An $O(M(n)\log n)$ algorithm for the Jacobi symbol

Paul Zimmermann
(joint work with Richard P. Brent)

INSTITUT NATIONAL DE RECHERCHE EN INFORMATIQUE ET EN AUTOMATIQUE | INRIA
centre de recherche NANCY - GRAND-EST

University of Leiden
January 20, 2010

From: Galbraith Steven
Date: Fri, Apr 17, 2009 at 4:26 PM
To: Paul Zimmermann, Pierrick Gaudry

Hi Paul and Pierrick,

Sorry to bother you.

The usual algorithm to compute the Legendre (or Jacobi) symbol
is closely related to Euclid's algorithm. There are variants of
Euclid for n-bit integers which run in O(M(n) \log(n)) bit operations.
Hence it is natural to expect a O(M(n) \log(n)) algorithm for
Legendre symbols.

I don't see this statement anywhere in the literature. Is this:

(a) in the literature somewhere
(b) so obvious no-one ever wrote it down
(c) false due to some subtle reason.

Thanks for your help.

Regards
Steven

(b) so obvious no-one ever wrote it down

This is what we first thought.

(b) so obvious no-one ever wrote it down

This is what we first thought.

However we soon realized it was not so easy...

(a) in the literature somewhere

Two MSB algorithms:

- “Algorithmic Number Theory” from Bach and Shallit, solution of Exercise 5.52 (Gauss, Bachmann) [sketch];
- a simpler algorithm mentioned by Schönhage in his “TP book”, but without details.

As far as we know, no subquadratic implementation exists, except that of Schönhage in the TP language.

Sage 4.3 (`sagemath.org`) on a 1.6Ghz Core 2 Duo:

```
sage: a=3^209590
sage: b=5^143067
sage: a.ndigits(), b.ndigits()
(100000, 100000)
sage: %timeit a.gcd(b)
10 loops, best of 3: 76.1 ms per loop
sage: %timeit a.jacobi(b)
10 loops, best of 3: 2.01 s per loop
```

(c) false due to some subtle reason

We'll try to show this is wrong in this talk!

The Jacobi symbol

$\left(\frac{b}{a}\right)$ or $(b|a)$ is defined for integers a, b, with *a odd positive*.

$$(b|a) = (b \bmod a|a)$$

$$(b|a) = (-1)^{(a-1)(b-1)/4}(a|b) \qquad \text{for } b \text{ odd positive}$$

$$(bc|a) = (b|a)(c|a)$$

$$(2|a) = (-1)^{(a^2-1)/8}$$

$$(-1|a) = (-1)^{(a-1)/2}$$

$$(b|a) = 0 \qquad \text{if } (a,b) \neq 1$$

In this talk we will propose a LSB algorithm, that can be easily implemented in $O(M(n)\log n)$ from a LSB gcd implementation.

We assume a is odd positive, b is even positive.
If b is negative, use $(b|a) = (-1)^{(a-1)/2}(-b|a)$.
If b is odd, use $(b|a) = (b+a|a)$.

Plan of the talk

- The Binary (Generalized) Division
- A Cubic LSB Algorithm
- A Quadratic LSB Algorithm
- A Subquadratic LSB Algorithm
- Implementation and Timings

The Binary Division

The Binary Division (and GCD)

A binary recursive gcd algorithm, Stehlé and Z., ANTS VI, 2004.

Classical (MSB) division forces 0's in the MSBs:

935	1110100111
714	1011001010
221	0011011101
51	0000110011
17	0000010001
0	0000000000

Binary Division (and GCD)

Binary (LSB) division forces 0's in the LSBs:

935	1110100111
714	1011001010
1292	1010000100
1360	1010101000
1632	1100100000
2176	1000100000000
0	0000000000000

Theory

$a, b \in \mathbb{Z}$ with $\nu_2(b) > \nu_2(a)$
$j = \nu_2(b) - \nu_2(a)$
There is a unique $|q| < 2^j$ such that $\nu_2(b) < \nu_2(r)$ and:

$$r = a + q2^{-j}b$$

q is the *binary quotient* of a by b
r is the *binary remainder* of a by b

Rationale: if a, b have both n bits, $b' = 2^{-j}b$ has $n - j$ bits, and qb' has about n bits, thus r has about the same bit-size as a, but at least $j + 1$ more zeros in the LSB.

