A Binary Recursive Gcd Algorithm

Damien Stehlé, Paul Zimmermann

▶ **To cite this version:**

Damien Stehlé, Paul Zimmermann. A Binary Recursive Gcd Algorithm. [Research Report] RR-5050, INRIA. 2002. inria-00071533

HAL Id: inria-00071533
https://inria.hal.science/inria-00071533

Submitted on 23 May 2006

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

INRIA

INSTITUT NATIONAL DE RECHERCHE EN INFORMATIQUE ET EN AUTOMATIQUE

A Binary Recursive Gcd Algorithm

Damien Stehlé and Paul Zimmermann

N° 5050

Décembre 2003

THÈME 2

ISSN 0249-6399 ISRN INRIA/RR--5050--FR+ENG

INRIA
LORRAINE

A Binary Recursive Gcd Algorithm

Damien Stehlé and Paul Zimmermann

Thème 2 — Génie logiciel
et calcul symbolique
Projet Spaces

Rapport de recherche n° 5050 — Décembre 2003 — 20 pages

Abstract: The binary algorithm is a variant of the Euclidean algorithm that performs well in practice. We present a quasi-linear time recursive algorithm that computes the greatest common divisor of two integers by simulating a slightly modified version of the binary algorithm. The structure of the recursive algorithm is very close to the one of the well-known Knuth-Schönhage fast gcd algorithm, but the description and the proof of correctness are significantly simpler in our case. This leads to a simplification of the implementation and to better running times.

Key-words: Gcd computation, Euclidean algorithm, Knuth-Schönhage algorithm, quasi-linear time complexity.

Unité de recherche INRIA Lorraine
LORIA, Technopôle de Nancy-Brabois, Campus scientifique,
615, rue du Jardin Botanique, BP 101, 54602 Villers-Lès-Nancy (France)
Téléphone : +33 3 83 59 30 00 — Télécopie : +33 3 83 27 83 19

Un Algorithme Récursif de Pgcd Binaire

Résumé : L'algorithme binaire est une variante de l'algorithme d'Euclide particulièrement efficace en pratique. Nous décrivons un algorithme récursif de complexité quasi-linéaire qui calcule le plus grand commun diviseur de deux entiers en simulant une légère modification de l'algorithme binaire. La structure de cet algorithme est très proche de celle du célèbre algorithme de calcul rapide de pgcd de Knuth et Schönhage, mais sa description ainsi que sa preuve sont nettement plus simples. Cela a pour conséquence une simplification de l'implantation et de meilleurs temps de calcul.

Mots-clés : Calcul de pgcd, algorithme d'Euclide, algorithme de Knuth-Schönhage, complexité quasi-linéaire.

1 Introduction

Gcd computation is a central task in computer algebra, in particular when computing over rational numbers or over modular integers. The well-known Euclidean algorithm solves this problem in time quadratic in the size of the inputs. Several variants of this algorithm have been presented in the last decades. For example, by taking centered quotients instead of positive quotients, one already gets a significant improvement (see [2] for a detailed analysis of Euclidean algorithms with diverse divisions). In 1970, Knuth [5] presented the first quasi-linear time gcd algorithm, based on fast multiplication [7]. The complexity of this algorithm was improved by Schönhage [6] to $O(n \log^2 n \log\log n)$. A comprehensive description of the Knuth-Schönhage algorithm can be found in [11]. The correctness of this algorithm is quite hard to establish, essentially because of the technical details around the so-called "fix-up procedure"[1]. This "fix-up procedure" is rather difficult to implement and makes the algorithm uninteresting unless for very large numbers (of length significantly higher than 10^5 bits).

In this paper, we present a variant of the Knuth-Schönhage algorithm that does not have the "fix-up procedure" drawback. To achieve this, we describe a new division (GB for Generalized Binary), which can be seen as a natural generalization of the binary division. It does not seem possible to use the binary division itself in a Knuth-Schönhage-like algorithm, because its definition is asymmetric: it eliminates least significant bits but it also considers most significant bits to perform comparisons. There is no such asymmetry in the GB division. The recursive GB Euclidean algorithm is much simpler to describe and to prove than the Knuth-Schönhage algorithm, and it has the same asymptotic complexity.

This simplification of the description turns out to be an important advantage in practice: we implemented the algorithm in GNU MP, and it revealed between three and four times faster than the implementations of the Knuth-Schönhage algorithm in Magma and Mathematica. In GNU MP, the algorithm used for gcd computations of large integers is the one of Sorenson [8], which is not quasi-linear. The cutoff with Sorenson's algorithm is around 2000 machine

[1] As an example, several mistakes can be noticed in the proof of [11], as it can be observed at the webpage http://www.cs.nyu.edu/cs/faculty/yap/book/errata.html

words, i.e. 64000 bits. As far as we know, our code is the fastest gcd routine for large integers.

The paper is organized as follows. In Section 2 we introduce the GB division and give some basic results about it. In Section 3, we describe precisely the new recursive algorithm. We prove its correctness in Section 4 and analyze its complexity in Section 5. Some implementation issues are discussed in Section 6.

Notations: The notations $O(.)$, $o(.)$ and $\Theta(.)$ are standard. The complexity is measured in elementary operations on bits. Unless specified explicitly, all the logarithms are taken in base 2. If a is a non-zero integer, $\ell(a)$ denotes the length of the binary representation of a, i.e. $\ell(a) = \lfloor \log |a| \rfloor + 1$. $\nu(a)$ denotes the 2-adic valuation of a, i.e. the number of consecutive zeroes in the least significant bits of the binary representation of a; by definition, $\nu(0) = \infty$. $r := a \bmod^c b$ denotes the centered remainder of a modulo b, i.e. r satisfies $a = r \bmod b$ and $-\frac{b}{2} \leq r < \frac{b}{2}$. We recall that $M(n) = \Theta(n \log n \log\log n)$ is the asymptotic time required to multiply two n-bit integers with Schönhage-Strassen multiplication (we refer to [10] for a complete description of basic arithmetic operations).

2 The Generalized Binary Division

In this section we first recall the binary algorithm. Then we define the GB division and give some basic properties about it. Subsection 2.3 explains how to compute modular inverses from the output of the Euclidean algorithm based on the GB division. This subsection is independent from the remainder of the paper but is justified by the fact that computing modular inverses is a standard application of the Euclidean algorithm.

