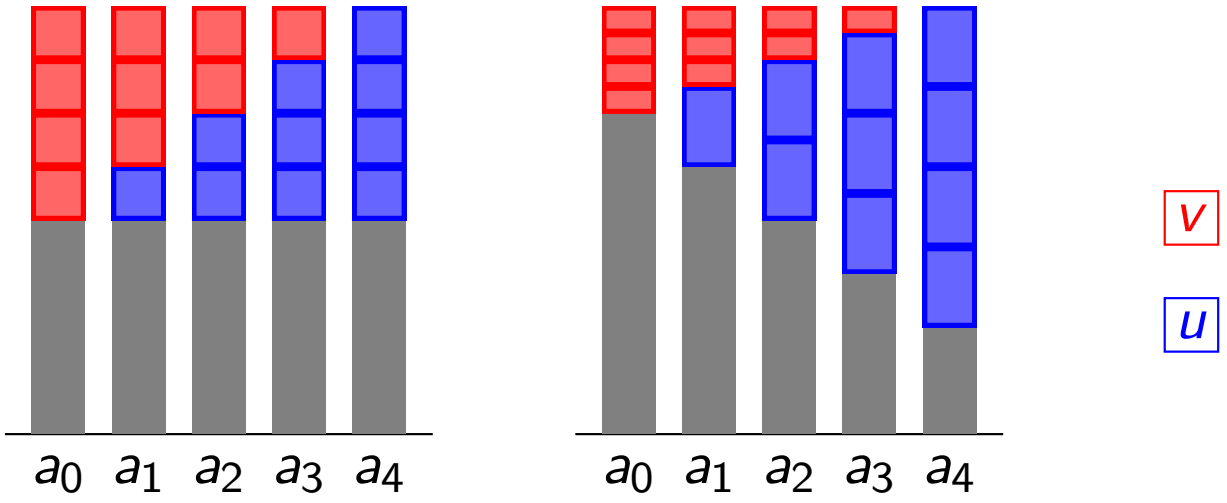


The ratio S is called the skewness; the polynomials are skewed.

Skewed polynomials

$\mathrm{Res}(u - vx, f_1) = F(u, v) = (u^d a_d + \cdots + u^i v^{d-i} a_i + \cdots + v^d a_0)$.

- If coefficients of f_1 have roughly the same size and both coefficients of $\phi(x)$ are bounded by A, then all $a_i u^i v^{d-i}$ have the same size.
- If the a_i are unbalanced, say $\frac{a_i}{a_{i+1}} \approx S > 1$, then with $|u| < A\sqrt{S}$ and $|v| < A/\sqrt{S}$, all $a_i u^i v^{d-i}$ have the same size.



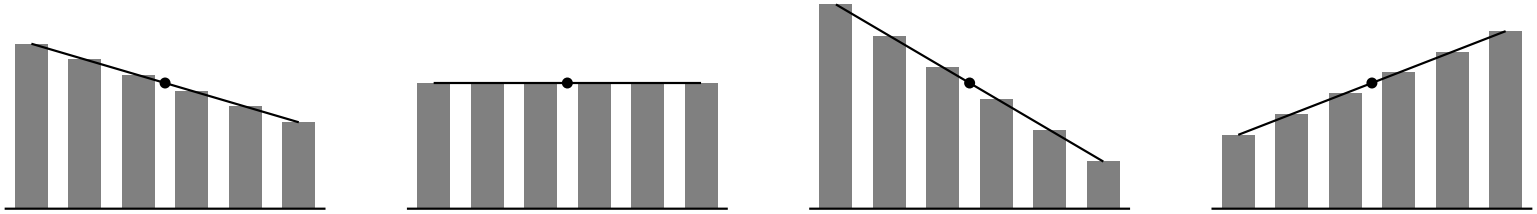
The ratio S is called the skewness; the polynomials are skewed.

Skewed norm

Definition

Given $P = \sum p_i x^i \in \mathbb{R}[x]$, the S-skewed (infinity) norm of P is:

$$\|P\|_S = \|P\|_{\infty,S} = \max_{0 \le i \le \deg P} |p_i S^{i-(\deg P)/2}|.$$



$S = 100$ $S = 1$ $S = 10000$ $S = 0.01$

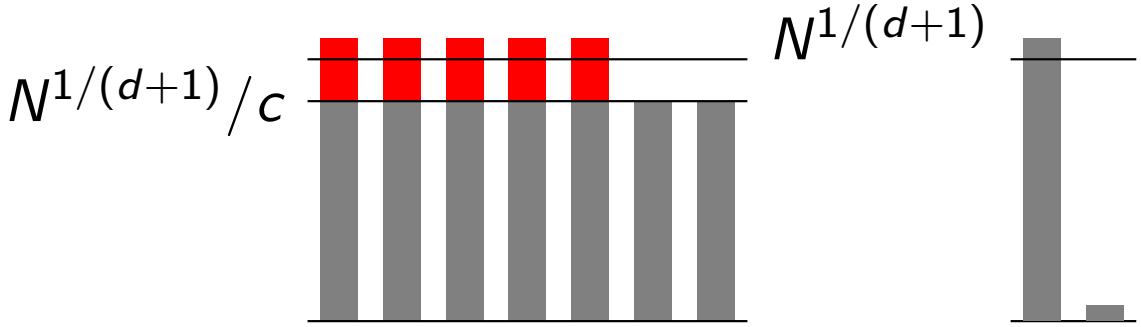
All polynomials above have the same S-skewed norms (with their respective S). If $\|P\| = \|Q\|_S$, then

$$\max\{\mathrm{Res}(u - vx, P),\ (u, v) \in [0, A]^2\}$$
$$= \max\{\mathrm{Res}(u - vx, Q), (u, v) \in [0, A\sqrt{S}] \times [0, A/\sqrt{S}]\}.$$

How do we find skewed polynomials?

When we revisited base-m, we chose a_d first, and then m.
This gave rise to:

- $a_d \approx a_{d-1} \approx N^{1/(d+1)}/c$.
- $m = \sqrt[d]{N/a_d} \geq N^{1/(d+1)} =$ the textbook base-m.

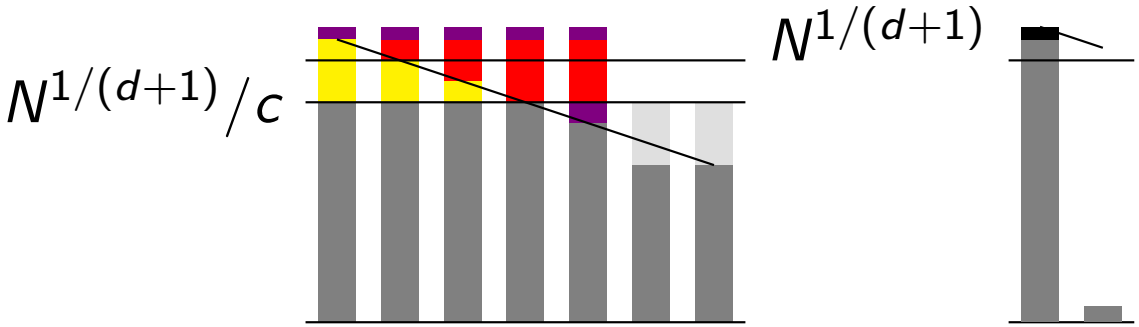


This polynomial pair is already somewhat skewed, we may turn it to our advantage.

How do we find skewed polynomials?

When we revisited base-m, we chose a_d first, and then m.
This gave rise to:

- $a_d \approx a_{d-1} \approx N^{1/(d+1)}/c$.
- $m = \sqrt[d]{N/a_d} \geq N^{1/(d+1)} =$ the textbook base-m.



This polynomial pair is already somewhat skewed, we may turn it to our advantage.

- Aim at the same skew-norm, starting from a smaller a_d (■):
  - bits we still have to cancel (■),
  - bits we no longer care about (■),
  - new bits to cancel (■),
  - a moderately larger rational norm because m got larger (■),

Analysis of skewed base-m

Analysis is a bit painful, but the outcome is quite clear:

With the same number of trials, we can expect to find smaller skewed-norms that in the non-skewed base-m.
$C^{1/(d+1)}$ is replaced by a mildly larger number

Refinements:

- Do not optimize a_0.
- Rationale: this makes it possible to form many linear combinations like $f_0 + tf_1$ and choose the best one.
  We'll get to that with root properties and root sieving.

Plan

Introduction

Size of the coefficients: possible and impossible things

Searching for good polynomials with base-m

Skewness

Non-monic linear polynomials: Kleinjung's 2005 algorithm

Better polynomials: non-monic f_0

In fact, f_0 can be non-monic:

$$f_0 = m_1 x - m_0.$$

Then, the common root modulo N must be $m = m_0/m_1$ and

$$\mathrm{Res}(f_1, m_1 x - m_0) = a_d m_0^d + a_{d-1} m_1 m_0^{d-1} + \cdots + a_0 m_1^d.$$

Remark: if the latter is equal to N, it implies

$$a_d m_0^d \equiv N \mod m_1.$$

First ingredient of Kleinjung's algorithms (2006 and 2008) is called Kleinjung "Lemma 2.1". It computes a reasonably good f_1 from a fixed choice of N, d, m_1, m_0 and a_d.

Kleinjung "Lemma 2.1"

Input: N, d, m_1, m_0, and some fixed coefficients $[a_j, \ldots, a_d]$

Output: A polynomial f_1 such that $\mathrm{Res}(f_1, m_1 x - m_0) = N$

First, compute

$$r_j = \frac{N - \sum_{i=j+1}^{d} a_i m_0^i m_1^{d-i}}{m_1^{d-j}}$$

Then, for $i = j-1, j-2, \ldots, 0$, compute:

- $r_i = \dfrac{r_{i+1} - a_{i+1} m_0^{i+1}}{m_1}$
- $a_i = \dfrac{r_i + t_i m_1}{m_0^i}$, where t_i is an integer such that

$$-\frac{m_0^i}{2} \le t_i < \frac{m_0^i}{2} \text{ and } t_i \equiv -\frac{r_i}{m_1} \bmod m_0^i$$

The output is $f_1 = a_d x^d + a_{d-1} x^{d-1} + \cdots + a_1 x + a_0$.

Kleinjung "Lemma 2.1" – example

Kleinjung's "Lemma 2.1" algorithm applied to RSA-155 with $d = 5$ and

$$\begin{aligned} a_5 &= 358870426380 \\ m_0 &= 31392776870911769459515309198 \\ m_1 &= 823916492006383 \end{aligned}$$

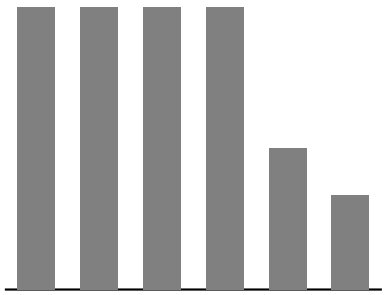
gives:

$$\begin{aligned} f_1 &= 358870426380x^5 \\ &+ 428308592054328x^4 \\ &- 1633687767207251072315485199 6x^3 \\ &- 1260161138731810732800612211 8x^2 \\ &- 196218554995115238458453047 51x \\ &- 836976378549559598530450289 9 \end{aligned}$$

Output of Kleinjung "Lemma 2.1"

If we apply Kleinjung "Lemma 2.1" with only the leading coefficient a_d fixed and with m_0 close to $\tilde{m}_0 = \sqrt[d]{\frac{N}{a_d}}$, the algorithm yields:

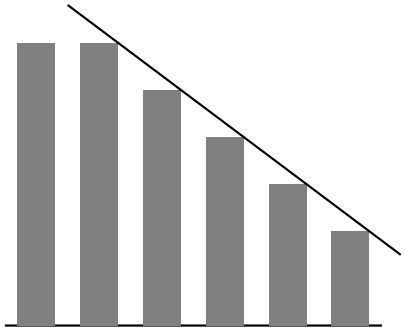
- a_{d-1} rather small: $|a_{d-1}| < m_1 + d a_d \frac{m_0 - \tilde{m}_0}{m_1}$.
- Other a_i's satisfy $|a_i| < m_1 + m_0$.



Output of Kleinjung "Lemma 2.1"

If we apply Kleinjung "Lemma 2.1" with only the leading coefficient a_d fixed and with m_0 close to $\tilde{m}_0 = \sqrt[d]{\frac{N}{a_d}}$, the algorithm yields:

- a_{d-1} rather small: $|a_{d-1}| < m_1 + d a_d \frac{m_0 - \tilde{m}_0}{m_1}$.
- Other a_i's satisfy $|a_i| < m_1 + m_0$.



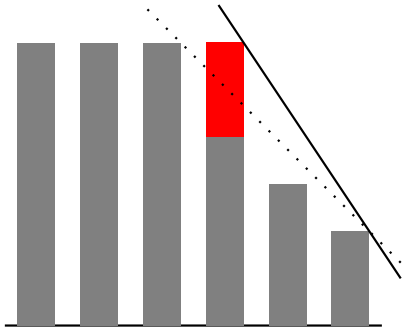
Our goal, and how we reach it

Have coefficient sizes which are a good match to some skewness.

Output of Kleinjung "Lemma 2.1"

If we apply Kleinjung "Lemma 2.1" with only the leading coefficient a_d fixed and with m_0 close to $\tilde{m}_0 = \sqrt[d]{\frac{N}{a_d}}$, the algorithm yields:

- a_{d-1} rather small: $|a_{d-1}| < m_1 + d a_d \frac{m_0 - \tilde{m}_0}{m_1}$.
- Other a_i's satisfy $|a_i| < m_1 + m_0$.



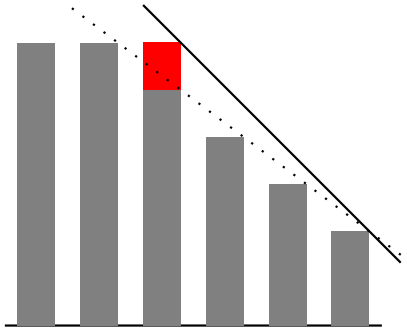
Our goal, and how we reach it

Have coefficient sizes which are a good match to some skewness.

Output of Kleinjung "Lemma 2.1"

If we apply Kleinjung "Lemma 2.1" with only the leading coefficient a_d fixed and with m_0 close to $\tilde{m}_0 = \sqrt[d]{\frac{N}{a_d}}$, the algorithm yields:

- a_{d-1} rather small: $|a_{d-1}| < m_1 + d a_d \frac{m_0 - \tilde{m}_0}{m_1}$.
- Other a_i's satisfy $|a_i| < m_1 + m_0$.



Our goal, and how we reach it

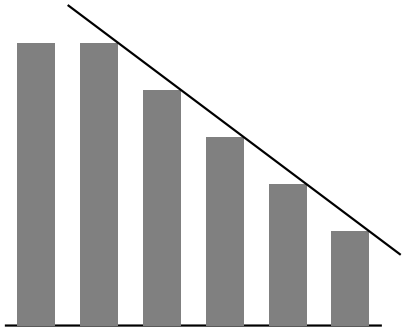
Have coefficient sizes which are a good match to some skewness.

- Find smart way to make a_{d-2} small.

Output of Kleinjung "Lemma 2.1"

If we apply Kleinjung "Lemma 2.1" with only the leading coefficient a_d fixed and with m_0 close to $\tilde{m}_0 = \sqrt[d]{\frac{N}{a_d}}$, the algorithm yields:

- a_{d-1} rather small: $|a_{d-1}| < m_1 + d a_d \frac{m_0 - \tilde{m}_0}{m_1}$.
- Other a_i's satisfy $|a_i| < m_1 + m_0$.



Our goal, and how we reach it

Have coefficient sizes which are a good match to some skewness.

- Find smart way to make a_{d-2} small.
- Rely on (mild) luck to make a_{d-3} small.

Making a_{d-2} small

Use the equation $a_d m_0^d \equiv N \mod m_1$.

Key idea

Build m_1 as a product of small primes. Use the combination of modular information to fabricate a small a_{d-2}.

- Let $\mathcal{P}$ be a set of small primes $\equiv 1 \mod d$ (m_1 will be a product of a subset of $\mathcal{P}$).
- Pick some a_d (e.g. smooth).
- Some primes $r \in \mathcal{Q} \subset \mathcal{P}$ give d solutions to $a_d x^d \equiv N \mod r$.
- Any choice of exactly one d-th root modulo each of those r's gives a value m_0 defined modulo $m_1 = \prod r$ by CRT.
  We may choose one which is close to $\tilde{m}_0 = \sqrt[d]{N/a_d}$.

Making a_{d-2} small

Many choices

Pick ℓ primes for which $a_d x^d \equiv N \mod r$ has d solutions.

- In total, d^ℓ possible choices for m_0.
- m_0 is a linear combination of ℓ values among $d \times \ell$.
  This follows from explicit Chinese Remainder Theorem.

Expand the value a_{d-2}/m_0 obtained by "Lemma 2.1" from the d-th roots of $m_0 \mod r$ that we have chosen.

- By restricting to 1st order terms, we get a linear combination.
- If a_{d-2}/m_0 ends up being close to an integer λ for some chosen m_0, then for $f_1' = f_1 - \lambda(m_1 x - m_0)x^{d-2}$, we have:
  - a'_{d-2}/m_0 close to 0,
  - a'_{d-1} does not change much.

Finding combinations that are close to $\mathbb{Z}$

The problem can be reduced to the following:

- ℓ sets $S_1, \ldots, S_\ell$, each containing d real numbers in $[0, 1)$.
- d^ℓ choices: $(s_1, \ldots, s_\ell)$ with each $s_i \in S_i$, and:

$$a_{d-2}/m_0 \bmod \mathbb{Z} \equiv \sum s_i.$$

- Naive complexity: $O(d^\ell)$.
- Better:

Finding combinations that are close to $\mathbb{Z}$

The problem can be reduced to the following:

- ℓ sets $S_1, \ldots, S_\ell$, each containing d real numbers in $[0, 1)$.
- d^ℓ choices: $(s_1, \ldots, s_\ell)$ with each $s_i \in S_i$, and:

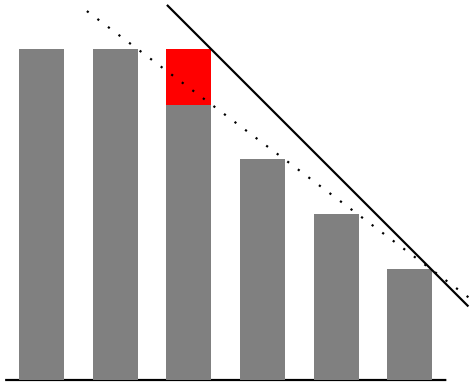
$$a_{d-2}/m_0 \bmod \mathbb{Z} \equiv \sum s_i.$$

- Naive complexity: $O(d^\ell)$.
- Better: $O(d^{\ell/2})$.

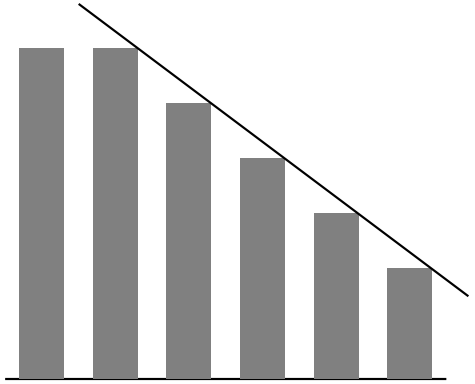
What do small combinations give ?

Algorithm has:

- a_d chosen small.
- a_{d-1} small by construction, $\approx m_1$.
- a_{d-2} small thanks to small combinations.



With some extra luck,
a_{d-3} may be somewhat smaller than expected.



CSE291-14: The Number Field Sieve

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

Emmanuel Thomé

February 1, 2022

Part 4b

Polynomial selection in NFS

Kleinjung's 2008 algorithm

Size optimization with translation and rotation

Plan

Kleinjung's 2008 algorithm

Size optimization with translation and rotation

Kleinjung's 2008 algorithm

New polynomial selection algorithm, presented at a workshop in Nancy in 2008.

- Better than the previous algorithm.
- Has never been published.
- Implemented in Cado-NFS.

Some features of this algorithm:

- A new way to make a_{d-2} small by construction.
- Yields very large skewness values.
- Large skewness is great for root-optimization (see later).

Handy tool

How to find polynomials pairs of the form:

$$f_1 = x^d + a_{d-2}x^{d-2} + \cdots + a_0,\ f_0 = m_1 x - m_0,\ \mathrm{Res}(f_0, f_1) = N.$$

Remark: we are looking for $a_d = 1$ and $a_{d-1} = 0$.

We want a_{d-2} small. Notation: $R = a_{d-2}m_0^{d-2} + \cdots + a_0 m_1^{d-2}$, so

$$N = \mathrm{Res}(f_0, f_1) = f(m_0/m_1)m_1^d = m_0^d + m_1^2 R$$

Reformulation

Put otherwise, given (N, d) we want (m_1, m_0) such that:

$$m_1^2 \mid (N - m_0^d), \quad \frac{\left|N - m_0^d\right|}{m_1^2 m_0^{d-2}} = \frac{|R|}{m_0^{d-2}} \approx a_{d-2} \text{ small.}$$

Divisibility + $m_0 \approx \sqrt[d]{N}$ only give $R \approx d m_0^{d-1} m_1 / m_1^2 \approx$ not small!

Solving the auxiliary problem

Set $\bar{m}_0 = \left\lceil \sqrt[d]{N} \right\rceil$. Good m_0's are close to $\bar{m}_0$.

- Let $\mathcal{P} = [P, 2P]$ be a range of prime numbers.
- List pairs

$$(p \in \mathcal{P}, i \in [-4P^2, +4P^2]) \text{ s.t. } p^2 \mid (N - (\bar{m}_0 + i)^d).$$

- Sort w.r.t. i. Find collisions.
- Each collision $(p_1, i), (p_2, i)$ yields $m_1 = p_1 p_2,\ m_0 = \bar{m}_0 + i$.

Applying Kleinjung "Lemma 2.1" to N, d, m_1, m_0, a_d and $a_{d-1} = 0$ gives:

$$|a_{d-2}| \leq \frac{4dm_0}{P^2} + 2P^2.$$

With this approach, we can obtain polynomials with $a_d = 1$ and $a_{d-1} = 0$. But this isn't so appealing.

More general situation

How can we apply previous tool to more general polynomials ?

$$f_1 = a_d x^d + a_{d-1} x^{d-1} + \cdots, \; f_0 = m_1 x - m_0, \; \mathrm{Res}(f_0, f_1) = N.$$

Goal: $N = m_1^d f(m_0/m_1) = a_d m_0^d + a_{d-1} m_0^{d-1} m_1 + m_1^2 R$ and $\frac{|R|}{m_0^{d-2}}$ small.

Consider the polynomial $f_1(x - \frac{a_{d-1}}{d a_d})$.

$$d^d a_d^{d-1} N = (d a_d m_0 + a_{d-1} m_1)^d + m_1^2 \left(d^d a_d^{d-1} R - \cdots \right).$$

Reduction to previous problem

Define $N' = d^d a_d^{d-1} N$, and $m_0' = (d a_d m_0 + a_{d-1} m_1)$.

- Fix a_d and compute N'.
- Search for m_1, m_0' as solutions to previous problem.
- From a winning m_0', find appropriate m_0 and a_{d-1}.

Time-consuming steps

When running Kleinjung's 2008 algorithm, a computationally expensive part is the search for collisions.

- Naive:
  - list many pairs (p, i).
  - sort w.r.t. i, and see if we have (p_1, i) and (p_2, i).
- In practice:
  - Generate many pairs. Dispatch only the i values in lists indexed by, e.g. $\lfloor i/2^{16} \rfloor$:
    `H[i // 2^16].append(i % 2^16)`
  - For each list, hit an array of 2^{16} memory bytes with the values in the list, and report if a common i is found.
  - When we do find a collision (a priori rarely), start over more cautiously.
  - And then there's possibly some extra work to do once we find (p_1, p_2) too.

Time-consuming steps

Another avenue for improvement reflects on the modular computations of the roots of $N - x^d \mod p_i$.

This can be amortized with a technique that reminds a bit of the special-q technique. Aim at $m_1 = p_1 p_2 q$ with fixed q.

Plan

Kleinjung's 2008 algorithm

Size optimization with translation and rotation

Size optimization – main idea

A "raw" polynomial pair (i.e., generated by one of the previous algorithms) may be optimized:

- translation: change (f_0, f_1) into
$$(f_0(X+t), f_1(X+t)),\ t \in \mathbb{Z}.$$
- rotation: change (f_0, f_1) into
$$(f_0, f_1 + rf_0),\ r \in \mathbb{Z}[X] \text{ with } \deg r < \text{some bound}.$$
This is subject to multiple constraints that depend on the polynomial pair we started with.

Goal of size-optimization

Given (f_0, f_1), find $t \in \mathbb{Z}$ and $r \in \mathbb{Z}[X]$ of degree less than d that minimize some size estimator for $f_1(X+t) + rf_0(X+t)$.

Remark: $\mathrm{Res}(f_1(X+t) + rf_0(X+t), f_0(X+t)) = \mathrm{Res}(f_0, f_1)$.

Size optimization – local descent

Hard to find the global minimum.

We have to settle for a local minimum.

Algo: start with the raw pair (f_0, f_1) and apply a local descent algorithm:

- apply a small translation or small rotation that reduce the norm;
- repeat until a (local) minimum is reached.

Works fine for $d \leq 5$.

Implemented in Cado-NFS. Used as a building block for others algorithms.

Size optimization – larger degree

For larger degrees, the local descent algorithm is often stuck in minima close to the starting polynomial pair.

Ideas for improvment:

- Apply some initial translations before calling the local descent algorithm to increase the number of starting polynomial pairs and to avoid being stuck in local minima too far away from the global minimum.
- Apply the LLL algorithm before calling the local descent algorithm to pre-optimize the starting polynomial pair.

Size optimization – initial translations

How to choose the initial translations ?

Choose integer approximations of roots of $\tilde{a}_{d-3}$, where the $\tilde{a}_i$'s are polynomials in t defined by

$$f_1(X+t) = \sum_{i=0}^{d} \tilde{a}_i(t) X^i.$$

The polynomial $\tilde{a}_{d-3}$ has degree 3 so it has at least one real root.

Example: for $d = 6$, $\tilde{a}_3(t) = 20a_6t^3 + 10a_5t^2 + 4a_4t + a_3$.

In Cado-NFS, other methods are also used:

- constants values: $i \times 10^j$ for small integers i and j
- more details in "Better polynomials for GNFS" and in the source code.

Optimization with LLL: the BBKZ algorithm

(Bai, Bouvier, Kruppa, Zimmemann)

Idea: Use the LLL algorithm to search for short vectors in the lattice spanned by the coefficients vector (of length $d+1$) of $f_1, f_0, Xf_0, X^2 f_0, \ldots$.

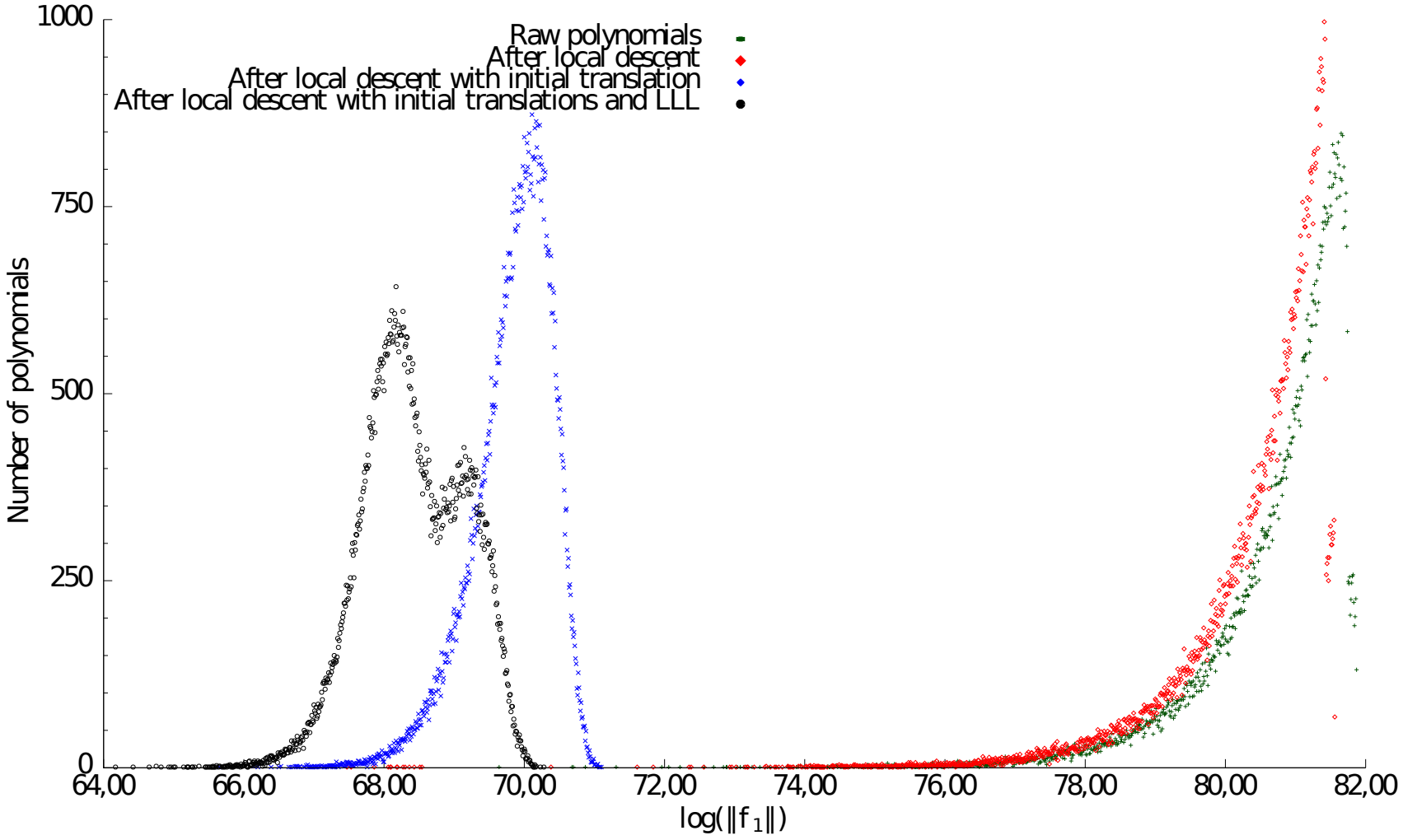
A vector of this lattice corresponds to a polynomial of the form $cf_1 + rf_0$, with $c \in \mathbb{Z}$ and r an integer polynomial.

New degree of freedom: will output polynomial pair $(\tilde{f}_0, \tilde{f}_1)$ such that $\mathrm{Res}(\tilde{f}_0, \tilde{f}_1) = c\,\mathrm{Res}(f_0, f_1)$. With previous methods, we always had $c = 1$.

This new method is used before the local descent algorithm and after the computation of the initial translations.

New initial translations can be computed to take advantage of this new degree of freedom.

Results – RSA-768 ($d = 6$)



CSE291-14: The Number Field Sieve

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

Emmanuel Thomé

February 1, 2022

Part 4c

Polynomial selection in NFS: estimators

The sieved range

Valleys and the starfish picture

Root properties

Forcing good $\alpha(f)$: the root optimization

Polynomial selection in Cado-NFS

Recap from last time

So far, we've only been interested in the size of the coefficients of our polynomials.

(skewed) (infinity) norm of a polynomial $f = \sum a_i x^i$

$$\|f\|_S = \|f\|_{\infty,S} = \max_i |a_i S^{i-(\deg f)/2}|.$$

It is ok as a first estimator, but can we do better?

- run the Number Field Sieve. It's costly.
- run a few sieving experiments.
  Could be well-suited to ~ 10 contenders.
- use other estimators, accurate/slow ones vs crude/fast ones.

Polynomial selection workplan

- Use our favorite method to produce polynomials with small coefficients, that is small (skewed) infinity norm. Use size-optimization on each pair.
- Collect many of them (thousands, millions), compute some slightly more accurate quality estimator. Rank the results.
- Perhaps refine the results with an even more accurate estimator.
- Perhaps run another range of optimizations, if relevant.
- Eventually keep about a dozen polynomials, and do sieving tests on each.

Plan

The sieved range

Valleys and the starfish picture

Root properties

Forcing good $\alpha(f)$: the root optimization

Polynomial selection in Cado-NFS

Notations

Notations: $(f_0, f_1), \deg f_1 = d$, $\phi(x) = a - bx$, coefficients of f_i unnamed.

The sieved values

Reminder

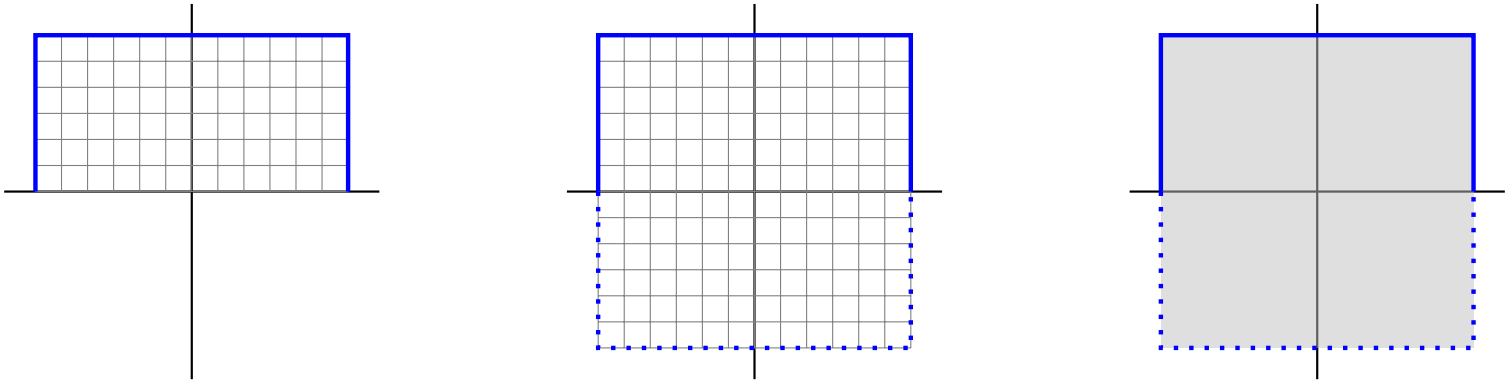
Given polynomials (f_0, f_1), we search for pairs (a, b) such that:

- $|a| \leq A$ and $0 \leq b \leq A$
  (no need to examine both (a, b) and $(-a, -b)$)
- For both $i = 0$ and 1, $\mathrm{Res}(a - bx, f_i) = F_i(a, b)$ is smooth.

We are interested in the values taken by the bivariate, homogenous polynomials F_0 and F_1:

- on the sieve region $[-A, A] \times [0, A]$.
- or more simply on $[-A, A]^2$, since the two problems are equivalent.
- Yes, we're losing track of the distinction between $\mathbb{Z}^2$ and $\mathbb{R}^2$.

Sieved range



More estimations

Count the smoothness probability for all norms obtained from $[-A, A]^2$ (want to maximize):

$$\frac{6}{\pi^2} \iint_{[-A,A]^2} \rho\left(\frac{\log|F_0(x,y)|}{\log B_1}\right) \rho\left(\frac{\log|F_1(x,y)|}{\log B_2}\right) \mathrm{d}x\,\mathrm{d}y.$$

ρ is the Dickman function. It is used to estimate the smoothness probability.

B_0 and B_1 are the smoothness bounds.

This adds the idea of averaging the values over the sieve region.

- It is more accurate than the infinity norm.
- Alas, it is **too costly**. No explicit formula for the function ρ.

L^2 rectangular norm

We want to have some notion of average norm over $[-A, A]^2$.

By homogeneity, $F_i(\lambda X, \lambda Y) = \lambda^{\deg f_i} F_i(X, Y)$.
Only $[-1, 1] \times [-1, 1]$ really matters.

L^2 norm

For F being either F_0, F_1, or the product of both, we want to minimize:

$$\iint\limits_{[-A,A]^2} F(x, y)^2 \, \mathrm{d}x \, \mathrm{d}y = A^{2 \deg F} \iint\limits_{[-1,1]^2} F(x, y)^2 \, \mathrm{d}x \, \mathrm{d}y$$

This L^2-norm is much, much easier to compute.
We use as our second main criterion of selection.

Computing the L^2 rectangular norm

```
var('X Y')
R.<x> = PolynomialRing(ZZ)
IPR.<a> = InfinitePolynomialRing(R)
d = 6;
f = SR(sum(a[i]*x^i for i in range(d+1)))
F = (f(x=X/Y)*Y^d).expand()
v = integrate(integrate(F^2,(X,-1,1)),(Y,-1,1)).expand()
v.coefficients()
```

This L^2 rectangular norm is expressed as a simple expression involving the coefficients.

Sieved range vs skewness

If we have skewness S

- $|a| < A\sqrt{S}$ and $0 \leq b < A/\sqrt{S}$
- We want $\mathrm{Res}(a - bx, f_i) = F_i(a, b)$ to be smooth.

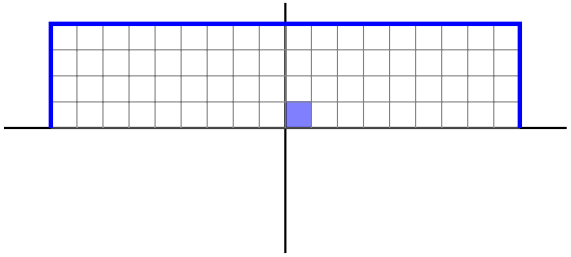
Let $F_{i,S}(X, Y) = F_i(X\sqrt{S}, Y/\sqrt{S}) \in \mathbb{R}[X, Y]$.

Again, looking at $F_{i,S}$ over $[-1, 1]^2$ is enough.

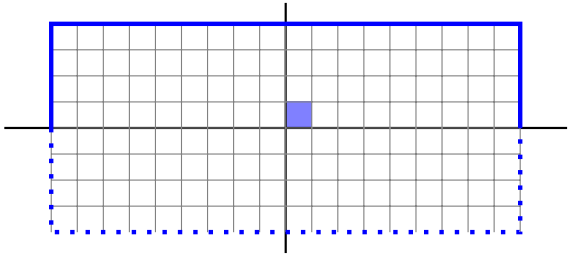
A squeeze mapping brings us back to the non-skewed case.
This will be implicit from now on.

Squeeze mapping

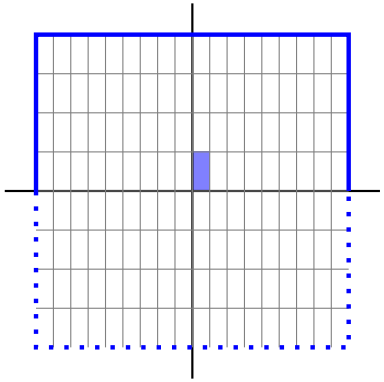
$[-A\sqrt{S}, A\sqrt{S}] \times$
$[0, A/\sqrt{S}]$

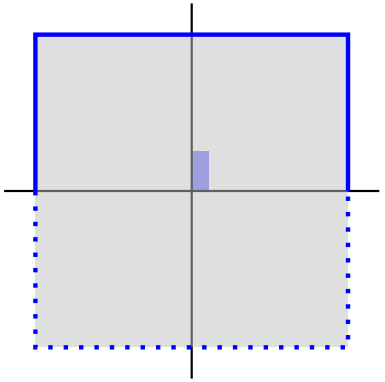


$[-A\sqrt{S}, A\sqrt{S}] \times$
$[-A/\sqrt{S}, A/\sqrt{S}]$



Squeeze by $\begin{pmatrix} 1/\sqrt{S} & 0 \\ 0 & \sqrt{S} \end{pmatrix}$: the unit area maps to area 1.





A digression: special-q vs sieved range

When special-q comes into play, something changes.

Quick refresh about special-q

Special-q is this idea of restricting our attention to situations where there is a known factor q in the things that we want to be smooth.

We can use special-q:

- to force a factor q in the factorization of $am_1 - bm_0$.
- or to force a certain prime ideal above q to appear in the factorization of $a - b\alpha$.

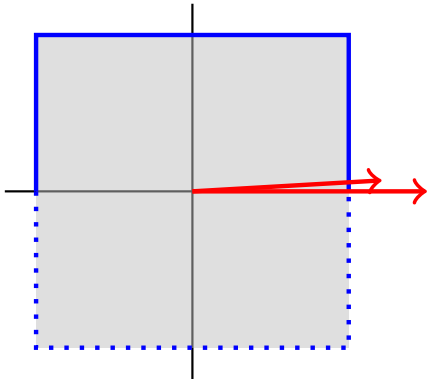
Special-q

How do we force q to divide $am_1 - bm_0$?

Here is a lattice

The set of (a, b) such that $q \mid am_1 - bm_0$ is a lattice (say $\mathcal{L}_q$).

Basis (if $\gcd(m_1, q) = 1$): $\begin{cases} (a_0, b_0) = (q, 0) \\ (a_1, b_1) = ((m_0/m_1) \bmod q, 1) \end{cases}$



combinations of the basis vectors
quickly get out of hand.

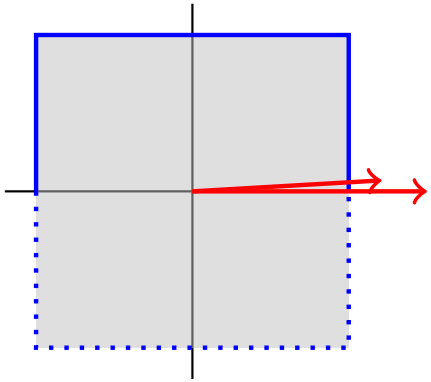
Special-q

How do we force q to divide $am_1 - bm_0$?

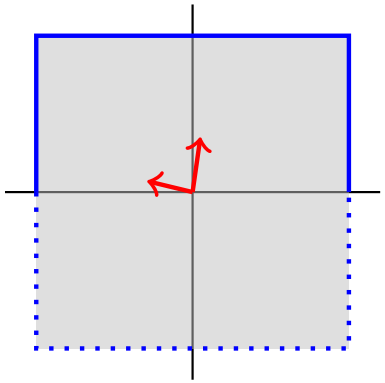
Here is a lattice

The set of (a, b) such that $q \mid am_1 - bm_0$ is a lattice (say $\mathcal{L}_q$).

Basis (if $\gcd(m_1, q) = 1$): $\begin{cases} (a_0, b_0) = (q, 0) \\ (a_1, b_1) = ((m_0/m_1) \bmod q, 1) \end{cases}$



combinations of the basis vectors quickly get out of hand.

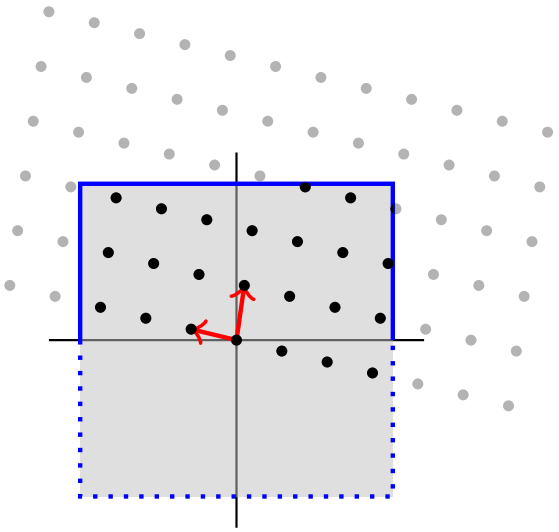


lattice reduction keeps things under control.

This approach is called lattice sieving.

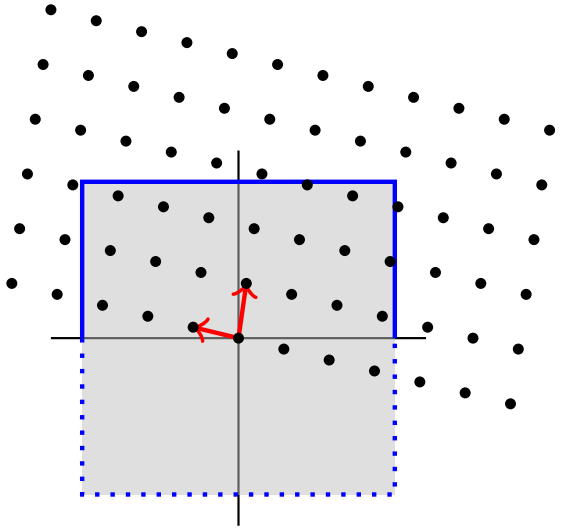
Sieved range with lattice sieving

- Choose q.
- Compute a reduced basis of the lattice $\mathcal{L}_q$.
- Sieve through all the linear combinations of the basis vectors that fall within the target region.
  - Loop through all primes,
  - Mark the locations where they divide, etc.



Sieved range with lattice sieving: better

- Fix in advance a set of combinations that we will explore each time. E.g. $[-2^{15}, 2^{15}) \times [0, 2^{15})$.
- Choose q.
- Compute a reduced basis of the lattice $\mathcal{L}_q$.
- Sieve through all the linear combinations of the basis vectors, whether or not they fit in our target range.



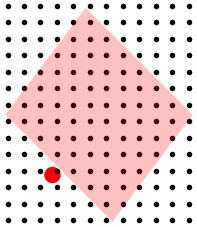
Why would we do that?

Legitimate fear: the set of (a, b) that we will sieve go way off range. But would they, really?

Sieving rectangle

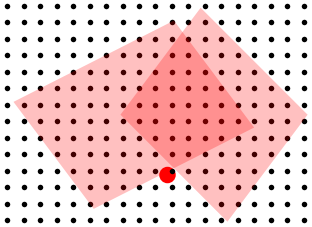
Sieving rectangle

Here, we depict the area of size $2^l \times 2^{l-1}$ for many q's.



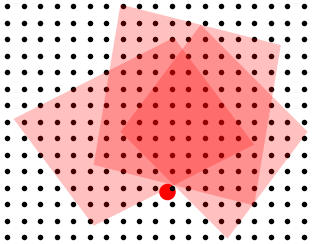
Sieving rectangle

Here, we depict the area of size $2^l \times 2^{l-1}$ for many q's.



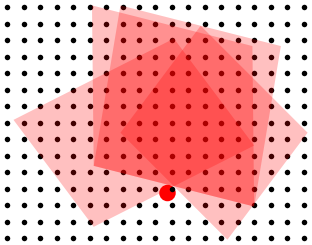
Sieving rectangle

Here, we depict the area of size $2^l \times 2^{l-1}$ for many q's.



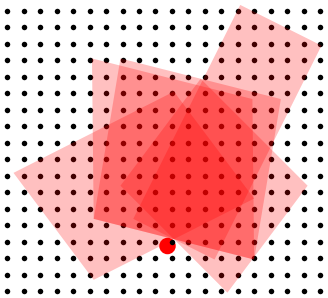
Sieving rectangle

Here, we depict the area of size $2^l \times 2^{l-1}$ for many q's.



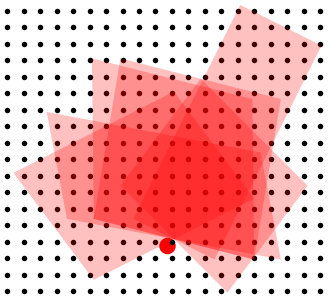
Sieving rectangle

Here, we depict the area of size $2^l \times 2^{l-1}$ for many q's.



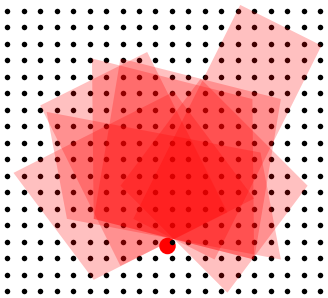
Sieving rectangle

Here, we depict the area of size $2^l \times 2^{l-1}$ for many q's.



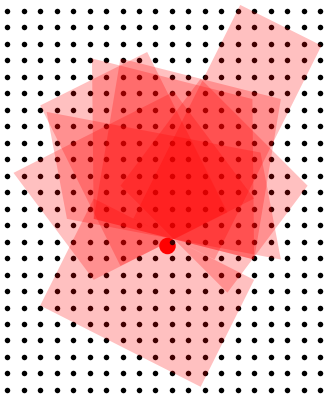
Sieving rectangle

Here, we depict the area of size $2^l \times 2^{l-1}$ for many q's.



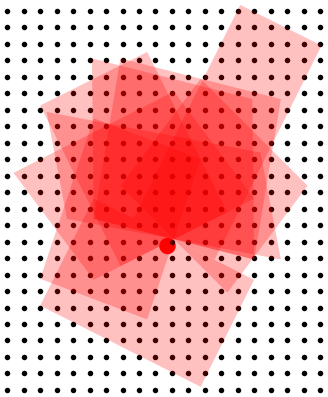
Sieving rectangle

Here, we depict the area of size $2^l \times 2^{l-1}$ for many q's.



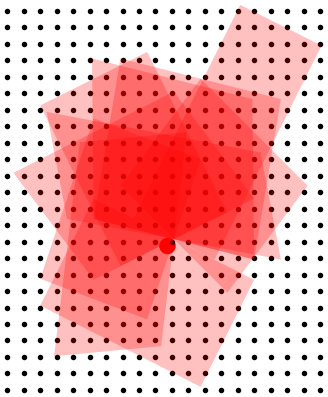
Sieving rectangle

Here, we depict the area of size $2^l \times 2^{l-1}$ for many q's.



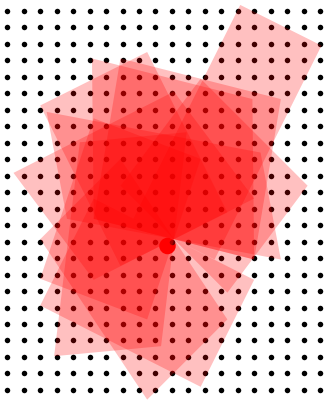
Sieving rectangle

Here, we depict the area of size $2^l \times 2^{l-1}$ for many q's.



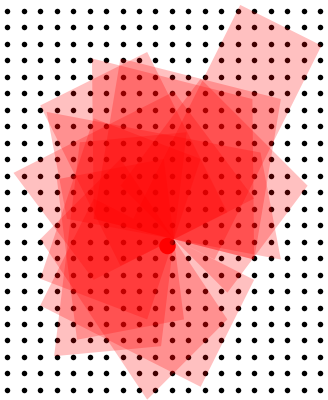
Sieving rectangle

Here, we depict the area of size $2^l \times 2^{l-1}$ for many q's.



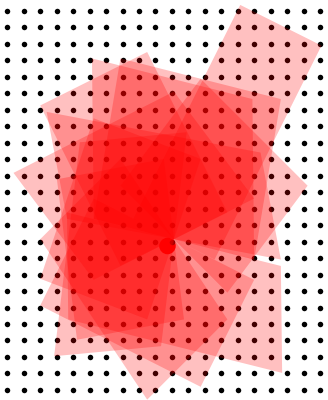
Sieving rectangle

Here, we depict the area of size $2^l \times 2^{l-1}$ for many q's.



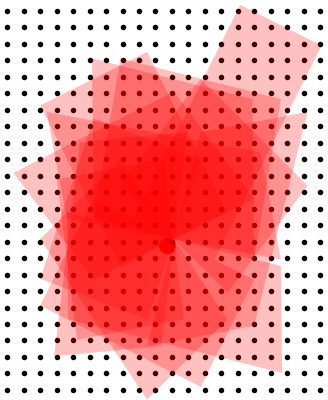
Sieving rectangle

Here, we depict the area of size $2^l \times 2^{l-1}$ for many q's.



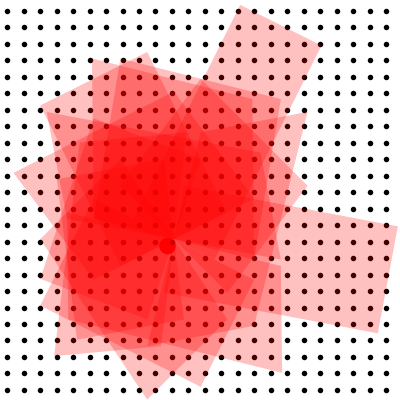
Sieving rectangle

Here, we depict the area of size $2^l \times 2^{l-1}$ for many q's.



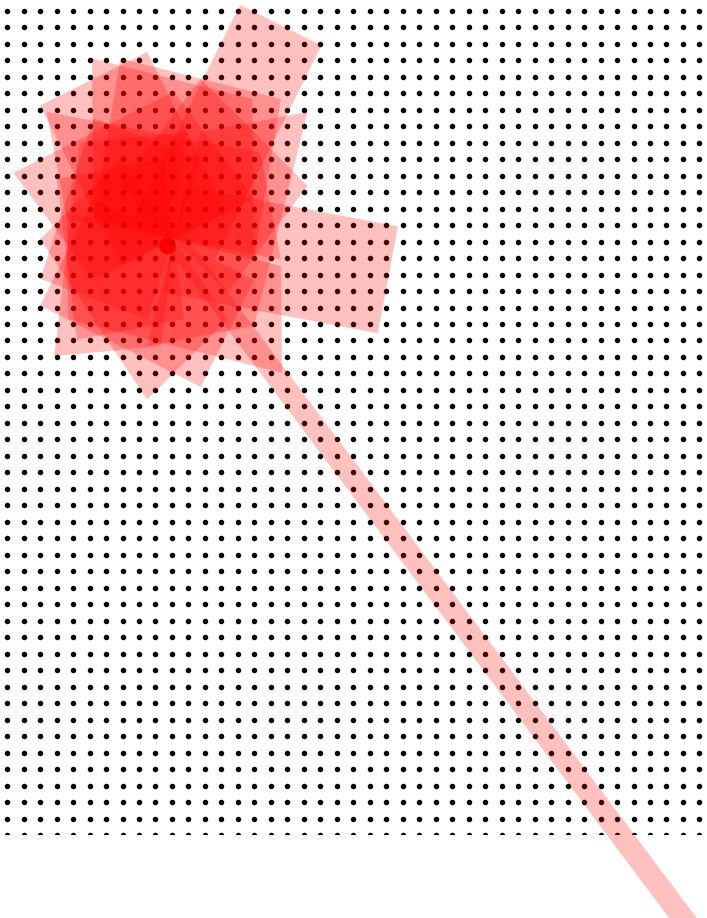
Sieving rectangle

Here, we depict the area of size $2^l \times 2^{l-1}$ for many q's.



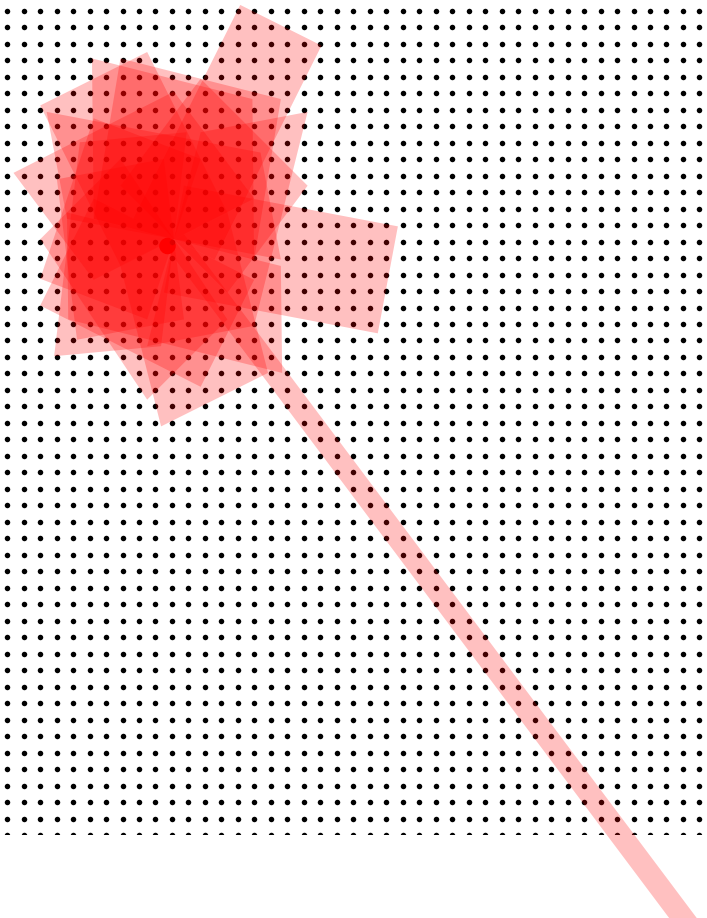
Sieving rectangle

Here, we depict the area of size $2^l \times 2^{l-1}$ for many q's.



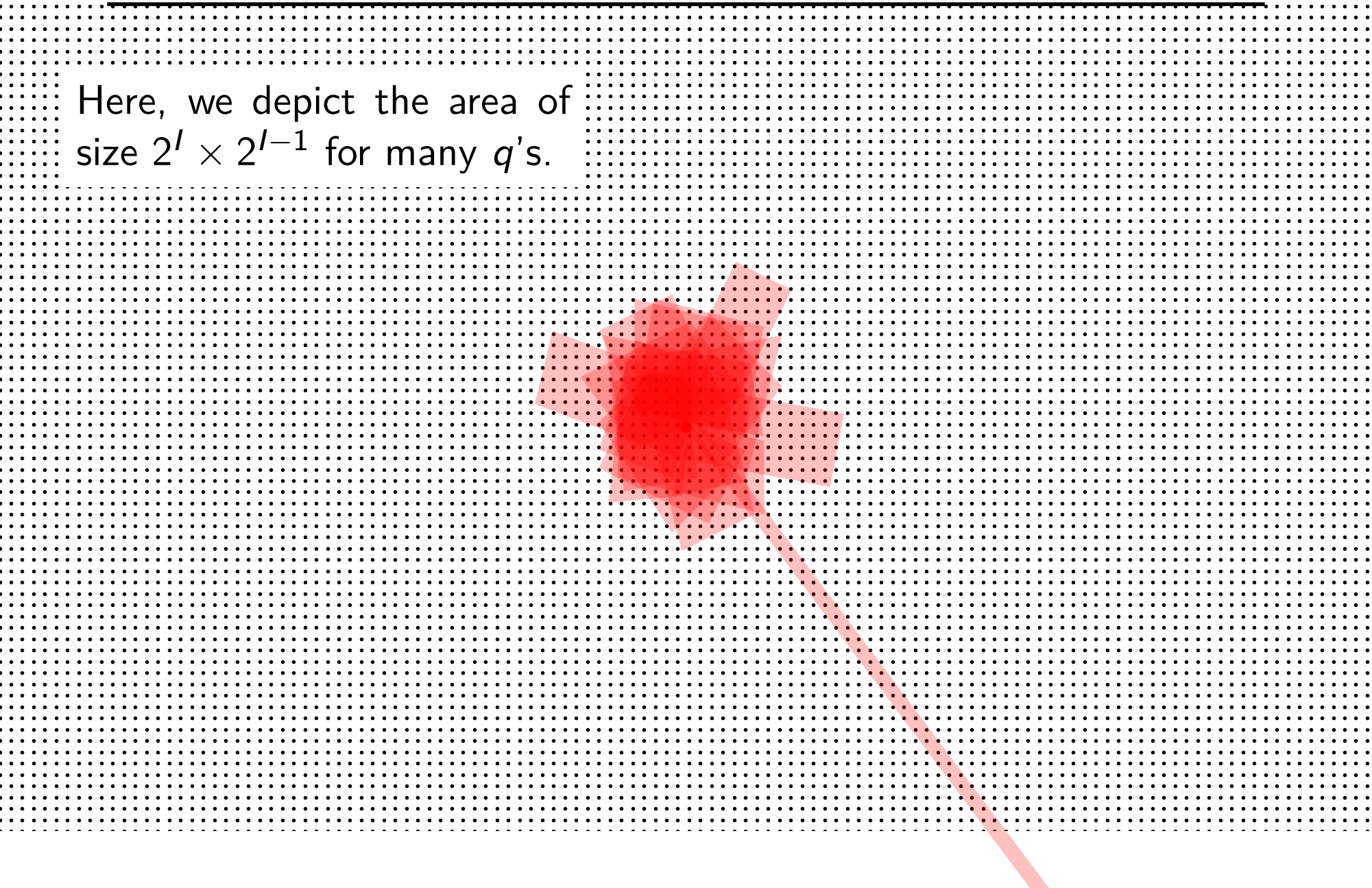
Sieving rectangle

Here, we depict the area of size $2^l \times 2^{l-1}$ for many q's.



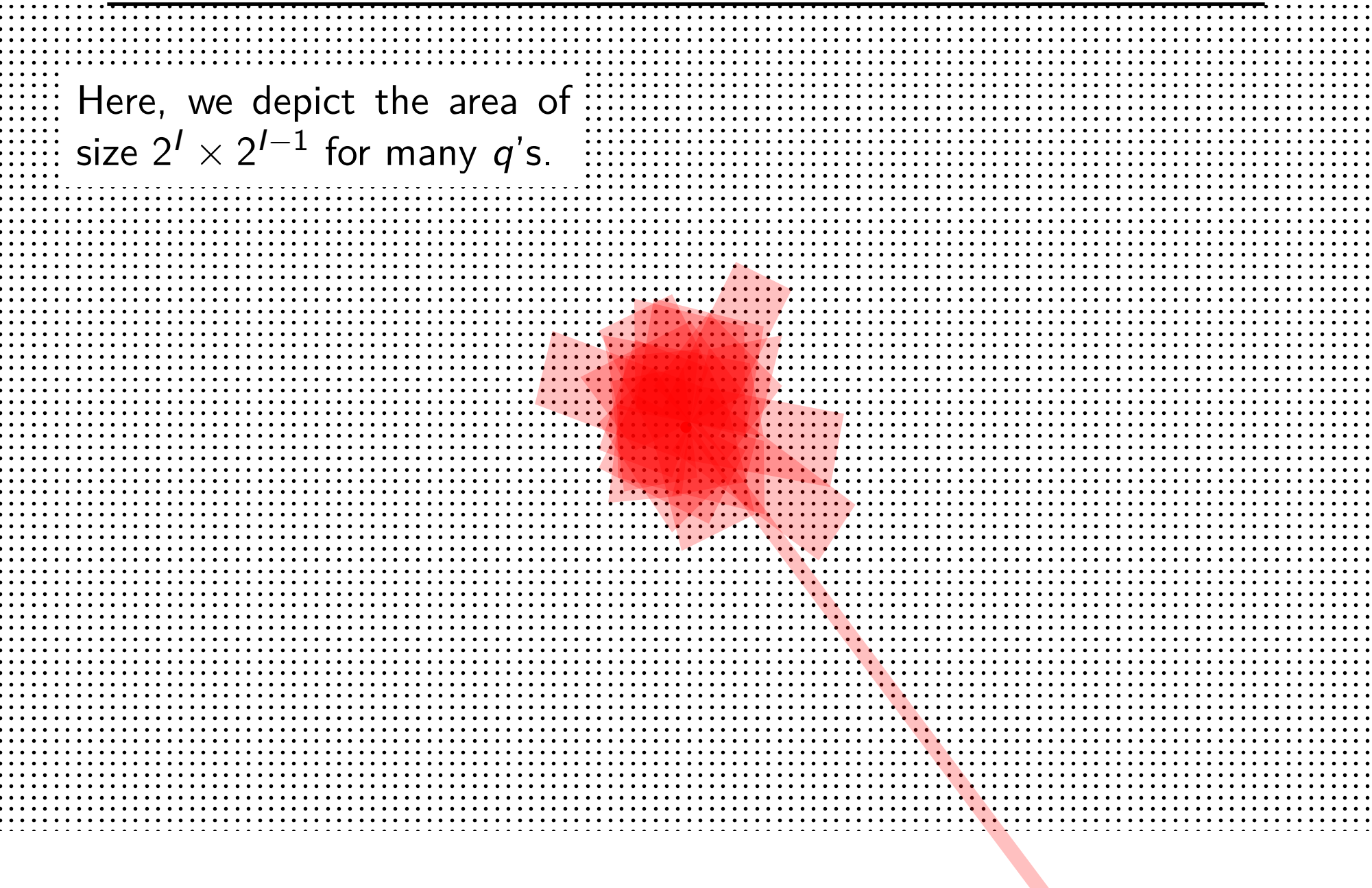
Sieving rectangle

Here, we depict the area of size $2^l \times 2^{l-1}$ for many q's.



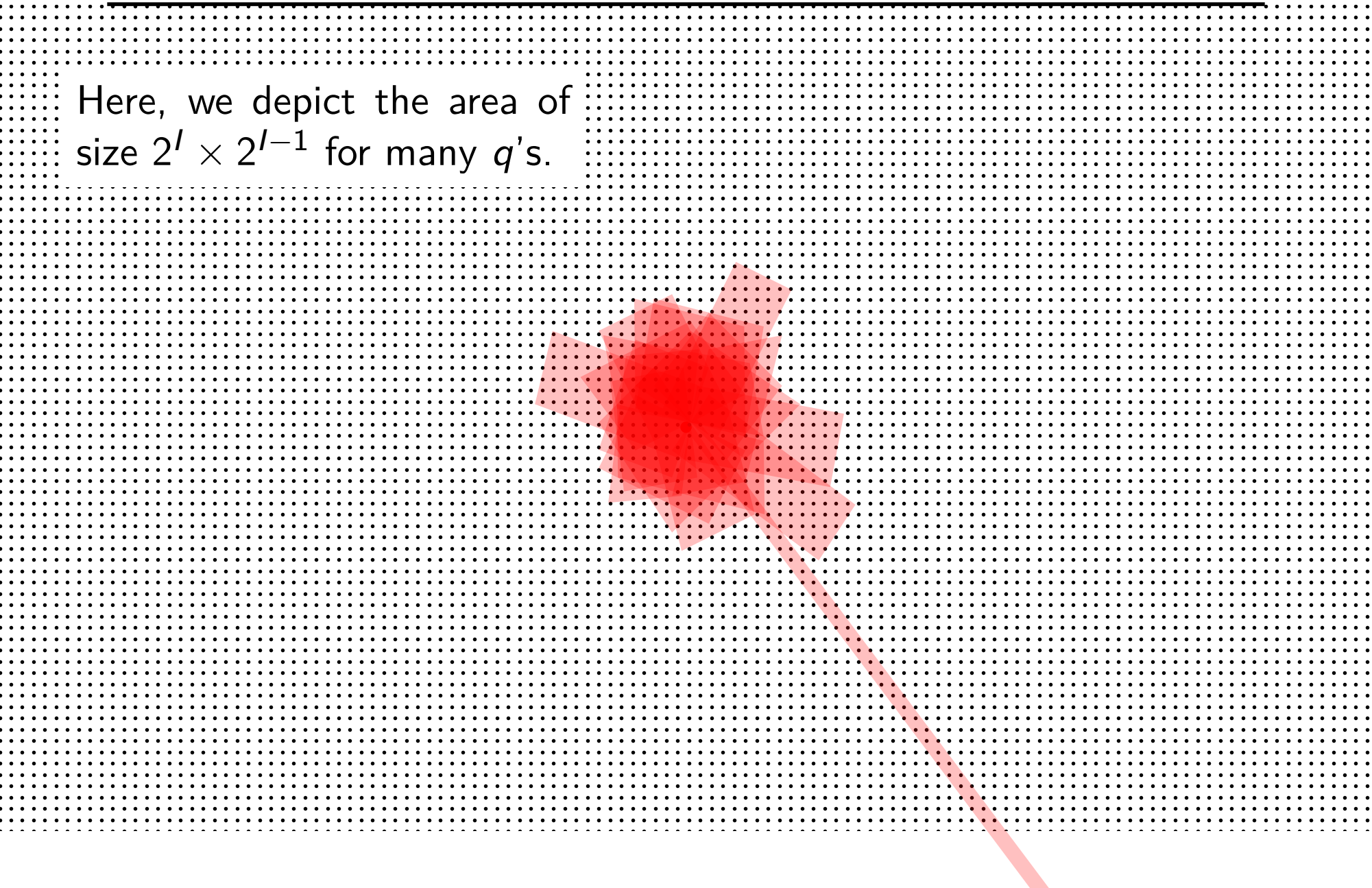
Sieving rectangle

Here, we depict the area of size $2^l \times 2^{l-1}$ for many q's.



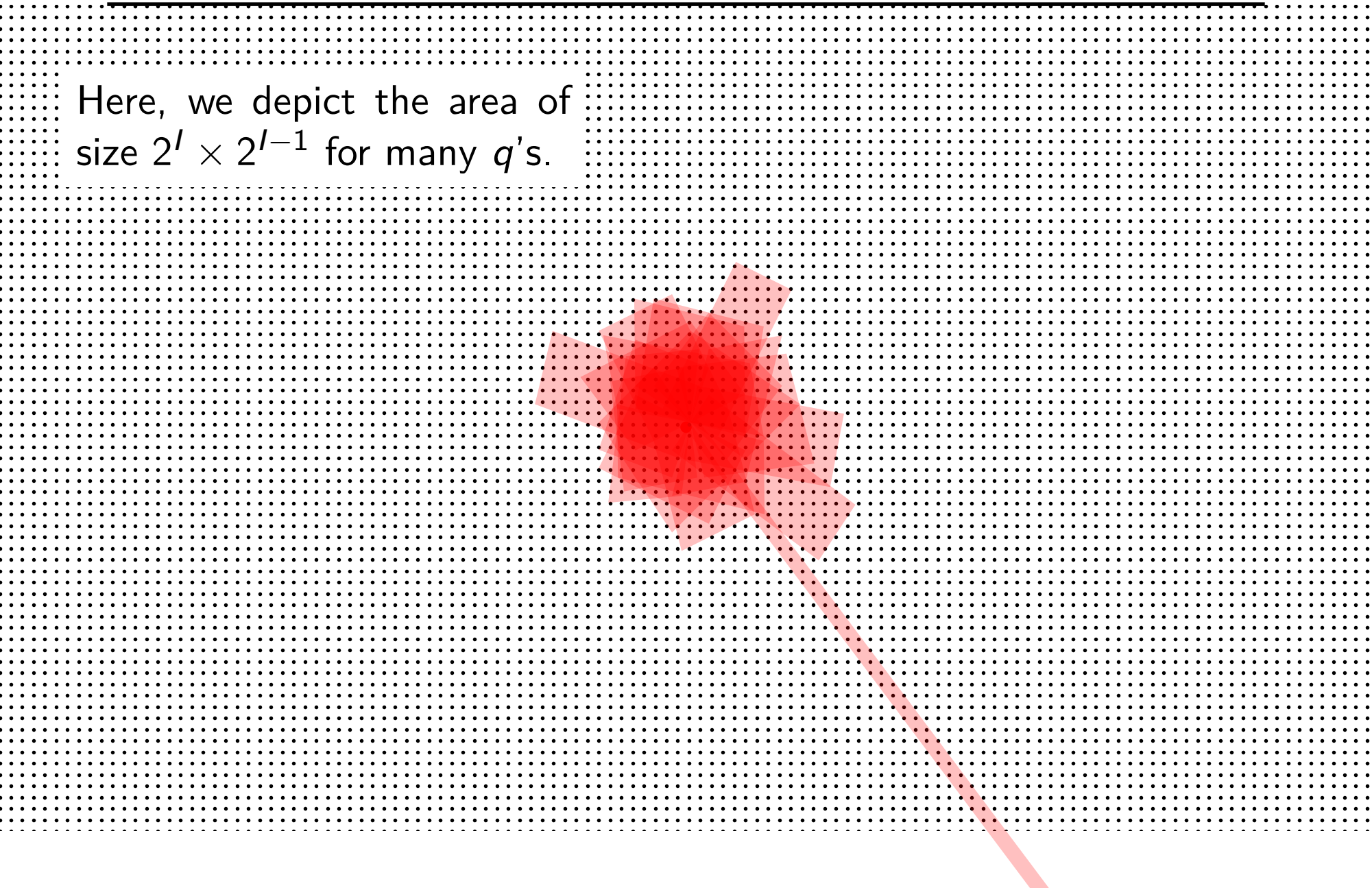
Sieving rectangle

Here, we depict the area of size $2^l \times 2^{l-1}$ for many q's.



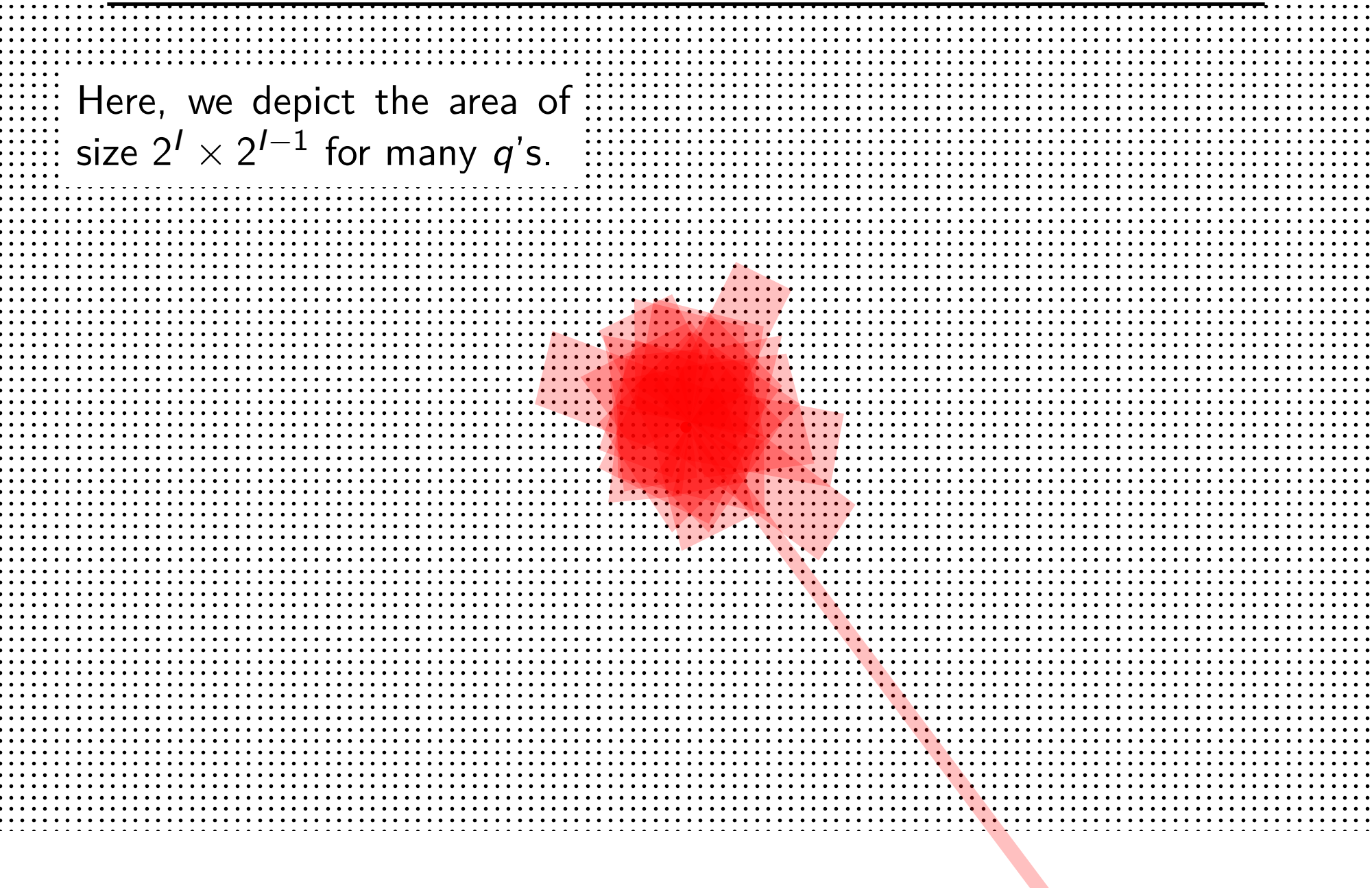
Sieving rectangle

Here, we depict the area of size $2^l \times 2^{l-1}$ for many q's.



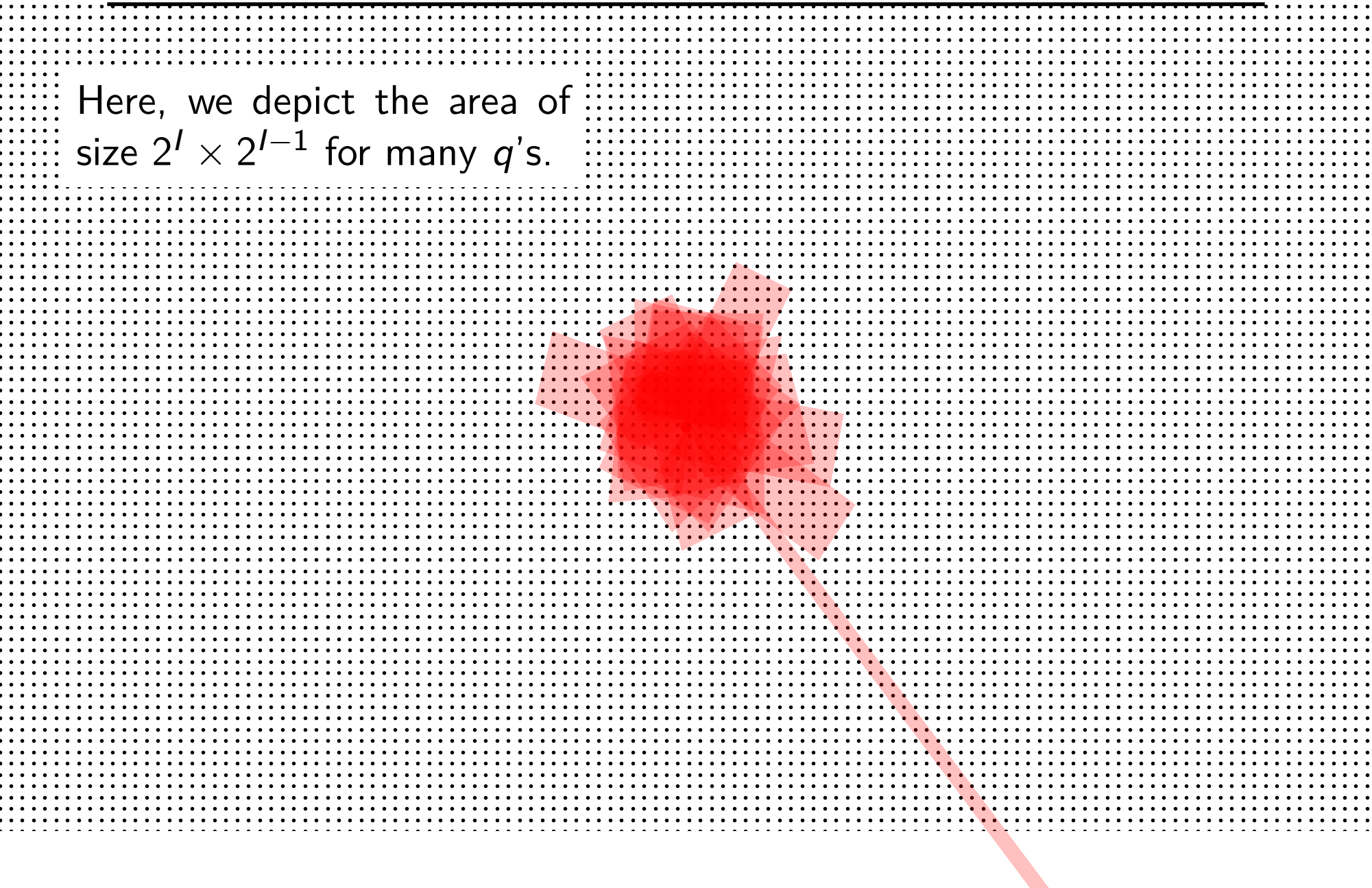
Sieving rectangle

Here, we depict the area of size $2^l \times 2^{l-1}$ for many q's.