Practice

$$j = \nu_2(b) - \nu_2(a) > 0$$

$$q \equiv -a/(b/2^j) \bmod 2^{j+1} \quad \text{(centered)}$$

Advantages of the Binary Division

- ⊕ simpler to compute (division mod 2^{j+1} instead of MSB division);
- ⊕ no "repair step" in the subquadratic GCD (see however Möller, Math. Comp., 2008);
- ⊕ an average reduction of two LSB bits per iteration;
- ⊖ an average increase of 0.05 MSB bit per iteration (analyzed precisely by Daireaux, Maume-Deschamps and Vallée, DMTCS, 2005).

Using the Binary Division for the Jacobi Symbol

It seems easy to adapt, using $b' = b/2^j$ odd:

$$(b|a) = (-1)^{j(a^2-1)/8}(b'|a)$$

$$(b'|a) = (-1)^{(a-1)(b'-1)/4}(a|b')$$

$$(a|b') = (a + qb'|b') = (r|b')$$

$$(r|b') = (-1)^{j(b'^2-1)/8}(r/2^j|b')$$

Using the Binary Division for the Jacobi Symbol

It seems easy to adapt, using $b' = b/2^j$ odd:

$$(b|a) = (-1)^{j(a^2-1)/8}(b'|a)$$

$$(b'|a) = (-1)^{(a-1)(b'-1)/4}(a|b')$$

$$(a|b') = (a + qb'|b') = (r|b')$$

$$(r|b') = (-1)^{j(b'^2-1)/8}(r/2^j|b')$$

However r can be negative!
Example: 935, 738, 1304, -240, 1184, -832, 768, -1024, 0.

Incompatible with fast binary GCD, which works with $r \bmod 2^{2k+1}$.

A Cubic LSB Algorithm

Binary Division with Positive Quotient

Instead of taking $q = a/(b/2^j)$ in $[-2^j, 2^j]$, take it in $[0, 2^{j+1}]$.

Since $q > 0$, if $a, b > 0$, all terms are non-negative.

Stopping GCD criterion: $a/2^{\nu_2(a)} = b/2^{\nu_2(b)}$.

Example: $935, 714 = 357 \cdot 2, 1292 = 323 \cdot 2^2, 1360 = 85 \cdot 2^4,$
$1632 = 51 \cdot 2^5, 2176 = 17 \cdot 2^7, 4352 = 17 \cdot 2^8.$

A Cubic LSB Algorithm

Algorithm CubicBinaryJacobi.

Input: $a, b \in \mathbb{N}$ with $\nu(a) = 0 < \nu(b)$

Output: Jacobi symbol $(b|a)$

1: $s \leftarrow 0$
2: $j \leftarrow \nu(b)$
3: **while** $2^j a \neq b$ **do**
4: $\quad b' \leftarrow b/2^j$
5: $\quad (q, r) \leftarrow \text{BinaryDividePos}(a, b)$
6: $\quad s \leftarrow (s + \frac{j(a^2-1)}{8} + \frac{(a-1)(b'-1)}{4} + \frac{j(b'^2-1)}{8}) \bmod 2$
7: $\quad (a, b) \leftarrow (b', r/2^j)$
8: $\quad j \leftarrow \nu(b)$
9: **if** $a = 1$ **then** return $(-1)^s$ **else** return 0

Cost of the Cubic Algorithm

Let n be the bit-size of the inputs a, b.

Each iteration costs $O(n)$ (unless j is large, but this is unlikely, and in this case (a, b) decrease even more).