2.1 The binary Euclidean algorithm

The binary division relies on the following properties: $\gcd(2a, 2b) = 2\gcd(a, b)$, $\gcd(2a+1, 2b) = \gcd(2a+1, b)$, and $\gcd(2a+1, 2b+1) = \gcd(2b+1, a-b)$. It consists of eliminating the least significant bit at each loop iteration. Here is a description of the binary algorithm:

The behavior of this algorithm is very well understood (we refer to [2] for an average complexity analysis). Although it is still quadratic in the size of

Algorithm Binary-Gcd.
Input: $a, b \in \mathbb{Z}$.
Output: $\gcd(a, b)$.

1. If $|b| > |a|$, return Binary-Gcd(b, a).
2. If $b = 0$, return a.
3. If a and b are both even then return 2.Binary-Gcd$(a/2, b/2)$.
4. If a is even and b is odd then return Binary-Gcd$(a/2, b)$.
5. If a is odd and b is even then return Binary-Gcd$(a, b/2)$.
6. Otherwise return Binary-Gcd$((a - b)/2, b)$.

Fig. 1. The binary Euclidean algorithm

the inputs, there is a significant gain towards the usual Euclidean algorithm, in particular because there is no need to compute any quotient.

2.2 The generalized binary Euclidean algorithm

In the case of the standard Euclidean division of a by b with $|a| > |b|$, one computes a quotient q such that when qb is substracted to a, the most significant bits of a vanish. The GB division is the reverse: in order to GB-divide a by b with $\nu(a) < \nu(b)$, one computes a quotient q such that when $q\frac{b}{2^{\nu(b)-\nu(a)}}$ is added to a, the least significant bits of a vanish.

Lemma 1 (GB Division). *Let a, b be non-zero integers with $\nu(a) < \nu(b)$. Then there exists a unique pair of integers (q, r) such that:*

- (1) $r = a + q\frac{b}{2^{\nu(b)-\nu(a)}}$,
- (2) $|q| < 2^{\nu(b)-\nu(a)}$,
- (3) $\nu(r) > \nu(b)$.

The integers q and r are called respectively the GB quotient and the GB remainder of (a, b). We define $GB(a, b)$ as the pair (q, r).

Proof. q is exactly $-\frac{a}{2^{\nu(a)}}.(\frac{b}{2^{\nu(b)}})^{-1} \bmod 2^{\nu(b)-\nu(a)+1}$. Since q is odd, the second condition is fulfilled and gives the uniqueness of q. As a consequence, $r = a + q\frac{b}{2^{\nu(b)-\nu(a)}}$ is unique. □

We now give two algorithms to compute q and r. The first one is is the equivalent of the naive division algorithm, and is asymptotically slower than the second one, which is the equivalent of the fast division algorithm. For most of the pairs (a, b), the Euclidean algorithm based on the GB division performs almost all its divisions on pairs (c, d) for which $\nu(d)-\nu(c)$ is small. This enlights the fact that the first algorithm suffices in practice.

Algorithm Elementary-GB.
Input: Two non-zero integers a, b satisfying $\nu(a) < \nu(b)$.
Output: $(q, r) = GB(a, b)$.

1. $q := 0$, $r := a$.
2. While $\nu(r) \leq \nu(b)$ do
3. $\quad q := q - 2^{\nu(r)-\nu(a)}$,
4. $\quad r := r - 2^{\nu(r)-\nu(b)}b$.
5. $q := q \bmod 2^{\nu(b)-\nu(a)+1}$, $r := q\frac{b}{2^{\nu(b)-\nu(a)}} + a$.
6. Return (q, r).

Fig. 2. Algorithm **Elementary-GB**

It is clear that the following result holds:

Lemma 2. *The algorithm* **Elementary-GB** *of Figure 2 is correct and if the input (a, b) satisfies $\ell(a), \ell(b) \leq n$, then it finishes in time $O(n(\nu(b) - \nu(a)))$.*

The second algorithm computes $GB(a, b)$ in quasi-linear time, in the case of Schönhage-Strassen multiplication, by using Newton's iteration. It is adapted from [4] and we refer to [3] for a proof of the following lemma.

Lemma 3. *The algorithm* **Fast-GB** *of Figure 3 is correct and in the case of Schönhage-Strassen multiplication, if a and b satisfy the conditions $\ell(a), \ell(b) \leq 2n$ and $\nu(b) - \nu(a) \leq n$, then it finishes in time $\frac{7}{2}M(n) + o(M(n))$.*

A GB Euclidean algorithm can be derived very naturally from the definition of the GB division, see Figure 4.

Algorithm Fast-GB.
Input: Two non-zero integers a, b satisfying $\nu(a) < \nu(b)$.
Output: $(q, r) = GB(a, b)$.

1. $A := -\frac{a}{2^{\nu(a)}}$, $B := \frac{b}{2^{\nu(b)}}$, $n := \nu(b) - \nu(a) + 1$.
2. $q := 1$.
3. For i from 1 to $\lceil \log n \rceil - 1$ do
4. $\quad q := q + q(1 - Bq) \bmod 2^{2^i}$.
5. $q' := Aq \bmod 2^n$.
6. $q' := q' + q(A - Bq') \operatorname{cmod} 2^n$.
7. $r := a + \frac{q'}{2^{n-1}} b$.
8. Return (q', r).

Fig. 3. Algorithm **Fast-GB**

Algorithm GB-gcd.
Input: Two non-zero integers a, b satisfying $\nu(a) < \nu(b)$.
Output: $\frac{g}{2^{\nu(g)}}$, where g is the greatest common divisor of a and b.

1. If $b = 0$, return $\frac{a}{2^{\nu(a)}}$.
2. $(q, r) := GB(a, b)$.
3. Return GB-gcd (b, r).

Fig. 4. The GB Euclidean algorithm

Lemma 4. *The GB Euclidean algorithm of Figure 4 is correct, and if we use the algorithm* **Elementary-GB** *of Figure 2, then for any input (a, b) satisfying $\ell(a), \ell(b) \leq n$, it finishes in time $O(n^2)$.*

Proof. Let $r_0 = a, r_1 = b, r_2, \ldots$ be the sequence of remainders that appear in the execution of the algorithm. We first show that this sequence is finite and thus that the algorithm terminates.