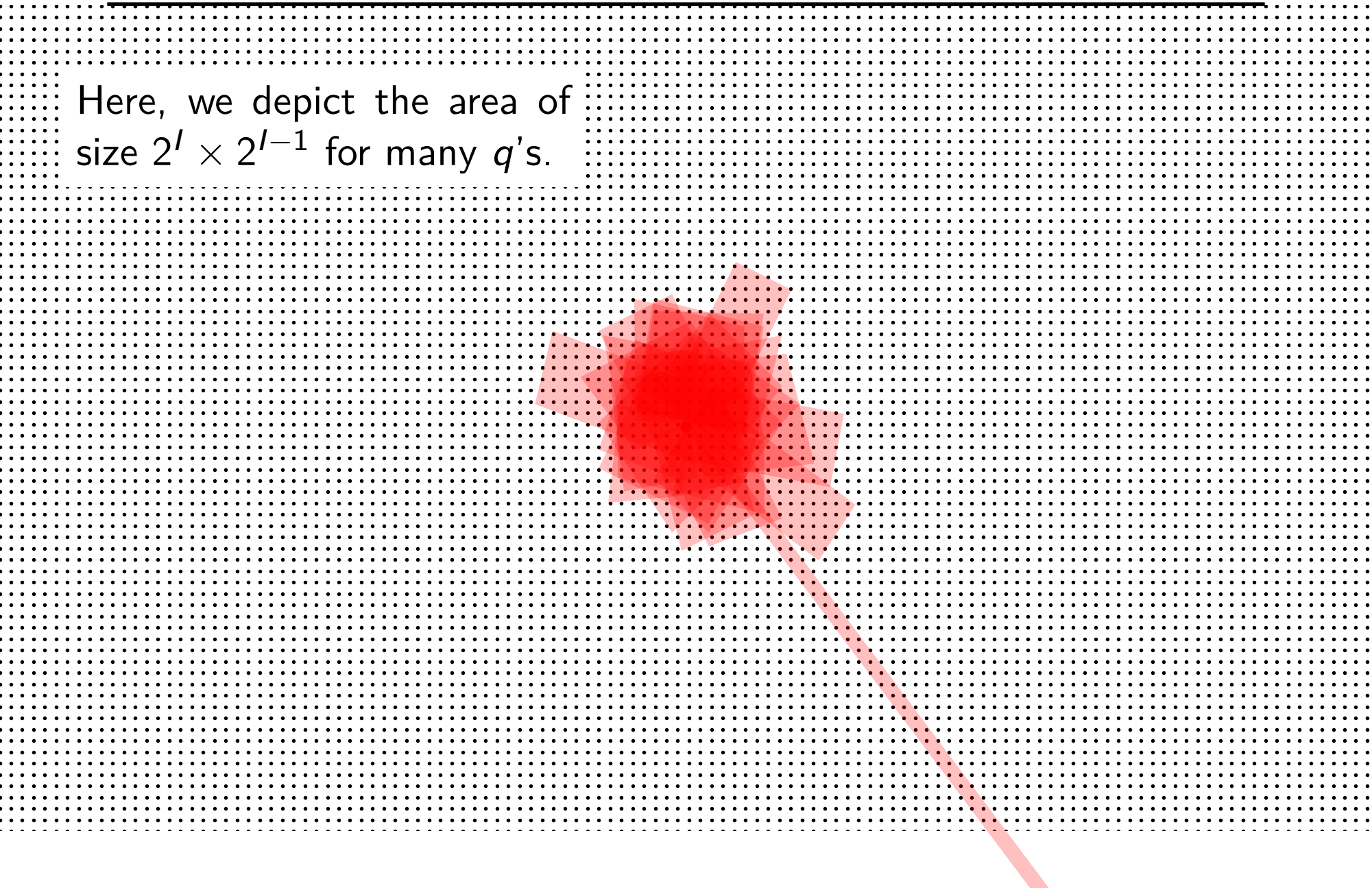
Sieving rectangle

Here, we depict the area of size $2^l \times 2^{l-1}$ for many q's.



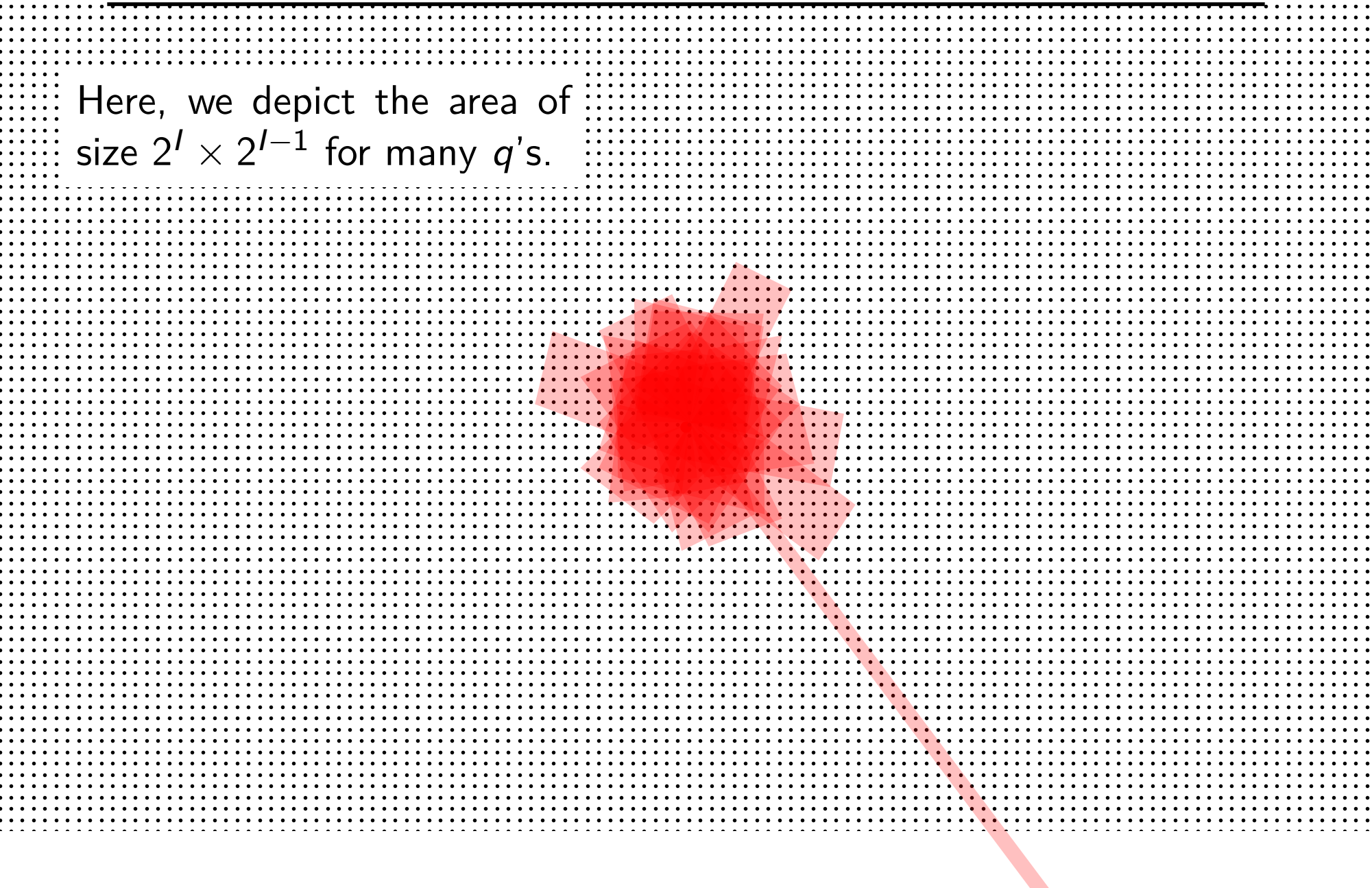
Sieving rectangle

Here, we depict the area of size $2^l \times 2^{l-1}$ for many q's.



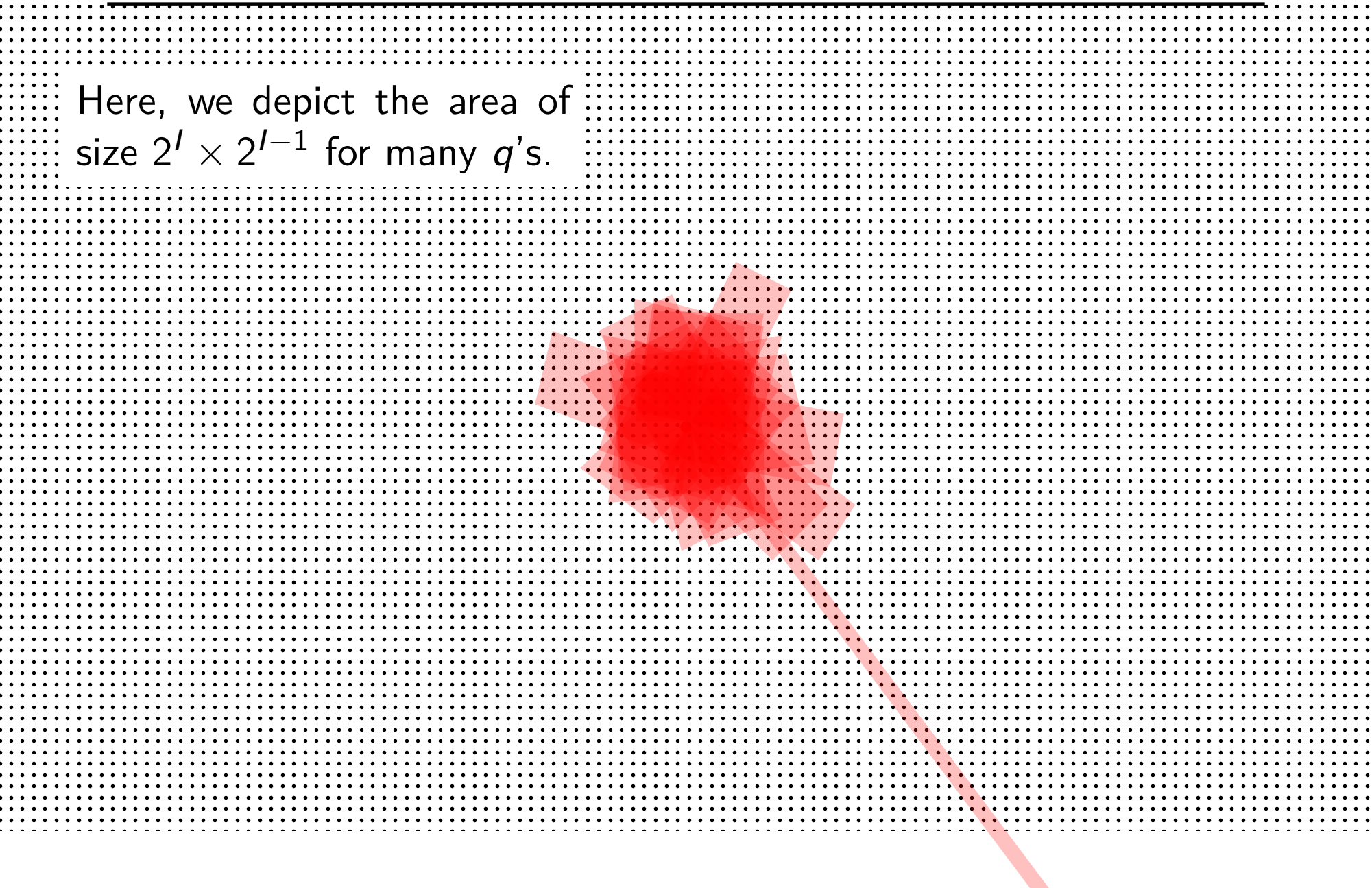
Sieving rectangle

Here, we depict the area of size $2^l \times 2^{l-1}$ for many q's.



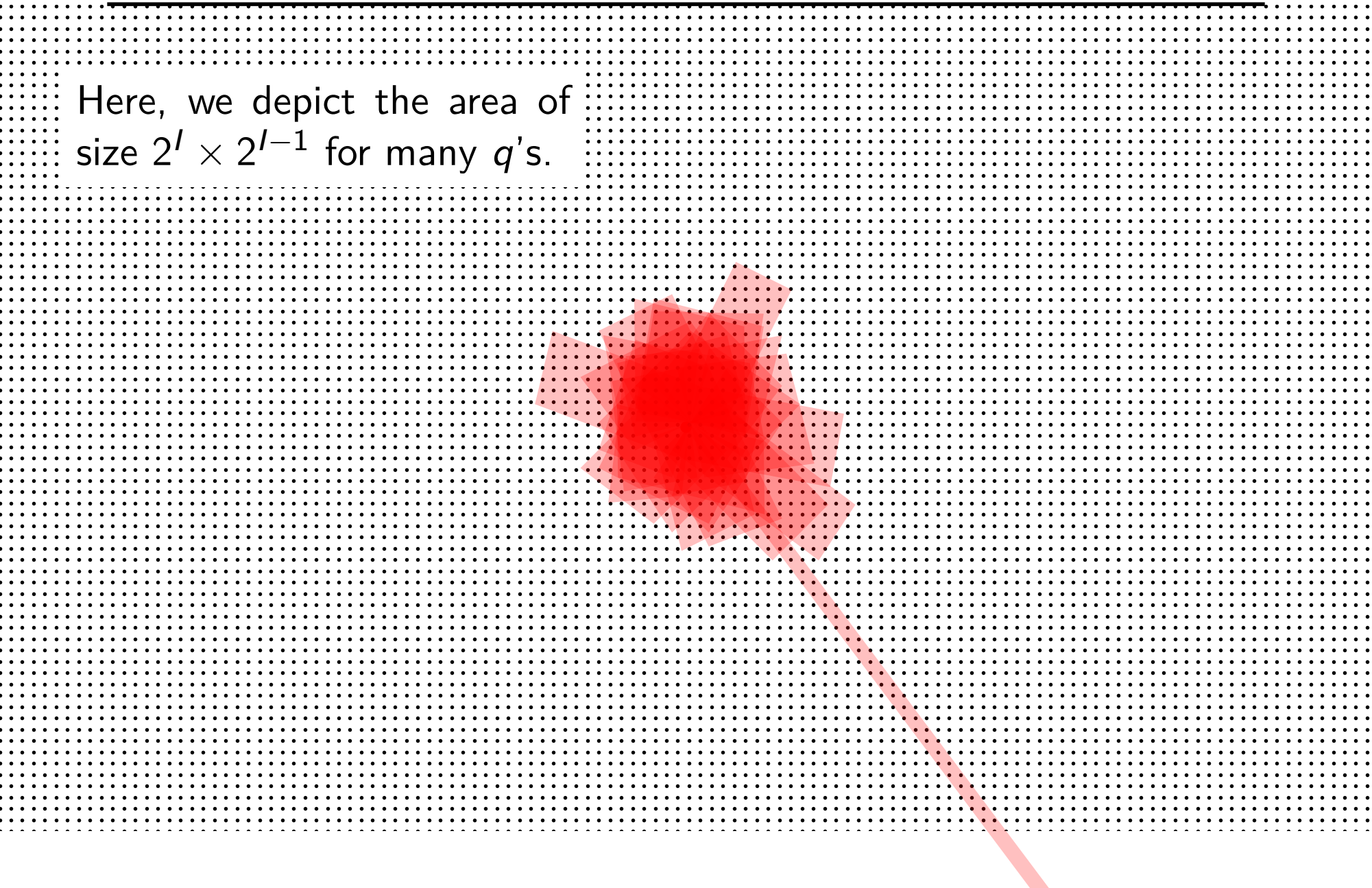
Sieving rectangle

Here, we depict the area of size $2^l \times 2^{l-1}$ for many q's.



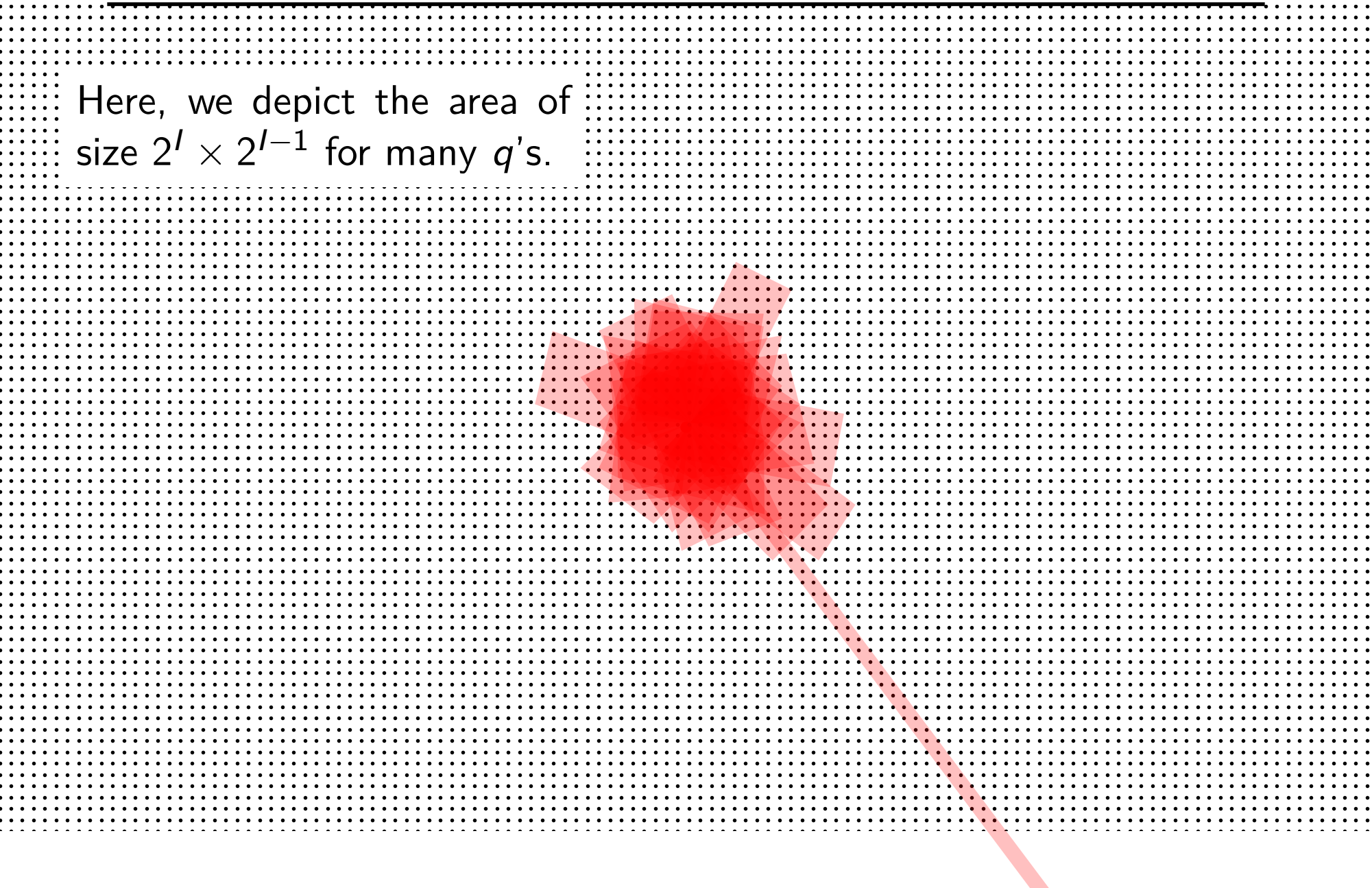
Sieving rectangle

Here, we depict the area of size $2^l \times 2^{l-1}$ for many q's.



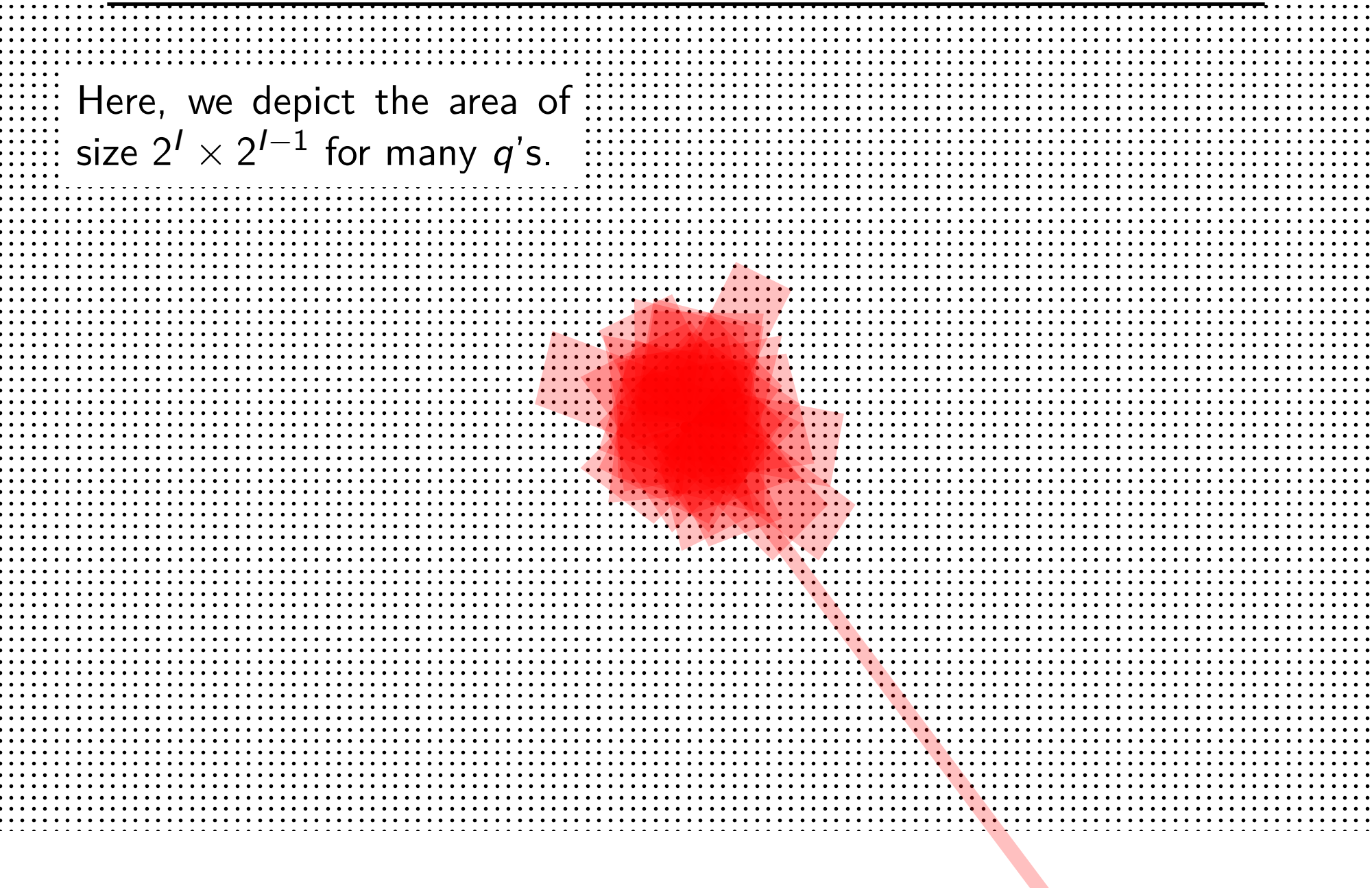
Sieving rectangle

Here, we depict the area of size $2^l \times 2^{l-1}$ for many q's.



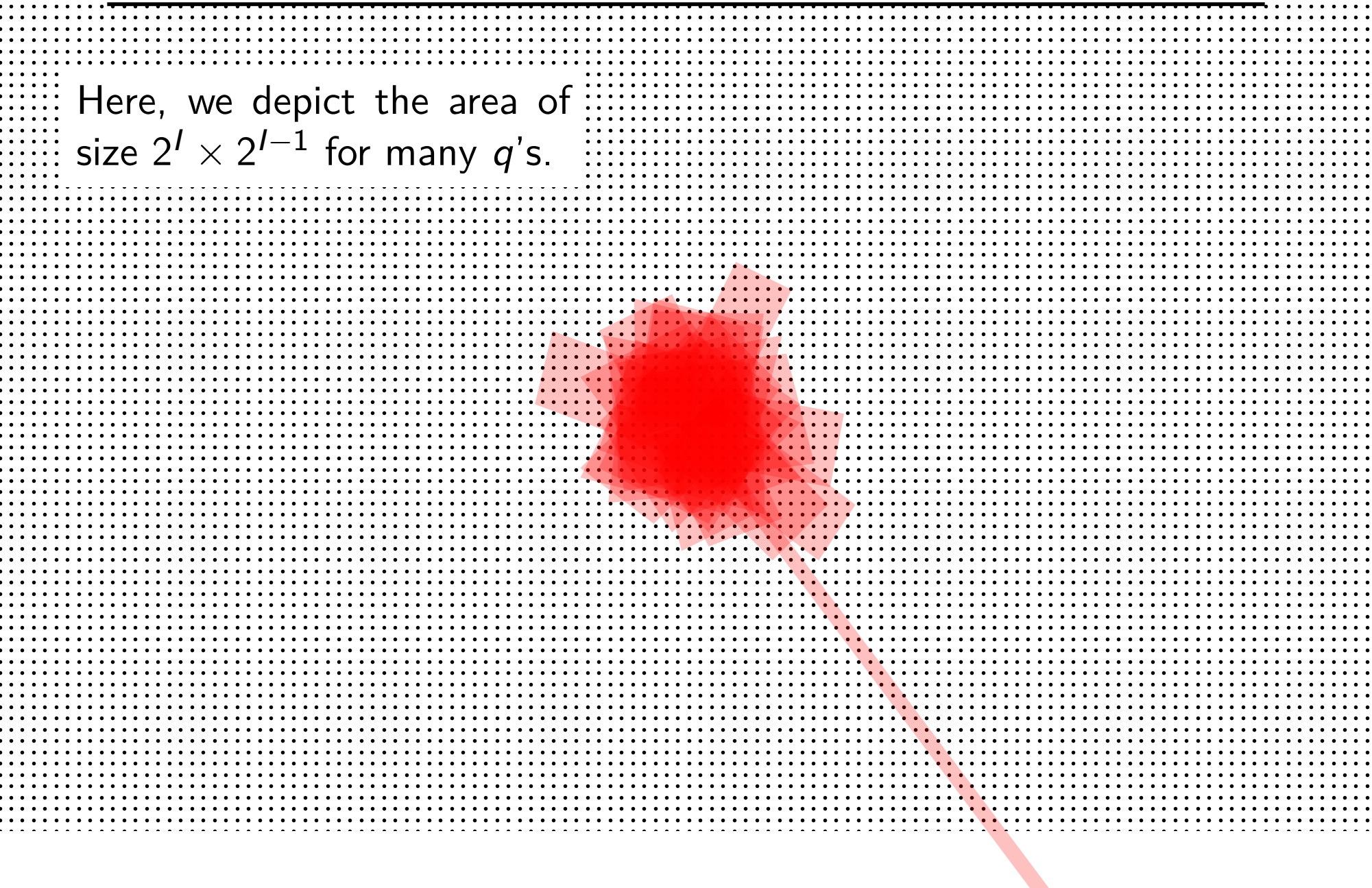
Sieving rectangle

Here, we depict the area of size $2^l \times 2^{l-1}$ for many q's.



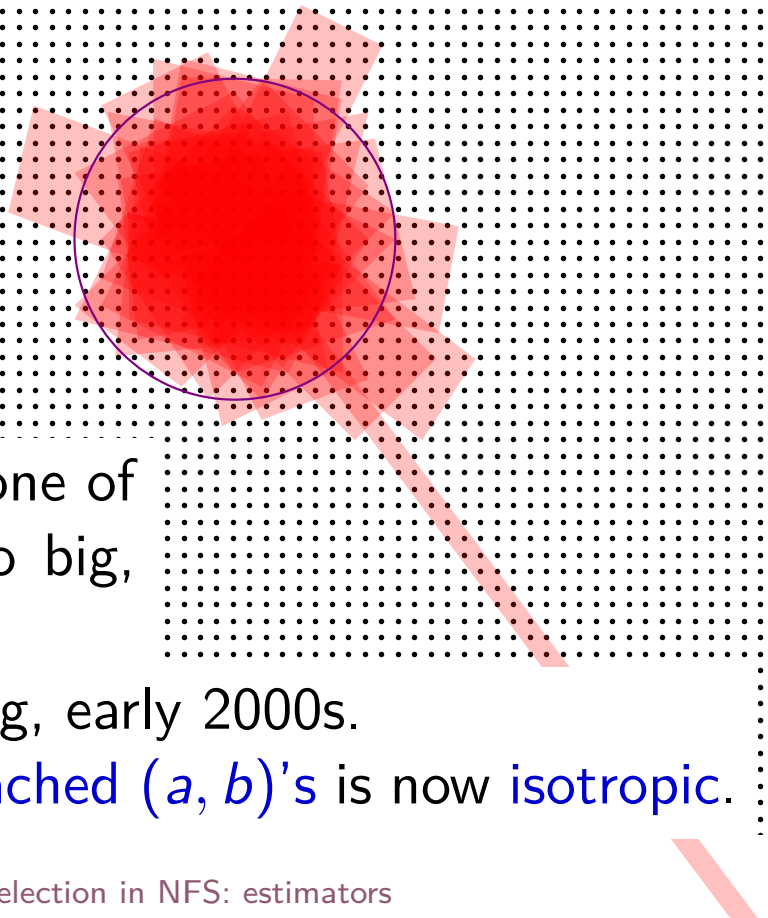
Sieving rectangle

Here, we depict the area of size $2^l \times 2^{l-1}$ for many q's.



Sieving rectangle

Here, we depict the area of size $2^l \times 2^{l-1}$ for many q's.



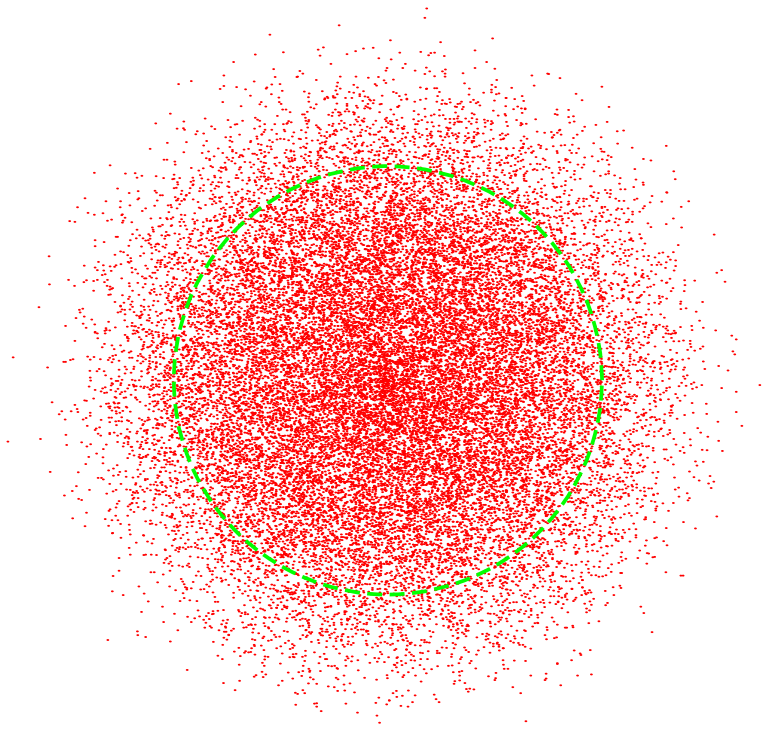
Except in rare cases where one of the basis vectors is way too big, things remain under control.

Idea by Franke and Kleinjung, early 2000s.

Notice that the region of reached (a, b)'s is now isotropic.

L^2 circular norm

- In practice, we escape the rectangle.
- With lattice sieving: circle.



Use a circular integral:

$$\iint_{[0,1]\times[0,2\pi[} F(r\cos\theta, r\sin\theta)^2 \, r\mathrm{d}r \, \mathrm{d}\theta$$

This L^2-norm is also easy to compute.

Computing the circular L^2 norm

```
var('X Y r t')
R.<x> = PolynomialRing(ZZ)
IPR.<a> = InfinitePolynomialRing(R)
d = 6;
f = SR(sum(a[i]*x^i for i in range(d+1)))
F = f.subs(x=r*cos(t),y=r*sin(t))
v = integrate(integrate(F^2*r,(r,0,1)),(t,0,2*pi))
v.expand().collect(pi)
```

L^2 circular norm – example

Example for circular L^2-norm for $d = 4$:

$$\|f\|^2 = \frac{\pi}{640}\Big(3a_2^2 + 5(a_3^2 + a_1^2) + 35(a_4^2 + a_0^2) + 6(a_0 a_4 + a_1 a_3) + 10(a_4 a_2 + a_2 a_0)\Big).$$

- Various formulæ for each degree (easy to hard-code).
- The fact that it is an L^2-norm may also be useful.

Plan

The sieved range

Valleys and the starfish picture

Root properties

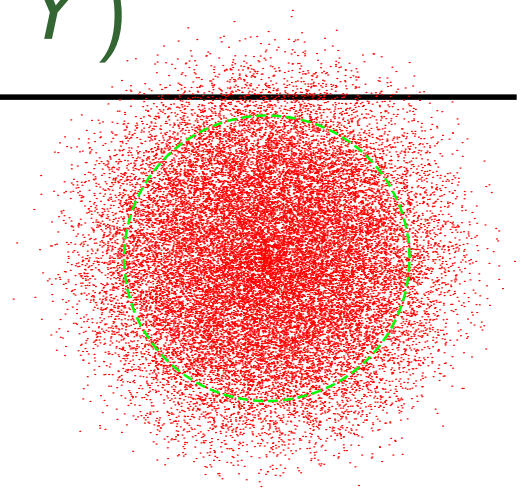
Forcing good $\alpha(f)$: the root optimization

Polynomial selection in Cado-NFS

Notations

Notations: $(f, \cdot), \deg f = d, f = \sum f_i x^i$, $\phi(x) = a - bx$.
(We only care about one polynomial at a time, or conceivably their product if we feel like it)

Small values of $F(X, Y)$



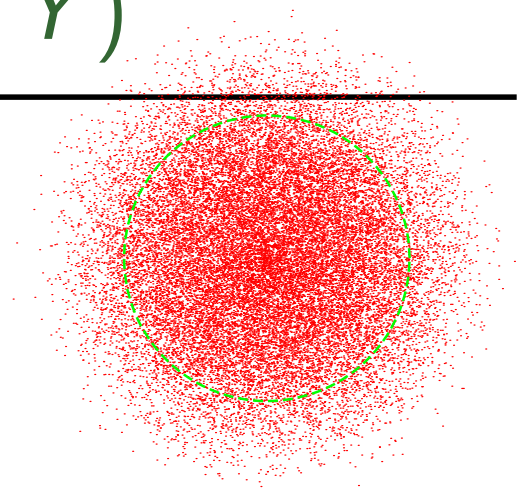
- The set of reached (a, b) values is isotropic (skewness aside, as usual)

- For (X, Y) on the unit circle: $F(X, Y) = Y^d f(X/Y) = (\sin\theta)^d \cdot f(1/\tan\theta)$.

On the unit circle, F takes various values, small and large.

The smallest ones are when...

Small values of $F(X, Y)$



- The set of reached (a, b) values is isotropic (skewness aside, as usual)

- For (X, Y) on the unit circle: $F(X, Y) = Y^d f(X/Y) = (\sin\theta)^d \cdot f(1/\tan\theta)$.

On the unit circle, F takes various values, small and large.

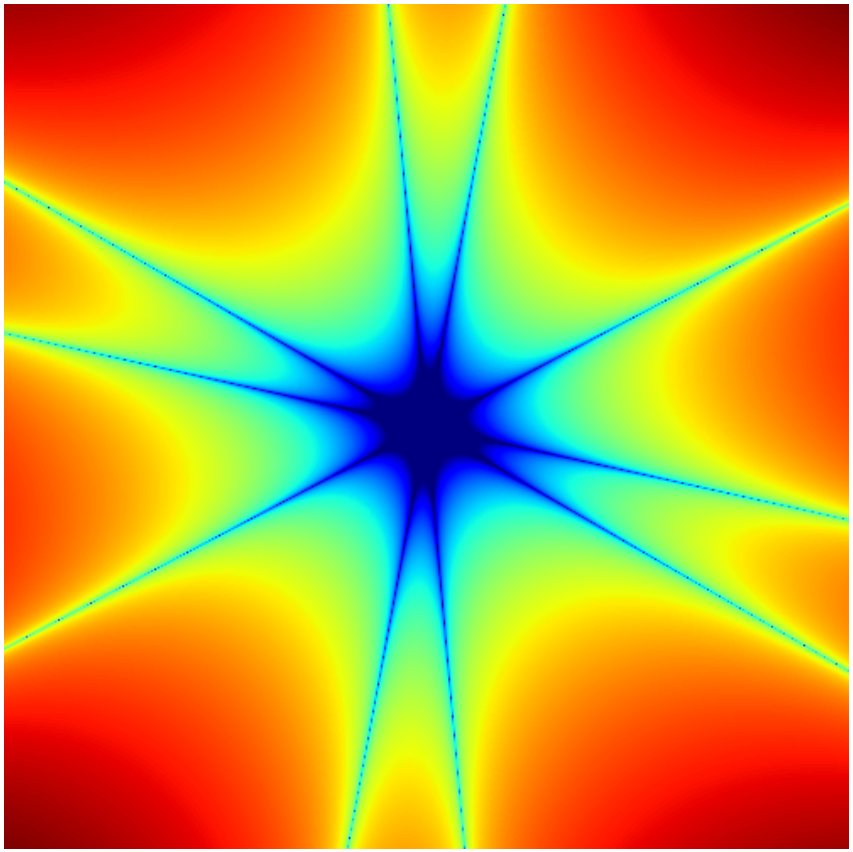
The smallest ones are when...
$X/Y = 1/\tan\theta$ is one of the real roots of f.

$\log|F(a, b)|$

The size of $\log|F(a, b)|$ depends on the number of real roots of f. More real roots = more places where we expect $F(a, b)$ to be very small and hence more likely to be smooth.

The starfish-like picture

Let $f = 4x^5 + 17x^4 - 18x^3 - 58x^2 + 6x + 1$. Plot $\log |F(a, b)|$.

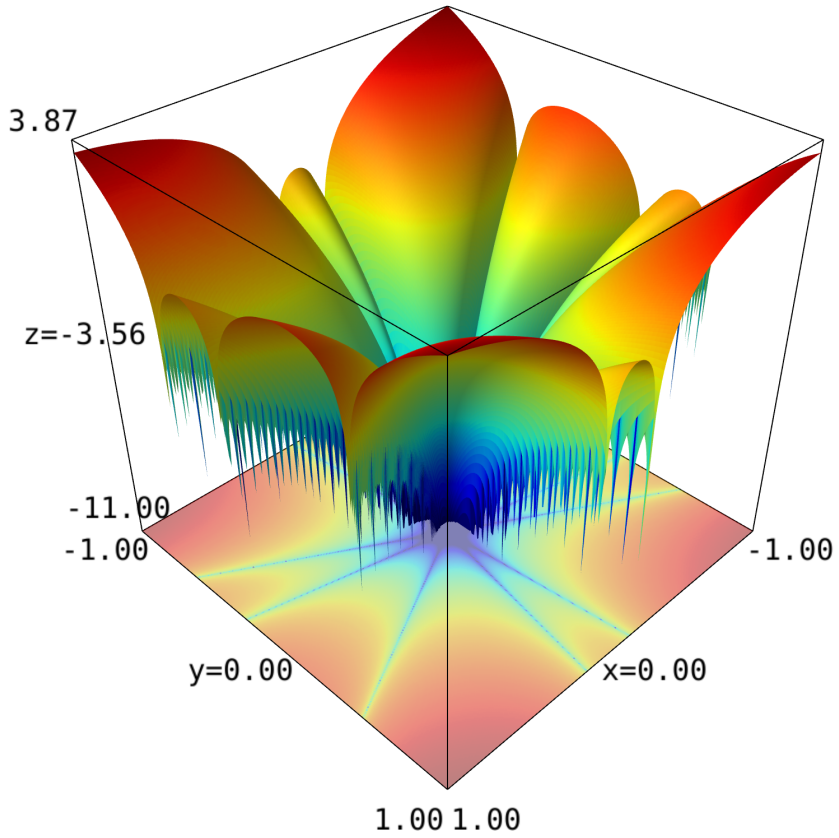


(sage code)

More real roots = hints about areas which are more worth looking into than others.

The starfish-like picture

Let $f = 4x^5 + 17x^4 - 18x^3 - 58x^2 + 6x + 1$. Plot $\log |F(a, b)|$.



(sage code)

More real roots = hints about areas which are more worth looking into than others.

Bernstein's integral calculation

What if we want to test for smoothness
only the (a, b) such that $|F(a, b)| < H$.

E.g. only the deep blue area in the previous picture.

The number of such pairs is actually easy to approximate (Bernstein, 2004).

$$S(H) = \#\{(a, b) \in \mathbb{R} \times \mathbb{R},\ |\mathrm{Res}(a - bx, f)| \leq H\},$$
$$\approx H^{2/\deg f} \int_{-\infty}^{\infty} \frac{\mathrm{d}t}{(f(t)^2)^{1/\deg f}}.$$

There are possible shortcuts to estimate this integral quickly.

Impact on how we collect relations

Does this integral calculation say that we should look only at the value below H?

- The branches along the real roots are very thin. Their contribution is expected to be very small.
- Sieving in these ranges is expected to be difficult.
- More generally, sieving in this oddly-shaped region (even if we do not explore branches) is not particularly appealing.
- Note though that sieving is not the only way to collect relations.

Current software does not explore the (a, b) pairs specifically in this region (but: more on this later).

However, this integral could be used as an estimator to compare polynomials. Cado-NFS does not do this.

Recap

At this point we know how to:

- generate polynomial pairs with small coefficient bounds.
- use translation and (thanks to skewness) rotation to reduce the coefficients even further.
- use various estimators to assess the average (log-)norm over a given region.
- draw fancy pictures.

Plan

The sieved range

Valleys and the starfish picture

Root properties

Forcing good $\alpha(f)$: the root optimization

Polynomial selection in Cado-NFS

Notations

Notations: $(f, \cdot), \deg f = d, f = \sum f_i x^i$, $\phi(x) = a - bx$.

An obvious lie

So far, we've relied on the heuristic:

> If $|F(a,b)| \approx H$, then $F(a,b)$ is smooth
> about as often as a random integer $\approx H$.

This is obviously wrong.

Examples:

- $3a^2 + b^2$ is never divisible by 5, 11, or 17 (for coprime a, b).
- OTOH, it is more often divisible by 7, 13, or 19.

This is related to ...

An obvious lie

So far, we've relied on the heuristic:

> If $|F(a,b)| \approx H$, then $F(a,b)$ is smooth
> about as often as a random integer $\approx H$.

This is obviously wrong.

Examples:

- $3a^2 + b^2$ is never divisible by 5, 11, or 17 (for coprime a, b).
- OTOH, it is more often divisible by 7, 13, or 19.

This is related to ... the number of roots modulo small primes.

Roots modulo small primes

- $3a^2 + b^2$ is never divisible by 5, 11, or 17 (for coprime a, b) because $3x^2 + 1$ has no roots modulo these primes.
- OTOH, it is more often divisible by 7, 13, or 19, because $3x^2 + 1$ has two roots modulo these primes.

This is a general phenomenon.

Restatement: in the number field $\mathbb{Q}[x]/(3x^2 + 1)$ we have:

- no prime ideal of norm 5, 11, or 17.
- two prime ideals of norm 7, 13, or 19.

Roots modulo small primes

- $3a^2 + b^2$ is never divisible by 5, 11, or 17 (for coprime a, b) because $3x^2 + 1$ has no roots modulo these primes.
- OTOH, it is more often divisible by 7, 13, or 19, because $3x^2 + 1$ has two roots modulo these primes.

This is a general phenomenon.

Restatement: in the number field $\mathbb{Q}[x]/(3x^2 + 1)$ we have:

- no prime ideal of norm 5, 11, or 17.
- two prime ideals of norm 7, 13, or 19.

Does this make a difference?

Does this make a difference

Answer #1: no, it does not make a difference

- On average, the number of prime ideals of norm below B is $\approx \pi(B)$.
- This follows from the so-called prime ideal theorem (Landau) or from the (much stronger) Chebotarev theorem.

Answer #2: in fact it does

- A deviation from the average is entirely possible for some range of primes.
- If this is the case for the small primes, the average contribution of small primes will be quite large.

Root properties: example

Example from RSA-250:

$$
\begin{aligned}
f = \; & 86130508464000x^6 \\
& - 66689953322631501408x^5 \\
& - 5273322103496633396619 8x^4 \\
& + 4626212456402143713674452346587 9x^3 \\
& - 3113627253613202265126907420550648326x^2 \\
& - 1721614429538740120011760034829385792019395x \\
& - 815835130764290488377337814383769841229611120 00
\end{aligned}
$$

Number of roots of f modulo small primes:

p	r_p	p	r_p	p	r_p	p	r_p	p	r_p
2	2	13	2	31	4	53	3	73	1
3	3	17	1	37	4	59	2	79	1
5	2	19	1	41	4	61	1	83	3
7	3	23	2	43	1	67	0	89	2
11	3	29	1	47	6	71	0	97	3

Contribution of small primes

A random integer has p-valuation> 0 with probability $1/p$, etc.

$$\begin{aligned}\mathrm{E}\left[\nu_p(x),\ x \text{ random}\right] &= \frac{1}{p} + \frac{1}{p^2} + \cdots \\ &= \frac{1}{p-1}.\end{aligned}$$

We need to compute the same for $F(a, b)$ with (a, b) coprime.

Definition of $\alpha(f)$

- define a score function that reflects the deviation:

$$\alpha(f) = \sum_{p \text{ prime}} \left(\frac{1}{p-1} - \mathrm{E}[\nu_p(F(a,b))] \right) \log p.$$

- Note that $\frac{\log p}{p} \to 0$. The contribution of small primes to $\alpha(f)$ is dominant.
- $F(a,b) \approx X$ is smooth with probability comparable to that of a random number $\approx Xe^{\alpha}$.
- We want $\alpha(f)$ well < 0.
  Having $\alpha(f) < -k \log 2$ is equivalent to a k-bit smaller norm !

A good f has small α (typically ≤ -7).

Estimation of $\mathrm{E}[\nu_p(F(a,b))]$.

$F(a,b)$ (for coprime a,b) is divisible by p when...

- number-theoretician answer:
  when an ideal of power-of-p norm divides $(a - b\alpha) \times J$.
- hurried practitioner answer, only valid if if $p \nmid f_d \operatorname{disc} f$:
  when a/b is one of the roots of f modulo p.

Splitting into multiple cases

As far as only what happens $\bmod p$ is concerned, how many non-equivalent values of (a, b) for coprime (a, b) do we have?

- $p^2 - 1$ if we count everyone but $(0, 0)$.
- But $F(\lambda a, \lambda b) = \lambda^{\deg f} F(a, b)$ when λ is invertible.
  So we can divide by $p - 1$.
- There are $p + 1$ different classes.
  In mathematical terms: this is the projective line $\mathbb{P}^1(\mathbb{Z}/p\mathbb{Z})$.

Estimation of $\mathrm{E}[\nu_p(F(a,b))]$.

For the primes for which $p \nmid f_d \operatorname{disc} f$, the contribution to $\alpha(f)$ is:

$$r_p \cdot \frac{1}{p+1} \cdot \left(1 + \frac{1}{p} + \frac{1}{p^2} + \cdots\right)$$
$$= r_p \cdot \frac{1}{p+1} \cdot \frac{p}{p-1}.$$

Problem: small primes are most likely to divide $p \nmid f_d \operatorname{disc} f$.
It is not reasonable to ignore this.

It takes some extra complications to compute the contributions of all primes, but this is very much doable.

$\alpha(f)$ joins the estimation toolbox

$\alpha(f)$ is easy to compute

It is easy to compute (a reasonable approximation of) $\alpha(f)$ by restricting to (say) primes < 2000.
It makes sense to range polynomials according to the value of

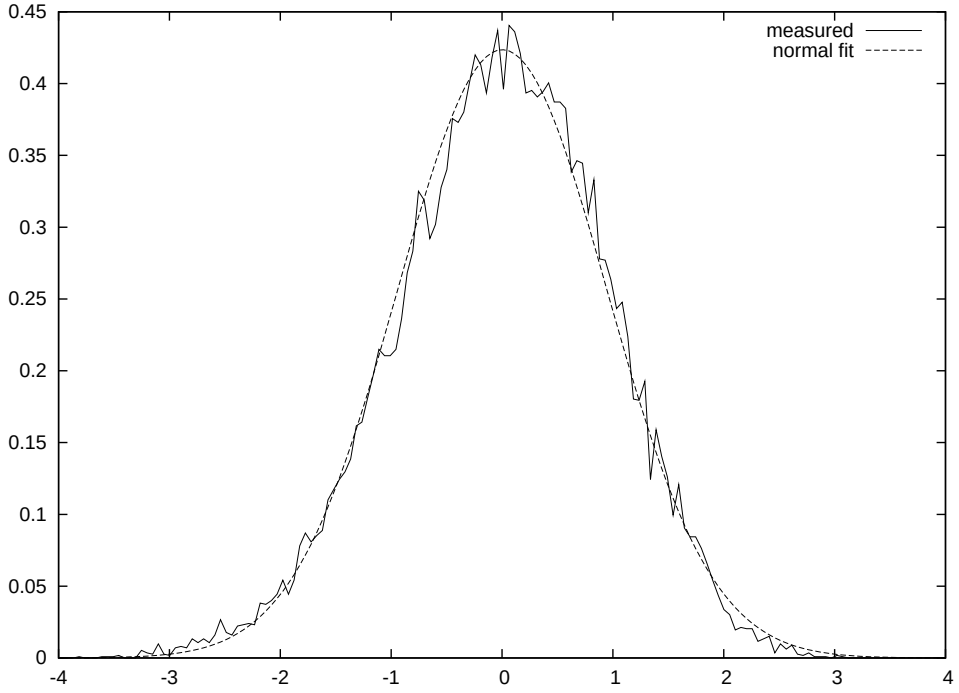
$$\log(\text{average } |F(a,b)|) + \alpha(f).$$

(`sage` code in Cado-NFS), (`C` code in Cado-NFS)

Distribution of $\alpha(f)$

As f varies over a large number of polynomials, $\alpha(f)$ follows a roughly normal distribution.

- It is not hard to estimate μ and σ empirically.
- It may even be doable in theory, but I'm not sure.



Examples of $\alpha(f)$

Examples:

- RSA-768, $\alpha(f) = -7.3$.
- RSA-240, $\alpha(f) = -8.48$.
- RSA-250, $\alpha(f) = -8.88$.

$p \mid f_d$ is an important case!

What happens for $(a : b) \sim (1 : 0) \mod p$ is as important as the other cases.
This is the reason why we often force f_d to have many small prime factors: it forces extra roots.

Predicting $\alpha(f)$

If we examine (say) $N = 2^{30}$ polynomials f, and we intend to keep the best (smallest) $\alpha(f)$, what should we hope for?

Experimentally, we can determine μ and σ for the normal fit of the distribution of $\alpha(f)$. WLOG, assume $\mu = 0$ and $\sigma = 1$.

Our problem is the determination of the max value of N random picks of a normal-distributed random variable X.

This is a well-known problem called order statistics.

- First-order $\sqrt{2 \log N}$.
- More terms are known, as well as empirical "corrections".
- This can be use to ponder whether "better" polynomials are still to be expected or not.

Plan

The sieved range

Valleys and the starfish picture

Root properties

Forcing good $\alpha(f)$: the root optimization

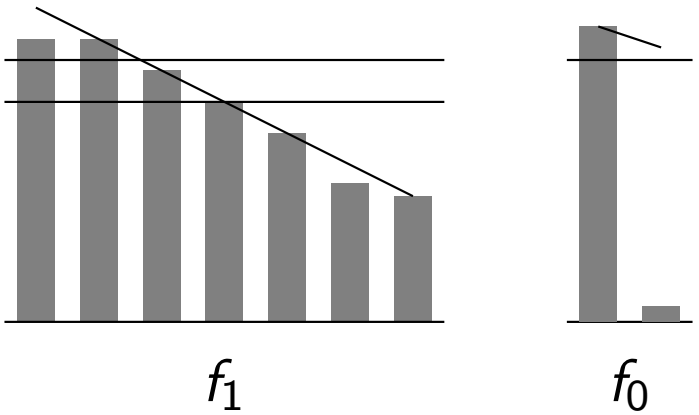
Polynomial selection in Cado-NFS

Notations

Notations: $(f_0, f_1), \deg f_1 = d$, coefficients of f_i unnamed.

Upsides of skewness

When we obtain a skewed polynomial pair, it is rare that the lower-degree coefficients play a role.



In most cases, the rotation $(f_0, f_1) \to (f_0, f_1 + \lambda f_0)$ will not change the coefficient sizes by much (if at all).

- What happens $\bmod p$ is changed completely. We may hope for improvements.
- We used rotation for size optimization already. Here we want to do it on a more limited scale.

Root properties of $f_1 + \lambda f_0$

The stupid way to do root optimization

- Iterate over all possible λ.
- For each, compute $\alpha(f_1 + \lambda f_0)$.
- Keep best.

Main problem: root finding mod many p for each λ.

Is it possible to compute $\alpha(f_1 + \lambda f_0)$ quickly? Not really.

Root optimization – the root sieve

The root sieve

- Iterate over all $p^k < B$ below some fixed bound.
- Iterate over all possible root values $i \bmod p^k$.
- For all λ such that $f_1(i) + \lambda f_0(i) \equiv 0 \mod p^k$, add to $T[\lambda]$ the contribution of having a root (typically $\frac{p \log p}{p^2-1}$).
- Keep the ones with largest recorded contribution.
- Beware: this costs $\widetilde{O}(B \cdot (B + \lambda_{\max}))$

It is not entirely straightforward.

- Some issues with finding the contribution of multiple roots correctly.
- Can gain a factor of two by looking at powers more precisely.

Plan

The sieved range

Valleys and the starfish picture

Root properties

Forcing good $\alpha(f)$: the root optimization

Polynomial selection in Cado-NFS

Polynomial selection in Cado-NFS

The degree is fixed beforehand. First phase:

- loop through all a_d in some range `[admin,admax]`.
  - It makes sense to look at only the multiples of some prescribed number, hence `incr`.
  - This can (MUST) be distributed over many machines! The workload is divided into many pieces according to `adrange`.
- run algorithms to find good polynomials based on a_d.
  The Cado-NFS implementation of Kleinjung's 2008 algorithm has a few arcane parameters (`P`, `nq`).
- size-optimize.
  - quantify the amount of effort we put into it: `sopteffort`.
- collect all these size-optimized polynomials in a central place.
  Limited-size priority queue of best polynomials: `keep`.

Polynomial selection in Cado-NFS (cont'd)

Second phase:

- For each of the best pairs of polynomials that are still in the priority queue, run root optimization.
- That can very well mean a sizable number of root optimization tasks, so distribution is again a good idea.
- Not as many parameters.
- Keep the very few best ones (≈ 10).
- For large-scale computations, run sieving tests with each.

CSE291-14: The Number Field Sieve

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

Emmanuel Thomé

February 3, 2022

Part 5a

Collecting relations in NFS

Introduction

Sieving and special-q

The sieving primes

Plan

Introduction

Sieving and special-q

The sieving primes

Plan

Introduction

- The name of the game
- Many ways to reach the same goal
- Terminology

The name of the game

The relation collection process as a whole is like this:

Input.

- 2 polynomials f_0 and f_1 such that $N \mid \mathrm{Res}(f_0, f_1)$;
  e.g. $\deg f_0 = 1$ *and* $\deg f_1 > 1$.
- ($f_{0,1}$ output by `polyselect`, or derived from SNFS setting).

Output.

- A set of many, many relations:

$$a, b : p_1, \ldots, p_k : q_1, \ldots, q_\ell.$$

  with p and q prime numbers below some bounds B_0 and B_1.
- Slight abuse of notations: the integers
  $F_0(a, b) = \mathrm{Res}(f_0, a - bx) = b^{\deg f_0} f_0(a/b) = \prod p_i$ and
  $F_1(a, b) = \mathrm{Res}(f_1, a - bx) = b^{\deg f_1} f_1(a/b) = \prod q_i$
  are often called norms.

What a relation encodes

As we have seen, a relation such as

$$a, b : p_1, \ldots, p_k : q_1, \ldots, q_\ell.$$

encodes a factorization in some algebraic structure, with some info that is only implicit.

Example: non-monic linear polynomial

If we have $f_0(x) = m_1 x - m_0$, the interpretation is:

$$\begin{aligned} \mathrm{Res}(a - bx, f_0(x)) &= m_1 a - m_0 b \\ &= \pm p_1 \times \cdots p_k. \\ a - b(m_0/m_1) &= \pm \frac{1}{m_1} p_1 \times \cdots p_k. \end{aligned}$$

Interpretation (2)

Interpretation on the algebraic side

If $\mathbb{Q}[x]/f_1(x) = \mathbb{Q}(\alpha)$, the interpretation of the right part is as follows.

- Assume for example that only q_1, q_3 are primes modulo which algebraic work is a bit harder.
- All other primes are "easy primes".

$$F_1(a,b) = q_1 \times \cdots q_k.$$

$$\langle a - b\alpha \rangle = J^{-1} \times \qquad \text{(reminder: } J = \langle 1, \alpha \rangle^{-1}\text{)}$$

$\times$ some ideals above q_1 and q_3 (work needed)

$\times$ some trivially described ideals above other primes.

Coprimality of a, b

Fact. If (a, b) gives a relation but $d = \gcd(a, b)$ is non-trivial, then $(a/d, b/d)$ gives almost the same relation.

Later on in the algorithm, those two relations cannot be non-trivially combined.

Consequence. The output of the algorithm must contain only primitive relations.

Rem. The number of coprime pairs is a constant fraction of the total number of pairs.

Rem. In practice, computing all the GCDs in advance is too costly: do it only on selected locations at the appropriate time.

Plan

Introduction

- The name of the game
- Many ways to reach the same goal
- Terminology

A promise from asymptotic analysis

When we did the analysis, we had:

- A large space $\mathcal{A}$ to choose from ($\#\mathcal{A} = L_N[1/3, \mathbf{+}]$)
- a probability π that $\phi(x) = a - bx$ give rise to a doubly-smooth relation. ($\pi = L_N[1/3, \mathbf{-}]$).
- a target number of relations, say $\mathbf{B} = L[1/3, \text{something}]$. constraint: $(\#\mathcal{A}) \times \pi \geq \mathbf{B}$.

Then in that case, given the quantities at stake, we claimed that we had a way to find AND output all $\mathbf{B}$ required relations for a cost of $(\#\mathcal{A})^{1+o(1)} = L_N[1/3, \mathbf{+} + o(1)]$.

Our claim: sieving can do that. Sieving = today.

Small excursion: good and bad ways to do otherwise.

Very naive algorithm

1. For a in a certain range:
2. For b in a certain range:
3. For all $p < B_0$
4. check if $p \mid F_0(a, b)$.
5. For all prime ideals $\mathfrak{p}$ of norm $< B_1$
6. check if $\mathfrak{p} \mid J \times \langle a - b\alpha \rangle$.
7. If smooth on both sides, print the relation.

Cost: $(\#\mathcal{A}) \times \mathbf{B} = L_N[1/3, \text{too much}]$.

This naive technique does not work.

Same, with ECM

1. For a in a certain range:
2. For b in a certain range:
3. Check smoothness of $F_0(a, b)$ with ECM;
4. Check smoothness of $F_1(a, b)$ with ECM;
5. If doubly-smooth, print the relation.

ECM takes time $L_B[1/2, \sqrt{2} + o(1)]$ to find prime factors below B.

With $B = L_N[1/3, \text{something}]$, that means

Same, with ECM

1. For a in a certain range:
2. For b in a certain range:
3. Check smoothness of $F_0(a, b)$ with ECM;
4. Check smoothness of $F_1(a, b)$ with ECM;
5. If doubly-smooth, print the relation.

ECM takes time $L_B[1/2, \sqrt{2} + o(1)]$ to find prime factors below B.
With $B = L_N[1/3, \text{something}]$, that means $L_N[1/6, \ldots]$.
$(\#\mathcal{A}) \times L_N[1/6, \ldots]$ is $(\#\mathcal{A})^{1+o(1)}$: that works!

- Good news: no memory needed. Infinitely parallelizable. May be relevant for GPU, FPGA, ASIC, at least to a certain extent.
- Bad news: not fast in practice.

Over the top

Another approach:

1. Divide the set of (a, b) into many subsets.
2. For each subset ($=$ many (a, b) pairs):
3. Compute product tree of all $F_0(a, b)$.
4. Compute product tree of all $F_1(a, b)$.
5. Compute remainder trees on both sides.
6. Recover smooth pairs, print relations.

As crazy as it may seem, it works, too!
Important detail: choose subsets so that products are balanced.

The sieving approach (today)

1. Divide the set of (a, b) into many subsets.
2. For each subset (= many (a, b) pairs):
3. For each $p < B_0$:
4. Mark (a, b) such that $p \mid F_0(a, b)$.
5. For each $p < B_1$:
6. Mark (a, b) such that $p \mid F_1(a, b)$.
7. For each (a, b) with a large recorded contribution:
8. Compute and factor $F_0(a, b)$.
9. Compute and factor $F_1(a, b)$.
10. Print relation.

Sieving per se (steps 3-6) costs $(\#\mathcal{A}) \sum_i \log \log B_i$ in total, and steps 8- are only executed for a small fraction π of the input.

Keep in mind: several methods

There is no single way to do relation collection

In practice, the most efficient way involves a blend of:

- Sieving (in order to detect);
- Trial-division;
- Re-sieving (in order to factor);
- ECM;
- Product trees (which can replace sieving entirely).

And in the case of NFS, we have two sides to deal with, with "norms" of very different size to factor. The chain of algorithms need not be the same on both sides.

Plan

Introduction

- The name of the game
- Many ways to reach the same goal
- Terminology

Factoring slang

- The name NFS highlights the importance of the sieving process.
  Yet, sieving is not the only way!
- QS variants (large primes, special-q) also come into play here, and lead to the presence of certain types of primes in the relations.

Our preferred terminology

- Large prime bound (for historical reasons): the largest prime that appears in the relations. This is independent of the method that we use for collecting relations.
- Algorithms such as sieving, product trees, etc, may be parameterized in many ways, and this conditions the shape of the relation that they output.

Plan

Introduction

Sieving and special-q

The sieving primes

Outer/inner aspects

Two directions of study:

- Outer aspect: how the work is divided into pieces in general;
- Inner aspect: what we do with each piece.

We start with the outer aspect.

Line sieving

Idea. The (a, b) search area is too large. Cut it into sub-areas that are handled sequentially / independently.

First strategy

Cut the sieving space (rectangle) in lines, according to b.

This is called line sieving. Has been widely used in the past.

- Line sieving alone is no longer competitive, because we can do better.
- However, line sieving is still part of special-q sieving.

Special-q

Better strategy: sieving by special-q

We want to focus on pairs (a, b) so that a specific prime number is forced to appear in the factorization of $\mathrm{Res}(F_0(x), a - bx)$ or $\mathrm{Res}(F_1(x), a - bx)$.

(good to know: this is one prime that we will not have discover in the factorization of the norm!)

Line sieving vs special-q sieving

Line sieving

The work area is divided in lines (constant b).

For each line, we have to loop through many primes to identify for which a we have a contribution to record as $T[a] += \log p$.

Special-q sieving

The work area is divided in sublattices of $\mathbb{Z}^2$.

Each piece of work explores combinations $i \times (a_0, b_0) + j \times (a_1, b_1)$, with (i, j) ranging over a fixed rectangle, e.g. $2^l \times 2^{l-1}$.

We have to loop through many primes to identify for which (i, j) we have a contribution to record as $T[(i, j)] += \log p$.

Plan

Sieving and special-q

- Special-q and lattice bases
- Forcing divisibility by an ideal
- Special-q: good or bad

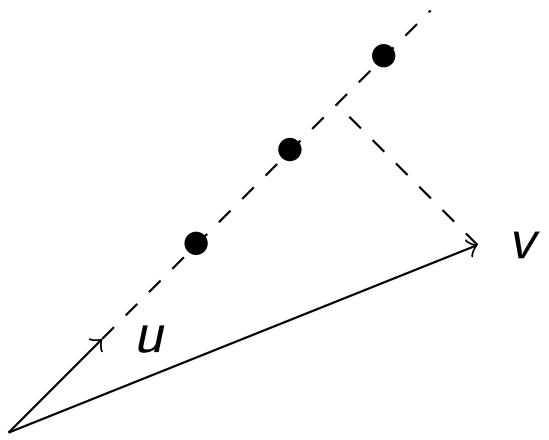
How do we define the q-lattice

Here is a lattice

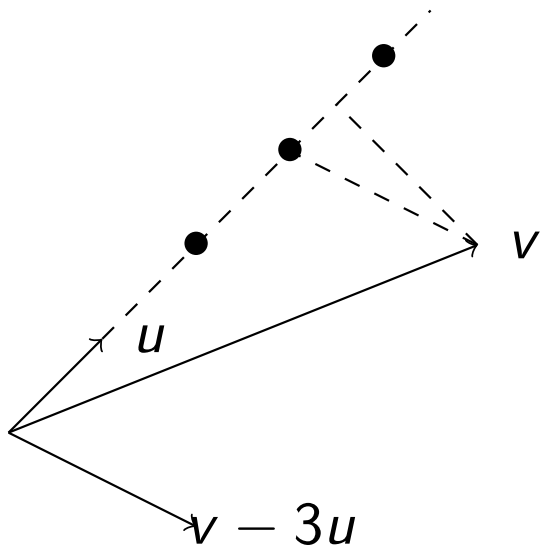
The set of (a, b) such that $q \mid am_1 - bm_0$ is a lattice (say $\mathcal{L}_q$).

Basis (if $\gcd(m_1, q) = 1$): $\begin{cases} (a_0, b_0) = (q, 0) \\ (a_1, b_1) = ((m_0/m_1) \bmod q, 1) \end{cases}$

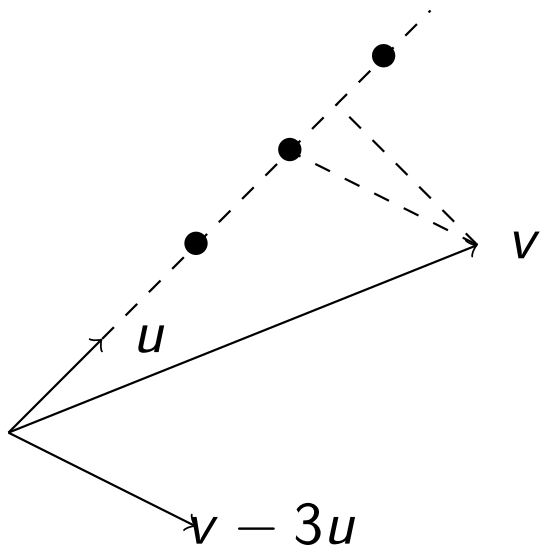
The Gauss reduction algorithm



The Gauss reduction algorithm



The Gauss reduction algorithm




```
while (!done) {
  v ← v − ⌊(v·u)/(u·u)⌉ u
  swap u and v
}
```

Repeating this produces an almost-orthogonal basis for $\mathcal{L}_{\mathfrak{q}}$.

This is is inherently attached to a scalar product. Here:

$$(a_0, b_0) \cdot (a_1, b_1) = a_0 a_1 + b_0 b_1.$$

The reduced basis

With Gauss reduction, we obtain the reduced basis.
We typically expect $a_0 \approx a_1 \approx b_0 \approx b_1 \approx \sqrt{q}$.

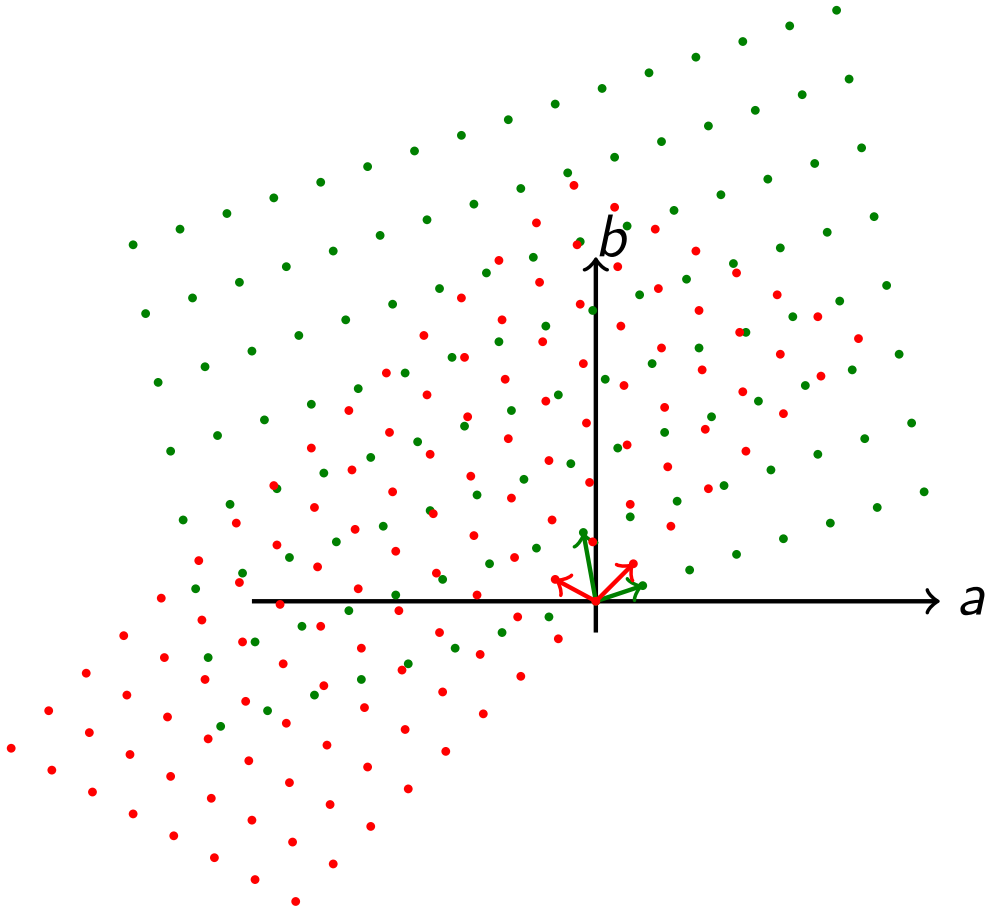
This basis defines the correspondence:

$$(a, b) = i(a_0, b_0) + j(a_1, b_1).$$

We have two 2-dimensional spaces:

- the (i, j) plane: always a fixed-size rectangle.
- and the (a, b) plane. The set of reached points is isotropic.

Special-q



In both cases, the reduced basis defines a change of basis.

q-lattice versus skewness

If our polynomial has skewness S, we have to adapt our lattice reduction in order to reach basis vectors with $a/b \approx S$.

Scalar product for skewed-Gauss

The adaptation to the skewed case simply uses the following alternate scalar product:

$$(a_0, b_0) \cdot (a_1, b_1) = a_0 a_1 + S^2 b_0 b_1.$$

E.g. the vectors $(\sqrt{S}, 1/\sqrt{S})$ and $(-\sqrt{S}, 1/\sqrt{S})$ are orthogonal with respect to this scalar product.

Skewed-Gauss reduction

We expect a pair of vectors (a_0, b_0) and (a_1, b_1) with entries:

$$a_0 \approx a_1 \approx \sqrt{qS}.$$
$$b_0 \approx b_1 \approx \sqrt{q/S}.$$

Note that it doesn't reduce much if $q < S$ (and the orders of magnitude above do not hold), but this case does not occur in practice.

Plan

Sieving and special-q

- Special-q and lattice bases
- Forcing divisibility by an ideal
- Special-q: good or bad

Most ideals are easy ones

Reminder from two weeks ago

An easy ideal $\mathfrak{q}$ represented by $(q, x - r)$ is the prime ideal above q that contains all algebraic integers that are $\mathcal{O}_K$-multiples of $(\alpha - r)$.

This implies that $\mathfrak{q}$ divides $J \times (a - b\alpha)$ if and only if $a/b \equiv r \bmod q$.

Basis of the lattice of (a, b) such that $\mathfrak{q} \mid (a - b\alpha) \times J$:

$$\mathcal{L}_{\mathfrak{q}} : \begin{cases} (q, 0) \\ (r, 1) \end{cases}$$

This can undergo Gauss reduction, and works the same way.

Special-q versus special-$\mathfrak{q}$

Remember: there might be several ideals above the same q.

- On the rational side ($\deg f_0 = 1$), q is enough to refer to a particular set of (a, b) such that $p \mid F_0(a, b)$.
- In contrast, on the algebraic side, we really need the description of an ideal, not just the information of the prime number.
  We may speak of special-$\mathfrak{q}$ sieving in this case (for terminology nerds).

Plan

Sieving and special-q

- Special-q and lattice bases
- Forcing divisibility by an ideal
- Special-q: good or bad

Special-q sieving brings many changes

Special-q sieving is not a way to do the same thing as before.

It is really an important change of the relation collection process.

- The division of the work is not the same as with line sieving.
- The relations that we obtain are different.

What are the advantages and disadvantages of special-q sieving?

Division of the work is different

With special-$\mathfrak{q}$ sieving, we have many $\mathfrak{q}$'s of the same size.

- The yield per $\mathfrak{q}$ is more stable.
- It is easier to make projections of the total yield.

In contrast, line sieving suffers from much more irregular yields, which also drop more quickly.

- The diminishing returns effect is significant.
- Projections are harder to make.

Not the same relations

Cons:

- We miss relations that are very smooth. If the primes involved in the factorization are smaller than all special-$\mathfrak{q}$, the (a, b)-pair belongs to no $\mathfrak{q}$-lattice.
  Such relations are extremely rare anyway.
- Some relations occur several times. If the factorization corresponding to an (a, b)-pair contains two primes of the sizes of the special-$\mathfrak{q}$'s, it belongs to the two $\mathfrak{q}$-lattices.

Pros:

- We know in advance that one norm is divisible by q.
- We avoid considering some positions that are obviously non-smooth. *e.g. when the norm is prime or almost prime.*

Rem. There are more primes and almost primes than very smooth numbers.

Choosing the side of the special-$\mathfrak{q}$

Question. On which side do we put the special-$\mathfrak{q}$?

Consider two numbers; the sum of their sizes is fixed.

Is it more likely for them to be simultaneously smooth if they have the same size or if they are unbalanced?

Choosing the side of the special-$\mathfrak{q}$

Question. On which side do we put the special-$\mathfrak{q}$?

Consider two numbers; the sum of their sizes is fixed.

Is it more likely for them to be simultaneously smooth if they have the same size or if they are unbalanced?

Answer. The concavity of $\log \rho$ tells that it is better to balance the sizes.

Choosing the side of the special-$\mathfrak{q}$

Consequence. Choose q on the side that gives the largest norms.

- In the GNFS case, the algebraic side is the heaviest.
- In the SNFS case, there are low-degree cases where it is better to put the special-$\mathfrak{q}$ on the rational side.

Rem. There might exist cases with no clear answer (esp. SNFS):

- In such cases, alternating the sides of the special-$\mathfrak{q}$ can make sense (recently: HSNFS-1024 DLP).
- Or it is even possible to work with hybrid special-q.
  The set of (a, b) such that $q \mid F_0(a, b)$ and $\mathfrak{q} \mid (a - b\alpha) \times J$ is the intersection of two lattices, and it is in turn a lattice: everything can work pretty much the same way!

Plan

Introduction

Sieving and special-q

The sieving primes

What do we have to do?

We have chosen two basis vectors, and we're going to explore (i, j) in a fixed-size rectangle $[-2^{I-1}, 2^{I-1}] \times [1, J]$. We have:

$$(a, b) = i \cdot (a_0, b_0) + j \cdot (a_1, b_1).$$

On each side, we want to sieve:

- Allocate a big array.
- Initialize each cell with $\log |F(a, b)|$ (for the appropriate F). This is (log-)norm initialization (not today)
- For many primes p, subtract $\log p$ from all array cells when p divides. This is sieving proper (today + Tuesday)

Once this is done, it takes some extra work to list the array cells with the smallest cofactors.

Preferred viewpoint: most general

WLOG, we will assume that we work on the number field side. Everything applies (only simpler, at times) to the rational side as well.

The rational analogue of "the ideal (p, r)" is "the ideal $(p, (m_0/m_1) \bmod p)$". In other words, r is implicit in the rational case, as it is directly inferred from the polynomial.

The sieving primes

Sieving involves a loop over primes. Which primes?

This is an implementation detail of sieving.

- The more relevant quantity globally is the large prime bound.
- Whether sieving deals with prime ideals of norm within one range or another is only of interest to sieving itself.

Terminology

- The factor base is the set of prime ideals that are considered during sieving.
- The sieving bound or factor base bound is the upper bound on the norms of the ideals in the factor base.

Beware

We had (and we still have) $\mathfrak{q}$, which encodes an ideal that we will force into all relations produced.

Now we also consider $\mathfrak{p}$, which is some ideal in the factor base.

So $\mathfrak{p}$ is not $\mathfrak{q}$(also, we want them coprime).

Challenge

Within the (i, j) rectangle, we want to identify locations where $\mathfrak{p}$ divides $(a - b\alpha) \times J$.

The $\mathfrak{p}$-lattice

Let $\mathfrak{p}$ be an ideal to be sieved.

Fact: The set of positions in the (i, j)-plane where $\mathfrak{p}$ divides $(a - b\alpha) \times J$ is a lattice $\mathcal{L}_{\mathfrak{p}}$.

We already encountered this for the easy primes, but this holds more generally.

- Each (power of a) prime ideal of inertia degree one gives rise to a lattice in the (a, b) plane.
- We'll see how it connects to the (i, j) plane.

Plan

The sieving primes

- Sieving primes in the (a, b) plane
- From (a, b) to (i, j)

prime ideal $\to$ lattice in (a, b) plane

Given a prime ideal $\mathfrak{p}$ above p.

- $\mathfrak{p}J^{-1} \cap (\mathbb{Z} + \alpha\mathbb{Z})$ is a lattice. It has a basis.
- Some cases are uninteresting: if $\mathfrak{p}$ has inertia degree > 1, then intersection points only have $p \mid \gcd(a, b)$ (except possibly if $\mathfrak{p} \mid J$).

The description of the basis in the (a, b) plane only involves some non-trivial work for the rare non-easy ideals. Anyway it can be done beforehand.

In effect, we are interested in the description of sets of sieving locations, and we know that each of these is going to be a lattice in the (a, b) plane.

Example of description

Example

Let $f = 3x^4 + x^3 + x^2 + x + 1$. Algebraic number theory tells us that $3\mathcal{O}_K$ splits into three ideals, of norm 3, 3, and 9.

$$3\mathcal{O}_K = \mathfrak{p}_1\mathfrak{p}_2\mathfrak{p}_3.$$

- $\mathfrak{p}_1 \mid (a - b\alpha) \times J$ iff $a \equiv b \mod 3$.
- $\mathfrak{p}_2 \mid (a - b\alpha) \times J$ iff $b \equiv 0 \mod 3$.
- $\mathfrak{p}_3$ never divides $(a - b\alpha) \times J$, unless a, b are both multiples of 3 (and we disregard this case).

Another example of description

Example

Let $f = 9x^4 - x^2 - 5$. Algebraic number theory tells us that $3\mathcal{O}_K$ splits into four ideals, each of norm 3.

$$3\mathcal{O}_K = \mathfrak{p}_1\mathfrak{p}_2\mathfrak{p}_3\mathfrak{p}_4.$$

- $\mathfrak{p}_1 \mid (a - b\alpha) \times J$ iff $a \equiv b \mod 3$.
- $\mathfrak{p}_2 \mid (a - b\alpha) \times J$ iff $a \equiv -b \mod 3$.
- Both $\mathfrak{p}_3$ and $\mathfrak{p}_4$ divide $(a - b\alpha) \times J$ iff $b \equiv 0 \mod 3$.
  - $\mathfrak{p}_3^2 \mid (a - b\alpha) \times J$ iff $3a - b \equiv 0 \mod 9$.
  - $\mathfrak{p}_4^2 \mid (a - b\alpha) \times J$ iff $3a + b \equiv 0 \mod 9$.

Compact encoding

All congruences we have to deal with in the (a, b) plane (for prime ideals and their powers) are of the form:

$$\lambda a - \mu b \equiv 0 \mod p^k.$$

- If $p \nmid \lambda$, WLOG we can assume $\lambda = 1$. This encodes the most common case of easy ideals ($(p, r) \to a - rb \equiv 0 \mod p$), and this extends to powers with $0 \leq r < p^k$.
- Or $p \mid \lambda$, whence $p \nmid \mu$ and WLOG we can assume $\mu = 1$. We obtain a relation of the form

  $$psa - b \equiv 0 \mod p^k$$

  with p^{k-1} possible choices for s. (Note: s not necessarily coprime to p.)

Compact encoding

Another way to look at this mathematically speaking is to relate this with the space $\mathbb{P}^1(\mathbb{Z}/p^k\mathbb{Z})$ which has $p^k + p^{k-1}$ elements:

- First case: the affine point $(r : 1)$ in $\mathbb{P}^1(\mathbb{Z}/p^k\mathbb{Z})$.
- Second case: the point $(1 : ps)$ in $\mathbb{P}^1(\mathbb{Z}/p^k\mathbb{Z})$ which is "at infinity". In such cases, the NFS folklore uses the term projective roots (they exist only projectively).

In Cado-NFS, the computation of the factor bases is done prior to sieving, and gives for each prime power lists of one of the two encodings above.