The number of iterations is $O(n^2)$ (see below).

Thus the total cost is $O(n^3)$.

A Quadratic LSB Algorithm

Lemma

The quantity $a + 2b$ is non-increasing in CubicBinaryJacobi.

Proof.

At each iteration, $a + 2b$ becomes:

$$\frac{2a}{2^j} + (1 + \frac{2q}{2^j})\frac{b}{2^j}.$$

If $j \geq 2$, $a + 2b$ is multiplied by a factor at most $9/16$: *good* iteration.

Lemma

The quantity $a + 2b$ is non-increasing in CubicBinaryJacobi.

Proof.

At each iteration, $a + 2b$ becomes:

$$\frac{2a}{2^j} + (1 + \frac{2q}{2^j})\frac{b}{2^j}.$$

If $j \geq 2$, $a + 2b$ is multiplied by a factor at most $9/16$: *good* iteration.
If $j = 1$ and $q = 1$, $a + 2b$ decreases, but with a factor that can be arbitrarily close to 1: *bad* iteration.

Lemma

The quantity $a + 2b$ is non-increasing in CubicBinaryJacobi.

Proof.

At each iteration, $a + 2b$ becomes:

$$\frac{2a}{2^j} + (1 + \frac{2q}{2^j})\frac{b}{2^j}.$$

If $j \geq 2$, $a + 2b$ is multiplied by a factor at most $9/16$: *good* iteration.
If $j = 1$ and $q = 1$, $a + 2b$ decreases, but with a factor that can be arbitrarily close to 1: *bad* iteration.
If $j = 1$ and $q = 3$, $a + 2b$ remains unchanged: *ugly* iteration.

Good iteration: $a = 9, b = 4$ gives $j = 2, q = 7$, $b' = 1, r/2^j = 4$, $a + 2b = 17$ becomes 9.

Good iteration: $a = 9, b = 4$ gives $j = 2, q = 7$, $b' = 1, r/2^j = 4$, $a + 2b = 17$ becomes 9.

Bad iteration: $a = 9, b = 6$ gives $b' = 3, r/2^j = 6$, $a + 2b = 21$ becomes 15.

Good iteration: $a = 9, b = 4$ gives $j = 2, q = 7$, $b' = 1, r/2^j = 4$, $a + 2b = 17$ becomes 9.

Bad iteration: $a = 9, b = 6$ gives $b' = 3, r/2^j = 6$, $a + 2b = 21$ becomes 15.

Ugly iteration: $a = 9, b = 10$ gives $b' = 5, r/2^j = 12$, $a + 2b = 29$ remains 29.

Lemma

If $\mu = \nu(a - b/2)$, there are exactly $\lfloor \mu/2 \rfloor$ ugly iterations starting from (a, b), followed by a good iteration if μ is even, otherwise by a bad iteration.

Example 1: $a - b/2 = 64 = 2^6$

$$(85, 42) \underbrace{\to}_{ugly} (21, 74) \underbrace{\to}_{ugly} (37, 66) \underbrace{\to}_{ugly} (33, 68) \underbrace{\to}_{good} (34, 38) \cdots$$

Example 2: $a - b/2 = 128 = 2^7$

$$(149, 42) \underbrace{\to}_{ugly} (21, 106) \underbrace{\to}_{ugly} (53, 90) \underbrace{\to}_{ugly} (45, 94) \underbrace{\to}_{bad} (47, 46) \cdots$$

A Quadratic LSB Algorithm

Main idea: from the 2-valuation of $a - b/2$, compute the number $m > 0$ of consecutive ugly iterations, and apply them all at once: *harmless* iteration.

The Jacobi symbol can also be easily updated for m consecutive ugly iterations.

Now we have only good (G), bad (B), or harmless (H) iterations, where HH is forbidden.

