Implementing the asymptotically fast version of the elliptic curve primality proving algorithm
François Morain

To cite this version:
François Morain. Implementing the asymptotically fast version of the elliptic curve primality proving algorithm. 2005. hal-00004136v2

HAL Id: hal-00004136
https://hal.archives-ouvertes.fr/hal-00004136v2
Submitted on 4 Feb 2005

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.
IMPLEMENTING THE ASYMPTOTICALLY FAST VERSION OF THE ELLIPTIC CURVE PRIMALITY PROVING ALGORITHM

F. MORAIN

Abstract. The elliptic curve primality proving (ECPP) algorithm is one of the current fastest practical algorithms for proving the primality of large numbers. Its running time cannot be proven rigorously, but heuristic arguments show that it should run in time $\tilde{O}((\log N)^5)$ to prove the primality of $N$. An asymptotically fast version of it, attributed to J. O. Shallit, runs in time $\tilde{O}((\log N)^4)$. The aim of this article is to describe this version in more details, leading to actual implementations able to handle numbers with several thousands of decimal digits.

1. Introduction

From the work of Agrawal, Kayal and Saxena [2], we know that determining the primality of an integer $N$ can be done in proven deterministic polynomial time $\tilde{O}((\log N)^{10.5})$. More recently, H.-W. Lenstra and C. Pomerance have announced a version in $\tilde{O}((\log N)^6)$. Building on the work of P. Berrizbeitia [3], D. Bernstein [6] and P. Mihăilescu & R. Mocenigo [31], independently, have given improved probabilistic versions with a claim of proven complexity of $\tilde{O}((\log N)^4)$, reusing classical cyclotomic ideas that originated in the Jacobi sums test [1, 11]. For more on primality before AKS, we refer the reader to [14]. For the recent developments, see [5].

All the known versions of the AKS algorithm are for the time being too slow to prove the primality of large explicit numbers. On the other hand, the elliptic curve primality proving algorithm [3] has been used for years to prove the primality of always larger numbers*. The algorithm has a heuristic running time of $\tilde{O}((\log N)^5)$. In the course of writing [33], the author rediscovered the article [28], in which an asymptotically fast version of ECPP is described. This version, attributed to J. O. Shallit, has a heuristic running time of $\tilde{O}((\log N)^4)$. The aim of this paper is to describe FASTECPP, give a heuristic analysis of it and describe its implementation.

Section 2 collects some well-known facts on imaginary quadratic fields, that can be found for instance in [13]. Section 3 presents the basic ECPP algorithm and analyzes it. In Section 4, the fast version is described and its complexity estimated. Section 5 explains the implementation and Section 6 gives some actual timings on large numbers.

2. Quadratic fields

A discriminant $-D < 0$ is said to be fundamental if and only if $D$ is free of odd square prime factors, and moreover $D \equiv 3 \mod 4$ or when $4 \mid D$, $(D/4) \mod 4 \in \{1, 2\}$. The quantity

$$D(X) = \#\{D \leq X, -D \text{ is fundamental}\}$$

is easily seen to be $O(X)$.

*See the web page of M. Martin, http://www.ellipsa.net/, or that of the author

Date: February 4, 2005.
A fundamental discriminant may be written as:

$$-D = \prod_{i=1}^{l} q_i^*$$

where all $q_i^*$’s are distinct and $q_i^*$ is either $-4$ or $\pm 8$, or $q_i^* = (-1/q_i) q_i$ for any prime $q_i$. The number of genera is $g(-D) = 2^{l-1}$ and Gauss proved that this number divides the class number $h(-D)$ of $K = \mathbb{Q}(\sqrt{-D})$. Moreover, Siegel proved that $h = O(D^{1/2+\varepsilon})$ asymptotically.

The rational prime $p$ is the norm of an integer in $K$, or equivalently, $4p = U^2 + DV^2$ in rational integers $U$ and $V$ if and only if the ideal $(p)$ splits completely in the Hilbert class field of $K$, denoted $K_H$, an extension of degree $h(-D)$ of $K$. The probability that a prime $p$ splits in $K$ is $1/(2h(-D))$.

Using Gauss’s theory of genera of forms, it is known that if $(\frac{q_i}{p}) = 1$ for all $i$ (equivalently, $(p)$ splits in the genus field of $K$), then the probability of $(p)$ splitting in $K_H$ is $g(-D)/h(-D)$.

