Quelques points de détail dans l'implémentation de NFS

Emmanuel Thomé
Université de Lorraine, CNRS, Inria, Nancy, France

Roscoff – Avril 2025

ANR-22-PECY-0010

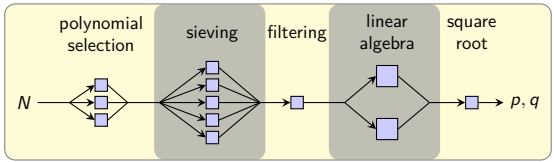
NFS = The Number Field Sieve

- ...factors integers ("semiprimes" such as RSA moduli) $\rightarrow \approx 2^{829} \approx 10^{250}$ (2020)
- ...computes discrete logarithms in finite fields $\rightarrow \approx 2^{795} \approx 10^{240}$ (2019)
- ...is a VERY complicated algorithm. Reference: my lectures notes on NFS.
- We don't have that much time, right?

This talk: some key aspects of sieving (relation collection) in NFS.

Summary of NFS

The NFS algorithm (1990) easily fills a one-semester graduate course.



Computational requirements are diverse.

- Sieving (relation collection) is the most expensive. It can be massively distributed.
- (Sparse) Linear algebra comes second.
  It is somewhat cheaper, but needs expensive hardware.
- There are also many auxiliary tasks.

Plan

Looking for smooth numbers in a sieving region

Computing norms

Adjusting the sieve area

Polynomial values

NFS selects a polynomial $f = f_d x^d + f_{d-1} x^{d-1} + \cdots + f_1 x + f_0 \in \mathbb{Z}[x]$.

- Coefficients (for GNFS) are typically such that $\prod |f_i| \approx N$.
  The degree matters a lot. Have in mind $d \approx 6$.

We define $F(X, Y) = f(X/Y)Y^d = f_d X^d + \cdots + f_0 Y^d$.

What relation collection is all about

Within some search space, find (a, b) such that $F(a, b)$ is a smooth number (all its prime factors are below some bound).

What does f look like?

- If we have $|f_i|$ roughly equal for all i, and we pick integers (a, b) such that $|a| \approx |b| \approx A$, then all summands of $F(a, b)$ are of roughly equal magnitude.
- It often happens that we have some skewness, e.g. $|f_i/f_{i+1}| \approx s$ for some $s > 1$. Then the same holds if we take $|a| \approx A\sqrt{s}$ and $|b| \approx A/\sqrt{s}$.

$$
\begin{aligned}
f_{\text{RSA-240}} &= 10853204947200\,x^6 \\
&- 4763683724115259920\,x^5 \\
&- 6381744461279867941961670\,x^4 \\
&+ 9744489348538648076906750670 37\,x^3 \\
&+ 1792005735336657213102106407380 61170\,x^2 \\
&+ 1595712553369335430496125795083146688523\,x \\
&- 2211755888422991175905645426099770 16567191860
\end{aligned}
$$

Approaches (1/4)

Naive:

```
for a in range(max_a):
    for b in range(max_b):
        test_smoothness(F(a, b))
```

It's very bad for multiple reasons.
It's so bad that it can even worsen the overall complexity estimate.

Approaches (2/4)

Sieving by rows (as in Eratosthenes):

```
T = { (a,b):0 for a in range(max_a) for b in range(max_b) }
for p in range(max_p):
    for b in range(max_b):
        a = starting_point_in_line  # sometimes several options
        while a < max_a:
            T[(a,b)] += log2(p)  # coarse approximation
            a += p
```

- The starting point in line b is the smallest a such that $F(a, b) \equiv 0 \bmod p$.
- At the end, `T[(a,b)]` contains a large value if many prime factors have hit. We compare this with $\log_2 |F(a, b)|$ and decide what to do.

`T[(a,b)]` vs $\log_2 |F(a,b)|$

If we end up with $\log_2 |F(a,b)| - \texttt{T[(a,b)]} = \varepsilon$, then:

- $F(a,b)$ is a product of primes below `max_p` and a cofactor $\approx 2^\varepsilon$.
  - this cofactor is not necessarily prime.
  - a source of inaccuracy: coarse $\log_2$ approximation.
  - (also: perhaps we didn't sieve prime powers).
- Rarely an all-or-nothing situation.
  If ε is "reasonably small", it makes sense to (compute and) factor the cofactor to see if all factors are below some "large prime bound".
- For record computations, "cofactoring" numbers above 100 bits is frequent.
- Potential work down the line means we don't want to engage in this for nothing.

Being too much off in the value ε is bad.

How constant is "roughly constant"?

Recall that all summands of $F(a, b)$ are roughly the same (max) size.

Can we say that $F(a, b)$ is a constant and compare all `T[(a,b)]` values to this constant?.

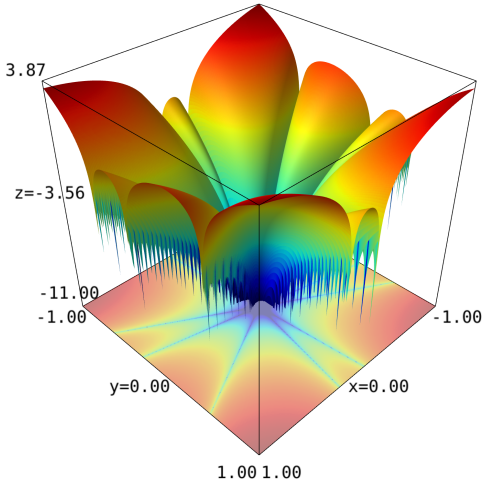
F is a homogenous polynomial. If skewness $= s$, $a = \sqrt{s}\cos\theta$, $b = (1/\sqrt{s})\sin\theta$, then

$$\begin{aligned} |F(a, b)| &= r^d s^{-d/2} f(s\cot\theta) \cdot (\sin\theta)^d \\ &= r^d \cdot K \cdot \widetilde{F}(\cos\theta, \sin\theta) \end{aligned}$$

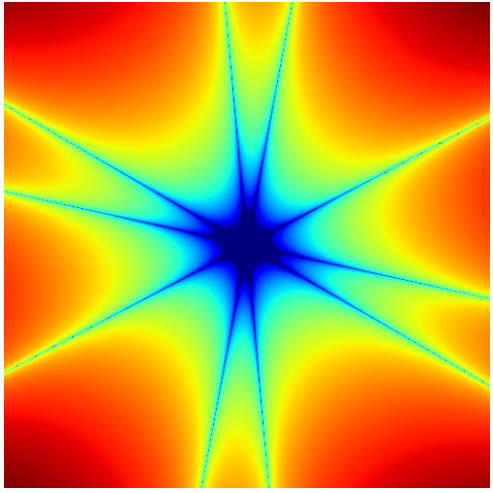
for some constant K and where $\widetilde{F}$ has all its coefficients roughly ≈ 1.