Algorithm QuadraticBinaryJacobi

1: $s \leftarrow 0, \quad j \leftarrow \nu(b), \quad b' \leftarrow b/2^j$
2: **while** $a \neq b'$ **do**
3: $\quad s \leftarrow (s + j(a^2-1)/8) \bmod 2$
4: $\quad (q, r) \leftarrow \text{BinaryDividePos}(a, b)$
5: $\quad$ **if** $(j, q) = (1, 3)$ **then** ▷ harmless iteration
6: $\quad\quad d \leftarrow a - b'$
7: $\quad\quad m \leftarrow \nu(d) \operatorname{div} 2$
8: $\quad\quad c \leftarrow (d - (-1)^m d/4^m)/5$
9: $\quad\quad s \leftarrow (s + m(a-1)/2) \bmod 2$
10: $\quad\quad (a, b) \leftarrow (a - 4c, b + 2c)$
11: $\quad$ **else** ▷ good or bad iteration
12: $\quad\quad s \leftarrow (s + (a-1)(b'-1)/4) \bmod 2$
13: $\quad\quad (a, b) \leftarrow (b', r/2^j)$
14: $\quad s \leftarrow (s + j(a^2-1)/8) \bmod 2, \quad j \leftarrow \nu(b), \quad b' \leftarrow b/2^j$
15: **if** $a = 1$ **then** return $(-1)^s$ **else** return 0

Analysis of the Quadratic Algorithm

Lemma

Algorithm QuadraticBinaryJacobi needs $O(n)$ iterations.

Proof.

Consider a block of three iterations (G, B, or H):

- G multiplies $a + 2b$ by at most $9/16 < 5/8$;
- HH is forbidden, thus we have either $HB = U^m B$ or BB;
- UB multiplies $a + 2b$ by at most $5/8$, and U^{m-1} leaves it unchanged;
- BB multiplies $a + 2b$ by at most $1/2 < 5/8$.

Thus each three iterations multiply $a + 2b$ by at most $5/8$, thus the number of iterations if $cn + O(1)$, where $c = 3/\log_2(8/5) \approx 4.4243$. □

A Subquadratic LSB Algorithm

A Subquadratic Algorithm

The subquadratic algorithm (HalfBinaryJacobi) is based on the quadratic one:

- using the same structure as classical half-gcd algorithms;
- using binary division with positive quotient;
- one call of HalfBinaryJacobi transforms (a, b) into (c, d), returns the corresponding 2×2 matrix R and an integer s such that:
$$(b|a) = (-1)^s (d|c).$$
- in some steps we have to "truncate" the inputs less than in a classical half-gcd, to guarantee the Jacobi symbol is correct;
- we have to "split" some harmless iterations to guarantee the algorithm invariant.

Algorithm HalfBinaryJacobi.

Input: $a \in \mathbb{N}, b \in \mathbb{N} \cup \{0\}$ with $0 = \nu(a) < \nu(b)$, and $k \in \mathbb{N}$
Output: two integers s, j and a 2×2 matrix R