### 3. The basic ECPP algorithm

We present a rough sketch of the ECPP algorithm, enough for us to estimate its complexity. We do not insist on what happens if one of the steps fails, revealing the compositeness of $N$. More details can be found in [3].

#### 3.1. Elliptic curves over $\mathbb{Z}/N\mathbb{Z}$.

For us, an elliptic curve $E$ modulo $N$ will have an equation $Y^2 \equiv X^3 + aX + b$ with $\gcd(4a^3 + 27b^2, N) = 1$ and we will use the set of points $E(\mathbb{Z}/N\mathbb{Z})$ defined as:

$$E(\mathbb{Z}/N\mathbb{Z}) = \{(x : y : z) \in \mathbb{P}^2(\mathbb{Z}/N\mathbb{Z}), y^2z \equiv x^3 + axz^2 + bz^3 \} \cup \{O_E = (0 : 1 : 0)\}$$

which resembles the definition of an actual elliptic curve if $N$ is prime, $\mathbb{P}^2(\mathbb{Z}/N\mathbb{Z})$ being the projective plane over $\mathbb{Z}/N\mathbb{Z}$. The important point here is that if $p$ is a divisor of $N$, we can reduce the curve $E$ and a point $P$ on it via a reduction modulo $p$ of each integer, yielding a point $P_p$ on $E_p$. Moreover, we can define an operation on $E(\mathbb{Z}/N\mathbb{Z})$, called pseudo-addition, that adds two “points” $P$ and $Q$ with the usual chord-and-tangent law. This operation either yields a point $R$ or a divisor of $N$ if any is encountered when dividing. If $R$ exists, then it has the property that $R_p$ is the sum of $P_p$ and $Q_p$ on $E_p$ for all prime factors $p$ of $N$. Note also that $O_E$ reduces to the ordinary point at infinity on $E_p$.

We will need to exponentiate points in $E$. This is best defined using the division polynomials (see for instance [4] for a lot of properties on these). Remember that over a field $K$ there exist polynomials $\phi_m(X, Y), \psi_m(X, Y), \omega_m(X, Y)$ such that

$$[m]P = P + \cdots + P = [m](X, Y) = (\phi_m(X, Y)\psi_m(X, Y) : \omega_m(X, Y) : \psi_m^3(X, Y)).$$

All these polynomials can be computed via recurrence formulas and there is an $O(\log m)$ algorithm for this task (a variant of the usual binary method for exponentiating).

We will take ([4]) for the definition of $[m]P$ over $\mathbb{Z}/N\mathbb{Z}$. We note here that if $\psi_m(X, Y) = 0$, then $[m]P$ is equivalent to the point $O_E$.

For the sake of presenting the algorithm in a simplified setting, we prove (compare [2]):

**Proposition 3.1.** Let $N'$ a prime satisfying $(\sqrt{N} - 1)^2 \leq 2N' \leq (\sqrt{N} + 1)^2$. Suppose that $E(\mathbb{Z}/N\mathbb{Z})$ is a curve over $\mathbb{Z}/N\mathbb{Z}$, that $P = (x : y : 1)$ is such that $\gcd(y, N) = 1$, $\psi_{2N'}(x, y) = 0$ but $\gcd(\psi_{N'}(x, y), N) = 1$. Then $N$ is prime.

**Proof:** suppose that $N$ is composite and that $p \leq \sqrt{N}$ is one of its prime factors. Let us look at what happens modulo $p$. By construction, $P_p$ is not a 2-torsion point on $E_p$. Since $\psi_{N'}(x, y)$ is invertible modulo $p$, then $[N'](P_p) \neq O_{E_p}$ and therefore $P_p$ is of order $2N'$ modulo $p$. This is impossible, since $2N' \geq (\sqrt{N} - 1)^2 \geq (p - 1)^2 > (\sqrt{p} - 1)^2 \geq \#E_p$ by Hasse’s theorem. □
3.2. **Presentation of the algorithm.** We want to prove that $N$ is prime. The algorithm runs as follows:

[Step 1.] Repeat the following: Find an imaginary quadratic field $K = \mathbb{Q}(\sqrt{-D})$ of discriminant $-D$, $D > 0$, such that

\[
4N = U^2 + DV^2
\]

in rational integers $U$ and $V$. For all solutions $U$ of (2), compute $m = N + 1 - U$; if one of these numbers is twice a probable prime $N'$, go to Step 2.

