

Re: Surrogate factoring, corrected algorithm

Source: <http://sci.tech-archive.net/Archive/sci.math/2005-02/6401.html>

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

Date: 02/17/05

Date: Wed, 16 Feb 2005 23:20:36 -0500

[JSH]

[...]

> *Then the corrected algorithm is as follows.*

As far as I can see, the only difference from the weekend's algorithm is that this one iterates over splittings of $T^3 j^2$ instead of over $T j^2$. I'll note up front that cubing T can enormously increase the number of distinct $f_1 f_2$ splittings, and so also the number of gcds tried.

> *Given an odd natural number M to be factored.*

>

> *Select j . Simplest is to let $j = \text{floor}(M/2)$.*

>

> *If j is even, add 1.*

I'm ignoring that part (I'm not avoiding even j), just because it never really mattered before.

> *Calculate T .*

>

> $T = M^2 - j^2$.

>

> *Factor T , and j .*

>

> *Iterate through integer f_1 and f_2 such that $f_1 f_2 = T^3 j^2$.*

As above, this appears to be the only difference from the weekend's algorithm, $s/T/T^3/$. Correct?

> *For each f_1 and f_2 calculate w using*

>

> $w = (f_1 - f_2) + 2Tj^2$

>

> *and*

>

> $w = (f_2 - f_1) + 2Tj^2$

sci.math: Re: Surrogate factoring, corrected algorithm

OK, over the weekend these were `_named_ "Ax"` instead of "w" — but same equations otherwise.

- > to handle plus or minus, and take the gcd of w with M.
- >
- > The gcd of w with M will be a single prime factor of M, for at least
- > one set of f_1 and f_2 .

Sorry, it didn't work for me. I started this time with my running example, $M = 112554401 * 221667653$ (I like that one because it's small enough so that the T's are easy to factor quickly by other simple methods, but its smallest factor is large enough that luck has scant chance of succeeding quickly).

The two previous versions of this algorithm didn't factor that M at any j in 1 thru 623, but succeeded at $j=624$.

This version didn't factor that M at any j in 1 thru 16, but did succeed once at $j=17$. In all, 62,320,128 gcds were tried through $j=17$, with 1 success.

$j=17$ was painfully expensive. Then $T^3 j^2 =$

$$2^9 3^6 5^3 7^3 11^3 17^2 191^3 7841^3 53633^3 56197^3 \\ 335417^3 14832053^3$$

and so there are

$$10 * 7^4 * 4^4 * 4^3 * 4^4 * 4^4 * 4^4 * 4^2 = 27,525,120$$

distinct ways to split $T^3 j^2$ as the product $f_1 f_2$ with f_1 and f_2 both integers ≥ 1 , and without regard to order; and so twice as many as that = 55,050,240 gcds tried at $j=17$ alone. In comparison, all of $j=1$ through $j=16$ computed a grand total of 7,269,888 gcds (with no success).

Of course the most troublesome bit is that factors still aren't found at some j, so at least one of {proof, algorithm, implementation} is wrong. I won't give a smaller failing example unless you say you'll actually look at it.

Implementation note: it's easiest to generate all $\langle f_1, f_2 \rangle$ splittings without trying to weed out reversals. For example, if $\langle 3, 5 \rangle$ is generated, also generate $\langle 5, 3 \rangle$. But then trying to weed out reversals later, by keeping track of the ones already seen, can require an enormous amount of memory, because there can be so many splittings now. In fact I killed my first run of the program during $j=17$, because the VM size had ballooned to over 1.2 gigabyte ($f_1 f_2 = T^3 j^2$, so these are quite large integers), and was hogging the whole machine.

So I changed the program to stop trying to weed out reversals, and then simply threw away the

$$w = (f_2 - f_1) + 2Tj^2$$

step. For if, e.g., $\langle 3, 5 \rangle$ and $\langle 5, 3 \rangle$ are both generated by the splitter, then the

$$w = (f_2 - f_1) + 2Tj^2$$

equation at $\langle 3, 5 \rangle$ will be computed by the first

$$w = (f_1 - f_2) + 2Tj^2$$

equation at $\langle 5, 3 \rangle$, and likewise the

$$w = (f_2 - f_1) + 2Tj^2$$

at $\langle 5, 3 \rangle$ is computed by the first

$$w = (f_1 - f_2) + 2Tj^2$$

at $\langle 3, 5 \rangle$. For similar reasons, there's no need to consider $\langle -f_1, -f_2 \rangle$ either (e.g., $-f_1 - -f_2 = f_2 - f_1$). It's only necessary to use the first w equation for all $f_1, f_2 \geq 1$ s.t. $f_1 f_2 = T^3 j^2$; the second w equation, and negations, necessarily lead to redundant gcd computations then, so can be skipped.