1: if $\nu(b) > k$ then return $0, 0, [1, 0; 0, 1]$
2: $k_1 \leftarrow \lfloor k/2 \rfloor$, $a_1 \leftarrow a \bmod 2^{2k_1+2}$, $b_1 \leftarrow b \bmod 2^{2k_1+2}$
3: $s_1, j_1, R \leftarrow \text{HalfBinaryJacobi}(a_1, b_1, k_1)$
4: $a' \leftarrow 2^{-2j_1}(R_{1,1}a + R_{1,2}b)$, $b' \leftarrow 2^{-2j_1}(R_{2,1}a + R_{2,2}b)$
5: $j_0 \leftarrow \nu(b')$, if $j_0 + j_1 > k$ then return s_1, j_1, R
6: $s_0 \leftarrow j_0({a'}^2 - 1)/8 \bmod 2$
7: $q, r \leftarrow \text{BinaryDividePos}(a', b')$, $b'' \leftarrow b'/2^{j_0}$
8: **if** $(j_0, q) = (1, 3)$ **then** ▷ harmless iteration
9: $d \leftarrow a' - b''$, $m \leftarrow \min(\nu(d) \text{ div } 2, k - j_1)$
10: $c \leftarrow (d - (-1)^m d/4^m)/5$, $s_0 \leftarrow s_0 + m(a' - 1)/2 \bmod 2$
11: $(a_2, b_2) \leftarrow (a' - 4c, 2(b'' + c))$
12: $Q \leftarrow [(4^m + 4(-1)^m), 2(4^m - (-1)^m); 2(4^m - (-1)^m), (4^{m+1} + (-1)^m)]/5$
13: **else** ▷ good or bad iteration
14: $s_0 \leftarrow s_0 + (a' - 1)(b'' - 1)/4 \bmod 2$
15: $(a_2, b_2) \leftarrow (b'', r/2^{j_0})$, $Q \leftarrow [0, 2^{j_0}; 2^{j_0}, q]$, $m \leftarrow j_0$
16: $s_0 \leftarrow s_0 + j_0(a_2^2 - 1)/8 \bmod 2$, $k_2 \leftarrow k - (m + j_1)$
17: $s_2, j_2, S \leftarrow \text{HalfBinaryJacobi}(a_2 \bmod 2^{2k_2+2}, b_2 \bmod 2^{2k_2+2}, k_2)$
18: Return $(s_0 + s_1 + s_2) \bmod 2$, $j_1 + j_2 + m$, $S \times Q \times R$

Implementation and Timings

Experimental Results: Small Numbers

For $a, b < 2^{26}$, the maximum number of iterations of the cubic algorithm is 64, with $a = 15548029$ and $b = 66067306$.
Heuristic: 50% good iterations, 25% bad, 25% ugly.

Conjecture. The number of iterations of CubicBinaryJacobi for n-bit inputs is $O(n)$.

For $a, b < 2^{20}$, the maximum number of iterations of the quadratic algorithm is 37 for $a = 933531, b = 869894$.
Heuristic: $8/15$ good, $4/15$ bad, $3/15$ harmless.

The binary division with positive quotient has an average increase of 0.65 bit per iteration (0.05 for the centered quotient), thus an average reduction of 1.35 bit per iteration (1.95).

Experimental Results: Large Numbers

Timings on a 2.83Ghz Core 2 with GMP 4.3.1, with inputs of one million 64-bit words.

