Building circuits for integer factorization

D. J. Bernstein

Thanks to:
University of Illinois at Chicago
NSF DMS–0140542
Alfred P. Sloan Foundation

I want to work for NSA. . .

. . . as an independent contractor.

Outline of business plan:

NSA sends me (pq, d), where

- pq is a 1024-bit integer;
- p, q are primes;

. . . as an independent contractor.

Outline of business plan:

NSA sends me (pq, d), where

- pq is a 1024-bit integer;
- p, q are primes;
- d is a large pile of cash.

One year later,
I send NSA (p, q) or (q, p).

Extremely important: Need

CONFIDENCE

that d dollars are more than enough
to compute (p, q) or (q, p)
in one year.

Extremely important: Need

CONFIDENCE

that d dollars are more than enough
to compute (p, q) or (q, p)
in one year.

If d is not enough,
NSA sends me to Guantanamo Bay.
Unacceptable risk.

Can we expect to achieve confidence that cost of factoring pq is $\leq d$?

Yes.

Expensive way to achieve confidence: Go ahead and spend $\leq d$ dollars factoring pq.

Goal: Achieve confidence with much less expense than doing the factorization.

Other issues, not as important:

- NSA would like minimum d
  ... but all d's in a wide range are acceptable.
- I would like my actual computation cost to be minimum
  ... but all costs in a wide range are acceptable.

CONFIDENCE is essential.
Minimization is not essential.

Minimum cost is not essential
but we can still aim for it.

Can we expect to achieve it? No!

Can never confidently state
lower bound on the cost.

People keep discovering
ways to reduce the cost.

Let's look at an example:
finding a good NFS polynomial.

Degree $1 + 5$ number-field sieve
is given n; chooses $m \approx n^{1/6}$;
expands n in base m as
$n = f_5 m^5 + f_4 m^4 + \cdots + f_0$,
maybe with negative coefficients;
contemplates polynomial values
$(c - dm)(f_5 c^5 + f_4 c^4 d + \cdots + f_0 d^5)$.

Have $f_5 \approx n^{1/6}$.
Typically all the f_i's
are on scale of $n^{1/6}$.

(1993 Buhler Lenstra Pomerance)

To reduce values by factor B:

Enumerate many possibilities
for m near $B^{0.25}n^{1/6}$.

Have $f_5 \approx B^{-1.25}n^{1/6}$.
f_4, f_3, f_2, f_1, f_0 could be
as large as $B^{0.25}n^{1/6}$.
Hope that they are smaller,
on scale of $B^{-1.25}n^{1/6}$.

Conjecturally this happens
within roughly $B^{7.5}$ trials.
Then $(c - dm)(f_5c^5 + \cdots + f_0d^5)$
is on scale of $R^6(n/B^3)^{2/6}$
for c, d on scale of R.

Improvement: Force f_4 to be small.

Say $n = f_5 m^5 + f_4 m^4 + \cdots + f_0$.

Choose integer $k \approx f_4/5f_5$.
Write n in base $m + k$:
$n = f_5(m+k)^5$
$\quad + (f_4 - 5kf_5)(m+k)^4 + \cdots$.

Now degree-4 coefficient
is on same scale as f_5.

Hope for small f_3, f_2, f_1, f_0.
Conjecturally this happens
within roughly B^6 trials.

Improvement: Skew coefficients.
(1999 Murphy, without analysis)

Enumerate many possibilities
for m near $Bn^{1/6}$.

Have $f_5 \approx B^{-5}n^{1/6}$.
f_4, f_3, f_2, f_1, f_0 could be
as large as $Bn^{1/6}$.

Force small f_4. Hope for
f_3 on scale of $B^{-2}n^{1/6}$,
f_2 on scale of $B^{-0.5}n^{1/6}$.

Conjecturally this happens within roughly $B^{4.5}$ trials: $(2+1)+(0.5+1)=4.5$.

For c on scale of $B^{0.75}R$ and d on scale of $B^{-0.75}R$ have $c-md$ on scale of $B^{0.25}Rn^{1/6}$ and $f_5c^5+f_4c^4d+\cdots+f_0d^5$ on scale of $B^{-1.25}R^5n^{1/6}$.

Product $R^6(n/B^3)^{2/6}$.

Improvement:
Control another coefficient.
(2004.11 Bernstein)

Say $n = f_5 m^5 + f_4 m^4 + \cdots + f_0$.

Choose integer $k \approx f_4/5f_5$
and integer $\ell \approx m/5f_5$.

Find all short vectors
in lattice generated by
$(m/B^3, 0, 0, 10f_5k^2 - 4f_4k + f_3)$,
$(0, m/B^4, 0, 20f_5k\ell - 4f_4\ell)$,
$(0, 0, m/B^5, 10f_5\ell^2)$,
$(0, 0, 0 \quad\quad\quad, m)$.