Compact encoding $\rightarrow$ lattice

Affine case

For a set of sieving locations modulo p^k, described in compact encoding by an affine integer $r < p^k$, the lattice basis is

$$\begin{cases} (a_0, b_0) = (p^k, 0) \\ (a_1, b_1) = (r, 1) \end{cases}$$

Projective case

For a set of sieving locations modulo p^k, described in compact encoding by an integer $s < p^{k-1}$ which denotes the point $(1 : ps)$ on the projective line, the basis is (assuming $\nu_p(ps) = c > 0$):

$$\begin{cases} (a_0, b_0) = (p^{k-c}, 0) \\ (a_1, b_1) = ((ps/p^c)^{-1} \bmod p^{k-c}, p^c) \end{cases}$$

Four points of view

- More mathematical: some power of a prime ideal.
- Also mathematical: a point in $\mathbb{P}^1(\mathbb{Z}/p^k\mathbb{Z})$.
- More down-to-earth: a lattice basis.
- More compact: an integer between 0 and $p^k + p^{k-1}$.
  E.g. by letting $p^k + s$ encode $(1 : ps)$

These are equivalent ways of describing the same thing: a set of (a, b) pairs where we know that some divisibility condition is met.

Plan

The sieving primes

Sieving primes in the (a, b) plane

From (a, b) to (i, j)

Where do we sieve in the (i, j) plane?

$\mathcal{L}_{\mathfrak{p}}$: locations of interest in the (a, b) plane.
$\mathcal{L}_{\mathfrak{p}} \cap \mathcal{L}_{\mathfrak{q}}$ has a basis in the (i, j) plane.

$$(a, b) = i \cdot (a_0, b_0) + j \cdot (a_1, b_1).$$

Example: affine case

$$\begin{aligned} a - rb &\equiv 0 \mod p^k \\ \Leftrightarrow (ia_0 + ja_1) - r(ib_0 + jb_1) &\equiv 0 \mod p^k \\ \Leftrightarrow i(a_0 - rb_0) + j(a_1 - rb_1) &\equiv 0 \mod p^k \\ \Leftrightarrow i - Rj &\equiv 0 \mod p^k \end{aligned}$$

with $R \equiv -\dfrac{a_1 - rb_1}{a_0 - rb_0} \mod p^k$ if the denominator is invertible!

Translation from (a, b) to (i, j)

This is preparatory work that must be done for each special-$\mathfrak{q}$:

Transforming the factor base

For each prime ideal (power) or more down-to-earth representation in the (a, b) plane, compute the down-to-earth representation in the (i, j) plane.

Anything can happen:

- affine in $(a, b) \to$ affine in (i, j).
- affine in $(a, b) \to$ projective in (i, j).
- projective in $(a, b) \to$ affine in (i, j).
- projective in $(a, b) \to$ projective in (i, j).

Ready!

At this point, our gear is packed, we're ready to sieve.

We have:

- a huge array $T[]$, indexed by (i, j).
  $T[]$ can be up to several gigabytes of RAM.

And we also have:

- A (long) list of prime (powers) p^k (not that we must sieve powers, but for sure we can).
- Each comes with a descriptions of the location of hits: places in the (i, j) plane where we want to subtract $\log p$.

CSE291-14: The Number Field Sieve

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

Emmanuel Thomé

February 8, 2022

Part 5b

Collecting relations in NFS

Sieving strategies

Bucket sieving and the Franke–Kleinjung enumeration

General organization of the siever, and main parameters

Multithreading and more implementation details

Multi-layer bucket sieving

Using sieving AND batch smoothness detection

Plan

Sieving strategies

Bucket sieving and the Franke–Kleinjung enumeration

General organization of the siever, and main parameters

Multithreading and more implementation details

Multi-layer bucket sieving

Using sieving AND batch smoothness detection

Various sieving strategies

Depending on p^k, we will use varying strategies:

- for very small p, we will use pattern-sieving;
- mildly larger primes (say up to 2^I) are done with line sieving ("small sieve" in Cado-NFS).
- even larger primes are sieved with bucket sieving, which can even be done in several stages.

Plan

Sieving strategies

- Pattern sieving
- Line sieving ("small sieve")
- Dividing into small regions

Pattern sieving

Idea. When p is tiny, there are many hits, and those are very close to each other.

We can use the fact that a processor likes to work with (64-bit) machine words rather than single bytes.

Used in Cado-NFS for $p = 2, 3, 5, 7$ and small powers.

Pattern sieving for 2 and its powers

For powers of 2:

- Prepare a block of 16 bytes.
- From the factor base data, we know where we should subtract $k \log 2$, for $k = 1, 2, 3$. Store this info in the block.
- Subtract this block from the sieving space, 2 unsigned long at a time.
- Alignment of the pattern changes with j.

Rem. No carries! Guaranteed! We do an `unsigned long`-level subtraction to emulate 8 independent byte subtractions.

Pattern sieving for 3

For 3:

- Still want to use unsigned longs. But the pattern must have a number of bytes that is divisible by 3.
- Use a block of 3 unsigned longs.
- Prepare and apply block as before.
- Alignment of the pattern changes with j.

More pattern sieving

Cado-NFS has more pattern sieving:

- Use SSE2 / AVX / ... to apply the pattern.
- Handle larger primes (3,5,7).
- Instead of doing the pattern separately for each prime, use just one longer pattern (like in wheel sieve).

Pros/Cons

Obviously, pattern sieving is only suitable for (very) small primes.

Plan

Sieving strategies

- Pattern sieving
- Line sieving ("small sieve")
- Dividing into small regions

Line sieving ("small sieve")

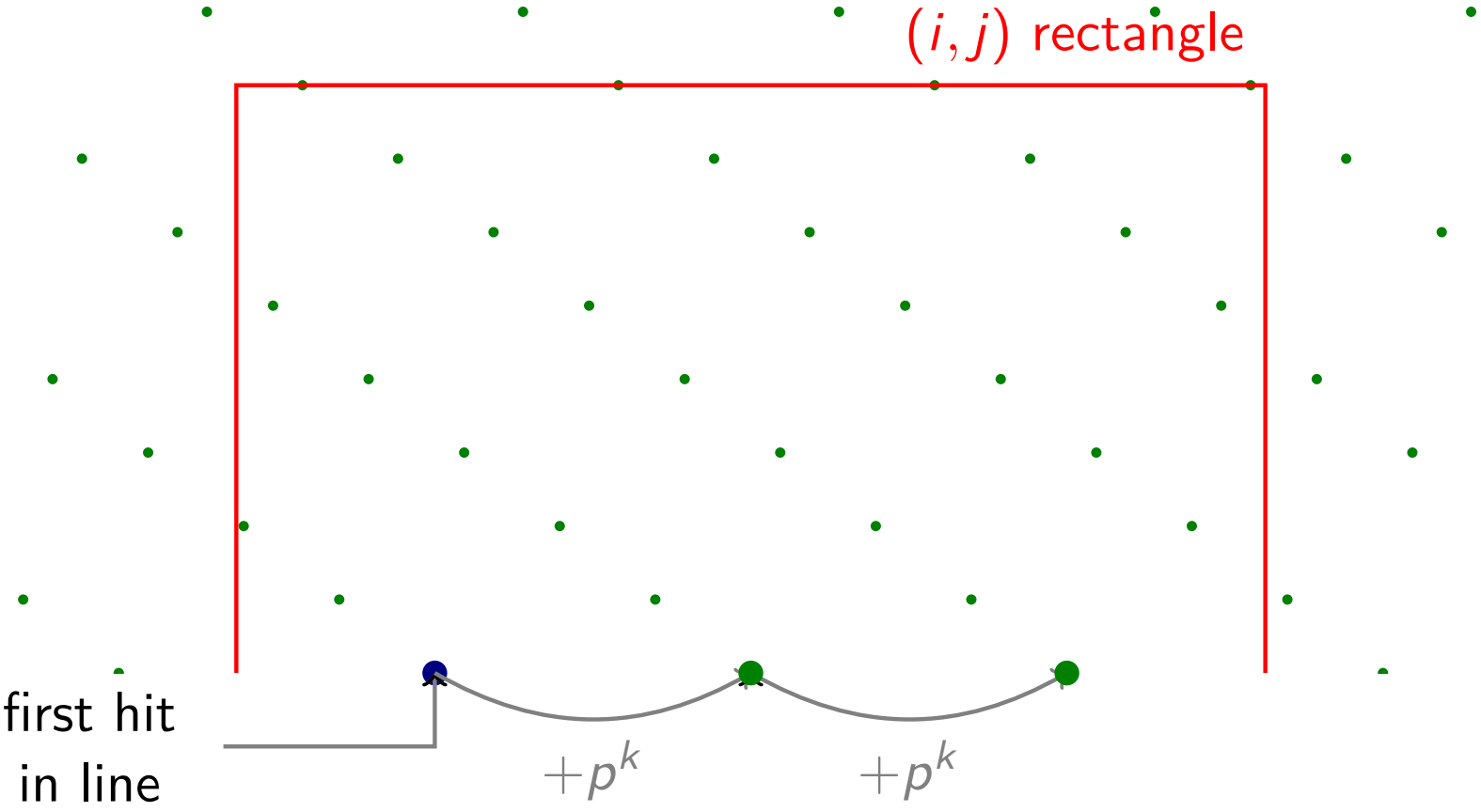
Warning. We use the terminology "line sieving" for a part of the lattice sieving. Do not forget we are still in a special-$\mathfrak{q}$ sublattice.

Context. This is used when $p^k < 2^I$.

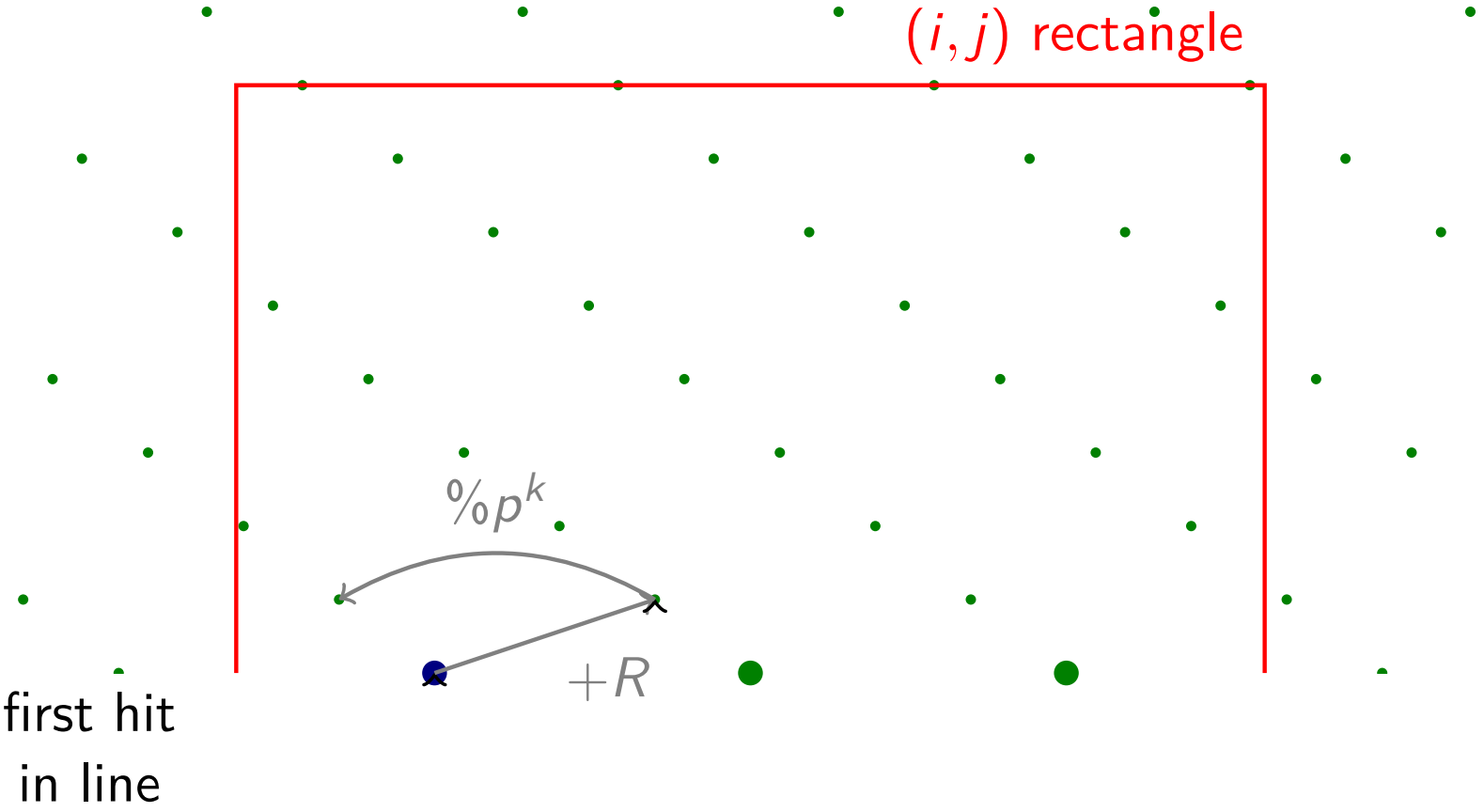
There is at least one hit per line of the (i, j)-plane.

Handle the j-lines one after the other (hence the name!).

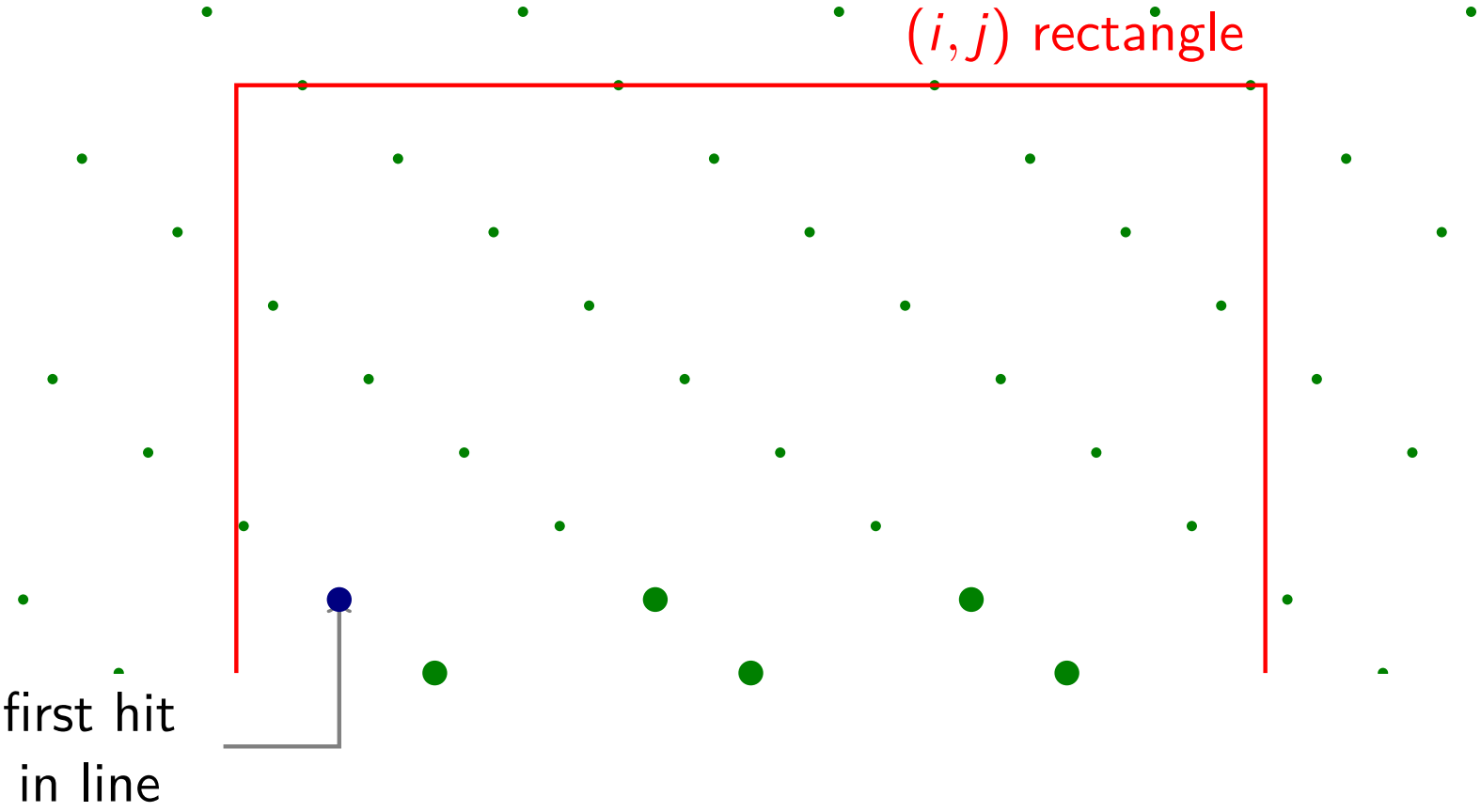
Line sieving ("small sieve")



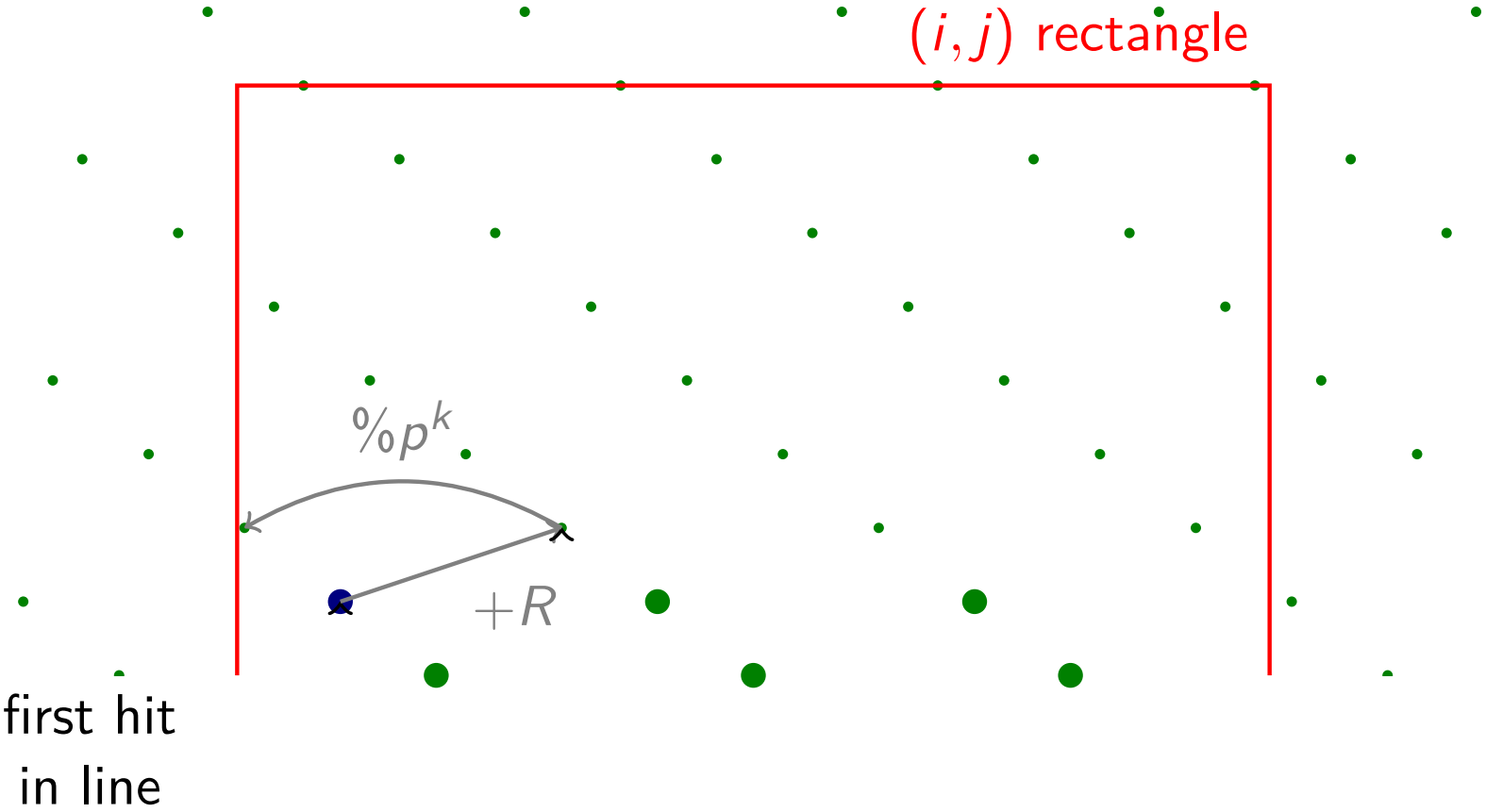
Line sieving ("small sieve")



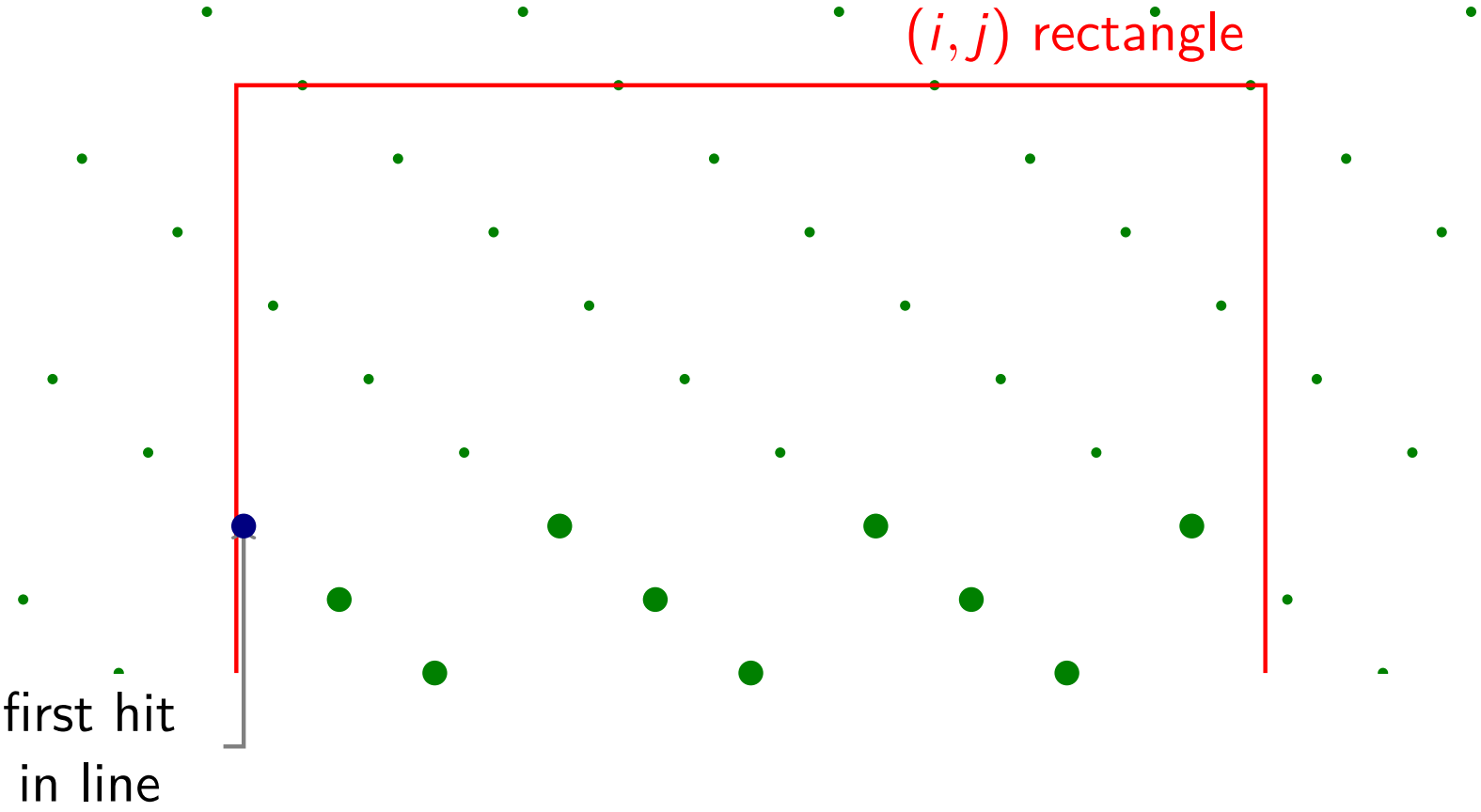
Line sieving ("small sieve")



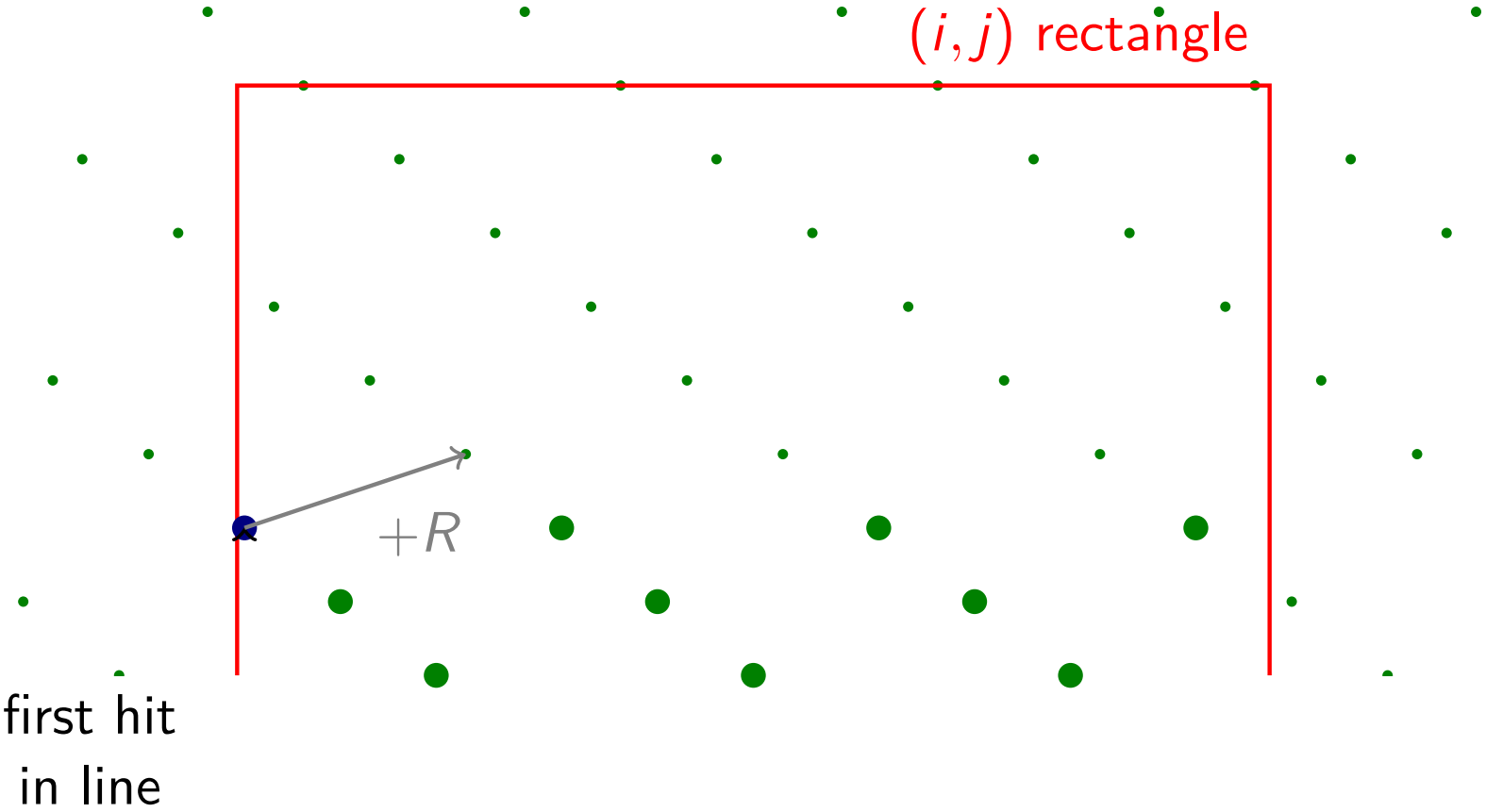
Line sieving ("small sieve")



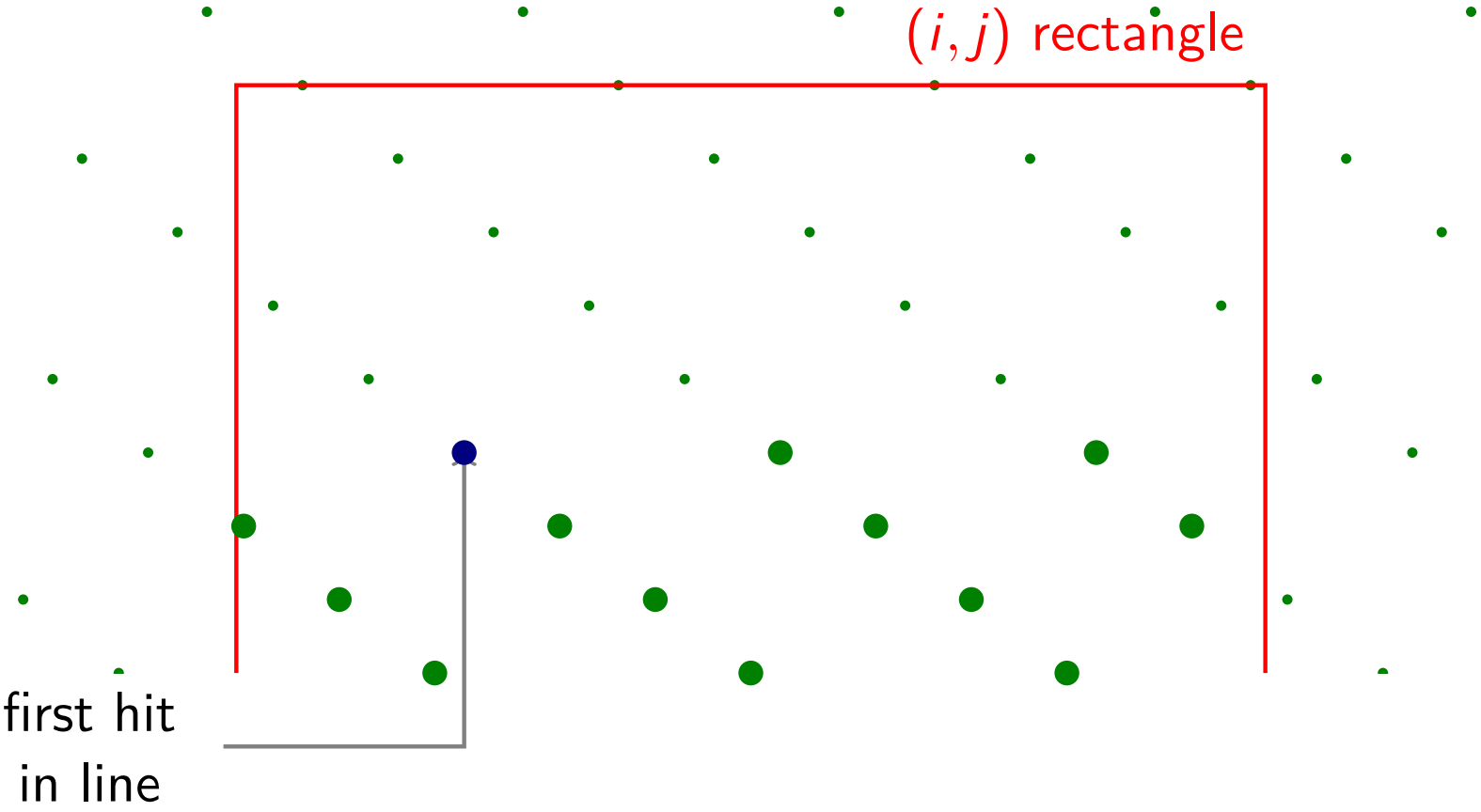
Line sieving ("small sieve")



Line sieving ("small sieve")



Line sieving ("small sieve")



Line sieving

1. Initialize: $j = 0$, $t = -2^{l-1} + (2^{l-1} \bmod p^k)$.
2. *[work on the current line]*
3. set $u = t$ *[first hit in line]*
4. while $u < 2^{l-1}$,
5. subtract $\log p$ at position (u, j);
6. $u \leftarrow u + p^k$
7. Increment j, stop if $j \geq J$.
8. $t \leftarrow t + R$ *[first or maybe second hit]*
9. if $t \geq p^k$, then $t \leftarrow t - p$ *[first hit, really]*
10. Go to step 2.

Line sieving

Tricks:

- For j even, we sieve only odd i (and add $2p^k$ instead of p^k).
- In the projective case (when $p \mid j$ is a necessary condition), it makes sense to adapt the algorithm, and not go through lines which have assuredly no hit.
  Note that this is a rare case, and it is not a problem if it is dealt with by special code.
- For a given small p^k, since there are many hits, we have some positive aspects implementation-wise:
  - Computation time is spent in the tightest loop.
  - Memory accesses are close to each other.

Plan

Sieving strategies

- Pattern sieving
- Line sieving ("small sieve")
- Dividing into small regions

The sieve area is too large

Problem: The sieve area $2^A = 2^I \times J$ is often large.

If we line-sieve factor base primes for the whole (i, j) rectangle, then despite the good things that we mentioned, we have bad cache behaviour because we will traverse a multi-GB memory area over and over again.

Full memory traversal is expensive: it causes all memory caches to be emptied and refilled constantly!

The sieve area is too large

Dividing into small regions

It is better to split the sieve area in pieces.

- we typically handle sub-areas of the (i, j) rectangle 64 KB at a time.
  - Index within such an area can be 16 bits.
  - Always fits in L2 cache.
  - L1 cache is often smaller, though.
- our hope is that this will minimize cache misses.

As I grows, line sieving only on regions of 64 KB at a time means:

- we deal with only a fixed number of lines at a time.
- Extreme cases:
  - 1 line when $I = 16$
  - line fragments when $I > 16$!

Sieving regions: doing it right

Challenge: for each new region:

- we must resume all small sieve computation where we left them.
- this means: store the "first hit in line" as it would have been deduced, had we kept going.

This is reasonable as long as we have something to do with line-sieved primes in each region.

Clearly, if $p^k \geq$ region size, we do not want to go through all this.

Plan

Sieving strategies

Bucket sieving and the Franke–Kleinjung enumeration

General organization of the siever, and main parameters

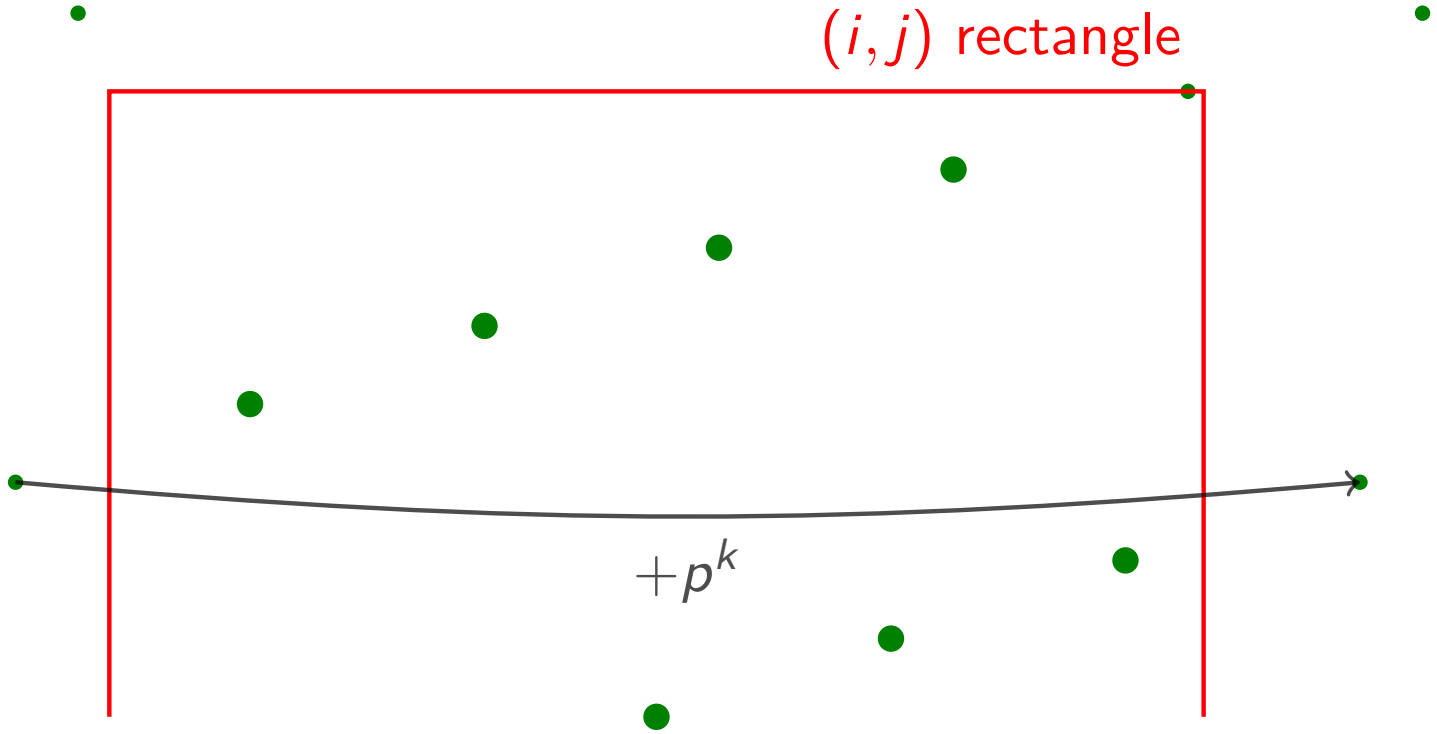
Multithreading and more implementation details

Multi-layer bucket sieving

Using sieving AND batch smoothness detection

Many misses

Problem: when $p^k > 2^l$, there are lines with no hit.



Worse: when $p^k \gg 2^l$, almost no line has a hit. For p^k larger than the region size, not even every region has a hit.

A new enumeration for primes above 2^I

Idea

Because p^k is larger than the width of the (i, j) rectangle, there might be a way to enumerate all hits efficiently.

The answer is the Franke–Kleinjung algorithm:

- Goal: obtain a basis of the lattice that is well adapted to enumerating the hits.
- We intend to run that in a tight loop, and infer the list of (possibly rare) regions where hits will occur.

Note: most of the following description assumes $k = 1$. The case $k > 1$ for such p is only WIP in Cado-NFS anyway at the moment, and is not sure to be worthwhile.

Franke–Kleinjung's lattice sieving

Idea. Adapt the stopping criterion in Gauss algorithm to get a basis adapted to the width 2^I of the search space.

Lemma

There exists a basis $\langle v_0 = (\alpha, \beta), v_1 = (\gamma, \delta)\rangle$ of $\mathcal{L}_{\mathfrak{p}}$ such that

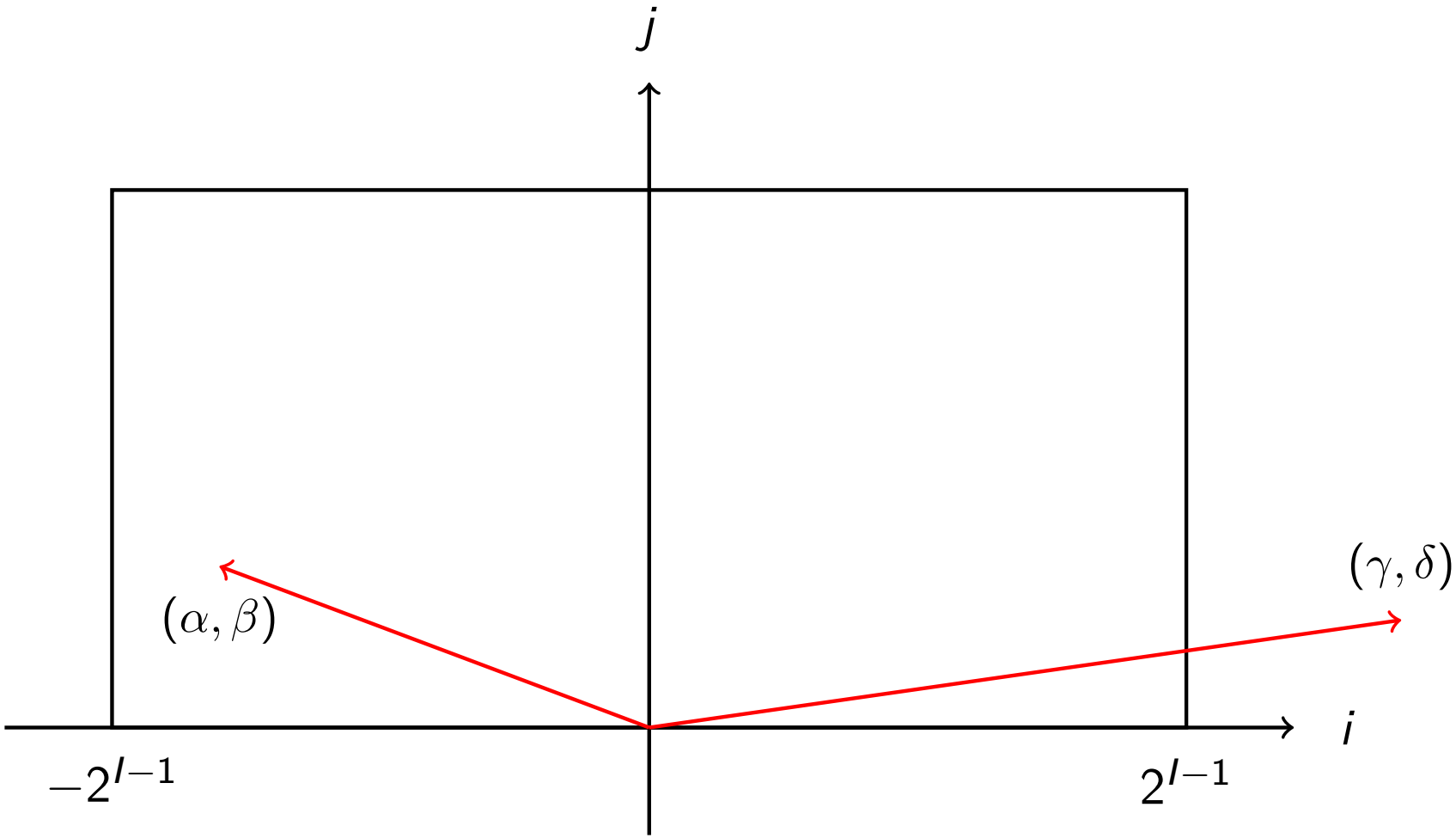
- *β and δ are positive;*
- $-2^I < \alpha \leq 0 \leq \gamma < 2^I$;
- $\gamma - \alpha \geq 2^I$.

The proof is neither really complicated, nor enlightening.

Implementation.

- Easy to get something that works; similar to Euclidean algorithm;
- Enumerating is costly because we have many primes: many hits to process.

Franke–Kleinjung's lattice sieving



Franke–Kleinjung's lattice sieving

Fact. This basis allows to iterate on the points of $\mathcal{L}_{\mathfrak{p}}$ that are exactly in the (i, j)-rectangle with no branching.

Let (i, j) be a valid hit. At most one of the following vectors is valid:

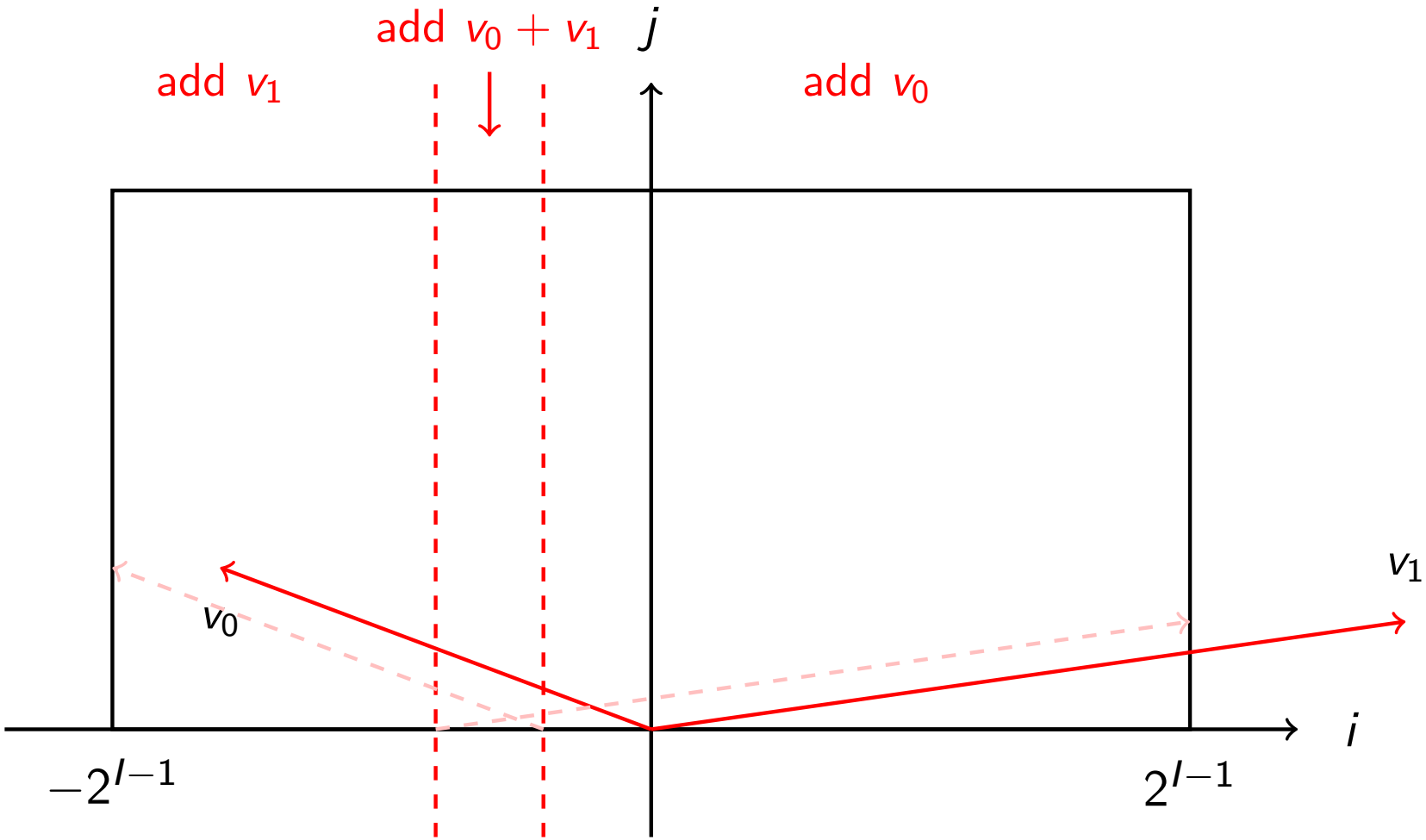
- $(i, j) + (\alpha, \beta)$;
- $(i, j) + (\gamma, \delta)$;

And if the two above vectors are invalid, then the following is valid:

- $(i, j) + (\alpha, \beta) + (\gamma, \delta)$.

Fact. Applying this rule starting with $(0, 0)$, we enumerate all points of $\mathcal{L}_{\mathfrak{p}}$ in the (i, j) rectangle.

Franke–Kleinjung's lattice sieving



Franke–Kleinjung's lattice sieving

Tricks:

- Deciding which vectors to add depends on easy bounds on i.
- This can be implemented in a branch-free way using `cmov`'s (nowadays, the C compiler does it for you).
- Assume that the memory layout is such that $T[(i,j)]$ is at address $j \cdot 2^I + (i - i_{\min})$. Then adding a vector to an index to the array T is done by adding an integer.

Handling the memory locality question

Problem. When $\mathfrak{p}$ is large, each hit in the (i, j)-rectangle is a cache miss.

If not careful, a lot of time is spent fetching remote memory into cache in order to subtract $\log p$ in the appropriate cell.

Think different!

- In term of information, the data produced by the sieve is a (huge) set of sieve updates, i.e. pairs (location, contribution).
- Instead of using directly the sieve update, store them for future use.
- Before applying the updates to the sieve array, sort them according to increasing locations.
- Then, applying the $\log p$'s contributions will be cache-compliant.

Handling the memory locality question

But. Sorting is not really local, is it?

- Sorting is a very well studied topic, and there exist variants that are local.
- For instance, `merge_sort` is quasi-linear even on a Turing machine.
- Even better: we do not need a perfect sorting, since at the L1 cache level, we can consider to have random access.

Solution: Bucket sorting.

Bucket sieving

We reuse our division of the (i, j) rectangle into regions.

Def. A region is a set of contiguous j-lines of the (i, j)-rectangle that fits (more or less) in the L1 cache.

The sieve array is therefore split in many regions.

- Prepare k buckets, i.e. storage for lists of sieve updates, each bucket corresponding to a region.
- Run the sieving à la Franke–Kleinjung for each $\mathfrak{p}$ in the factor base; for each hit, append the sieve update in the appropriate bucket.
- For each region, apply all the updates of the corresponding bucket.

Big steps of bucket sieving

1. Allocate buckets
2. **Fill buckets**: for both sides, and for all prime ideals above the bucket threshold,
   - Iterate through locations of updates with the Franke–Kleinjung enumeration.
   - Append each update in the right bucket (lower bits of memory location and some info related to p).
3. Loop over all regions (64 KB each):
   - Initialize (log-)norms.
   - **Apply buckets**: read locations from buckets, do subtractions.
   - do the line sieving for the small primes (both sides), for this region only.
   - Look for survivors that deserve further investigation.

Bucket sieving

Analysis.

When sieving, we have k pointers (one for each bucket) advancing in parallel.

If the cache can handle k cache lines, there are no misses.

Furthermore, processors tend to be optimized for linear memory access: we can hope for automatic prefetching.

Caveat.

Ever heard about TLB? (Translation look-aside buffer)

This is a key element of the virtual memory mechanism. One can think of it as a cache for the big table that contains the correspondence physical / logical addresses.

If the number of buckets k is larger than the TLB, we often have TLB misses.

Bucket sieving — details

Bucket updates:

- Buckets take a lot of memory (but then we do not need all the sieving area in memory).
- Do not need to store the high bits of j (= bucket id).
- Do not need to store $\log p$. Indeed, primes are sieved in increasing order, so we can just remember the few events when $\log p$ increases in a bucket.
- For re-sieving (later), it would be nice to have p, but store only 16 bits of p, called a hint.
- Conclusion. Only 32 bits per update.

Plan

Sieving strategies

Bucket sieving and the Franke–Kleinjung enumeration

General organization of the siever, and main parameters

Multithreading and more implementation details

Multi-layer bucket sieving

Using sieving AND batch smoothness detection

Recap

We have seen so far:

- How we sieve, for tiny, for small, and also for larger primes.
- The (i, j) sieving rectangle is divided into regions, typically of 64 KB.
- The bucket sieving process computes in advance bucket updates. Buckets are attached to regions.

Recap

We thus have the bits and pieces for an algorithm that identifies (i, j) coordinates for which...

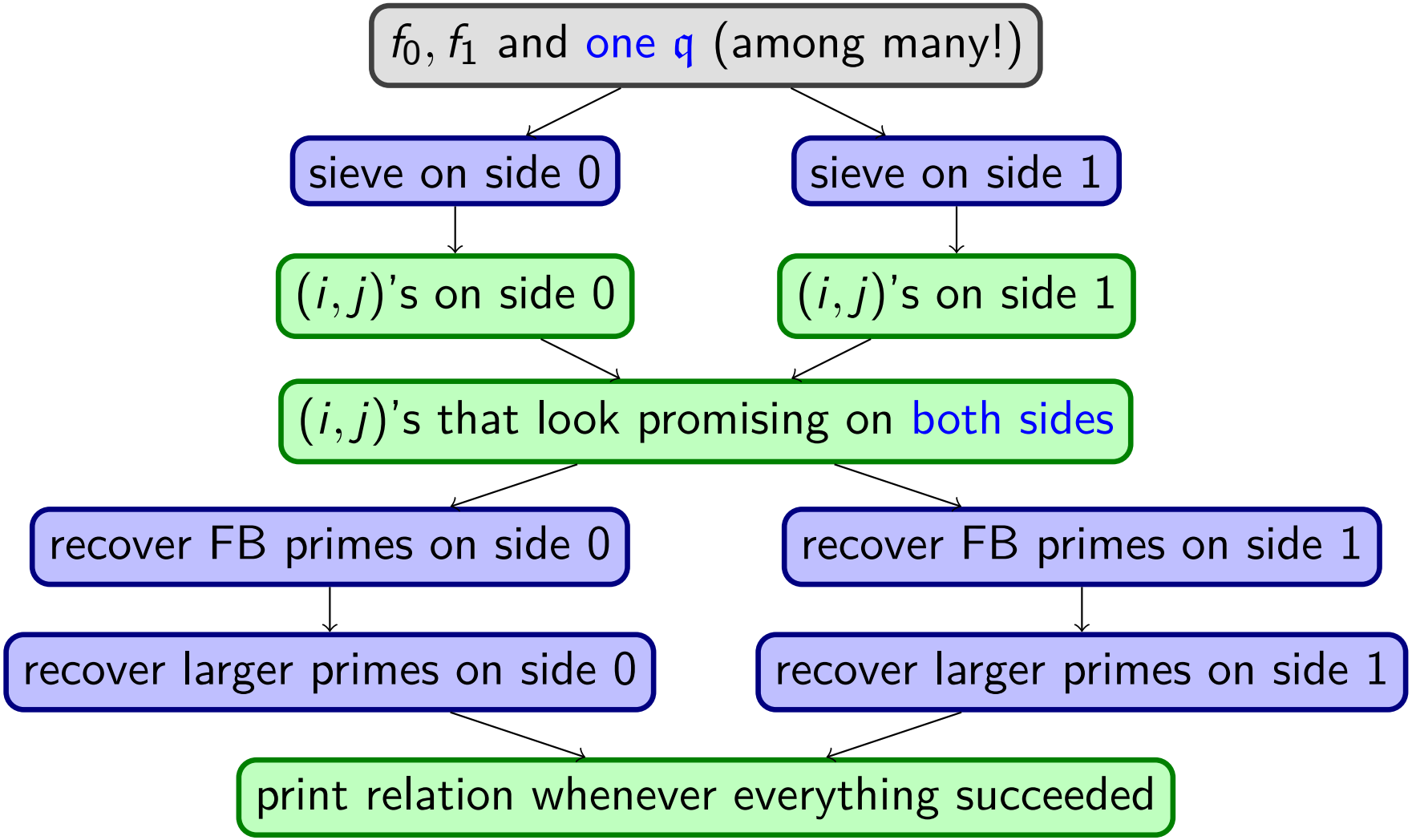
- $\mathrm{Res}(a - bx, f_0(x))$ and $\mathrm{Res}(a - bx, f_1(x))$ are both smooth;
- or at least, both have a large contribution from FB primes. In this last case the norms might be almost smooth, and it might well be that this is good enough for our purposes.

Recap

Next: we need to recover the complete factorization for all these promising locations (i, j).

- Some factors come from the factor base: some work is needed to find them again: re-sieving.
- When norms are just promising, not completely smooth, cofactorization will be used to find the remaining factors.

Graphically



Parameters

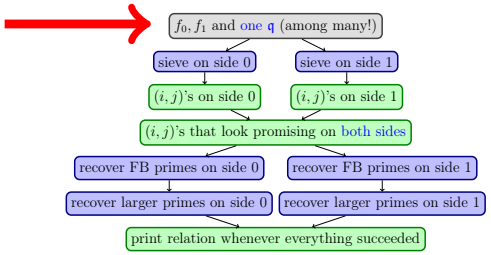
Cado-NFS's lattice siever is called `las`.

The `las` program is quite versatile, and is controlled by a host of (command-line) parameters.

Almost every transition in the previous slide can be controlled by a parameter.

Parameters: the special-q's

Several parameters can be used to select q's.



First: `sqside` is (integer) side 0 or 1.

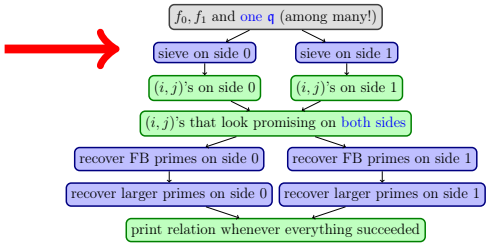
`q0`, `q1`: all special-q in the range $q \in [q_0, q_1)$.

`q0`, `q1`, and `-random-sample` n:
only n special-q's, evenly spaced across $[q_0, q_1)$.

`q0`, `rho`: just one single special-q, the ideal $(q_0, x - \rho)$.

`las` also agrees to work with an arbitrary list of special-q's given by the `todo` parameter.

Parameters: the sieving parameters



`I` or `A`: size of the (i, j) sieving area.

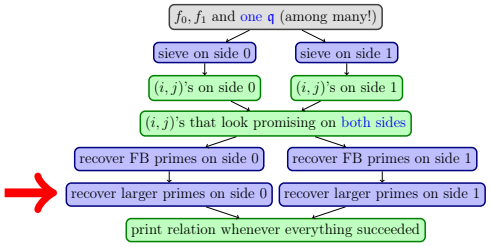
`lim0`, `lim1`: factor base bounds on both sides.
We sieve for $p <$ `lim0` on side 0.

`bkthresh`: when we start to use Franke–Kleinjung enumeration (a good default value is 2^I);

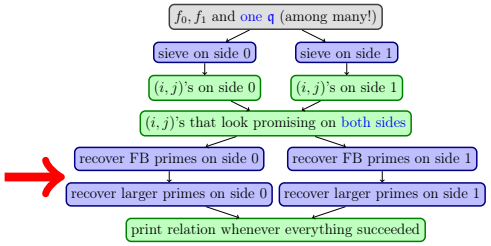
`tdthresh`, `bkthresh1`, `adjust-strategy`: more advanced parameters.

Parameters: large primes

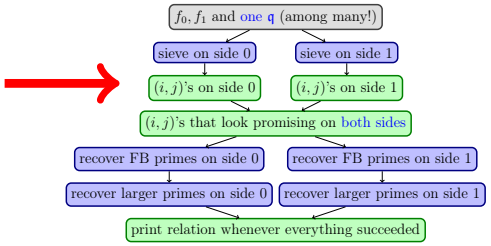
`lpb0`, `lpb1`: bit size of the largest primes that we tolerate in relations.



`mfb0`, `mfb1`: maximum bit size of norms after removing all sieved primes.



Parameters: after-sieve threshold



`lambda0`, `lambda1`: coarse-grain test.

We would like to keep (i, j) whenever the unsieved part is below `mfb` on both sides.

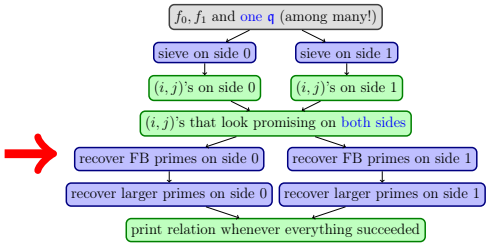
- However we do not know the bit size of the unsieved part, just a coarse approximation of it.
- We compare this approximation with `lambda` $\times$ `lpb`. `lambda` defaults to `mfb`/`lpb`, but it is allowed to put explicit values.

Re-sieving

If a cell passes the λ-threshold on both sides, we need to compute the corresponding norms and test them for smoothness according to the large prime bound: need exact cofactor.

- Norm computation: not so much tricks to do, since we need an exact result.
- We handle a region at once and deal with all the survivors: we can therefore do again the line sieving for the small primes (and divide the norm by p).
- The buckets are also applied again, but this time the primes are reconstructed from the hints and trial divided in the norm.
- The remaining part of the norms are tested against the exact threshold, and possibly factored with ECM (cofactorization).

Parameters: trial division



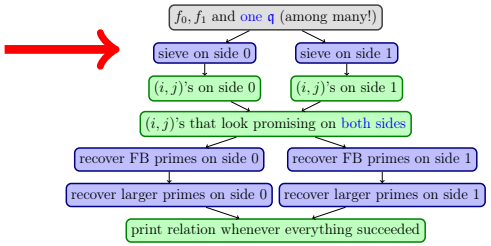
`tdthresh`:

For the smallest primes, divide
them out at the promising (i, j) positions by trial division only.

Rationale: when p is really small, it is quick enough to trial-divide all the survivors.

Wait, we have two sides!

The activity on both sides is interleaved.



1. Fill the buckets on side 0;
2. Fill the buckets on side 1;
3. Loop on each region:
4. Initialize norms on side 0;
5. Pattern- and line- sieve on side 0;
6. Apply bucket updates on side 0;
7. Initialize norms on side 1 (only survivors);
8. Pattern- and line- sieve on side 1;
9. Apply bucket updates on side 1;
10. Cofactorize both sides (includes re-sieving).

Plan

Sieving strategies

Bucket sieving and the Franke–Kleinjung enumeration

General organization of the siever, and main parameters

Multithreading and more implementation details

Multi-layer bucket sieving

Using sieving AND batch smoothness detection

How do we parallelize?

1. Allocate buckets
2. **Fill buckets**: for both sides, and all (large-ish) prime ideals

   ...
3. Loop over all regions (64 KB each):
   - **Apply buckets**
   - and do very local things on with this 64 KB region.

Parallelize in "two directions"

We parallelize steps 1 and 2 in one direction, and step 3 in another.

Multi-threading

This way of organizing the computation is "easy" to parallelize in a shared-memory environment.

- Steps 1 and 2 (allocate / fill buckets): each thread has its own set of buckets and takes care of part of the factor base.
- Thereafter, all buckets are readable by everyone.
- Step 3: Region are processed independently by all threads.
  - Region r is processed by thread $r \bmod T$ (if T threads).

Advantage: The memory pressure per core is reduced.

Splitting the factor base

For steps 1 and 2, we want to split the factor base in many "slices" so that, for the overhwelming majority of them:

- slices contain roughly an equal number of prime ideals.
- slices contain no more than 2^{16} prime ideals.
- all prime ideals in a slice are easy ideals, and not projective roots in the (i, j) plane.
- all prime ideals in a slice have equal $\lfloor \log_\beta |\mathrm{Norm}(\mathfrak{p})| \rceil$.
- all prime ideals in a slice are above prime number with equal number of roots modulo p.

These restrictions are here so that the inner loop can be streamlined as much as possible.

Processing regions in parallel

Thread k deals with regions $k, k+T, \ldots$.

- To make the "small sieve" efficient, the per-line first hits for each prime must be remembered from one region to the next. Some annoying arithmetic adjustments are necessary.
- It gets worse when lines are larger than one region.
- More pragmatic approach: precompute the starting points for all "small-sieved" primes and all regions. That's cheap enough.

Plan

Sieving strategies

Bucket sieving and the Franke–Kleinjung enumeration

General organization of the siever, and main parameters

Multithreading and more implementation details

Multi-layer bucket sieving

Using sieving AND batch smoothness detection

Memory footprint

With bucket sieving, most of the memory goes into the storage of bucket updates.

$$\text{number of updates} \approx \#\mathcal{A} \times \sum_{\substack{\mathfrak{p} \in \mathcal{F}_0 \\ \mathfrak{p} \text{ bucket-sieved}}} \frac{1}{|\mathrm{Norm}\,\mathfrak{p}|}.$$

$$\approx \#\mathcal{A} \times (\log\log \mathtt{lim} - \log\log \mathtt{bkthresh}).$$

Orders of magnitude:

- $\#\mathcal{A} \approx 2^{34}$. `lim` around 2^{32}.
- $I \approx 16$ and $\mathtt{bkthresh} = 2^I$.
- $\log\log 2^{32} - \log\log 2^{16} \approx 0.69$
- 4 bytes per update. Total about 13G.

Too many buckets at a time

When $\#\mathcal{A} \approx 2^{34}$ and regions are 64 KB:

- That makes about 2^{18} buckets that we are writing to.
- (way) more than the CPU can handle efficiently.

We can alleviate both problems at once with multi-layer bucket sieving.

Multi-layer

Simple idea (but not trivial implementation-wise):

- For primes above some threshold (say about 2^{I+8}), dispatch into "big buckets" that correspond to "big regions", equivalent to 2^8 regions at a time.

  Cado-NFS calls this threshold `bkthresh1`.

- Subdivide the region processing into blocks 2^8 regions. When we begin such a block:
  - we bucket-sieve the primes between `bkthresh` and `bkthresh1`.
  - and do a secondary pass on the updates from the current "big bucket".

How do we handle prime ideals?

- Largest factor base prime ideals.
  - $\#\mathcal{A} \times (\log\log \texttt{lim} - \log\log \texttt{bkthresh1})$ updates in total.
  - Dispatched at the beginning of sieving into $\lceil \#\mathcal{A}/2^{24} \rceil$ "big buckets".
  - Re-dispatched in a second pass when we begin processing the first region in a batch of 2^8.
- Prime ideals between `bkthresh` and `bkthresh1`.
  - Dispatched among the "next 2^8 regions" only when we're about to process them.
  - This is fine, since these have many hits anyway.
  - Need memory for 2^{24} $(\log\log \texttt{bkthresh1} - \log\log \texttt{bkthresh})$ updates.
- Primes below 2^I are line-sieved only when we process a region.

Pros/Cons

Two advantages:

- We limit the number of buckets that we are writing to.
- The memory cost is reduced:

$$\text{number of updates} \approx \#\mathcal{A} \times (\log\log \texttt{lim} - \log\log \texttt{bkthresh1}) + 2^{24}\,(\log\log \texttt{bkthresh1} - \log\log \texttt{bkthresh})\,.$$

Downside: the processing of the large primes requires two steps.

Two-layer bucket sieving matters a lot in large computations.

- it can probably be improved: the current "hard" cutoffs are not ideal.
- it might make sense to have three layers at some point.

Plan

Sieving strategies

Bucket sieving and the Franke–Kleinjung enumeration

General organization of the siever, and main parameters

Multithreading and more implementation details

Multi-layer bucket sieving

Using sieving AND batch smoothness detection

Use case

Two fundamental differences between time and memory requirements of sieving and batch smoothness detection.

- Sieving: dependence on $\#\mathcal{A}$.
- Batch smoothness detection: dependence on the number of integers we want to test for smoothness.

Since NFS sieving has two sides, it makes sense to:

- Do sieving on the "hard" side.
- Once the rare survivors are identified, feed them to batch smoothness detection.
- Batch smoothness detection can even run asynchronously: do a dozen special-q's, then do batch smoothness detection on the survivors, etc.

Part 5c

Collecting relations in NFS: two further topics

Norm initialization

Adjusting the $\mathfrak{q}$-lattice

Plan

Norm initialization

Adjusting the $\mathfrak{q}$-lattice

Norms: what is it about?

Context

- We have the polynomials f and g.
- We have chosen a special-$\mathfrak{q}$ on one of the two sides. So we are working with

$$(a, b) = (i \cdot (a_0, b_0) + j \cdot (a_1, b_1))$$

  for (i, j) in the (i, j) rectangle.
- We will sieve on both sides because we want both $|\mathrm{Res}(a - bx, f(x))|$ and $|\mathrm{Res}(a - bx, g(x))|$ to be smooth.

Sooner or later we will need to compare two things

- The accumulated contribution of sieved factor base primes p;
- and the size of the $|\mathrm{Res}(a - bx, f(x))|$ (on the side we're sieving on).

Norms: what is it about?

- The accumulated contribution of sieved factor base primes p;
- vs: the size of the $|\mathrm{Res}(a - bx, f(x))|$.

We do this comparison additively

- at the beginning of the computation, we store $\log|\mathrm{Res}(a - bx, f(x))|$ at the location corresponding to (i, j).
- each time we identify a location where p divides the norm, we subtract $\log p$ to that location.

We choose the log base so that the computation can be done using single-byte integer arithmetic.

This inaccuracy is an acceptable trade-off.

Norms: what is it about?

Each cell contains an 8-bit integer:

- Initialized with the logarithm of the norm.
- During sieving, the logarithm of the prime to be divided is subtracted.
- After sieving, positions where the cell contains a small integer are promising: only consider those for cofactorization.

After sieving, the cell corresponding to (i, j) contains an approximation of the log of the part of $\mathrm{Res}(a - bx, f(x))$ that is made of primes that were not sieved.

In NFS, depending on this log value, we may decide to:

- strive to factor them (e.g. with ECM);
- or ignore them because they stand too little chance of being interesting in the end.

Norm initialization in Cado-NFS (I)

When developing an NFS sieving program, it is important to be precise with norm computations at the beginning, because this is an important debugging asset.

Early versions of Cado-NFS:

- precise (but slowish) computation of norms;
- various tricks come into play;
- computation of algebraic norms more expensive;
- still claims a share of the total computation cost that is way too large (10%).

Norm initialization in Cado-NFS (II)

More recent improvements in the norm computation (version 2.1, improved in 2.3):

- Unify algebraic and rational norm initialization by computing piecewise linear approximations of polynomials that are accurate up to a multiplicative factor.
- Neighbouring cells often have the same value. We compute the value changes instead (easy for a linear polynomial).

Piecewise linear approximations

Input:

- a polynomial $\tilde{f}(x)$ with real coefficients. $\tilde{f}(x)$ is such that

$$\tilde{F}(i,j) = j^{\deg \tilde{f}} \tilde{f}(i/j) = \mathrm{Res}(a - bx, f(x)).$$

  The $\mathfrak{q}$-lattice defines a homography that yields $\tilde{F}$ from F:

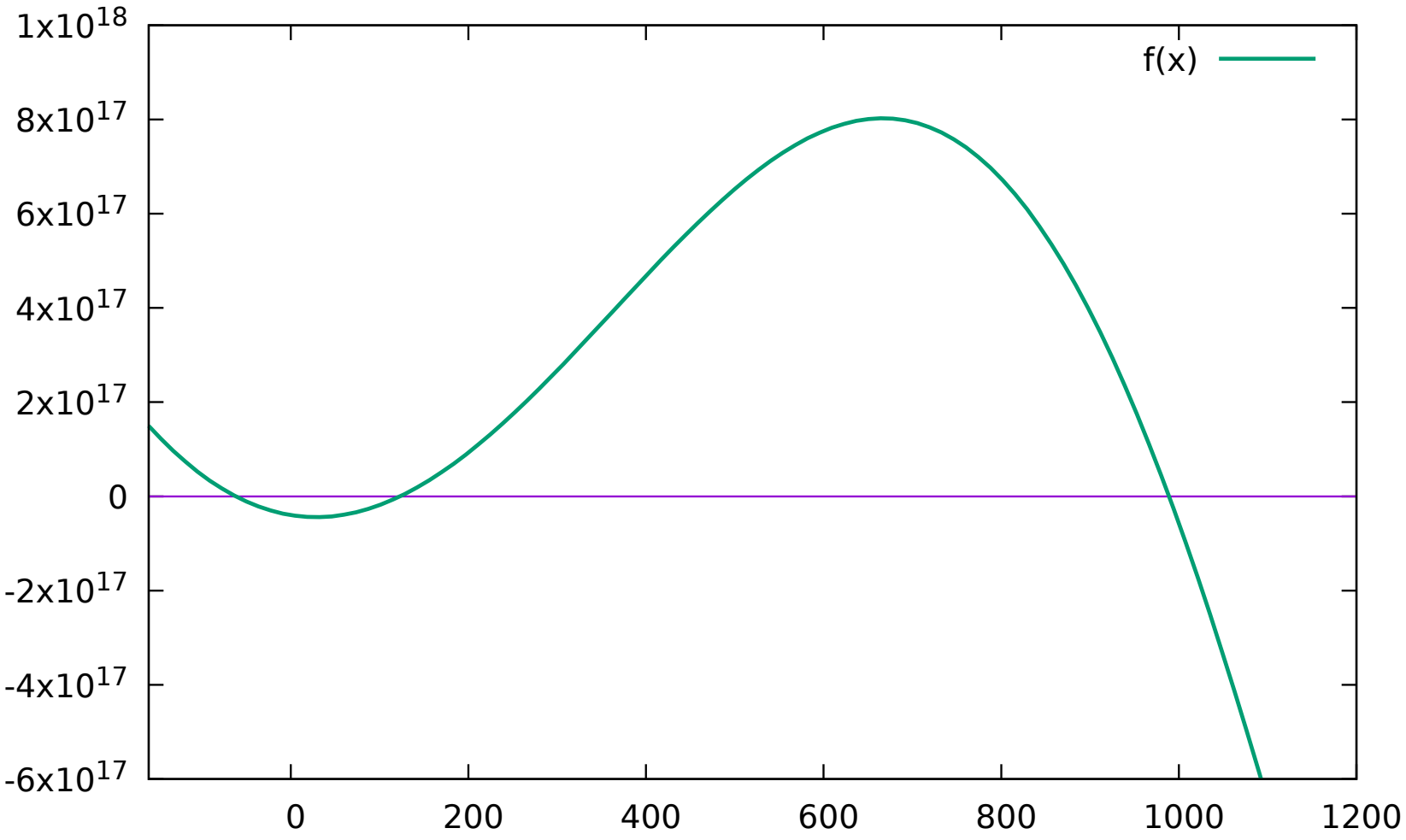
$$\tilde{F}(i,j) = F(ia_0 + ja_1, ib_0 + jb_1).$$

- an inaccuracy tolerance τ;
- a range of interest $[-2^{l-1}, 2^{l-1}] \times [1, J]$.

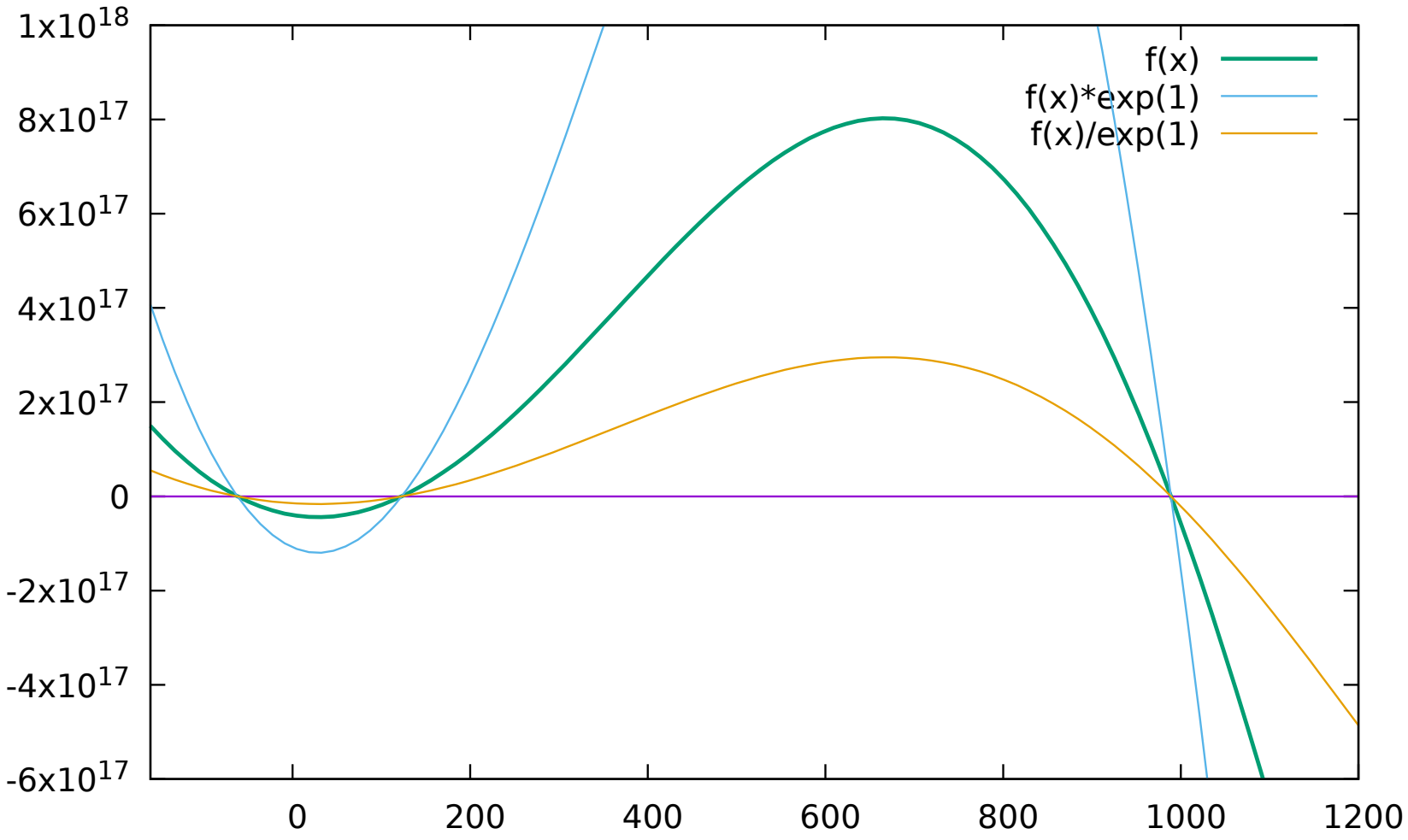
output: a list of linear functions $u_0, \ldots, u_{k-1}$ and consecutive intervals $R_0, \ldots, R_{k-1}$ such that $\cup_s R_s = [-2^{l-1}, 2^{l-1}]$ and

$$\forall s,\ \forall i \in R_s,\ e^{-\tau}|\tilde{F}(i,1)| \leq u_s(i) \leq e^{\tau}|\tilde{F}(i,1)|$$

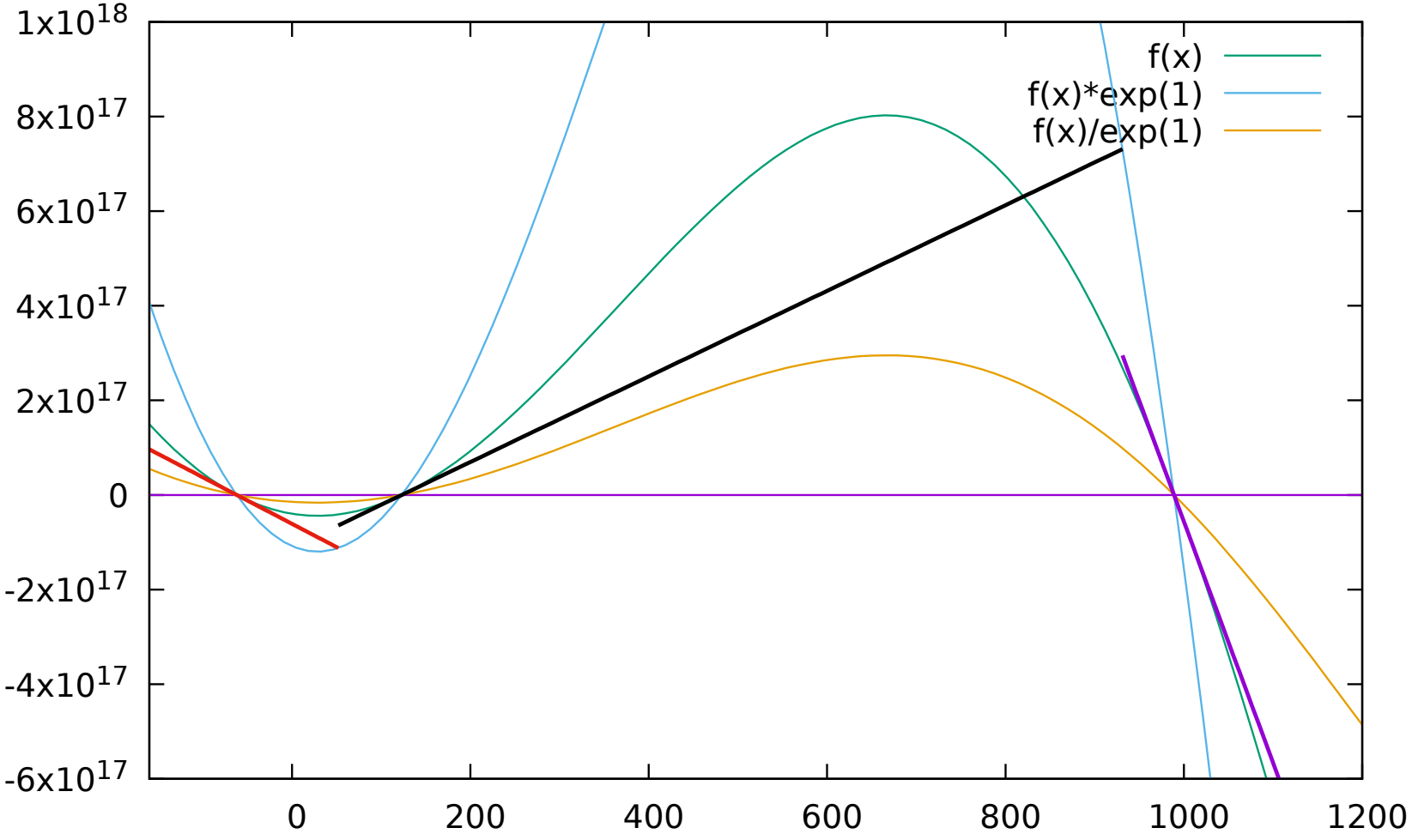
Piecewise linear approximations: example



Piecewise linear approximations: example



Piecewise linear approximations: example



Piecewise linear approximations

Once we have the PL approximations correct, then we can say that:

$$|\log|\tilde{F}(i,1)| - \log|\text{approximation}(i)|| \le \tau;$$
$$\left|\log|\tilde{F}(i,j)| - j^{\deg f}\log|\text{approximation}(i/j)|\right| \le \tau + (\deg f)\log j.$$

- it makes sense to compute PL approximations for more than just line $j = 1$.
- If we computed PL approximations for j_0, then the inaccuracy drops to $(\deg f)(\log j - \log j_0)$, so approximations on lines spaced in a geometric progression are fine.