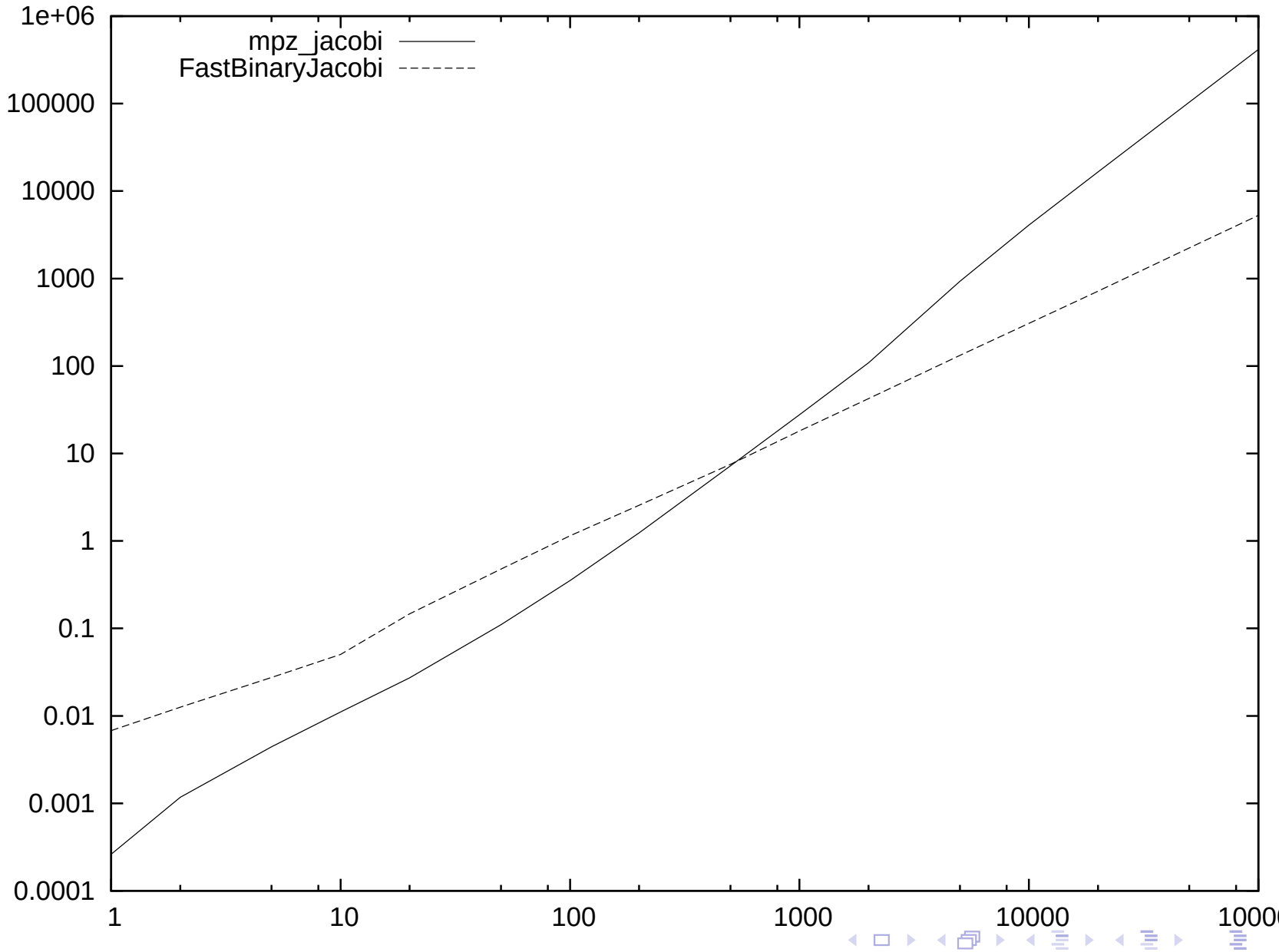
GMP's fast gcd takes 45.8s.

An implementation of the (fast) binary gcd takes 48.3s and 32,800,000 iterations.

Our implementation FastBinaryJacobi takes 83.1s and 47,500,000 iterations.

$$\frac{1.95}{1.35} = 1.4444 \approx \frac{47,500,000}{32,800,000} = 1.4482$$

Our implementation is faster than GMP's $O(n^2)$ code up from 535 words (about 10,000 decimal digits).


mpz_jacobi
FastBinaryJacobi
1e+06
100000
10000
1000
100
10
1
0.1
0.01
0.001
0.0001
1
10
100
1000
10000
100000

Concluding Remarks

- first complete (description + code) subquadratic Jacobi algorithm
- first LSB algorithm for that problem
- does not need to compute the (MSB) quotient sequence
- we can use the centered quotient with the "cubic" algorithm. Moreover we can choose $q \pm 2^{j+1} \in [-2^{j+1}, 2^{j+1}]$ such that $bq/2^j$ has sign opposite to a. We then gain on average 2.19 bits per iteration, against 1.95 for the centered quotient, 1.35 for the positive quotient, and 1.42 for Stein's "binary gcd".

Preprint and GMP code available from:
`http://www.loria.fr/~zimmerma/papers/#jacobi`

Thanks to:

- Steven Galbraith for asking the original question;
- Damien Stehlé for suggesting using the LSB algorithm;
- Arnold Schönhage for his comments and pointers to earlier work.