(see: Elkenbracht-Huizing, Montgomery)

In 3D: $z = |\log_2 F(x, y)|$



In 2D: $z = |\log_2 F(x, y)|$



Sad but true: we need to compute norms.

We can't afford being too imprecise here.

- We need to compute $\log_2 |F(a,b)|$ with some reasonable accuracy
- Then we can compare `T[(a,b)]` to it.

Challenge #1 (spoiler alert: solved)

For a given range, compute $\log_2 |F(a,b)|$ for all (a,b).
To fix ideas and simplify things:

- Forget about skewness.
- $-2^{15} \leq a < 2^{15}$, and $0 \leq b < 2^{15}$.

Quality criterion: error ≤ 1 bit, with at most a few exceptions.

Plan

Looking for smooth numbers in a sieving region

Computing norms

Adjusting the sieve area

Facts about norm computation

Computing $F(a, b)$ first is not the way to go:

- With long integer arithmetic: too expensive at this point.
- With double precision: still too expensive, and cancellations might bite.

The degree one case

In the degree one case, say $F = vX - uY$, things are not too bad.

To compute $\log_2 |F(a, 1)|$ for all integers a in range:

- Compute root $z = u/v$. Note that for $x = z + 2^k/v$, we have $F(x, 1) = 2^k$.
- Compute the boundary points $a_i = (z + 2^{k_0+i}/v)_i$ and $a'_i = (z - 2^{k_0+i}/v)_i$, keeping only the meaningful intervals.
- `memset` whole chunks of memory with identical values, since $\log_2 |F|$ doesn't change in the intervals $[a_k, a_{k+1})$.

To extend this to $|F(a, b)|$ for all b, use $F(a, b) = b^d \cdot F(a/b, 1)$.

- Reuse the same boundary points (caveat about meaningful intervals).
- Add $d \cdot \log_2 |b|$.

It breaks down in larger degree

Larger degree:

- More roots.
- Computing them is more expensive. Precision issues.
- No obvious analogue to the above.

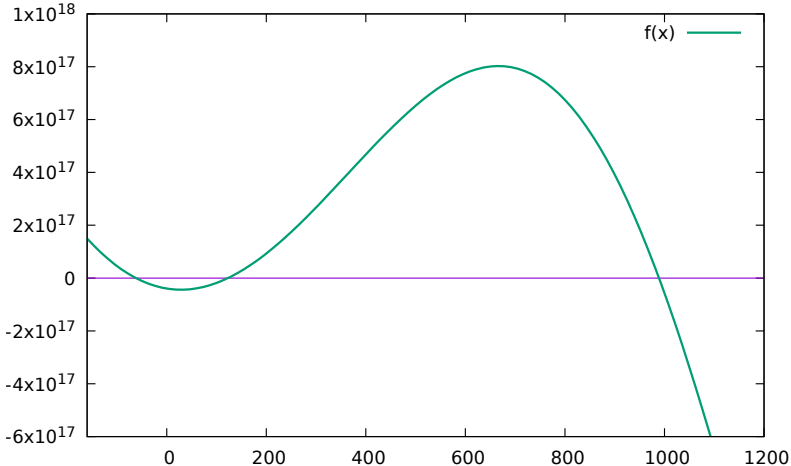
(Obvious) suggestion:

- provide a piecewise linear approximation...
- ...that is accurate up to a multiplicative factor (say 2).

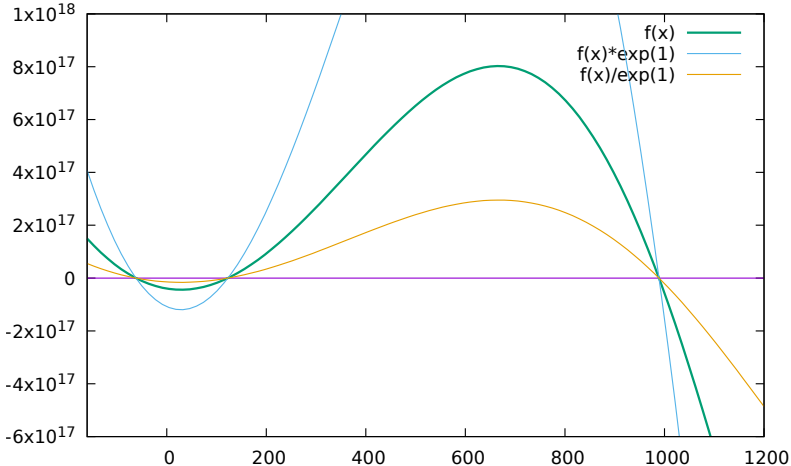
Data structure: a list of linear functions $u_0, \ldots, u_{k-1}$ and consecutive intervals $R_0, \ldots, R_{k-1}$ such that $\cup_s R_s = [-2^{15}, 2^{15}]$ and

$$\forall s,\ \forall i \in R_s,\ \frac{1}{2}|F(i,1)| \leq |u_s(i)| \leq 2|F(i,1)|$$

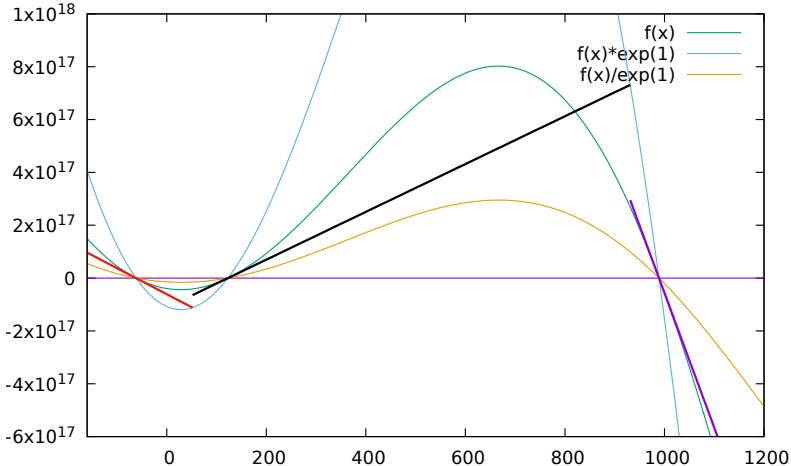
Piecewise linear approximations: example



Piecewise linear approximations: example



Piecewise linear approximations: example



All good?