We have for any $k \geq 0$, $|r_{k+2}| \leq |r_{k+1}| + |r_k|$, so that $|r_k| \leq 2^{n+1} \left(\frac{1+\sqrt{5}}{2}\right)^k$. Moreover, 2^k divides $|r_k|$, which gives the inequalities $2^k \leq |r_k| \leq 2^{n+1} \left(\frac{1+\sqrt{5}}{2}\right)^k$ and $(1 - \log \frac{1+\sqrt{5}}{2})k \leq n$. Therefore there are $O(n)$ remainders in the remainder sequence. Let $t = O(n)$ be the length of the remainder sequence. Suppose that r_t is the last non-zero remainder. From Lemma 2, we know that each of the calls to a GB-division involves a number of bit operations bounded by $O(\log |r_k|(\nu(r_{k+1}) - \nu(r_k))) = O(n(\nu(r_{k+1}) - \nu(r_k)))$, so that the overall complexity is bounded by $O(n\nu(r_t))$.

The correctness comes from the fact that if $g = \gcd(a, b)$ and $(q, r) = GB(a, b)$, then $\gcd(b, r)/g$ is a power of 2. □

A better bound on $|r_k|$ and thus on t is proved in Section 5.1. Nonetheless the present bound is sufficient to guarantee the quasi-linear time asymptotic complexity of the recursive algorithm. The improved bound only decreases the multiplying constant of the asymptotic complexity.

For $n \geq 1$ and $q \in [-2^n + 1, 2^n - 1]$ we define the matrix $[q]_n = \begin{pmatrix} 0 & 2^n \\ 2^n & q \end{pmatrix}$. Let r_0, r_1 be two non-zero integers with $0 = \nu(r_0) < \nu(r_1)$, and $r_0, r_1, r_2, \ldots$ be their GB remainder sequence, and $q_1, q_2, \ldots$ be the corresponding quotients: $r_{i+1} = r_{i-1} + q_i \frac{r_i}{2^{\nu(r_i) - \nu(r_{i-1})}}$ for any $i \geq 1$. Then the following relation holds for any $i \geq 1$:

$$\begin{pmatrix} r_i \\ r_{i+1} \end{pmatrix} = \frac{1}{2^{\nu(r_i)}} [q_i]_{n_i} \ldots [q_1]_{n_1} \begin{pmatrix} r_0 \\ r_1 \end{pmatrix},$$

where $n_j = \nu(r_j) - \nu(r_{j-1}) \geq 1$ for any $j \geq 1$.

In what follows, we use implicitly the following simple fact several times.

Lemma 5. *Let r_0, r_1 be two non-zero integers with $0 = \nu(r_0) < \nu(r_1)$, and r_0, r_1, r_2, ... be their GB remainder sequence. Let $d \geq 0$. Then there exists a unique $i \geq 0$ such that $\nu(r_i) \leq d < \nu(r_{i+1})$.*

2.3 Computing modular inverses

Let a, b be two non-zero integers with $0 = \nu(a) < \nu(b)$. Suppose that we want to compute the inverse of b modulo a, by using an extended version of the Euclidean algorithm based on the GB division. The execution of the GB Euclidean algorithm gives two integers A and B such that $Aa + Bb = 2^\alpha g$, where $\alpha = O(n)$ and $g = \gcd(a, b)$. From such a relation, it is easy to check that $g = 1$. Suppose now that the inverse B' of b modulo a does exist. From the relation $Aa + Bb = 2^\alpha$, we know that:

$$B' = \frac{B}{2^\alpha} \bmod a.$$

Therefore, in order to obtain B', it is sufficient to compute the inverse of 2^α modulo a. By using Newton's iteration (like in the algorithm **Fast-GB** of Figure 3), we obtain the inverse of a modulo 2^α. This gives x and y that satisfy: $xa + y2^\alpha = 1$. Clearly y is the inverse of 2^α modulo a.

Since multiplication, Newton's iteration and division on numbers of size $O(n)$ can be performed in time $O(M(n))$, given two n-bit integers a and b, the additional cost to compute the inverse of b modulo a given the output of an extended Euclidean algorithm based on the GB division is $O(M(n))$.

3 The Recursive Algorithm

We now describe the recursive algorithm based on the GB division. This description closely resembles the one of [11]. It uses two routines: the algorithm **Half-GB-gcd** and the algorithm **Fast-GB-gcd**. Given two non-zero integers r_0 and r_1 with $0 = \nu(r_0) < \nu(r_1)$, the algorithm **Half-GB-gcd** outputs the GB remainders r_i and r_{i+1} of the GB remainder sequence of (r_0, r_1) that satisfy $\nu(r_i) \leq \ell(r_0)/2 < \nu(r_{i+1})$. It also outputs the corresponding matrix $2^{-(n_1+\ldots+n_i)}[q_i]_{n_i} \ldots [q_1]_{n_1}$. Then we describe the algorithm **Fast-GB-gcd**, which, given two integers a and b, outputs the gcd of a and b by making successive calls to the algorithm **Half-GB-gcd**.

3.1 Half-GB-gcd

The algorithm **Half-GB-gcd** works as follows: a quarter of the least significant bits of a and b is eliminated by doing a recursive call on the low $\ell(a)/2$ of the bits of a and b. Then the two corresponding remainders (a', b') of the GB remainder sequence of (a, b) are computed thanks to the returned matrix R_1. A single step of the GB Euclidean algorithm is performed on (a', b'), which gives a new remainder pair (b', r). Then there is a second recursive call on approximately $\ell(a)/2$ of the least significant bits of (b', r). The size of the inputs of this second recursive call is similar to the one of the first recursive call. Finally, the corresponding remainders (c, d) of the GB remainder sequence of (a, b) are computed thanks to the returned matrix R_2, and the output matrix R is calculated from R_1, R_2 and the GB quotient of (a', b'). Figure 5 illustrates the execution of this algorithm.

