

## Re: SF: Back to theory

**Source:** <http://sci.tech-archive.net/Archive/sci.math/2005-02/10026.html>

---

**From:** Tim Peters (*tim.one\_at\_comcast.net*)

**Date:** 02/28/05

Date: Sun, 27 Feb 2005 22:59:57 -0500

[Tim Peters]

[...]

> *I want to elaborate on that, because some results may be artifacts of  
> the way primes are chosen. I pick an N-bit prime like so: pick a  
> random integer i uniformly from the range  $2^{*(N-1)} \leq i < 2^{*N}$ , and  
> then find the smallest prime  $\geq i$ . But not all primes in range are  
> equally likely to get picked -- due to the way I'm choosing them, the  
> probability of picking a particular prime p is proportional to 1 + the  
> number of composites immediately preceding p. I can't imagine why that  
> would bias the results, but can't swear that it doesn't. Another thing  
> to test, anyway.*

Doesn't appear to matter. Changed the code to precompute a list of all primes p s.t.  $2^{*(N-1)} \leq p < 2^{*N}$ , and then picked primes uniformly at random from that list.

[...]

> *nbits 20: 1000 of 1000 factored, 100%  
> gcds: actual 209,981,730 expected random 383,897,119 no factor 0*

That was from using gcd(f-g, M) over all  $f*g = (M-j)^6$ , for j=1, j=2, ... until a factor was found.

Stats were indistinguishable after changing the way primes got picked.

Details:

nbits 20: 1000 of 1000 factored, 100%  
gcds: actual 208,382,882 expected random 386,954,503 no factor 0  
algorithm beat random-gcd 844 of 1000 factoring cases  
random - actual: min -1578591  
          mean 178571.621  
          max 494966  
          median(s) [227482, 229270]

Smallest winning j; one '\*' per 2 (or fraction thereof) cases

1 \*\*\*\*\*

2 \*\*\*\*\*  
3 \*\*\*\*\*  
4 \*\*\*\*\*  
5 \*\*\*\*\*  
6 \*\*\*\*\*  
7 \*\*\*\*\*  
8 \*\*\*\*\*  
9 \*\*\*\*\*  
10 \*\*\*  
11 \*\*\*\*\*  
12 \*\*\*\*  
13 \*\*\*\*\*  
14 \*\*\*\*  
15 \*\*\*\*\*  
16 \*\*\*\*  
17 \*\*\*\*\*  
18 \*\*  
19 \*\*\*\*\*  
20 \*\*  
21 \*\*\*\*\*  
22 \*\*\*  
23 \*\*\*\*\*  
24 \*\*\*  
25 \*\*\*\*\*  
26 \*\*\*\*  
27 \*\*\*\*\*  
28 \*\*\*\*  
29 \*\*\*\*\*  
30 \*  
31 \*\*\*\*\*  
32 \*\*\*  
33 \*  
34 \*\*  
35 \*\*\*\*\*  
36 \*  
37 \*\*\*\*\*  
38 \*  
39 \*\*\*\*  
41 \*\*\*\*\*  
42 \*  
43 \*\*\*\*  
44 \*  
45 \*  
47 \*\*\*\*  
49 \*\*\*  
51 \*\*\*  
53 \*\*\*  
55 \*\*\*\*\*  
57 \*\*  
58 \*  
59 \*\*\*

```

60 *
61 **
63 *
65 *
66 *
70 *
71 *
77 *
79 *
80 *
81 *
83 *
85 *
86 *
94 *
137 *

```

[...]

Sanity check: Here's a run against an implementation that `_does_` pick `f` and `g` at random repeatedly, from `1..M-1`, until `gcd(f-g, M)` reveals a factor. It's the same test scaffolding, and the point of this is to see whether an error in the scaffolding could lead me to conclude that "this doesn't look random" too (note that I cut this run back to 15-bit primes, just to speed it up):

```

nbits 15: 1000 of 1000 factored, 100%
gcds: actual 11,658,968 expected random 12,083,917 no factor 0
algorithm beat random-gcd 631 of 1000 factoring cases
random - actual: min -82310
               mean 424.949
               max 15385
               median(s) [4044, 4094]

```

So those were within spitting distance. It was a surprise to me that the actual algorithm beat the theoretical one in 63% of the trials, but on reflection something close to that is expected. The `_means_` between actual and theoretical should be about the same, and  $11,658,968/1000$  is close to  $12,083,917/1000$ . But the "# of trials until first success" statistic follows the geometric distribution, which is monotonically decreasing, so the mostly likely outcome is one trial before success, the second-most likely two trials, and so on: we're more likely to beat the mean than to exceed it.

A "typical" example for that run was  $p=32633$  and  $q=19993$ . The probability of success on a trial is then  $(p+q-2)/(M-1) \approx 8.0658e-5$ , and the probability of failure  $y = 1-x \approx 0.999919342$ . The mean # of trials before success is  $1/x \approx 12398$ . The probability that we'll succeed before 12398 trials is the sum for  $i$  in  $1..12397$  of  $y^{i-1} * x \approx 0.632$ . That's suspiciously close to the  $631/1000$  observed, but that's really the first example I looked at.

I'll pick another from that run: 26371 \* 19577,  $x=(p+q-2)/(M-1)$ ,  $y=1-x$ ,  $1/x \approx 1.1236$ , and sum for  $i$  in  $1..11235$  of  $y^{(i-1)*x} \approx 0.632$  again. Since the primes here all have the same number of bits, it's not going to vary much.