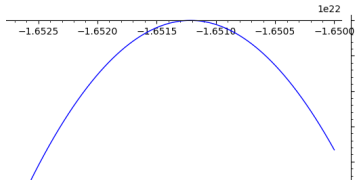
It mostly works, and mostly solves our problem.

- Compute set of real roots.
- For each root r, define $T_r(x) = f'(r) \cdot (x - r)$ as one of the u_i.
- Find the endpoints of the validity interval for $T_r(x)$ (i.e. around r): These are zeros of $|T_r(x) - 2F(x, 1)|$ and $\left|T_r(x) - \frac{1}{2}F(x, 1)\right|$.
- For each hole, try to find a new approximating line.
- Recurse until there are no holes left.

We then proceed in each interval as in the degree one case.

As in the degree one case, this generalizes to $F(a, b)$ with $b > 1$ fairly easily.

Almost all good



Some pitfalls with precision, though.
Real numbers are messy and hard.

- Some root computations will be wrong, or missed.
- Some recursions might go into infinite loops.
- Must pay extra attention to endpoint computations, and avoid meaningless intervals.
- Difficult (impossible?) to guarantee correctness in general.
- It makes sense to fall back to higher precision (possibly integer) code if we encounter problems that we can detect.

Problem #1 solved

Integer computation: up to 25% of the sieving time in computing norms!!

PL approximation: down to 0.3%,
at which point there's no need in optimizing it further.

Plan

Looking for smooth numbers in a sieving region

Computing norms

Adjusting the sieve area

More approaches to sieving

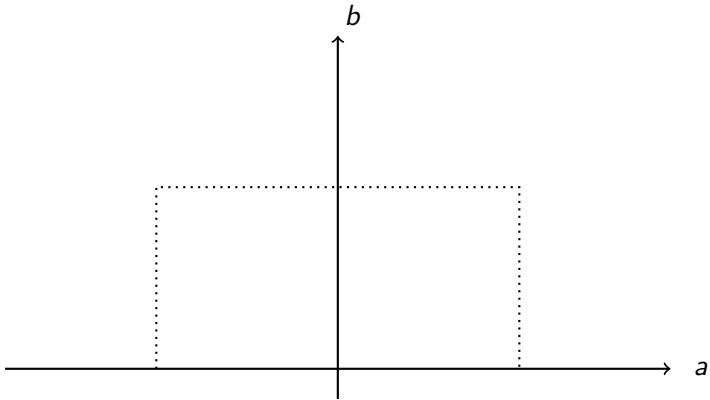
Davis and Holdridge 1982: special-q sieving.

- Choose a prime q among many.
- Look for values (a, b) such that $F(a, b) \equiv 0 \mod q$.
- This implies $f(a/b) \equiv 0 \mod q$.
  IOW $a - br \equiv 0 \mod q$ where r is one of the (at most d) roots of f modulo q.

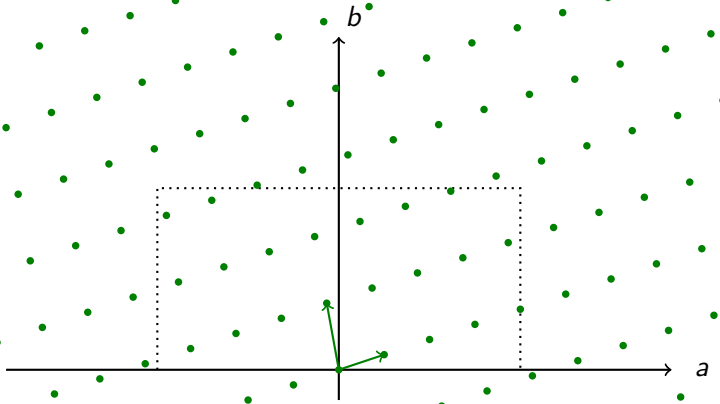
A choice of q (and r) gives a lattice of (a, b) pairs to look into: "q-lattice".

(Davis & Holdridge 1982, Pollard 1993)

Special-$\mathfrak{q}$

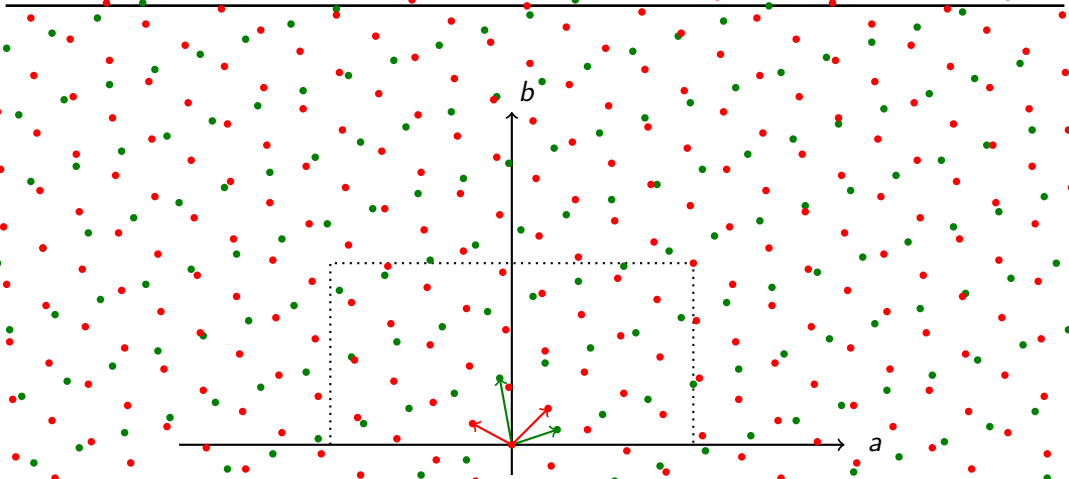


Special-q



Points where q divides.