Hope for j below B^1
with $(10f_5k^2 - 4f_4k + f_3)$
$\quad + (20f_5k\ell - 4f_4\ell)j$
$\quad + (10f_5\ell^2)j^2$
below m/B^3 modulo m.

Write n in base $m + k + j\ell$.
Obtain degree-5 coefficient
on scale of $B^{-5}n^{1/6}$;
degree-4 coefficient
on scale of $B^{-4}n^{1/6}$;
degree-3 coefficient
on scale of $B^{-2}n^{1/6}$.
Hope for good degree 2.

Conjecturally succeed
within roughly $B^{3.5}$ trials.

Saves time as soon as B exceeds
ratio of lattice-reduction time
between dimensions $1, 4$.

Faster polynomial search
$\Rightarrow$ can afford larger B
$\Rightarrow$ smaller polynomial values
$\Rightarrow$ faster factorization.

Claims of the form
"Factoring pq costs $\geq d$,"
or "factoring pq with the
number-field sieve costs $\geq d$,"
are inherently untrustworthy
and frequently wrong.

Many people claimed that NFS
would cost more than QS
for 120-digit integers.
That's speculation, not science.
They were wrong.

Erroneous lower-bound claims
occur in other contexts too.

Fast integer multiplication
(time exponent $1 + o(1)$)
has now set ECPP speed records.
(2004 Morain talk: "More and
more powerful computers
$\Rightarrow$ fast methods begin to be
fast in real life")

Many people had claimed
that fast multiplication
is of no practical interest.
They were wrong.

In contrast, claims of the form
"Factoring pq costs $\leq d$"
are sometimes justified.

But not always! Check the details.

Example: "These integers have smoothness probability $\geq 2 \cdot 10^{-11}$ ... since $\rho(10) = 2.77\ldots \cdot 10^{-11}$" is unjustified speculation.

"These integers have
smoothness probability $\geq 2 \cdot 10^{-11}$
. . . by extrapolation from
smaller factorization experiments
using $\exp(\sqrt[3]{\text{blah blah blah}})$"
is unjustified speculation.

"These integers have
smoothness probability $\geq 2 \cdot 10^{-11}$
. . . as shown by smoothness
tests on a uniform random sample
of 10^{15} of these integers"
is justified—but not cheaply.

Can much more quickly
obtain good lower bounds
on smoothness probabilities.

Define S as the set of
1000000-smooth integers $n \geq 1$.

The Dirichlet series for S
is $\sum [n \in S] x^{\lg n} =$
$(1 + x^{\lg 2} + x^{2 \lg 2} + x^{3 \lg 2} + \cdots)$
$(1 + x^{\lg 3} + x^{2 \lg 3} + x^{3 \lg 3} + \cdots)$
$(1 + x^{\lg 5} + x^{2 \lg 5} + x^{3 \lg 5} + \cdots)$
$\cdots$
$(1 + x^{\lg 999983} + x^{2 \lg 999983} + \cdots)$.

Replace primes $2, 3, 5, 7, \ldots, 999983$ with slightly larger real numbers $\overline{2} = 1.1^{8}$, $\overline{3} = 1.1^{12}$, $\overline{5} = 1.1^{17}$, $\ldots$, $\overline{999983} = 1.1^{145}$.

Replace each $2^a 3^b \cdots$ in S with $\overline{2}^a \overline{3}^b \cdots$, obtaining multiset $\bar{S}$.

The Dirichlet series for $\bar{S}$ is $\sum [n \in \bar{S}] x^{\lg n} =$

$(1 + x^{\lg \overline{2}} + x^{2 \lg \overline{2}} + x^{3 \lg \overline{2}} + \ldots)$
$(1 + x^{\lg \overline{3}} + x^{2 \lg \overline{3}} + x^{3 \lg \overline{3}} + \ldots)$
$(1 + x^{\lg \overline{5}} + x^{2 \lg \overline{5}} + x^{3 \lg \overline{5}} + \ldots)$
$\ldots$
$(1 + x^{\lg \overline{999983}} + x^{2 \lg \overline{999983}} + \ldots)$.

This is simply a power series
$c_0 y^0 + c_1 y^1 + \cdots =$
$(1 + y^8 + y^{2 \cdot 8} + y^{3 \cdot 8} + \cdots)$
$(1 + y^{12} + y^{2 \cdot 12} + y^{3 \cdot 12} + \cdots)$
$(1 + y^{17} + y^{2 \cdot 17} + y^{3 \cdot 17} + \cdots)$
$\cdots (1 + y^{145} + y^{2 \cdot 145} + \cdots)$
in the variable $y = x^{\lg 1.1}$.