[Step 2.] Build an elliptic curve $E$ over $\mathbb{Q}$ having complex multiplication by the ring of integers $\mathcal{O}_K$ of $K$.

[Step 3.] Reduce $E$ modulo $N$ to get a curve $E$.

[Step 4.] Find $P = (x : y : 1)$, $\gcd(y, N) = 1$ on $E$ such that $\psi_{2N'}(x, y) = 0$, but $\gcd(\psi_{N'}(x, y), N) = 1$. If this cannot be done, then $N$ is composite, otherwise, it is prime by Proposition 3.1.

[Step 5.] Set $N = N'$ and go back to Step 1.

3.3. **Analyzing ECPP.** We will now analyze all steps of the above algorithm and give complexity estimates using the parameter $L = \log N$. One basic unit of time will be the time needed to multiply two integers of size $L$, namely $O(L^{1+\mu})$, where $0 \leq \mu \leq 1$ ($\mu = 1$ for ordinary multiplication, or $\epsilon > 0$ for any fast multiplication method).

Clearly, we need $\log N$ steps for proving the primality of $N$. We consider all steps, one at a time, easier steps first.

3.4. **Analysis of Step 4.** Finding a point $P$ can be done by a simple algorithm that looks for the smallest $x$ such that $x^3 + ax + b$ is a square modulo $p$ and then extracting a square root modulo $p$, for a cost of $O((\log N)^{2+\mu})$. Note that we can do without this with the trick described in [3, §8.6.3], though we do not need this at this point.

Computing $\psi_{N'}(x, y)$ costs $O((\log N)^{2+\mu})$, and we need $O(1)$ points on average, so this step amounts for $O((\log N)^{2+\mu})$.

3.5. **Analyzing Step 2.** The original version is to realize $K_H/K$ via the computation of the minimal polynomial $H_D(X)$ of the special values of the classical $j$-invariant at quadratic integers. More precisely, we can view the class group $Cl(-D)$ of $K$ as a set of primitive reduced quadratic forms of discriminant $-D$. If $(A, B, C)$ is such a form, with $B^2 - 4AC = -D$, then

\[
H_D(X) = \prod_{(A,B,C)\in Cl(-D)} \left( X - j((-B + \sqrt{-D})/(2A)) \right).
\]

In [15], it is argued that the height of this polynomial is well approximated by the quantity:

\[
\pi \sqrt{D} \sum_{[A,B,C]\in Cl(-D)} \frac{1}{A},
\]

which can be shown to be $O((\log h)^2)$.

Evaluating the roots of $H_D(X)$ and building this polynomial can be done in $\tilde{O}(h^2)$ operations (see [15]). Note that this step does not require computations modulo $N$.

Alternatively, we could use the method of [12 8] for computing the class polynomial and get a proven running time of $\tilde{O}(h^2)$, but assuming GRH.
3.6. Analyzing Step 3. Reducing $E$ modulo $N$ is done by finding a root of $H_D(X)$ modulo $N$. This can be done with the Cantor-Zassenhaus algorithm (see [22] for instance). Briefly, we split recursively $H_D(X)$ by computing $\gcd((X + a)^{(N-1)/2} - 1, H_D(X)) \mod N$ for random $a$’s.

Computing $(X + a)^{(N-1)/2} \mod (N, H_D(X))$ costs $O((\log N)M(N,h)) = O(LM(N,h))$ where $M(N,d)$ is the time needed to multiply two degree $d$ polynomials modulo $N$. A gcd of two degree $d$ polynomials costs $M(N,d) \log d$ (see [22, Ch. 11]). The total splitting requires $\log h$ steps, but the overall cost is dominated by the first one, hence yields a time:

$$O(M(N,h) \max(L, \log h)).$$

We can assume that $M(N,d) = O(d^{1+\nu}L^{1+\mu})$ where again $0 \leq \nu \leq 1$.

3.7. Analysis of Step 1. This is the crucial step that will give us the clue to the complexity. Given $D$, testing whether \( m \) is satisfied involves the reduction of the ideal $(N, \sqrt{-D})$ that lies above $(N)$ in $K$, where $r^2 \equiv -D \mod (4N)$ (if $N$ is prime...). This requires the computation of $\sqrt{-D} \mod N$, using for instance the Tonelli-Shanks algorithm, for the cost of one modular exponentiation, i.e., a $O(L^{1+\mu})$ time. Then it proceeds with a half gcd like computation, for a cost of $O(L^{1+\mu})$ (see also section 5.2 below).