(log)norm computation for linear polynomials

Input: $\tilde{u} = c_1 x + c_0$. Want to compute $\log_\beta |c_1 i + c_0 j|$ for some j, and for all i: $-2^{l-1} \leq i < 2^{l-1}$.

- compute real root $\zeta = -(c_0 j)/c_1$.
- set $i = i_{\min} = -2^{l-1}$.
- compute $y = \left\lfloor \log \tilde{U}(i,j) \right\rceil$.
- if $i < \zeta$, compute i' s.t. $\left\lfloor \log \tilde{U}(i',j) \right\rceil = y - 1$.
  (if $i > \zeta$, aim at $y + 1$ instead).
- fill the table with y until location i', and resume from there.

Finding i' is easy enough precisely because $\tilde{u}$ is a linear approximation.

Plan

Norm initialization

Adjusting the $\mathfrak{q}$-lattice

A foreword on J (integer bound)

Duh, some special-$\mathfrak{q}$'s are much faster than others:

```
# 3 relation(s) for side-1 (1310000579,873183740)
# Time for this special-q: 28.8720s [norm 0.0280+0.0200, [
    sieving 26.5920 (22.4760 + 1.0400 + 3.0760),          [
    factor 2.2320 (1.8160 + 0.4160)]
```

while for others:

```
# 19 relation(s) for side-1 (1310009947,283600118)
# Time for this special-q: 126.4920s [norm 0.1800+0.4480,
    sieving 111.3360 (86.7800 + 6.4040 + 18.1520),
    factor 14.5280 (11.7120 + 2.8160)]
```

Explanation comes from J.
(I'm not talking about the ideal J here!)

Lattice sieving

We sieve for $(a, b) = i \cdot \vec{u_0} + j \cdot \vec{u_1}$.

- $(\vec{u_0} = (a_0, b_0), \vec{u_1} = (a_1, b_1))$ basis of the lattice $\mathcal{L}_{\mathfrak{q}}$.
- We pick $\vec{u_0}$ the shortest vector in $\mathcal{L}_{\mathfrak{q}}$.
- We let typically $-I/2 \leq i < I/2$ and $0 \leq j < J$, have in mind $J = I/2$ too.
- We have $\det \mathcal{L}_{\mathfrak{q}} = q$.
  Furthermore, because of skewness, $a_i/b_i \approx s$.
- Bottom line: the vectors $\vec{u'_i} = (a_i\sqrt{\frac{1}{qs}}, b_i\sqrt{\frac{s}{q}})$ are relevant to plot.

Following slide: plot $[-I/2, I/2]\vec{u'_0} + [0, I/2]\vec{u'_1}$.

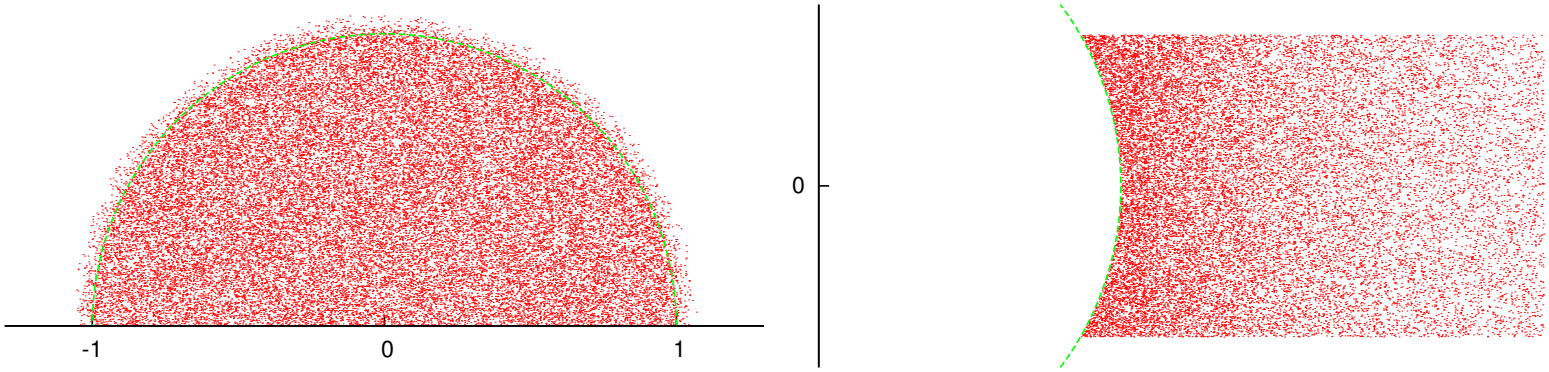
What happens?

It might really happen that the sieving rectangle looks pretty distorted.

If $\vec{u_0}$ is really really short, then $\vec{u_1}$ is somewhat bigger, since the lattice determinant is constrained.

As $\mathfrak{q}$ varies, one may wonder how the quotient of complex numbers $\frac{u_1}{u_0}$ evolves. There are theorems for that.

A familiar shape



(a): u_0

(b): ratio $\tau = u_1/u_0$
(turn your head clockwise)

It's even possible to write down the density around $u_1/u_0 = x + iy$, which is $\frac{3}{\pi y^2}\mathrm{d}x\mathrm{d}y$.

So, bad stuff happens, sometimes.

Historical provision in `las` against that

Some straightforward options.

- Discard $\mathfrak{q}$ when it so happens that u_1/u_0 has large imaginary part (that is, when $\vec{u_0}$ is exceptionally small).
- Other option: limit J to a smaller value in that case. `las` does exactly this.

The issue we observe is that J is sometimes reduced a LOT.

Better strategy for adjusting I, J

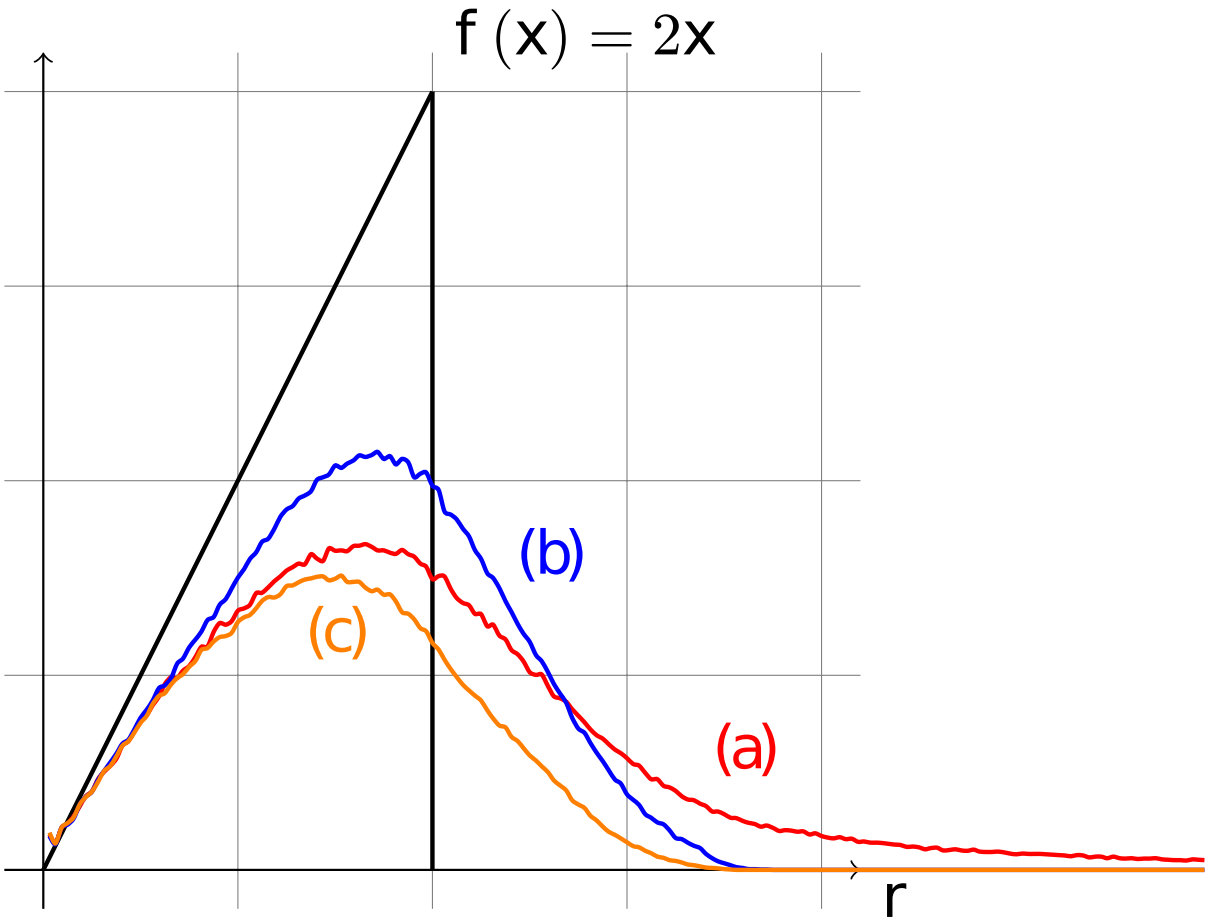
Assume we have in mind an area $A \approx 2^{31}$ within the (i, j) plane.

We sieve for $(a, b) = i \cdot \vec{u_0} + j \cdot \vec{u_1}$.

Obvious "we should rather do this" strategy.

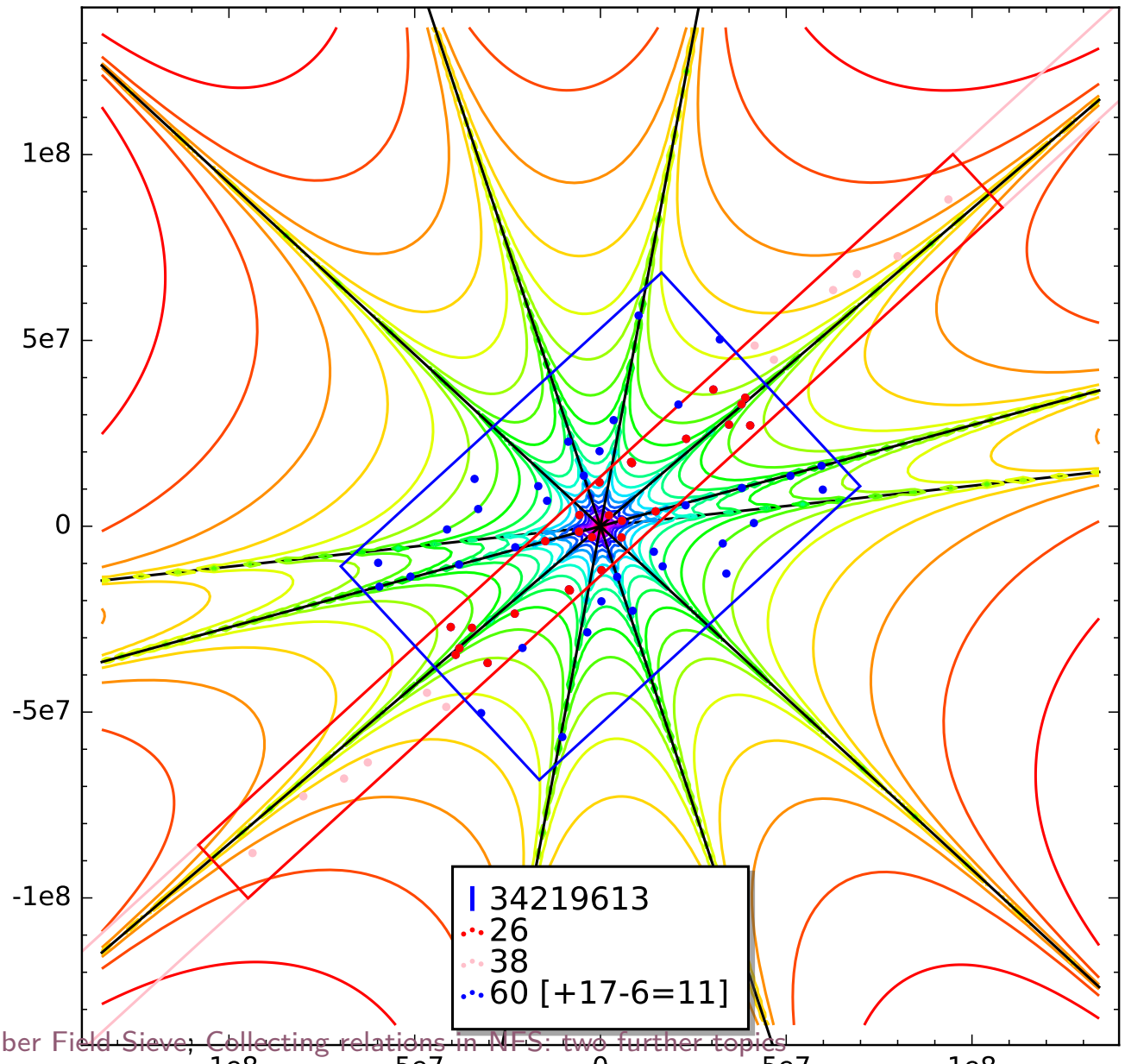
- Multiply the shortest vector by the largest interval.
- Ah, yes, but our I is limited to 2^{16}. (no longer in the development version — still at work!)
- One of the intervals, for sure, will be at least $\sqrt{A}$. Depending on u_1/u_0, we might prefer $\sqrt{A \cdot S}$ for some $S > 1$. The closest power of two is within $[\sqrt{AS/2}, \sqrt{2AS}]$.
- J, on the other hand, is not limited. So we should allow swapping vectors and reducing I instead for distorted lattices.

Relevant curve

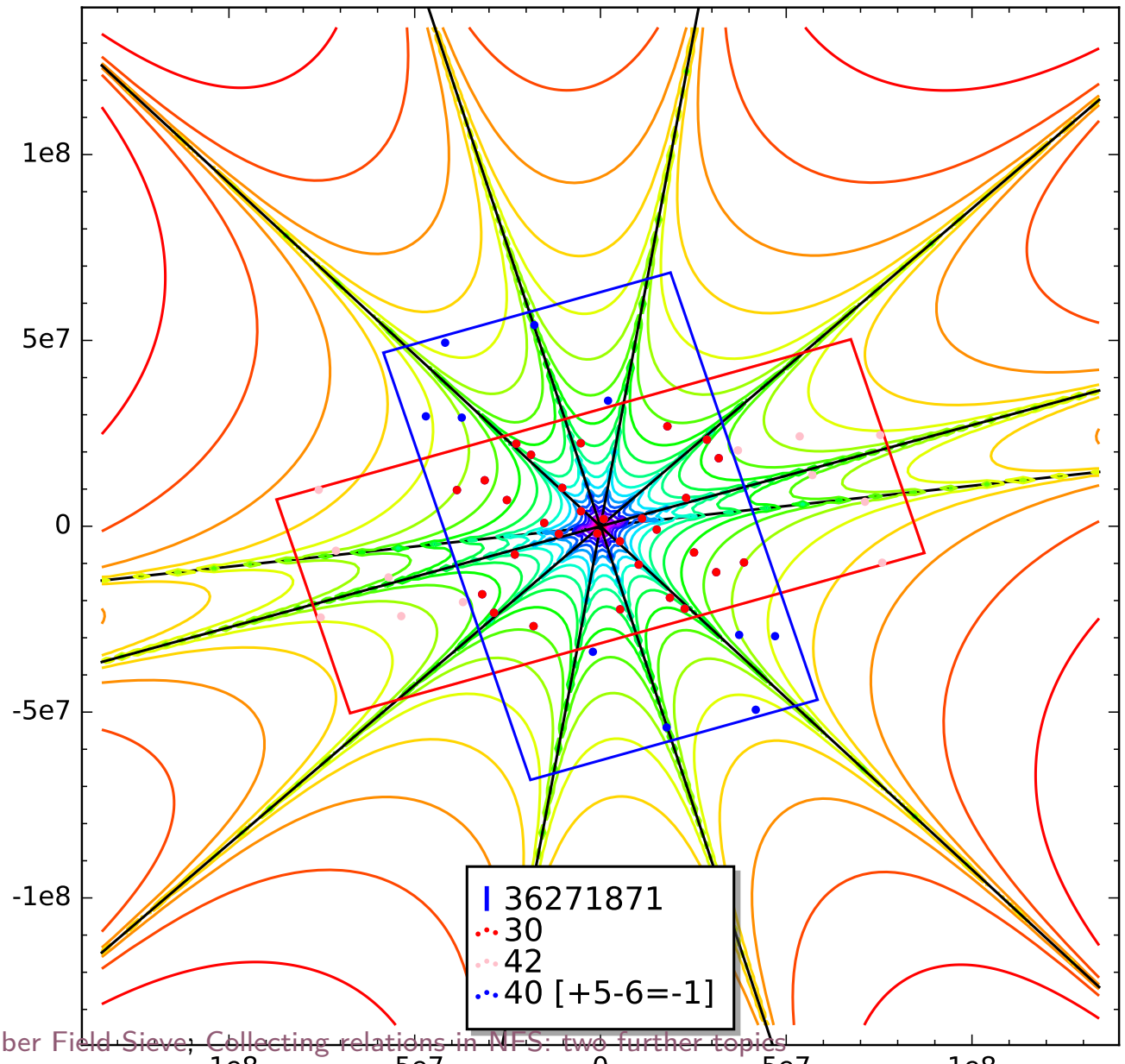


This is by no means a game changer, but will avoid wasting some special-$\mathfrak{q}$'s.

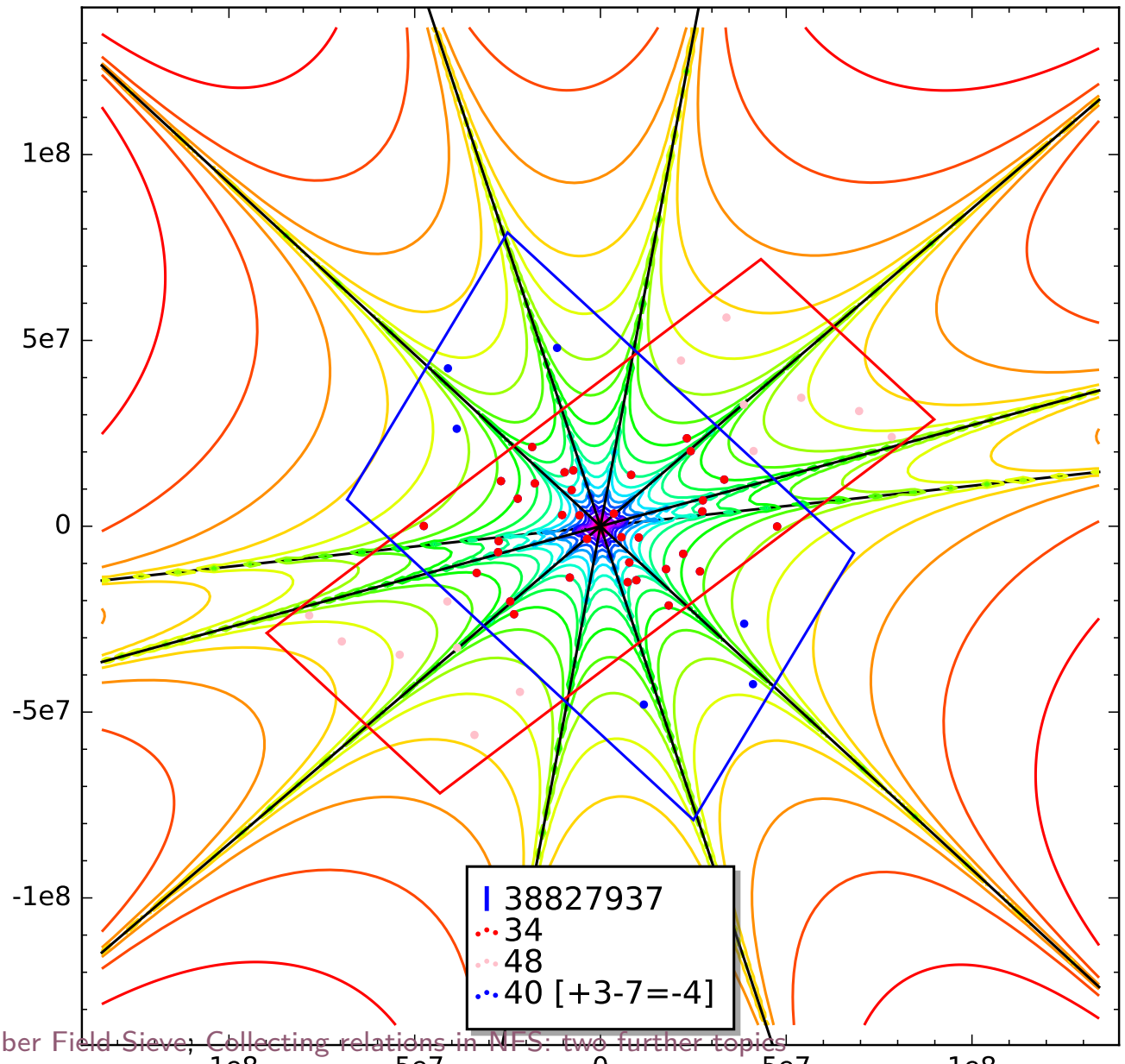
Some example plots



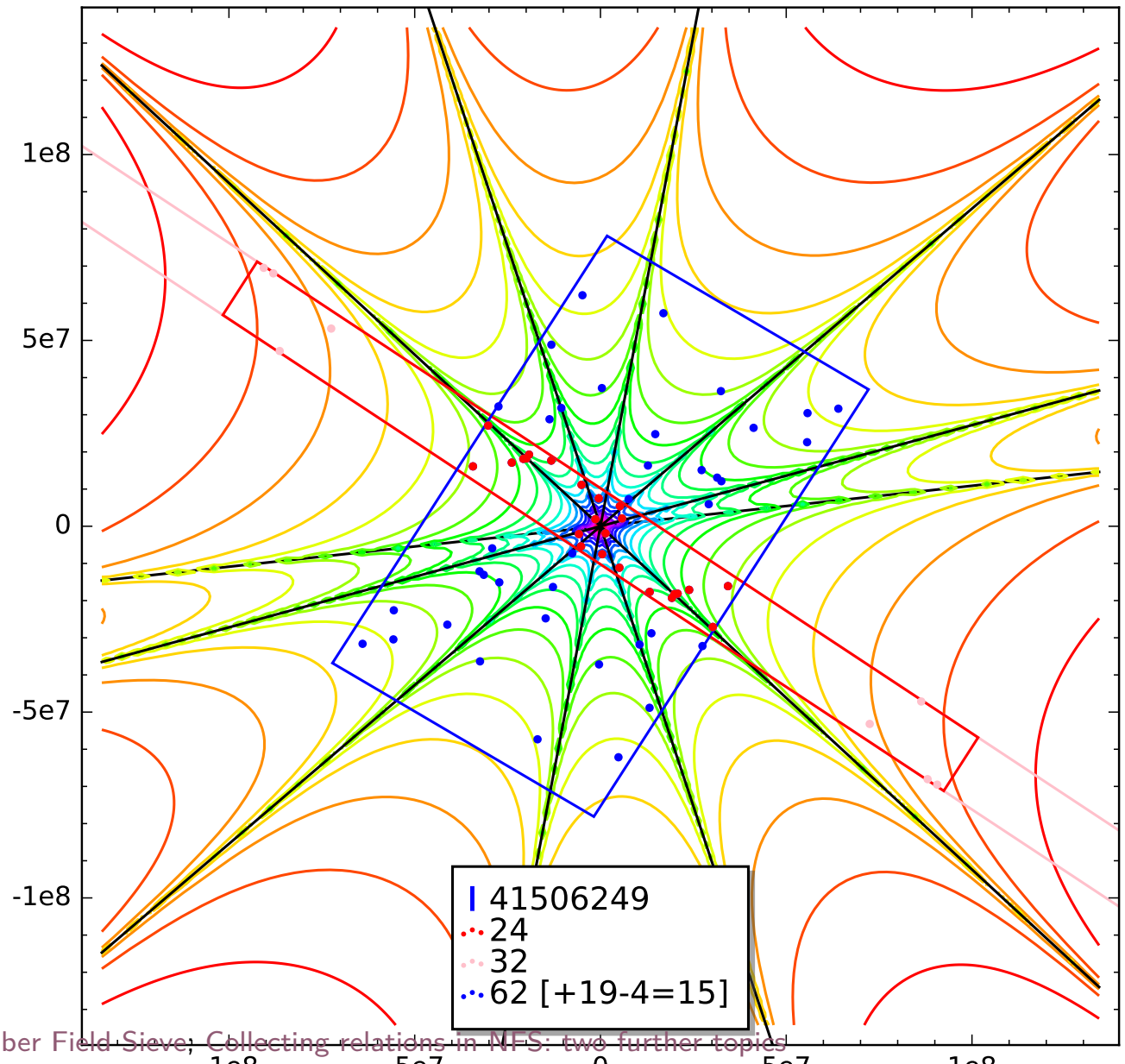
Some example plots



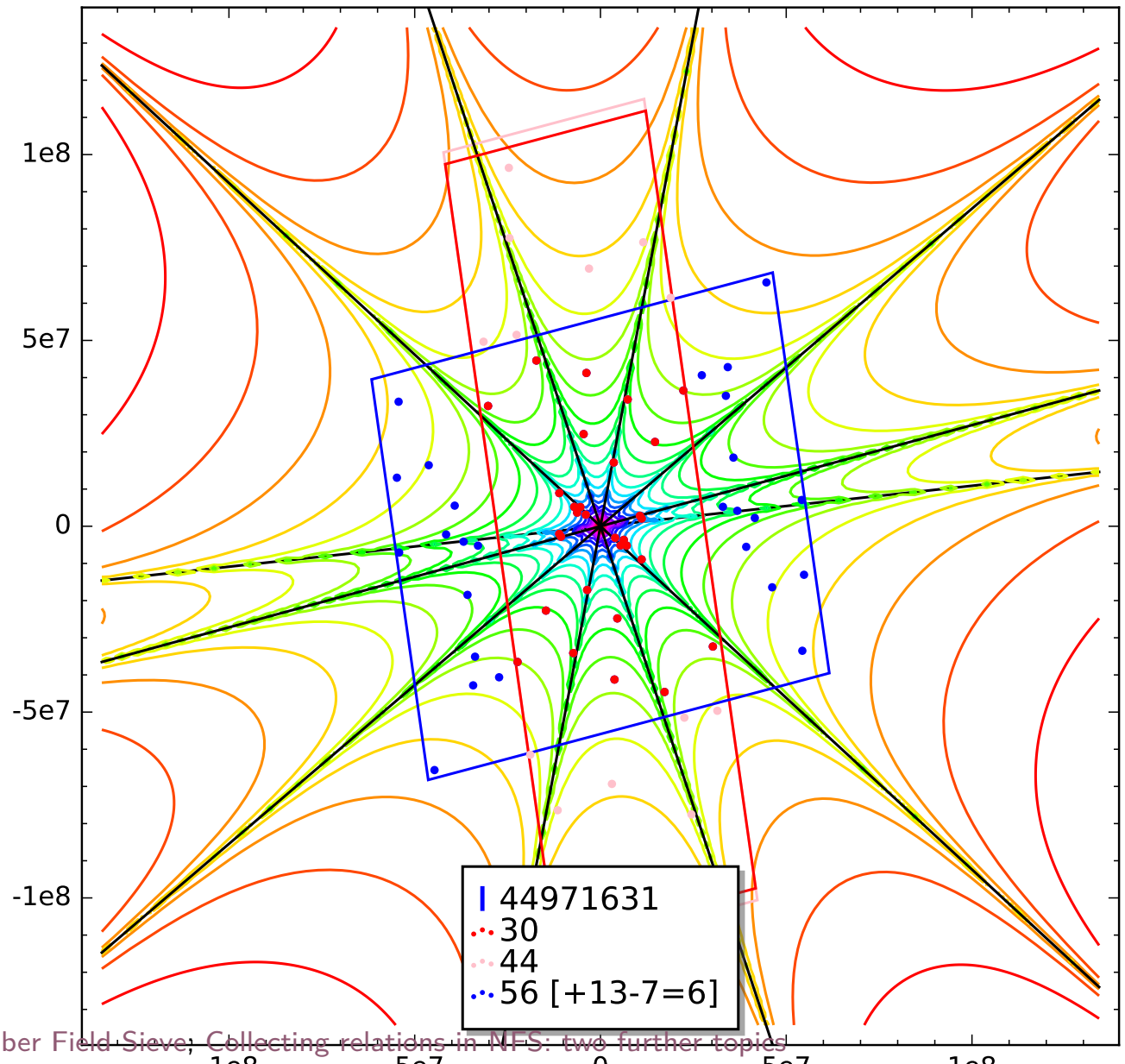
Some example plots



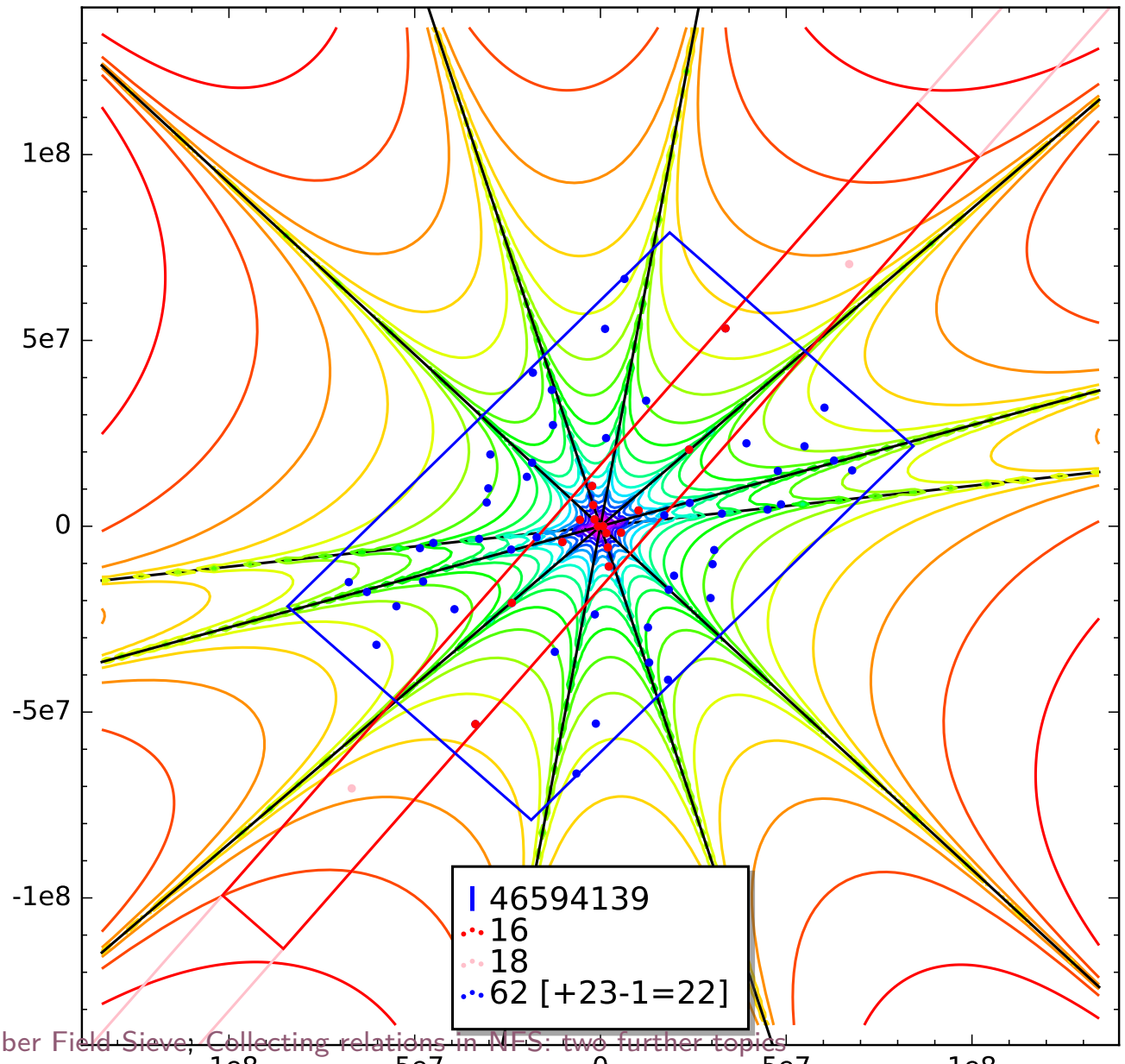
Some example plots



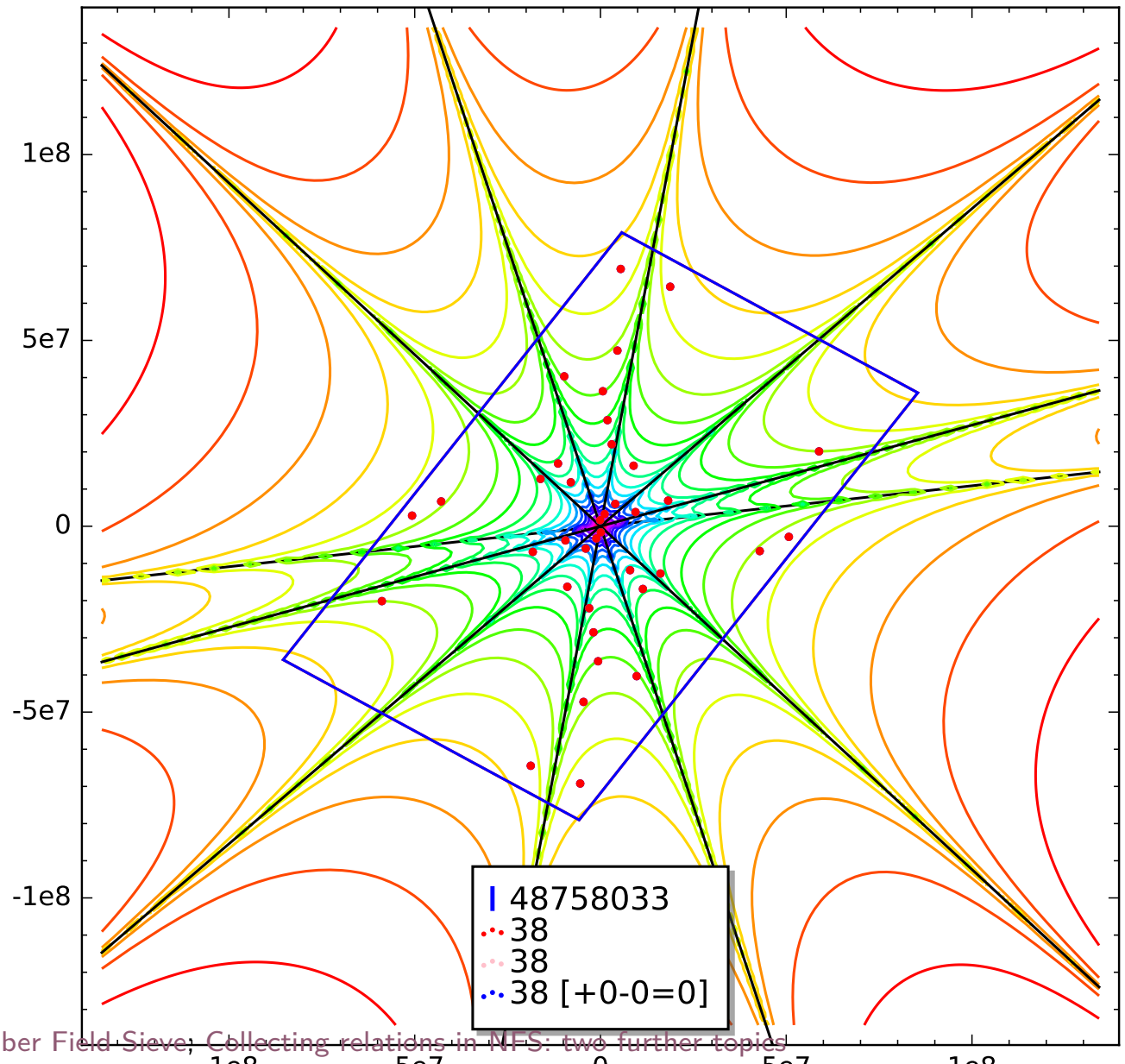
Some example plots



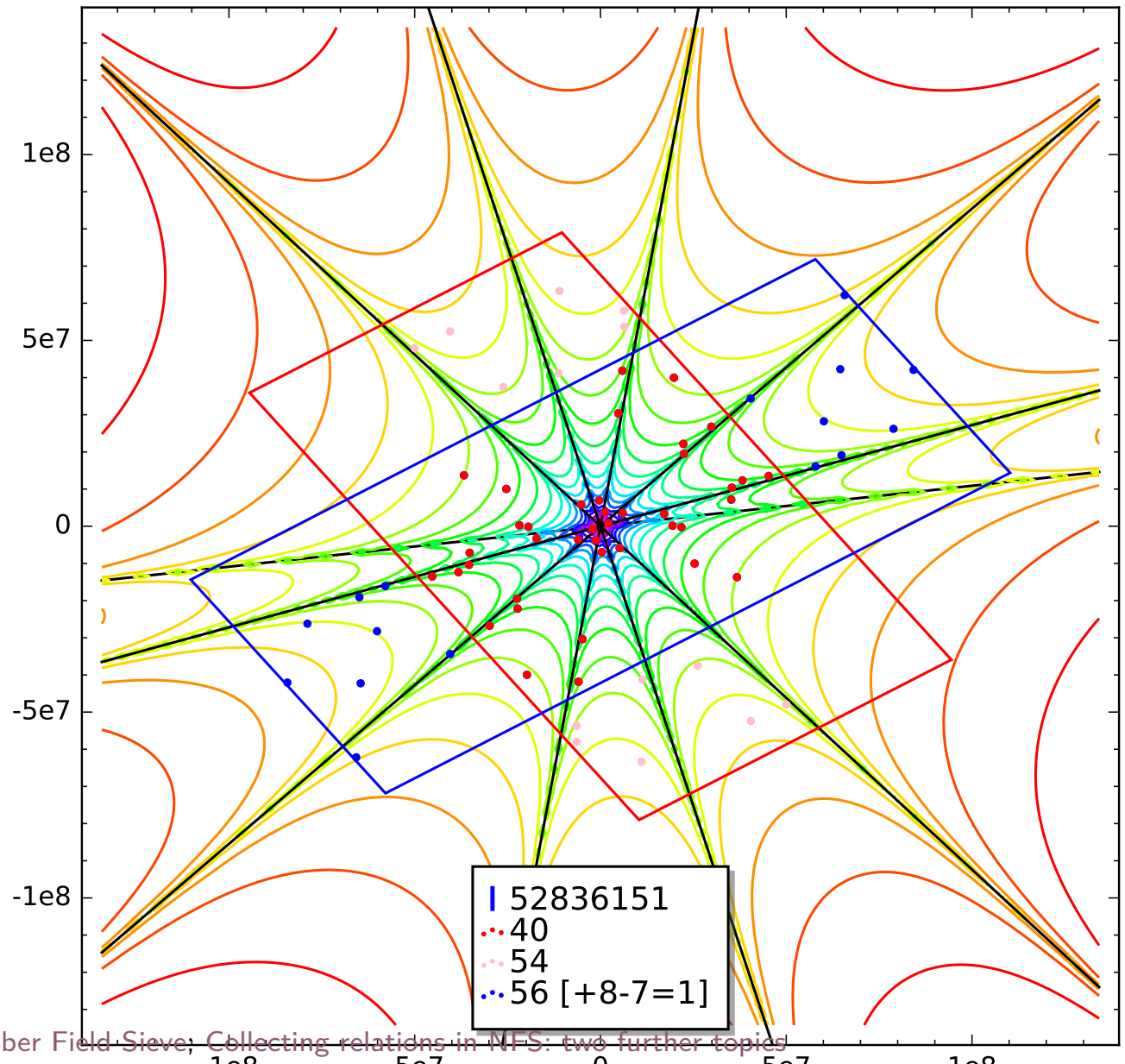
Some example plots



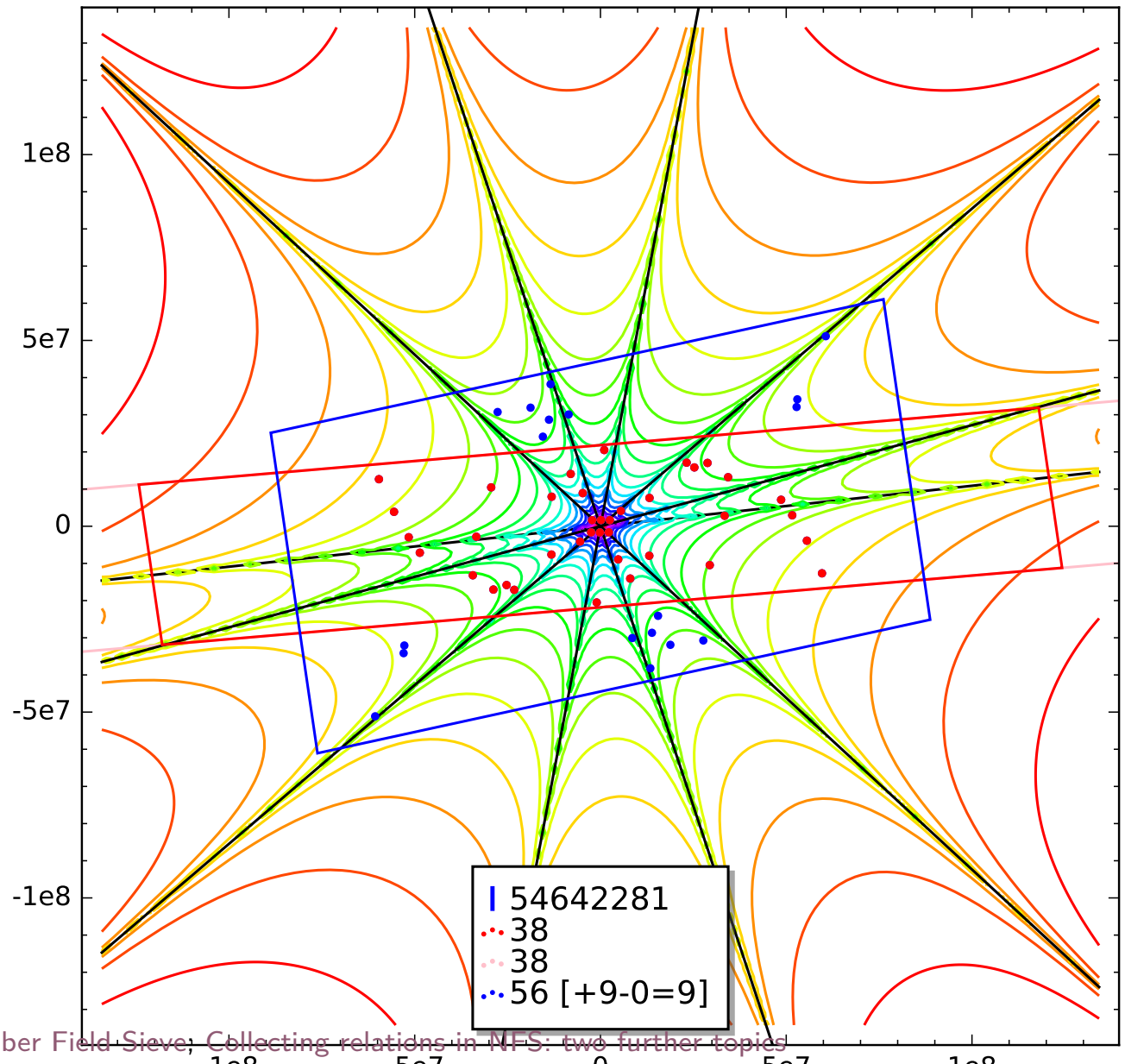
Some example plots



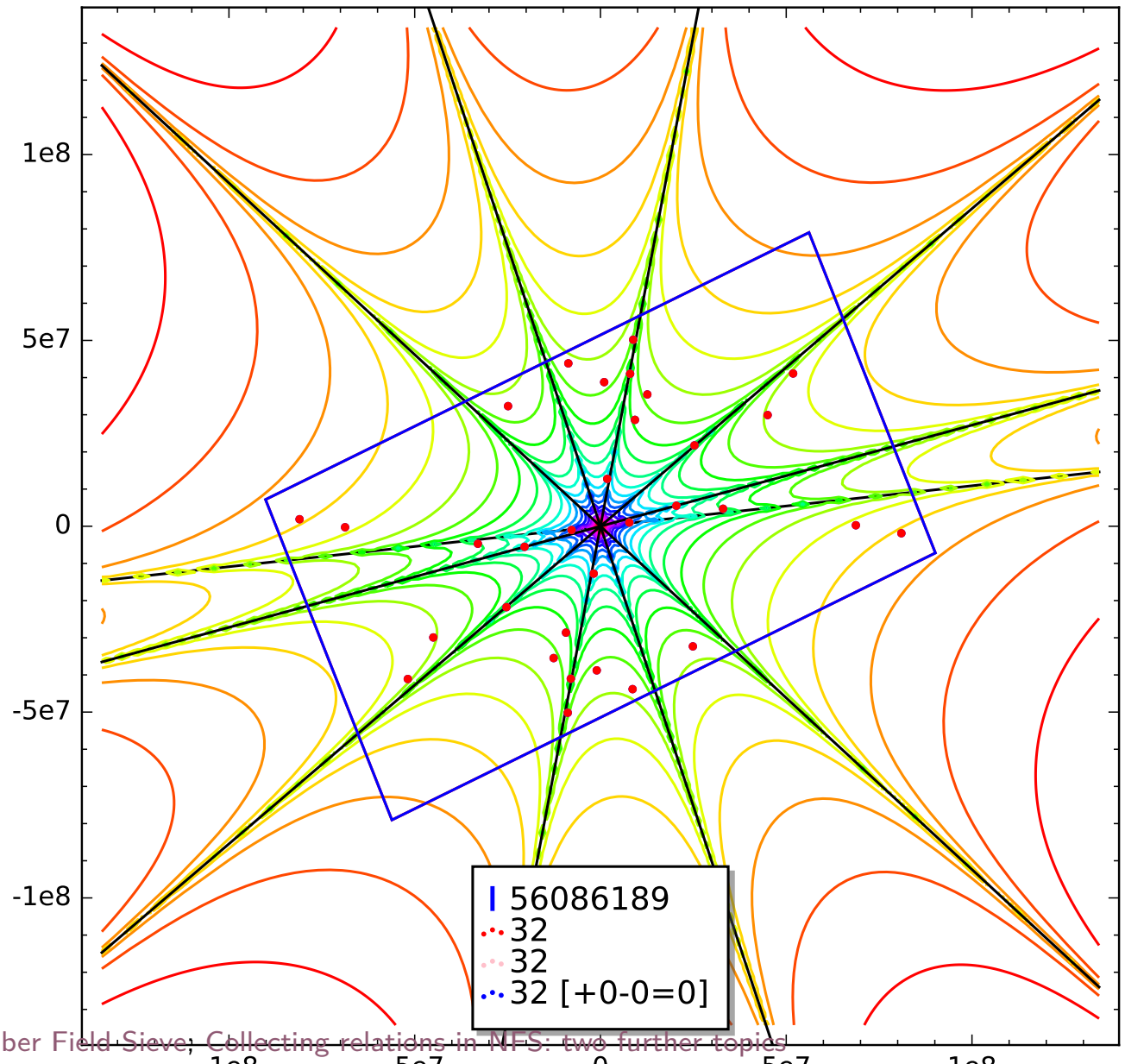
Some example plots



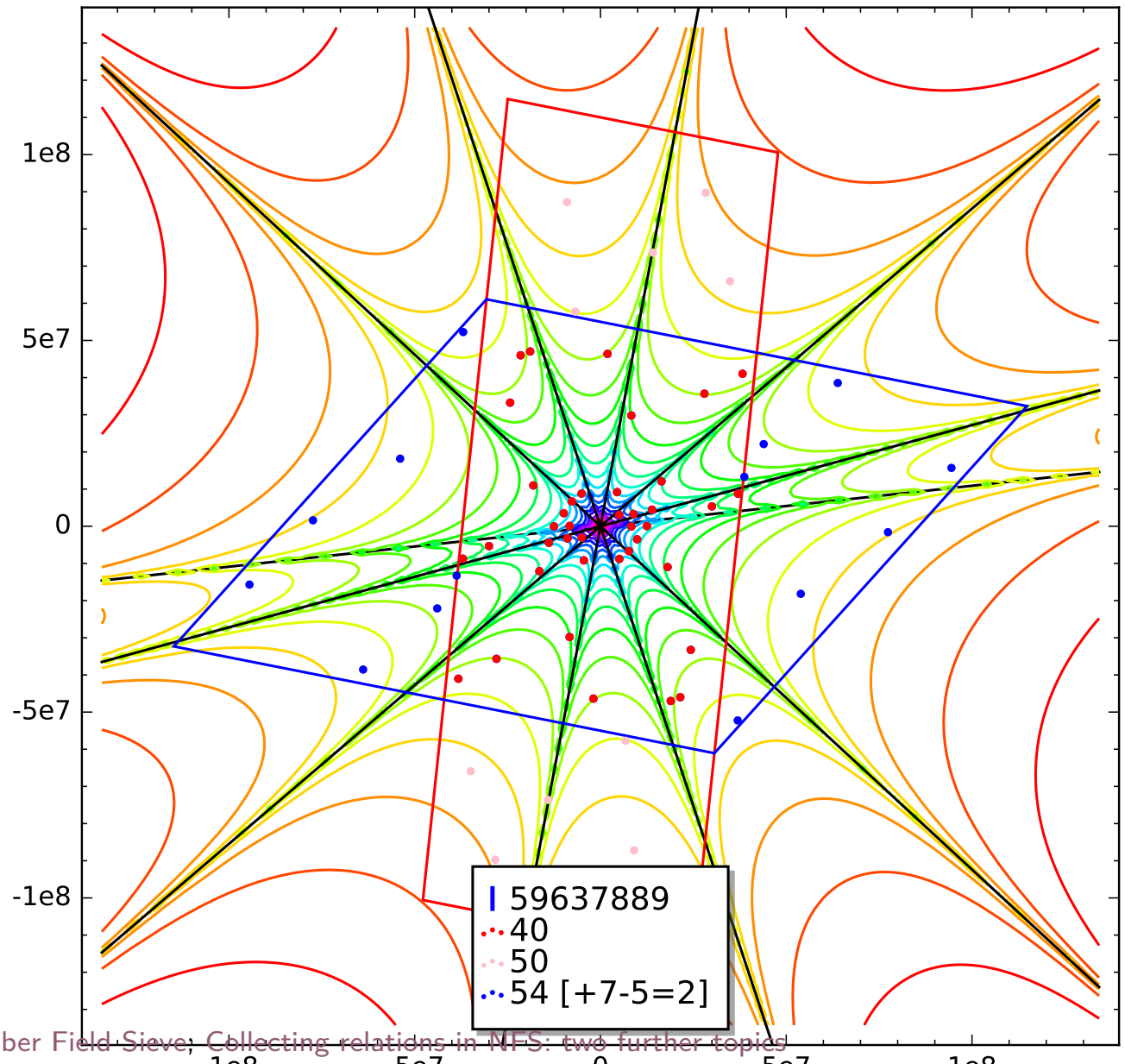
Some example plots



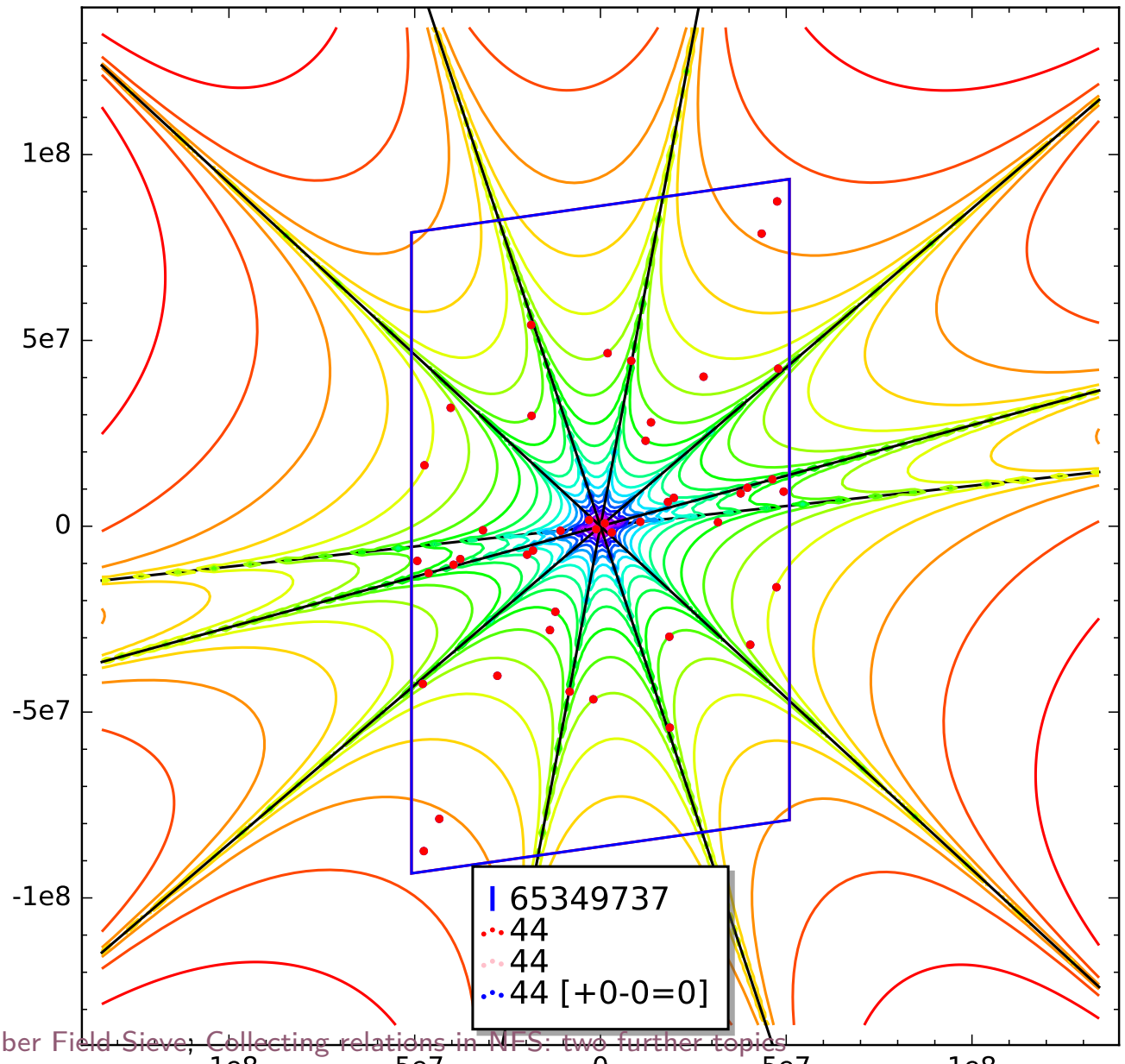
Some example plots



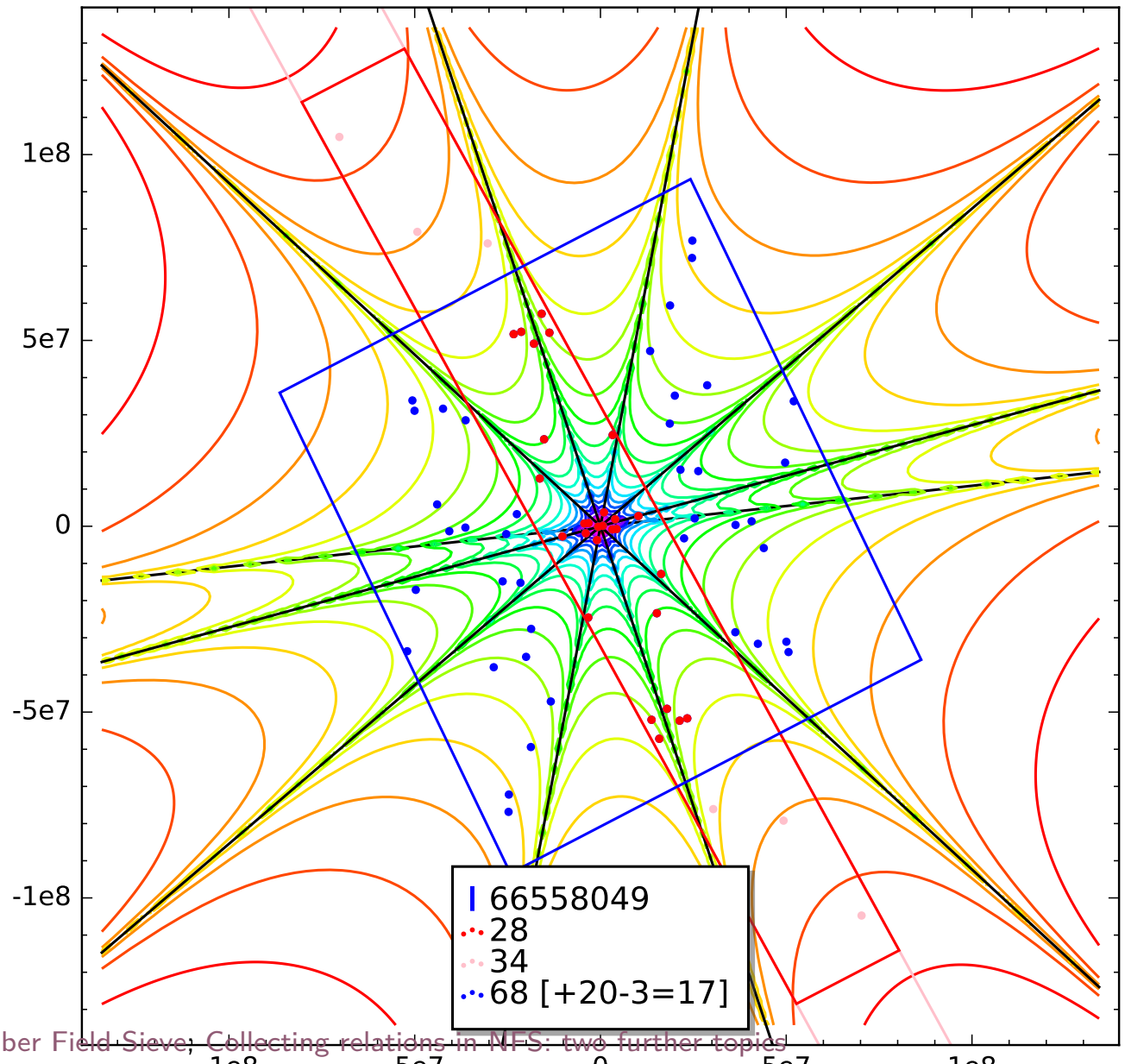
Some example plots



Some example plots



Some example plots



CSE291-14: The Number Field Sieve

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

Emmanuel Thomé

February 10, 2022

Part 6a

The filtering step

The filtering step

Duplicates

Singletons and “clique” removal

Merge

Practical aspects

Plan

The filtering step

Duplicates

Singletons and “clique” removal

Merge

Practical aspects

Plan

The filtering step

- Our input
- Our goal
- Terminology and main steps

Once we have collected all relations

The output of the relation collection can be viewed as a matrix

- a row corresponds to a relation;
- a column corresponds to a prime ideal;
- the coefficient is the valuation of the corresponding ideal in the corresponding relation.

Orders of magnitude

This is the dataset that comes directly from the relation collection.

year	what?	bits	dd	rows	columns
2010	RSA	768	232	64e9	35e9
2017	DLP	768	232	9.1e9	?
2019	RSA	795	240	8.9e9	2.4e9
2019	DLP	795	240	3.8e9	1.0e9
2020	RSA	829	250	8.7e9	6.5e9

Note: vast difference in the numbers comes from very different parameter choices.
The dataset weighs between hundreds of GB and several TB.

Sparsity

Another trait of the dataset.

Relations (rows) are VERY sparse.

Typical: 20 to 25 non-zero entries per row on average.

Rows and columns

There is no connection between rows and columns in this matrix.

We may sort rows and columns in any way we like, provided that we keep track of the permutation.

This is in contrast with other linear algebra problems for which the notion of "diagonal element" is meaningful. Here it is not.

Plan

The filtering step

- Our input
- Our goal
- Terminology and main steps

This is already linear algebra

We need to do some linear algebra

- In the factoring case, the matrix is defined over $\mathbb{F}_2$.
  We are looking for several left nullspace elements.
- In the discrete logarithm case (later!), the matrix is defined over $\mathbb{Z}/\ell\mathbb{Z}$ for some large prime number ℓ.
  We are looking for a right kernel vector.

The filtering step is the beginning of the linear algebra step.
(but they are often regarded as separate steps.)

Domain of the matrix coefficients

Factoring usage

Since we ultimately work modulo 2, all coefficients are either 0 or 1.

Discrete logarithm usage

Coefficients can be larger integers a priori, BUT the vast majority are ± 1. We can really assume that this is always the case.

Goal of the filtering step

The filtering step is a preprocessing step, which aims at reducing the matrix size.

Another appropriate term can be that filtering is actually creating some preconditioner.

Preprocessing operations

We want to apply a chain of elementary matrix transformations

$$M \to M'$$

such that a solution to the linear system $xM' = 0$ (or $M'x = 0$) leads to a solution of the linear system we started with.

Transformations

We can do the following things:

- Sort rows and columns.
- Remove duplicate rows.
- We will also see how, under certain conditions, we can:
  - Take out some rows or columns.
  - Replace rows by linear combinations (as is done in Gaussian elimination).

Old terminology

Until the late 1990s, the term structured Gaussian elimination was often used to refer to what is now known as filtering.

The filtering step

- The filtering step modifies the matrix built from the relations.
- It outputs a new matrix that is used as input by the linear algebra computation that comes next.
- The "quality" of the matrix produced by the filtering step impacts the time spent in the linear algebra step.

Quality metrics

- We want the new matrix to be much smaller
- We want to maintain sparsity.
- Ultimately we want to minimize the cost of the subsequent linear algebra step.

The filtering step is memory-bound and I/O-bound. CPU-wise, no serious computation is done during filtering.

Plan

The filtering step

- Our input
- Our goal
- Terminology and main steps

Definitions

Definition (Excess)

The excess of set of relations (resp. a matrix) is the difference between the number of relations (resp. the number of rows) and the number of prime ideals (resp. the number of columns).

Definitions

Definition (Weight)

- The weight of a column (resp. a row) in a matrix is the number of non-zero elements in this column (resp. this row).
- The weight of an ideal (resp. a relation) is the weight of the associated column (resp. row).
- The total weight of a matrix is the number of non-zero elements of this matrix.

Definitions

Definition (Density)

- The density of a column (resp. a row) in its weight divided by its size.
- The density of a matrix is its average row density.

Inside the filtering step

The filtering step is split in 4 stages:

- duplicate removal: very easy, uses hash tables to remove relations that appear more than once;
- singleton removal: remove useless rows and columns;
- "clique" removal: use the excess to reduce the size of the matrix;
- merge: beginning of a Gaussian elimination.

Plan

The filtering step

Duplicates

Singletons and “clique” removal

Merge

Practical aspects

Plan

Duplicates

- Analysis of the cause
- Dealing with it

Why do we have duplicates?

Duplicate relations are completely identical rows.

This can only happen if the corresponding pairs (a, b) and (a', b') are such that:

- $(a - b\alpha) = (a' - b'\alpha) \times$ a unit in $\mathcal{O}_K$.
- and the same thing on the other side, which probably means $a = \pm a'$ and $b = \pm b'$ (if we have a rational side).

In most configurations, duplicate relations mean identical (a, b)'s.

Can we really have identical (a, b)'s?

If we do only line sieving: we can't.

- By design, we only process an (a, b) pair once.
- For a large scale computation, accidentally sieving the same sub-range twice may happen, though.

For special-$\mathfrak{q}$ sieving, the situation is a bit different, since a given (a, b) pair may belong to two distinct lattices $\mathcal{L}_{\mathfrak{q}}$ and $\mathcal{L}_{\mathfrak{q}'}$.

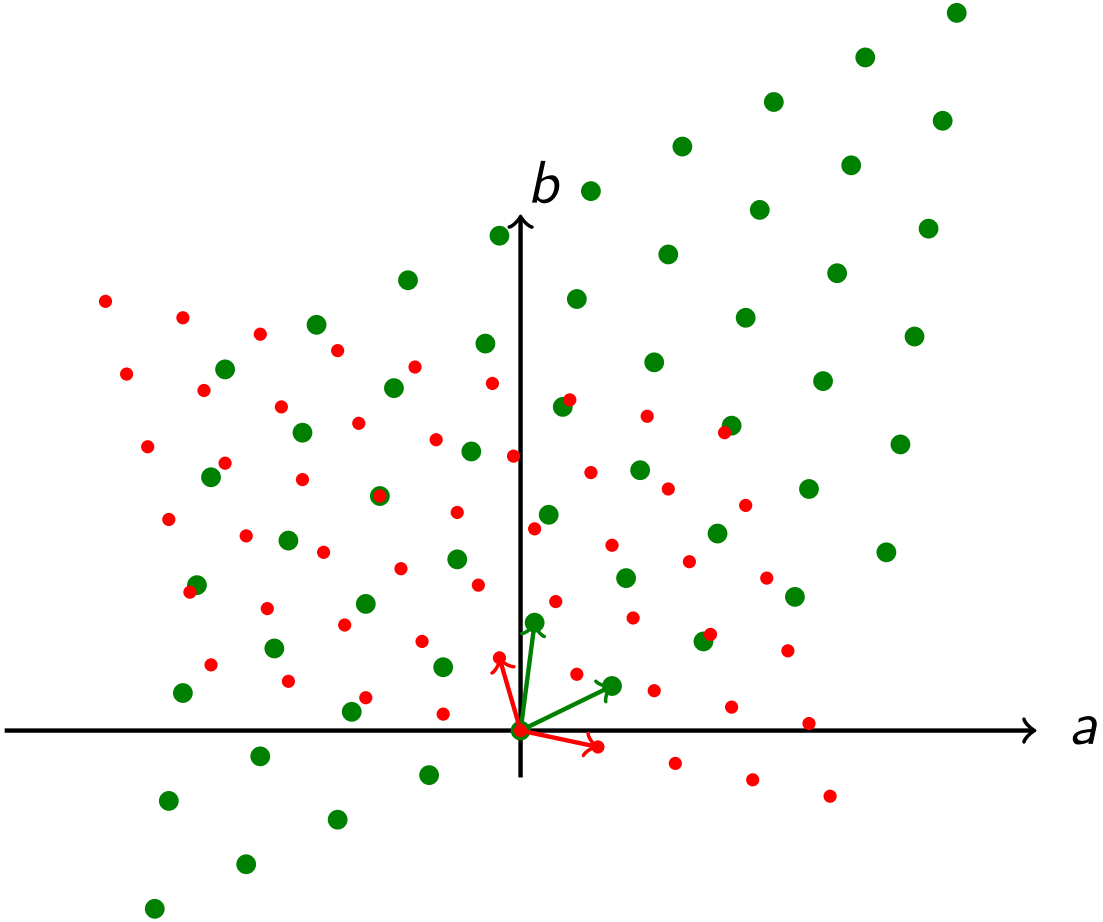
Points in two different lattices

If (a, b) is in two different lattices $\mathcal{L}_{\mathfrak{q}}$ and $\mathcal{L}_{\mathfrak{q}'}$, then the factorization of $\langle a - b\alpha \rangle \times J$ involves both $\mathfrak{q}$ and $\mathfrak{q}'$.

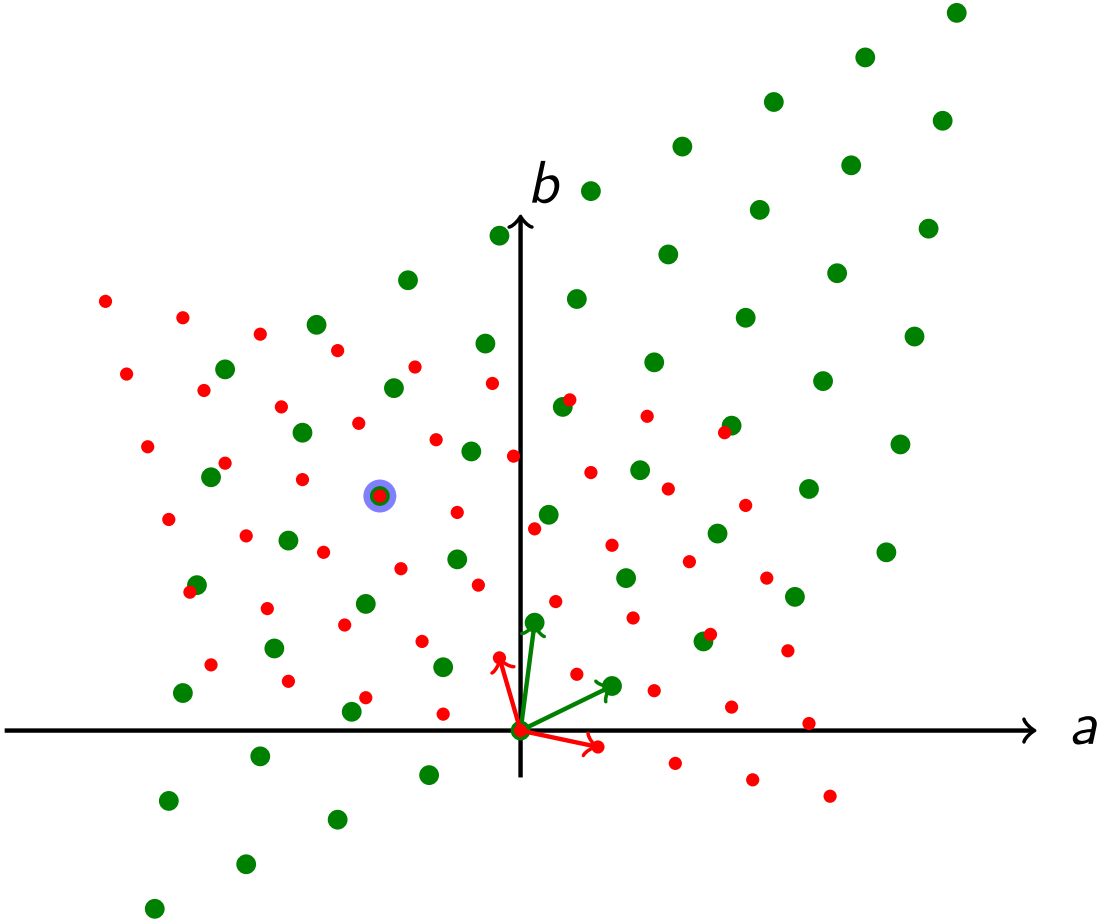
An approach that does not work to avoid duplicates:

> If two (or more) prime ideals from the special-$\mathfrak{q}$ range appear in the factorization of $\langle a - b\alpha \rangle \times J$, keep the relation only if the current special-$\mathfrak{q}$ is the largest of the two (or more).

Intersections of two lattices

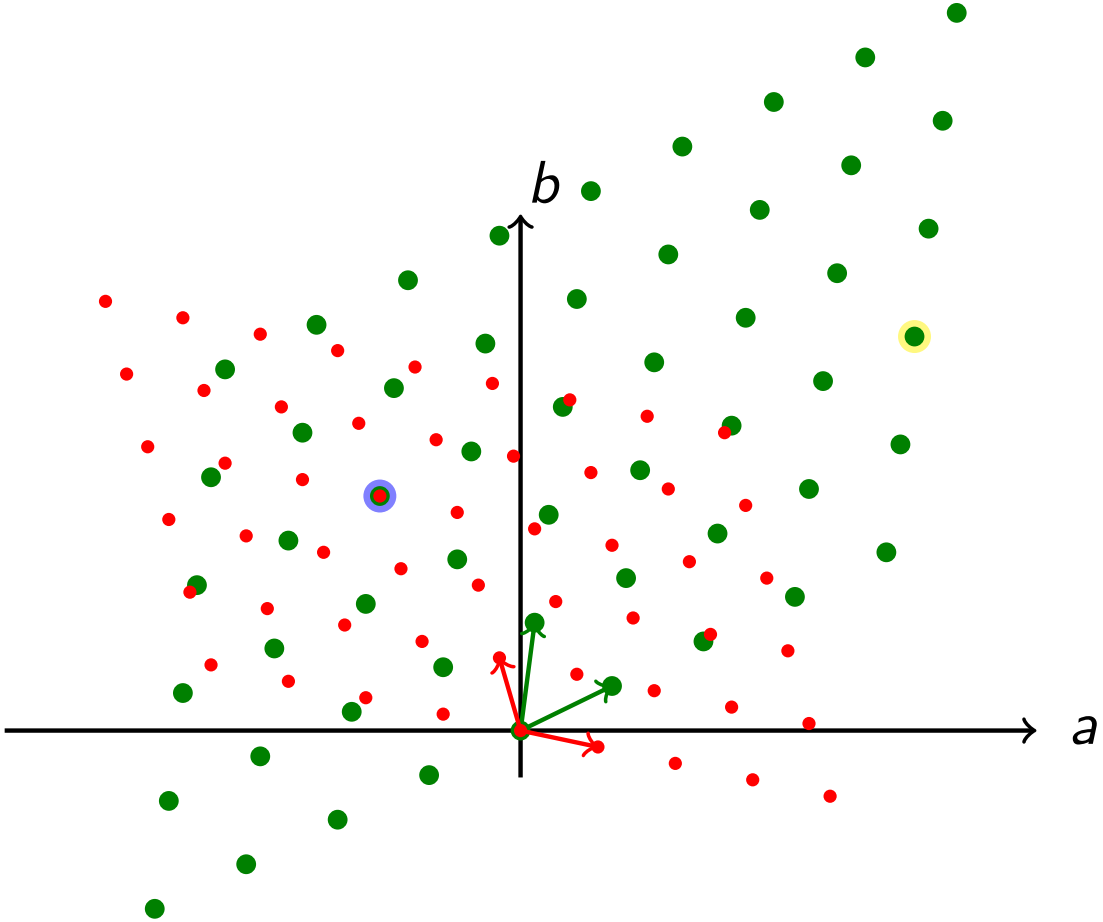


Intersections of two lattices



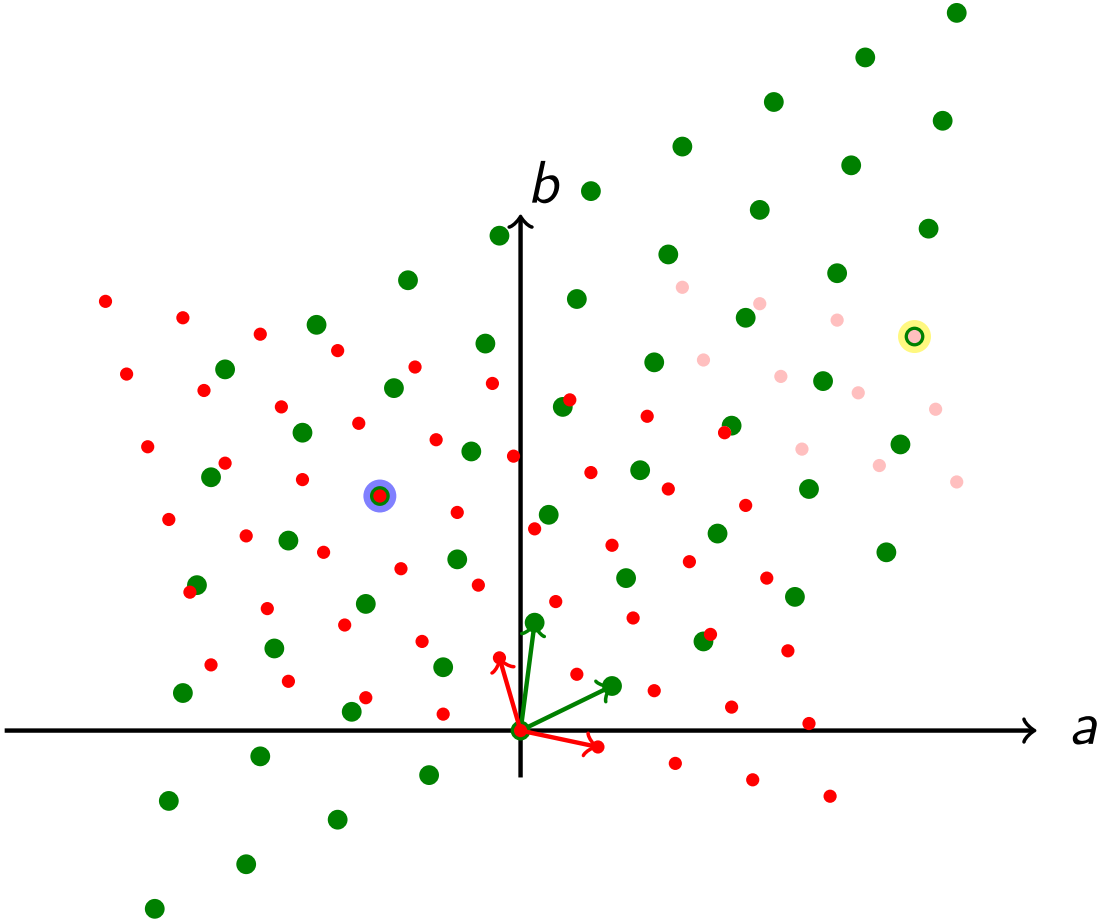
Point ● is in $\mathcal{L}_{\mathfrak{q}} \cap \mathcal{L}_{\mathfrak{q}'}$ and is reached twice.

Intersections of two lattices



Point ● is in $\mathcal{L}_{\mathfrak{q}} \cap \mathcal{L}_{\mathfrak{q}'}$ and is reached twice.
Point ● is in $\mathcal{L}_{\mathfrak{q}} \cap \mathcal{L}_{\mathfrak{q}'}$ and is reached only once.

Intersections of two lattices



Point ● is in $\mathcal{L}_{\mathfrak{q}} \cap \mathcal{L}_{\mathfrak{q}'}$ and is reached twice.
Point ● is in $\mathcal{L}_{\mathfrak{q}} \cap \mathcal{L}_{\mathfrak{q}'}$ and is reached only once.

Avoiding duplicates is not trivial

On-the-fly duplicate removal

Trying to avoid duplicates altogether with special-$\mathfrak{q}$ sieving is a more subtle task than one may think.

It is possible to do it, though, with moderate overhead cost. Cado-NFS has this functionality.

Some factors affect the number of duplicates, such as the size of the special-$\mathfrak{q}$ range.

Good rule of thumb: keep $q_{\max}/q_{\min}$ under control (say 2).

Plan

Duplicates

- Analysis of the cause
- Dealing with it

Dealing with duplicates

Good news: dealing with duplicates is very easy.

- Only (a, b) matters.
- We can use a hash table. But a hash table with several billion entries could require some thought.

Cado-NFS uses a few very basic techniques:

- Hash (a, b),
- split the input according to $H(a, b) \bmod K$ (on disk),
- de-duplicate each subset.

Very simple-minded and strongly I/O-bound, but does the job.

Plan

The filtering step

Duplicates

Singletons and “clique” removal

Merge

Practical aspects

Singleton removal

- A singleton is a column of weight 1.
- The removal of a singleton is the removal of the column and of the row corresponding to this single non-zero coefficient.
- Example:

$$\begin{pmatrix} 0 & 1 & 1 & 0 & 1 & 1 \\ 1 & 1 & 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix}$$

- Removing a singleton reduces the total weight of the matrix but cannot reduce the excess.

Singleton removal

- A singleton is a column of weight 1.
- The removal of a singleton is the removal of the column and of the row corresponding to this single non-zero coefficient.
- Example:

$$\begin{pmatrix} 0 & 1 & 1 & 0 & 1 & 1 \\ 1 & 1 & 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix}$$

- Removing a singleton reduces the total weight of the matrix but cannot reduce the excess.