Special-q

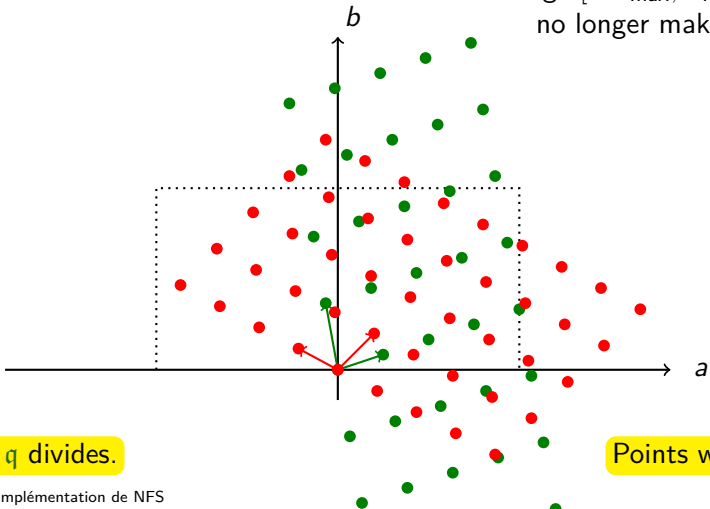


Points where $\mathfrak{q}$ divides.

Points where $\mathfrak{q}$ divides

Special-$\mathfrak{q}$

The range $[-A_{\max}, A_{\max}] \times [0, B_{\max}]$ no longer makes much sense.



Points where $\mathfrak{q}$ divides.

Points where $\mathfrak{q}$ divides

Approaches to relation collection (3/4)

Do fixed-size (i, j) coordinates, supported by $\vec{u_0} = (a_0, b_0)$ and $\vec{u_1} = (a_1, b_1)$.

```
for q in range(max_q):
    u0, u1 = basis_of_lattice
    T = { (i,j):0 for i in range(max_i) for j in range(max_j) }
    for j in range(max_j):
        i = starting_point_in_line
        while i < max_i:
            T[(i,j)] += log2(p)
            i += p
    # scan T for high values, convert (i,j) to (a,b), investigate.
```

(Approach 4/4: Franke& Kleinjung 2005)

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 $\mathcal{L}_q$.
- We pick $\vec{u_0}$ the shortest vector in $\mathcal{L}_q$.
- We let typically $-I/2 \leq i < I/2$ and $0 \leq j < J$, have in mind $J = I/2$ too.
- We have $\det \mathcal{L}_q = q$.
- Furthermore, because of skewness, $a_i/b_i \approx s$.
  It makes sense to look at $\vec{u_i'} = (a_i\sqrt{\frac{1}{s}}, b_i\sqrt{s})$ and $F(X\sqrt{s}, Y/\sqrt{s})$, meaning that WLOG we may assume $s = 1$.

Pictures = plots of $[-I/2, I/2]\vec{u_0'} + [0, I/2]\vec{u_1'}$.

(a, b) versus (i, j)

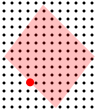
How are the reached (a, b)'s distributed?

- It depends on the basis vectors $\vec{u_0}$ and $\vec{u_1}$.
- $\vec{u_0}$ is pretty much isotropic.
- We know $\det(\vec{u_0}, \vec{u_1}) = q$, but $\vec{u_1}$ can be quite large.

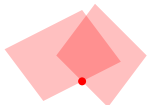
Challenge #2

Characterize, possibly optimize, the area of (a, b) pairs reached by lattice sieving.
Quality criterion: avoid excessively large vectors, favor those such as $|F(a, b)|$ is small.