Compute series mod (e.g.) y^{2910};
i.e., compute $c_0, c_1, \ldots, c_{2909}$.
$\bar{S}$ has $c_0 + \cdots + c_{2909}$ elements
$\leq 1.1^{2909} < 2^{400}$, so S has
at least that many elements $< 2^{400}$.

So have guaranteed lower bound on number of 1000000-smooth integers in $[1, 2^{400}]$.

Can compute an upper bound to check looseness of lower bound.

If looser than desired,
move 1.1 closer to 1.
Achieve any desired accuracy.

What about more complicated notions of smoothness?

Can modify Dirichlet series
in many interesting ways to
modify notion of smoothness.

Use $1 + x^{\lg \overline{999983}}$ instead of
$(1 + x^{\lg \overline{999983}} + x^{2 \lg \overline{999983}} + \cdots)$
to throw away n's having
more than one factor 999983.

Multiply $c_0 y^0 + \cdots + c_{2909} y^{2909}$
by $x^{\lg \overline{1000003}} + \cdots + x^{\lg \overline{999999937}}$
to allow n's that are
1000000-smooth integers $< 2^{400}$
times one prime in $[10^6, 10^9]$.

What about polynomial values?

Twisted Dirichlet series for
powers of an invertible ideal P
of the ring of integers of
$\mathbf{Q}(\alpha)/(\alpha - m)(f_5\alpha^5 + \cdots + f_0)$:
$1 + [P]x^{\lg N(P)} + [P]^2x^{2\lg N(P)} + \ldots$
where [] is class, N is norm.

Replace N with $\overline{N}$,
multiply for various P's
to see distribution of
smooth ideals in each class.
Check that small principal ideals
correspond to $(c - dm)(\cdots)$.

This is much more complicated
than simply using ρ;
but it gives us **CONFIDENCE**
regarding smoothness probabilities.
Reasonably small CPU time.

Trickier type of tradeoff:
Are we willing to sacrifice
CPU time in the factorization
to gain confidence?

Let's look at one proposal:

Build $y^{1/2} \times y^{1/2}$ mesh of simple processors.

Build y^ϵ pairs (p, i)
into each processor.

Spread c's among processors.

Each processor is $\#c$ for one c.

#1 (2, 1) (5, 2)	#2 (2, 2) (7, 1)	#3 (2, 3) (7, 2)
#4 (2, 4)	#5 (2, 5)	#6 (3, 1)
#7 (3, 2)	#8 (3, 3)	#9 (5, 1)

Given n:

For each (p, i), processor
generates ith multiple s of p
in $\{n+1, n+2, \ldots, n+y\}$,
if there is one,
and sends (p, s) to $\#(s-n)$
through the mesh.

With random routing:
$y^{1/2+\epsilon}$ time, $y^{1+\epsilon}$ hardware.

(2001.03 Bernstein talk,
"The NSA sieving circuit")

But does routing really work?
Packets bump into each other.
Even worse, in linear algebra,
many packets are aimed at a
small part of the mesh.

Gain confidence by switching
to a mesh-sorting algorithm:

> Circuits for linear algebra
>
> ... Sort all the integers j and
> pairs (i, j) in order of j ...

(2001.11 Bernstein, "Circuits
for integer factorization")

I speculate that routing works.

No evidence that it's bad.

Obviously worth exploring:

> Should j and (i, j) be assigned permanently to cells? . . . There is a huge literature on mesh routing and mesh sorting, with dozens of potentially useful techniques.

(2001.11 Bernstein, same paper)

But sorting *definitely* works
and isn't much slower.

Another choice that affects
both speed and confidence:
Which computers to use?

Some of the d dollars
will be spent buying
(or renting) computers.

Can buy special-purpose
computers; but *should* I?
What do I want in a computer?

Let's look at some options. . .

An old computer, the MasPar:
16384 parallel processors
in a 2-dimensional mesh,
each connected to neighbors.

200000 32-bit additions
per second per processor.
No longer sounds impressive.

"SIMD": global instructions
transmitted to all processors;
no need to store instructions
in each processor.

Was used for factorizations.

Currently available for \$50:
correlation-detector chip.

One billion times per second:
Given input bit sequences
$b_0, b_1, b_2, \ldots, b_{63}$,
$c_0, c_1, c_2, \ldots, c_{63}$,
computes $b_0c_0 + \cdots + b_{63}c_{63}$
and > 100 shifted correlations;
merges into a detector sequence.

The speed is inspirational.
Might try to use this for
factorization—but it clearly
was designed for something else.

Another interesting computer:
the human brain.

Roughly 10^{12} neurons
in a 3-dimensional mesh,
each connected to 100 neighbors.
Each neuron stores 1 byte,
performs 100 ops/second.

Designed for vision processing
and other pattern-matching tasks.
Hard to use for factorization.

Draws about 20 watts—
but relies on $>$ 100-watt
"body" for energy acquisition.