Singleton removal

- A singleton is a column of weight 1.
- The removal of a singleton is the removal of the column and of the row corresponding to this single non-zero coefficient.
- Example:

$$\left(\begin{array}{c|ccccc}
0 & 1 & 1 & 0 & 1 & 1 \\
\hline
1 & 1 & 0 & 1 & 0 & 1 \\
\hline
0 & 0 & 1 & 1 & 0 & 0 \\
0 & 0 & 1 & 0 & 0 & 1 \\
0 & 0 & 0 & 1 & 0 & 1 \\
0 & 0 & 0 & 0 & 1 & 0 \\
0 & 0 & 0 & 0 & 1 & 1 \\
0 & 0 & 0 & 0 & 0 & 1
\end{array}\right)$$

- Removing a singleton reduces the total weight of the matrix but cannot reduce the excess.

Singleton removal

- A singleton is a column of weight 1.
- The removal of a singleton is the removal of the column and of the row corresponding to this single non-zero coefficient.
- Example:

$$\begin{pmatrix} 0 & 1 & 1 & 0 & 1 & 1 \\ \cancel{1} & \cancel{1} & \cancel{0} & \cancel{1} & \cancel{0} & \cancel{1} \\ 0 & 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix}$$

- Removing a singleton reduces the total weight of the matrix but cannot reduce the excess.

Singleton removal

- A singleton is a column of weight 1.
- The removal of a singleton is the removal of the column and of the row corresponding to this single non-zero coefficient.
- Example:

$$\begin{pmatrix} \cancel{0} & \cancel{1} & \cancel{1} & \cancel{0} & \cancel{1} & \cancel{1} \\ \cancel{1} & \cancel{1} & \cancel{0} & \cancel{1} & \cancel{0} & \cancel{1} \\ \cancel{0} & \cancel{0} & 1 & 1 & 0 & 0 \\ \cancel{0} & \cancel{0} & 1 & 0 & 0 & 1 \\ \cancel{0} & \cancel{0} & 0 & 1 & 0 & 1 \\ \cancel{0} & \cancel{0} & 0 & 0 & 1 & 0 \\ \cancel{0} & \cancel{0} & 0 & 0 & 1 & 1 \\ \cancel{0} & \cancel{0} & 0 & 0 & 0 & 1 \end{pmatrix}$$

- Removing a singleton reduces the total weight of the matrix but cannot reduce the excess.

Singleton removal

Why can we remove singletons?

Any solution to $xM' = 0$ or $M'x = 0$ immediately leads to a solution for the original matrix M.

Note: this "leads to" part may require us to memorize the deleted row. (This is only for DLP.)

Singleton removal

- Only need to know if a coefficient is non-zero or not, not the actual value.
- For each row, we store the indices of the columns with non-zero coefficients (but not the value of the coefficients).
- We also store and maintain the weight of each column.
- Removing a singleton can create other singletons, it may be necessary to loop through the matrix more than once.

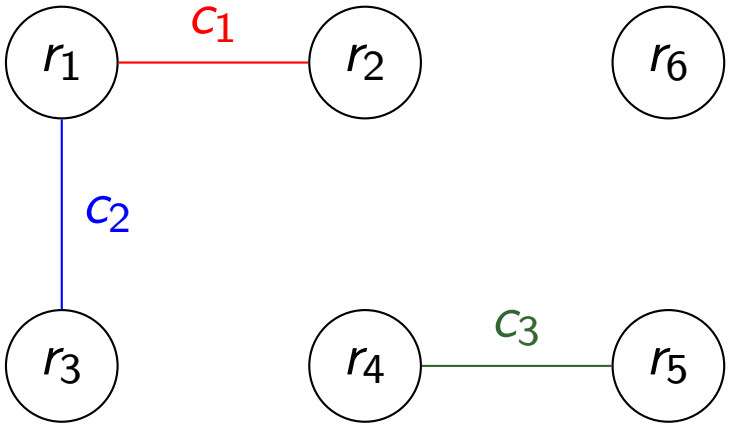
"Clique" removal

- While the excess is larger that what is needed, it is possible to remove some rows. (we may need to memorize them, though.)
- We can choose which row is deleted, how do we choose?
- Put otherwise: what is the best possible use of the excess that we have?

"Clique" removal

- Remark: if a row containing a column of weight 2 is removed, this column becomes a singleton and can be removed.
- A "clique" is a connected component of the graph where the nodes are the rows and the edges are the columns of weight 2.
- It has NOTHING TO DO with cliques in graph theory.
- If any row in a "clique" is removed, the creation of singletons leads to the removal of all rows of the "cliques".
- Example:

$$\begin{pmatrix} 1 & 1 & 0 & 0 \\ 1 & 0 & 0 & 1 \\ 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$



"Clique" removal

- When removing a "clique", one more row than column is removed, so the excess is reduced by 1. And the total weight of the matrix is reduced.
- The "clique" removal algorithm is quite simple:

  while the excess allows it, remove a "clique".
- How do we choose the "clique" to remove?
  Multiple cost functions have been explored over the years.
  Current wisdom is mainly based on empirical evidence.
- In order to compute and remove "cliques", we only need to know if a coefficient is non-zero or not, not the actual value.

The purge binary in Cado-NFS

- In Cado-NFS, the singleton and "clique" removal are done by the purge binary.
- Algorithm:
  - Input: the relations and a target excess.
  - Output: the remaining relations (and separately, the deleted relations if needed).
  - 1. Remove all singletons.
  - 2. Remove some "cliques".
  - 3. Go to 1 if the excess is larger than the target value.
- The purge binary is the same for factorization and discrete logarithm computations.

Plan

The filtering step

Duplicates

Singletons and “clique” removal

Merge

Practical aspects

Merge

- Merge is the beginning of a Gaussian elimination: combinations of rows are performed to create singletons that are then removed.

Definition (k-merge)

Let $k \geq 2$ be an integer. Let C be a column of weight k and $r_1, \ldots, r_k$ the k rows corresponding to the k non-zero coefficients of C. A k-merge is a way to perform successive rows additions of the form $r_i \leftarrow \alpha_{ij} r_i + \beta_{ij} r_j$, with $i \neq j$ and $1 \leq i, j \leq k$, such that the column C becomes a singleton, which will be deleted.

- For factorization, $\alpha_{ij} = \beta_{ij} = 1$; for discrete logarithm, they are usually very small (almost always ± 1).
- Merge does not change the right kernel (if we memorize the removed rows) nor the left kernel.

Merge – example

Example of a 2-merge and a 3-merge in the case of factorization.

$$\begin{pmatrix} 1 & 1 & 0 & 1 & 0 \\ 1 & 0 & 1 & 0 & 1 \\ 1 & 1 & 1 & 0 & 0 \\ 0 & 1 & 1 & 1 & 0 \\ 1 & 1 & 0 & 1 & 1 \\ 1 & 1 & 1 & 1 & 0 \end{pmatrix}$$

Merge – example

Example of a 2-merge and a 3-merge in the case of factorization.

$$\begin{pmatrix} 1 & 1 & 0 & 1 & 0 \\ 1 & 0 & 1 & 0 & 1 \\ 1 & 1 & 1 & 0 & 0 \\ 0 & 1 & 1 & 1 & 0 \\ 1 & 1 & 0 & 1 & 1 \\ 1 & 1 & 1 & 1 & 0 \end{pmatrix}$$

Merge – example

Example of a 2-merge and a 3-merge in the case of factorization.

$$\begin{pmatrix} 1 & 1 & 0 & 1 & 0 \\ 0 & 1 & 1 & 1 & 0 \\ 1 & 1 & 1 & 0 & 0 \\ 0 & 1 & 1 & 1 & 0 \\ 1 & 1 & 0 & 1 & 1 \\ 1 & 1 & 1 & 1 & 0 \end{pmatrix}$$

Merge – example

Example of a 2-merge and a 3-merge in the case of factorization.

$$\left(\begin{array}{ccccc} 1 & 1 & 0 & 1 & 0 \\ 0 & 1 & 1 & 1 & 0 \\ 1 & 1 & 1 & 0 & 0 \\ 0 & 1 & 1 & 1 & 0 \\ \cancel{1} & \cancel{1} & \cancel{0} & \cancel{1} & \cancel{1} \\ 1 & 1 & 1 & 1 & 0 \end{array}\right)$$

Merge – example

Example of a 2-merge and a 3-merge in the case of factorization.

$$\begin{pmatrix} 1 & 1 & 0 & 1 & 0 \\ 0 & 1 & 1 & 1 & 0 \\ 1 & 1 & 1 & 0 & 0 \\ 0 & 1 & 1 & 1 & 0 \\ \cancel{1} & \cancel{1} & \cancel{0} & \cancel{1} & \cancel{1} \\ 1 & 1 & 1 & 1 & 0 \end{pmatrix}$$

Merge – example

Example of a 2-merge and a 3-merge in the case of factorization.

$$\begin{pmatrix} 1 & 1 & 0 & 1 & 0 \\ 0 & 1 & 1 & 1 & 0 \\ 0 & 0 & 1 & 1 & 0 \\ 0 & 1 & 1 & 1 & 0 \\ \cancel{1} & \cancel{1} & \cancel{0} & \cancel{1} & \cancel{1} \\ 0 & 0 & 1 & 0 & 0 \end{pmatrix}$$

Merge – example

Example of a 2-merge and a 3-merge in the case of factorization.

$$\begin{pmatrix} 1 & 1 & 0 & 1 & 0 \\ 0 & 1 & 1 & 1 & 0 \\ 0 & 0 & 1 & 1 & 0 \\ 0 & 1 & 1 & 1 & 0 \\ 1 & 1 & 0 & 1 & 1 \\ 0 & 0 & 1 & 0 & 0 \end{pmatrix}$$

Merge – how do we do it?

- For a k-merge with $k \geq 3$, there is more than one way to perform the additions to create a singleton.
- We want to minimize the fill-in of the matrix, i.e., we want to minimize the increase of the total weight of the matrix.

Merge – simple approach

Simple approach:

- Select the row with smallest weight among our set of k rows.
- Subtract an appropriate combination of this row to the other rows.

The Markowitz rule

If the lightest row among k has weight w, the incurred fill-in (difference in total matrix weight) is:

$$\begin{aligned}&(k-1)(w-1)-(k-1)-w\\&=(k-1)(w-2)-w+2-2\\&=(k-2)(w-2)-2\end{aligned}$$

Merge – minimal spanning tree

The simple approach may be refined:

- In order to find the way of performing a k-merge that minimize the fill-in, we compute a minimal spanning tree.
- The minimal spanning tree is computed on the graph where:
  - the nodes are the k rows involved in the k-merge;
  - the weight of the edge between two rows is the weight of the sum of these two rows.
- This minimal fill-in may be below the Markowitz estimate.

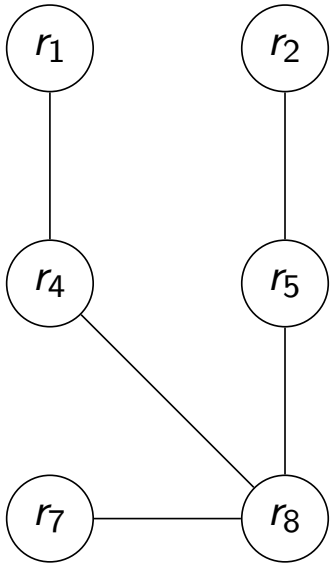
This approach matches the simple one if the MST happens to be a star (with arm length 1).

Merge – example

Example of the use of a minimal spanning tree for a 6-merge in the case of factorization.

$$\begin{pmatrix} 0 & 1 & 1 & 0 & 1 & 1 \\ 1 & 1 & 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix}$$

	r_2	r_4	r_5	r_7	r_8
r_1	4	2	4	2	3
r_2		4	2	4	3
r_4			2	2	1
r_5				2	1
r_7					1



Merge – implementation

- All possible k-merges (for $2 \leq k \leq k_{\max}$) are in a priority queue sorted (in ascending order) by the fill-in implied by the merges.
- Each time a merge is performed, the matrix changes and the minimal spanning trees of the other merges can change.
- Unfortunately, it is too costly to recompute all minimal spanning trees after each change. We approximate the fill-in with Markowitz count.

$$(k-2)(w_{\min}-2)-2$$

Merge – implementation

- Merge reduces the size of the matrix but increases the total weight of the matrix:
- Performing a k-merge reduces the number of rows and columns of the matrix by 1.
- The excess is not modified.
- A 2-merge reduce the total weight of the matrix by at least 2.
- But, in general, performing a k-merge, with $k \geq 3$, increases the total weight of the matrix.

- In merge, the values of the non-zero coefficients matter.

Merge – stopping criteria

- `merge` must be stopped while the matrix is still sparse.
- A possible criteria is to use an estimate of the time of the linear algebra step, based on information such as:
  - The current matrix size. (which $\searrow$ as time goes).
  - The current total matrix weight. (which $\nearrow$ as time goes).
  - Finer grained data can also be used.
- We can stop `merge` once the estimate that is used starts to increase.
- In practice, `merge` is performed until a given average weight per row is reached.

The binaries for merge in Cado-NFS

- The merge stage is split in two binaries in Cado-NFS.
- For factorization:
  - `merge`
  - `replay`
- For discrete logarithm:
  - `merge-dl`
  - `replay-dl`
- The binaries `replay` and `replay-dl` build the matrix for the linear algebra step from the results of the merge algorithm.

Plan

The filtering step

Duplicates

Singletons and “clique” removal

Merge

Practical aspects

Oversieving

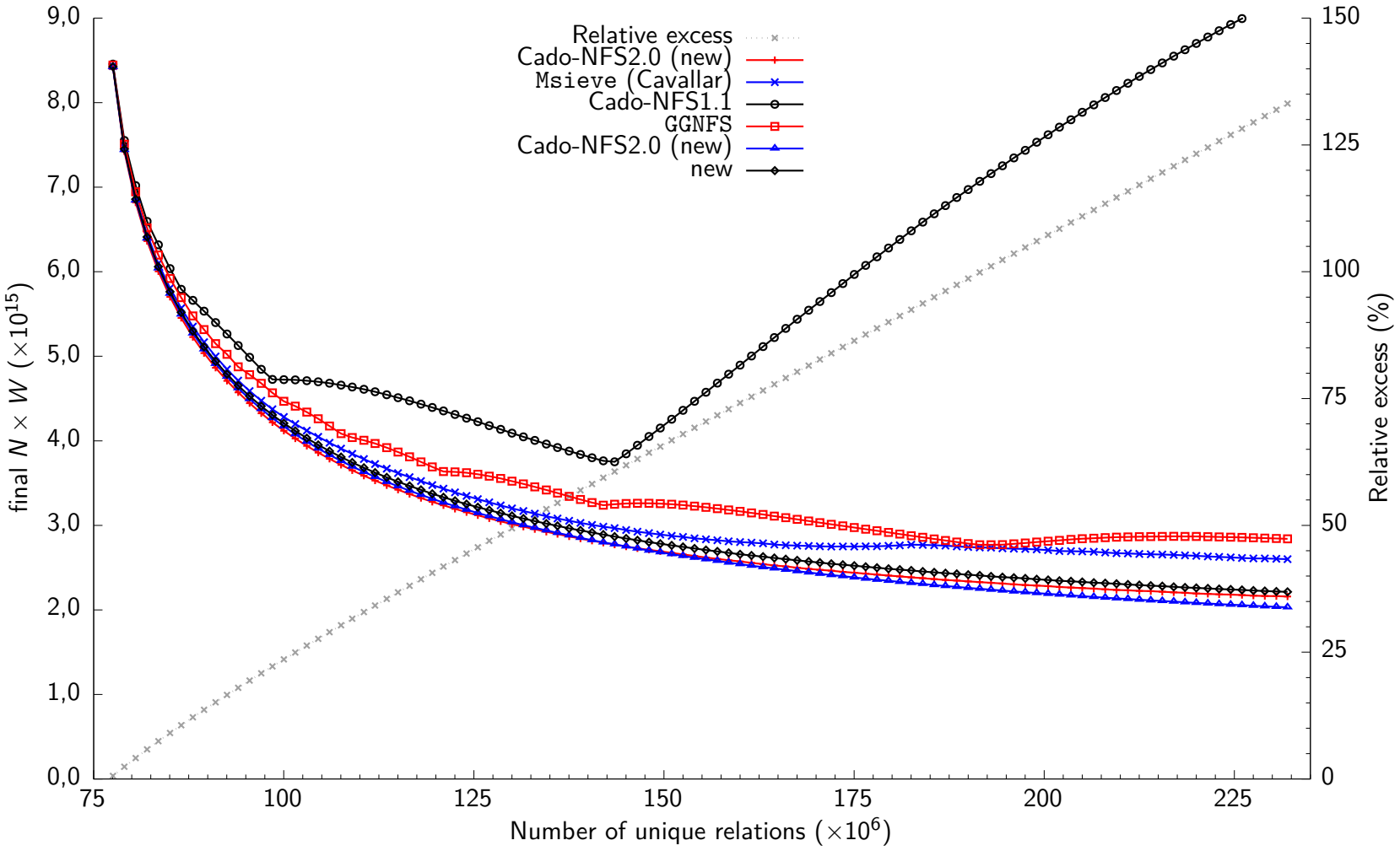
- By computing more relations in the sieving step, we can increase the excess at the beginning of `purge`.
- So more "cliques" can be removed.
- If the weight function used in the "clique" removal algorithm is well-designed, the filtering step should produce a better matrix.
- Trade-off: between the additional time spent in the sieving step and the gain in the linear algebra step.
- The additional time spent in the filtering step due to the additional relations is negligible compared to the time spent in the other steps.

Oversieving – experiment

- Illustrate oversieving with RSA-155.
- We computed a lot more relations that is needed.
- We performed the filtering step with different weight functions and increasing sets of relations.

- To compare results, we used $N \times W$, where
  - N = number of rows of the final matrix
  - W = total weight of the final matrix

Oversieving – results



Oversieving lessons

State-of-the-art implementations have a very flat curve that gives the linear algebra cost estimate as a function of the stopping point.

This explains why the stopping point of the `merge` process is approached in a fairly relaxed way.

- Aim at a ballpark estimate of an average row density ≈ 200.
- See where this brings us.
- Possibly look at what happens for 150, 175, 225, 250.
- It probably makes little difference in the end.

Some experimental data

year	what?	bits	dd	relations	unique	singl	merged
2010	RSA	768	232	64e9	48e9	2.5e9	192e6
2017	DLP	768	232	11e9	9.1e9	?	23.5e6
2019	RSA	795	240	8.9e9	6.0e9	1.2e9	282e6
2019	DLP	795	240	3.8e9	2.4e9	150e6	36e6
2020	RSA	829	250	8.7e9	6.1e9	1.8e9	405e6

Memory footprint

This all depends on the choice of parameters, but overall the `purge` step (singleton and "clique" removal) has the largest memory requirement.

- Input and output data sets are bulky.
- Need to keep lots of info in RAM.
- RSA-829: 1.4TB RAM needed (this has significantly improved since).

The `merge` step generally has a slightly smaller memory footprint. RSA-829: 450GB.

Parallelizing

Recent work has proven that `merge` can be efficiently parallelized, with no noticeable impact on the output quality.

- This is still single-machine parallelization. Multi-machine parallelization attempts have been unsuccessful thus far.
- The `merge` process remains strongly I/O-bound.

Simulation of the filter step

How can we predict the size of the matrix that we obtain after the filter step?

Simulation of the filter step

How can we predict the size of the matrix that we obtain after the filter step?

- Approach 1: create fake relations up to the expected required number, and see what we get after duplicates/singleton/clique/merge.
  - Data processing cost is identical to the real computation.
  - May be hard to use if we want to explore many different parameter sets.

Simulation of the filter step

How can we predict the size of the matrix that we obtain after the filter step?

- Approach 1: create fake relations up to the expected required number, and see what we get after duplicates/singleton/clique/merge.
  - Data processing cost is identical to the real computation.
  - May be hard to use if we want to explore many different parameter sets.
- Approach 2: do the same on a reduced-scale model, say 10- or 100-fold.
  - This works surprisingly well.
  - Still WIP in Cado-NFS, but results are very promising.

CSE291-14: The Number Field Sieve

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

Emmanuel Thomé

February 15, 2022

Part 6b

Linear algebra: introduction

Context

Algorithms

How to multiply a sparse matrix by a vector?

Thread and MPI-level parallelism

Plan

Context

Algorithms

How to multiply a sparse matrix by a vector?

Thread and MPI-level parallelism

The linear algebra step

The linear algebra step per se comes right after the filtering step and before the characters step. However from an abstract point of view, both of the latter are also linear algebra.

Key facts for a bird's eye view:

- The linear algebra step in the Number Field Sieve is the second hardest problem after the sieving step, in terms of computation time.
- In contrast to sieving, which is embarrassingly parallel, linear algebra is more difficult to parallelize. We are led to use hardware with fast interconnect capabilities, and that tends to be more expensive.
- There are some subtle differences between linear algebra for factoring and linear algebra for the discrete logarithm.

Multiple questions

We commonly refer to the linear algebra step. Yet one may ask several questions:

- is the system homogenous / inhomogenous?
- what are the unknowns? Rows or ideals?
- is the system singular?
- is there such a thing as a partial solution?
- does the world collapse if there is an error in my matrix?

Context recap: factoring

We must combine relations so that they consist of only squares.

This rewrites as a linear system.

- Sparse matrix M: relations = rows, columns = prime ideals.
- We seek several (say 64) solutions v to the system

$$vM = 0 \mod 2.$$

Context for DLP

We will learn about the NFS-DL variant for discrete logarithms, which brings in the following differences:

- Linear algebra over $\mathbb{Z}/\ell\mathbb{Z}$.
- Coefficients are most often ± 1.
- We look for a right kernel vector.

Things in common

The matrix in both cases is (almost) the same. It is large and very sparse.

- Theory: in the matrix before the filtering, we can show that asymptotically the number of non-zero coefficients per row is $O(\log\log(\text{number of rows}))$.
- Practice: filtering comes into play and asymptotics are only asymptotics.
  - 2019: 795-bit factoring: 282M rows/cols, density ≈ 200.
  - 2019: 795-bit DLP: 36M rows/cols, density ≈ 250.

Square matrices?

Both examples favored rectangular matrices.

In both cases the excess will be mostly consumed by filtering. However some mild excess is ok, we can in both cases pad with zero columns, so that we get a square matrix.

Exact versus numeric

In both cases, we are dealing with exact linear algebra.

Exact linear algebra

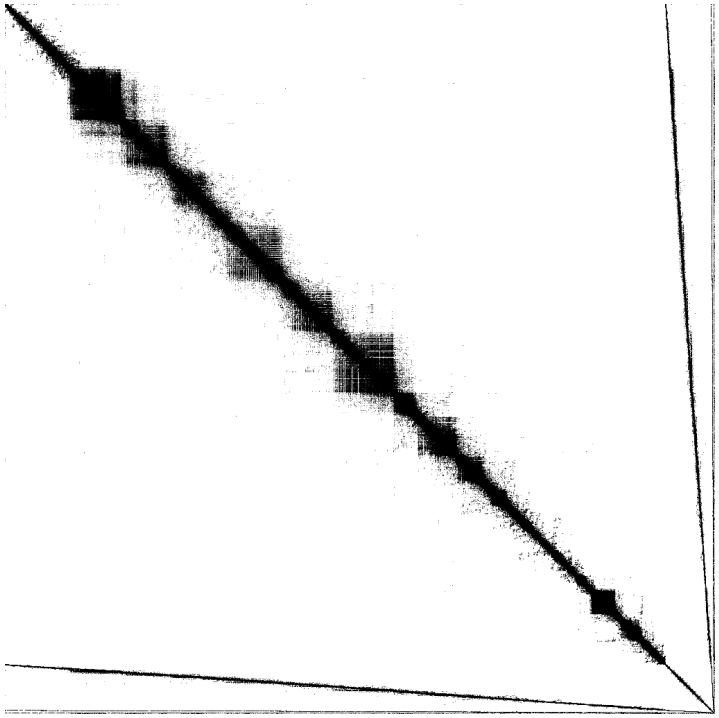
This implies two things.

- We really want the solution, not an approximate one.
- There is no notion of convergence anyway.

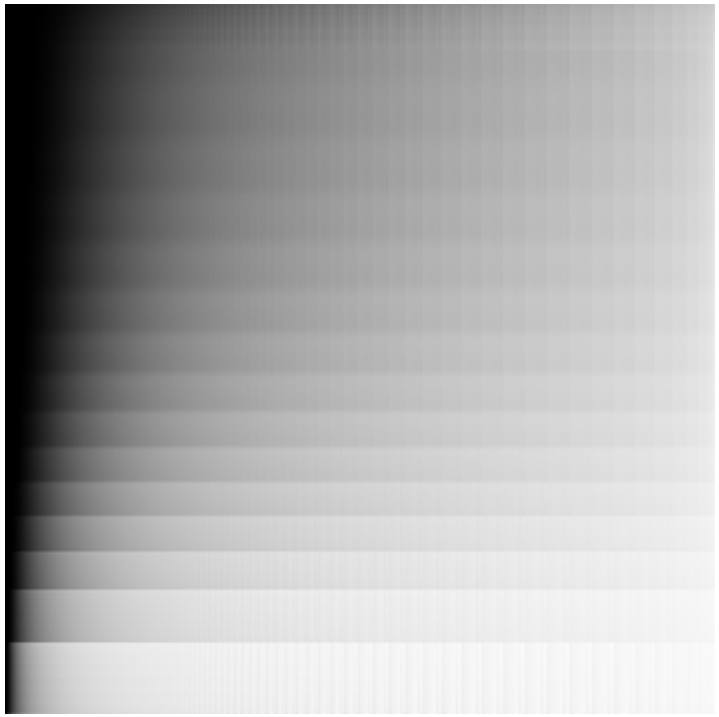
90% to 100% of the nice "linear algebra for PDE solving" book on your shelf is useless.

$\mathbb{F}_2$ is exact, and positive characteristic

The matrices are not the same, either.



(some PDE example)



(a factoring matrix)

Nothing like "lower-order" either

It is not uncommon that the far-from-diagonal blocks in a PDE matrix are handled in a delayed way.

- Sometimes, we structurally have zero blocks there.
- Often, coefficients in this blocks have lower-order significance can be ignored, or approximated.

None of these shortcuts is valid in our case.

Plan

Context

Algorithms

How to multiply a sparse matrix by a vector?

Thread and MPI-level parallelism

Which direction

We have seen that both $vM = 0$ and $Mv = 0$ existed as problems to be solved.

For most of the internal dealings of the linear algebra solving, this does not matter much. We can transpose everything if we so desire.

Choice for exposition

In these slides, we present the case where we want to solve

$$Mv = 0$$

over some field $K = \mathbb{F}_\ell$, and we definitely have $\ell = 2$ on the radar.

Black box

Caveat

We like to use N for the number of rows and columns, but we must pay attention not to confuse that with the integer to be factored.

From now on, given that we have no use of the integer to be factored, N denotes the number of rows.

We have an $N \times N$ matrix M. We want to solve $Mw = 0$.

The matrix M is large, (very) sparse, and defined over $K = \mathbb{F}_\ell$.

Because of sparsity, we want a black box algorithm.

$$v \longrightarrow \boxed{M} \longrightarrow M \times v$$

Gauss?

Why can't we use Gaussian elimination?

Any algorithm that modifies the matrix inevitably causes fill-in.

- Smart recipes can be used to minimize fill-in somewhat, but it is still there.
- Example for the RSA768 matrix: a dense bit matrix of this size costs 4200TB to store.

This is the justification for black-box algorithms.

Are black boxes useful at all?

An example in numerical analysis.

- Take a random vector v.
- Iterate $v \leftarrow Mv/\|Mv\|$.
- If M has a dominant eigenvalue λ, $\frac{\|Mv\|}{\|v\|} \to |\lambda|$.

If we can do such things, no doubt we can do more.

How much of this black-box technology applies to finite fields?

Some terminology

In the numerical linear algebra world, the following distinction exists:

- direct methods may modify the matrix: Gauss, LU, ...
- indirect (or iterative) methods: same as black-box methods.

In the numerical context, an indirect method that involves an $N \times N$ matrix can very well obtain a satisfactory result with only $o(N)$ applications of the black box.

In contrast, in the exact setting, we typically need $\Theta(N)$ applications of the black box!

Existing black box algorithms

The numerical context knows several indirect methods, often ranked according to the trade-off between iteration complexity, numerical stability, and convergence speed.

In practice, the simplest one is the Lanczos method.

- Very simple iteration ($\approx$ Gram-Schmidt orthogonalization).
- We work with the symmetric matrix $M^T M$.
- Application to finite fields is a bit of a gamble, since we have isotropic vectors: $x^T M^T M x = 0 \mod \ell$.
  - Not that much of a problem in large characteristic. Failure rate about $1/\ell$, no big deal.
  - Much more annoying if $\ell = 2$!!

Adaptation to finite fields

A very inefficient method to adapt to any finite field:

- Embed into $\mathbb{F}_{2^n}$, consider $M^T DM$ with D diagonal.
- This entails a more than n-fold overhead.

Much better alternatives:

- Finite field-native algorithms.
- Block methods.

Block algorithms

Extend the black box notion, pretty much in a SIMD way.

$$\mathbf{v} \longrightarrow \boxed{M} \longrightarrow M \times \mathbf{v}$$

$\mathbf{v}$ and $M\mathbf{v}$: blocks of vectors.

Example for binary matrices

A "block black box" makes perfect sense over $\mathbb{F}_2$.

- Let $\mathbf{v}$ be a sequence of N 64-bit integers.
  This can be viewed as 64 vectors of bits.
- Compute $M\mathbf{v}$ with bitwise XORs on 64-bit types.

$$\mathbf{v}'_i = \sum_{j, M_{i,j} \neq 0} \mathbf{v}_j.$$

Main observations and questions

- We do more work in almost the same time.
- Can this be put to some use? E.g., use the block black box fewer times that we would have used the (non-SIMD) black box?

Algorithms for finite fields

- 1986: Wiedemann's algorithm.
- 1991-1995: Block Lanczos algorithm (Montgomery), Block Wiedemann algorithm (Coppersmith).

The probability of success of these algorithms has been studied a lot by the computer algebra community (mid 1990s to early 2010s, mostly).

Plan

Context

Algorithms

How to multiply a sparse matrix by a vector?

Thread and MPI-level parallelism

First things first: transposition

All matrix-times-vector implementations can be transposed easily to give an implementation of $v \leftarrow Mv$ based on an example code that does $v \leftarrow vM$, and vice-versa.

- This is true at least in theory, as there's really a generic code transform which does that.
- In practice, it depends on how the code is written.
- Performance may not be 1:1 (read and write are not exactly the same!)

Simplistic approach

Assume that we have two large memory areas for the input and output vectors. (64-bit integers, or integers mod ℓ).

```
for(i = 0 ; i < N ; ++i) {
   w_i = 0;
   for(j = 0 ; j < N ; ++j) {
      if (M_{i,j} != 0)
         w_i = w_i + M_{i,j} v_j;                    // XOR or addmul_si
   }
}
```

Multiple problems.

Some obvious improvements

Only store the locations (and values) of non-zero coefficients in M.

```
for(i = 0 ; i < N ; ++i) {
  w_i = 0;
  for(j ∈ indices of non-zero coefficients in row i) {
    w_i = w_i + M_{i,j} v_j;                    // XOR or addmul_si
  }
}
```

- Possible discussions about the data format (size, random access, etc). Note that in our context, adapting to any data format is certainly affordable!
  (basically, data is code. Decide on the code flow, then adjust the data format accordingly)
- Still very poor performance.

Accesses are very scattered

The j indices of non-zero coefficients in a row are very far apart.

- Most reads of v_j will be cache misses.
- This yields (inverse) throughputs of hundreds of CPU cycle per coefficient.

Improvement strategies

Straightfoward approach: split the matrix in blocks.

- Either fixed-size blocks (but what about the processing order?)
- Or: NW, NE, SE, SW and recurse, until some cut-off.

Problem: density is not uniform at all!
The vast majority of coefficients will still incur cache misses.

Nonuniform density

Because of the nonuniform density:

- Coefficients in the heaviest columns are processed very fast, even with the naive methods.
- As we reach more sparse areas, performance drops sharply.

This calls for some kind of adaptive mechanism.

- Process the "dense" vertical band with the heaviest columns with a fast simple-minded approach.
- Find a way to deal with the more sparse parts.

Two-pass

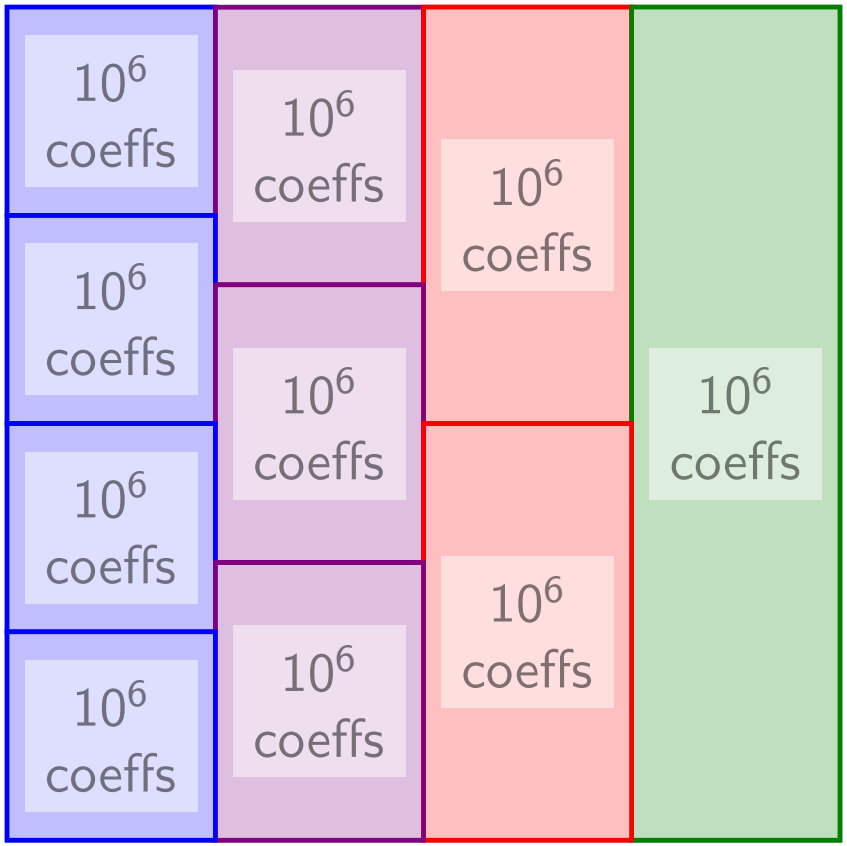
Preferred approach (totally reminiscent of bucket sieving!)

Assume that column density of M is $\searrow$. Let the control flow evolve as column density decreases.

- Have several temporary lists, on per column density cutoff.
- Load coefficients, arrange them in temporary memory in a way that is:
  - quickly accessible at the moment we do the memory store.
  - convenient to read back when we eventually store to the output vector.
- Access the temporary lists, store to output vector.

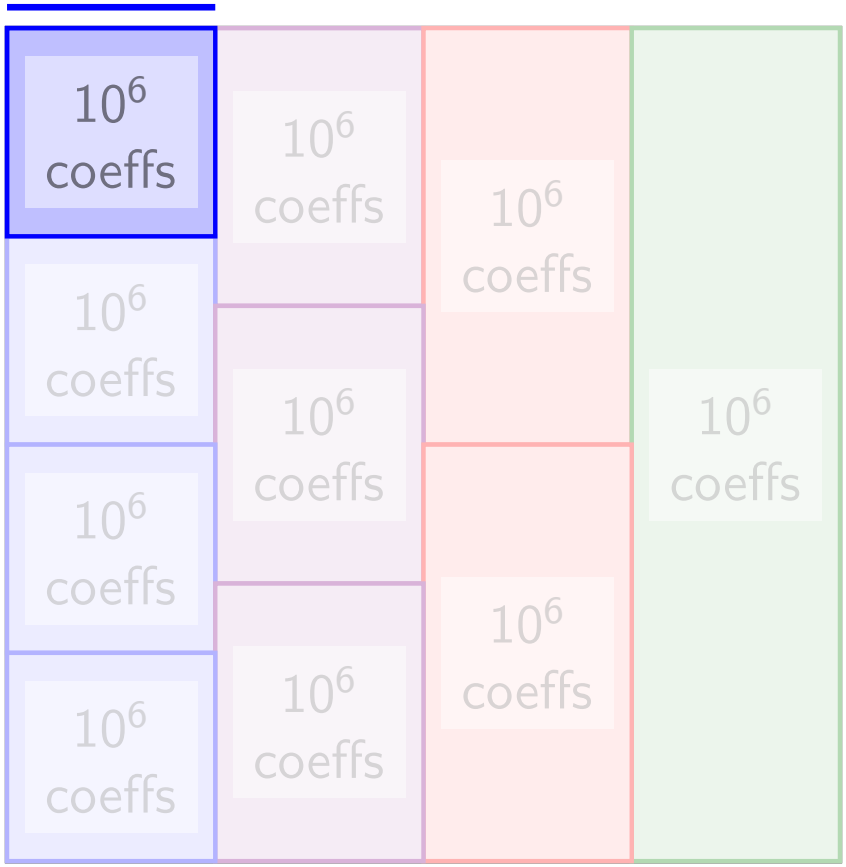
Example for $M \times v$

Note: The dimensions of the vertical bands may be uneven.
We want to limit to the reach of efficient random access.



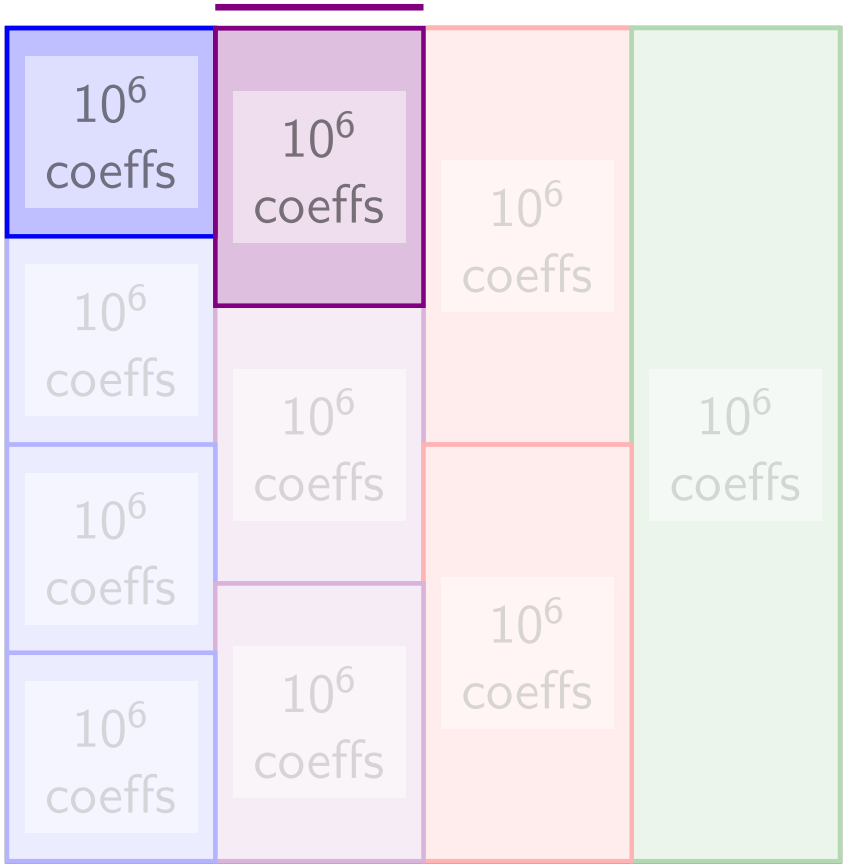
Example for $M \times v$

Load a batch of coefficients from the source vector.
Copy to list L_1, by increasing row index.



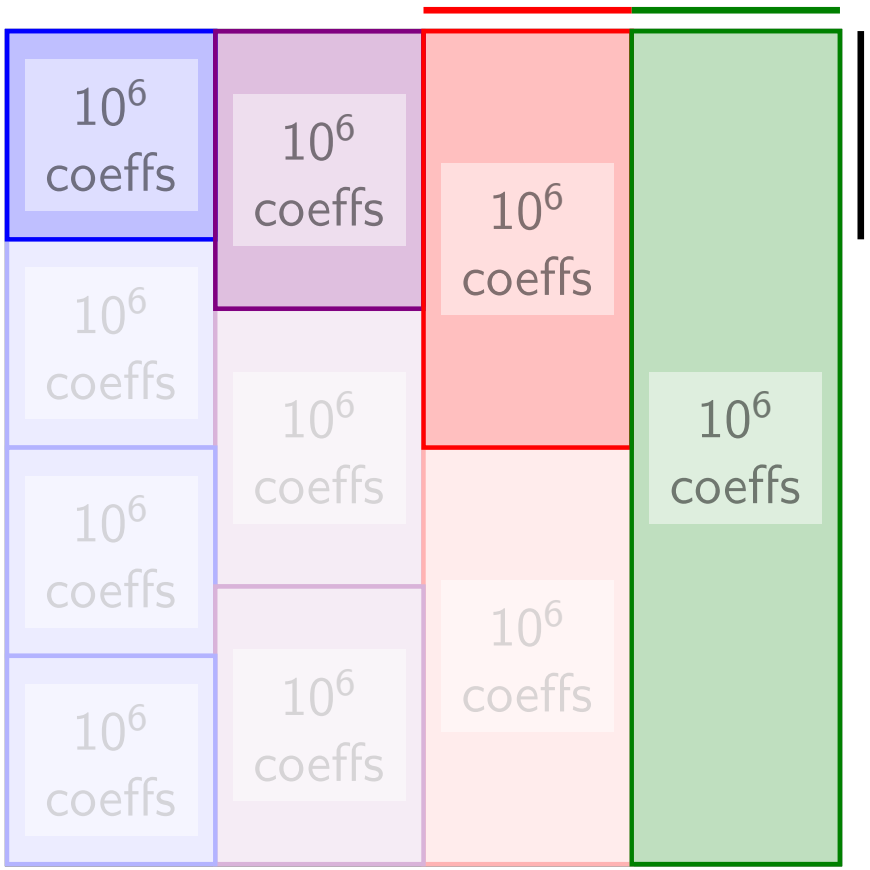
Example for $M \times v$

Load some new coefficients. Do the same, fill list L_2.



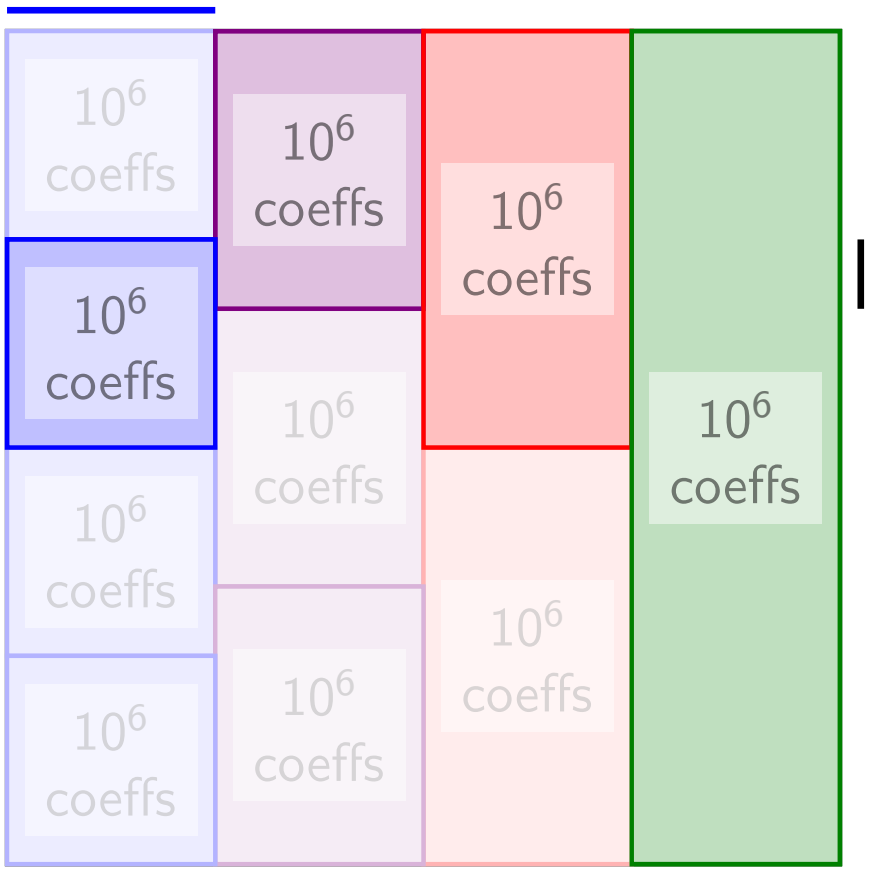
Example for $M \times v$

We have four lists which can be read in order, allowing us to compute a sequence of coefficients from the destination vector.



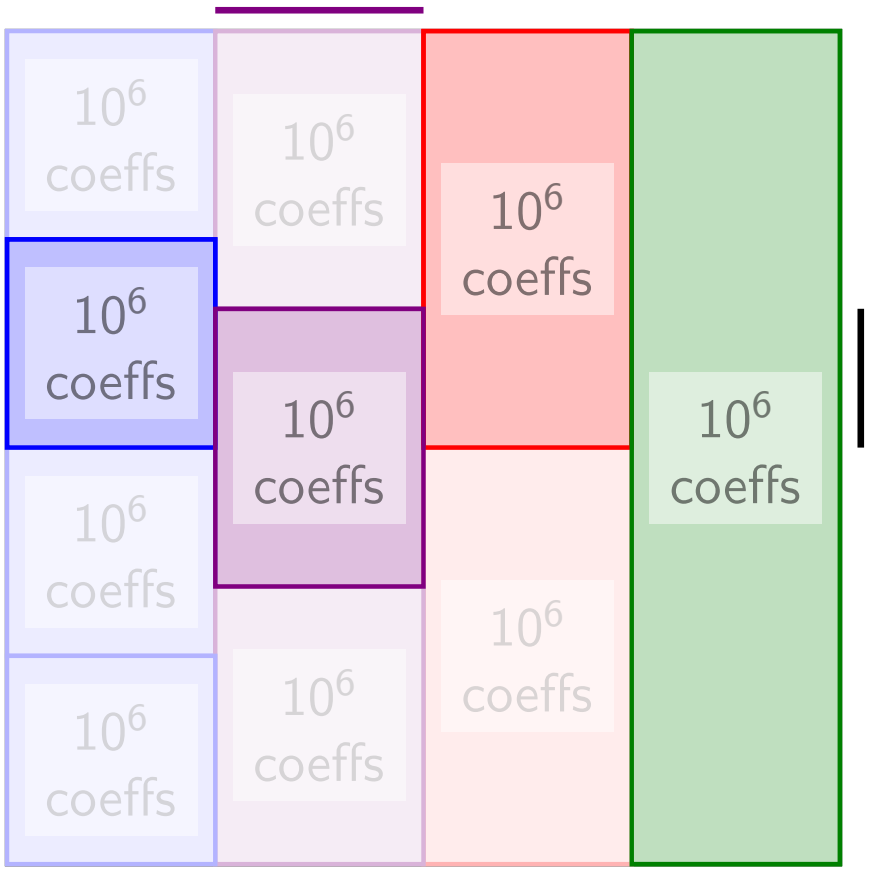
Example for $M \times v$

Periodically, we need to read again a batch of coefficients from the source vector, in order to refill the list L_1.



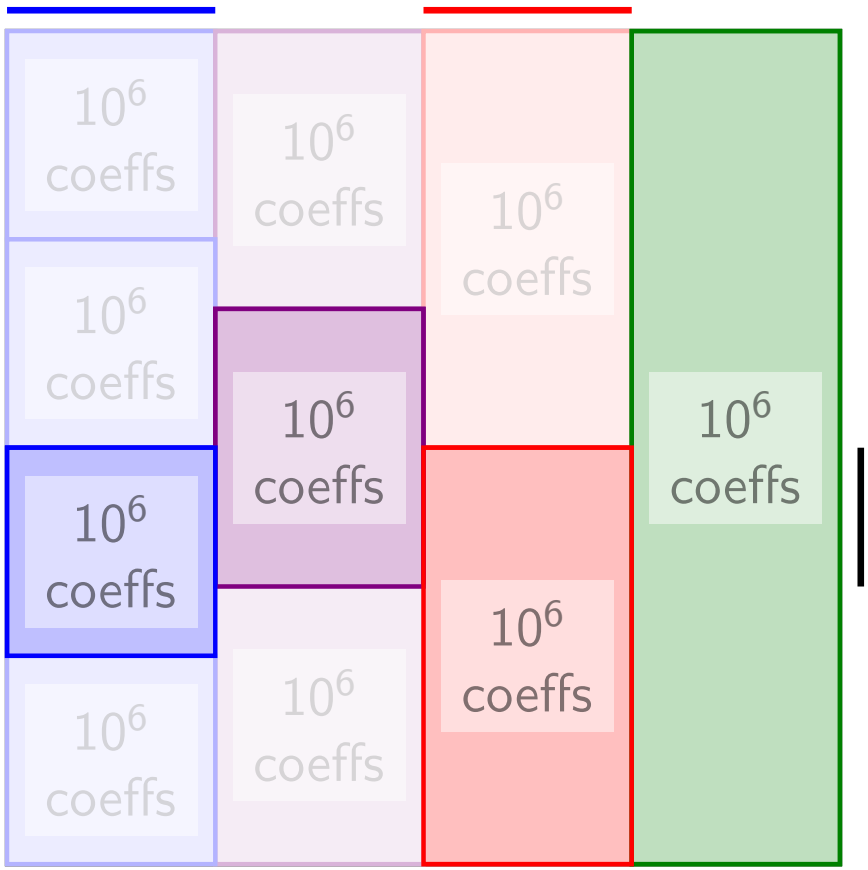
Example for $M \times v$

Periodically, we need to read again a batch of coefficients from the source vector, in order to refill the list L_1.



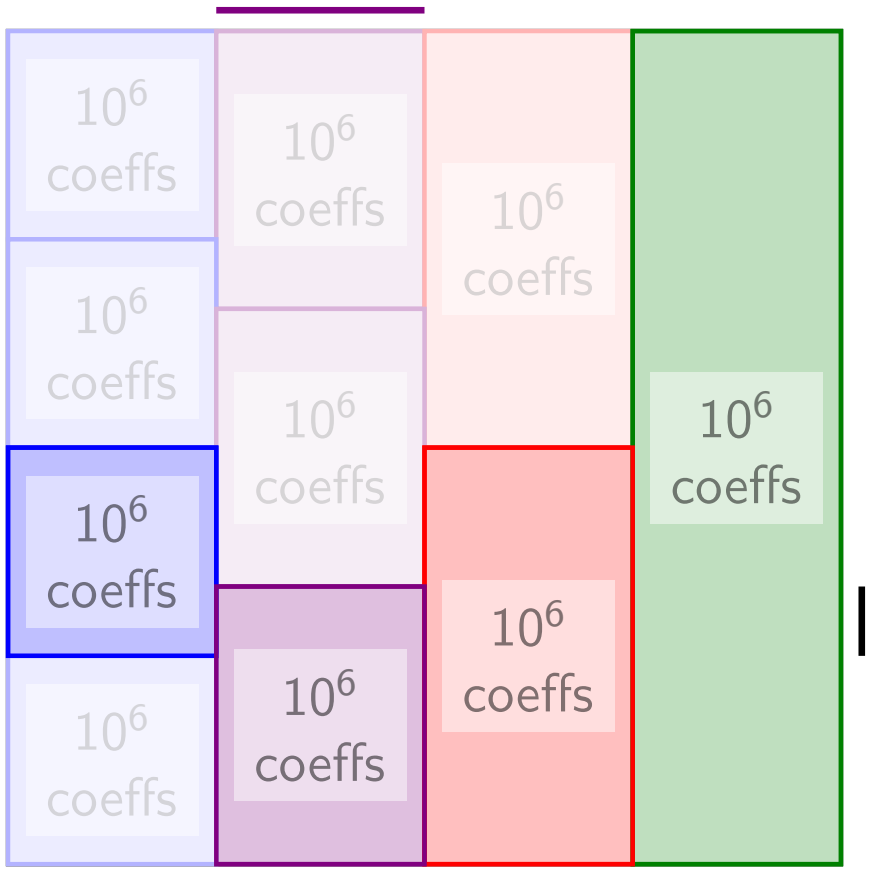
Example for $M \times v$

Periodically, we need to read again a batch of coefficients from the source vector, in order to refill the list L_1.



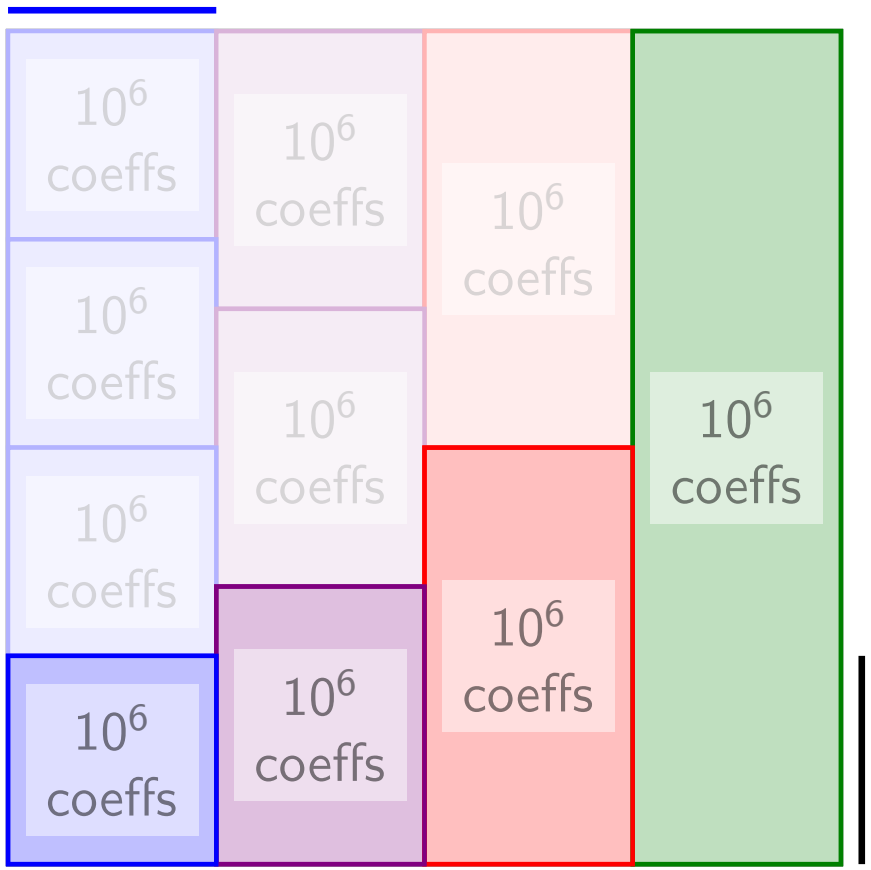
Example for $M \times v$

Periodically, we need to read again a batch of coefficients from the source vector, in order to refill the list L_1.



Example for $M \times v$

Periodically, we need to read again a batch of coefficients from the source vector, in order to refill the list L_1.



Analysis

- Each time the source vector is read, its data is used as much as we can.
  This amortizes the cost of reading data outside cache!
- We fill the list L_i with source vector coefficients exactly in the order they will be read later on.
- The destination vector is written to progressively.
  At the same time, data is read from many lists L_i in parallel.

Note: this applies also to $v \times M$:

- progressive writes $\rightarrow$ progressive reads.
- random-access reads $\rightarrow$ random-access $+=$.

Bucketized matrix-times-vector

Implementation details are somewhat hairy.

- Ideally, dynamic tuning would be welcome: we need to appreciate at runtime how many random-access writes / parallel reads can be sustained by multiple cores simultaneously. Hard-coded thresholds are unsatisfactory.
- Cutoffs don't necessarily match our picture that well.
- Maybe a two-layer approach could make sense.

Plan

Context

Algorithms

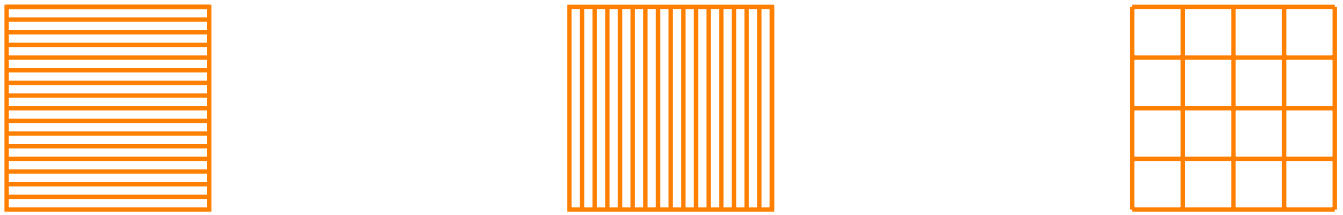
How to multiply a sparse matrix by a vector?

Thread and MPI-level parallelism

Thread- and MPI- level

To a certain extent, the concerns are similar.

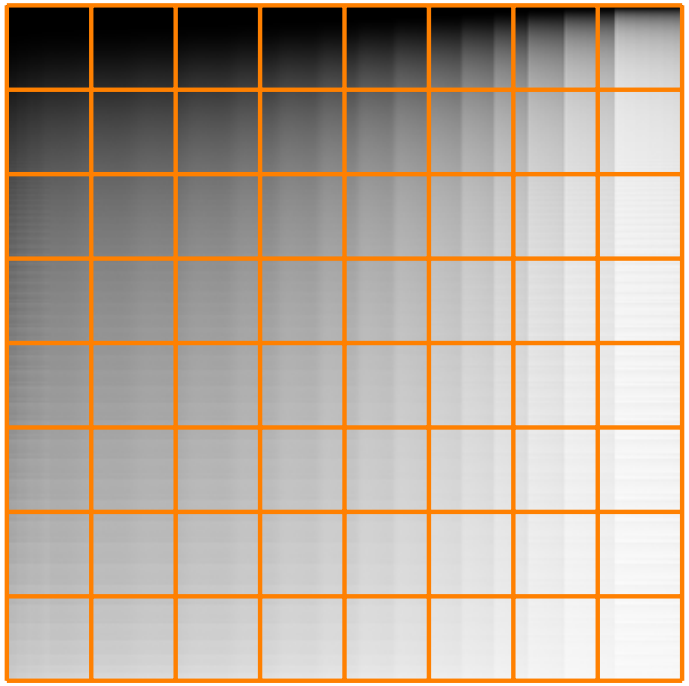
How could we split our matrix ?



The latter is preferred because it goes in the direction of favoring data locality.

Data exchange happens on each row or each column of the mesh.

Plans for splitting the matrix



Quite clearly a bad idea.

- Some nodes or threads would have a much harder time than others doing the product.
- All other nodes will have to wait for them.

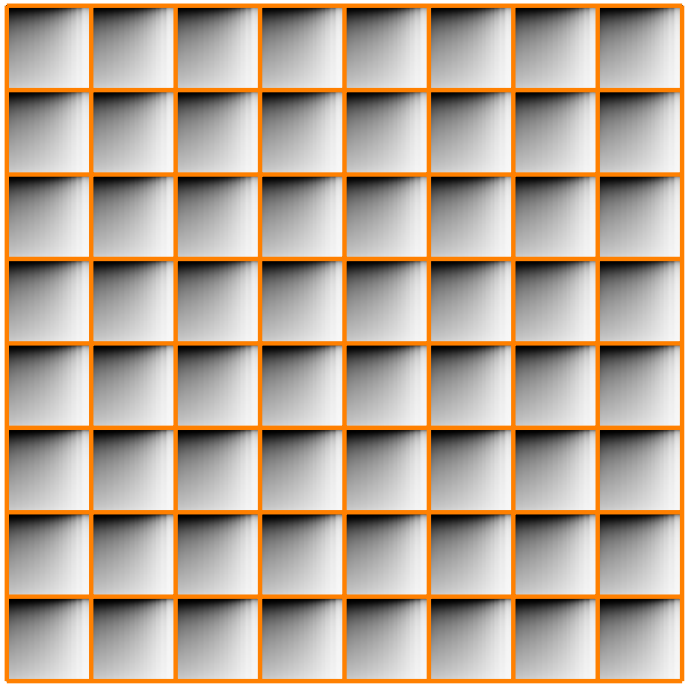
We may reorganize data

We are not bound to the internal representation of the matrix. We may rearrange coefficients (in a compatible way) so the splitting becomes better.

Balancing per-node weights

We may:
- sort columns, and distribute evenly $\Rightarrow T \in \mathfrak{S}_N$.
- sort rows, and distribute evenly $\Rightarrow S \in \mathfrak{S}_N$.

...because $Mv = 0$ or $SMTv = 0$ are reducible to one another !

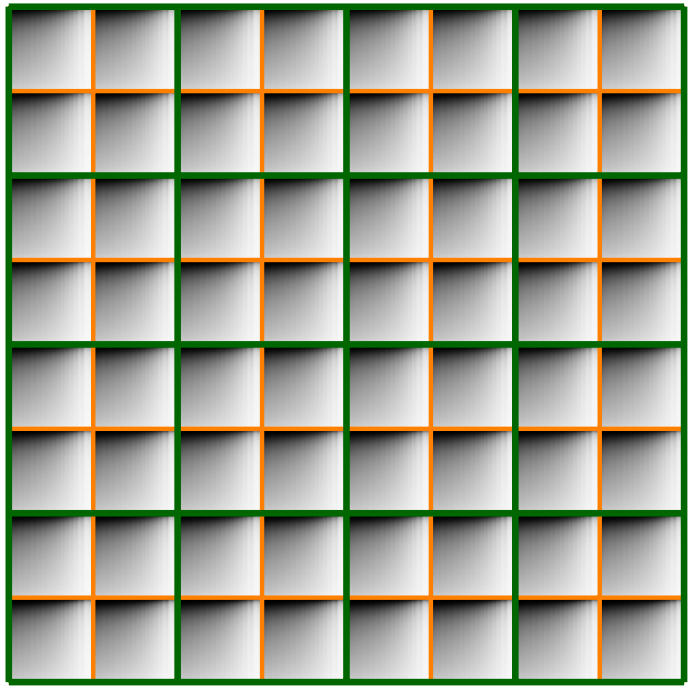


But the balancing story goes much beyond this.

Balancing per-node weights

We may:
- sort columns, and distribute evenly $\Rightarrow T \in \mathfrak{S}_N$.
- sort rows, and distribute evenly $\Rightarrow S \in \mathfrak{S}_N$.

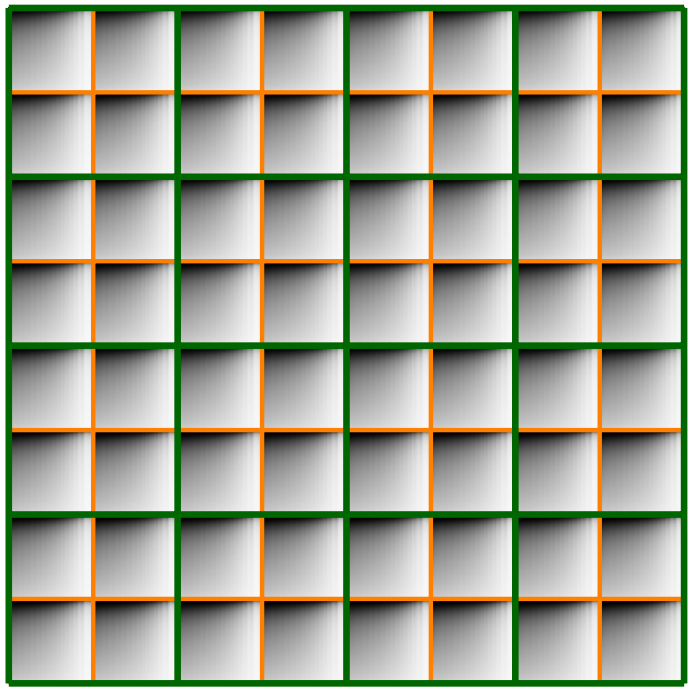
. . . because $Mv = 0$ or $SMTv = 0$ are reducible to one another !



- Core-level splits
- Node-level splits

But the balancing story goes much beyond this.

Balancing per-node weights



- Core-level splits
- Node-level splits

In Cado-NFS, the core-level and thread-level splits are specified with `thr=` and `mpi=` (here, 2x2 and 4x4).

A simple communication algorithm

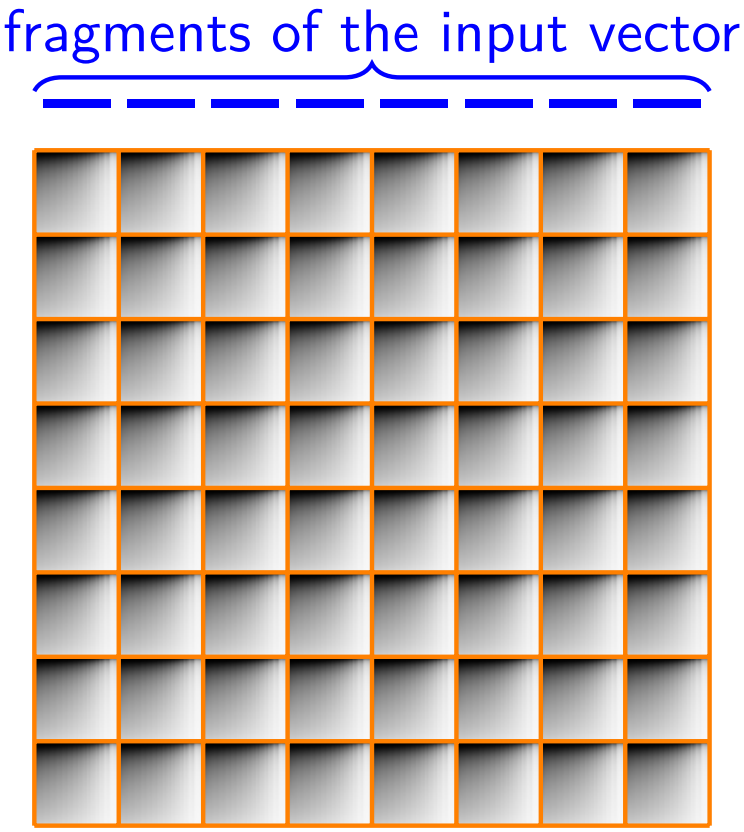
How do several nodes work together for computing $M \times v$?

Simplifying assumptions:
- square mesh of size $t \times t$.
- ignore core split vs node split.

Task list for each node (mesh row i, mesh column j):

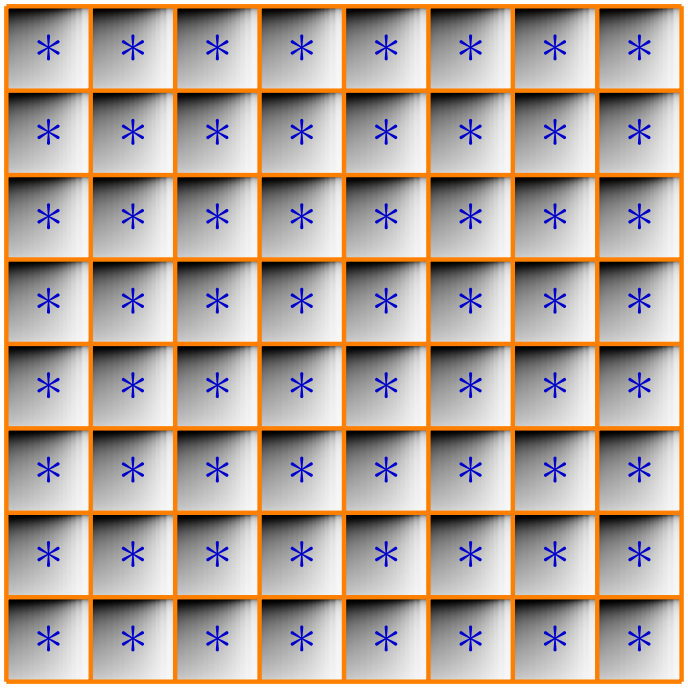
- Have a fraction $(1/t)$ of the input vector (j-th part).
- Compute the local product.
- Sum all the contributions across one row, for the fraction of the destination vector.
  Accumulate the sum on the i-th node of the row.
- On each column, the j-th node broadcasts the part of the computed vector for the next product.

Simple MPI product, graphically



- Initial situation:
  all fragments of the input vector are present in columns.

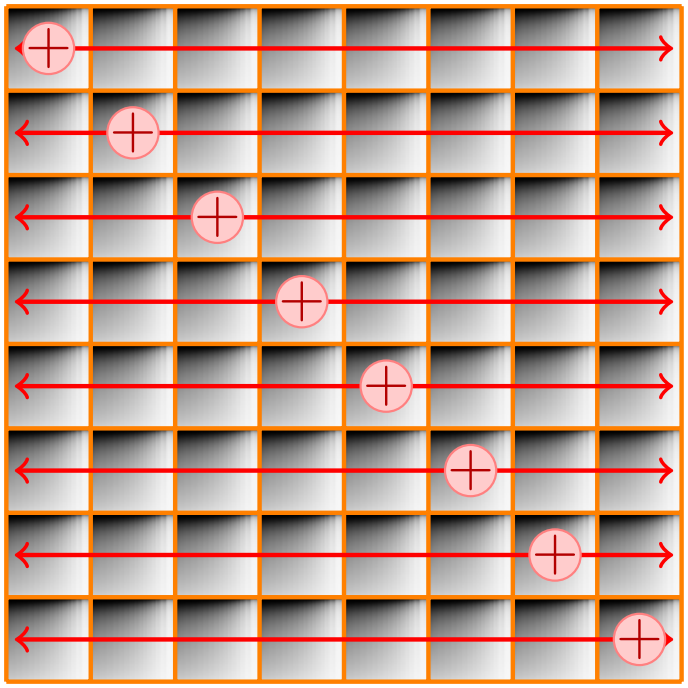
Simple MPI product, graphically



- Step 1: local multiplications.

No communication here.

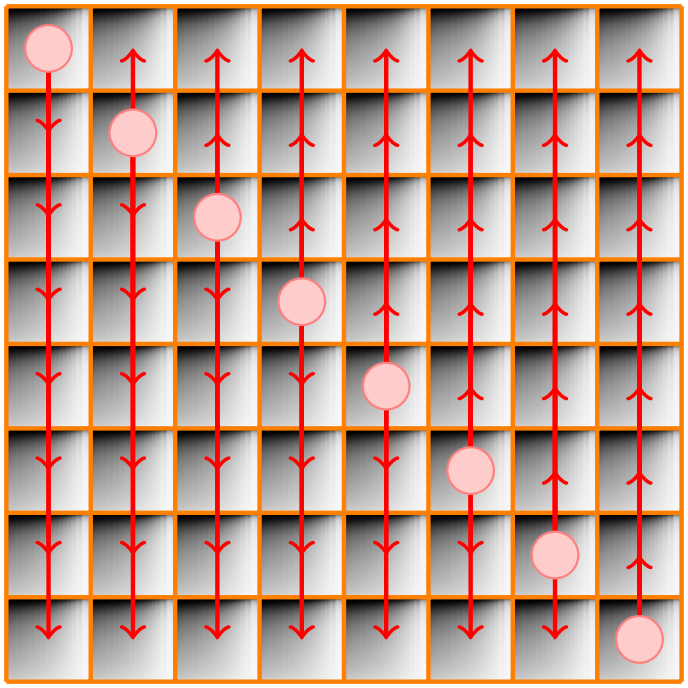
Simple MPI product, graphically



- Step 2: reduction across rows.

  `MPI_Reduce`

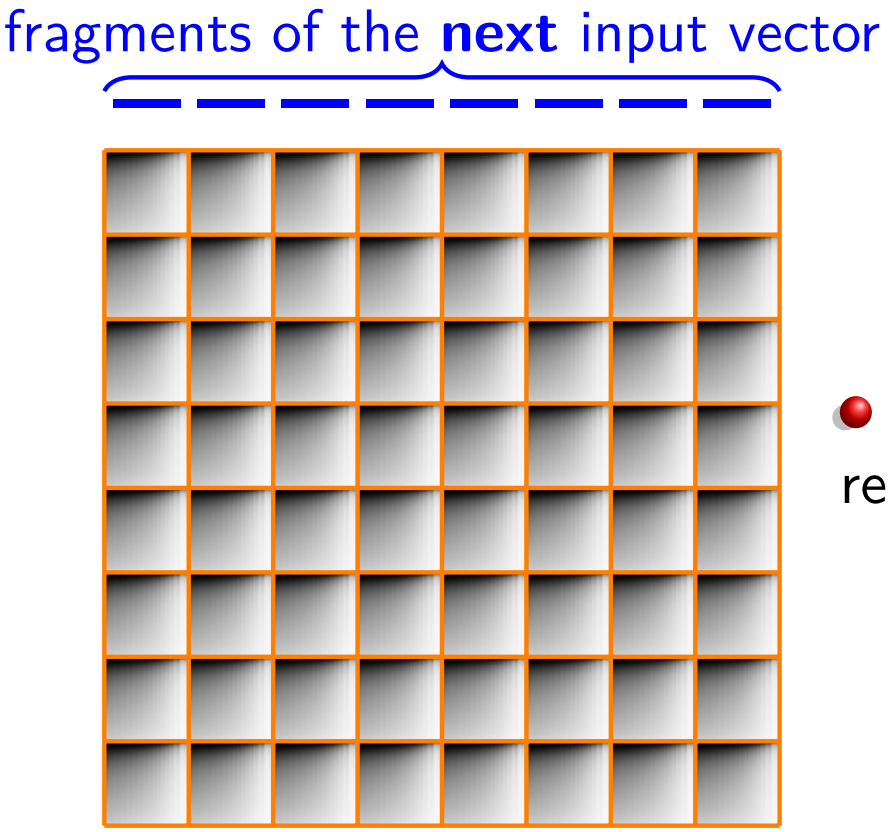
Simple MPI product, graphically



- Step 3: broadcast across columns.

  `MPI_Bcast`

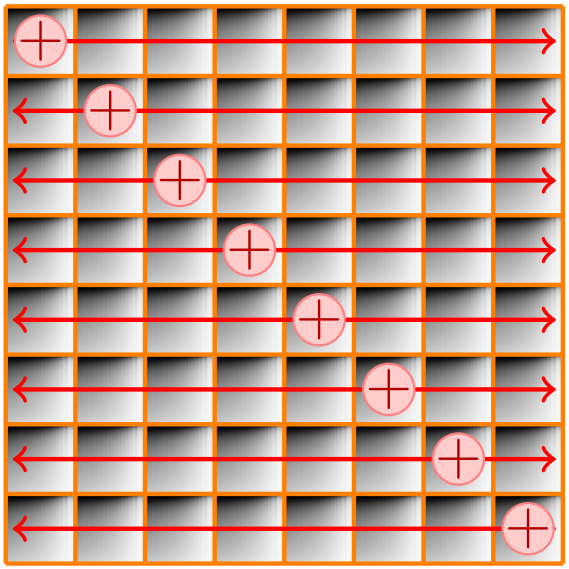
Simple MPI product, graphically



- Final situation:
  ready for next iteration !

There are some downsides with this approach.

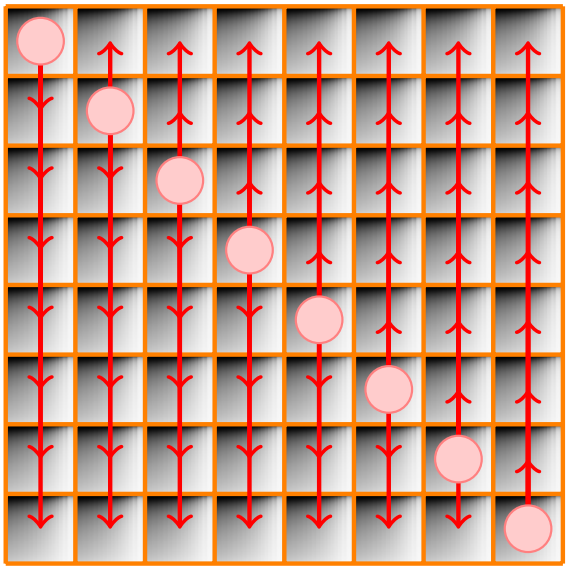
Limitations of the simple scheme



During the reduction step.

- I/O not balanced across rows.
- If linear complexity algorithm:
  - work ratio $1 : n$.
- If log-complexity algorithm:
  - work ratio $1 : \log_2 n$.
  - Logarithmic delay incurred.

Limitations of the simple scheme

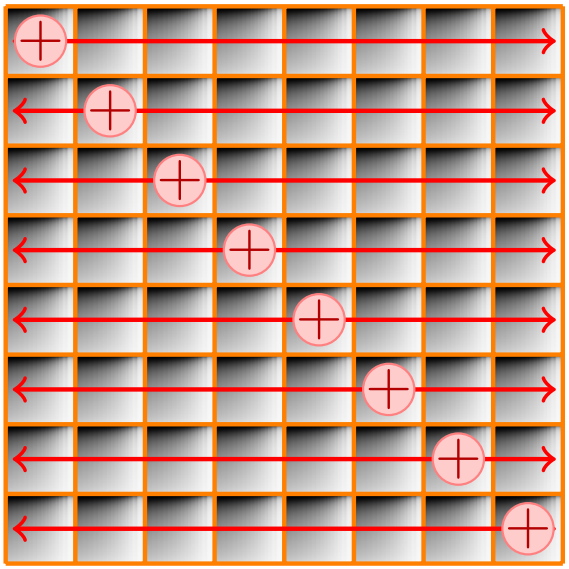


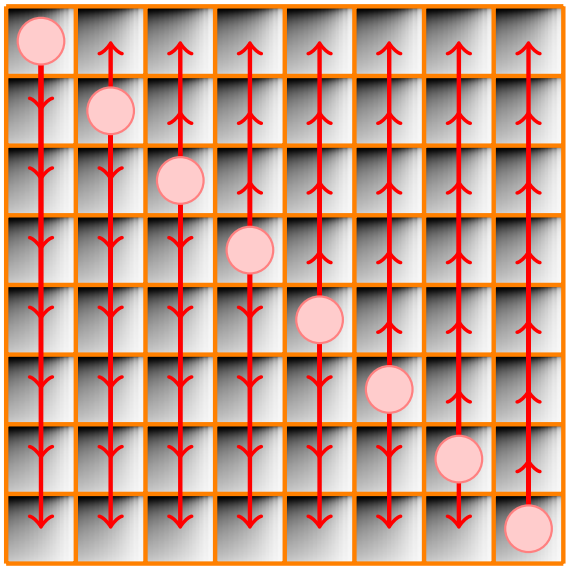
During the broadcast step.

- Exactly the same issue.
- Hypothetical hardware-level multicast? The answer is mostly no.

(AFAIK, no MPI impl. uses IB mcast !)

Limitations of the simple scheme



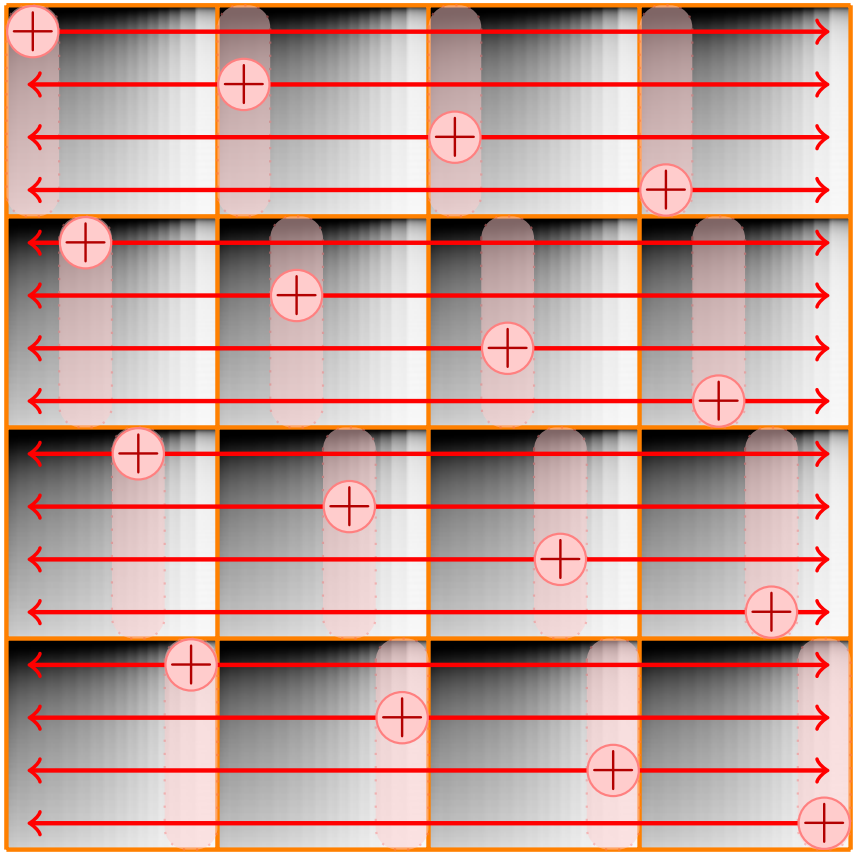


(Obvious) answer

It is better to parallelize work.