Note that in the description of the algorithm **Half-GB-gcd** in Figure 6, a routine GB' is used. This is a simple modification of the division GB: given a and b as input with $0 = \nu(a) < \nu(b)$, it outputs their GB quotient q, and $\frac{r}{2^{\nu(b)}}$ if r is their GB remainder.

3.2 Fast-GB-gcd

The algorithm **Fast-GB-gcd** uses several times the algorithm **Half-GB-gcd** in order to decrease the lengths of the remainders quickly.

The main advantage towards the other quasi-linear time algorithms is that if a matrix R is returned by a recursive call of the algorithm **Half-GB-gcd**, then it contains only "correct quotients". There is no need going back in the GB remainder sequence to make the quotients correct, and thus no need to store the sequence of quotients. The underlying reason is that the remainders are shortened by the least significant bits, and since the carries go in the direction of the most significant bits, these two phenomena do not interfere. For that reason, the algorithm is as simple as the Knuth-Schönhage algorithm in the case of polynomials.

4 Correctness of the Recursive Algorithm

In that section, we show that the algorithm **Fast-GB-gcd** of Figure 7 is correct. We first give some results about the GB division, and then we show the

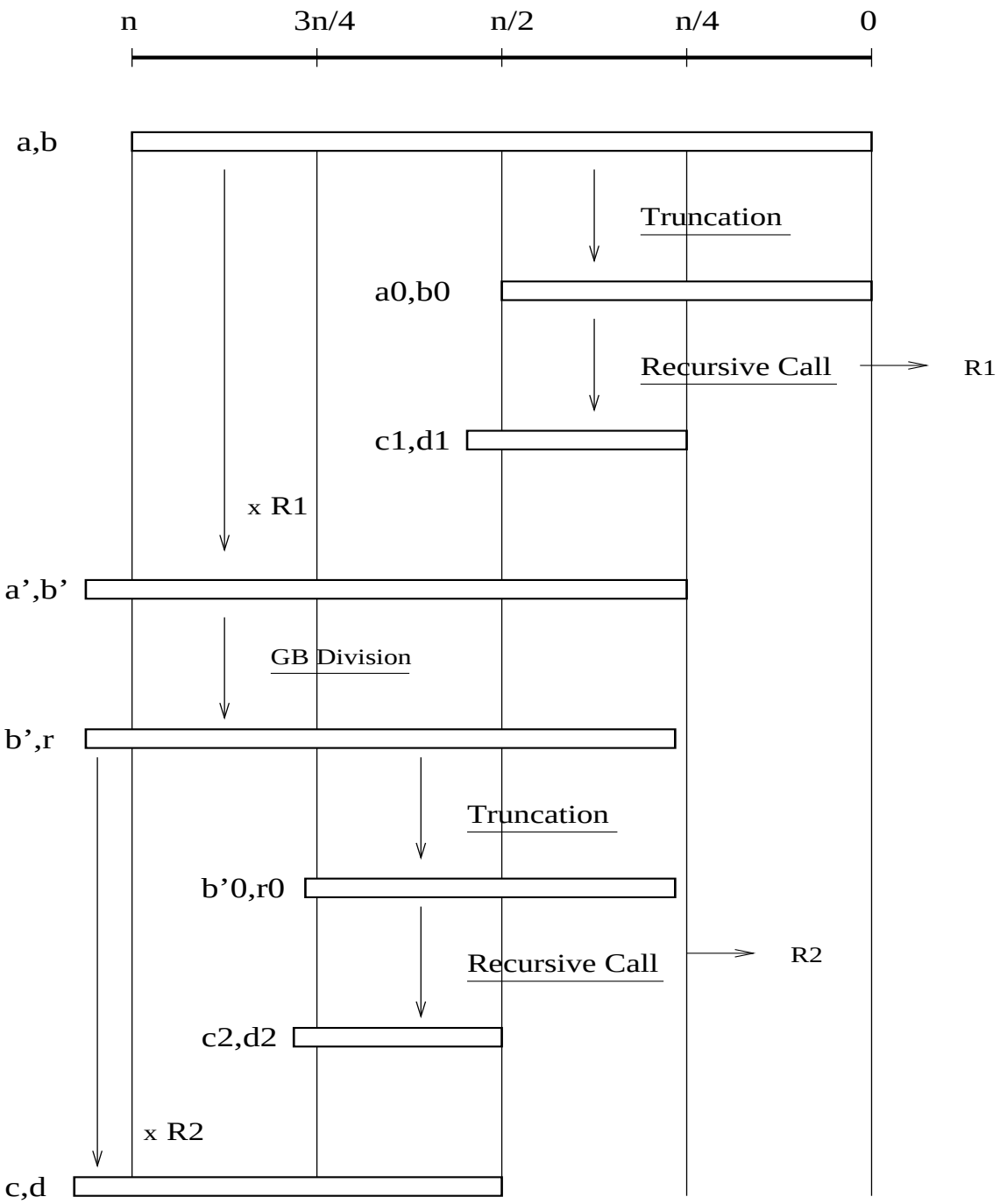

Fig. 5. The algorithm **Half-GB-gcd**

Algorithm Half-GB-gcd.
Input: a, b satisfying $0 = \nu(a) < \nu(b)$.
Output: An integer j, an integer matrix R and two integers c and d such that $\begin{pmatrix} c \\ d \end{pmatrix} = 2^{-2j} R \begin{pmatrix} a \\ b \end{pmatrix}$, and $c^* = 2^j c$, $d^* = 2^j d$ are the two consecutive remainders of the GB remainder sequence of (a, b) that satisfy
$$\nu(c^*) \leq \ell(a)/2 < \nu(d^*) \text{ and } 0 = \nu(c) < \nu(d).$$