In the event that equation \( \mathbf{3} \) is solvable, then we need check that $m = 2N'$ and test $N'$ for primality, which costs again some $O(L^{1+\mu})$.

The heuristic probability of $m$ being of the given form is $O(1/L)$. Though quite realistic, it is impossible to prove, given the current state of the art in analytical number theory. Using this heuristics, we expect to need $O(L)$ splitting $D$’s. Let us take all discriminants less than $D_{\text{max}}$. They have class number close to $h$, though quite realistic, it is impossible to prove, given the current state of the art in analytical number theory. Using this heuristics, we expect to need $O(L)$ splitting $D$’s. Let us take all discriminants less than $D_{\text{max}}$. They have class number close to $h$, this gives a very small $O(L^{1+\mu})$.

Turning to complexity, the cost of Step 1 is then that of $O(L^2)$ solving of $\mathbf{3}$, followed by $O(L)$ probable primality tests:

$$O(L^2(\sqrt{-D} \mod N)^{2+\mu} + L^{1+\mu})) + O(L \cdot \left(\frac{L^{2+\mu}}{\text{probable primality}}\right)),$$

which is dominated by the first cost, namely $O(L^{4+\mu})$.

3.8. Adding everything together. Taking $D = O(L^2)$ readily implies $h = O(L)$, so that the cost of Step 2 is $O(L^2)$, and that of Step 3 is $O((\log L) L^{3+\mu+\nu})$, which dominates Step 4. All in all, we get that ECPP has heuristic complexity $O(L^{4+\mu})$ for one step, and therefore $O(L^{5+\mu})$ in totality.

3.9. Remark. In practice, the dominant term of the complexity of Step 1 is $O(n_D L^{2+\mu})$ where $n_D$ is the number of $D$’s for which we try solve equation $\mathbf{3}$. Depending on implementation parameters and real size of $N$, this number $n_D$ can be quite small. This gives a very small apparent complexity to ECPP, somewhere in between $L^3$ and $L^4$ and explains why ECPP seems so fast in practice (see for instance [21]).

4. The fast version of ECPP

4.1. Presentation. When dealing with large numbers, all the time is spent in the finding of $D$, which means that a lot of squareroots modulo $N$ must be computed. A first way to reduce the computations, alluded to in [3, §8.4.3], is to accumulate squareroots, and reuse them, at the cost of some multiplications. For instance, if one has $\sqrt{-3}$ and $\sqrt{5} = \sqrt{-20}/\sqrt{-4}$, then we can build $\sqrt{-15}$, etc.
A better way that leads to the fast version consists in computing a basis of small square roots and build discriminants from this basis. Looking at the analysis carried out above, we see that we need $O(L^2)$ discriminants to find a good one. The basic version finds them by using all discriminants that are of size $O(L^2)$. As opposed to this, one can build those discriminants as $-D = (-p)(q)$, where $p$ and $q$ are taken from a pool of size $O(L)$ primes.

More formally, we replace Step 1. by Step 1’. as follows:

[Step 1’.]
1.1. Find the $r = O(L)$ smallest primes $q^*$ such that $\left(\frac{q^*}{N}\right) = 1$, yielding $\mathcal{Q} = \{q_1^*, q_2^*, \ldots, q_r^*\}$.
1.2. Compute all $\sqrt{q^*}$ mod $N$ for $q^* \in \mathcal{Q}$.
1.3. For all pairs $(q_i^*, q_j^*)$ of $\mathcal{Q}$ for which $q_i^* q_j^* = -D < 0$, try to solve equation (2).

The cost of this new Step 1 is that of computing $r = O(L)$ square roots modulo $N$, for a cost of $O(L \cdot L^2 + \mu \cdot \text{probable primality})$. Then, we still have $O(L^2)$ reductions. The new overall cost of this phase decreases now to:

$$O(L \cdot \sqrt{\text{square roots of } q^*}) + O(L^2 \cdot L^{1+\mu}) + O(L \cdot L^{2+\mu})$$

which yields namely $O(L^{3+\mu})$. Note here how the complexity decomposes as 3 = 1 + 2 or 2 + 1 depending on the sub-algorithms.

Putting everything together, we end up with a total cost of $O(L^{4+\mu})$ for this variant of ECPP.