Another interesting computer:
the Internet.

Huge general-purpose computer.
"A powerful multicomputer,
much larger than a major city."

Includes millions of chips,
millions of network connections.

Notable difference from the
other computer examples:
the chips are considerably
faster than the connections.

Has been used for factorizations.

Many people are saying that special-purpose computers are much more cost-effective than general-purpose computers: speedups of 1000 or more for large factorization problems.

That's a terribly strange thing to say!

We normally think of a general-purpose computer as simulating *any* computer (up to a similar size) without much loss of price-performance ratio.

e.g. One 2-tape Turing machine can simulate any Turing machine with slowdown $T \mapsto O(T \lg T)$; reasonably small O constants.

e.g. Athlon quickly simulates G5.

Unless we want the last ounce of speed, we're happy with a general-purpose computer.

Lack of efficient simulation
tells us that a machine has a
basic architectural deficiency.

e.g. 1-tape Turing machines
cannot efficiently simulate
more tapes. Too local!
(Many easy test problems.)

e.g. 2-tape Turing machines
cannot efficiently simulate
random-access machines.
Too sequential!
(Harder to find test problems.)

What we're seeing now
in integer factorization:
random-access machines cannot
efficiently simulate circuits.

An easy test problem:
sort n integers in $[1, n]$.
(Many other test problems:
e.g., multiply n-bit numbers.)

Mesh-sorting circuit of
size $n^{1+o(1)}$ takes time $n^{1/2+o(1)}$.
Random-access machine of
size $n^{1+o(1)}$ takes time $n^{1+o(1)}$.
These $o(1)$'s are fairly small.

Architectural deficiency in
random-access machines:
no parallelism.

Bad fix: Discard the concept
of general-purpose computers;
throw away modularity and the
efficiencies of the mass market.

Good fix: Switch to a
better architecture—
a general-purpose computer
that *can* efficiently
simulate large circuits.

Don't want to lose confidence!

Extreme example: Don't want to assume quantum computers.
"Can they be built?"

(Cryptographers need to be ready with post-quantum cryptography in case they *are* built.)

Don't want to use dim-3 mesh.
Don't want long wires.
Don't want global RAM.
Don't want global clocks.
Don't want global instructions.
Don't want large chips.

Resulting computer architecture:
chip is a dimension-2
mesh of dinky little processors,
each connected to neighbors;
computer is a dimension-2
mesh of chips,
each connected to neighbors.

Clearly buildable at huge sizes,
cost scaling linearly with
number of chips—and can
still do fast sorting etc.

I'm now designing a DLP
plus mesh programming tools.

Compared to the Internet:
more parallelism in chips;
chips balanced with network.

Compared to Computational RAM:
a complete local network;
no global clocks;
larger DLPs.

...

Compared to the MasPar
(which was easy to program):
almost identical,
except no global clocks.

Assume a good computer.
What factorization algorithms
am I investigating?

Broadly classify NFS sieving
options by asymptotic price-
performance ratio for testing
smoothness of $y^{2+o(1)}$ numbers;
i.e., by scalability.

RAM sieving: $y^{3+o(1)}$.
Same for parallel trial division
(Georgia Cracker, TWINKLE).
Useful for Internet factorizations.

"There are several ways to achieve cost $y^{2.5+o(1)}$: parallel Pollard rho, for example, or sieving via Schimmler's algorithm" (2001.11 Bernstein, "Circuits for integer factorization," Section 5, "Circuits to find smooth numbers"). Same for TWIRL etc.

"Parallel ECM or HECM ... $y^{2+o(1)}$" (2001.11 Bernstein). Also $y^{2+o(1)}$, but clearly faster: sieving plus rho plus early-abort ECM (2001.11 Bernstein).

NFS price-performance ratio is
$\exp((\beta + o(1)) \sqrt[3]{(\log n)(\log \log n)^2})$
assuming standard conjectures.

sieving	linalg	β
RAM	RAM	2.85 . . . standard
RAM	RAM	2.76 . . . improved
mesh	RAM	2.37 . . .
mesh	mesh	2.36 . . .
ECM	RAM	2.08 . . .
ECM	mesh	1.97 . . .

(2.37, 1.97: 2001.11 Bernstein;
RAM 2.76: 2002.04 Pomerance)

Of course, $o(1)$ is not 0,
but can draw some conclusions
about large numbers:

- Linear-algebra choice is
  clearly much less important
  than sieving choice.
- Communication costs keep
  the price-performance exponent
  above the operation exponent.

Alternative: Apply ECM directly to pq.

Usually ignored: "Many more operations than NFS."

But simple algorithm, minimal communication. Easy to obtain confidence.

For speed, want a very fast multiplication circuit. Standard multipliers are suboptimal, even for 64 bits!