

Re: SF: Back to theory

Source: <http://sci.tech-archive.net/Archive/sci.math/2005-02/9498.html>

From: Tim Peters (*tim.one_at_comcast.net*)

Date: 02/26/05

Date: Sat, 26 Feb 2005 01:26:59 -0500

[Nora Baron, to JSH]

- > *You have made this so much more complicated than it needs to*
- > *be. Here is a simpler approach that may accomplish much the*
- > *same and it is also much more general.*
- >
- > *Assume M is the number to be factored. Pick a (small)*
- > *integer j . Let $T = M - j$. Thus T is a function of both*
- > *M and j .*
- >
- > *Factor T . Assume you have split it into two factors,*
- > *f and g . Thus $T = f * g$.*

As for James's methods, the test I'll talk about below tries all possible ways of splitting $T=f*g$ s.t. f and g are integers ≥ 1 and $f \geq g$. I'm skipping $f < g$ cases since $\text{gcd}(f-g, \text{whatever}) = \text{gcd}(g-f, \text{whatever})$ (someone may wish to sue me over this, but in my universe gcd always returns a non-negative result).

- > *Now let X be some rational function of f and g . One*
- > *possible choice might be,*
- >
- > $X = (f - g)/(f + g)$.
- >
- > *Finally, let $Y = M / X$. Thus $M = X * Y$.*
- >
- > *Note that X and Y are both functions of the factors of*
- > *T . Also both are rational numbers. There is some chance*
- > *that the numerator of X has a factor in common with M .*

It's always ambiguous in these cases whether it's assumed that rationals are or aren't expressed in normalized form, where by "normalized" I mean $\text{gcd}(\text{numerator}, \text{denominator}) = 1$. I'm not sure it matters here to the results. It matters pragmatically quite a bit, because another gcd calculation is required to create a normalized rational. The program below doesn't do that, so doesn't even bother to compute $f+g$. It just tries $\text{gcd}(f-g, M)$ for all $\langle f, g \rangle$ splittings as mentioned above. If it had to compute a normalized X , the count of the number of gcd s performed reported

below would double.

- > *This is, certainly, surrogate factoring. Really, your*
- > *underlying idea is not that different. What you are doing*
- > *essentially is finding a more complex function to define*
- > *X as a function of f and g.*

Of course I agree, but I'm almost as insignificant as you are these days
<wink>.

- > *Here is how this might work with $M = 15$. Let $j = 1$.*
- > *Then $T = M - j = 14$. You factor T as $f * g = 7 * 2$. Then*
- > *you note that*
- >
- > *$X = (7 - 2) / (7 + 2) = 5/9$.*
- >
- > *Right there, in the numerator, you have a factor that*
- > *divides M .*
- >
- > *Interestingly, the denominator, 9, also has a factor in*
- > *common with M .*
- >
- > *Let's try it on a bigger number. Say $M = 77$. This time let*
- > *$j = 2$. $T = 75 = 3 * 5 * 5$. Let $f = 25$ and $g = 3$. Then*
- >
- > *$X = (f - g) / (f + g) = (25 - 3) / (25 + 3) = 22/28 = 11/14$.*
- >
- > *Note that the numerator of X , namely 11, is a factor of 77.*
- > *And again, the denominator, 14, also has a factor in common*
- > *with 77.*
- >
- > *Pretty amazing, eh?*

Heh. Heh heh.

- > *I am a little surprised myself. I typed out this whole*
- > *message in the last 15 minutes. I just made up the function*
- > *that defines X , $X = (f - g) / (f + g)$. I could have chosen an*
- > *infinity of other functions. Also, quite honestly, I just*
- > *made up the two examples. I must confess that I did try $j = 1$*
- > *with the second example mentally, and saw it was not going*
- > *to work, so I tried $j = 2$. No other trial-and-error.*

OK, I plugged this into the same test scaffolding I used to test assorted
JSH algorithms. In all cases, I used the sequence $j=1, j=2, j=3, \dots$ until
a factor was found.