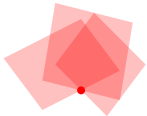
Sieving rectangle



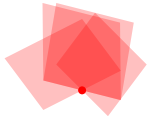
Sieving rectangle



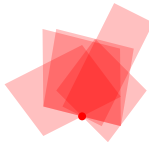
Sieving rectangle



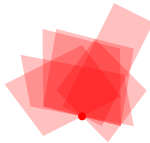
Sieving rectangle



Sieving rectangle



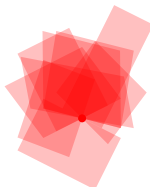
Sieving rectangle



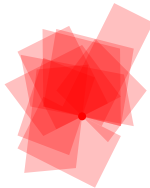
Sieving rectangle

Sieving rectangle

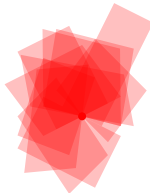
Sieving rectangle



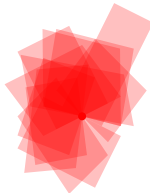
Sieving rectangle



Sieving rectangle



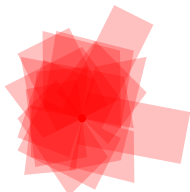
Sieving rectangle



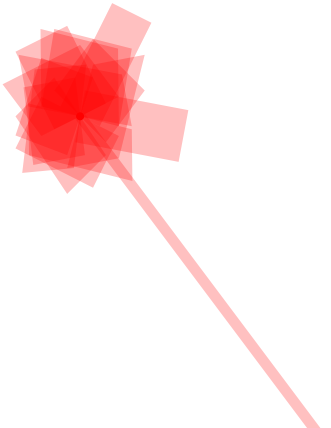
Sieving rectangle

Sieving rectangle

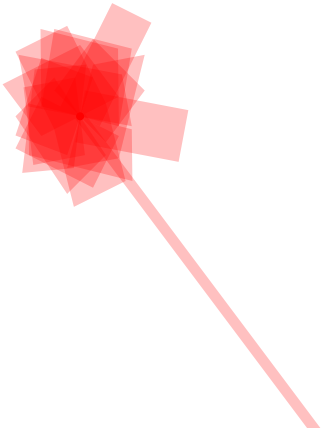
Sieving rectangle



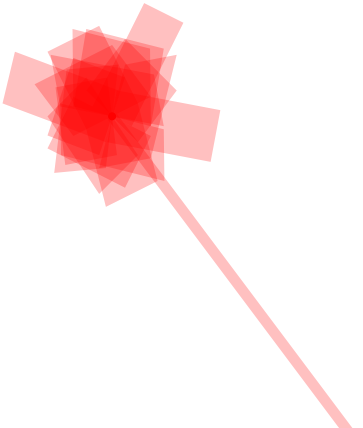
Sieving rectangle



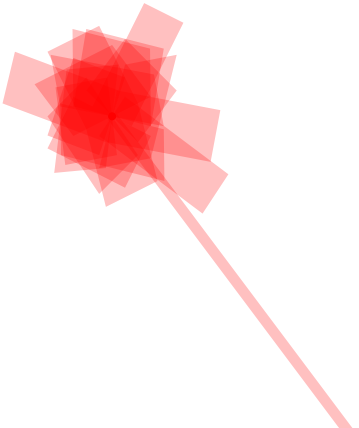
Sieving rectangle



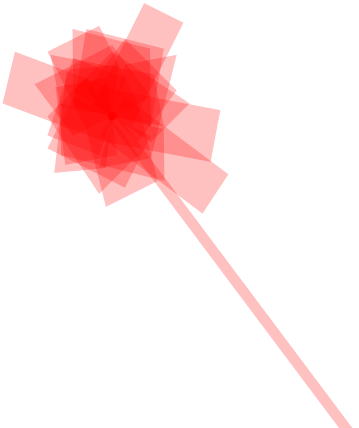
Sieving rectangle



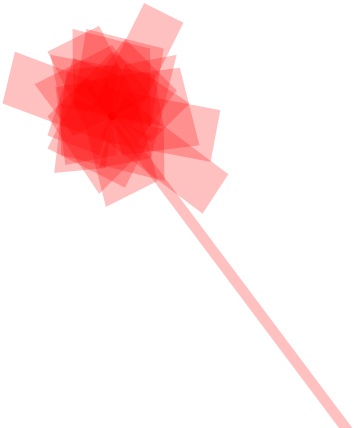
Sieving rectangle



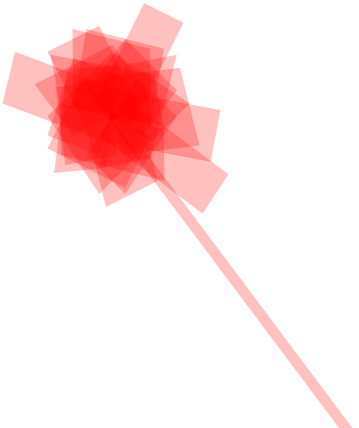
Sieving rectangle



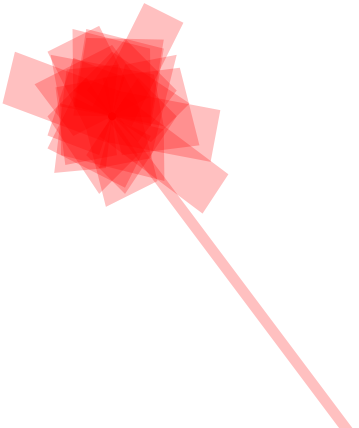
Sieving rectangle



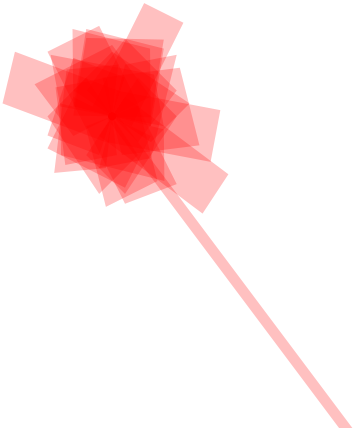
Sieving rectangle



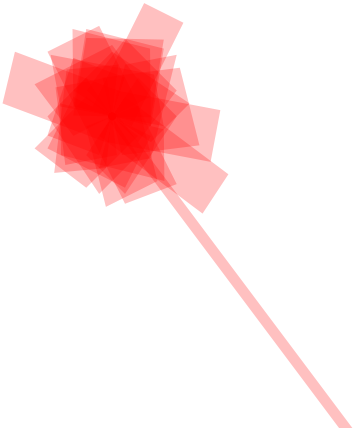
Sieving rectangle



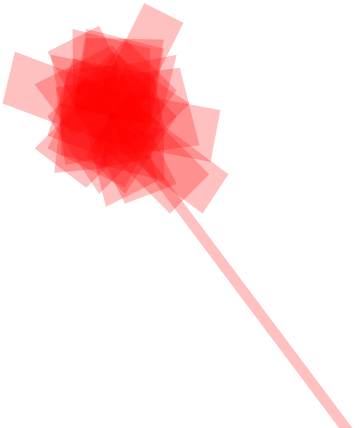
Sieving rectangle



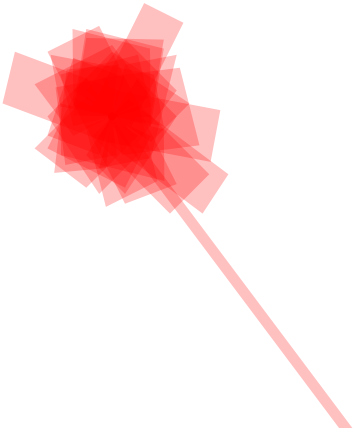
Sieving rectangle



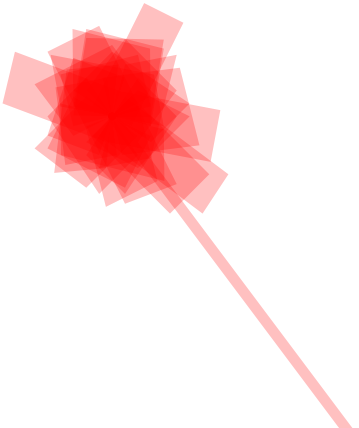
Sieving rectangle



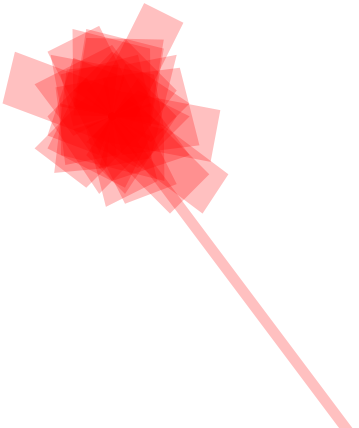
Sieving rectangle



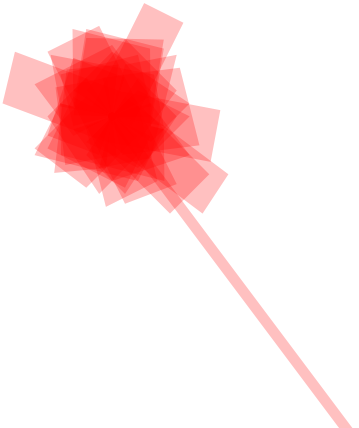
Sieving rectangle



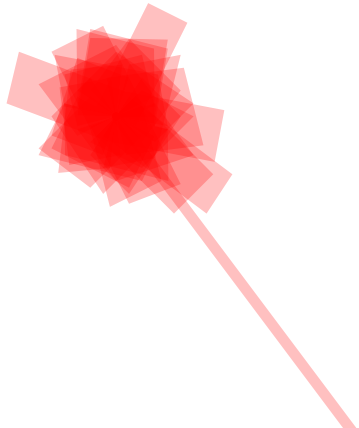
Sieving rectangle



Sieving rectangle



Sieving rectangle