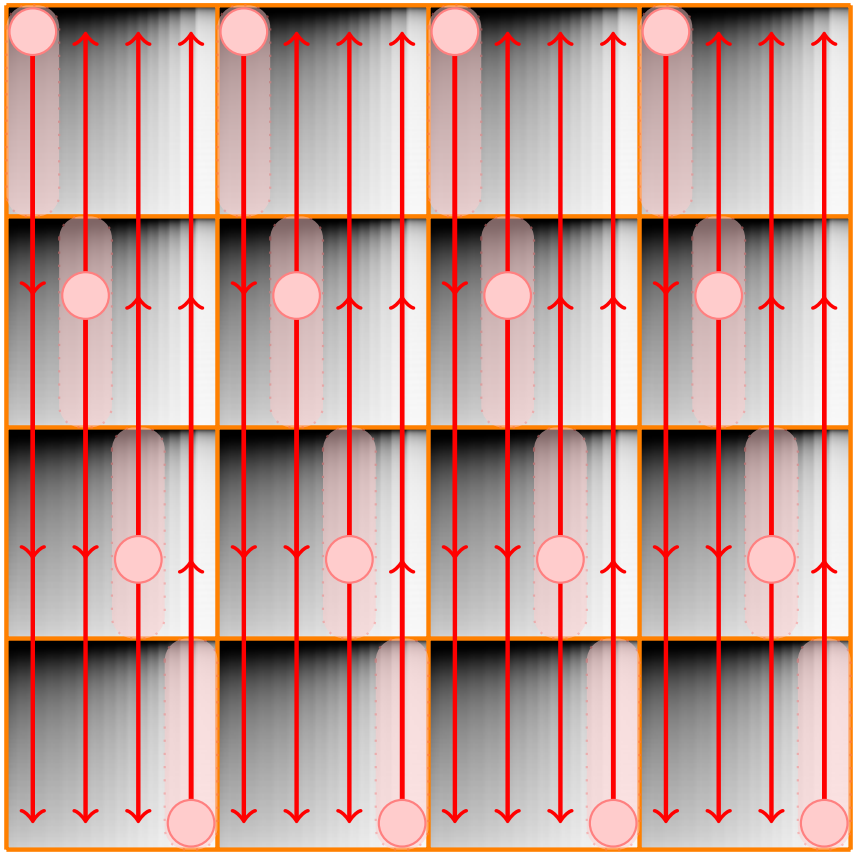
- Reduction step to collect parts of size $\frac{1}{t^2}$ on each t nodes.
- Bcast step to run t broadcasts of size $\frac{1}{t^2}$ in each col.

Parallelized MPI collectives



This even bears a name in MPI dialect: • `MPI_Reduce_scatter`.

Parallelized MPI collectives



This even bears a name in MPI dialect:

- `MPI_Reduce_scatter`.
- `MPI_AllGather`.

Important for performance: `cpubinding`

Cado-NFS uses a feature called CPU binding.

Threads are "pinned" to specific cores, or groups of cores.

We prevent them from wandering too far from the memory they use the most.

A configuration file has to be passed to `bwc.pl`. It should be adjusted specifically for each machine.

- Cado-NFS looks for a section that matches the current machine, and then looks for an entry that matches the `thr` parameter.
- An example file (good starting point) with documentation is in `parameters/misc/cpubinding.conf`
- Trial and error is the only good way.

Building for MPI

Caveat: by default, Cado-NFS builds in non-MPI mode.

In order to build for MPI, one must pass an environment variable to `make`, e.g.:

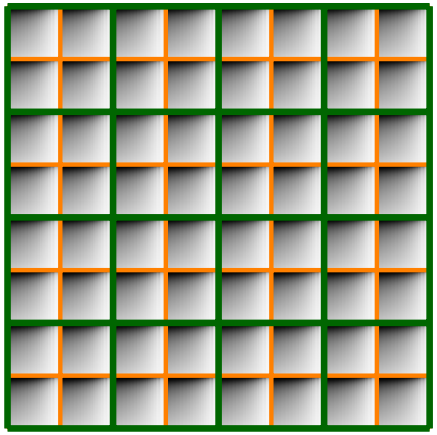
```
MPI=/opt/openmpi-x.y.z/ make -j4
```

The MPI-enabled binaries should preferrably not be mixed with the non-MPI ones. Wipe your build directory first.

Caveat: jitter

We have:

- sorted columns, and distributed them evenly.
- sorted rows, and distributed them evenly.



Things don't always go well

Use case: we split a matrix with average density ≈ 200 across (say) 64 nodes with 64 threads on each (4096 cores total)

- Even though rows and columns are evenly balanced globally, the weights of the 4096 sub-matrices can vary a lot!
- The CPU-bound workload per thread will likely differ, and this will cause wait times. Averting this is really hard.

CSE291-14: The Number Field Sieve

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

Emmanuel Thomé

February 17, 2022

Part 6c

Sparse linear algebra algorithms

The Lanczos algorithm

The Wiedemann algorithm

Computing the linear generator in Wiedemann

Block algorithms

Plan

The Lanczos algorithm

The Wiedemann algorithm

Computing the linear generator in Wiedemann

Block algorithms

Lanczos

Here we assume $K = \mathbb{F}_\ell$, with ℓ large. "almost characteristic zero".

Lanczos requires a symmetric matrix so we consider $A = M^T M$.

Temporarily inhomogenous

The Lanczos algorithm is easier to state for an inhomogenous linear system, so let $b = Az$ for some random $z \in K^N$. We will solve

$$Av = b$$

from which we will have $A(v - z) = 0$.

A few definitions

Def. Let $y \in K^N$. Krylov subspace $\mathcal{K}_{A,y} = \left\langle y, Ay, \ldots, A^i y, \ldots \right\rangle$.

- $\dim \mathcal{K}_{A,y} \leq N$.
- $\mathcal{K}_{A,y}$ has a known basis.

Def. (pseudo-) scalar product associated to A: $(u, v) \stackrel{\text{def}}{=} u^T A v$.

Note: over a finite field, there are isotropic vectors.

Gram-Schmidt orthogonalization process:

- build an orthogonal basis from an arbitrary one.
- defined in characteristic zero for a real scalar product, but let's see.

GSO in positive characteristic

We take the method for its merits.

- It builds a sequence of vectors with $(e_i, e_j) = 0$ if $i \neq j$.
- We believe for a moment that nothing fails.
- We'll see what might fail and why.

Apply GSO to the basis $(A^i b)_i$ of $\mathcal{K}_{A,b}$. Denote $S_i = \langle b, \ldots, A^i b \rangle$.

$$e_0 \leftarrow b,$$

$$e_j \leftarrow A^j b - \sum_{i<j} \frac{(A^j b, e_i)}{(e_i, e_i)} e_i = A^j b - \sum_{i<j} \frac{b^T A^{j+1} e_i}{e_i^T A e_i} e_i.$$

Prop. $(e_i, e_j) = 0$ if $i \neq j$.
Note that $\langle e_0, \ldots, e_i \rangle = S_i$. Optimization: replace $A^j b$ by $A e_{j-1}$.

Lanczos (cont'd)

$$e_j \leftarrow Ae_{j-1} - \sum_{i<j} \frac{(Ae_{j-1}, e_i)}{(e_i, e_i)} e_i = Ae_{j-1} - \sum_{i<j} \frac{e_{j-1}^T A^2 e_i}{e_i^T A e_i} e_i,$$

Note that
$i < j-2 \Rightarrow Ae_i \in S_{j-2} \subset e_{j-1}^{\perp} \Rightarrow (Ae_{j-1}, e_i) = (e_{j-1}, Ae_i) = 0.$

$$\begin{aligned} e_j &\leftarrow Ae_{j-1} - \frac{(Ae_{j-1}, e_{j-1})}{(e_{j-1}, e_{j-1})} e_{j-1} - \frac{(Ae_{j-1}, e_{j-2})}{(e_{j-2}, e_{j-2})} e_{j-2}, \\ &\leftarrow Ae_{j-1} - \frac{e_{j-1}^T A^2 e_{j-1}}{e_{j-1}^T A e_{j-1}} e_{j-1} - \frac{e_{j-1}^T A^2 e_{j-2}}{e_{j-2}^T A e_{j-2}} e_{j-2} \end{aligned}$$

Algorithm. compute this, maintaining $O(1)$ vectors.

What do we have to do ? Examine failure cases.

Lanczos over $\mathbb{F}_\ell$: failure cases

Two possible reasons for stopping:

- We may reach an isotropic (a.k.a. self-orthogonal) vector: $(e_i, e_i) = 0$.
  - We have $(e_i, e_i) = e_i^T A e_i = (Me_i)^T Me_i = 0$.
  - Me_i might be isotropic for the "standard" bilinear form, but heuristically Prob $\approx \frac{1}{\ell}$ only.
- Eventually, we reach $e_i = 0$ at the end. This means success.
  - This implies that $\langle e_0, \ldots, e_{i-1} \rangle = \langle b, Ae_0, \ldots, Ae_{i-1} \rangle$.
  - Let z be a solution to $Az = b$ (z is not known). Let $w = \sum_{j<i} \frac{(e_j, z)}{(e_j, e_j)} e_j = \sum_{j<i} \frac{e_j^T b}{e_j^T A e_j} e_j$.
  - By construction, $\forall j,\ (e_j, w - z) = 0$.
    Thus $w - z \in \operatorname{Ker} M$ (and $Aw = b$) with proba $\approx 1 - \frac{1}{\ell}$.
  - If we started with $b = Az$ (z known), this gives $w - z \in \operatorname{Ker} M$.

Lanczos: remarks

Note: As is, the Lanczos algorithm does not work over $\mathbb{F}_2$ because for $\ell = 2$, a failure probability of $\frac{1}{\ell}$ at each step is a lot.

Complexity:

- N products $A \times v$,
- hence $2N$ products M (or M^T) times v.

Important (mis-)features:

- Needs fast operations for both M^T and M.
- Must keep track of several vectors.

Plan

The Lanczos algorithm

The Wiedemann algorithm

Computing the linear generator in Wiedemann

Block algorithms

The Wiedemann algorithm

The Wiedemann algorithm for $Mv = 0$ over $\mathbb{F}_p$ is easy.

- Pick $x, y \in \mathbb{F}_\ell^N$ at random.
- Compute $a_i = x^T M^i y$. These are all scalars.
- Compute the generator F of this linear recurring sequence.
- $\hat{F}$ divides the minimal polynomial μ_M. Hope $X^\lambda \hat{F} = \mu_M$.
- We then have $M^\lambda \hat{F}(M) y = 0$. Which means $M^{\lambda-1} \hat{F}(M) y \in \operatorname{Ker} M$.

This is very accessible to proofs of success probabilities.

The Wiedemann algorithm: workflow

Implementation of the Wiedemann algorithm is fairly straightforward.

- Computation of the sequence of a_i.
- Computation of the linear generator F.
- Computation of the kernel vector.

The sequence of a_i

- $i \leftarrow 0$;
- $v \leftarrow y$;
- While $i < 2N$.
  - $a_i \leftarrow x^T v \in \mathbb{F}_\ell$.
  - $v \leftarrow Mv$;
  - $i \leftarrow i + 1$.
- return $(a_i)_i$, sequence of $2N$ elements of $\mathbb{F}_\ell$.

Cost

To compute $2N$ terms, we need:

- Exactly $2N$ matrix-times-vector products.
- If the weight of M is W, this means $\approx 2N \times W$ operations.
  here, operation $=$ addition in $\mathbb{F}_\ell$.

The linear generator

Linear generator

The linear generator of the sequence is such that:

$$\forall i \geq d, F_0 a_i + F_1 a_{i-1} + \cdots + F_d a_{i-d} = 0.$$

Note. The set of polynomials $\sum_{i=0}^{d} F_i X^i$ is an ideal of $\mathbb{F}_\ell[X]$, and $\hat{\mu_M}$ belongs to it. So $d \leq N$.

The linear generator

Another point of view

Let $A(X) = \sum_{i \leq 2N} a_i X^i$, then:

$$A(X)F(X) = (\text{terms of deg} < N) + (\text{terms of deg} \geq 2N).$$

By construction, there is an infinite precision solution to $(\sum a_i X^i)F(X) = G(X)$, and looking at precision $2N$ will be sufficient to find it.

Several possible restatements ($\deg F \leq N$ and $\deg G < N$):

- $A(X)F(X) - X^{2N}R(X) = G(X)$.
- $A(X) = \dfrac{G(X)}{F(X)} + O(X^{2N})$.
- $A(X)F(X) = G(X) + O(X^{2N})$.

$O(X^i)$ means X^i times any polynomial in $\mathbb{F}_\ell[X]$

Computing the linear generator

Various algorithms can be used to compute F.

- The Berlekamp-Massey algorithm (from coding theory).
- The Euclidean algorithm!

We have several ways to do this in time $O(N^2)$ or even $O(N \log^2 N)$. More on this later.

Probabilistic aspect

We hope that we'll find a generator F which is such that $X^\lambda \hat{F} = \mu_M$. with $\lambda \geq 1$.

Reconstructing the solution

To compute $\hat{F}(M)y$, the process is similar to the first phase:

- $k \leftarrow 0$;
- $v \leftarrow y$;
- $w \leftarrow 0$;
- While $k \leq \deg F$;
  - $w \leftarrow w + v \times$ (coefficient of degree k in $\hat{F}(X)$);
  - $v \leftarrow Mv$;
  - $k \leftarrow k + 1$.
- return w.

Cost

N matrix-times-vector products.

Wiedemann: summary

The Wiedemann algorithm costs about $3N$ matrix-times-vector products.

Probability of failure is $O(1/\ell)$.
(main failure case: $\nu_X(\mu_M) = 1$, $\dim \operatorname{Ker} M = 1$, and $y \in \operatorname{Im} M$).

Comparison with the Lanczos method

The Wiedemann algorithm:

- costs $3N$ matrix-times-vector products.
- has a three-stage workflow which is a little bit more complicated than the Lanczos algorithm.

The Lanczos algorithm (not described):

- costs only $2N$ matrix-times-vector products.
- is comparatively slightly simpler.

Neither is really usable over $\mathbb{F}_2$.

Plan

The Lanczos algorithm

The Wiedemann algorithm

Computing the linear generator in Wiedemann

Block algorithms

Linear generator

The problem of computing the linear generator is central in the Wiedemann algorithm.

Next few slides: a brief review of how we can do in quasi-linear time, with a view towards a possible generalization.

Problem

Problem statement

Given $A \in \mathbb{F}_\ell[X]$ with $\deg A < 2N$, find $F, G \in \mathbb{F}_\ell[X]$ such that:

- $\deg F \leq N$ and $\deg G < N$. IOW, $\max(\deg F, 1 + \deg G) \leq N$.
- $A(X)F(X) = G(X) + O(X^{2N})$.

We may look at the linear algebra point of view.

- Degrees of freedom: $N + 1$ (coefficients of F).
- Constraints: N (coefficients of degree N to $2N - 1$).

But of course we can do much better than $O(N^3)$ here!

Fixed versus infinite precision

The series $A(X)$ is a truncation (to degree $2N$) of the series $\sum a_i X^i$.

By construction, $(a_i)_i$ is linearly generated with a generator of degree at most N.

The Berlekamp-Massey algorithm finds this generator $F(X)$.
If we ever attempt to compute $A(X)F(X)$ with more terms of the series $A(X)$, we will see that the trailing terms are zero!

Berlekamp-Massey vs Euclid

While we often look at the problem with high degrees first (Euclid), the Berlekamp-Massey presentation (low degrees first) generalizes much better.

Berlekamp-Massey point of view

- Form solutions to $A(X)F(X) = G(X) + O(X^t)$, for increasing values of t (starting with $t = 1$).
- We work with two candidates at a time.
  $F(X)$ and $G(X)$ are extended to matrices.
- The value $t = 2N$ is the target of this process.
- Do so in a way that $\max(\deg F, 1 + \deg G)$ does not grow too fast (not as fast as t).

Example

Let $N = 4, \ell = 17$, and $A = 2 + 5X + 3X^2 + X^3 + \cdots$.

We work with two candidates (two series in $\mathbb{F}_\ell[[X]]$).

$$\begin{pmatrix} 1 \\ X \end{pmatrix} \cdot A = \begin{pmatrix} 2 \\ 0 \end{pmatrix} + \begin{pmatrix} 5 + 3X + X^2 + \cdots \\ 2 + 5X + 3X^2 + \cdots \end{pmatrix} \cdot X$$

$$\begin{pmatrix} 1 \\ X + 3 \end{pmatrix} \cdot A = \begin{pmatrix} 2 \\ 6 \end{pmatrix} + \begin{pmatrix} 5 + 3X + X^2 + \cdots \\ 0 - 3X + 6X^2 \cdots \end{pmatrix} \cdot X$$

$$\begin{pmatrix} X \\ 3 + X \end{pmatrix} \cdot A = \begin{pmatrix} 2X \\ 6 \end{pmatrix} + \begin{pmatrix} 5 + 3X + \cdots \\ -3 + 6X + \cdots \end{pmatrix} \cdot X^2$$

$$\begin{pmatrix} 5 - 3X \\ 3X + X^2 \end{pmatrix} \cdot A = \begin{pmatrix} 2X - 7 \\ 6X \end{pmatrix} + \begin{pmatrix} -4 + \cdots \\ -3 + \cdots \end{pmatrix} \cdot X^3.$$

Berlekamp-Massey

At each step, we decide on the linear combination to use based on the degree t coefficients on the right-hand side.

- Which row we add to the other depends on which is smallest with respect to $\max(\deg F, 1 + \deg G)$.
- This smallest row is eventually multiplied by X, while the degree of the other is unchanged.
- On average $\max(\deg F, 1 + \deg G)$ grows like $t/2$.
- Complexity: N steps, $O(N)$ at each step, so $O(N^2)$.

Berlekamp-Massey

Key aspects

The computation involves matrices of polynomials.
The control flow is directed by the knowledge of:

- the knowledge of $\max(\deg F, 1 + \deg G)$ for each candidate.
- the error matrix $E(X) = (A(X)F(X) - G(X)) \operatorname{div} X^t$

The output is a matrix of polynomials $\pi(X)$ that encodes the necessary transformations to move from the pair of solutions (F, G) at $t = 1$ to the pair of solutions at some larger value of t.

Berkekamp-Massey, recursively

- Compute the initial error matrix $E(X)$.
- Truncate $E(X)$ to degree N (=half of $2N$).
- Recurse and find a matrix such that $\pi(X)E(X) = O(X^N)$. keep track of $\max(\deg F, 1 + \deg G)$ for each candidate.
- Multiply $\pi(X)$ by the full $E(X)$, get coefficients of degrees N to $2N - 1$. (middle product)
- Recurse and find a second matrix $\pi'(X)$.
- Compute $\pi'(X) \cdot \pi(X) \cdot \begin{pmatrix} 1 \\ X \end{pmatrix}$. (polynomial product)

Benefit: complexity is driven by large polynomial multiplications, doable in quasi-linear time.
The complexity of the linear generator step becomes $\widetilde{O}(N)$.

Plan

The Lanczos algorithm

The Wiedemann algorithm

Computing the linear generator in Wiedemann

Block algorithms

Block algorithms

Two popular block algorithms, with block size n:

- Block Lanczos (BL). $\frac{2N}{n-0.76}$ black box applications (for $\ell = 2$);
- Block Wiedemann (BW). In its simplest form: $\frac{3N}{n}$.

There are, however,

- multiple aspects beyond just this computational cost
- and multiple ways to parameterize BW, which end up modifying the picture a lot.

Montgomery's block Lanczos algorithm

BL (Montgomery) is a terrible mess, notationally speaking.

Key idea:

- Try to "orthogonalize" a sequence of subspaces of dim $= n$.
- When ℓ is small, the dimension of our subspaces may decrease in the process. (whenever we hope to find n new vectors, we find only $n - 0.76$ on average when $\ell = 2$.)

Problem with BL

- The procedure we have given does build a nice sequence of spaces, until it collapses.
- rank(W_i) decreases slowly to 0.

$$V_0 \longrightarrow \begin{array}{l} \mathcal{W}_0, \text{ dimension } n_0 \leq n \\ n - n_0 \text{ vectors dropped} \end{array}$$

$$V_1 = AW_0 \longrightarrow \begin{array}{l} \mathcal{W}_1, \text{ dimension } n_1 \leq n_0 \\ n_0 - n_1 \text{ vectors dropped} \end{array}$$

$$V_2 = AW_1 \longrightarrow \begin{array}{l} \mathcal{W}_2, \text{ dimension } n_2 \leq n_1 \\ n_1 - n_2 \text{ vectors dropped} \end{array}$$

Problem with BL

- The procedure we have given does build a nice sequence of spaces, until it collapses.
- rank(W_i) decreases slowly to 0.

$$V_0 \longrightarrow \mathcal{W}_0, \text{ dimension } n_0 \leq n$$
$$V_0 \longrightarrow n - n_0 \text{ vectors dropped}$$

$$V_1 = AW_0 \longrightarrow \mathcal{W}_1, \text{ dimension } n_1 \leq n_0$$
$$V_1 = AW_0 \longrightarrow n_0 - n_1 \text{ vectors dropped}$$

$$V_2 = AW_1 \longrightarrow \mathcal{W}_2, \text{ dimension } n_2 \leq n_1$$
$$V_2 = AW_1 \longrightarrow n_1 - n_2 \text{ vectors dropped}$$

What makes BL work

Solution to the problem: reinject vectors from previous steps to make the thing work.

It is possible to obtain a recurrence equation with small depth, but presenting it is really painful.
$\Rightarrow$ I'm deliberately skipping details here.

Various presentations: Montgomery (1995), Montgomery & Elkenracht-Huizing (1996), T. (2017).

Limitations of the block Lanczos algorithm

The BL algorithm does not offer a huge lot of parameterization opportunities.

- If one wants to involve multiple cores and nodes, all have to participate in the same matrix-times-vector product at each iteration.
- The implementation must keep track of a significant number of vectors, and does dot products at each iteration.
- AFAIK, there is no known mechanism to quickly validate some intermediary checkpoint data.

CSE291-14: The Number Field Sieve

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

Emmanuel Thomé

February 22, 2022

Part 6d

The block Wiedemann algorithm

Coppersmith's block Wiedemann algorithm

Parallelization levels

Parallelization of the linear generator step

Plan

Coppersmith's block Wiedemann algorithm

Parallelization levels

Parallelization of the linear generator step

Block Wiedemann

BW is a direct translation of Wiedemann to using vector blocks.

Things to do:

- properly define the notion of linear generator.
- show that using vector blocks reduces the number of needed iterations.

The expected benefits versus Wiedemann are clear:

- Better use of arithmetic power of CPUs (block operations).
- Hopefully better success probability.

We may state it and use it either over $K = \mathbb{F}_2$, or $K = \mathbb{F}_\ell$.

This presentation: try to solve $M \times v = 0$.

BW: the blocking parameters

The Wiedemann algorithm had vectors x and y in $\mathbb{F}_\ell^N$.

Blocking parameters

Block Wiedemann chooses two parameters m and n.

- x becomes a block of m vectors: $x \in \mathbb{F}_\ell^{N \times m}$.
- y becomes a block of n vectors: $y \in \mathbb{F}_\ell^{N \times n}$.

BW workplan

Let m and n be the blocking parameters.

- Initial setup. Choose starting blocks of vectors x and y.
- Sequence computation. Want L first terms of the sequence:

$$a_i = x^T M^k y \ (a_i \text{ are } m \times n \text{ matrices !}).$$

  The length L will be given by the analysis.
- Compute some sort of linear generator.
- Build solution as:

$$v = \sum_{k=0}^{\deg f} M^k y f_k.$$

  coefficients f_k here are $n \times r$ matrices, so that can combine things together.

Plan

Coppersmith's block Wiedemann algorithm

- The sequence step: `krylov`
- The linear generator step: `lingen`
- The solution step in Block Wiedemann: `mksol`

BW: the sequence step

- $i \leftarrow 0$;
- $v \leftarrow y$, a block of n vectors;
- While $i < L$, where L is the length:
  - $a_i \leftarrow x^T v$, an $m \times n$ matrix;
  - $v \leftarrow Mv$;
  - $i \leftarrow i + 1$.
- return $(a_i)_i$, sequence of L matrices of size $m \times n$.

For example, a straightforward case with bits: our black box deals with (say) 64-bit machine words. BW with $n = 64$ is as here.

In Cado-NFS, this sequence step is done by the `krylov` program.

BW: sub-sequences

What if we have $n = 192$, and our black box still does only 64-bit?

Split y into three pieces

$y_{0\ldots63}$, $y_{64\ldots127}$, $y_{128\ldots191}$

- $i \leftarrow 0$;
- $v \leftarrow y_{0\ldots63}$, a block of 64 vectors;
- While $i < L$, where L is the length:
  - $a_i \leftarrow x^T v$, an $m \times 64$ matrix;
  - $v \leftarrow Mv$;
  - $i \leftarrow i + 1$.
- return $(a_i)_i$, sequence of L matrices of size $m \times 64$.

- $i \leftarrow 0$;
- $v \leftarrow y_{64\ldots127}$, a block of 64 vectors;
- While $i < L$, where L is the length:
  - $a_i \leftarrow x^T v$, an $m \times 64$ matrix;
  - $v \leftarrow Mv$;
  - $i \leftarrow i + 1$.
- return $(a_i)_i$, sequence of L matrices of size $m \times 64$.

- $i \leftarrow 0$;
- $v \leftarrow y_{128\ldots191}$, a block of 64 vectors;
- While $i < L$, where L is the length:
  - $a_i \leftarrow x^T v$, an $m \times 64$ matrix;
  - $v \leftarrow Mv$;
  - $i \leftarrow i + 1$.
- return $(a_i)_i$, sequence of L matrices of size $m \times 64$.

Collect the different pieces of matrices a_i into $(a_i)_i$, sequence of L matrices of size $m \times n$.

Sub-sequences

Sub-sequences

In BW, the processing with the various sub-sequences is completely independent.
We define sub-sequences that match the optimal block width.

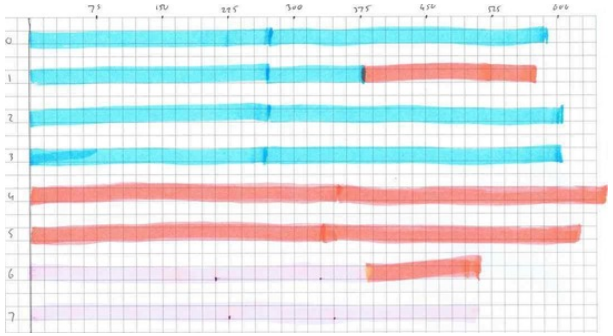
- over $\mathbb{F}_2$, we may for example define sub-sequences of width 64.
- over $\mathbb{F}_\ell$, we will probably define sub-sequences of width 1. This will STILL be block Wiedemann because we have matrices $(a_i)_i$ to handle, yet our black box will not really do blocks by itself.

A sub-sequence is identified by which range of columns of y it processes. Its output is the same range of columns for all the final matrices $(a_i)_i$.

Sub-sequences

Example: RSA-768 — we had $n = 512 = 8 \times 64$

- A contributor in Japan had a slow network and preferred to use two sub-sequences at the same time in interleaving fashion.
- Other 6 sub-sequences were processed independently in France and Switzerland.
- By periodically saving iterates $M^i y$ (say when $1000 \mid i$), we have a trivial chekpoint/restart feature.
- We actually exchanged sub-sequences to adapt to the various processing speeds.



More recent examples

In the 2016 kilobit hidden-SNFS-DLP computation, we had $n = 12$.

- Each black box deals with one product at a time.
- 12 independent sequences, 6 on each side of the ocean.
- Progress leveling every now and then (by hand).

More of the same with the records in 2019-2020, which all used multiple sequences.

BW: the length

The length of the sequence step is:

$$L = N/m + N/n + O(n/m + m/n).$$

The $O()$ term is less than 1000 for all practical ranges.
In practice we always have $\max(n/m, m/n) \ll N$.

Number of matrix-times-vector products

Whether or not we split into sub-sequences, the L steps of the sequence computation are performing:

$$n \times L = N \cdot (1 + n/m + o(1))$$

matrix-times vector products.
This is better than the $2N$ we had with the Wiedemann algorithm.

Plan

Coppersmith's block Wiedemann algorithm

- The sequence step: `krylov`
- The linear generator step: `lingen`
- The solution step in Block Wiedemann: `mksol`

The name of the game

Out of the sequence computation step (`krylov`), we have:

$$A(X) \in \mathbb{F}_\ell[X]^{m\times n},\ \deg A = \frac{N}{m} + \frac{N}{n}.$$

Wanted: matrix linear generator

We search for $F(X) \in \mathbb{F}_\ell[X]^{n\times n}$ and $G(X) \in \mathbb{F}_\ell[X]^{m\times n}$ such that:

$$A(X)F(X) = G(X) + O(X^{N/m+N/n}),$$
$$\deg F, G \leq \frac{N}{n}.$$

This involves arithmetic with matrices of polynomials.

Lingen algorithms

Several algorithms, rediscovered multiple times.
Costs (with $m = n$, to make things simpler):

- Coppersmith, 1994. $O(nN^2)$.
- Beckermann-Labahn, 1994. $O(nN^2)$, but also fast version $O(n^2(n + \log N)N \log N)$. This is the most general setting.
- T. 2001. $O(n(n + \log N)N \log N)$.
- Giorgi, Lebreton, 2014. Current state of the art, + online behaviour.

[T. 2001] is used for large NFS computations.

The `lingen` computation has significant memory requirements.
(Proportional to the input size when m and n are constant.)

Linear generator: basic idea

Fairly similar to Berlekamp-Massey.

- Analyze what can be done in quadratic complexity.
- Then build a recursive version.

Linear generator: quadratic base case

Recall $A(X) \in \mathbb{F}_\ell[X]^{m \times n}$. Ultimate goal:

$$A(X)F(X) = G(X) + O(X^L).$$

- Work with $m + n$ candidates (in columns) at a time.
- Extend F and G to
  - $F(X) \in \mathbb{F}_\ell[X]^{n \times (m+n)}$, and
  - $G(X) \in \mathbb{F}_\ell[X]^{m \times (m+n)}$.
- Initial error matrix $E(X) = (A(X)F(X) - G(X)) \operatorname{div} X^{\text{something}}$. We have $E(X) \in \mathbb{F}_\ell[X]^{m \times (m+n)}$.

We want a transformation matrix $\pi(X) \in \mathbb{F}_\ell[X]^{(m+n) \times (m+n)}$ such that

$$E(X)\pi(X) \equiv 0 \mod X^t$$

Linear generator: quadratic base case

What does it take to get $E(X)\pi(X) \equiv 0 \mod X^t$?

- we need to find solutions to $m \times (m+n) \times t$ linear constraints.
- with $\deg \pi < d$, we have $(m+n) \times (m+n) \times d$ degrees of freedom.
- Therefore, we should be able to do it with $d \approx \frac{m}{m+n} t$.

Advancing by t steps (in time $O(t^2)$)

We find $\pi(X)$ such that $E(X)\pi(X) \equiv 0 \mod X^t$ by setting approximately $\frac{m}{m+n} t$ coefficients in each matrix entry in $\pi(X)$. This is completely doable with a sort of Gaussian elimination.

More precise complexity: dependence on m and n is subtle.

Linear generator: recursive

- Compute an initial "error matrix" $E(X)$.
- Truncate to degree $\lceil L/2 \rceil$.
- Recurse, find $\pi(X)$ such that $E(X) \times \pi(X) \equiv 0 \mod X^{\lceil L/2 \rceil}$.
- Middle product (full) $E(X) \times \pi(X) \div X^{\lceil L/2 \rceil}$.
- Recurse a second time.
- Multiply $\pi(X)\pi'(X)$.

Again, quasi-linear algorithms and so on. More on this later.

One or many solutions ?

The linear generator step works with $m + n$ candidates internally, but eventually finds n solutions.

This is exactly similar to Berlekamp-Massey working with 2 candidates internally, but finding one generator.

How do we tell generators from non-generators eventually? By observing the fact that all matching columns in the error vector are canceled all of a sudden.

Interesting part of the linear generator

The linear generator step really outputs $F(X) \in \mathbb{F}_\ell[X]^{n \times n}$.
Input length: $L \approx \frac{N}{m} + \frac{N}{n}$. Output length: $\frac{m}{m+n} L \approx \frac{N}{n}$.

More on the linear generator matrix

The linear generator matrix, from a computer algebra perspective, has many interesting properties.

- Its determinant is close to (the reciprocal of) χ_M.
- Its Smith normal form is very close to the Smith normal form of $M - XI_N$ (invariant factors).
- $F(X)$ is very much a useful computer algebra thing!

However, we will not need these fancy properties.

Plan

Coppersmith's block Wiedemann algorithm

- The sequence step: `krylov`
- The linear generator step: `lingen`
- The solution step in Block Wiedemann: `mksol`

Equation for one solution vector

The generator is a matrix in $\mathbb{F}_\ell[X]^{n\times n}$.

- Each column of this matrix will yield a solution of the linear generator problem.

$$(\sum_i a_i X^i)F(X) = G(X). \quad \text{(infinite precision!)}$$

  One column is made of n polynomials.
- There is mathematical ground to say that the set of columns of the generator matrix form a basis of the set of solutions (according to a $\mathbb{Z}[X]$-module structure that is not hard to introduce).

Next step: move from a solution of the linear generator problem to a solution of the homogenous linear problem that we try to solve.

Many coefficients

Assume $\deg F \leq N/n$, $\deg G < N/n$. Write matrix $F(X)$ as

$$F_{i,j}(X) = \sum_{k=0}^{N/n} f_{i,j,k} X^k.$$

Coefficients of degree $N/n + d$ in $A(X)F(X)$ are zero, $\forall d \geq 0$.

$$\forall d \geq 0 \quad [X^{N/n+d}](A(X)F(X)) = 0.$$

More precisely, if columns j of F and G have degrees $\delta_{j,F}$ and $\delta_{j,G}$, then coefficients of degree $\delta_j = \max(\delta_{j,F}, 1 + \delta_{j,G})$ and above in the j-th column of $A(X)F(X)$ are zero.

Fact: we have columns j for which $\delta_j > \delta_{j,F}$.

Many coefficients

Since $a_i = x^T M^i y$, we have (still for any j, and δ_j as above):

$$\forall d \geq 0 \quad [X^{\delta_j + d},\ \text{column } j](A(X)F(X)) = 0,$$

$$\sum_{k=0}^{\delta_{j,F}} a_{d+\delta_j-\delta_{j,F}+k}[X^{\delta_{j,F}-k},\ \text{column } j]F(X) = 0,$$

column i of y

$$x^T M^d \cdot M^{\delta_j - \delta_{j,F}} \underbrace{\sum_{k=0}^{\delta_{j,F}} M^k \sum_{i=0}^{n} y_i f_{i,j,\delta_{j,F}-k}}_{v_j} = 0.$$

- $M^{\delta_j - \delta_{j,F}} v_j$ is orthogonal to many vectors in $\mathbb{F}_\ell^N$.
  We can quantify the probability that it be zero: it is high.
- we may rearrange the expression so that v_j really looks like a combination of evaluations of polynomials at M (and y).

Equation for one solution vector

A combination of polynomial evaluations

$$v_j = \sum_{k=0}^{\delta_{j,F}} M^k \sum_{i=0}^{n} y_i f_{i,j,\delta_{j,F}-k}.$$

The equation of the solution is a bit like this:

$$v_j = \widehat{F_{0,j}}(M)y_0 + \widehat{F_{1,j}}(M)y_1 + \cdots + \widehat{F_{n-1,j}}(M)y_{n-1}.$$

where $(\widehat{F_{0,j}} \cdots \widehat{F_{n-1,j}}) = (X^{\delta_{j,F}} F_{i,j}(1/X))_i$.

This is:

- a bit like what we had with the Wiedemann algorithm
- except that we blend the different columns of the vector block y together.

mksol: procedures

$$v_j = \widehat{F_{0,j}}(M)y_0 + \widehat{F_{1,j}}(M)y_1 + \cdots + \widehat{F_{n-1,j}}(M)y_{n-1}.$$

This evaluation, called mksol, can be arranged in multiple ways.

- Compute all n solutions (w_0 to w_{n-1}) that are given by the n columns of F (not all will be linearly independent).
  Output would be a block of n vectors.
- Restrict to only r among n solutions. E.g. $r = 64$ or $r = 1$.
  Output would be a block of r vectors.
- Evaluate with a Horner scheme or not.

1st approach: `mksol`, n solutions, no Horner

- $k \leftarrow 0$;
- $v \leftarrow y$, a block of n vectors;
- $w \leftarrow 0$, a block of n solutions;
- While $k \leq \deg F$;
  - $\forall i,\ w \leftarrow w + v_i \times$ (coefficients of degree k in $\widehat{F_{i,0\cdots n-1}}(X)$);
  - $v \leftarrow Mv$; (block width is n)
  - $k \leftarrow k + 1$.
- return w.

1st approach: `mksol`, n solutions, no Horner

- $k \leftarrow 0$;
- $v \leftarrow y$, a block of n vectors;
- $w \leftarrow 0$, a block of n solutions;
- While $k \leq \deg F$;
  - $\forall i,\ w \leftarrow w + v_i \times$ (coefficients of degree k in $\widehat{F_{i,0\cdots n-1}}(X)$);
  - $v \leftarrow Mv$; (block width is n)
  - $k \leftarrow k + 1$.
- return w.

- Our black box deals with n vectors at a time (or, equivalently, we may split into sub-sequences).
- Note: we're reusing exactly the same vector iterates $M^i y$ as in the `krylov` step.
- This used to be the way I had always used BW until 2016.
- For $K = \mathbb{F}_2$ and $n = 64$, this is an entirely valid way to proceed. Not much else to do.

Benefits of using the same iterates

Since we use the same iterates $M^i y$ as in the `krylov` step, we can trade storage for more parallelism.

If we saved a few iterates $M^i y$ in the `krylov` (e.g. for $1000 \mid i$):

- As we already said, this provides checkpoint/restart for `krylov`.
- But this also allows us to compute the result of the `mksol` as the sum of many independent calculations.

k intermediary vectors saved $\leftrightarrow$ k-fold distribution for `mksol`.

2nd approach: fewer solutions, no Horner

- $k \leftarrow 0$;
- $v \leftarrow y$, a block of n vectors;
- $w \leftarrow 0$, a block of r vectors. Goal: solutions s to $s + r - 1$;
- While $k \leq \deg F$;
  - $\forall i,\ w \leftarrow w + v_i \times (\text{coefficient of degree } k \text{ in } \widehat{F_{i,s\cdots s+r-1}}(X))$;
  - $v \leftarrow Mv$; (block width is n)
  - $k \leftarrow k + 1$.
- return w.

- This saves a little bit on the vector multiplication part.
- We are still going through the same vector iterates.

`mksol` cost, no Horner

The degree of F is $\approx N/n$. Therefore the previous process does N/n application of the black box, of width n.

- In this setting, `mksol` costs N matrix-times-vector products.
- The total cost of `krylov+mksol` is now

$$(2 + n/m)N$$

  matrix-times-vector products.
- Better than non-block (if $m > n$), but still more expensive than (block) Lanczos.
- Increasing m and n only works to a certain extent, since the linear generator step becomes more expensive as $m + n$ grows.

3rd approach: `mksol`, n solutions, Horner

- $k \leftarrow \deg F$;
- $v \leftarrow y$, a block of n vectors;
- $w \leftarrow 0$, a block of n solutions;
- While $k \geq 0$, where L is the length:
  - $w \leftarrow Mw$ (block width is n);
  - $\forall i,\ w \leftarrow w + y_i \times$ (coefficients of degree k in $\widehat{F_{i,0\cdots n-1}}(X)$);
  - $k \leftarrow k - 1$.
- return w.

- We are no longer using the same iterates.
- However, we can still reuse $M^{1000}y$ in order to compute the contribution of the terms of degree 1000 to 1999 in the sum!

A piece of the Horner computation

$$\text{fragment of } v_j = \sum_{k=1000}^{1999} M^k \sum_{i=0}^{n} y_i f_{i,j,\delta_{j,F}-k},$$

This is exactly the same as a degree-999 evaluation of the same kind, with M^{1000} as a starting vector.

- This means that with Horner evaluation, we can still benefit from the checkpoints that we have saved in the Krylov space.
- However, our computation $w \leftarrow Mw$ is still operating on a block of n vectors.

4th approach: `mksol`, r solutions, Horner

- $k \leftarrow \deg F$;
- $v \leftarrow y$, a block of n vectors;
- $w \leftarrow 0$, a block of r vectors. Goal: solutions s to $s + r - 1$;
- While $k > 0$, where L is the length:
  - $w \leftarrow Mw$ (block width is now r here);
  - $\forall i,\ w \leftarrow w + y_i \times$ (coefficient of degree k in $\widehat{F_{i,s\cdots s+r-1}}(X)$);
  - $k \leftarrow k - 1$.
- return w.

We can do new things!

- $r = 1$ solution with only N/n matrix times vector products, with a block width of 1 (typical with large ℓ).
- or $r = 64$ solutions with
  - rN/n matrix times vector products,
  - or equivalently, N/n matrix times vector (block) products, with a block width of 64.

Improved cost

New cost: rN/n for `mksol` (for r solutions).

The total cost of `krylov+mksol` is now

$$(1 + n/m + r/n)N$$

matrix-times-vector products.

References: Kaltofen95, FGHT17.

New

In this setting, for N large enough and fixed r, we can choose parameters so that the cost of BW is

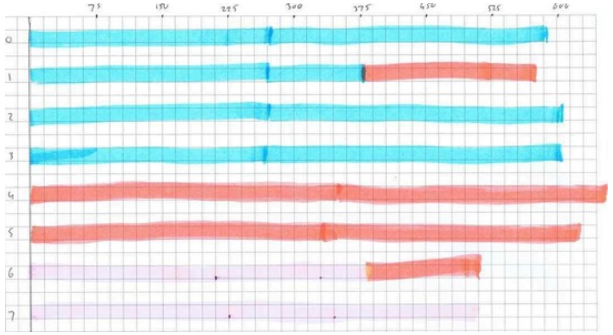
$$(1 + o(1))N$$

matrix-times-vector products.

Splitting the computation in pieces

In `krylov` we may periodically save the vectors $M^{k \times 1000} y$.

- This makes it possible to checkpoint and restart.
- Of course we cannot compute from iteration $k \times 1000$ until we have at least reached this iteration.



mksol checkpoints

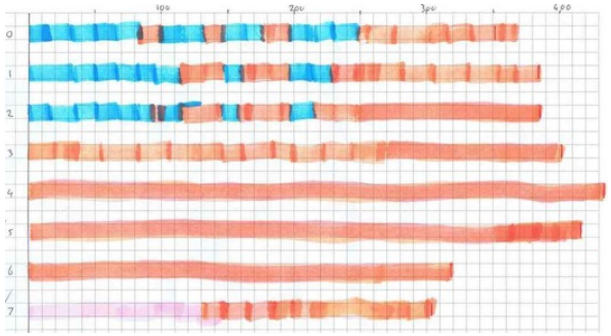
These same checkpoints can also be used:

- by mksol/no-horner, trivially;
- by mksol/horner also: we let $M^{k \times 1000} y$ play the role of y, and we compute a part of the final sum.

Of course the value interval=1000 can be adjusted:

- Smaller = more checkpoints, more disk, many independent tasks;
- Larger = fewer checkpoints, fewer (longer) tasks.

Note: all necessary checkpoints are already there when mksol starts! We can do everything in parallel if we want.



Example

In FGHT17, we had $N = 28.3 \times 10^6$ and $m = 24$, $n = 12$.

Total number of products: 44×10^6.

We could have made this lower but:

- we were not absolutely confident about whether the `lingen` step would go smoothly;
- this was our very first experiment with this strategy.

	sieving	linear algebra		
		sequence	generator	solution
cores	≈3000	2056	576	2056
CPU time (core)	240 years	123 years	13 years	9 years
calendar time	1 month	1 month		

Example

In BGGHTZ20, for DLP240, we had

$$N = 36 \times 10^6, \quad m = 48, \; n = 16.$$

Total number of products: 50×10^6.

	sieving	linear algebra		
		sequence	generator	solution
cores	$\geq$10000	3072	576	26880
CPU time (core)	2400 years	700 years	12 years	70 years
calendar time	6 months	3 months	62h	1 day

Example

In BGGHTZ20, for RSA240, we had

$$N = 282 \times 10^6, \quad m = 512 = 8 \times 64, \; n = 256 = 4 \times 64, \; r = 64.$$

Total number of products (block width 64): 7.7×10^6.

	sieving	linear algebra		
		sequence	generator	solution
cores	$\geq$10000	2048	512	2048
CPU time (core)	800 years	70 years	10 months	13 years
calendar time	2 months	37 days	13h	7 days

Note: linear algebra computation done in best-effort mode, calendar time is not really meaningful.

Guarding against errors

We can check the data on disk. It is useful because data on disk could be corrupted (disk errors, disk full, ...).

Simple idea:
- let C_0 be a random vector (or vector block);
- compute $C_{1000} = (M^T)^{1000} C_0$ (pre-compute);
- check that $C_{1000}^T (M^{k \times 1000} y) = C_0^T (M^{(k+1) \times 1000} y)$.
- we detect errors with good probability.

Caveat: C_0 must not have zero coefficients: it would limit our ability to detect errors.

Different steps

There are two ways to run the block Wiedemann algorithm.

BW has several steps, and Cado-NFS has several binaries.
Some steps are computational, some are mere bookkeeping.

Steps in BW

- Let m, n be... .....
- Let x, y be... .....
- Compute C_{1000}. .....
- Compute $A(X) = \sum_i {}^t x M^i y\, X^i$. .....
- Compute $F(X)$. .....
- Compute $\sum_i M^i y f_i$. piecewise,... then the sum

Steps in BWC

- command line
- `prep`
- `secure`
- `krylov`
- `lingen`
- `mksol`
- `gather`

Plan

Coppersmith's block Wiedemann algorithm

Parallelization levels

Parallelization of the linear generator step

Block algorithms

Both block algorithms we know of use a block black box.

That black box is able to deal with blocks of (say) n_1 vectors at the same time.

- When the base field is $\mathbb{F}_2$, we probably want to choose $n_1 = 64$, while for larger fields it is likely that $n_1 = 1$ is best.
- Per se, the black box rather offers a SIMD mode of operation (a.k.a. table soccer) rather than parallelism.
- Whenever we can do some n_1, it is trivial to emulate n_1 twice or three times larger (with a loop!)

The 1st level of "parallelism" is SIMD

SIMD level: optimal value for n_1

When matrices are sparse, most of the time in the matrix-times-vector operation comes from memory throughput rather than from CPU computation.

- Using SSE-2 (128-bit) types instead of 64-bit types might take a bit less than twice the time per iteration.
- But it is not even clear.
- Furthermore, doing too much SIMD can hamper parallelism at higher levels.

Thread-level / SMP

One core has the matrix data and multiplies it by a block of n_1 vectors

↓

n_2 cores each have $1/n_2$-th of the matrix data and collectively work to multiply it by a block of n_1 vectors

The 2nd level of parallelism is threads (intra-node, SMP)

Implicitly, the thread level can make nice use of shared memory.

- NUMA is something we have to pay attention to,
- our communication pattern must be well thought.

MPI-level / cluster

One node (n_2 cores) has the matrix data and multiplies it by a block of n_1 vectors

↓

n_3 nodes each have $1/n_3$-th of the matrix data and collectively work to multiply it by a block of n_1 vectors

The 3rd level of parallelism is MPI (inter-node)

The interconnect topology is important. Again, we must pay attention to our communication pattern.

Distribution

One cluster (n_3 nodes) has the matrix data and some init data, and is busy for time T

↓

n_4 clusters each have the matrix data and some init data, and are busy for time T/n_4

The 4th level of parallelism is the distribution level
Only the block Wiedemann algorithm can do this

Practically no communication between clusters, at this level (only dispatch & reconcile).

Three caveats

We must pay attention to three important things:

- Scaling, esp. at the MPI- (n_3) and thread- (n_2) levels, because communication costs are pure overhead.
- Global block size ($n_1 n_4$), and how it should not go out of control.
- Choice of n_1.

Scaling

For fixed n_1:

- we expect levels $2, 3, 4$ to bring time T to $T/(n_2 n_3 n_4)$;
- in practice it might not be so, esp. if n_2 and n_3 are large.

Answers:
- careful implementation and thread placement. CPU binding is particularly important.
- well-organized communication patterns.

Block size

Given our presentation with multiple levels, the block size that we see from the global algorithm point of view is $n = n_1 n_4$.

- Block Wiedemann `lingen` has some cost related to the block size, of the order of $\widetilde{O}(nN)$.
  We must really pay attention to it.
- Block Lanczos, too, has some additional costs that are proportional to n ($n = n_1$ for BL, since $n_4 = 1$).

Choosing n_1 properly

When $K = \mathbb{F}_2$, a black box iteration with $n_1 = 1$ or $n_1 = 8$ take the same time. The time is well sub-linear until some block size, and then super-linear.

Two examples on my laptop:

matrix	`rsa100`, 135krows 100 iterations	`c163`, 10Mrows 4 iterations
$n_1 = 8$	2.25	16.80
$n_1 = 16$	2.75	19.79
$n_1 = 32$	3.93	23.65
$n_1 = 64$	5.00	27.44
$n_1 = 128$	5.85	35.25
$n_1 = 256$	17.86	68.40

This is no definite truth, but it indicates that 128-bit looks like a sweet spot.

Choosing n_1 properly

Whatever the sweet spot, a large n_1 certainly forces us to reduce n_4 if we would like their product to remain bounded.

$\Rightarrow$ too much SIMD may actually be a nuisance.

Plan

Coppersmith's block Wiedemann algorithm

Parallelization levels

Parallelization of the linear generator step

FFT in the linear generator step

The main operations of the linear generator step in BW are

Multiplications of matrices of polynomials over finite fields.

We want to use asymptotically fast algorithms.

here!

- First approach: $c_{i,j} = \sum_k a_{i,k} \times b_{k,j}$
- Better complexity: use the fact that we are using FFT-based algorithms.
  - Compute all forward transforms $\widehat{a_{i,k}}$.
  - Compute all forward transforms $\widehat{b_{k,j}}$.
  - Compute all convolutions $\widehat{c_{i,j}} = \sum_k \widehat{a_{i,k}} * \widehat{b_{k,j}}$
  - Compute all inverse transforms $\widehat{\widehat{c_{i,j}}} = c_{i,j}$.
- Caveat: memory goes totally out of control.

linear

Memory cost of fast multiplication

How much memory do we need to multiply two integers of the same size?

Input size		Peak memory
bits	MB	MB
2^{23}	1	18
2^{24}	2	28
2^{25}	4	49
2^{26}	8	90
2^{27}	16	172
2^{28}	32	336
2^{29}	64	664
2^{30}	128	1320
	...	
2^{40}	128GB	1.3TB

One Fourier transform = about 5 times the input size!

Parallelization of the linear generator step

Two reasons to parallelize:

- Use more CPU power and get the result faster.
- Have more memory available.

This requires appropriate scheduling of the computation of the transforms.

Guiding principles:

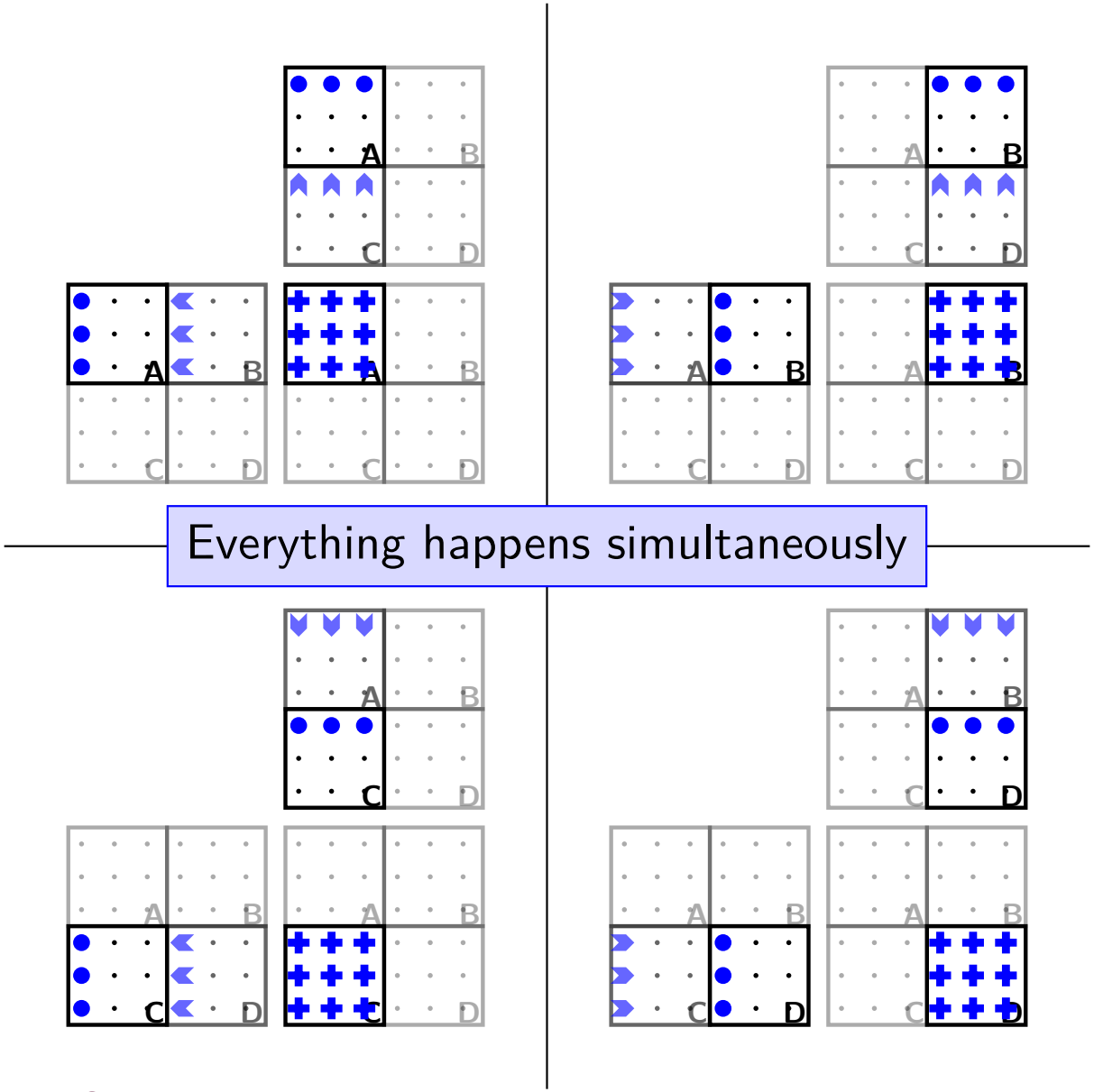
- limit the lifetime of transforms as much as we can.
- adapt the control flow when relevant.

Parallelization of the linear generator step

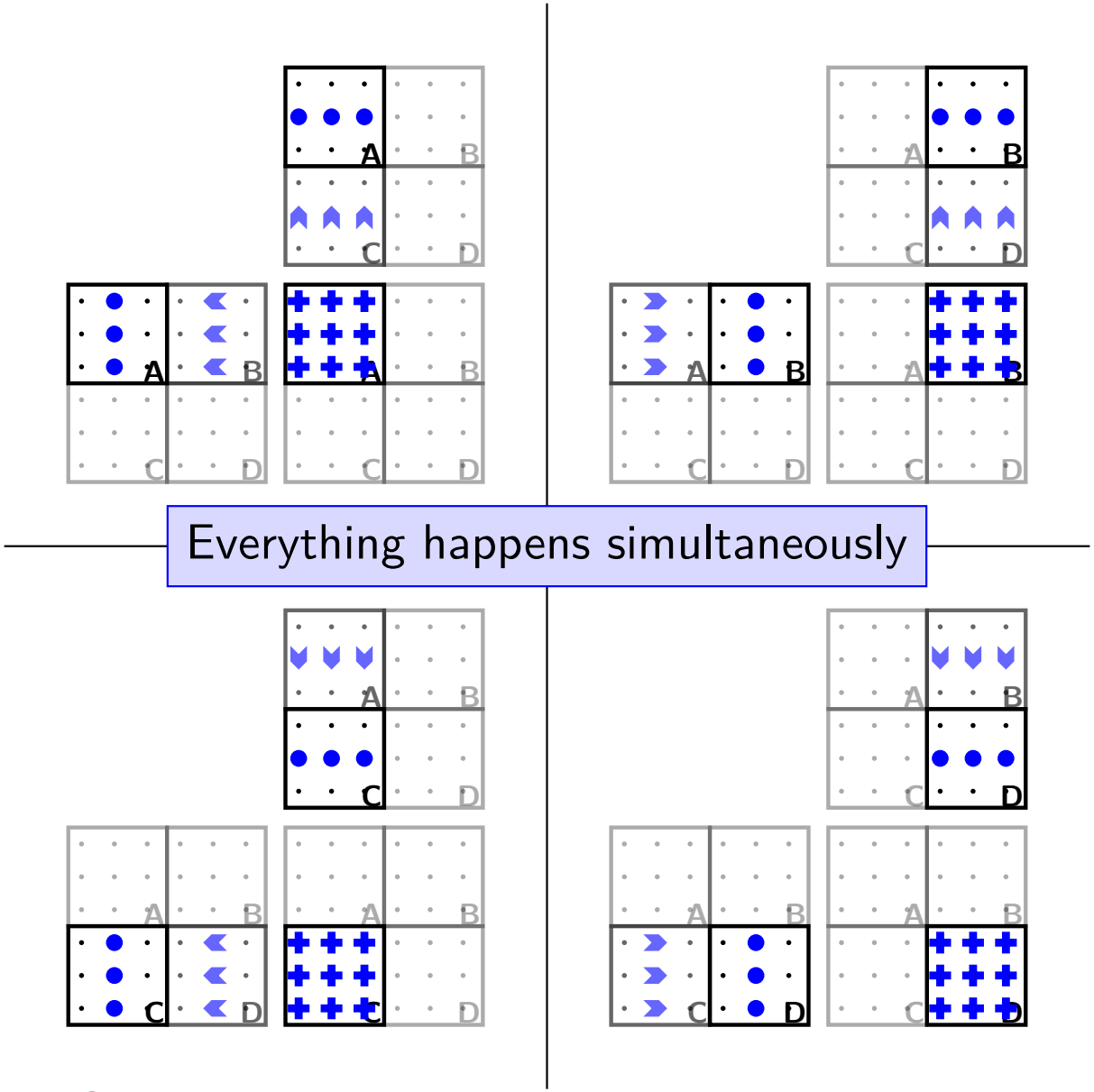
Typical context:

- r^2 nodes participate in a big matrix product of two $n \times n$ matrices. (n: dozens)
- Each "owns" a submatrix $\frac{n}{r} \times \frac{n}{r}$ of both inputs and the output.
- Simple case: each node is ok with allocating space for $\frac{n^2}{r^2}$ transforms, but not much more.

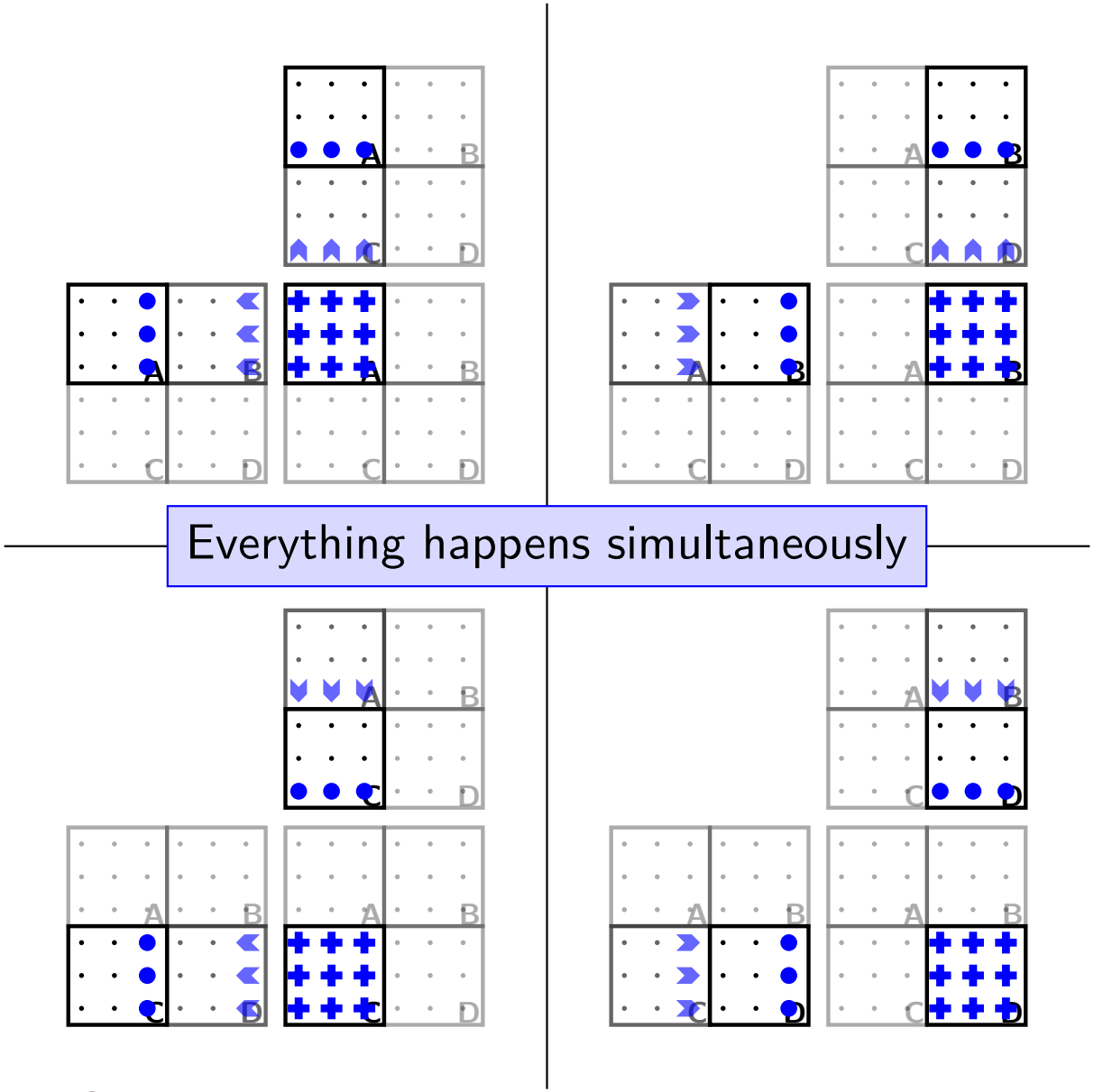
Parallelizing `lingen` carefully



Parallelizing `lingen` carefully



Parallelizing `lingen` carefully

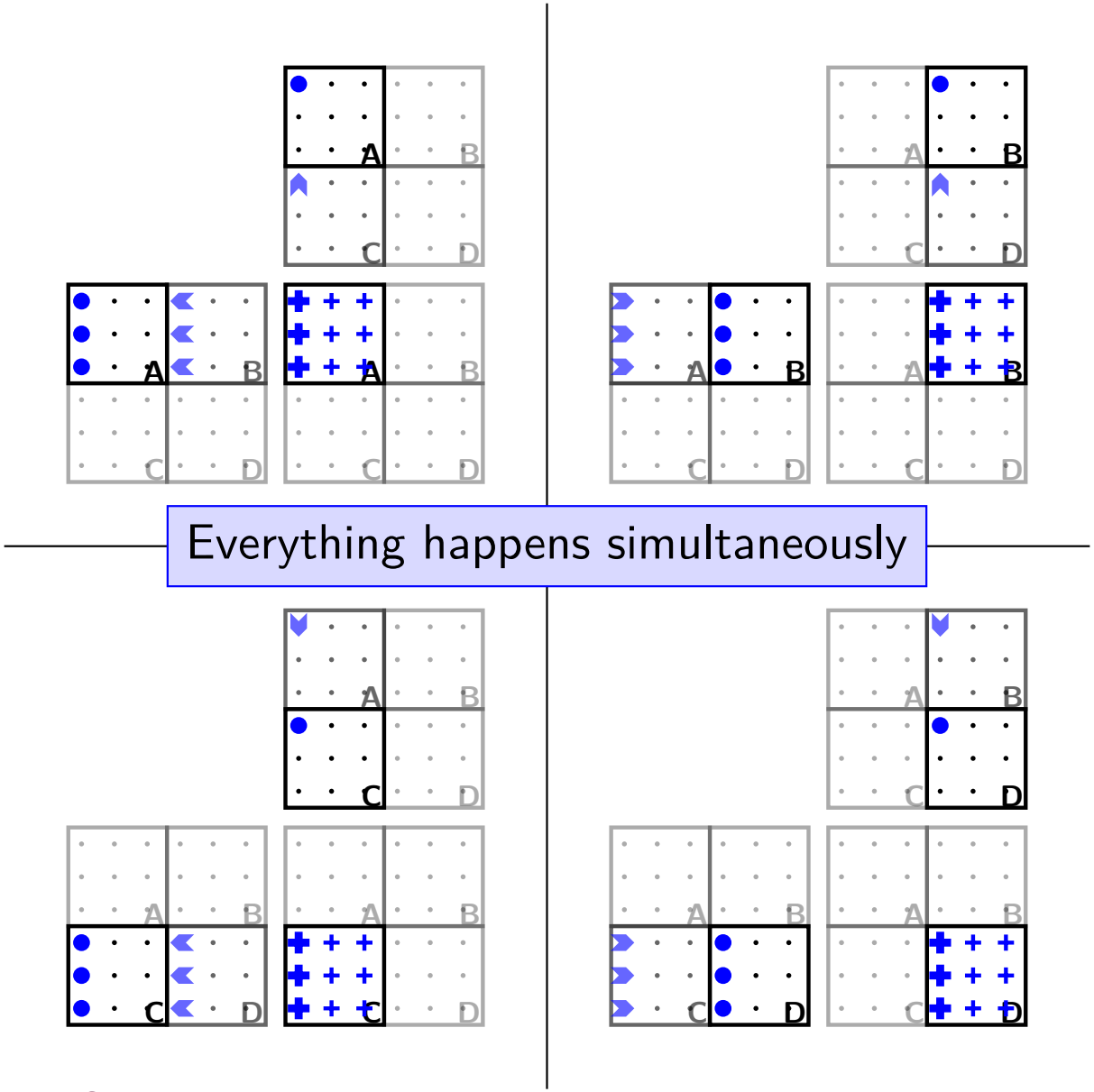


Memory cost

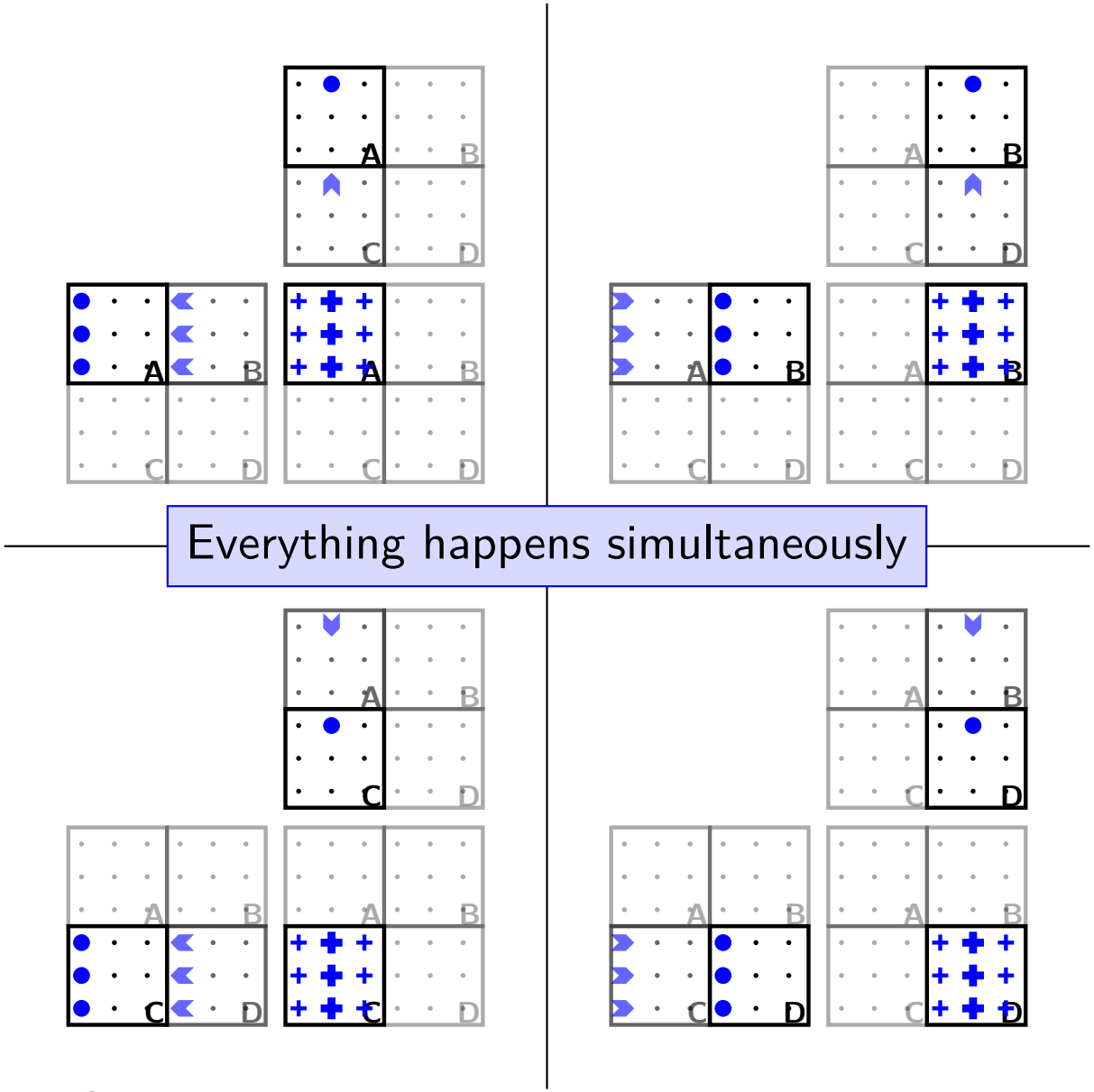
Each node here needs space for $\frac{n^2}{r^2}$ AND for $2(r-1)\frac{n}{r}$ transforms from other nodes.

This may be too much in certain cases.