1. $k := \lfloor \ell(a)/2 \rfloor$.
2. If $\nu(b) > k$, then return $0, \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}, a, b$.
3. $k_1 := \lfloor k/2 \rfloor$.
4. $a := a_1 2^{2k_1+1} + a_0$, $b := b_1 2^{2k_1+1} + b_0$ with $0 \leq a_0, b_0 < 2^{2k_1+1}$.
5. $j_1, R_1, c_1, d_1 :=$ **Half-GB-gcd**(a_0, b_0).
6. $\begin{pmatrix} a' \\ b' \end{pmatrix} := 2^{2k_1+1-2j_1} R_1 \begin{pmatrix} a_1 \\ b_1 \end{pmatrix} + \begin{pmatrix} c_1 \\ d_1 \end{pmatrix}$.
7. If $\nu(b') + j_1 > k$, then return j_1, R_1, a', b'.
8. $(q, r) := GB'(a', b')$, $j_0 := \nu(b')$.
9. $k_2 := k - (\nu(b') + j_1)$.
10. $\frac{b'}{2^{\nu(b')}} := b'_1 2^{2k_2+1} + b'_0$, $r := r_1 2^{2k_2+1} + r_0$ with $0 \leq b'_0, r_0 < 2^{2k_2+1}$.
11. $j_2, R_2, c_2, d_2 :=$ **Half-GB-gcd**(b'_0, r_0).
12. $\begin{pmatrix} c \\ d \end{pmatrix} := 2^{2k_2+1-2j_2} R_2 \begin{pmatrix} b'_1 \\ r_1 \end{pmatrix} + \begin{pmatrix} c_2 \\ d_2 \end{pmatrix}$.
13. Return $j_1 + j_0 + j_2$, $R_2[q]_{j_0} R_1$, c, d.

Fig. 6. Algorithm **Half-GB-gcd**

Algorithm Fast-GB-gcd.
Input: a, b satisfying $0 = \nu(a) < \nu(b)$.
Output: $g = \gcd(a, b)$.

1. If $b = 0$, return a.
2. $j, R, a', b' :=$ **Half-GB-gcd**(a, b).
3. Return $g :=$ **Fast-GB-gcd**(a', b').

Fig. 7. Algorithm **Fast-GB-gcd**

correctness of the algorithm **Half-GB-gcd** which clearly implies the correctness of the algorithm **Fast-GB-gcd**.

4.1 Some properties of the GB division

The properties described below are very similar to the ones of the standard Euclidean division that make the Knuth-Schönhage algorithm possible.

Lemma 6. *Let a, b, a' and b' be such that $a' = a \bmod 2^l$ and $b' = b \bmod 2^l$ with $l \geq 2\nu(b) + 1$. Assume that $0 = \nu(a) < \nu(b)$. Let $(q, r) = GB(a, b)$ and $(q', r') = GB(a', b')$. Then $q = q'$ and $r = r' \bmod 2^{l-\nu(b)}$.*

Proof. By definition, $q = -a\left(\frac{b}{2^{\nu(b)}}\right)^{-1} \bmod 2^{\nu(b)+1}$. Therefore, since $l \geq 2\nu(b)+1$, $\frac{b}{2^{\nu(b)}} = \frac{b'}{2^{\nu(b')}} \bmod 2^{\nu(b)+1}$, $a = a' \bmod 2^{\nu(b)+1}$ and $q = q'$. Moreover, $r = a + q\frac{b}{2^{\nu(b)}}$ and $r' = a' + q\frac{b'}{2^{\nu(b')}}$. Consequently $r = r' \bmod 2^{l-\nu(b)}$. □

From the previous result, we obtain that:

Lemma 7. *Let a, b, a' and b' such that $a' = a \bmod 2^{2k+1}$ and $b' = b \bmod 2^{2k+1}$, with $k \geq 0$. Suppose that $0 = \nu(a) < \nu(b)$. Let $r_0 = a$, $r_1 = b$, r_2, ... be the GB remainder sequence of (a, b), and let $q_1, q_2, \ldots$ be the corresponding GB quotients: $r_{j+1} = r_{j-1} + q_j \frac{r_j}{2^{n_j}}$, with $n_j = \nu(r_j) - \nu(r_{j-1})$. Let $r'_0 = a'$, $r'_1 = b'$, r'_2, ... be the GB remainder sequence of (a', b'), and let $q'_1, q'_2, \ldots$ be the corresponding GB quotients: $r'_{j+1} = r'_{j-1} + q'_j \frac{r'_j}{2^{n'_j}}$, with $n'_j = \nu(r'_j) - \nu(r'_{j-1})$. Then if r_{i+1} is the first remainder such that $\nu(r_{i+1}) > k$, then for any $j \leq i$, we have $q_j = q'_j$ and $r_{j+1} = r'_{j+1} \bmod 2^{2k+1-\nu(r_j)}$.*

Proof. We prove this result by induction on $j \geq 1$. This is true for $j = 1$, since $a' = a \bmod 2^{2k+1}$ and $b' = b \bmod 2^{2k+1}$. Suppose now that $2 \leq j \leq i$. We use Lemma 6 with $\frac{r_{j-1}}{2^{\nu(r_{j-1})}}, \frac{r_j}{2^{\nu(r_{j-1})}}, \frac{r'_{j-1}}{2^{\nu(r_{j-1})}}, \frac{r'_j}{2^{\nu(r_{j-1})}}$ and $l = 2k + 1 - 2\nu(r_{j-1})$. By induction, modulo $2^{2k+1-\nu(r_{j-1})}$, $r_{j-1} = r'_{j-1}$ and $r_j = r'_j$. Since $j \leq i$, we have by definition of i, $2k+1-2\nu(r_{j-1}) \geq 2(\nu(r_j) - \nu(r_{j-1})) + 1$, and consequently we can apply Lemma 6.

Thus $q_j = q'_j$ and $r_{j+1} = r'_{j+1} \bmod 2^{2k+1-\nu(r_j)}$. □

4.2 Correctness of the Half-GB-gcd algorithm

To show the correctness of the algorithm **Fast-GB-gcd**, it suffices to show the correctness of the algorithm **Half-GB-gcd**, which is established in the following theorem.

Theorem 8. *The algorithm* **Half-GB-gcd** *of Figure 6 is correct.*

Proof. We prove the correctness of the algorithm by induction on the size of the inputs. If $\ell(a) = 1$, then the algorithm finishes at Step 2 because $\nu(b) \geq 1$. Suppose now that $k \geq 2$ and that $\nu(b) \leq k$.