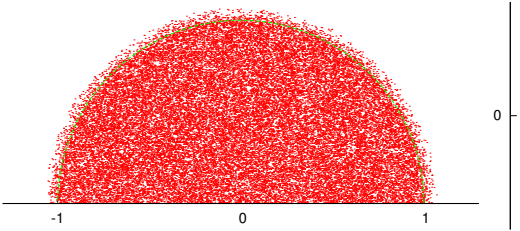
What's going on?

It might really happen that the sieving rectangle looks pretty distorted.

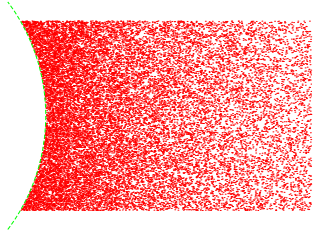
If $\vec{u_0}$ is really really short, then $\vec{u_1}$ is somewhat bigger, since the lattice determinant is constrained.

As 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): $u_0/\sqrt{q}$



(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.

- Strategy (a): ignore the issue.
- Strategy (c):
  - Limit J to a small value if $\vec{u_1}$ is large.
  - 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).

The issue we observe with (c) is that J is sometimes reduced a LOT.

Effect of strategy (c)

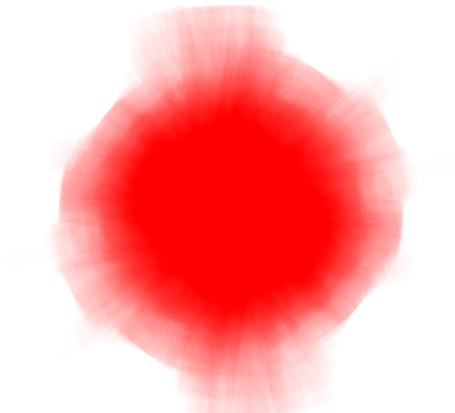
Very skewed lattice $\Rightarrow$ J reduced a lot $\Rightarrow$ some special-q's are much faster:

```
# 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)]
```

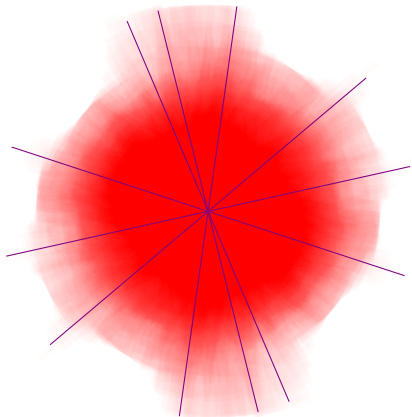
while for others:

```
# 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)]
```

Sieving rectangle with (c) (RSA-240, $\sim$600 sq's)



Sieving rectangle with (c) (RSA-240, $\sim$600 sq's)



Strategy (b): adjusting I, J better

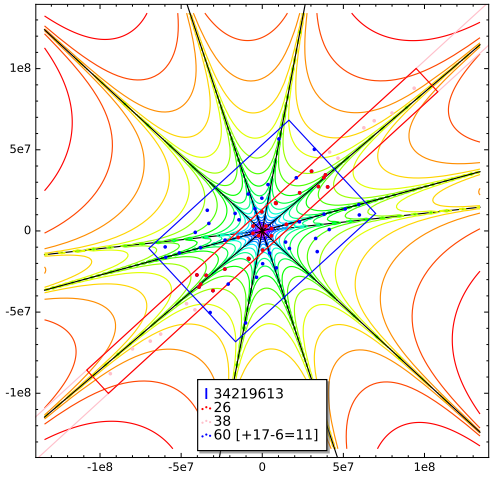
Assume we have in mind an area $A \approx 2^{31}$ within the (i, j) plane.

We sieve for $(a, b) = i \cdot \vec{u_0} + j \cdot \vec{u_1}$.

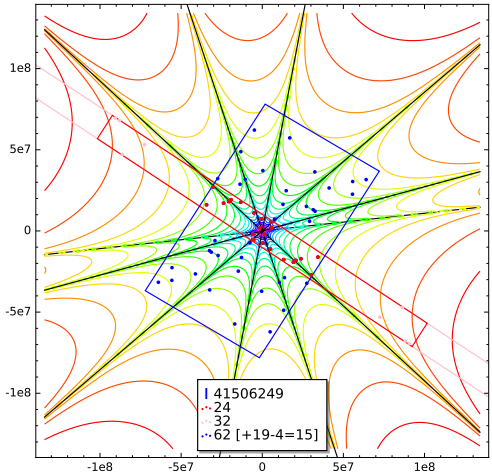
Obvious "we should rather do this" strategy (b):

- Multiply the shortest vector by the largest interval.
- If $|\vec{u_0}| \approx \frac{\sqrt{q}}{2^k}$, take:
  - I a power of two close to $\sqrt{A} \cdot 2^k$.
  - J a power of two close to $\sqrt{A} \cdot 2^{-k}$.