Despite that it's using much smaller integers than JSH's methods, it runs
much slower than those. I expected that. Despite sometimes-hysterical
posts about this point, factoring James's $T=M^2-j^2$ truly isn't hard, and
then iterating over all splittings of T^6 can generate an enormous number of

gcds to try. Iterating over the splittings of the comparatively teensy $T=M-j$ in `_this_` method yields far fewer splittings to try. Since most j "don't work" here either as M gets larger, we end up needing to `_factor_` much more often in this method, and factoring is more expensive than doing gcds (I noted once in `sci.crypt` that I've never hit a case in James's assorted algorithms that required using a j larger than $\log_2(M)$ except when M was the square of a prime; you already know that j much larger than that is sometimes needed in this method; but note that JSH methods can burn thru hundreds of millions of gcds before j even reaches 10).

So it's slow for that reason, but apparently for another too. Here's "the standard test" output after picking 1000 pairs of 15-bit primes at random (I gave a precise account of what that means in an earlier post), and using this algorithm to factor their product:

```
nbits 15: 1000 of 1000 factored, 100%
gcds: actual 23,150,579 expected random 12,122,784 no factor 0
```

So all of them factored, which you expected. "actual" is the total number of gcds it actually computed (and would be twice this if it normalized X first). "expected random" is the expected (mean) number of gcds that would have been needed to factor the same products by a method that repeatedly picks k purely at random from $1..M-1$ until $1 < \gcd(k, M) < M$. "no factor" is the number of gcds wasted on products that failed to factor (of which there are none here, so that's 0).

It's not surprising that it's doing worse than random-gcd, as measured by number of gcds required. It's possibly surprising that it's doing worse in this respect than JSH's latest specified algorithm, but not if you've tested a lot of these things <wink>. I can slash the number of gcds needed, and speed it by a factor > 3 , just by iterating over all the ways to split T^6 (instead of plain T) as $f*g$. But that leads to a much bigger surprise! Like so:

```
nbits 15: 1000 of 1000 factored, 100%
gcds: actual 7,765,039 expected random 12,139,989 no factor 0
```

Congratulations! This is the first "surrogate factoring" algorithm I've tested that performs better than random-gcd (well, it's still much slower than random-gcd, but time isn't the measure here). I suggest we christen it the Baron-Harris method <LOL>. I don't have time to think about "why" now, but it sure was a surprise to me -- all I had in `_mind_` by moving to T^6 was to increase the quality of the pseudorandom number generator "f-g".

Looks like that holds for products of 20-bit primes too (these take a `_lot_` longer to run, so I stopped after 75 -- out of time for tonight):

```
nbits 20: 75 of 75 factored, 100%
gcds: actual 13,320,947 expected random 28,033,396 no factor 0
```

Just one example from that run, as a sanity check:

$M = 533237 * 755309 = 402758705233$
 $j = 23$
 $T = M - j = 402758705210 = 2 \ 5 \ 19 \ 4931 \ 429889$
 $T^6 = \text{obvious}$
 $f = 501359048354607032677714075483363995707002203040$
 $g = 8513747362502192556250$
check for yourself that $T^6 = f * g$ (it does)
 $\text{gcd}(f - g, M) = 533237$ bingo
number of gcds tried = 156,337

Sanity checked.

- > *What's the point?*
- >
- > *The point is, I have modified and greatly simplified your*
- > *central idea – and at the same time, greatly generalized it –*
- > *and on these two simple little examples – the only ones*
- > *I have tried – it works. It might work on lots of other*
- > *examples. If it doesn't, I can try changing the function*
- > *that defines X. I can add parameters, like your A, y, and z.*
- > *Your central theme, surrogate factoring of the number*
- > *$T = M - j$, is still there, and it might even explain why*
- > *it works (if it does).*

Not to mention that Baron–Harris probably always finds a factor, and seemingly needs fewer gcds than poke–and–hope.

- > *After all, the factors of M ought to be related somehow to the factors*
- > *of T.*

There doesn't currently seem to be a good reason to believe that's the case, right? For example, if you could deduce the factors of M from those of $M - j$ or $M + j$ efficiently, you could pick up a pile of RSA challenge checks tomorrow. Here's a start: if you know all the primes dividing $N - 1$ or $N + 1$, you don't have to bother checking whether they divide N <wink — but all the results I've read about this are as trivial and unexploitable as that one>.

- > *In the case of RSA numbers, in general you expect T to be easier*
- > *to factor than M.*

Yup. The other half is that if M could be factored from a factorization of T efficiently, it's more than a little puzzling that nobody yet has found a way to do that. Well, maybe tomorrow's Baron–Harris–Decker–Filho method will finally crack that nut.