Since $2\lfloor \frac{k}{2} \rfloor + 1 < \ell(a)$, Step 5 is a recursive call (and the inputs satisfy the input conditions). By induction j_1, R_1, c_1 and d_1 satisfy $\begin{pmatrix} c_1 \\ d_1 \end{pmatrix} = 2^{-2j_1} R_1 \begin{pmatrix} a_0 \\ b_0 \end{pmatrix}$, and $2^{j_1} c_1$ and $2^{j_1} d_1$ are the consecutive remainders r'_{i_1} and r'_{i_1+1} of the GB remainder sequence of $r'_0 = a_0$ and $r'_1 = b_0$ that satisfy $\nu(r'_{i_1}) \leq k_1 < \nu(r'_{i_1+1})$. From Lemma 7, we know that $2^{-j_1} R_1 \begin{pmatrix} a \\ b \end{pmatrix}$ are two consecutive remainders $2^{j_1} a' = r_{i_1}$ and $2^{j_1} b' = r_{i_1+1}$ of the GB remainder sequence of $r_0 = a$ and $r_1 = b$, and they satisfy $r_{i_1} = r'_{i_1}$ and $r_{i_1+1} = r'_{i_1+1}$ modulo 2^{k_1+1}. From these last equalities, we have that $\nu(r_{i_1}) = \nu(r'_{i_1}) \leq k_1 < \nu(r_{i_1+1}) \leq \nu(r'_{i_1+1})$. Therefore, if the execution of the algorithm stops at Step 7, then the output is correct.

Otherwise, r_{i_1+2} is computed at Step 8. At Step 9 we compute $k_2 = k - \nu(r_{i_1+1})$. Step 7 ensures that $k_2 \geq 0$. Since $\nu(r_{i_1+1}) > \lfloor k/2 \rfloor$, we have $k_2 \leq \lceil k/2 \rceil - 1$. Therefore Step 11 is a recursive call (and the inputs satisfy the input conditions). By induction, j_2, S_2, c_2 and d_2 satisfy: $\begin{pmatrix} c_2 \\ d_2 \end{pmatrix} = 2^{-2j_2} R_2 \begin{pmatrix} b'_0 \\ r_0 \end{pmatrix}$, and $2^{j_2} c_2$ and $2^{j_2} d_2$ are the consecutive remainders r'_{i_2} and r'_{i_2+1} of the GB remainder sequence of (b_0, r'_0). Moreover, $\nu(r'_{i_2}) \leq k_2 < \nu(r'_{i_2+1})$. From Lemma 7, we know that $2^{-j_2} S_2 \begin{pmatrix} 2^{j_1} b' \\ 2^{j_1} r' \end{pmatrix}$ are two consecutive remainders r_i and r_{i+1} (with $i = i_1 + i_2 + 1$) of the GB remainder sequence of (a, b), that $\frac{r_i}{2^{j_1+\nu(b')}} = 2^{j_2} c_2 \bmod 2^{k_2+2}$ and that $\frac{r_{i+1}}{2^{j_1+\nu(b')}} = 2^{j_2} d_2 \bmod 2^{k_2+1}$. Therefore the following sequence of inequalities is valid: $\nu(r_i) = j_1 + j_2 + \nu(b') \leq k < \nu(r_{i+1})$. This ends the proof of the theorem. □

5 Analyses of the Algorithms

In that section, we first study the GB Euclidean algorithm. In particular we give a worst-case bound regarding the length of the GB remainder sequence. Then we bound the complexity of the recursive algorithm in the case of the use of Schönhage-Strassen multiplication, and we give some intuition about the average complexity.

5.1 Analysis of the length of the GB remainder sequence

In this subsection, we first bound the size of the matrix $\frac{1}{2^{\nu(r_i)}}[q_i]_{n_i} \dots [q_1]_{n_1}$, where the q_j's are the GB quotients of a truncated GB remainder sequence $r_0, \dots, r_{i+1}$ with $\nu(r_i) \leq \nu(r_0) + d < \nu(r_{i+1})$ for some $d \geq 0$. This will make possible the analysis of the lengths and the number of the GB remainders. As mentioned in Section 2, this subsection is not necessary to prove the quasi-linear time complexity of the algorithm.

Theorem 9. *Let $d \geq 1$. Let r_0, r_1 with $0 = \nu(r_0) < \nu(r_1)$, and $r_0, r_1, r_2, \dots, r_{i+1}$ their first GB remainders, where i is such that $\nu(r_i) \leq d < \nu(r_{i+1})$. We consider the matrix $\frac{1}{2^{\nu(r_i)}}[q_i]_{n_i} \dots [q_1]_{n_1}$, where the q_j's are the GB quotients and $n_j = \nu(r_j) - \nu(r_{j-1})$ for any $1 \leq j \leq i$. Let M be the maximum of the absolute values of the four entries of this matrix. Then we have:*

- *If $d = 0$ or 1, $M = 1$,*
- *If $d = 3$, $M \leq 11/8$,*
- *If $d = 5$, $M \leq 67/32$,*
- *If $d = 2, 4$ or $d \geq 6$, $M \leq \frac{2}{\sqrt{17}} \left(\left(\frac{1+\sqrt{17}}{4} \right)^{d+1} - \left(\frac{1-\sqrt{17}}{4} \right)^{d+1} \right)$.*

Moreover, all these bounds are reached, in particular, the last one is reached when $n_j = 1$ and $q_j = 1$ for any $1 \leq j \leq i$.

The proof resembles the worst case analysis of the Gaussian algorithm in [9].

Proof. We introduce a partial order on the 2×2 matrices: $A < B$ if and only if for any coordinate $[i, j]$, $A[i, j] \leq B[i, j]$. First, the proof can be restricted to the case where the q_j's are non-negative integers, because we have the inequality $|[q_i]_{n_i} \dots [q_1]_{n_1}| \leq [|q_i|]_{n_i} \dots [|q_1|]_{n_1}$. This can be easily showed by induction on

i by using the properties: $|A.B| \leq |A|.|B|$, and if the entries of A, A', B, B' are non-negative, $A \leq A'$ and $B \leq B'$ implies $A.B \leq A'.B'$.