Sieve area with strategy (b)

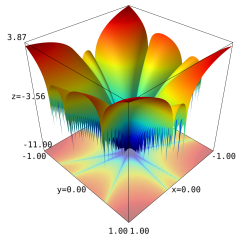


Sieve area with strategy (b)



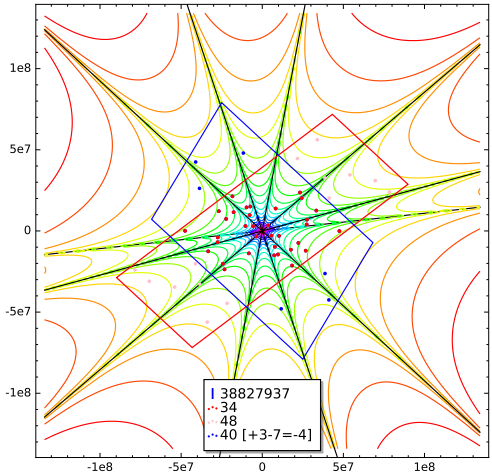
Turning the basis around

Strategy (b) also tries alternative bases.
Those may work better with $F(x, y)$.

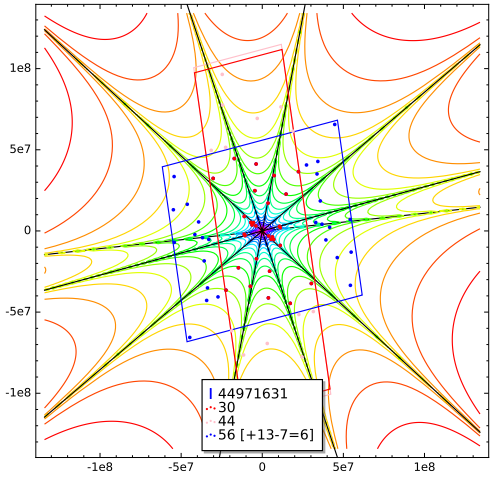


- $(\vec{u_0}, \vec{u_1} \pm \vec{u_0})$
- $(\vec{u_0}, \vec{u_1} \pm 2\vec{u_0})$
- ...any "short" matrix in $\mathrm{SL}_2(\mathbb{Z})$ can be used.

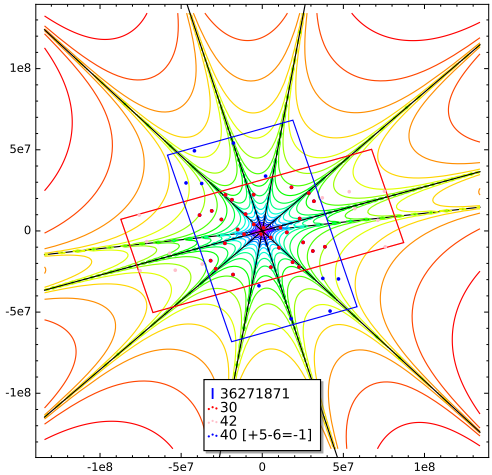
Sieve area with strategy (b)



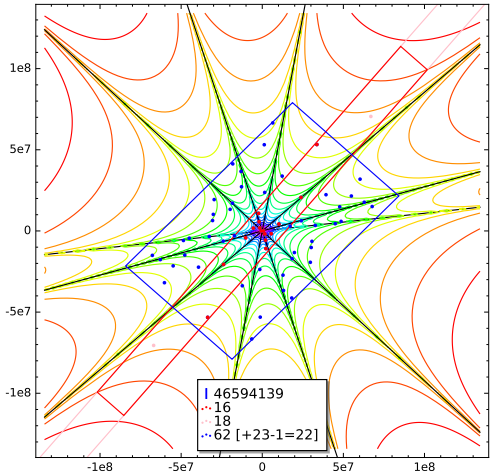
Sieve area with strategy (b)



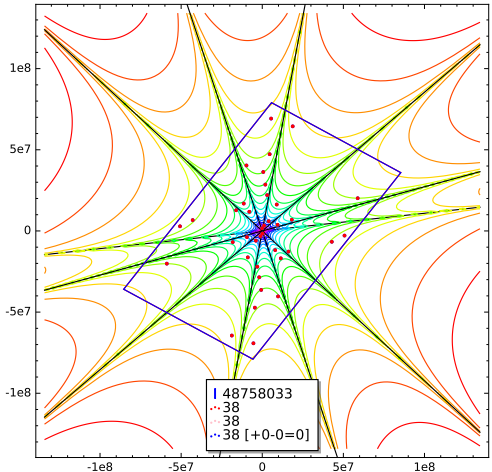
Sieve area with strategy (b)



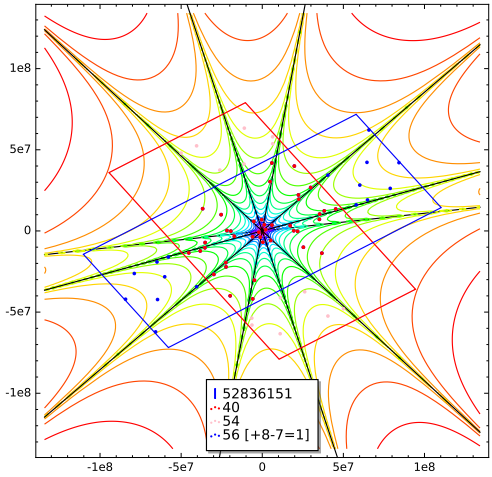
Sieve area with strategy (b)



