# Sieve Filter

## Abstract

Before I explain the problem I give brief explanation of this part of the algorithm

## Background (ChatGPT)

**Polynomial Selection**: The process begins with the selection of two polynomials that share a root modulo the number to be factored. The choice of these polynomials significantly affects the algorithm's efficiency.

**Sieving**: In this step, the algorithm searches for smooth numbers (numbers that factor completely into small primes) over two factor bases (one for each number field defined by the polynomials). This step produces relations between the primes in the factor bases and the values of the polynomial evaluations.

**Matrix Step**: The relations found during the sieving step are used to build a large, sparse matrix. Linear algebra techniques, typically involving row reduction or the block Lanczos algorithm, are used to find dependencies among these relations.

**Square Root Step**: Finally, the dependencies found in the matrix step are used to construct a congruence of squares modulo the number to be factored. This step often involves complex computations in multiple number fields.

## Background

We have:

1. Two polynomial f(x), g(x). f(x) degree = ~6, g(x) degree = 1
2. In Cado-NFS the use the word “side”.
   1. Side == 1 -> f(x), also called algebraic
   2. Side == 0 -> g(x), also called rational
3. We have two factor bases side = 0 and side = 1
4. The factor bases are a group of pairs of number (p,r)
   1. p is a prime number
   2. r is the root of the polynomial(f(x) or g(x)) mod p
5. We will call those bases β0 and β1
6. We have relations a,b we need to find
7. The formula is f
8. The formula is g
9. We need to find a and b that both fN(a,b) and gN(a,b) are smooth above the respective factor bases β0, β1

## Sieving

**Division is expensive**.

Each a,b pair will result in two numbers fN(a,b) and gN(a,b). and we need to make sure we can factorize those number above the respective factor bases.

**The rest of the document I will focus on the algebraic base, which is the more expensive anyway. But u can assume we need to do it for the rational base as well**

### Brute force

For each number fN(a,b) we need to run over all pairs (p,r) in β1 and try to divide fN(a,b) with those number. If after dividing all numbers the result is not equal to 1 it means this specific a,b are not smooth over β1 and we discard these a,b

### Simple but Important optimization

**There is a mathematical rule that if fN(a,b) is divided by a specific prime p if and only if a = br mod p**

this way we don’t have to divide each fN(a,b) with all primes in database, we just divide the ones that comply with the rule above

Of course this is the first filter which saves a lot

### Sieve

The main idea behind the sieve is how to choose the sieve area. Usually it is (-A < I < A)x(1 < J < B) and it is called a AB-plane (or IJ-plane). than we run over all a,b pairs in that plane and check if the fN(a,b) is smooth over β1

### Sieve – Special q method

Cado-NFS use some form of this algorithm. Here is a short explanations.

Because f(x) is of higher degree the number fN(a,b) usually is a very large number and most of them will not be smooth over β1.

So Pollard came with the idea to take a big prime number q and go over a,b such that fN(a,b) is divisible by q. this way fN(a,b) is a much smaller number and more likely to be smooth over β1. There is an entire proof why this result +- stay the same

If we go back to initial rule **a = br mod p** we can run a first filter on the AB-plane to use only **a = br mod q**

This initial filter minimizes the size of the plane drastically.

**In my implementation this is the point I reached**

**One very important note if I use the filter above the AB-plane becomes sparse. So in the papers and inside the code the use a conversion method to covert the AB-plane into another coodrdinate system (ij) which is continuous I will write where is the code it is done, but not fully understand it.**

### Cado-NFS Special q method

In their implementation there is another filter using logarithmic norm that reduce the AB-plane size even more.

I will try to give a Bottom Up of what I understand so far.

search\_survivors

search\_survivors\_in\_line

Lognorms array

Lognorms bases basbound

Q base

### 

**search\_survivors\_in\_line. (file: las-unsieve.cpp, line: 583)**

This method is kind of the end. One we reach this point it is very simple.

Input:

1. ij-line. A single line from the ij-plane
2. lognorm bound for both sides (rational and algebraic)
3. **a magic array (SS[2][])** for both rational and algebraic

output:

1. the list of ij that survived the magic array test

At this point it is very simple because ij is converted to an index in the magic array and we simple make sure that the values from the magic array do not exceed the bound.

This simple test filters around 90% of the ij.

**search\_survivors. (file: las-process-bucket-region.cpp, line: 356)**

this method already has the magic arrays and simply calls the search\_survivors\_in\_line function with different j value (different line of the ij-plane) and gives the correct offset from the magic arrays

The beauty is that the magic arrays are calculate per special q in the beginning before everything even start and they give this amazing filter.

This function:

1. loops over all j values (line: 421)
2. fix the magic arrays (line: 425)
3. calls search\_survivors\_in\_line function (line: 442)

**qlattice\_basis (file: las-qlattice)**

this gets the special q as input and create the a0,b0,a1,b0 conversion from AB-plane to ij-plane

it is a short file and there is documentation inside.

it uses some gauss skew technique which is beyond my understanding.

**las-norms**

this files in my POV hold all the magic, if u can manage to understand all of it and explain it to me it will be awesome (it is 1300 lines of code)

I will focus here on a more specific area.

In file las-norms.cpp line 540. It is the constructor of the **lognorm\_smart** class. This class is created per side (rational and algebraic)

At a later part of the code a call is made to **lognorm\_smart** method called fill. Which fills the magic arrays for rational and algebraic method.

## What do I need?

**The most urgent and important things is to understand how the magic array get filled and how to convert a,b relation into i,j and from i,j to find the correct index inside the magic array**

of course any other information will be very helpful

## What do I know so far?

The magic array is a logarithmic norm approximation of something (I think it the prime numbers of the factor bases). The approximation of the logarithmic norm result is a single byte of memory. This is why the arrays are unsigned char.

Thanks

Lior