Consequently we are looking for the maximal elements for the partial order $>$ in the set: $\{\Pi_{n_1+\ldots+n_i \leq d}\ [q_i]_{n_i} \ldots [q_1]_{n_1}\ /\ \forall 1 \leq j \leq i,\ 0 < q_j \leq 2^{n_j} - 1 \text{ and } n_j \geq 1\}$.
We can restrict the analysis to the case where $n_1 + \ldots + n_i = d$ and all the q_j's are maximal, which gives the set: $\{\Pi_{n_1+\ldots+n_i=d}\ [2^{n_i} - 1]_{n_i} \ldots [2^{n_1} - 1]_{n_1}\}$.

Remark now that $[2^n - 1]_n \leq [1]_1^n$ for any $n \geq 3$. Therefore, it is sufficient to consider the case where the n_j's are in $\{1, 2\}$. Moreover, for any integer $j \geq 0$, $[3]_2[1]_1^j[3]_2 \leq [1]_1^{j+4}$, and we also have the inequalities $[3]_2^2 \leq [1]_1^4$, $[1]_1^5.[3]_2 \leq [1]^7$, $[3]_2.[1]_1^5 \leq [1]^7$ and $[1]_1^2.[3]_2.[1]_1^2 \leq [1]_1^6$.

From these relations, we easily obtain the maximal elements:

- For $d = 1$, $[1]_1$.
- For $d = 2$, $[1]_1^2$ and $[3]_2$.
- For $d = 3$, $[1]_1^3$, $[3]_2.[1]_1$ and $[1]_1.[3]_2$.
- For $d = 4$, $[1]_1^4$, $[3]_2.[1]_1^2$, $[1]_1.[3]_2.[1]_1$ and $[1]_1^2.[3]_2$.
- For $d = 5$, $[1]_1^5$, $[3]_2.[1]_1^3$, $[1]_1^2.[3]_2.[1]_1$, $[1]_1.[3]_2.[1]_1^2$ and $[1]_1^3.[3]_2$.
- For $d = 6$, $[1]_1^6$, $[3]_2.[1]_1^4$, $[1]_1^3.[3]_2.[1]_1$, $[1]_1.[3]_2.[1]_1^3$ and $[1]_1^4.[3]_2$.
- For $d = 7$, $[1]_1^7$, $[1]_1.[3]_2.[1]_1^4$ and $[1]_1^4.[3]_2.[1]_1$.
- For $d \geq 8$, $[1]_1^d$.

The end of the proof is obvious once we note that $2^{-d}[1]_1^d = \begin{pmatrix} u_{d-1} & u_d \\ u_d & u_{d+1} \end{pmatrix}$, with $u_i = u_{i-2} + \frac{1}{2}u_{i-1}$, $u_0 = 0$ and $u_1 = 1$. □

From this result on the quotient matrices, we can easily deduce the following on the size of the remainders and on the length of the remainder sequence.

Theorem 10. *Let r_0, r_1 be two non-zero integers with $0 = \nu(r_0) < \nu(r_1)$, and $r_0, r_1, r_2, \ldots, r_{d+1}$ their complete GB remainder sequence, i.e. with $r_{d+1} = 0$. Assume that $8 \leq j \leq d$. Then:*

$$2^j \leq |r_j| \leq \frac{2}{\sqrt{17}} \left[|r_0| \left(\left(\frac{1+\sqrt{17}}{4} \right)^{j-1} - \left(\frac{1-\sqrt{17}}{4} \right)^{j-1} \right) + |r_1| \left(\left(\frac{1+\sqrt{17}}{4} \right)^{j} - \left(\frac{1-\sqrt{17}}{4} \right)^{j} \right) \right].$$

These lower and upper bounds are reached together for

$$\begin{pmatrix} r_0 \\ r_1 \end{pmatrix} = [1]_1^{-d} \begin{pmatrix} 2^d \\ 0 \end{pmatrix}.$$

Moreover, if $\ell(r_0), \ell(r_1) \leq n$, *then we have:* $d \leq n/\log(\frac{\sqrt{17}-1}{2})$.

As a comparison, we recall that the worst case for the standard Euclidean division corresponds to the Fibonacci sequence, with $d = n/\log(\frac{1+\sqrt{5}}{2}) + o(n)$. Note that $1/\log(\frac{1+\sqrt{5}}{2}) \approx 1.440$ and $1/\log(\frac{\sqrt{17}-1}{2}) \approx 1.555$.

5.2 Complexity bound for the recursive algorithm

In what follows, $H(n)$ and $G(n)$ respectively denote the maximum of the number of bit operations done by the algorithms **Half-GB-gcd** and **Fast-GB-gcd**, given as inputs two integers of length at most n.

Lemma 11. *The two following relations hold:*

- $G(n) = H(n) + G(cn) + O(n)$,
- $H(n) = 2H(\lfloor \frac{n}{2} \rfloor + 1) + O(M(n))$,

where $c = \left(1 + \log(\frac{1+\sqrt{17}}{4})\right)/2$.

Proof. The first relation is an obvious consequence of Theorem 9. We now prove the second relation. The costs of Steps 1, 2, 3, 4, 7, 9 and 10 are negligible. Steps 5 and 11 are recursive calls and the cost of each one is bounded by $H(\lfloor \frac{n}{2} \rfloor + 1)$. Steps 6, 12 and 13 consist of multiplications of integers of size $O(n)$. Finally, Step 8 is a single GB division, and we proved in Lemma 3 that it can be performed in time $O(M(n))$. □

From the previous result and the fact that $c < 1$, one easily obtains the following theorem:

Theorem 12. *The algorithm* **Fast-GB-gcd** *of Figure 7 runs in quasi-linear time. More precisely* $G(n) = O(M(n) \log n)$.

The constants that one can derive from the previous proofs are rather large, and not very significant in practice. In fact, for randomly chosen n-bit integers, the quotients of the GB remainder sequence are $O(1)$, and therefore Step 8 of the algorithm **Half-GB-gcd** has a negligible cost. Moreover, the worst-case analysis on the size of the coefficients of the returned matrices gives a worst-case bound $O\left(\left(\frac{1+\sqrt{17}}{2}\right)^n\right)$, which happens to be $O(2^n)$ in practice. With these two heuristics, the "practical" cost of algorithm **Half-GB-gcd** satisfies, with Strassen's multiplication of 2×2 matrices:

$$H(n) = 2H(\frac{n}{2}) + 19M(\frac{n}{4}).$$

This gives $H(n) \approx \frac{19}{8}M(n)\log n$ and $G(n) \approx \frac{19}{4}M(n)\log n$. A similar heuristic analysis gives the same estimates for the Knuth-Schönhage algorithm.