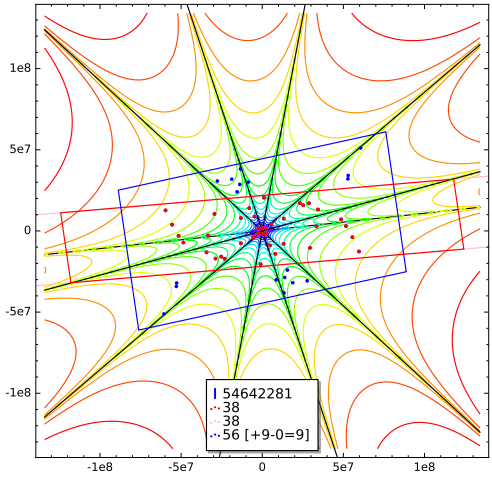
Sieve area with strategy (b)



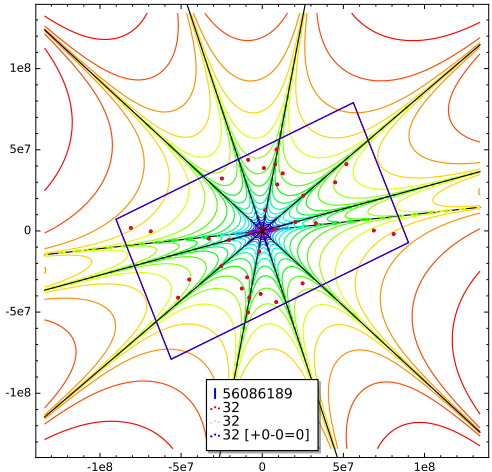
Sieve area with strategy (b)



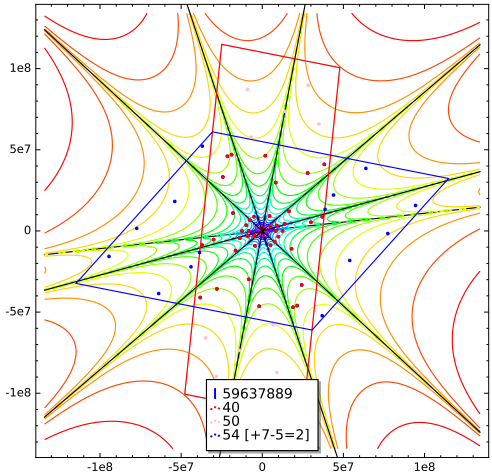
Sieve area with strategy (b)



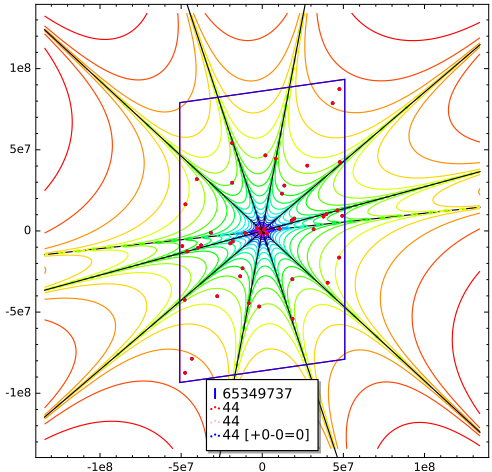
Sieve area with strategy (b)



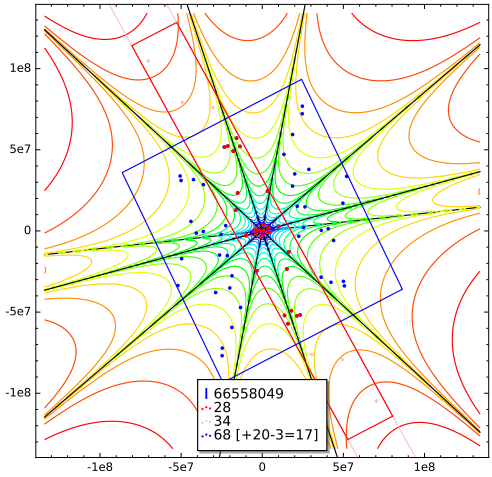
Sieve area with strategy (b)



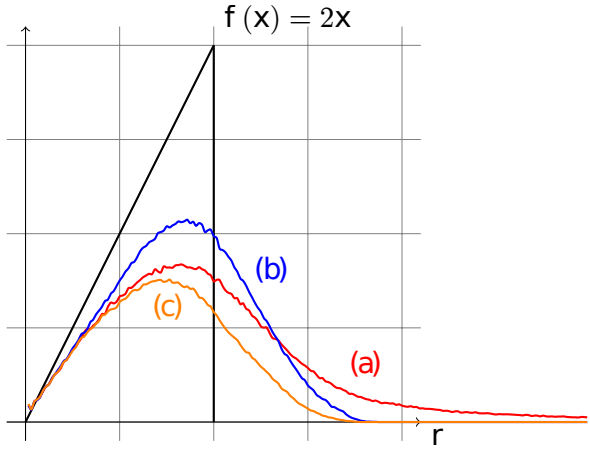
Sieve area with strategy (b)



Sieve area with strategy (b)



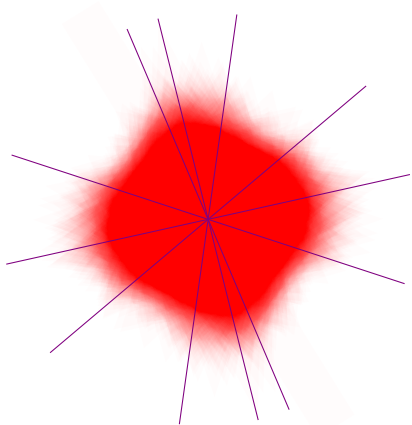
Average norm of (a, b)



This is not always a game changer, but avoids wasting some special-q's.

Sieving rectangle with (b) (RSA-240, $\sim$600 sq's)

Sieving rectangle with (b) (RSA-240, $\sim$600 sq's)



Example results (RSA240 data)

Strategy (c):

```
# Total 29435 reports [5.7s/r, 46.0r/sq] in 2.76e+03 elapsed s [6074.9% CPU
```

Strategy (b):

```
# Total 36472 reports [4.85s/r, 57.0r/sq] in 2.91e+03 elapsed s [6085.7% CP
```

- More time overall because we don't give up on skewed special-q's.
- This pays off because we look at more interesting (a, b) pairs.