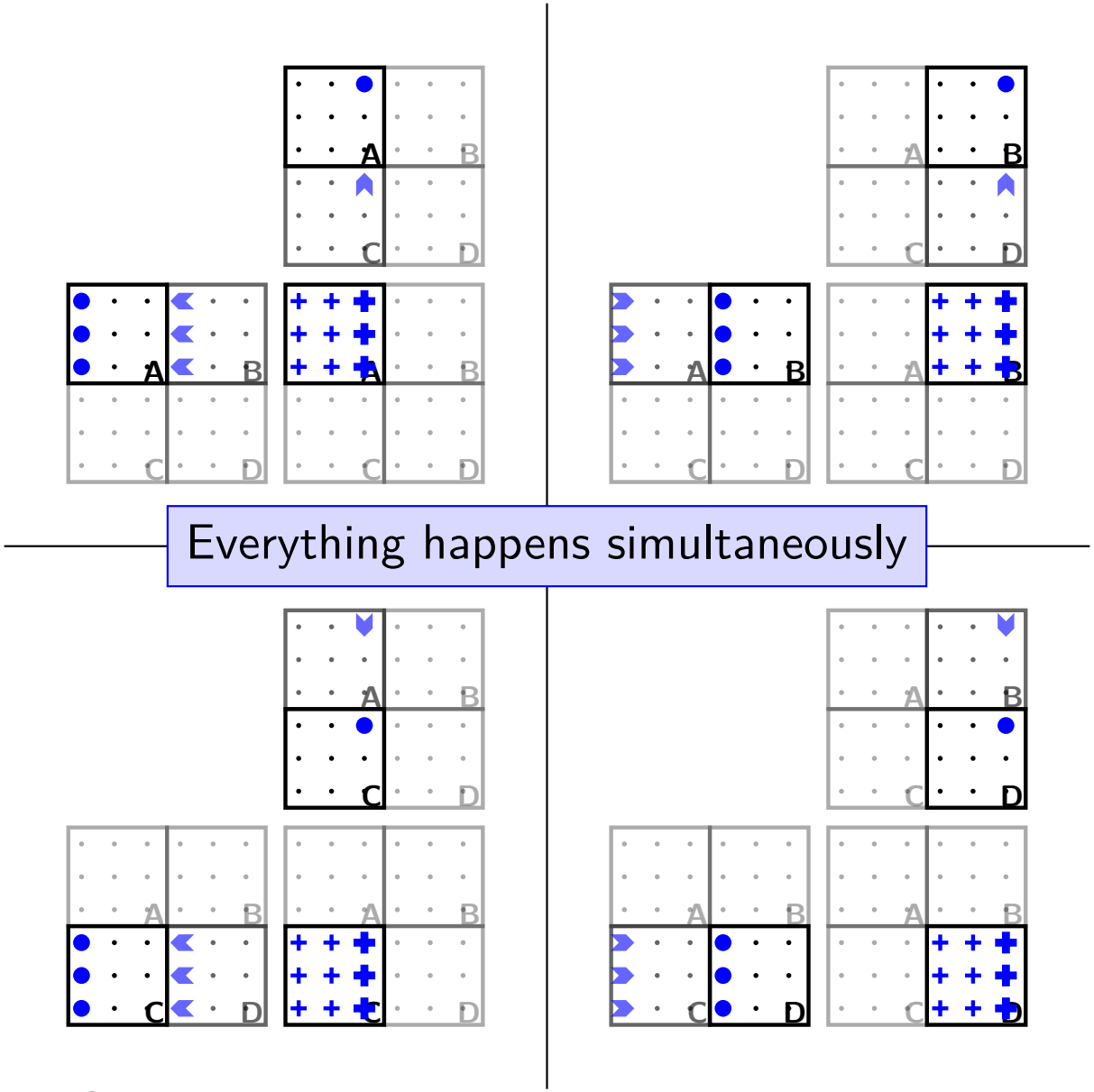
Parallelizing `lingen`: less memory



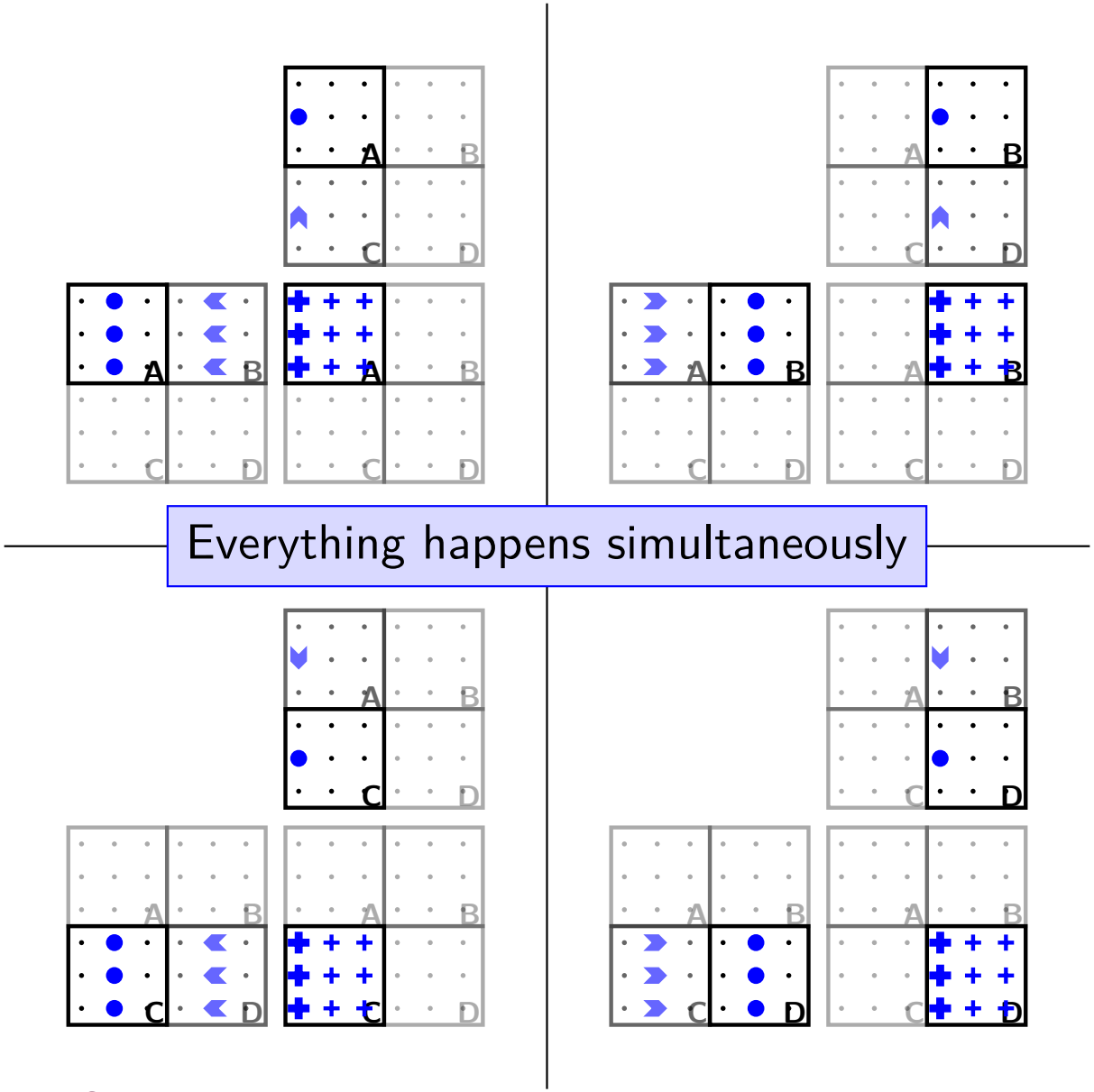
Parallelizing `lingen`: less memory



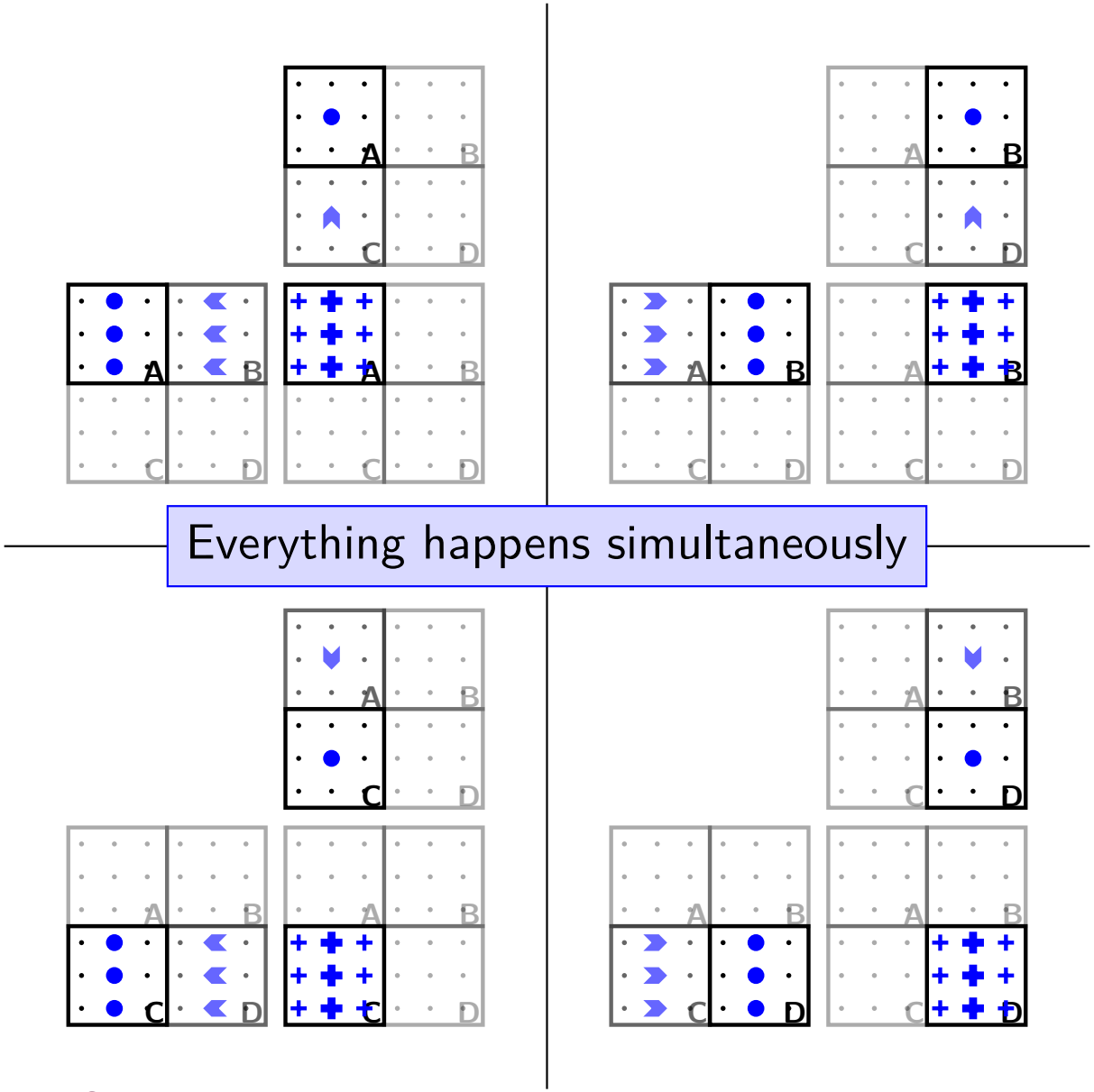
Parallelizing `lingen`: less memory



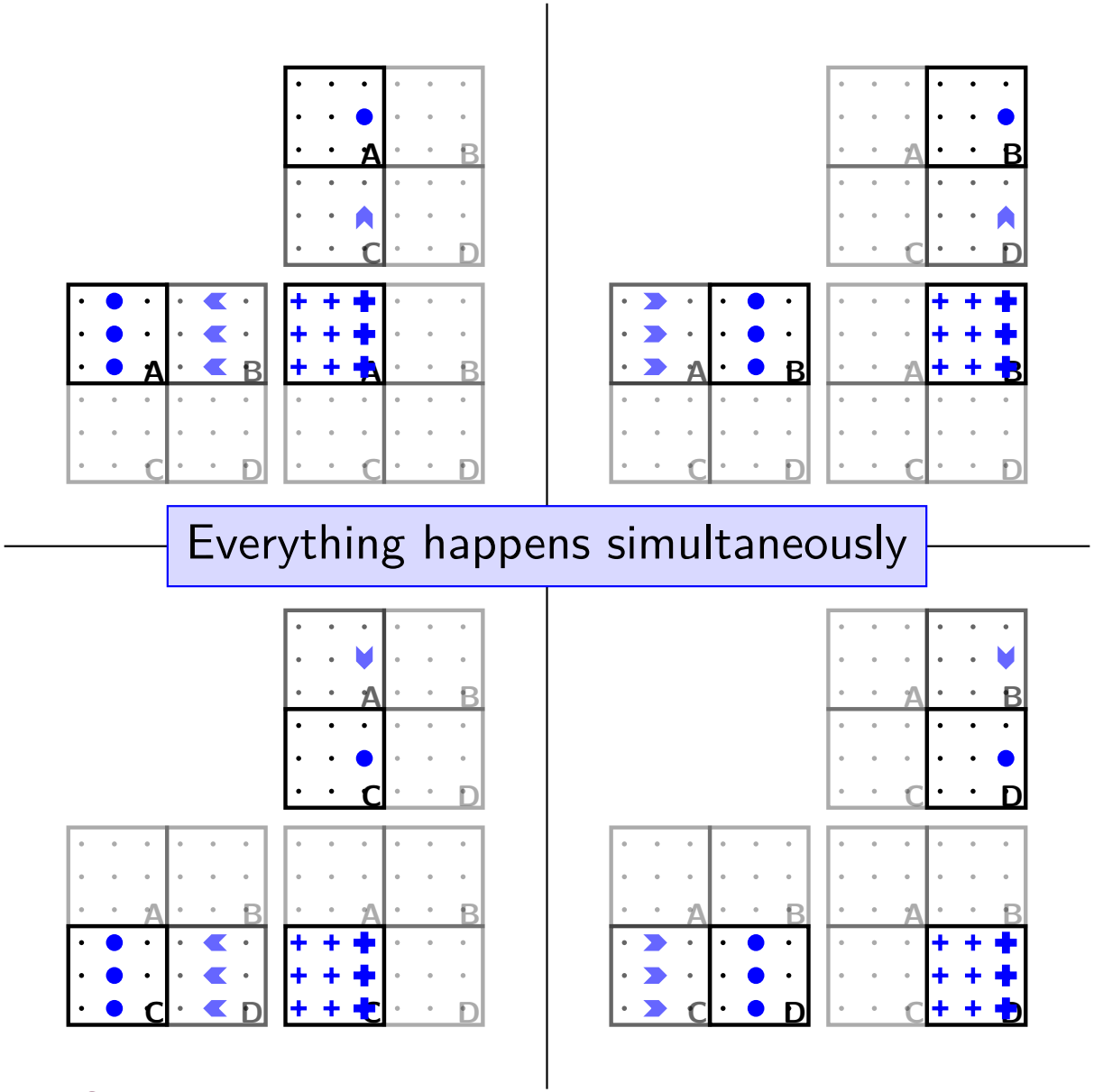
Parallelizing `lingen`: less memory



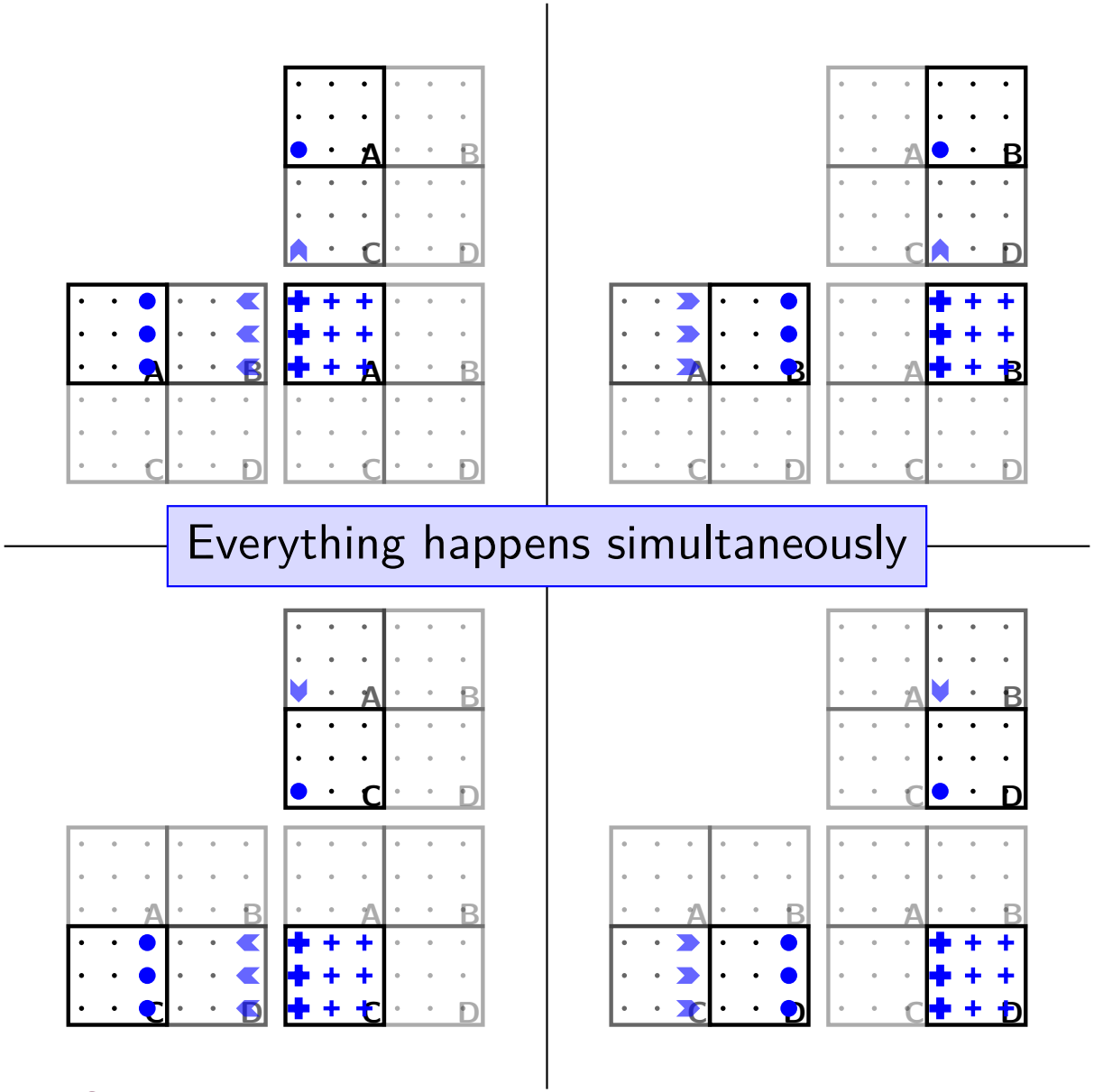
Parallelizing `lingen`: less memory



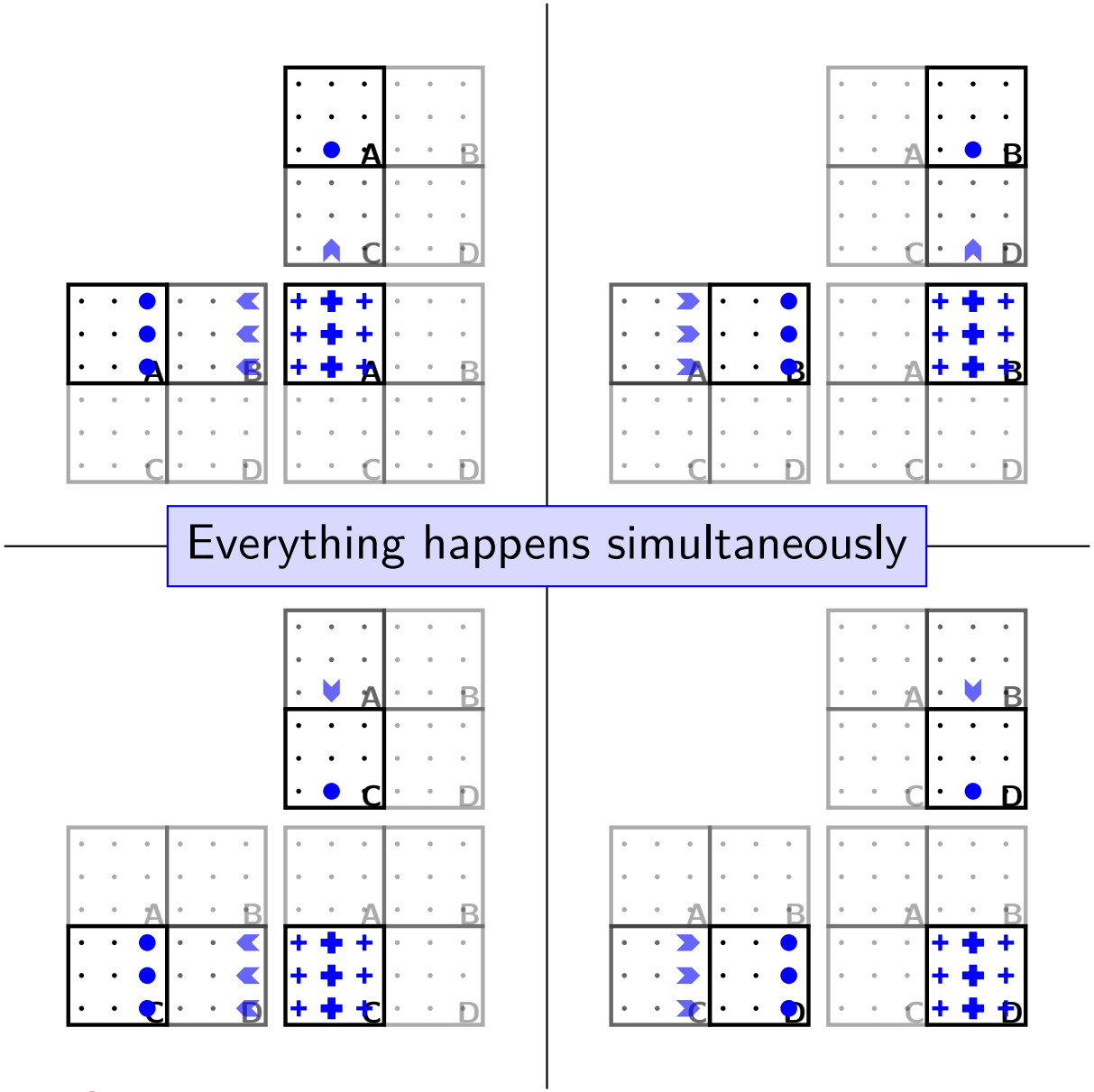
Parallelizing `lingen`: less memory



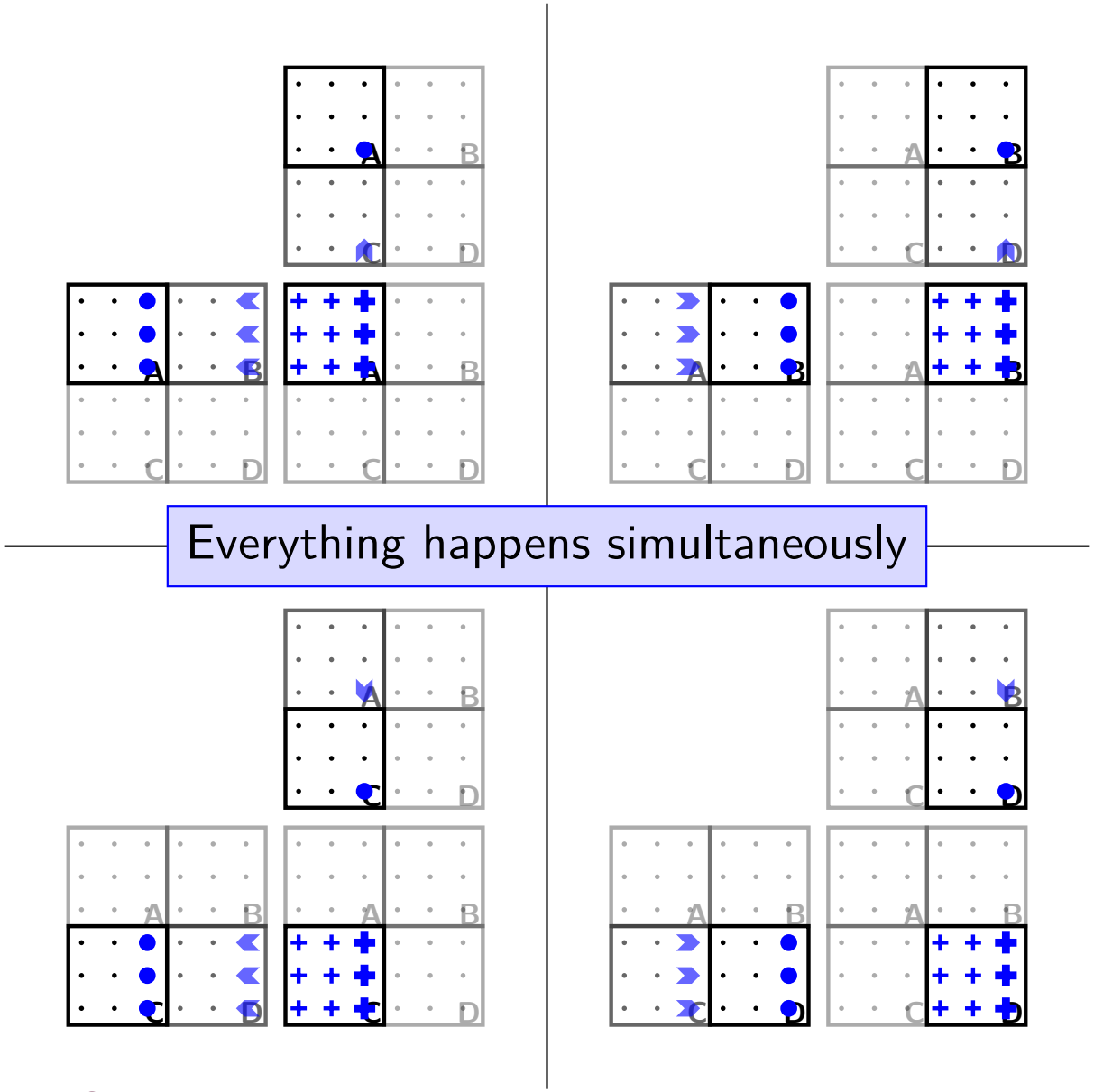
Parallelizing `lingen`: less memory



Parallelizing `lingen`: less memory



Parallelizing `lingen`: less memory

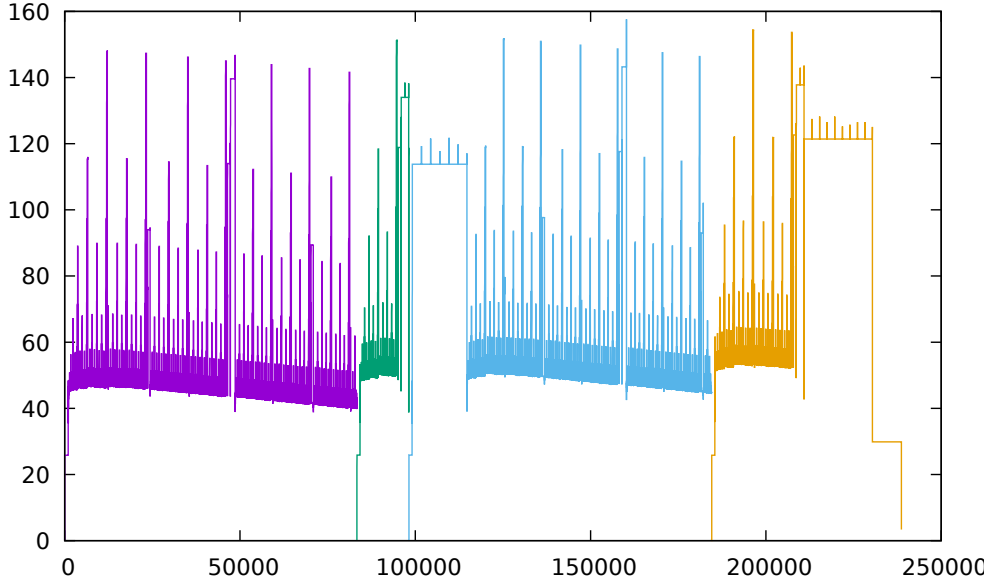


Memory cost

Each node here needs space for $\frac{n^2}{r^2} + (r-1)\frac{n}{r} + (r-1)$ transforms.

- This is achieved only by reorganizing the scheduling of computations and communications.
- Now this may still be too much in certain cases. Then we may want to split the computation even more, at the expense of recomputing several transforms.

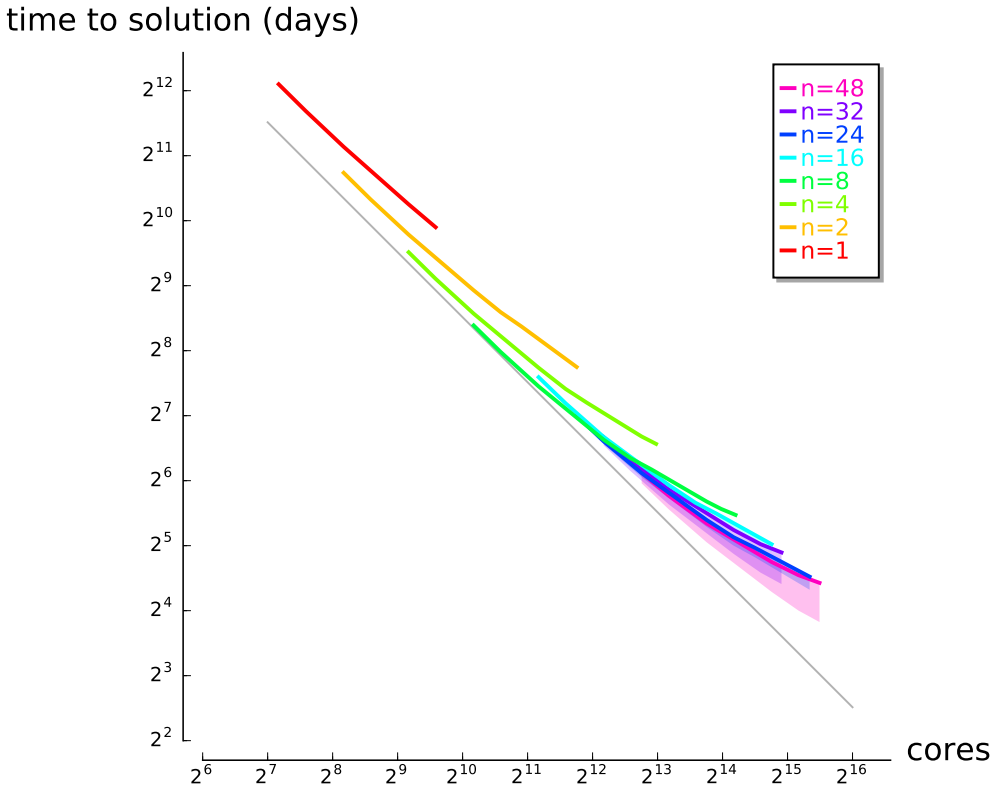
Keeping track of memory is important!



We can adjust the scheduling at each recursion depth.

Better memory usage → better scaling

We can predict the total runtime of BW quite well.



BW scales! (more than people tend to think).

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

Introduction

The lifting approach

Couveignes' algorithm

Another CRT method

Montgomery's algorithm

Conclusion

Plan

Introduction

The lifting approach

Couveignes' algorithm

Another CRT method

Montgomery's algorithm

Conclusion

Forcing squares

Recall where relations are coming from:

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

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 $S(r)$ mod 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

Introduction

The lifting approach

Couveignes' algorithm

Another CRT method

Montgomery's algorithm

Conclusion

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

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

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

Introduction

The lifting approach

Couveignes' algorithm

Another CRT method

Montgomery's algorithm

Conclusion

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 $T(m)$ mod 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

Introduction

The lifting approach

Couveignes' algorithm

Another CRT method

Montgomery's algorithm

Conclusion

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

Introduction

The lifting approach

Couveignes' algorithm

Another CRT method

Montgomery's algorithm

Conclusion

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.

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

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:

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

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 $2d$:

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

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

Introduction

The lifting approach

Couveignes' algorithm

Another CRT method

Montgomery's algorithm

Conclusion

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.

CSE291-14: The Number Field Sieve

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

Emmanuel Thomé

March 1st, 2022

Part 8a

Discrete logarithms in finite fields

Black-box algorithms for discrete logarithm

Index Calculus in $\mathbb{F}_p$

Building the diagram

What we would like to do with relations

Plan

Black-box algorithms for discrete logarithm

Index Calculus in $\mathbb{F}_p$

Building the diagram

What we would like to do with relations

The discrete logarithm problem

Definition of DLP in a cyclic group

In a cyclic group G, the Discrete Logarithm Problem (DLP) is, given a generating element g and a target h, to find x such that

$$g^x = h.$$

The solution $x \stackrel{\text{def}}{=} \log_g h$ is defined modulo the order of the group.

Many cryptographic primitives can be based on a cyclic group (DSA, ElGamal, ...). Their security relies on the hardness of DLP in the underlying group.

- Some groups have trivially easy DLP, and must not be used for cryptography.
- DLP is often harder. (FF, EC: no poly-time algorithm known).

Pohlig–Hellman

Let G be a cyclic group of size $n = \prod_{i=1}^{k} \ell_i^{e_i}$.

Theorem (Pohlig–Hellman): It is possible to compute a discrete logarithm in G by computing e_i discrete logarithms in groups of size ℓ_i, for $1 \leq i \leq k$.

Tools: Elementary group theory (to deal with prime powers) and Chinese Remainder Theorem (to combine results from different primes).

Asymptotically, computing a discrete logarithm in a cyclic group of order n is (up to a polynomial factor) as hard as computing a discrete logarithm with a black box algorithm in a cyclic group of order ℓ, where ℓ is the largest prime factor of n.

$\Rightarrow$ For cryptography, do not use groups with smooth order.

Pollard ρ for discrete logarithm

Let $h = g^x$. Computing the discrete logarithm x can be done with Pollard ρ algorithm by looking for an equality of the form

$$g^{\alpha_1} h^{\beta_1} = g^{\alpha_2} h^{\beta_2}.$$

Then, x is a solution of the equation $(\alpha_2 - \alpha_1)x = \beta_1 - \beta_2$ modulo the group order.

The "random" function must be defined so that it is possible to keep track of the exponent of g and h.

Example of a "random" function:

$$f(t) = \begin{cases} ht, & \text{for } t \in G_0 \\ t^2, & \text{for } t \in G_1 \\ gt, & \text{for } t \in G_2 \end{cases},$$

where $G_0 \cup G_1 \cup G_2$ is a partition of the group G.

Complexity: $O(\sqrt{\#G})$.

Multiple variants of Pollard ρ

Following the same basic idea, Pollard ρ has several known variations, notably "Parallel collision search".

This is part of the old algorithmic number theory folklore, and is still the state of the art for problems such as ECDLP.

Most of the improvements on this in the last 2 decades or so have been about the possibility of winning a constant factor below 2, or about usefully employing platform X for this problem (X=FPGA, GPU, ...).

Baby-step Giant-step algorithm

Let $h = g^x$. Write $x = iM + j$ for a chosen integer M, with $0 \leq i \leq \#G/M$ and $0 \leq j \leq M$.

Goal: find i and j such that $h(g^{-M})^i = g^j$.

Algorithm:

- compute $\gamma = g^{-M}$.
- Baby steps: compute $S = \{g^j \mid 0 \leq j \leq M\}$.
- Giant steps: for $0 \leq i \leq \#G/M$, compute $h\gamma^i$ and stop if it is in S.

Complexity: $O(\sqrt{\#G})$ (deterministic, proven).

- if M is chosen to be $\lceil \sqrt{\#G} \rceil$
- if the test "is in S" is done in $O(1)$ (e.g., with hash tables)

Shoup's theorem

Let G be a cyclic group of prime order ℓ.

Shoup's theorem: any "generic" algorithm that solves the discrete logarithm problem in G must perform at least $\Omega(\sqrt{\ell})$ group operations.

"generic" means that the algorithm has access to the group structure only *via* to two oracles: one for performing group operations and one for testing for equality in the group.

In practice, we always have access to much more information on the group !

So there is still hope to find better algorithms than Pollard ρ and BSGS for specific groups.

Similarity with factoring

Note that we have analogies:

- Pollard ρ for DL $\leftrightarrow$ Pollard ρ for factoring.
- BS/GS for DL $\leftrightarrow$ Pollard-Strassen.

As we will see, this carries over to the index calculus setting.

Plan

Black-box algorithms for discrete logarithm

Index Calculus in $\mathbb{F}_p$

Building the diagram

What we would like to do with relations

Plan

Index Calculus in $\mathbb{F}_p$

- $L(1/2)$ algorithm
- $L(1/3)$ algorithm: NFS-DL

Index Calculus: algorithm for $\mathbb{F}_p$

Lots of common points with Dixon's algorithm.

Based on Kraitchik's idea of combination of congruences.
Formalized in the 1970's. Proven $L_p(1/2)$.

Let $h = g^x$, where g is a generator of (a subgroup of) $\mathbb{F}_p^\times$.

- We are interested in the factorization of $h \times g^i \bmod p$ (seen as an integer) only if it is smooth.
- We fix a smoothness bound B.

A relation is interpreted as an equality between the unknown x and the (unknown) logarithms of the elements of the factor base:

$$h \times g^i \equiv p_1^{e_{i,1}} \times \cdots \times p_k^{e_{i,k}} \pmod{p}$$
$$x + i \log_g(g) \equiv e_{i,1} \log_g p_1 + \cdots + e_{i,k} \log_g p_k \pmod{p-1}$$

Index Calculus: algorithm for $\mathbb{F}_p$

Algorithm:

- Pick i at random. Test divisibility by all primes below B.
  If $h \times g^i \bmod p$ is B-smooth, keep the relation:

$$h \times g^i \equiv p_1^{e_{i,1}} \times \cdots \times p_k^{e_{i,k}} \pmod{p}.$$

- Solve the linear system to find x (and the logarithms of all elements of the factor base that appear in at least one relation).
  This is a linear algebra problem modulo (a factor of) $p-1$.

Index Calculus: example

Example with $p = 107$, $\ell = 53$, $g = 2$ and $h = 43$.

$$h \times g^{12} \bmod 107 = 6 = 2 \times 3$$
$$h \times g^{22} \bmod 107 = 45 = 3^2 \times 5$$
$$h \times g^{36} \bmod 107 = 50 = 2 \times 5^2$$
$$h \times g^{46} \bmod 107 = 54 = 2 \times 3^3.$$

$$\Longrightarrow \begin{pmatrix} -1 & 1 & 1 & 0 \\ -1 & 0 & 2 & 1 \\ -1 & 1 & 0 & 2 \\ -1 & 1 & 3 & 0 \end{pmatrix} \begin{pmatrix} x \\ \log_g(2) \\ \log_g(3) \\ \log_g(5) \end{pmatrix} = \begin{pmatrix} 12 \\ 22 \\ 36 \\ 46 \end{pmatrix}$$

Solving the linear system modulo $\ell = 53$ gives $x \equiv 6 \mod \ell$.
Indeed, $2^6 \equiv \pm 43 \mod p$.

We don't care about the 2-part!

Note that the previous example (on purpose) is not helpful to tell apart the cases $\log_g h = 6$ and $\log_g h = 6 + 53$.

- This is expected, as we're only trying to illustrate our capacity to solve the problem modulo large (prime) factors of $p - 1$.
- There are always easy ways to find out the discrete logarithm modulo other small factors.
  Extreme case modulo 2: the Legendre symbol.
- Pohlig-Hellman/CRT can be used to combine the information (possibly from various sources) and obtain "the" logarithm.

Main steps of index calculus

This (very basic) index calculus method goes through the following steps.

- Collect relations.
- Solve linear system modulo ℓ.

Complexity: $L_p(1/2)$, proven (like Dixon's algorithm).

Next: how do we use a Number Field Sieve-like setup?

Plan

Index Calculus in $\mathbb{F}_p$

- $L(1/2)$ algorithm
- $L(1/3)$ algorithm: NFS-DL

Setting from here on

- a finite field $\mathbb{F}_q$,
  - Our main case of interest is $q = p$ prime.
  - But $q = p^k$ a prime power sometimes.
- a prime factor ℓ of $q - 1$, without multiplicity.
  This technicality implies that the ℓ-th roots of unity in $\mathbb{F}_q^\times$ can be used as representatives for the quotient group $(\mathbb{F}_q^\times)/(\mathbb{F}_q^\times)^\ell$.

Goal

We want to find a non-trivial discrete logarithm map

$$\mathbf{L} : \begin{cases} \mathbb{F}_q^\times & \to & \mathbb{Z}/\ell\mathbb{Z}, \\ x & \mapsto & \mathbf{L}(x). \end{cases}$$

with $\operatorname{Ker} \mathbf{L} = (\mathbb{F}_q^\times)^\ell$.

Setting from here on

Goal

We want to find a non-trivial discrete logarithm map

$$\mathbf{L} : \begin{cases} \mathbb{F}_q^\times & \to & \mathbb{Z}/\ell\mathbb{Z}, \\ x & \mapsto & \mathbf{L}(x). \end{cases}$$

with $\operatorname{Ker} \mathbf{L} = (\mathbb{F}_q^\times)^\ell$.

There is more than one solution to this problem.

- Any two solutions are proportional, since $(\mathbb{F}_q^\times)/(\mathbb{F}_q^\times)^\ell$ is a 1-dimensional vector space.
- Each can be linked to the logarithm in some base, but we don't really have to care, as $\log_a b \equiv \mathbf{L}(b)/\mathbf{L}(a) \mod \ell$.
- Given $\mathbf{L}$, we can find out the corresponding base easily.

The diagram

The diagram for factoring looked like:

$$
\begin{tikzcd}
 & \mathbb{Z}[x] \arrow[dl, "x \to m"'] \arrow[dr, "x \to \alpha"] & \\
\text{subring of } \mathbb{Q} \quad \mathbb{Z}[m] \subset \mathbb{Q} \arrow[dr, "\bmod N"'] & & \mathbb{Z}[\alpha] \quad \text{subring of } \mathbb{Q}(\alpha) \arrow[dl, "\alpha \to m"] \\
 & \mathbb{Z}/N\mathbb{Z} &
\end{tikzcd}
$$

Some adaptation work ahead

Several differences.

- The structure below should be $\mathbb{F}_q$, or even better, a subgroup of $\mathbb{F}_q^\times$.
- It might be useful to write down something with only group morphisms throughout the diagram. (not rings)
- We know that $\mathbb{F}_q$ is field. We can do more things in a field than in $\mathbb{Z}/N\mathbb{Z}$.

Write something multiplicative

$$\begin{tikzcd}
 & \boxed{\begin{array}{c}\text{group generated by}\\ \mathbb{Z}[x] - \langle p, x - m\rangle\end{array}} \arrow[dl, "x \to m"'] \arrow[dr, "x \to \alpha"] & \\
\boxed{\text{subgroup of } \mathbb{Q}^\times} \arrow[dr, "\text{mod } p"'] & & \boxed{\text{subgroup of } \mathbb{Q}(\alpha)^\times} \arrow[dl, "\alpha \to m"] \\
 & \mathbb{F}_p^\times &
\end{tikzcd}$$

Write something multiplicative

$$
\begin{tikzcd}
 & \boxed{\begin{array}{c}\text{group generated by}\\ \mathbb{Z}[x] - \langle p, x-m\rangle\end{array}} \arrow[dl, "x \to m"'] \arrow[dr, "x \to \alpha"] & \\
\boxed{\begin{array}{c}\text{subgroup of } \mathbb{Q}^\times\\ \text{mod } \ell\text{-th powers}\end{array}} \arrow[dr, "\text{mod } p"'] & & \boxed{\begin{array}{c}\text{subgroup of } \mathbb{Q}(\alpha)^\times\\ \text{mod } \ell\text{-th powers}\end{array}} \arrow[dl, "\alpha \to m"] \\
 & (\mathbb{F}_p^\times)/(\mathbb{F}_p^\times)^\ell &
\end{tikzcd}
$$

- Goal 1: build the diagram.
  - How do we create the two sides?
  - Generalization to two number fields.
- Goal 2: understand it and use it.
  - How do we make sense of relations?
  - How do we formulate the linear algebra problem?
- Goal 3: how do we compute discrete logarithms in $\mathbb{F}_p$?

Plan

Black-box algorithms for discrete logarithm

Index Calculus in $\mathbb{F}_p$

Building the diagram

What we would like to do with relations

Polynomial selection methods

Easy starting point: any polynomial selection methods that works for N (for factoring) will work for p (for DLP in $\mathbb{F}_p$).
Several computations actually used that in the past.

Rephrasing of the polynomial selection problem:

Find two polynomials f_0, f_1 with a common known root in the target ring (here, $\mathbb{F}_p$).

- HUGE difference with the $\mathbb{Z}/N\mathbb{Z}$ case: we can find roots of polynomials in $\mathbb{F}_p$ in polynomial time.
- Can we use that to find a better pair of polynomials?

Joux-Lercier polynomial selection (2003)

Let d be a target degree.

Algorithm:

- Pick any irreducible polynomial f_0 of degree $d+1$ and very small coefficients (good for NFS).
- Test if f_0 has a root m modulo p (and find it).
- There are polynomials f_1 of degree $\leq d$ that also have m as a root modulo p.
  - f_0 is NOT one of them. (degree too large!)
  - The set of such polynomials f_1 is a lattice. Find a small (irreducible) one.
  - Any solution will be coprime with f_0 over $\mathbb{Z}$.

Joux-Lercier polynomial selection (2003)

Input: p prime, degree d

Output: f_0, f_1, m with $f_0, f_1 \in \mathbb{Z}[x]$ irreducible of degrees $d+1, d$, $f_0(m) = f_1(m) = 0 \mod p$

repeat

  Choose f_0 of degree $d+1$ and tiny coefficients, irreducible in $\mathbb{Z}[x]$ and having a root m modulo p

  $$\text{LLL}\left.\begin{bmatrix} p & & & \\ -m & 1 & & \\ & \ddots & \ddots & \\ & & -m & 1 \end{bmatrix}\right\}_{d+1} = \begin{bmatrix} c_0 & c_1 & \cdots & c_d \\ & & * & \\ & & & \end{bmatrix}$$

  $f_1 \leftarrow \sum_{i=0}^{d} c_i x^i$

until f_1 is irreducible in $\mathbb{Z}[x]$

return (f_0, f_1, m)

Joux-Lercier polynomial selection (2003)

Joux-Lercier polynomial selection creates two number fields of degrees $d+1$ and d.

- It is certainly not a big deal as far as building the diagram goes.
  - Both f_0 and f_1 have the root m modulo p.
  - Number field $K_0 = \mathbb{Q}(\alpha_0)$. Map to $\mathbb{F}_p$ sends α_0 to m mod p.
  - Number field $K_1 = \mathbb{Q}(\alpha_1)$. Map to $\mathbb{F}_p$ sends α_1 to m mod p.
- Defining relations as simultaneous occurrences of $\mathrm{Res}(\cdot, f_0)$ and $\mathrm{Res}(\cdot, f_1)$ being smooth is easy as well.
- We'll need need to be somewhat more formal in order to properly make sense of relations.

Joux-Lercier polynomial selection

Search is pretty simple.

- Pick good-looking f_0
  (upper bound on coefficients, upper bound on $\alpha(f_0)$, ...).
- Run LLL, see the quality of the resulting f_1
  (metrics: size of coefficients, and $\alpha(f_1)$, ...).
- Break ties with some sample sieving.

Many unexplored things:

- There's no ultra-specialized algorithm like Kleinjung's algorithms.
- Our real focus of interest is the infinity (max-) norm, while lattice reduction gives a Euclidean short vector. Any possible improvement here, no idea.

Joux-Lercier polynomial selection in practice

As a rule of thumb, for identical bit size:

- If NFS-factoring would like a $(\deg f_0 = 1, \deg f_1 = d)$ pair, with even $d = 2d'$, then a JL pair with $(1 + d', d')$ would be a better choice at this size (for NFS-DL).
- For odd d, the JL construction puts us further from the optimum, so factoring-like polynomial selection can win.

Example for 240 decimal digits:

- Factoring (RSA-240): $(\deg f_0 = 1, \deg f_1 = 6)$.
- DLP (DLP-240, current record): $(\deg f_0 = 4, \deg f_1 = 3)$.

Plan

Black-box algorithms for discrete logarithm

Index Calculus in $\mathbb{F}_p$

Building the diagram

What we would like to do with relations

Fast forward to after relation collection

Relation collection works exactly the same way.

Of course, we want to keep track valuations in $\mathbb{Z}$, and no longer reduce them modulo 2.

A relation is...

- $a - b\alpha_0$ factors into small prime ideals in $\mathcal{O}_{K_0}$.

$$(a - b\alpha_0)\mathcal{O}_{K_0} = \prod_i \mathfrak{p}_i^{e_i}.$$

- $a - b\alpha_1$ factors into small prime ideals in $\mathcal{O}_{K_1}$.

$$(a - b\alpha_1)\mathcal{O}_{K_1} = \prod_j \mathfrak{q}_j^{g_j}.$$

- and both map to the finite field element $(a - bm) \in \mathbb{F}_p^\times$.

Why is it not a trivial question?

$$(a - b\alpha_0)\mathcal{O}_{K_0} = \prod_i \mathfrak{p}_i^{e_i} \qquad \text{and} \qquad (a - b\alpha_1)\mathcal{O}_{K_1} = \prod_j \mathfrak{q}_j^{g_j}.$$

- Can we turn this into an additive relation involving logarithms?

$$\sum_i e_i \mathbf{L}(\text{image of } \mathfrak{p}_i) \text{ “=” } \sum_j g_j \mathbf{L}(\text{image of } \mathfrak{q}_j).$$

- Main problem: ideals are not elements. No image in $\mathbb{F}_p$!!

An idealized approach

Assume that we know the class number and the unit group in both number fields. This is totally unrealistic!

Rosy setup

- Class number h_0 in K_0 is such that for any ideal $\mathfrak{p}$ in $\mathcal{O}_{K_0}$, the ideal $\mathfrak{p}^{h_0}$ is a principal ideal. Same on the other side.
  Small note: we may safely assume that ℓ and h_0 are coprime.
- Knowledge of the unit group: any unit can be rewritten as a combination of some known generators.

Approach:

- For all ideals, compute a generator $\gamma_{\mathfrak{p}}$ of the ideal $\mathfrak{p}^{h_0}$.
- Then all valuations of $(a - b\alpha_0)^{h_0} / \prod_i \gamma_{\mathfrak{p}_i}^{e_i}$ cancel. This is a unit!
- Decompose this unit w.r.t. the generating system.

An idealized approach

The idealized approach rewrites $(a - b\alpha_0)$ and $(a - b\alpha_1)$ as products of elements.

$$(a - b\alpha_0)^{h_0} = u_1^{\text{something}} \times \cdots \times u_{\text{something}}^{\text{something}} \times \prod_i \gamma_{\mathfrak{p}_i}^{e_i}.$$

- These elements form a generating system of the set of $(a - b\alpha_0)$ we are considering, and we have an explicit decomposition.
- We can map each of these elements (u_i and $\gamma_{\mathfrak{p}_i}$) to $\mathbb{F}_p$, and write a linear relation involving **L**(only elements of $\mathbb{F}_p$).
- Alas, it is not practical.

CSE291-14: The Number Field Sieve

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

Emmanuel Thomé

March 3rd, 2022

Part 8b

Virtual and individual logarithms

Making sense: virtual logarithms

Computing Schirokauer maps

Schirokauer maps and sparse linear algebra

Computing individual logarithms

The descent

Plan

Making sense: virtual logarithms

Computing Schirokauer maps

Schirokauer maps and sparse linear algebra

Computing individual logarithms

The descent

Plan

Making sense: virtual logarithms

- Something that works
- Characterizing elements
- Virtual logarithms

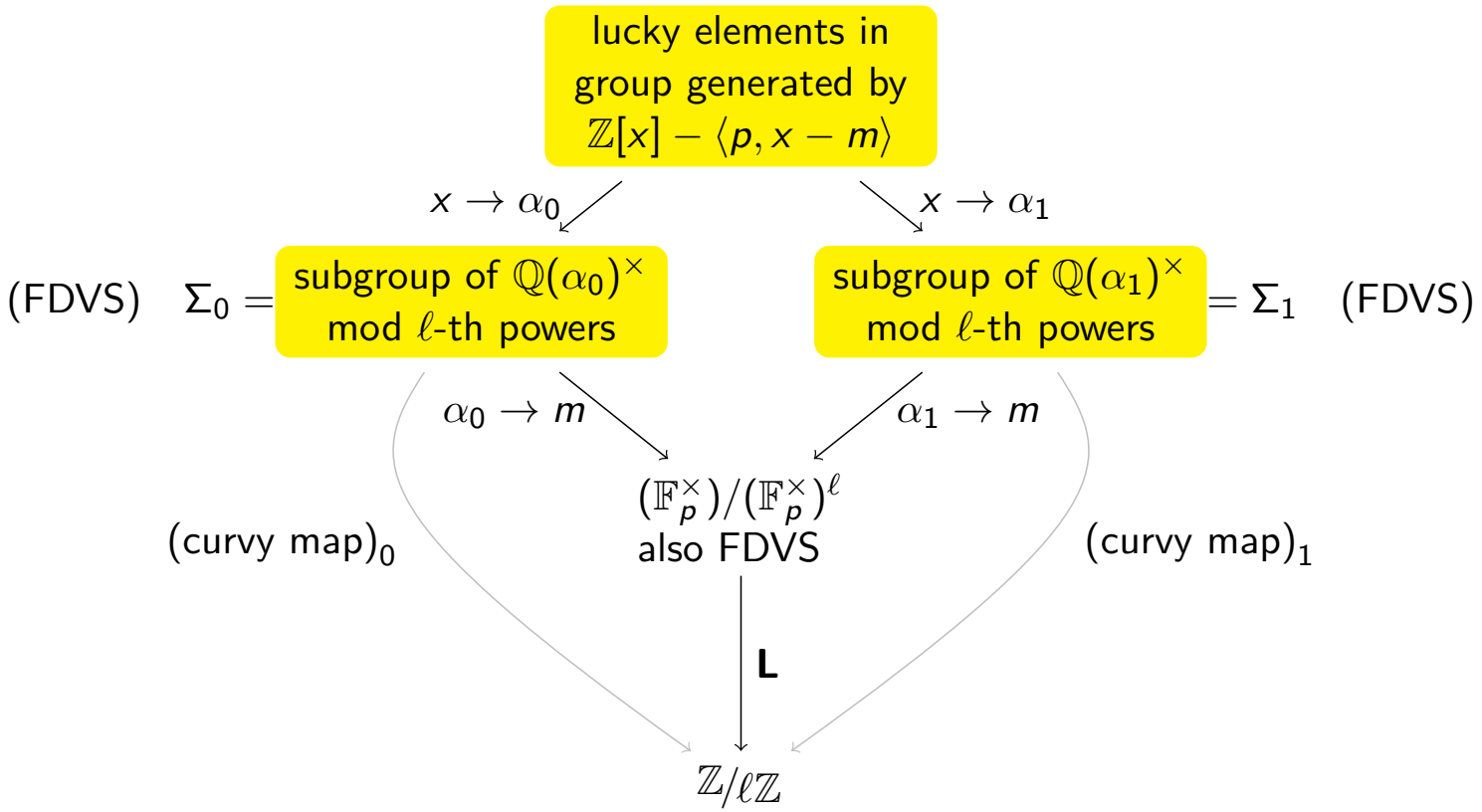
What do we need?

Algebraic structure of the set of $(a - b\alpha_0)$:

- (abelian) subgroup of $\mathbb{Q}(\alpha_0)^\times$.
- $\mathbb{Z}$-module, if we so wish.
- It is also finitely generated, thanks to smoothness. (We just proved it!)

If we look at this set of $(a - b\alpha_0)$ modulo ℓ-th powers, we see that it is a finite dimensional $\mathbb{Z}/\ell\mathbb{Z}$-vector space.

Vector spaces



Our goal (as stated): find **L**.

Alternative goal: find the two curvy linear forms, + a lifting map.

Previous picture is important

Digression: how can we characterize elements in Σ_0 and Σ_1?

Note: in the following slides, α, K, Σ denote things that can be instantiated on either side of the diagram.

Plan

Making sense: virtual logarithms

- Something that works
- Characterizing elements
- Virtual logarithms

Riddles

I'm thinking of some rational number.

What does it take to identify it without ambiguity?

- valuations (in $\mathbb{Z}$) at all primes.
- sign.

Riddles

I'm thinking of some B-smooth rational number.

What does it take to identify it without ambiguity?

- valuations (in $\mathbb{Z}$) at all primes below B.
- sign.

Riddles

I'm thinking of some B-smooth rational number mod ℓ-th powers.

What does it take to identify it without ambiguity?

- valuations (in $\mathbb{Z}/\ell\mathbb{Z}$) at all primes below B.
- ~~sign~~. (not needed since ℓ odd).

Next: we want to generalize.

Characterizing an algebraic number

I'm thinking of some element of Σ (Σ_0 or Σ_1).
i.e. some B-smooth element of $\mathbb{Q}(\alpha)^\times$ mod ℓ-th powers.

To identify it without any ambiguity, we need:

- valuations modulo ℓ at all prime ideals whose norm is $\leq B$.
- any information that can break ties. Ties are related to units.
  - in our rosy picture, valuations at the generating units played this role.
  - but in fact, we do not have to take this long-winded path. Other functions could do.

Characterizing with characters

Requirements for tie breakers

Suppose that we have a finite set of characters $\{\chi\}$ such that:

- The domain of all χ is $\mathbb{Q}(\alpha)^\times$ (focus: B-smooth elements).
- $\chi : (\mathbb{Q}(\alpha)^\times, \times) \to (\mathbb{Z}/\ell\mathbb{Z}, +)$.
- Characters χ are efficiently computable.
- $\mathcal{O}_K^\times \cap \left(\bigcap_\chi \mathrm{Ker}\,\chi\right) = (\mathcal{O}_K^\times)^\ell$.

Then two elements that agree on all ideal valuations AND on all of these characters must be equal modulo ℓ-th powers.

In the NFS-DL context, we use the Schirokauer maps for these characters.

Valuations and characters

Consider the set $\{\nu_{\mathfrak{p}}\}_{N(\mathfrak{p})\leq B} \cup \{\chi_1, \ldots, \chi_c\}$.

- $\nu_{\mathfrak{p}}$ are valuations $\bmod \ell$ at each ideal that is allowed to appear.
- χ_i are our characters.

All of these are linear forms defined on Σ.

As such, they are elements of the dual space Σ^*.

The curvy map $\Sigma \to \mathbb{Z}/\ell\mathbb{Z}$ that we are looking for is also an element of Σ^*.

Plan

Making sense: virtual logarithms

- Something that works
- Characterizing elements
- Virtual logarithms

Virtual logarithms

If $\mathbf{B} = \{\nu_{\mathfrak{p}}\} \cup \{\chi_1, \ldots, \chi_c\}$ is a basis of the dual space Σ^*.
(analogue for $\mathbb{Q}$: elements of $\mathbf{B}$ would be just "valuations at primes")

- Then the curvy map $\Sigma_0 \to \mathbb{Z}/\ell\mathbb{Z}$ decomposes along $\mathbf{B}_0$.
  $=$ There exists an identity of linear forms:

$$(\text{curvy map})_0 = \sum_{e \in \mathbf{B}_0} \lambda_e e.$$

- Because $a - b\alpha_0$ and $a - b\alpha_1$ are known to map identically through $(\mathbb{F}_p^\times)/(\mathbb{F}_p^\times)^\ell$, then both curvy maps must agree:

$$(\text{curvy map})_0(a - b\alpha_0) = (\text{curvy map})_1(a - b\alpha_1).$$
$$\sum_{e \in \mathbf{B}_0} \lambda_e e(a - b\alpha_0) = \sum_{e' \in \mathbf{B}_1} \lambda_{e'} e'(a - b\alpha_1).$$

Virtual logarithms

$$\sum_{e \in \mathbf{B}_0} \lambda_e e(a - b\alpha_0) = \sum_{e' \in \mathbf{B}_1} \lambda_{e'} e'(a - b\alpha_1).$$

Virtual logarithms

The virtual logarithm of $e \in \mathbf{B}$ is the coefficient λ_e.

- It can be the virtual logarithm at some ideal, or the virtual logarithm at some character.
- Relations in NFS-DL express necessary conditions on all these virtual logarithms.
- As previously mentioned, our linear system finds solutions up to a constant factor, since all constraints are homogeneous.

The "too many characters" case

Other case: $\mathbf{B} = \{\nu_{\mathfrak{p}}\} \cup \{\chi_1, \ldots, \chi_c\}$ contains a basis of the dual space Σ^*.

- Then there is some ambiguity in the decomposition along $\mathbf{B}_0$.
- This means that a full vector space of (all "correct") virtual logarithm choices work. None is privileged.

Bear in mind

Virtual logarithm = the coefficient that goes alongside a "valuation at $\mathfrak{p}$" or "character value at χ" and meets the conditions that come from the diagram.

If everything were simple and easy, we would only have valuations at a set of elements, and these coefficients would be logarithms of images of these elements in $\mathbb{F}_p$.

Linear algebra

The linear algebra problem

A pair (a, b) yields a constraint that binds the virtual logarithms. With sufficiently many constraints, we have the correct kernel. Equations in this linear system are hybrid:

- Coefficients $\nu_{\mathfrak{p}}(a - b\alpha)$ come from valuations (very sparse), and coefficients $\chi(a - b\alpha)$ are likely to be dense.
- The system can be written as $M \times v = 0$, or $(M|C) \times v = 0$. (depending on whether we include the character block C in M or not.)

This is homogeneous, and we are looking for a right kernel vector.

This is subtly different from the factoring case.

Our characters here have to enter before linear algebra.

Plan

Making sense: virtual logarithms

Computing Schirokauer maps

Schirokauer maps and sparse linear algebra

Computing individual logarithms

The descent

Requirements

Requirements for characters $\Sigma \to \mathbb{Z}/\ell\mathbb{Z}$

We want a finite set of characters $\{\chi\}$ such that:

- The domain of all χ is $\mathbb{Q}(\alpha)^\times$ (focus: B-smooth elements).
- $\chi : (\mathbb{Q}(\alpha)^\times, \times) \to (\mathbb{Z}/\ell\mathbb{Z}, +)$.
- Characters χ are efficiently computable.
- $\mathcal{O}_K^\times \cap \left(\bigcap_\chi \mathsf{Ker}\,\chi\right) = (\mathcal{O}_K^\times)^\ell$.

We will build such characters with ℓ-adic arithmetic.

Setting

We can safely assume that:

- we deal with elements that are coprime to $\ell\mathcal{O}_K$. (B-smooth elements and their products certainly are).
- the ideal $\ell\mathcal{O}_K$ does not ramify (ℓ is coprime to $\operatorname{disc} K$), and ℓ is coprime to f_d as well.

Let us suppose that the defining polynomial factors modulo ℓ as

$$f(x) \mod \ell = (\text{degree } d_1) \times \cdots \times (\text{degree } d_k).$$

(with no repeated factors, by our assumption above.)

Step 1: $\Sigma \to \mathcal{O}_K/\ell\mathcal{O}_K$

The quotient $\mathcal{O}_K/\ell\mathcal{O}_K$ is a product of fields, not a field.

The projection is a simple coefficient-wise reduction.

$$R : \left\{ \begin{array}{rcl} \mathcal{O}_K & \to & \mathcal{O}_K/\ell\mathcal{O}_K \cong \mathbb{F}_{\ell^{d_1}} \times \cdots \times \mathbb{F}_{\ell^{d_k}} \\ \phi(\alpha), \phi \in \mathbb{Z}[x] & \mapsto & \phi(R(\alpha)), \phi \in \mathbb{Z}/\ell\mathbb{Z}[x]. \\ u/v & \mapsto & R(u)/R(v). \end{array} \right.$$

Elements coprime to $\ell\mathcal{O}_K$ map to invertible elements, so the projection R is well defined on Σ as well.

For any element $u \in \Sigma$:

- the first component of $R(u)$ is in $(\mathbb{F}_{\ell^{d_1}})^\times$. Then its $(\ell^{d_1} - 1)$-th power is 1.
- by extension, the $\mathrm{lcm}(\{\ell^{d_i} - 1\}_i)$-th power of $R(u)$ is 1 on all components, so it is really 1.

Precision ℓ^2

To obtain something useful, we need R to project modulo ℓ^2 instead of just ℓ.

Schirokauer maps

Let $R : \mathcal{O}_K \mapsto \mathcal{O}_K/\ell^2\mathcal{O}_K$.
Let $X_u(x)$ be the degree-$(d-1)$ polynomial such that:

$$R(u(\alpha))^{\mathrm{lcm}(\{\ell^{d_i}-1\}_i)} = 1 + \ell \cdot X_u(R(\alpha)) \mod \ell^2.$$

What is X_{uv}? Since $(1+\ell x)(1+\ell y) = (1+\ell(x+y)) \mod \ell^2$, we have:

$$X_{uv} = X_u + X_v.$$

Therefore $u \mapsto X_u$ is a morphism from $(\Sigma, \times)$ to $(\mathbb{Z}/\ell\mathbb{Z}^d, +)$.

Sage code

```
ZP.<x>=ZZ['x']
f=ZP([randrange(-100,100) for i in range(6)])
ell=1009
assert gcd(ell,f.discriminant()*f[f.degree()]) == 1
f2=ZP.change_ring(R2)(f)
ZR2ell=quotient(ZP.change_ring(R2),f2/f2[f2.degree()])
big=lcm([ell^(fac[0].degree())-1
    for fac in f.change_ring(GF(ell)).factor()])
def schirokauer_map(u):
    # u a polynomial with coefficients in ZZ
    return vector([GF(ell)(ZZ(c)//ell)
        for c in list(ZR2ell(u)^big-1)])
```

Sage code

```
sage: u=ZP([randrange(10) for i in range(5)])
sage: v=ZP([randrange(10) for i in range(5)])
sage: schirokauer_map(u)
(772, 522, 920, 969, 740)
sage: schirokauer_map(u*v)
(415, 10, 406, 215, 473)
sage: schirokauer_map(u)+schirokauer_map(v)
(415, 10, 406, 215, 473)
```

Do we meet the requirements?

If units are well-behaved, yes.

- In some highly unlikely scenarios, we might need larger ℓ-precision.
- Some more dramatic failures are guarded by Leopoldt's conjecture (widely believed to be true).

In all likelihood, if u is a unit, then $X(u) = 0$ iff $u \in (\mathcal{O}_K^\times)^\ell$.

The set of coordinates of X can be used as our desired set of characters $\{\chi\}$.

Faster computation

It is actually a little bit faster to compute the Schirokauer maps piecewise modulo each of the factors.

We may decide to compute only part of the coordinates, if we know that the unit rank is not that big.

Plan

Making sense: virtual logarithms

Computing Schirokauer maps

Schirokauer maps and sparse linear algebra

Computing individual logarithms

The descent

The linear algebra problem

Values of the Schirokauer maps

If we write the linear system as $(M \mid C) \times v = 0$, note that:

- M: very sparse, small coefficients.
- C: dense, coefficients modulo ℓ.

Note that we cannot first ignore the C block, and add it afterwards. It won't work. So the factoring case $(v \times M)$ and DLP case $(M \times v)$ are very different.

First approach

The first approach is to solve the system $(M \mid C) \times v = 0$ as is.

- Each matrix-times-vector operation involves multiplication of several full-length integers modulo ℓ.
  - Typically, "normal" matrix coefficients are ± 1, and processing them entail one addition modulo ℓ.
  - Multiplications cost $O(\log \ell)$ times more, and dealing with the Schirokauer block ends up being a large part of the overall cost.
- This also means more code, more storage, etc.
  DLP240: $M \approx 70G$, $C \approx 30G$.

Note that in this context, one kernel vector is enough.

Alternative approach

It is way better to use the block Wiedemann in a smart way.

Assumptions:

- r columns in the Schirokauer block,
- BW blocking parameters m, n such that $n \geq r$.
  For simplicity, assume $n = r$.

Use only M as the matrix, and in BW set $y = C$.

Given one column j of the linear generator matrix, we have:

$$\begin{aligned} 0 = &(C_0 f_{0,j,0} + \cdots + C_{r-1} f_{r-1,j,0}) + \\ &M(C_0 f_{0,j,1} + \cdots + C_{r-1} f_{r-1,j,1}) + \\ &M^2(C_0 f_{0,j,2} + \cdots + C_{r-1} f_{r-1,j,2}) + \cdots \end{aligned}$$

Alternative approach

$$
\begin{aligned}
0 = {} & (C_0 f_{0,j,0} + \cdots + C_{r-1} f_{r-1,j,0}) + \\
& M(C_0 f_{0,j,1} + \cdots + C_{r-1} f_{r-1,j,1}) + \\
& M^2(C_0 f_{0,j,2} + \cdots + C_{r-1} f_{r-1,j,2}) + \cdots \\
0 = {} & C \times \text{some vector} + M \times \text{some other vector},
\end{aligned}
$$

which is exactly what we're looking for. In particular, we need only look at one of the possible solutions.

This brings only benefits, since in practice:

- r is at most $(\deg f_0 - 1) + (\deg f_1 - 1) \leq 5$. (DLP240: $r = 4$).
- We like to choose n so that some distribution is possible. (DLP240: $n = 16$).

Plan

Making sense: virtual logarithms

Computing Schirokauer maps

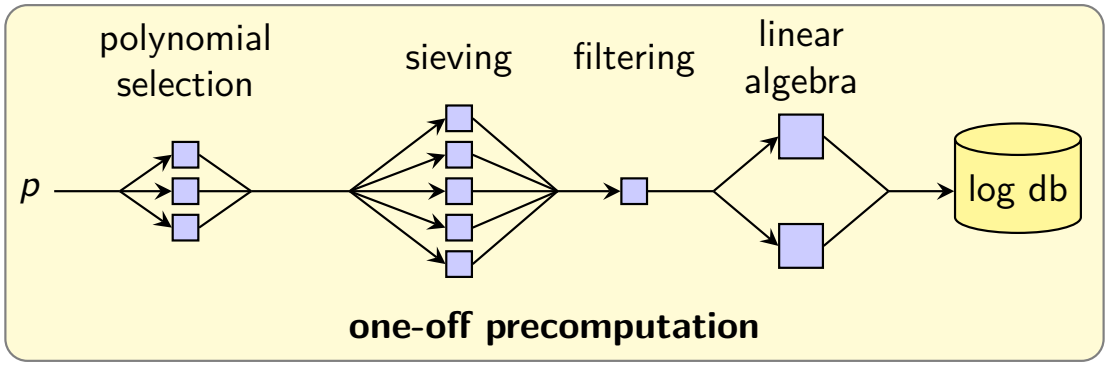
Schirokauer maps and sparse linear algebra

Computing individual logarithms

The descent

Where are we?

After the linear algebra step, we have a big database of (virtual) logarithms for many ideals and Schirokauer maps.



Cost thus far:

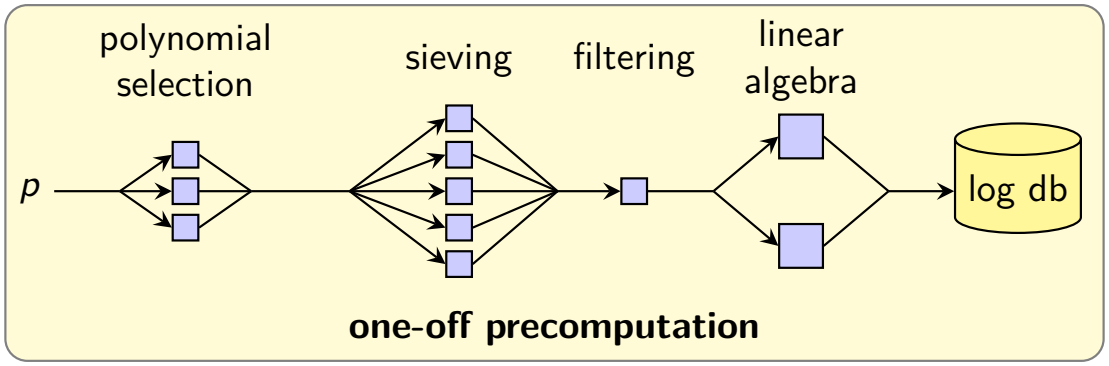
- the analysis is exactly the same.
- Joux-Lercier polynomial selection doesn't change anything.

Cost of NFS-DL precomputation

$$\text{relation collection} \approx \text{sieving} \approx L_p(1/3, (64/9)^{1/3} + o(1)).$$

Where are we?

After the linear algebra step, we have a big database of (virtual) logarithms for many ideals and Schirokauer maps.



This is not enough.

- We want to compute $\mathbf{L}(x)$ for arbitrary things.
- Anyway, the keys of our database are NOT elements of $\mathbb{F}_p^\times$.

How do we compute $\mathbf{L}(h)$ for an arbitrary $h \in \mathbb{F}_p^\times$, given our database?

Not so easy to get it right

General idea

If $h \in \mathbb{F}_p^\times$ is something that decomposes along our factor base, we're good.
Caveat: our factor base consists of ideals.

The process consists of two steps.

- An initial lift of our target data on one of the sides.
- A descent process that allows us to relate our target to the things that are referenced in the database.

The earliest version of this process, although in a different context, appeared in Coppersmith's 1984 paper on $L(1/3)$ discrete logarithms in $\mathbb{F}_{2^n}$.

Random self-reduction

If we want to compute $\mathbf{L}(h)$, note that for any invertible e mod ℓ, we have:

$$\mathbf{L}(h) = \frac{1}{e}\mathbf{L}(h^e).$$

This implies that we have worst-case to average-case reduction for this problem.

We will use this randomization property soon.

Lifting / Finding preimages

Our element $h \in \mathbb{F}_p^\times$ must be lifted on one of the two sides.

- From a mathematical standpoint, both sides would "work".
- In practice, one side is often "smaller" than the other, and therefore preferred.

In the Joux-Lercier case, we choose the larger-degree, smaller-coefficients polynomial f_0, whose degree is asymptotically

$$\deg f_0 \approx \log_{\log p} L_p(1/3, \cdot).$$

In what follows, $(f, K, \alpha, \ldots)$ are shorthands for $(f_0, K_0, \alpha_0, \ldots)$

Lifting / Finding preimages

The map from K to $\mathbb{F}_p^\times$ sends α to m mod p.

Naive approach

First stategy: lift h to some integer representative, viewed as an element of K.

Our hope: maybe h (or h^e) is a smooth integer.
Would it be useful?

Can we use a smooth integer?

We are able to evaluate **L** on any element which factors into ideals of norm below $B \approx L_p(1/3)$.

We have:

- $h \approx L_p(1)$. In time $L_p(1/3)$, we can hope for some h^e to be $L_p(2/3)$-smooth.
- This would be useful if we were lifting to the rational field, but not here.
  The norm $\mathrm{Norm}_{K/\mathbb{Q}}$ is a lot larger! Not all small primes factor into small prime ideals in the number field.
- We have $\mathrm{Norm}_{K/\mathbb{Q}}(h) \approx L_p(1)^{\log_{\log p} L_p(1/3)} \approx L_p(4/3)$, which is way too large.

Rational reconstruction

Using rational reconstruction

Second approach: lift h to a rational representative such that

$$h \equiv \frac{u}{v} \mod p.$$

Such (u, v) can be obtained with the extended Euclidean algorithm, with $u \approx v \approx \sqrt{p}$.

Two half-size numbers have better chances of smoothness than one that is twice as large.

Yet, this still does not work, as it does not change the asymptotics.

Use sieving

Using rational reconstruction + sieving

Third approach: use two consecutive rational representations of h:

$$h \equiv \frac{u_0}{v_0} \equiv \frac{u_1}{v_1} \mod p.$$

Then, look for multipliers k_0 and k_1 such that

- $k_0 u_0 + k_1 u_1$ is a smooth integer;
- $k_0 v_0 + k_1 v_1$ is a smooth integer.

This can be done with sieving

This might have a tremendous practical impact, but does not really solve the problem. We're dealing with BIG integers and norms here.

Lift to algebraic numbers

Use the fact that $\alpha \to m$ to lift h to something that involves α.

We have more freedom, we have some hope to obtain smaller things, and in particular a smaller algebraic norm.

Lifting to algebraic numbers

Joux–Lercier polynomial selection:
two polynomials of degree $(d, d+1)$

Baby example for a 30-digit prime factor.

$$\begin{aligned} p &= 303754470965461741563252151428 7 \\ f &= x^3 - x + 1 \\ m &= 178827864877625171826943706505 7 \\ g &= 5383086967x^2 - 13169419660x - 2588305959 \end{aligned}$$

$\rightarrow$ no rational side.

Lifting to algebraic numbers (Joux-Lercier)

Lift $h \in \mathbb{F}_p^\times$ to a fraction $\dfrac{u_0 + \cdots + u_d\alpha^d}{v_0 + \cdots + v_d\alpha^d} \mapsto h$.

Define the following lattice

$$L = \begin{bmatrix} p & & & & & & \\ -m & 1 & & & & & \\ & \ddots & \ddots & & & & \\ & & -m & 1 & & & \\ h & 0 & \dots & 0 & 1 & & \\ & \ddots & & & & \ddots & \\ & & h & & & & 1 \end{bmatrix}$$

Each row vector in this lattice is such that

$$(u_0 + \cdots + u_d\alpha^d) - h(v_0 + \cdots + v_d\alpha^d) \mapsto 0.$$

Lifting to algebraic numbers (Joux-Lercier)

target: $h = 928006098329594493306911381 86 \in \mathbb{F}_p^\times$.

Matrix: $$\begin{bmatrix} 303754470965461741563252151428 7 & 0 & 0 & 0 & 0 & 0 \\ -178827864877625171826943706505 7 & 1 & 0 & 0 & 0 & 0 \\ 0 & -178827864877625171826943706505 7 & 1 & 0 & 0 & 0 \\ 928006098329594493306911381 86 & 0 & 0 & 1 & 0 & 0 \\ 0 & 928006098329594493306911381 86 & 0 & 0 & 1 & 0 \\ 0 & 0 & 928006098329594493306911381 86 & 0 & 0 & 1 \end{bmatrix}$$

Apply LLL:

$$\begin{bmatrix} 52606 & 44203 & -25671 & -6448 & -66975 & 5015 \\ 25671 & 26935 & 44203 & -5015 & -1433 & -66975 \\ 44203 & -69874 & -26935 & -66975 & 71990 & 1433 \\ 71307 & -42105 & 21380 & 106583 & -10333 & 35519 \\ -49927 & -50582 & 20725 & -71064 & -131769 & -25186 \\ 3531 & -5555 & -115510 & 101265 & 36952 & -39608 \end{bmatrix}$$

Lifting to algebraic numbers (Joux-Lercier)

target: $h = 928006098329594493306911381 86 \in \mathbb{F}_p^\times$, lift as $y \in \mathbb{Z}$
Each row of LLL output is $[u_0, u_1, u_2, v_0, v_1, v_2]$:

$$\texttt{PREIMAGE}(h) = \frac{u_0 + u_1 x + u_2 x^2}{v_0 + v_1 x + v_2 x^2}$$

First row gives:

$$h \equiv \frac{52606 + 44203m - 25671m^2}{-6448 - 66975m + 5015m^2} \bmod p$$

Is this a win?

This method encompasses rational reconstruction, and is compatible with sieving as well.

Is the norm smaller?

Some analysis

- The dimension of the lattice is $2d + 2 \approx \log_{\log p}(L_p(1/3))$.
- Its determinant is $p = L_p(1)$.
- The L^2 norm of its small vectors are expected to be

$$(\text{approximation factor}) \times \det^{1/\dim} \approx L_p(1)^{1/\log_{\log p} L_p(1/3)} \approx L_p(2/3).$$

- The algebraic norm of the resulting elements can be expected to be

$$L_p(2/3)^d \approx L_p(1).$$

It is a win, because the algebraic norm is what we need for the rest of the process.

Summary of the initial step

- Try many h^e simultaneously.
- Form lattices, pick short vectors. Perhaps do some sieving.
- Compute algebraic norms, try to factor them into prime factors below some bound $B_{\text{init}} \approx L_p(2/3)$. ECM is here to help.
- This costs $L_p(1/3)$.

We are left with some algebraic number whose ideal factorization involves only prime ideal of norm below $L_p(2/3)$.

An example

The initial step for the DLP-240 individual logarithm computation obtains $\frac{u(\alpha)}{v(\alpha)} \mapsto (\text{target})^e$, with:

$$
\begin{aligned}
U(\alpha) &= 111158146549876214366517263 13 \\
&\quad + 84293370936324477859384006099\alpha \\
&\quad + 101013631005385213436119240940\alpha^2 \\
&\quad - 40191063714065449830545857096 7\alpha^3 \\
V(\alpha) &= -14022053826479031727483421649 3 \\
&\quad + 15321432893351276964037257839 2\alpha \\
&\quad - 10489210686953161453362359657\alpha^2 \\
&\quad - 62477932655742140204933047385 5\alpha^3 \\
U(\alpha)\mathcal{O}_K &= \mathfrak{p}_3 \times \mathfrak{p}_5 \times \mathfrak{p}_{113} \times \mathfrak{p}_{1543} \times \mathfrak{p}_{6dd} \times \mathfrak{p}_{10dd} \times \mathfrak{p}_{18dd} \times \mathfrak{p}_{20dd} \times \mathfrak{p}_{20dd} \times \mathfrak{p}_{22dd} \times \mathfrak{p}_{25dd} \\
V(\alpha)\mathcal{O}_K &= \mathfrak{p}_3 \times \mathfrak{p}_7 \times \mathfrak{p}_{41} \times \mathfrak{p}_{113} \times \mathfrak{p}_{401} \times \mathfrak{p}_{14dd} \times \mathfrak{p}_{15dd} \times \mathfrak{p}_{19dd} \times \mathfrak{p}_{22dd} \times \mathfrak{p}_{24dd} \times \mathfrak{p}_{26dd}
\end{aligned}
$$

Plan

Making sense: virtual logarithms

Computing Schirokauer maps

Schirokauer maps and sparse linear algebra

Computing individual logarithms

The descent

The descent

The descent is the process through which we will try to compensate all the oversize ideals in the previous factorization.

Note

The initialization step used only one side (side 0) of the NFS diagram.
The descent process involves BOTH SIDES.

The main idea is to find some $a - bx$ such that:

- $(a - b\alpha_0)\mathcal{O}_{K_0}$ is divisible by one of these oversize ideals.
- Both $(a - b\alpha_0)\mathcal{O}_{K_0}$ AND $(a - b\alpha_1)\mathcal{O}_{K_1}$ are somewhat smooth ideals.

It may be too much of a stretch to aim at our ultimate factor base bound just yet, though.

The descent

Starting point (initialization):

- $\left\{ \begin{array}{l} \text{side 0: } S_0(\alpha_0) = \text{something}, \\ \text{side 1: } S_1(\alpha_1) = 1, \end{array} \right\}$ and $\frac{S_0(m)}{S_1(m)} = \text{target}$.
- $S_0(\alpha_0)\mathcal{O}_{K_0}$ is divisible by some large ideals.

With a properly chosen $a - bx$, we have after this first step:

- $\left\{ \begin{array}{l} S_0'(\alpha_0) = S_0(\alpha_0)/(a - b\alpha_0), \\ S_1'(\alpha_1) = S_1(\alpha_1)/(a - b\alpha_1), \end{array} \right\}$ and $\frac{S_0'(m)}{S_1'(m)} = \text{target}$.
- $S_0'(\alpha_0)\mathcal{O}_{K_0}$ has one large ideal less, and might have gained a few other ideals.
- $S_1'(\alpha_1)\mathcal{O}_{K_1}$ might have gained a few other ideals.

Finding the multiplier

How do we choose $a - bx$? With special-q sieving!

- The ideal that we're trying to cancel has a norm $\approx B_{\text{init}}$.
- If we can obtain smoothness with a new bound $B_1 \leq B_{\text{init}}^c$ for some $c < 1$, we're probably good.

Analysis of the descent steps

Here's how the analysis of the descent could work.

$$\begin{aligned}
\text{Norm}\,\mathfrak{q} &\approx B_{\text{init}} \approx L_p(2/3, k) \\
\mathfrak{q}\text{-lattice coefficients} &\approx \sqrt{\text{Norm}\,\mathfrak{q}} \approx L_p(2/3, k/2) \\
\text{Norm}(a - b\alpha_0) &\approx L_p(1, \text{something} \times k/2) \\
B_1 &\approx L_p(2/3, kc). \\
\text{prob. smoothness} &\approx L_p(1/3, -\frac{1}{6c} \times \text{something})
\end{aligned}$$

We might keep going, and reach $L_p(2/3, kc^n)$ after n steps.

Messy, but works. This yields a cost of $L(1/3, \cdot)$, smaller than the rest.

This is pretty much what we do in practice, although the different bounds are rather chosen by hand.

Alternative descent strategy

Another approach could be not to search for $a - b\alpha$, but for larger degree polynomials with degree $\delta = \log_{\log p} L_p(1/6, \cdot)$ (at the first step).

- The $\mathfrak{q}$-lattice coefficients are smaller in this case: $(\operatorname{Norm} \mathfrak{q})^{\frac{1}{\delta}}$.
- This makes it possible to obtain smaller norms.

Alternative descent strategy

$$\text{Norm}\, u \approx L(1)$$

$$\text{Norm}\, \mathfrak{q} \approx L(1 - 1/3 = 2/3 = 1/3 + 1/3)$$

$$\rightarrow \delta = \text{dim}(\mathfrak{q}\text{-lattice}) \approx \log_{\log p} L_p(1/6)$$

$$\rightarrow \|\textstyle\sum_i a_i \alpha^i = \phi(\alpha)\|_\infty \approx |a_i| \approx L_p(2/3 - 1/6 = 1/2 = 1/3 + 1/6)$$

$$\rightarrow \text{Norm}(\phi(\alpha)) \approx \|\phi\|_\infty^d \|f\|^\delta \approx L_p(\max(1/3 + 1/6 + 1/3, 2/3 + 1/6) = 5/6)$$

$$\text{Norm}\, \mathfrak{q}' \approx L(5/6 - 1/3 = 1/2 = 1/3 + 1/6)$$

$$\rightarrow \delta = \text{dim}(\mathfrak{q}\text{-lattice}) \approx \log_{\log p} L_p(1/12)$$

$$\rightarrow \|\textstyle\sum_i a_i \alpha^i = \phi(\alpha)\|_\infty \approx |a_i| \approx L_p(1/2 - 1/12 = 5/12 = 1/3 + 1/12)$$

$$\rightarrow \text{Norm}(\phi(\alpha)) \approx \|\phi\|_\infty^d \|f\|^\delta \approx L_p(\max(1/3 + 1/12 + 1/3, 2/3 + 1/12) = 9/12)$$

etc. . .

Eventually (and probably very quickly!) $\log_{\log p} L_p(1/(3 \cdot 2^i))$ is very small. For comparison, $\log_{\log p} L_p(1/6) \approx \sqrt{d}$.

Alternative descent strategy

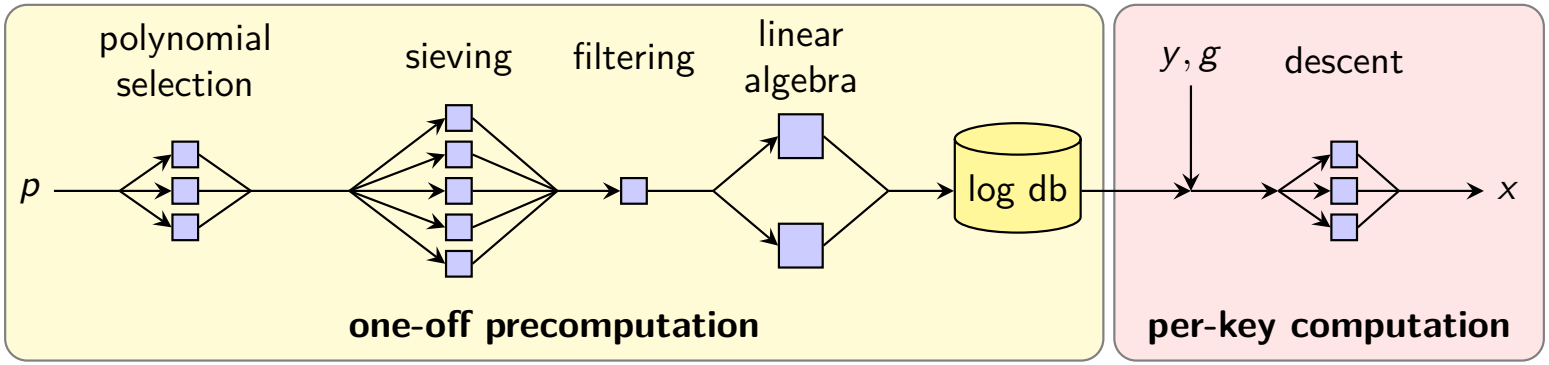
Asymptotically, this gives a much nicer descent complexity, quite in line with Coppersmith's descent idea from 1984.

In practice, higher degree sieving is not used in the descent, but it was a close call for the first step of some of the latest computations.

Overall complexity

While it is a complicated process, the descent is comparatively a lot cheaper than the rest of the computation.

- The initialization step is the most expensive part, with cost $L_p(1/3, 1.232 + o(1))$ with an early-abort strategy.
- The rest of the descent is cheaper.



Typical data (elapsed time using many machines):

	precomputation	per-key
Logjam (512 bits)	week	minutes
DLP-240 (795 bits)	months	hours

CSE291-14: The Number Field Sieve

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

Emmanuel Thomé

March 8, 2022

Part 9

Variants of NFS

Introduction

SNFS

NFS-DL

The (defunct) Function Field Sieve

Extensions to other finite fields

The Multiple Number Field Sieve

$o(1)$

Plan

Introduction

SNFS

NFS-DL

The (defunct) Function Field Sieve

Extensions to other finite fields

The Multiple Number Field Sieve

$o(1)$

Main arguments about complexity of NFS

The cost the factorization of N with GNFS is:

$$L_N(1/3, (64/9)^{1/3} + o(1))$$

as a function of N, as N grows to infinity.

However, NFS for factoring is not the whole story.

- We mentioned NFS-DL over finite fields.
- SNFS is also a special case.
- and there are more weird ways to use NFS, including algebraic curves.

The complexity is $L(1/3, c + o(1))$ most of the time, but with varying c.

CEP and analogues

The Canfield-Erdős-Pomerance theorem is fundamentally important for the analysis of NFS.

CEP with the L function

A random integer $n \leq L_x(a, \alpha)$ is $L_x(b, \beta)$-smooth with probability:

$$L_x\left(a - b, -\frac{\alpha}{\beta}(a - b)(1 + o(1))\right).$$

This theorem has useful analogues in other contexts:

- Number fields: we saw it with NFS.
  The smoothness probability according to given norm bounds obeys the same rules.
- Polynomials over finite fields.
- Function fields.