I conclude there's no reason to suspect an error in the test scaffolding, or in the theoretical analysis of random  $f-g$  behavior.

[Nora Baron]

> *Certainly your experiments look like a lot more than chance*  
> *deviations from poke-and-hope expected. Try  $X = f^2 - g^2$  also,*  
> *if you don't mind.*

Obliging: as explained in another post, picking integers  $f$  and  $g$  truly at random and computing

$$\gcd(f-g, M)$$

for a two-prime product  $M$  should have exactly the same probability of revealing a factor as picking a single integer  $f$  at random and computing

$$\gcd(f, M)$$

But

$$\gcd(f^2 - g^2, M)$$

should do better than that. I didn't finish analyzing that because my primary interest is still finding out why the simpler  $f-g$  is doing better than random here.

Anyway, same algorithm as reported on above, except chewing through

$$\gcd(f^2 - g^2, M)$$

for all  $f*g = (M-j)^6$ , for  $j=1, j=2, \dots$ , until a factor is found. As a practical matter:

$$\begin{aligned} \gcd(f^2 - g^2, M) &= \\ \gcd((f+g)*(f-g), M) &= \\ \gcd((f+g)*(f-g) \bmod M, M) &= \\ \gcd((f \bmod M + g \bmod M)*(f \bmod M - g \bmod M), M) & \end{aligned}$$

so I work with  $f \bmod M$  and  $g \bmod M$  instead of with  $f$  and  $g$  directly. Since  $f$  and/or  $g$  are large compared to  $M$  ( $f*g = (M-j)^6$  with  $j$  usually very small compared to  $M$ ), working mod  $M$  slashes the size of the integers involved, speeding the arithmetic without affecting final results.

nbits 20: 1000 of 1000 factored, 100%  
gcds: actual 153,958,462 expected random 385,974,211 no factor 0  
algorithm beat random-gcd 913 of 1000 factoring cases  
random - actual: min -738967

mean 232015.749  
max 506577  
median(s) [270688, 270983]

Smallest winning j; one '\*' per 2 (or fraction thereof) cases

1 \*\*\*\*\*  
2 \*\*\*\*\*  
3 \*\*\*\*\*  
4 \*\*\*\*\*  
5 \*\*\*\*\*  
6 \*\*\*  
7 \*\*\*\*\*  
8 \*\*\*\*\*  
9 \*\*\*\*\*  
10 \*\*\*\*\*  
11 \*\*\*\*\*  
12 \*\*\*\*\*  
13 \*\*\*\*\*  
14 \*\*\*  
15 \*\*\*\*\*  
16 \*\*\*\*\*  
17 \*\*\*\*\*  
18 \*\*  
19 \*\*\*\*\*  
20 \*\*  
21 \*\*\*\*\*  
22 \*\*  
23 \*\*\*\*\*  
24 \*\*\*  
25 \*\*\*\*\*  
26 \*\*  
27 \*\*\*\*\*  
28 \*\*\*  
29 \*\*\*\*\*  
30 \*  
31 \*\*\*\*\*  
32 \*\*\*  
33 \*\*\*\*\*  
34 \*  
35 \*\*\*\*\*  
37 \*\*\*\*\*  
38 \*\*  
39 \*  
40 \*\*  
41 \*\*\*\*\*  
42 \*  
43 \*\*\*  
44 \*  
47 \*\*  
49 \*\*\*\*\*

51 \*  
 52 \*  
 53 \*\*  
 55 \*  
 56 \*  
 57 \*  
 59 \*  
 62 \*  
 65 \*  
 67 \*  
 71 \*  
 73 \*  
 76 \*  
 77 \*  
 85 \*  
 101 \*

Example:

$M = 667309 * 950813 = 634486072217$   
 $j = 5$   
 $T = M - j = 634486072212 = 2^2 3^3 31 5623 33703$   
 $T^6 = \text{obvious}$   
 $f = 2606116434608914667447816958799078656$   
 $g = 25034489761792661602095127717552464$   
 check that  $f * g = T^6$  yup  
 $f' = f \text{ mod } M = 26328009161$   
 $g' = g \text{ mod } M = 193985391557$   
 $\text{gcd}((f'+g')*(f'-g'), M) = 667309$  bingo ( $f'-g'$  gave this)  
 gcds tried 15,587

You probably wanted me to try  $f^2 - g^2$  with  $f * g = T$  instead of  $T^6$ ? I started to, but it would take unreasonably long to complete a 1000-product test run. Same reason as before: not enough splittings per  $j$  to have a decent shot at success per  $j$ , so have to use many  $j$ , so have to factor many  $T$ 's, and factoring is much more expensive than a gcd. For example, the first one it happened to try was  $M = 768353 * 896647$ , and that didn't factor until  $j=9041$ , with  $f=5503650$  and  $g=125179$ . It computed fewer gcds in all (128079) to find a factor than expected-random (413779), but factoring 9041 numbers about the same size as  $M$  to get there is very expensive (note that it averaged only  $128079/9041 \approx 14$  gcds per  $T$  factored!). The next few it tried:

$M = 635729 * 840557$  factored at  $j=7913$   
 $M = 718973 * 949129$  factored at  $j=3189$   
 $M = 543203 * 639697$  factored at  $j=12221$   
 $M = 766999 * 951941$  factored at  $j=17531$

and I started to nod off waiting for that last one to crack.