6 Implementation issues

We have implemented the algorithms described in this paper in GNU MP [1]. In this section, we first give some "tricks" that we implemented to improve the efficiency of the algorithm, and then we give some benchmarks.

6.1 Some savings

First of all, note that some multiplications can be saved easily from the fact that when the algorithm **Fast-GB-gcd** calls the algorithm **Half-GB-gcd**, the returned matrix is not used. Therefore, for such "top-level" calls to the algorithm **Half-GB-gcd**, there is no need computing the product $R_2[q]_{j_0}R_1$.

Note also that for interesting sizes of inputs (our implementation of the recursive algorithm is faster than usual Euclidean algorithms for several thousands of bits), we are in the domain of Karatsuba and Toom-Cook multiplications, and not far below the domain of the FFT-based multiplication. This leads to some improvements. For example, the algorithm **Fast-GB-gcd** should use calls to the algorithm **Half-GB-gcd** in order to gain γn bits instead of $\frac{n}{2}$, with a constant $\gamma \neq \frac{1}{2}$ that has to be optimized.

Amongst others, below a certain threshold in the size of the inputs (namely several hundreds of bits), a naive quadratic algorithm that has the requirements

of algorithm **Half-GB-gcd** is used. Moreover, each time the algorithm has to compute a GB quotient, indeed it computes several of them in order to obtain a 2×2 matrix with entries of length as close to the size of machine words as possible. This is done by considering only the two least significant machine words of the remainders (which gives a correct result, because of Lemma 6).

6.2 Comparison to other implementations of subquadratic gcd

We compared our implementation in a development version of GNU MP with those of Magma V2.10-12 and Mathematica 5.0, which both provide a subquadratic integer gcd. This comparison was performed on an Athlon MP 2200+, `laurent3.medicis.polytechnique.fr`. Our implementation wins over the quadratic gcd of GNU MP up from about 2500 words of 32 bits, i.e. about 24000 decimal digits. We used as test numbers both the worst case of the classical subquadratic gcd, i.e. consecutive Fibonacci numbers F_n and F_{n-1}, and the worst case of the binary variant, i.e. G_n and $2G_{n-1}$, where $G_0 = 0$, $G_1 = 1$, $G_n = -G_{n-1} + 4G_{n-2}$, which give all binary quotients equal to 1.

type, n	Magma V2.10-12	Mathematica 5.0	Fast-GB-gcd (GNU MP)
F_n, 10^6	2.89	2.11	0.70
F_n, $2 \cdot 10^6$	7.74	5.46	1.91
F_n, $5 \cdot 10^6$	23.3	17.53	6.74
F_n, 10^7	59.1	43.59	17.34
G_n, $5 \cdot 10^5$	2.78	2.06	0.71
G_n, 10^6	7.99	5.30	1.94

Fig. 8. Timings in sec. of the gcd routines of Magma, Mathematica and our implementation in GNU MP.

Our experiments show that our implementation in GNU MP of the binary recursive gcd is 3 to 4 times faster than the implementations of the classical recursive gcd in Magma or Mathematica. This ratio does not vary much with the inputs, since ratios for F_n and G_n are quite similar.

Acknowledgements.

The first author thanks Brigitte Vallée for introducing him to fast gcd algorithms and for very helpful discussions on that topic. We also thank the Medicis center for allowing the comparison to Magma and Mathematica.

References

1. The GNU MP Bignum Library. *Downloadable at http://www.swox.com/gmp/.*
2. A. Akhavi and B. Vallée. Average-bit complexity of Euclidean algorithms. In *Proc. of ICALP'00*, volume 1853 of *Lecture Notes in Computer Science*, pages 374–387. Springer-Verlag, 2000.
3. G. Hanrot, M. Quercia, and P. Zimmermann. The middle product algorithm. *Rapport de Recherche INRIA 4664*, 2002.
4. A. Karp and P. Markstein. High precision division and square root. *ACM Transactions on Mathematical Software*, 23:561–589, 1997.
5. D. Knuth. *The Art of Computer Programming, vol. 2: Seminumerical Algorithms.* Addison-Wesley, 1981.
6. A. Schönhage, A. F. W. Grotefeld, and E. Vetter. *Fast Algorithms, A Multitape Turing Machine Implementation.* BI-Wissenschaftsverlag, 1994.
7. A. Schönhage and V. Strassen. Schnelle Multiplikation grosser Zahlen. *Computing*, 7:281–292, 1971.
8. J. Sorenson. Two fast gcd algorithms. *Journal of Algorithms*, 16:110–144, 1994.
9. B. Vallée. Gauss' algorithm revisited. *Journal of Algorithms*, 12:556–572, 1991.
10. J. von zur Gathen and J. Gerhard. *Modern Computer Algebra.* Cambridge University Press, 2nd edition, 2003.
11. C. K. Yap. *Fundamental Problems in Algorithmic Algebra.* Oxford University Press, 2000.

Unité de recherche INRIA Lorraine
LORIA, Technopôle de Nancy-Brabois - Campus scientifique
615, rue du Jardin Botanique - BP 101 - 54602 Villers-lès-Nancy Cedex (France)

Unité de recherche INRIA Futurs : Parc Club Orsay Université - ZAC des Vignes
4, rue Jacques Monod - 91893 ORSAY Cedex (France)
Unité de recherche INRIA Rennes : IRISA, Campus universitaire de Beaulieu - 35042 Rennes Cedex (France)
Unité de recherche INRIA Rhône-Alpes : 655, avenue de l'Europe - 38334 Montbonnot Saint-Ismier (France)
Unité de recherche INRIA Rocquencourt : Domaine de Voluceau - Rocquencourt - BP 105 - 78153 Le Chesnay Cedex (France)
Unité de recherche INRIA Sophia Antipolis : 2004, route des Lucioles - BP 93 - 06902 Sophia Antipolis Cedex (France)

Éditeur
INRIA - Domaine de Voluceau - Rocquencourt, BP 105 - 78153 Le Chesnay Cedex (France)
http://www.inria.fr
ISSN 0249-6399