This goes very heuristic, very quickly

Significant caveat about the extension of CEP to number fields or function fields:

- we can only say that this is valid for a fixed number/function field.
- the uniformity of the result across a range of number/function fields is a more difficult aspect. Note that this typically depends on our input!

As a general rule, a lot of these more subtle things are either proven, or at least more accessible to proofs in the polynomial / function field case than in the integer / number field case.

The more down-to-earth approach is not too much concerned about these subtleties, since anyway a lot of things are really heuristic.

Main questions for complexity analysis

- What is the target on the bottom of the diagram?
- What mathematical objects do we have on both sides?
- What objects are we trying to decompose, along which basis?
- What are the sizes (on both sides)?

Notations

We're reusing some notations that we already used for the analysis:

- $\log_{\log N} L_N(1/3, \delta)$ algebraic degree.
- $L_N(1/3, \beta)$ smoothness bound $\approx \#$ relations.
- $L_N(1/3, \alpha)$ bounds on a and b in $a - bx$.

Asymptotically, relation collection and linear algebra are always in the same ballpark, so we may think of α and β as being equal.

A comparison point

Recall the key asymptotic aspects of the analysis of GNFS.

- One rational side, and one algebraic side of degree

$$d = \log_{\log N} L_N(1/3, \delta + o(1)) = (\delta + o(1)) \left(\frac{\log N}{\log \log N} \right)^{1/3}.$$

- ...

A comparison point

- Coefficients of f_0 and f_1 both $\approx m = L_N(2/3, \frac{1}{\delta} + o(1))$.
- Two norms, of absolute values

$$|\operatorname{Res}(a - bx, f_0)| \approx L_N(2/3, \frac{1}{\delta} + o(1))$$
$$|\operatorname{Res}(a - bx, f_1)| \approx L_N(2/3, \frac{1}{\delta} + \alpha\delta + o(1)).$$

- Choose δ to minimize the product of norms, and eventually equate the remaining things to get the final asymptotic complexity estimate.

The size of the norms matters!

The key thing is really $\frac{1}{\delta} + \frac{1}{\delta} + \alpha\delta$. How it connects to the final complexity is not the important part of the machinery.

Plan

Introduction

SNFS

NFS-DL

The (defunct) Function Field Sieve

Extensions to other finite fields

The Multiple Number Field Sieve

$o(1)$

The Special Number Field Sieve (SNFS)

Remember: NFS started with a very exceptional case.

$$N = 2^{128} + 1, \quad f_0 = x - 2^{43}, \quad f_1 = x^3 + 2.$$

The coefficients of f_1 are extremely small.

- It is highly unlikely to find such a good polynomial pair with other random integers of that size.
- Argument: Say we constrain coefficients of f_0 and f_1 to absolute values 2^{43} and 2: only $2^{43\times 2+1\times 4} = 2^{90}$ integers.

Let us define a family of integers such that this sort of extraordinary things are possible.

An SNFS family

Let $(N_i)_{i \geq 0}$ be a family of integers such that...

- Informally: the NFS analysis works in an almost dream-like way.
- More formally: we'll try to do this dream analysis and come back to what the exact definition of the family should be.

An SNFS family

Let $(N_i)_{i \geq 0}$ be a family of integers such that...

- There exists a polynomial f_1 of degree $d = \log_{\log N} L_N(1/3, \delta + o(1))$, for some δ to be determined...
- with coefficients in $L_N(2/3, o(1))$...
- and such that there is a polynomial f_0 with coefficients just large enough so that $\mathrm{Res}(f_0, f_1) = N$. That means coefficients of size $L_N(2/3, 1/\delta)$.

Then in this case, we end up with:

$$|\mathrm{Res}(a - bx, f_0)| \approx L_N(2/3, \frac{1}{\delta} + o(1))$$
$$|\mathrm{Res}(a - bx, f_1)| \approx L_N(2/3, \alpha\delta + o(1)).$$

Consequence for the analysis

If all of this actually happens:

- $\frac{1}{\delta} + \alpha\delta$ is minimized with $\delta = 1/\sqrt{\alpha}$.
- The smoothness probability is $\frac{2\sqrt{\alpha}}{3\beta}$, and the constraint is:

$$2\alpha - \frac{2\sqrt{\alpha}}{3\beta} = \beta.$$

- We finish with $\alpha = \beta$ and $2\sqrt{\alpha} = 3\beta^2$, which leads $\beta = (2/3)^{2/3} = (4/9)^{1/3}$ and:

$$2\beta = (32/9)^{1/3}.$$

If we go backwards, this gives $\delta = (3/2)^{1/3}$.

What is SNFS

A family of SNFS integers

Let $(N_i)_{i \geq 0}$ be a family of integers such that for each N_i:

- There exists a polynomial f_1 of degree $d = \log_{\log N_i} L_{N_i}(1/3, (3/2)^{1/3} + o(1))$.
- with coefficients in $L_{N_i}(2/3, o(1))\ldots$
- and such that there is a polynomial f_0 with coefficients of size $L_{N_i}(2/3, 1/\delta)$ such that $\mathrm{Res}(f_0, f_1) = N_i$ (or a multiple of N_i).

Then the asymptotic cost of factoring an integer in this family is heuristically

$$L_{N_i}(1/3, (32/9)^{1/3} + o(1)).$$

Does this mean anything?

The previous definition is, in essence, an asymptotic one.

The question that gets often asked is

Let $N =$ blah. Is N an SNFS target?

The most useful answer is that if N can be obtained as the resultant of two polynomials with:

- an algebraic polynomial of a reasonable degree;
- an algebraic polynomial with super small coefficients;
- and a rational polynomial with "normal" coefficients

then YES, we're going to obtain something better than GNFS.

Examples of SNFS numbers

Integers of the form $r^e - s$ with r, s bounded by a constant are easily obtained as resultants of an SNFS polynomial pair.

Note however that the same is not true of their composite factors, which might be easier to tackle with GNFS!

Example: the 1214-bit integer $3^{766} + 1$ was factored using some ECM first, and then some (fairly easy) GNFS for the rest.

$$3^{766}+1=2\times5\times656822133606644237\times2102130222907676881\times68749406802433157741\times$$

$$27742388167909428696921049 69\times11913498249982150275039859349\times$$

$$30386584928990666330278777161645922849\times$$

$$30388934198614663079171397316787470704219965175523938580742484290 9\times$$

$$626943698188540315697357582114234580866611225002057825604732642929274209337\times$$

$$164321999730592745446748732152195678871262953538403127505289468338239875785 7$$

Plan

Introduction

SNFS

NFS-DL

The (defunct) Function Field Sieve

Extensions to other finite fields

The Multiple Number Field Sieve

$o(1)$

Analysis of NFS-DL

If we use the GNFS polynomial selection to compute discrete logarithms modulo a prime p, then the analysis (of the precomputation phase) is exactly the same, and N is replaced by p.

$$L_p(1/3, (64/9)^{1/3} + o(1)).$$

Analysis of NFS-DL (JL)

If d is the GNFS-factoring preferred degree, Joux-Lercier works with degrees D and $D+1$, with $D = d/2$.

- $\deg f_0 = D+1$, ridiculously small coefficients.
- $\deg f_1 = D$, coefficients around $p^{1/D}$.

Then for $|a| \approx |b| \approx A$, we have:

$$|\operatorname{Res}(a - bx, f_0)| \approx A^{D+1}$$

$$|\operatorname{Res}(a - bx, f_1)| \approx A^D p^{1/D}$$

And then

$$A^{2D+1} p^{1/D} \approx L_p(2/3, \alpha\delta + \frac{2}{\delta}).$$

Same as GNFS-factoring, but the balance is a bit weird:

- $\alpha\delta$ spreads out between both sides,
- and $\frac{2}{\delta}$ is only on one side.

Plan

Introduction

SNFS

NFS-DL

The (defunct) Function Field Sieve

Extensions to other finite fields

The Multiple Number Field Sieve

$o(1)$

The (defunct) Function Field Sieve

FFS is an interesting variation of NFS.

We trade all number fields for function fields.

A function field is attached to a curve, with all the algebraic geometry theory that comes with it.

We are interested in high degree extensions of very small finite fields (think $\mathbb{F}_2$).

Note

The Function Field Sieve is now almost completely superseded by the quasi-polynomial algorithm(s) for DLP in small-characteristic fields.

Analogy

$N \in \mathbb{Z}$, n bits.	$H \in \mathbb{F}_2[x]$, $\deg H = n$.
$N = \mathrm{Res}(f_0, f_1 \in \mathbb{Z}[x])$ $\phi = a - bx \in \mathbb{Z}[x]$ Number field defined by f_0	$H = \mathrm{Res}_y(f_0, f_1 \in \mathbb{F}_2[x, y])$ $\phi = a(x) - b(x)y \in \mathbb{F}_2[x, y]$ Function field defined by f_0 (curve defined by f_0)
Ideal $\langle a - b\alpha \rangle \to$ prime ideals Norm p	Function $\phi(x, y) \to$ places $2^{\deg p(x)}$
Smoothness of integers Canfield-Erdős-Pomerance	Smoothness of polynomials (various analogues)

Analogy

$N \in \mathbb{Z}$, n bits.	$H \in \mathbb{F}_2[x]$, $\deg H = n$.
$N = \mathrm{Res}(f_0, f_1 \in \mathbb{Z}[x])$ $\phi = a - bx \in \mathbb{Z}[x]$ Number field defined by f_0	$H = \mathrm{Res}_y(f_0, f_1 \in \mathbb{F}_2[x, y])$ $\phi = a(x) - b(x)y \in \mathbb{F}_2[x, y]$ Function field defined by f_0 (curve defined by f_0)
Ideal $\langle a - b\alpha \rangle \to$ prime ideals Norm p	Function $\phi(x, y) \to$ places $2^{\deg p(x)}$
Smoothness of integers Canfield-Erdős-Pomerance	Smoothness of polynomials (various analogues)
All n-bit integers are different	All $\mathbb{F}_{2^n}$ are isomorphic!

Everything works pretty much the same, and actually some tricky things become easier (cofactorization).

FFS allows crazy things

Polynomial selection for FFS is very special.

- Decide on your dream sizes and degrees.
- Choose $f_0(x, y)$ at random.
- Choose $f_1(x, y)$ at random.
- Repeat until $\mathrm{Res}_y(f_0, f_1)$ has an irreducible factor with the desired degree.

This means that FFS reaches the SNFS complexity.

FFS works also for other fields (see later).

FFS allows crazy things

More fun stuff: x and y really have symmetric roles in FFS.

The following works (assuming correct degrees and so on):

- Choose $f_0(x, y) = y - (\text{random polynomial in } x)$.
- Choose $f_1(x, y) = x - (\text{random polynomial in } y)$.
- Work with $\phi(x, y)$ with well chosen degrees.

In that situation, both f_0 and f_1 define function fields of degree 1, meaning: $\mathbb{F}_2[x]$, really!

(in practice, we might prefer the larger degree choices, so that we can benefit from somewhat distorted smoothness situations, via analogues of the Murphy-α value).

Coppersmith's algorithm

Coppersmith's algorithm is a special case of FFS (but was invented 10 years earlier), working with the polynomials $f_0 = y^{2^k} - R(x)$ and $f_1 = y - x^h$.

- This is very special because f_0 has degree a power of two. In algebraic geometry terms: a purely inseparable cover of $\mathbb{P}^1$.
- In more concrete terms, this makes it possible to share the factor base between the two sides, which is a unique feature.
- The complexity is $L_{2^n}(1/3, (32/9)^{1/3} + o(1))$ when 2^n is such that 2^k is really the sweet spot for the degree. The general FFS is much more flexible.

Yet, Coppersmith's algorithm is a great landmark, being the very first $L(1/3)$ algorithm!

NFS/FFS on more things

The general methodology of finding smooth things also works on the number fields and function fields that we consider.

- In a family of number fields defined by an NFS-like polynomials (very important assumption!), we can:
  - try to factor $a - b\alpha$.
  - and eventually do fancy things like class group and unit group computations in time $L(1/3)$.
- In a family of high genus algebraic curves with defining polynomials having a prescribed form ($\deg_x H$ and $\deg_y H$ both somewhat large), we can mimic FFS.
  - We can decompose into low-degree prime divisors.
  - Thanks to a descent procedure, we can solve the DLP!

Note that in these cases, we deal with only one side.

Plan

Introduction

SNFS

NFS-DL

The (defunct) Function Field Sieve

Extensions to other finite fields

The Multiple Number Field Sieve

$o(1)$

Multiple kinds of finite fields

Multiple ways to create a k-bit finite field:

- $\mathbb{F}_p$, for a k-bit prime p.
- $\mathbb{F}_{2^k} = \mathbb{F}_2[x]/f(x)$ for an irreducible $f \in \mathbb{F}_2[x]$ of degree $n = k$.
- More generally, $\mathbb{F}_{p^n}$, with $n \log_2 p \approx k$.

$\mathbb{F}_p$ and $\mathbb{F}_{2^k}$ are two opposite ends of the spectrum.

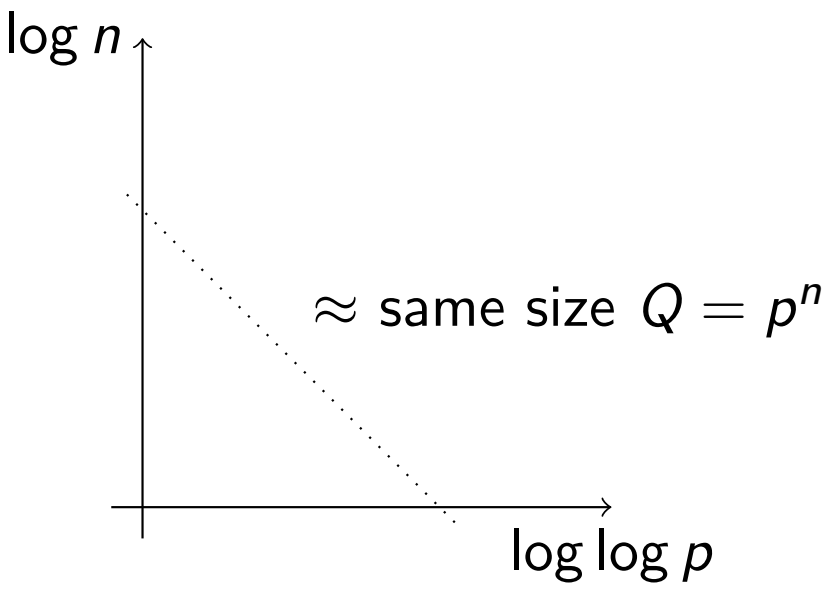
Fields of (roughly) the same size are along the line:

$$\log n + \log \log p = \log(k \log 2) = \text{constant}.$$

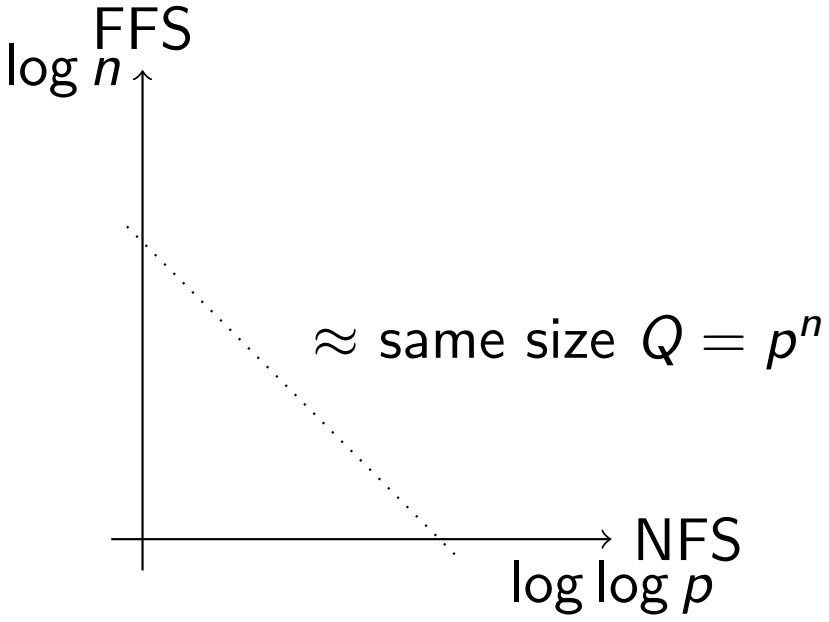
When we compare finite fields, we usually denote $Q = p^n$.

How does the DLP cost evolve as a function of Q?

Multiple kinds of finite fields



Multiple kinds of finite fields



- Prime fields: $\deg = 1$, characteristic is large: NFS.
- Binary fields: $p = 2$, everything is in the degree: FFS.
- Does it extend to more fields?
  What happens when $\log\log p \approx \log n$?

Plan

Extensions to other finite fields

- Small characteristic
- Large characteristic
- Medium characteristic
- Small characteristic again

Easy: extending FFS

(reminder: FFS is dead!)

It is easy to reformulate FFS with some characteristic p larger than 2, and see what constraints must be put on p.

The FFS range

Answer: FFS works as long as

$$p \leq L_{Q=p^n}(1/3, o(1))$$

which we can also interpret as

$$\begin{aligned} \log n &\geq 2 \log \log p - \log o(1) + 2 \log \log \log Q. \\ &\geq 2 \log \log p + \text{away from 0 quickly enough.} \end{aligned}$$

FFS range: asymptotic meaning

The FFS range

If $(\mathbb{F}_{Q_i})_{i \geq 0}$ is a family of finite fields, with $Q_i = p_i^{n_i}$, and F is a function such that:

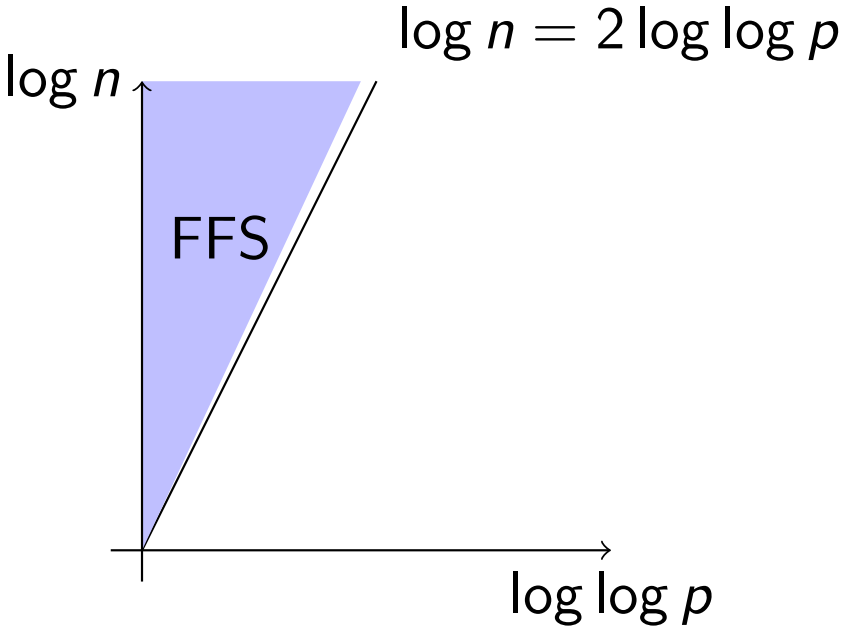
$$F(Q) \leq L_Q(1/3, o(1)) \text{ (as } Q \to \infty)$$
$$p_i \leq F(Q_i)$$

Then precomputation phase of the DLP in $(\mathbb{F}_{Q_i})_{i \geq 0}$ costs:

$$L_{Q_i}(1/3, (32/9)^{1/3} + o(1)).$$

This is informally called the small characteristic range.

FFS range: graphically



Plan

Extensions to other finite fields

- Small characteristic
- Large characteristic
- Medium characteristic
- Small characteristic again

Extending NFS

There are various ways to deal with fields of rather large characteristic, and small degree.

- Extension proposed early on by Schirokauer.
- Better version by generalizing Joux-Lercier.
  - Choose d according to Q, pick $D = d/2$.
  - Pick small degree $D+1$ polynomial with a degree n factor mod p (notation: $\psi(x)$).
  - Find small vector in lattice of dimension $D+1$ and determinant Q spanned by $p, \ldots, p^n x$, and $\psi(x)$ and its multiples.
  - This works as long as $n \leq D+1$, which translates to

$$p \geq L_Q(2/3, o(1)).$$

NFS range: asymptotic meaning

The NFS range

If $(\mathbb{F}_{Q_i})_{i \geq 0}$ is a family of finite fields, with $Q_i = p_i^{n_i}$, and F is a function such that:

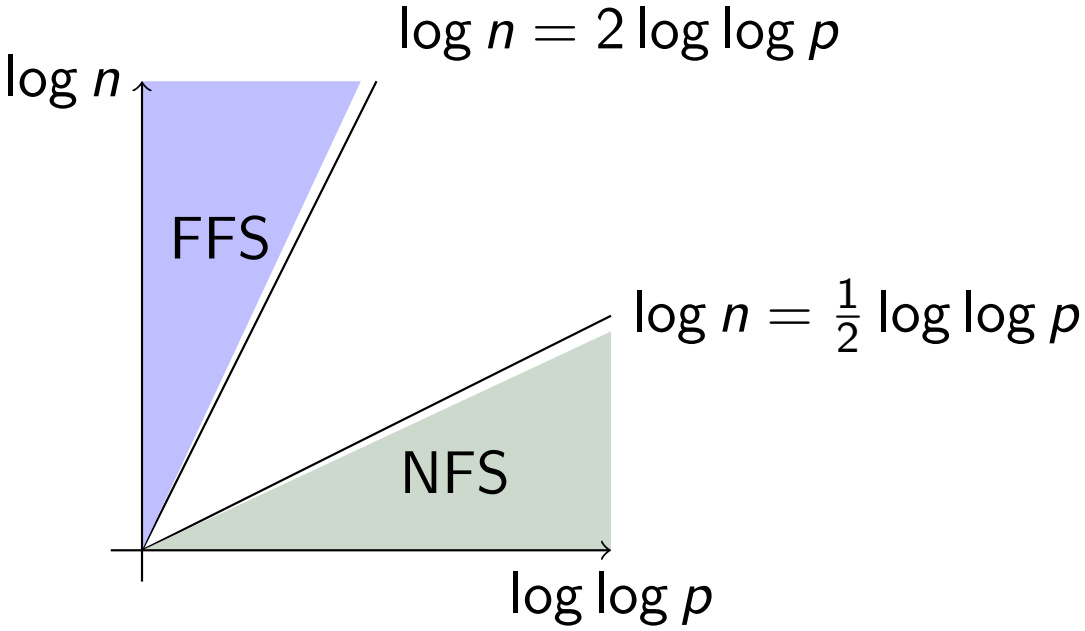
$$F(Q) \geq L_Q(2/3, o(1)) \text{ (as } Q \to \infty)$$
$$p_i \geq F(Q_i)$$

Then precomputation phase of the DLP in $(\mathbb{F}_{Q_i})_{i \geq 0}$ costs:

$$L_{Q_i}(1/3, (64/9)^{1/3} + o(1)).$$

This is informally called the large characteristic range.

NFS range: graphically



Plan

Extensions to other finite fields

- Small characteristic
- Large characteristic
- Medium characteristic
- Small characteristic again

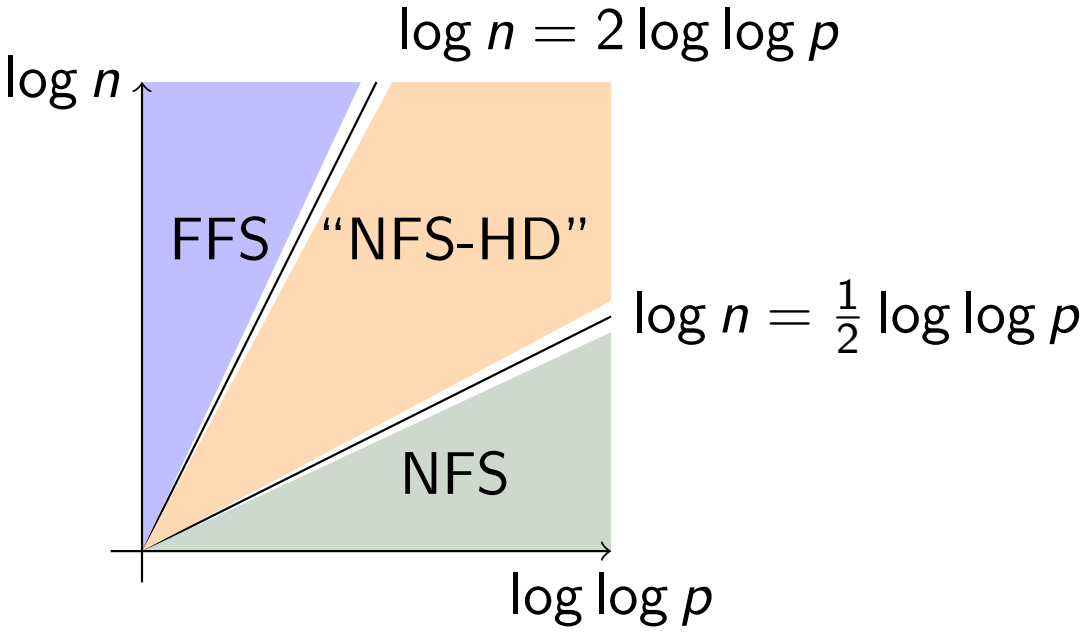
The remaining cases

The "middle case" was long thought as being much harder.

The first $L_Q(1/3)$ algorithm was found in 2006. Since in the middle case, p itself is less than $L_Q(2/3, o(1))$, it is actually quite simple:

- Take small f_0 with $\deg f_0 = n$, and simply take $f_1 = f_0 + p$.
- This is really as simple as it can get. Provided we are away from both boundaries, the complexity is $L_Q(1/3, (128/9)^{1/3} + o(1))$.
- This can be improved: other methods reach $L_Q(1/3, (96/9)^{1/3} + o(1))$

Boundaries are a nightmare



The complexity is easier to express away from the boundaries.

Many other variants

There are multiple variants of the above algorithms, with competing polynomial selection methods.

- Some adjustments for the boundary cases.
- Some finer-grain asymptotic estimates.
- Adaptation to composite degree.
- Adaptation to fields of SNFS-like characteristic.
- Use of towers of number fields.
- ...

Plan

Extensions to other finite fields

- Small characteristic
- Large characteristic
- Medium characteristic
- Small characteristic again

The quasi-polynomial algorithm

In the FFS range, a different approached was introduced in 2013, reaching eventually quasi-polynomial complexity.

- First QP complexity by BGJT14.
- Multiple variants, practical improvements, and experimental results.
- The algorithm is so different that proving stuff is possible!

Theorem (Quasi-polynomial algorithm)

Given any p and any n, the DLP in $\mathbb{F}_{p^n}^\times$ can be solved in expected time

$$C_{QP}(p^n) = (pn)^{2\log_2 n + O(1)}.$$

This means that FFS is all but dead, except very close to the boundary.

Small, medium, large

Finite fields are not T-shirts, unfortunately.

The small, medium, and large characteristic regimes are asymptotic considerations!!
They only make real sense for an infinite family of target fields.

In practice:

- Comparing $\log \log p$ and $\log n$ may give a rough idea.
- In cases where the answer seems unclear, the only way to decide which method is best is by experimenting!

Plan

Introduction

SNFS

NFS-DL

The (defunct) Function Field Sieve

Extensions to other finite fields

The Multiple Number Field Sieve

$o(1)$

Multiple number fields

MNFS is a variant of NFS, proposed by Coppersmith in 1993.

- Asymptotically, this beats NFS.

  $$L_N(1/3, ((64/9)^{1/3} = 1.923) + o(1)) \to L_N(1/3, 1.902 + o(1)).$$

- This was later adapted to the discrete logarithm context.

The MNFS diagram

$$\begin{tikzcd}
 & \mathbb{Z}[x] \arrow[dl] \arrow[d] \arrow[dd] \arrow[ddr] \arrow[ddrr] & & & \\
\mathbb{Q} \arrow[dr] & & \mathbb{Q}(\alpha_1) \arrow[dl] & \mathbb{Q}(\alpha_2) \arrow[dll] & \cdots \arrow[dlll] \quad \mathbb{Q}(\alpha_k) \arrow[dllll] \\
 & \mathbb{Z}/N\mathbb{Z} & & &
\end{tikzcd}$$

- Pick $a - bx$.
- See if $\mathrm{Res}(a - bx, f_0)$ is smooth (sieve).
- If yes, for $j \in [1, k]$:
  - See if $\mathrm{Res}(a - bx, f_j)$ is smooth.
  - If yes, we have a relation!
- Goal $(R(m))^2 \equiv (A_1(m) \cdots A_k(m))^2 \mod N$.

Basic idea

Informally, the idea is to "share" some computation.

Polynomial selection is not too hard.

- Pick a good polynomial pair (f_0, f_1).
- Derive f_j as $u_j f_0 + v_j f_1$ for a few small multipliers.
- We might as well adjust polynomial selection criteria a little bit.

Orders of magnitude

In Coppersmith's analysis, we need $L_N(1/3, 0.13\ldots)$ number fields (current range: this means dozens of number fields).

The rational side is easier, and is the one that is shared.

The algebraic sides have smaller smoothness bounds, yet the algebraic norms are bigger. Smoothness is rare!

What about practice?

Coppersmith's MNFS has never been used beyond very small proof-of-concept experiments.

Implementation and optimization is certainly not a trivial task, but not insurmountable either.

- Whether it is possible/useful to sieve is not entirely clear. Per-pair ECM, or product trees, can be ways to go.
- Linear algebra has to deal with a weird matrix which deserves special treatment.

The determination of the crossover point between the usual NFS and MNFS is completely open.

Plan

Introduction

SNFS

NFS-DL

The (defunct) Function Field Sieve

Extensions to other finite fields

The Multiple Number Field Sieve

$o(1)$

What is $o(1)$?

GNFS complexity is

$$L_N(1/3, (64/9)^{1/3} + o(1)).$$

Here, $o(1)$ is a function that tends to zero as N tends to ∞.

This is a lot different from an asymptotic complexity bound of the form $O(\text{blah})$. Here, we have a multiplicative unknown function that belongs to $L_N(1/3, o(1))$.

What can you put in $L_N(1/3, o(1))$?

$$L_N(1/3, o(1)) = \exp\left(o((\log N)^{1/3}(\log\log N)^{2/3})\right).$$

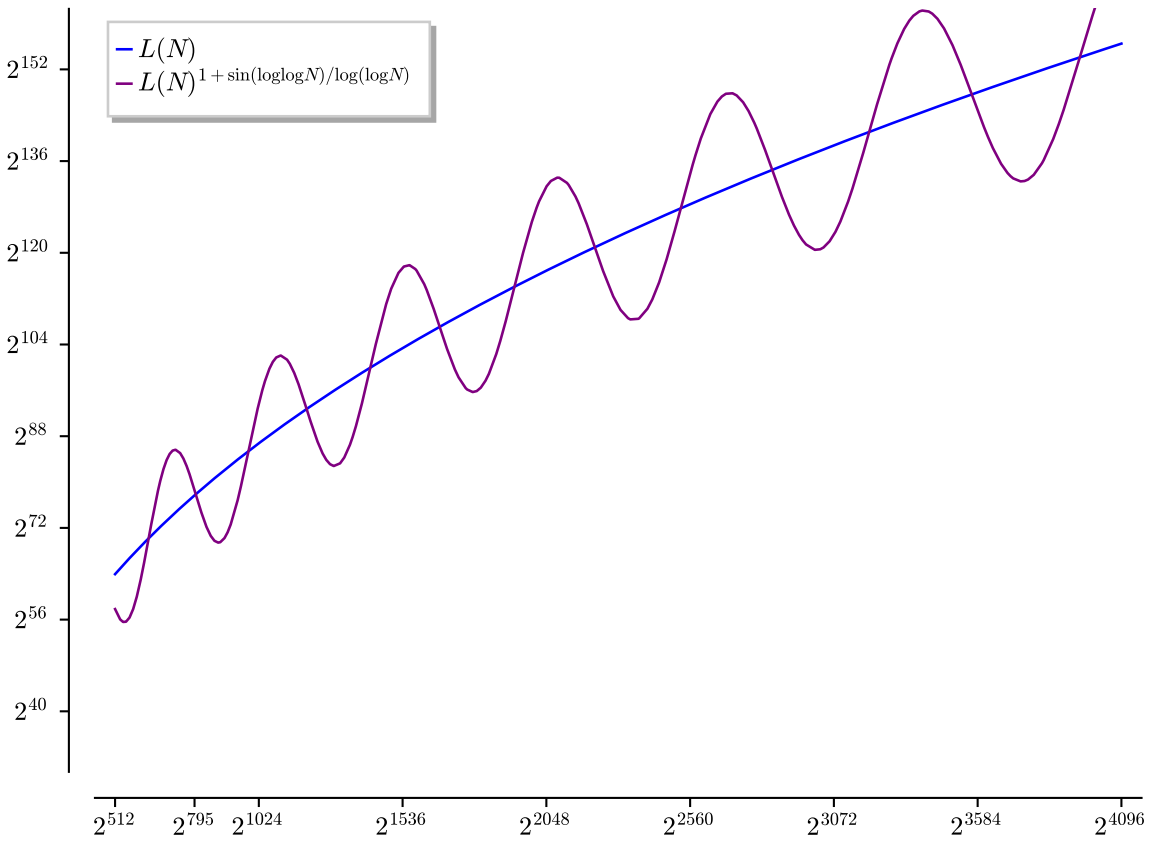
- Any polynomial in $\log N$ is in $L_N(1/3, o(1))$.
- Even super-polynomial $\exp((\log N)^x)$ for $x \leq 1/3$ is.
- Oscillatory behaviour can also be swallowed, e.g. the following completely imaginary multiplier is in $L_N(1/3, o(1))$:

$$2 - \cos\left(2\pi \cdot 3^{1/3}\left(\frac{\log N}{\log\log N}\right)^{1/3}\right).$$

(rationale: after all, what can we say of what happens when the "ideal" degree is halfway between two integers?)

We don't know the truth

Comparison of $L(N) = L_N(1/3, (64/9)^{1/3})$ with something that would be consistent with our generous asymptotic inaccuracy:



Does that matter?

FAQ: how expensive is RSA-$(n + 200)$ compared to RSA-n?

We often do as if the cost were $c \cdot L(N)$ for some unknown constant c, instead of the not really useful $L(N)^{1+o(1)}$.

In the 1990s, some words of caution were usually going with these bodacious approximations.

This has since become the accepted practice for establishing key size recommendations.

- The usual RSA key size $\leftrightarrow$ symmetric key size correspondence table comes from here.
- Shorter range extrapolations usually do:

$$\text{cost at } n + k \text{ bits} \approx \frac{L(2^{n+k})}{L(2^n)} \times \text{cost of academic record at } n \text{ bits.}$$

What if we're totally wrong?

Here are two expressions, both consistent with $L(N)^{1+o(1)}$.

$$g_0(N) = L(N) \qquad g_1(N) = L(N)^{1/(1+22/\log\log N)}.$$

- Based on $g_0(N) = L(N)$, we can compute $\frac{g_0(2^{2048})}{g_0(2^{512})} \approx 2^{28}$. We're tempted to think that RSA-2048 is several hundred millions times harder than RSA-512.
- Based on $g_1(N)$, we can compute $\frac{g_1(2^{2048})}{g_1(2^{512})} \approx 2^{8}$. The conclusion is very different!
- What gives?

Where does $o(1)$ come from?

While $o(1)$ is used in several places to simplify calculus, its origin can be traced to smoothness estimates.

Smoothness (Hildebrand)

Under some conditions on x, y:

$$\frac{\Psi(2^x, 2^y)}{2^x} = \rho(x/y)\left(1 + O\left(\frac{\log(1+\frac{x}{y})}{y}\right)\right).$$

Furthermore, the asymptotic behaviour of ρ is

$$\rho(u) = \exp(-u \log u \cdot (1 + o(1))) = u^{-u\cdot(1+o(1))}.$$

The main inaccuracy is the asymptotic expansion of $\log \rho$.

More terms of $\log \rho$

Can we compute more terms of $\log \rho$? YES!

With some computer algebra, computing hundreds of terms in the asymptotic expansion of $\log \rho$ is eminently possible, eventually leading to more terms replacing $o(1)$ in the NFS asymtptotic complexity.

We get a bivariate series in $\frac{\log \log \log N}{\log \log N}$ and $\frac{1}{\log \log N}$.

Is it useful?

What if we replace $o(1)$ by (say) a hundred terms of the series for our estimate?

Bad news: for $N < \exp(\exp(22)) \approx 2^{5 \text{ billion bits}}$, the series diverges!

The (log of the) NFS complexity is approximated by the first term of a divergent series (in the practical range).

Note that this is not unlike our example function $g_1(N)$.

The main problem is our attempt to write a self-contained asymptotic estimate for the probability of smoothness, seeking a closed formula. Explicit numerical estimates of smoothness probabilities should be of better value.

CSE291-14: The Number Field Sieve

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

Emmanuel Thomé

March 10, 2022

Part 10

Records and some recent stuff

A brief timeline

Hidden SNFS primes, up to kilobit size

Latest factoring and DLP records

Plan

A brief timeline

Hidden SNFS primes, up to kilobit size

Latest factoring and DLP records

|

- Late 1970s, Schroeppel: first analysis of CFRAC. $L()$ notation.
- 1980s, Pomerance + many: the quadratic sieve and its variants.
- 1983, Coppersmith: $L(1/3)$ algorithm for DLP in $\mathbb{F}_{2^n}$.
- 1985, Lenstra: ECM.
- 1986, Wiedemann: sparse linear algebra over finite fields.
- 1988, Pollard: Factoring with cubic integers.
- 1989, Lenstra, Manasse: factoring with electronic mail.
- 1990, Lenstra+others: The (special) number field sieve.
- 1990-1993, (many): GNFS.
  - Adleman: quadratic characters.
  - Pollard: lattice sieving.
  - Couveignes, Montgomery: square root.

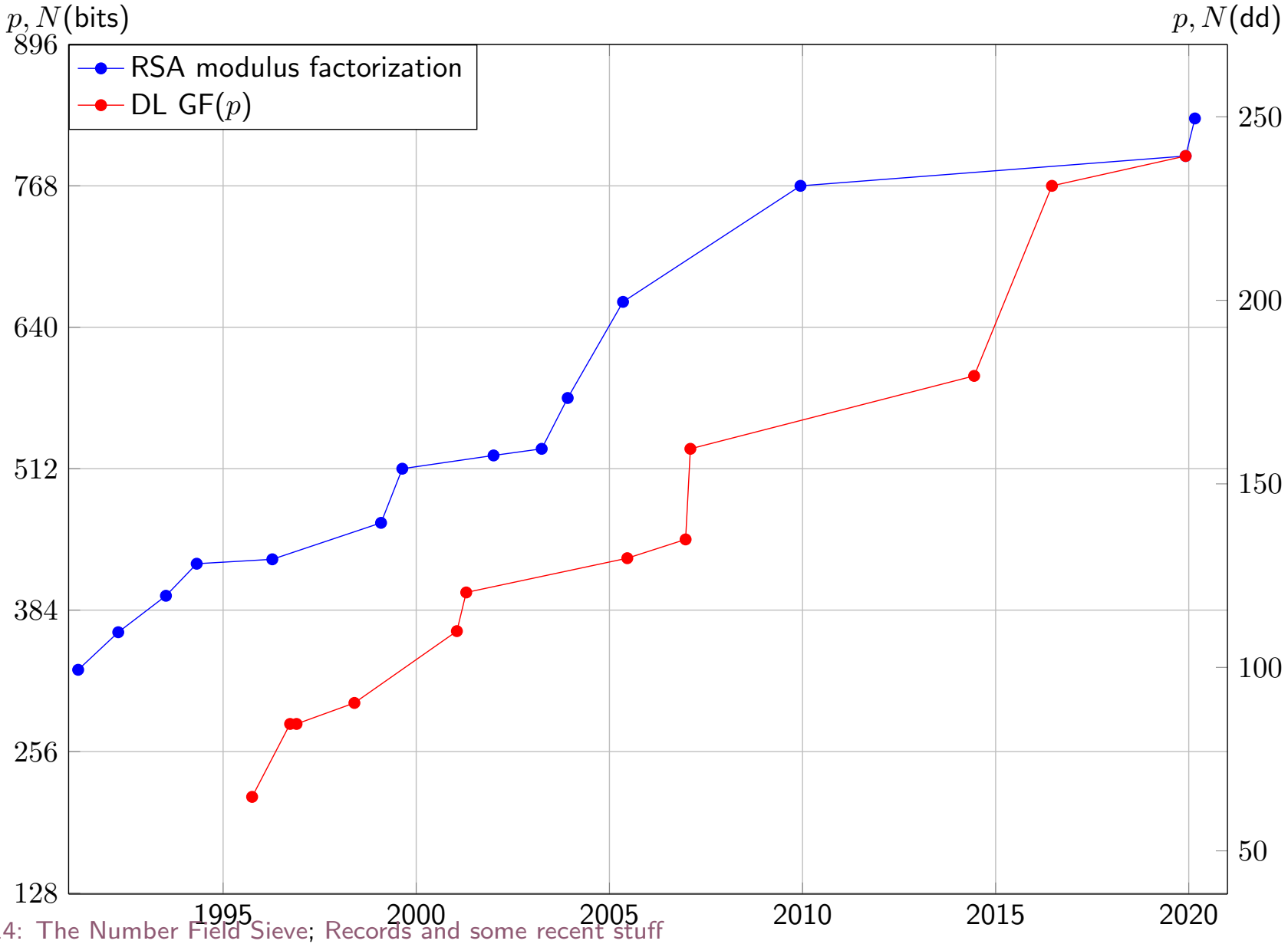
II

- 1992: early days of DSA.
- 1990-1993, Gordon: NFS for discrete logarithms.
- 1993, Schirokauer: Schirokauer maps.
- 1993, Coppersmith: the Multiple Number Field Sieve.
- 1994, Coppersmith: Block Wiedemann.
- 1994, Adleman: FFS.
- 1996, Weber: first practical NFS-DL computations.
- 1999, Lenstra+many: RSA-512.
- 2000, Bernstein: product trees.
- Early 2000s, Kleinjung: improvements to GNFS polynomial selection and to lattice sieving.
- 2002, Joux-Lercier: improvements to NFS-DL.

III

- 2002, Thomé: first use of Block Wiedemann for large computations.
- 2006, Joux-Lercier-Smart-Vercauteren: $L(1/3)$ for all fields.
- 2007: Kilobit SNFS.
- 2007, (many): development of Cado-NFS begins.
- 2008-2009: RSA-768.
- 2013: last big FFS computation $\mathbb{F}_{2^{809}}$.
- 2014: quasi-polynomial in $\mathbb{F}_{2^n}$.
- 2015: The Tower Number Field Sieve.
- 2015: Logjam. Individual logarithms are cheap.
- 2016: DLP-768 (232 digits).
- 2016: hidden SNFS kilobit DLP.
- 2018-2019: DLP-240, RSA-240, RSA-250.
- 2016-2021: several records for extension fields, finally using TNFS.

Timeline of records



Computational cost

Comparing computations is not a trivial task.

- Caveat: we only have published, academic records.
- All record computations generally use a scattered variety of resources.
- The only reasonable thing to do is to give what would have been the total cost if the computation had been run on one single resource type (and document that resource type).
- By definition, the unit of computational power depends on the point in time when the computation is done.
  For about 20 years, the trend of scaling all computational costs to unique computational unit (e.g. MIPS-years) has been all but abandoned.
- Hyperthreading complicates things even more. The usual approach is to count physical CPU time.

Plan

A brief timeline

Hidden SNFS primes, up to kilobit size

Latest factoring and DLP records

Hidden SNFS primes, up to kilobit size

Next few slides: takeaways from a computation done in 2016 (Fried, Gaudry, Heninger, Thomé, EC 2017).

- Relation to NFS in practice.
- We can get something that is cryptographically relevant, for a moderate computational cost.

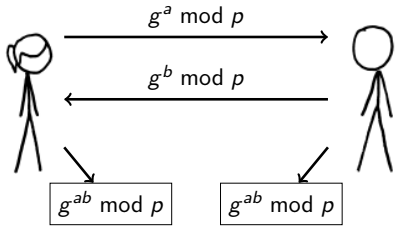
Plan

Hidden SNFS primes, up to kilobit size

- $(\mathbb{Z}/p\mathbb{Z})^*$ in crypto
- Backdooring primes
- Can one unveil the trapdoor?
- Computing DL mod 1024-bit primes with Cado-NFS
- Outcome and lessons

$(\mathbb{Z}/p\mathbb{Z})^*$, a.k.a. MODP groups

For Diffie-Hellman, for DSA: we've been using $(\mathbb{Z}/p\mathbb{Z})^*$ groups for decades.



Today (and whether we like it or not), FF DH and FF DSA are still very widespread.

- TLS
- SSH
- IPsec
- ...

Various measurements show their endured prevalence.

Who says which are the primes we use?

For a given key size, it should be fine if everybody uses the same p.

It is almost "One prime to rule them all"

De facto: a few primes are very widespread, promoted by:

- Standards (RFCs, ...).
- Implementations (Apache, OpenSSL, ...), or manufacturers of dedicated equipment (Cisco, Juniper, ...).

Who has a say on what primes go there?

The 1992 controversy

Beginning of the 1990s = early days of DSA.
Year 1992: panel at Eurocrypt, CACM article in July, article by Gordon at Crypto.

Is it a good idea to standardize primes?

Most important points raised by (Lenstra and) McCurley:

> So far, it has not been demonstrated that trapdoor moduli for the discrete logarithm problem can be constructed such that a) they are hard to detect, and b) knowledge of the trapdoor provides a quantifiable computational advantage for parameter sizes that could actually be computed by known methods, even with foreseeable machines.
>
> —K. S. McCurley, EC92 panel.

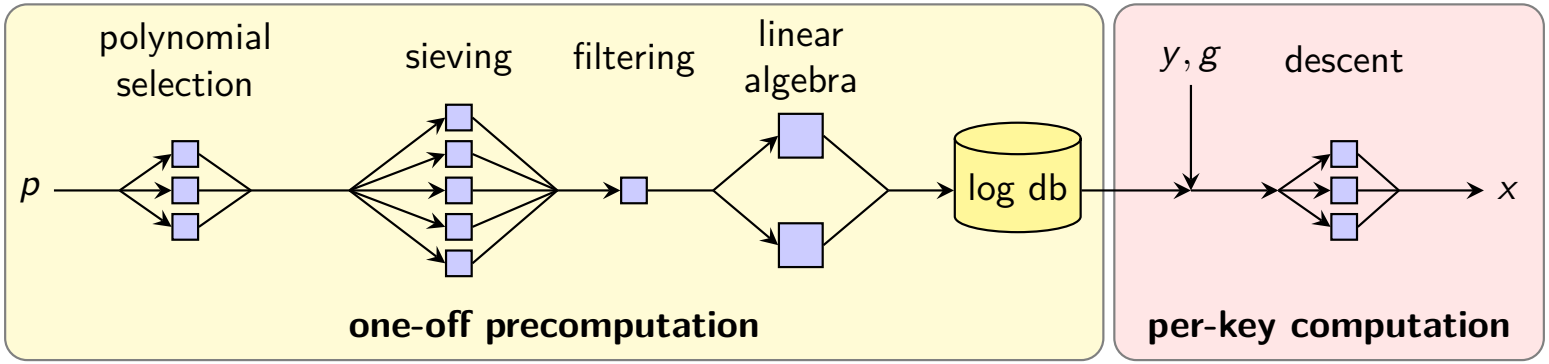
Part of the 1992 discussions focused on why a lower bound on p should be 1024 bits, not 512.

But the above points seemed to suffice to settle the discussion on the trapdoor: too conspicuous, and not a game-changer anyway.

1992 context

In 1992, NFS was still a new algorithm.

- Many practical challenges were yet to be solved.
- Linear algebra appeared a daunting task.
- This is even more true for NFS-DL: first preprint in April 1990.
- Algorithms for individual logs in NFS-DL took years to settle.



All these hurdles have long been passed.

Interlude

Some of the implications of the practice of NFS-DL took a long time to percolate and reach the use of FF-DLP in practice.

Until Logjam, many people overlooked the difference between precomputation (offline) and individual log (online) time for NFS-DL.

		Precomputation core-years	Individual Log core-time
RSA-512	[Cavallar et al. 1999]	1	—
DH-512	[Adrian et al. 2015]	10	10 mins
RSA-768	[Kleinjung et al. 2009]	1,000	—
DH-768	[Kleinjung et al. 2016]	5,000	2 days
RSA-240	[Boudot et al. 2020]	900	—
DH-240	[Boudot et al. 2020]	3,000	1 day

What does it look like in 2016?

Many primes are found in the wild with unknown provenance.

We cannot tell whether they have been chosen with malice.

- 1024-bit primes in Apache http software;
- RFC 5114 primes ($\geq$1024 bits);
- 2048-bit prime used in IACR 2015 BOD election;
- ...

We wish to investigate how trapdoors can be designed, and how easier they make the DLP computations.

RFC5114

```
Network Working Group                                        M. Lepinski
Request for Comments: 5114                                       S. Kent
Category: Informational                                  BBN Technologies
                                                             January 2008
```

Additional Diffie-Hellman Groups for Use with IETF Standards

2. Additional Diffie-Hellman Groups

This section contains the specification for eight groups for use in IKE, TLS, SSH, etc. There are three standard prime modulus groups and five elliptic curve groups. All groups were taken from publications of the National Institute of Standards and Technology, specifically [DSS] and [NIST80056A]. Test data for each group is provided in Appendix A.

2.1. 1024-bit MODP Group with 160-bit Prime Order Subgroup

The hexadecimal value of the prime is:

```
p = B10B8F96 A080E01D DE92DE5E AE5D54EC 52C99FBC FB06A3C6
    9A6A9DCA 52D23B61 6073E286 75A23D18 9838EF1E 2EE652C0
    13ECB4AE A9061123 24975C3C D49B83BF ACCBDD7D 90C4BD70
    98488E9C 219A7372 4EFFD6FA E5644738 FAA31A4F F55BCCC0
    A151AF5F 0DC8B4BD 45BF37DF 365C1A65 E68CFDA7 6D4DA708
    DF1FB2BC 2E4A4371
```

The hexadecimal value of the generator is:

```
g = A4D1CBD5 C3FD3412 6765A442 EFB99905 F8104DD2 58AC507F
    D6406CFF 14266D31 266FEA1E 5C41564B 777E690F 5504F213
    160217B4 B01B886A 5E91547F 9E2749F4 D7FBD7D3 B9A92EE1
    909D0D22 63F80A76 A6A24C08 7A091F53 1DBF0A01 69B6A28A
    D662A4D1 8E73AFA3 2D779D59 18D08BC8 858F4DCE F97C2A24
    855E6EEB 22B3B2E5
```

The generator generates a prime-order subgroup of size:

```
q = F518AA87 81A8DF27 8ABA4E7D 64B7CB9D 49462353
```

Here is p
Here is $q \mid (p-1)$
Please use for crypto.

Supported by:

- 900K (2.3%) HTTPS hosts
- 340K (13%) IPsec hosts

Plan

Hidden SNFS primes, up to kilobit size

- $(\mathbb{Z}/p\mathbb{Z})^*$ in crypto
- Backdooring primes
- Can one unveil the trapdoor?
- Computing DL mod 1024-bit primes with Cado-NFS
- Outcome and lessons

NFS goes very well in special cases

For arbitrary p (or N for factoring), there's a lower bound on how small f and g can be (e.g. by counting).

Factoring knows about especially easy integers

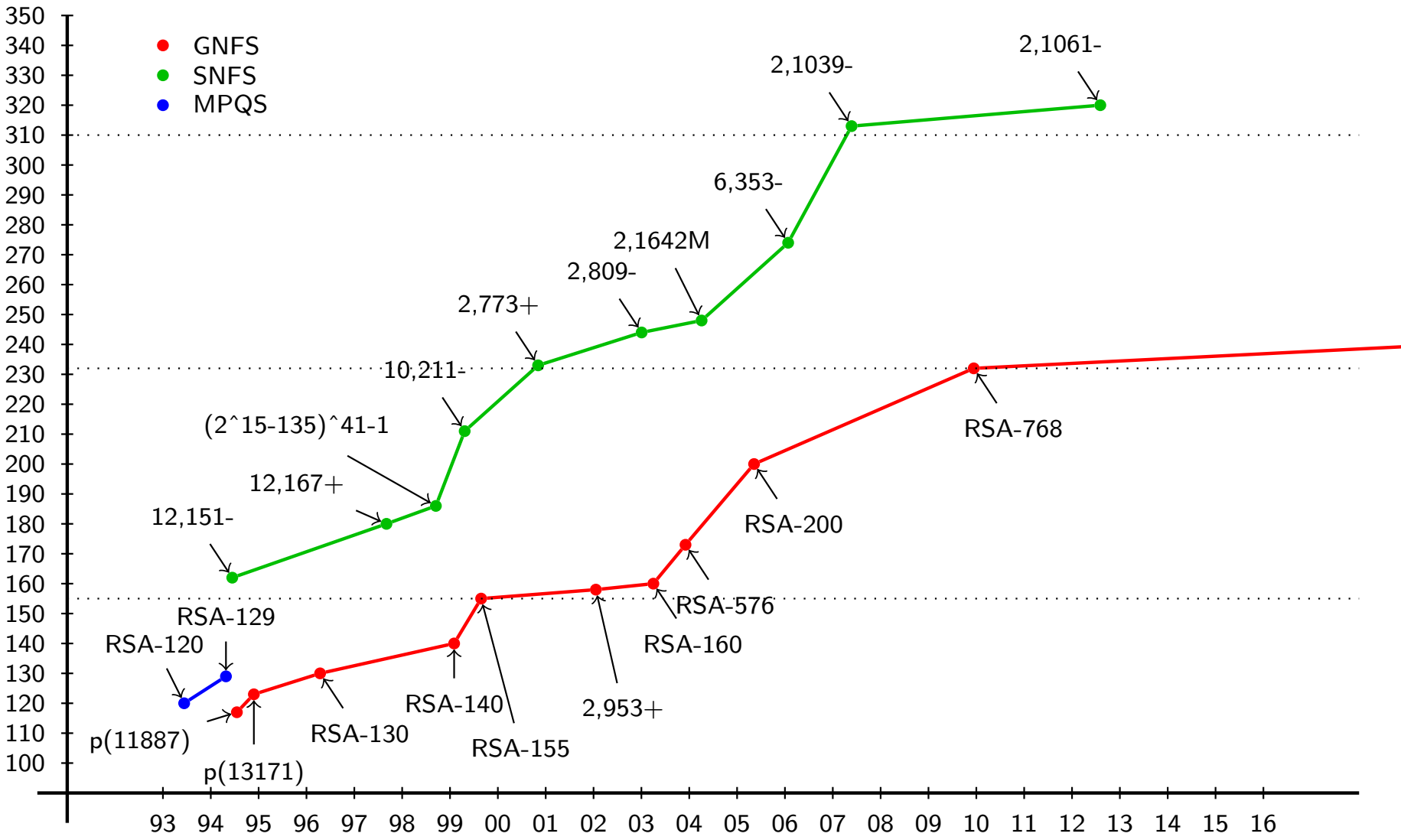
Say if $N = r^e - s$ with r, s small. We pick:

- $f = r^{e \bmod k} X^k - s$ with small k to our liking,
- and $g = X - r^{\lfloor e/k \rfloor}$

This is the special NFS (SNFS, as opposed to GNFS).
Applies in particular to the Cunningham tables.
Likewise, we have an SNFS-DL for "attacker-friendly primes".

Next: timeline of factoring records for SNFS and GNFS, compared.

SNFS versus GNFS (factoring) records



We may ease our task even more

DLP mod attacker-friendly primes may be well within reach while DLP mod "normal" primes of the same size is still remote.

But there is more !

So-called DSA primes

DSS promotes primes with a moderate size subgroup of $(\mathbb{Z}/p\mathbb{Z})^*$
E.g. 1024-bit prime p with 160-bit prime q dividing $p-1$.
RFC5114 promotes examples of such primes.

If a DSA prime is also attacker-friendly, then (S)NFS-DL linear algebra is modulo q, not modulo $p-1$. This is an additional win for the attacker.

Fantasy of a body tinkering with standards

What if we can design attacker-friendly DSA primes?

Heidi hides her polynomials

Heidi, a mischievous protocol designer

- chooses secret polynomials f and g;
- publishes $p = \mathrm{Res}(f, g)$ and pushes for its widespread use.
- p has a (say) 160-bit prime factor q.
- Knowing f and g, Heidi can run SNFS-DL.
  Linear algebra is to be done $\mod q$.

D. Gordon (Crypto 1992): a way to do just that.
This construction is still efficient today.

How to trapdoor a DSA prime [Gordon92]

Want to construct primes p, q such that $q \mid p - 1$ and

$$f(x) = f_6 x^6 + \cdots + f_0, \qquad g(x) = g_1 x - g_0$$

such that $p \mid \mathrm{Res}(f, g)$.

Slow algorithm:

1. Choose random f, g.
2. Check if $p = \mathrm{Res}(f, g)$ prime.
3. Factor $p - 1$ with ECM.
4. Repeat until $p - 1$ has 160-bit prime factor.

How to trapdoor a DSA prime [Gordon92]

Want to construct primes p, q such that $q \mid p - 1$ and

$$f(x) = f_6 x^6 + \cdots + f_0, \qquad g(x) = g_1 x - g_0$$

such that $p \mid \mathrm{Res}(f, g)$.

Better algorithm:

1. Choose $f(x)$, q, g_0.
2. Want $q \mid \mathrm{Res}(f(x), g_1 x - g_0) - 1$.
3. Compute $G(g_1) = \mathrm{Res}(f(x), g_1 x - g_0) - 1$.
4. Compute root $G(r) \equiv 0 \bmod q$; $g_1 = r + cq$.
5. Repeat until $\mathrm{Res}(f(x), g_1 x - g_0)$ prime.

Note that this implies that the target size for g_1 is larger than q.

Plan

Hidden SNFS primes, up to kilobit size

- $(\mathbb{Z}/p\mathbb{Z})^*$ in crypto
- Backdooring primes
- Can one unveil the trapdoor?
- Computing DL mod 1024-bit primes with Cado-NFS
- Outcome and lessons

Can we tell whether p has a trapdoor?

This looks nice for Heidi, but won't work if the primes she pushes for is conspicuously weird.

E.g. you shouldn't do DLP in $(\mathbb{Z}/p\mathbb{Z})^*$ for $p = 2^{1024} - 105$.

However if Heidi allows herself sufficient freedom in choosing the coefficients of f, then p looks innocuous.

Detecting the trapdoor

- "Easy" if $g(x) = x + g_0$ or similar.
  1. Brute force leading coefficient f_d of f.
  2. Search values of g_0 near $(p/f_d)^{1/d}$.
  3. Use LLL to search for other small coefficients of f.
- If $g(x) = g_1 x + g_0$ don't know a way that doesn't require brute forcing coefficients of f or g.
- **Open Problem**: Given $p = \mathrm{Res}(f, g_1 x + g_0)$ and f has small coefficients, find f, g.

Crafting the trapdoor

- 1992-era parameters: 512-bit p, 160-bit q
  - Forces $\deg f = 3$; suboptimal for NFS.
  - f chosen from small set so not well hidden.

> ... this trap only makes sense for primes up to [600 bits]. Furthermore, this kind of trap can be detected, although this requires more work than an average user will be able to invest.
>
> —A. Lenstra, EC92 Panel.

- DSA standard: optional "verifiably random" prime generation.

Crafting the trapdoor in the modern era

Gordon's trapdoor construction remains best construction.

- Modern parameters: 1024-bit p, 160-bit q
  - Can choose $\deg f = 6$, optimal for NFS.
  - Choose $|f_i| \approx 2^{11}$.
  - Brute force search to find $f \approx 2^{80} \approx$ cost of Pollard rho for q.
  - Don't know of better way to detect trapdoor.

Exploiting the trapdoor in the modern era

We generated a target 1024-bit prime in 12 core-hours.

The public part:

$$
\begin{aligned}
p &= 163323987240443679101402070093049155030989439806917519173580070791569227728932850358498862854399351423733697660534800194492724828721314980248259450358792069235991826588944200440687094136669506349093691768902440555341493237296555254247379422702221515929837629813600812082006124038089463610239236157651252180491 \\
q &= 112032031118307126198843367430018230602909671047 3 ,
\end{aligned}
$$

and Heidi's hidden polynomials:

$$
\begin{aligned}
f &= 1155\,x^6 + 1090\,x^5 + 440\,x^4 + 531\,x^3 - 348\,x^2 - 223\,x - 1385 \\
g &= 567162312818120432489991568785626986771201829237408\,x \\
&\quad -663612177378148694314176730818181556491705934826717 .
\end{aligned}
$$

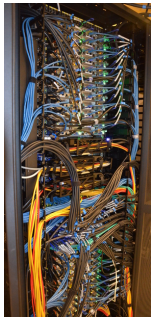
Plan

Hidden SNFS primes, up to kilobit size

- $(\mathbb{Z}/p\mathbb{Z})^*$ in crypto
- Backdooring primes
- Can one unveil the trapdoor?
- Computing DL mod 1024-bit primes with Cado-NFS
- Outcome and lessons

Computation timings

We used only two clusters. Linear algebra was done on higher-end hardware with fast interconnect (Infiniband FDR 56Gbps, Cisco UCS 40Gbps)



Used parameters $m = 24$, $n = 12$ for block Wiedemann.

	sieving	linear algebra			individual log
		sequence	generator	solution	
cores	≈3000	2056	576	2056	500–352
CPU time (core)	240 years	123 years	13 years	9 years	10 days
calendar time	1 month	1 month			80 minutes

Computation went smoothly, of course

On the bright side, our computation took almost exactly the predicted time (both CPU time and wall-clock time).

Yet we did have our share of mishaps.

- UPenn: deal with cluster being kicked out of the computer room with 2-day notice, and moved 2 miles south with no decent network connection.
  raspberry pi's + university wifi + ...
- Nancy: of course not everything was coded yet when we started...

Comparison with other computations

Our computation: $\log_2 p \approx 1024$, $\log_2 q \approx 160$: 400 core-years.

Safe prime of the same size: expect lin.alg $7\times$ harder.

768-bit GNFS-DLP (Kleinjung et al., 2017): ≈ 5000 core-years.

2048-bit trapdoored p, like here: expect similar to GNFS-1340.

Some conspicuous SNFS primes found in the wild ($q = (p-1)/2$):

- $p = 2^{1024} - 1093337$: doable but harder than our p!
  - polynomial not as good as ours: α value is bad; sieving $3\times$ harder
  - linear algebra mod $q = (p-1)/2$.
- $p = 2^{784} - 2^{28} + 1027679$ (exercise) ≈ 60 core-years.

Plan

Hidden SNFS primes, up to kilobit size

- $(\mathbb{Z}/p\mathbb{Z})^*$ in crypto
- Backdooring primes
- Can one unveil the trapdoor?
- Computing DL mod 1024-bit primes with Cado-NFS
- Outcome and lessons

Danger of over-interpreting the result

We have found no poorly-hidden trapdoored prime in the wild.

- either because the trap was well hidden (after all, the recipe dates back to 1992).
- or because there was no trapdoor at all.

If Heidi designed RFC5114 and suggested the primes used in Apache and so on, she might be caught red-handed in the future. There is no plausible deniability.

Not clear that Heidi is at ease about such a scenario.

Anyway, now the RFCs have ditched the RFC5114 primes.

Lessons

1024-bit DLP can be easy for an attacker that maliciously chose the prime to his liking.

We found no easy way to prove that a trapdoor is present.

Verifiable randomness is necessary.

- It's not even the question of accusing anyone of wrongdoing. We found no smoking gun.
- But the lack of verifiable randomness is a major hindrance for trust in cryptographic standards.

Of course **people still get it awfully wrong**.

E.g. the standardized French and Chinese elliptic curves are really really bad to this regard.

Plan

A brief timeline

Hidden SNFS primes, up to kilobit size

Latest factoring and DLP records

Pre-2019 state-of-the-art

RSA-768: 02/2008–12/2009. About 1,500 core-years in total.

- large-scale improvement compared to the previous RSA-200 record. RSA-768 was a much larger undertaking.
- coordination of multiple computer clusters.
- fancy block Wiedemann, multi-country.

DLP-768: 06/2016: About 5,300 core-years.

- Much more efficient than previous 180-digit record thanks to Joux-Lercier polynomial selection.
- First apparent involvement of government computational resources (BSI) in an academic computation.

Recent results

Our DLP-240 computation was faster than the previous DLP-768 computation, while of course we tackled a harder challenge.

This is also true if we try to measure the cost on the same hardware that was used for the DLP-768 computation.

What are the important things in this computation?

A simple rule of thumb

We look for smoothness with respect to a bound L.

A prime should appear either often, or very rarely.

- below some bound B, we strive to find all pairs (a, b) such that primes below B appear in the factorization.
  We do this with sieving.
- "large primes" (LPs) such that $B \leq p < L$:
  allowed if we happen to find them.
  Limit to a few LPs per relation (e.g., 2, sometimes 3).

The relations that we like to see

✓	$5^2 \cdot 11 \cdot 23 \cdot 287093 \cdot 870953 \cdot 20179693 \cdot 28306698811 \cdot 47988583469$	$2^3 \cdot 5 \cdot 7 \cdot 13 \cdot 31 \cdot 61 \cdot 14407 \cdot 26563253 \cdot 86800081 \cdot 269845309 \cdot 802234039 \cdot 1041872869 \cdot 5552238917 \cdot 12144939971 \cdot 15856830239$
✓	$3 \cdot 1609 \cdot 77699 \cdot 235586599 \cdot 347727169 \cdot 369575231 \cdot 9087872491$	$2^3 \cdot 3 \cdot 5 \cdot 13 \cdot 19 \cdot 23 \cdot 31 \cdot 59 \cdot 239 \cdot 3989 \cdot 7951 \cdot 2829403 \cdot 31455623 \cdot 225623753 \cdot 811073867 \cdot 1304127157 \cdot 78955382651 \cdot 129320018741$
✓	$5 \cdot 1381 \cdot 877027 \cdot 15060047 \cdot 19042511 \cdot 11542780393 \cdot 13192388543$	$2^4 \cdot 5 \cdot 13 \cdot 31 \cdot 59 \cdot 823 \cdot 2801 \cdot 26539 \cdot 2944817 \cdot 3066253 \cdot 87271397 \cdot 108272617 \cdot 386616343 \cdot 815320151 \cdot 1361785079 \cdot 12322934353$
✓	$2^3 \cdot 5^2 \cdot 173 \cdot 971 \cdot 613909489 \cdot 929507779 \cdot 1319454803 \cdot 2101983503$	$2^7 \cdot 3^2 \cdot 5 \cdot 29 \cdot 1021 \cdot 42589 \cdot 190507 \cdot 473287 \cdot 31555663 \cdot 654820381 \cdot 802234039 \cdot 19147596953 \cdot 23912934131 \cdot 52023180217$
✗	$2^2 \cdot 15193 \cdot 232891 \cdot 19514983 \cdot 139295419 \cdot 540260173 \cdot 606335449$	$2^2 \cdot 3^4 \cdot 13 \cdot 19 \cdot 74897 \cdot 1377667 \cdot 55828453 \cdot 282012013 \cdot 802234039 \cdot 3350122463 \cdot 35787642311 \cdot 37023373909 \cdot 128377293101$
✗	$2^2 \cdot 5^4 \cdot 439 \cdot 1483 \cdot 13121 \cdot 21383 \cdot 67751 \cdot 452059523 \cdot 33099515051$	$2^2 \cdot 3^3 \cdot 11 \cdot 13 \cdot 19 \cdot 5023 \cdot 3683209 \cdot 98660459 \cdot 802234039 \cdot 1506372871 \cdot 4564625921 \cdot 27735876911 \cdot 32612130959 \cdot 45729461779$

small primes: abundant $\rightarrow$ dense column in the matrix

large primes: rare $\rightarrow$ sparse colum, limit to 2 or 3 on each side.

The relations that we like to see

✓	$5^2 \cdot 11 \cdot 23 \cdot 287093 \cdot 870953 \cdot 20179693 \cdot 28306698811 \cdot 47988583469$	$2^3 \cdot 5 \cdot 7 \cdot 13 \cdot 31 \cdot 61 \cdot 14407 \cdot 26563253 \cdot 86800081 \cdot 269845309 \cdot 802234039 \cdot 1041872869 \cdot 5552238917 \cdot 12144939971 \cdot 15856830239$
✓	$3 \cdot 1609 \cdot 77699 \cdot 235586599 \cdot 347727169 \cdot 369575231 \cdot 9087872491$	$2^3 \cdot 3 \cdot 5 \cdot 13 \cdot 19 \cdot 23 \cdot 31 \cdot 59 \cdot 239 \cdot 3989 \cdot 7951 \cdot 2829403 \cdot 31455623 \cdot 225623753 \cdot 811073867 \cdot 1304127157 \cdot 78955382651 \cdot 129320018741$
✓	$5 \cdot 1381 \cdot 877027 \cdot 15060047 \cdot 19042511 \cdot 11542780393 \cdot 13192388543$	$2^4 \cdot 5 \cdot 13 \cdot 31 \cdot 59 \cdot 823 \cdot 2801 \cdot 26539 \cdot 2944817 \cdot 3066253 \cdot 87271397 \cdot 108272617 \cdot 386616343 \cdot 815320151 \cdot 1361785079 \cdot 12322934353$
✓	$2^3 \cdot 5^2 \cdot 173 \cdot 971 \cdot 613909489 \cdot 929507779 \cdot 1319454803 \cdot 2101983503$	$2^7 \cdot 3^2 \cdot 5 \cdot 29 \cdot 1021 \cdot 42589 \cdot 190507 \cdot 473287 \cdot 31555663 \cdot 654820381 \cdot 802234039 \cdot 19147596953 \cdot 23912934131 \cdot 52023180217$
✗	$2^2 \cdot 15193 \cdot 232891 \cdot 19514983 \cdot 139295419 \cdot 540260173 \cdot 606335449$	$2^2 \cdot 3^4 \cdot 13 \cdot 19 \cdot 74897 \cdot 1377667 \cdot 55828453 \cdot 282012013 \cdot 802234039 \cdot 3350122463 \cdot 35787642311 \cdot 37023373909 \cdot 128377293101$
✗	$2^2 \cdot 5^4 \cdot 439 \cdot 1483 \cdot 13121 \cdot 21383 \cdot 67751 \cdot 452059523 \cdot 33099515051$	$2^2 \cdot 3^3 \cdot 11 \cdot 13 \cdot 19 \cdot 5023 \cdot 3683209 \cdot 98660459 \cdot 802234039 \cdot 1506372871 \cdot 4564625921 \cdot 27735876911 \cdot 32612130959 \cdot 45729461779$

small primes: abundant $\rightarrow$ dense column in the matrix
large primes: rare $\rightarrow$ sparse colum, limit to 2 or 3 on each side.

Before linear algebra, the filtering step tries to do as many cheap combinations as it can, so as to get a smaller matrix.

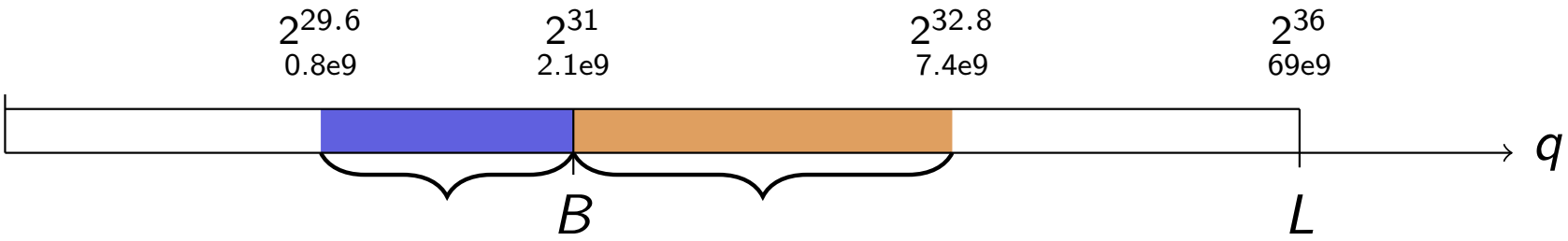
The combination cost

Relations with 2 LPs or less are a blessing.

- They easily participate in cheap combinations.
- If we have only 2-LP relations, filtering will get rid of most of them.
  We are left with a number of primes to combine that is roughly the number of primes below B.
- Caveat: two sides to deal with.

We must pay attention to the special-q as well! How does it compare to B?

Strategy for RSA-240



$q < B$: allow
2 LPs on side 0,
3 LPs on side 1.

$B \leq q < L$: allow 2 LPs on each side.
(q counts as an extra LP on side 1.)

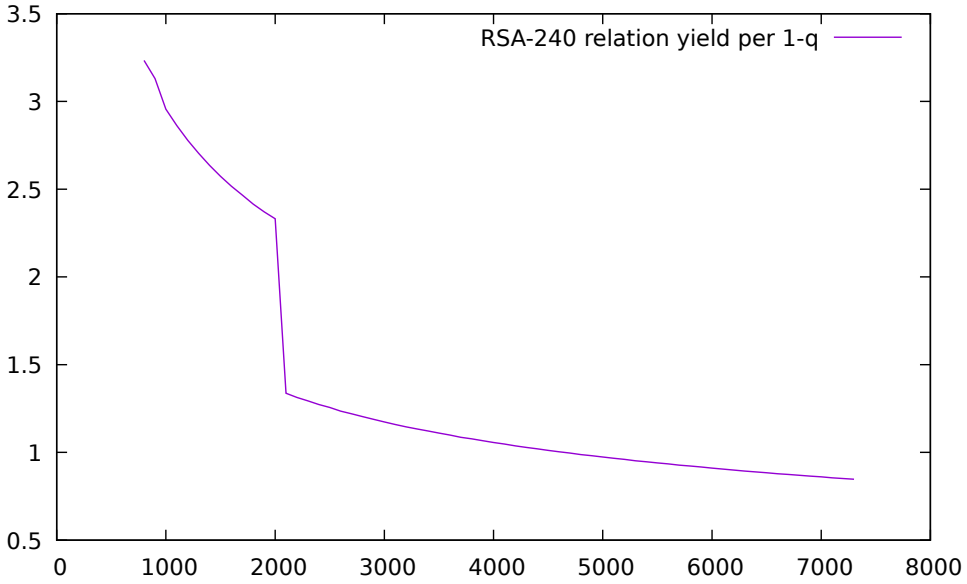
This strategy makes it easy to get rid of most $p \geq B$ on side 0 before we enter linear algebra proper.

We still have many on side 1, but that is not too bad because linear algebra in the factoring context is reasonable.

Unstable yield, but we know what we're doing

Note that we change the relation collection criteria radically depending on q!

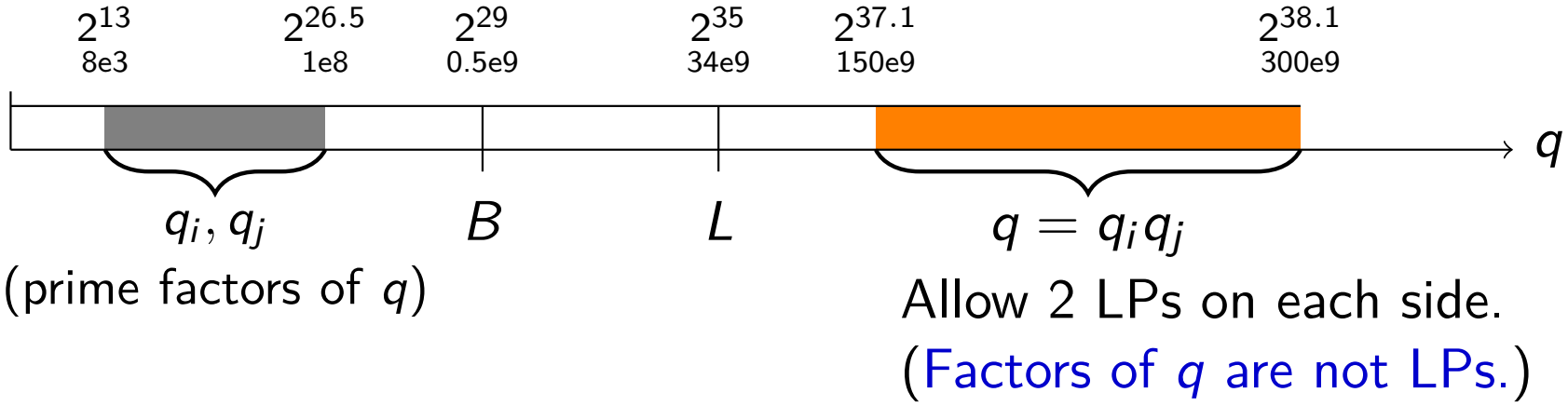
The yield changes (plot from this data)



This is expected, and fits well with our goal!

Strategy for DLP-240

For DLP-240, we used composite q, to avoid the disadvantage of having q in the LP range.



This strategy was efficient in reducing the combination work to essentially primes $p < B$ only.

Alternative to sieving

In all cases, we have an "easy" and a "hard" side, depending on the size of the norms.

Relation collection is about restricting attention to a subset of (a, b)'s. There's one side that we have to do first.

If we do the "hard" side first, not very many of the (a, b) pairs are left.

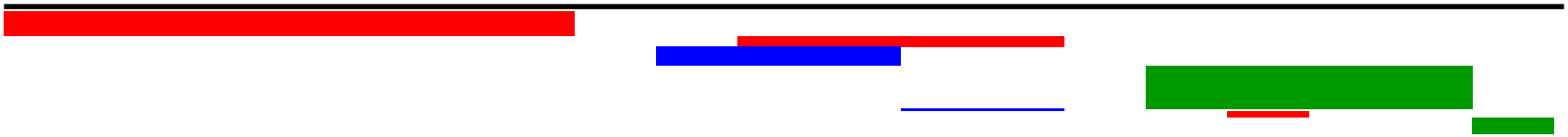
- In some situations, this selection is so drastic that it may make sense to process these few pairs one by one instead of doing sieving on the other side.
- This is exactly what we did for the previous records, using product trees (for some parameter ranges).

Summary of relation collection

- Tried-and-true techniques do work. Many low-level improvements in the deep aspects of special-q sieving.
- Seldom used techniques such as composite special-q or batch smoothness detection played a key role.
- We tailored the relation collection step so that the subsequent filtering step works well. (choice of q ranges, number of LPs.)

Relation collection is by far the most expensive step, which ran over several months. The distribution of the work raises several interesting issues as well.

Approximative timeline and core-hours



2018/08 - 2019/03	**DLP-240** relation collection. 4k cores working in parallel.	21M c · h
2019/05 - 2019/08	**DLP-240** linear algebra (sequences)	5M c · h
2019/04 - 2019/06	**RSA-240** relation collection. 4.3k cores working in parallel.	7M c · h
2019/10 - 2020/02	**RSA-250** relation collection. 12k cores working in parallel.	21M c · h
2019/07 - 2019/08	**RSA-240** linear algebra (sequences)	0.6M c · h
2019/11	**RSA-240** linear algebra (wrap up)	0.1M c · h
	DLP-240 linear algebra (wrap up)	0.7M c · h
2020/02	**RSA-250** linear algebra	2M c · h

caveat: time windows often include partially idle periods

Relations, matrix size, core-years timings

	RSA-240	DLP-240	RSA-250
polynomial selection	76 core-years	152 core-years	150 core-years
$\deg f_0, \deg f_1$	1, 6	3, 4	1, 6
relation collection	794 core-years	2400 core-years	2450 core-years
raw relations	8.93G	3.82G	8.75G
unique relations	6.01G	2.38G	6.13G
filtering	days	days	days
after singleton removal	2.60G $\times$ 2.38G	1.30G $\times$ 1.00G	2.74G $\times$ 2.62G
after clique removal	1.18G $\times$ 1.18G	150M $\times$ 150M	1.82G $\times$ 1.82G
after merge, + density	282M, $d = 200$	36M, $d = 253$	405M, $d = 252$
linear algebra	83 core-years	625 core-years	250 core-years
m, n	512,256	48,16	1024,512
characters, sqrt, ind log	days	days	days

Data & reproducibility info: gitlab.inria.fr/cado-nfs/records.

Conclusions

- More than just records, we developed efficient parameterization strategies for further computations.
- We developed an extensive simulation framework to guide the parameter choices. Not perfect.
- We show that our implementation scales well and can tackle larger problems. No technology barrier at this point.

Comparisons:

- Comparison with previous record (DLP-768, 232 digits, 2016): On identical hardware, our DLP-240 computation would have taken less time than the 232-digits computation.
- FF-DLP is not much harder than integer factoring.

For future projects, we intend to keep the focus on our capacity to anticipate the computational cost, and to harness large computing power.