4.2. Remarks.

4.2.1. Complexity issues. We can slightly optimize the preceding argument, by using all subsets of $\mathcal{Q}$ and not only pairs of elements. This would call for $r = O(\log \log N)$, since then $2^r = L^2$ could be reached. Though useful in practice, this phase no longer dominates the cost of the algorithm.

Moreover, we can see that several phases of fastECPP have cost $\tilde{O}(L^3)$, which means that we would have to fight hard to decrease the overall complexity below $\tilde{O}(L^4)$.

4.2.2. A note on discriminants. Note that we use fundamental discriminants only, as non fundamental discriminants lead to curves that do not bring anything new compared to fundamental ones. Indeed, if $D = f^2D$, with $D$ fundamental, then there is a curve having CM by the order of discriminant $D$. Writing $4N = U^2 + Df^2V^2$, its cardinality is $N + 1 - U$, the same as the corresponding curve associated to $D$.

4.2.3. A note on class numbers. As soon as we use composite discriminants $-D$ of the form $q_{i1}^* q_{i2}^*$, Gauss’s theorem tells us that the class number $h(-D)$ is even. This could bias our estimation, but we conjecture that the effect is not important.

5. Implementation

5.1. Computing class numbers. In order to make the search for $D \in \mathcal{D}$ efficient, it is better to control the class number beforehand. Tables can be made, but for larger computations, we need a fast way to compute $h(-D)$. Subexponential methods exist, assuming the Generalized Riemann Hypothesis. From a practical point of view, our $D$’s are of medium size. Enumerating all forms costs $O(h^2)$ with a small constant, and Shanks’s baby-steps/giant-steps algorithm costs $O(\sqrt{h})$ but with a large constant. It is better here to use the explicit formula of Louboutin [29] that yields a practical method in $O(h)$ with a very small constant.
5.2. An improved Cornacchia algorithm. Step 1 needs squareroots to be computed, and some half gcd to be performed. Briefly, Cornacchia’s algorithm runs as follows (see [34]):

procedure Cornacchia(d, p, t)
\{t is such that \( t^2 \equiv -d \pmod{p}, \frac{p}{2} < t < p \}\n\begin{enumerate}
  \item[(a)] \( r_{-2} = p, r_{-1} = t \); \( w_{-2} = 0, w_{-1} = 1 \);
  \item[(b)] for \( i \geq 0 \) while \( r_{i-1} > \sqrt{p} \) do
    \begin{enumerate}
      \item \( r_{i-2} = a_i r_{i-1} + r_i, 0 \leq r_i < r_{i-1} \);
      \item \( w_i = w_{i-2} + a_i w_{i-1} \) (\star);
      \item if \( r_{i-1}^2 + dw_{i-1}^2 = p \) then return \((r_{i-1}, w_{i-1})\) else return \( \emptyset \).
    \end{enumerate}
\end{enumerate}

We end the for loop once we get \( r_{i-1} \leq \sqrt{p} < r_{i-2} \). As is well known, the \( a_i \)'s are quite small and we can guess their size by monitoring the number of bits of the \( r_i \)'s, thus limiting the number of long divisions. One can use a fast variant for this half gcd if needed, in a way reminiscent of Knuth.

Moreover, from the theory, we know that this algorithms almost always returns that the empty set in step 2c), since the probability of success if \( 1/(2h(-d)) \). Therefore, when \( h \) is large, we can dispense of the multiprecision computations in equation (\star). We replace it by single precision computations:

\[ w_i = w_{i-2} + a_i w_{i-1} \mod 2^{32} \]

and at the end, we test whether \( r_{i-1}^2 + dw_{i-1}^2 = p \mod 2^{32} \). If this is the case, then we redo the computation of the \( w_i \)'s and check again again.

5.3. Factoring \( m \). Critical parameters are that related to the factorization of \( m \), since in practice we try to factor \( m \) to get it of the form \( cN' \) for some \( B \)-smooth number \( c \).

As shown in [20], the number of probable prime tests we will have to perform is \( t = O((\log N)/(\log B)) \) and we will end up with \( N' \) such that \( N/N' \approx \log B \).

For small numbers, we can factor lots of \( m \) doing the following. In a first step, we compute

\[ r_i = (N + 1) \mod p_i \]

for all \( p_i \leq B \)'s, which costs \( \pi(B)(\log N)^{1+\varepsilon} \), where \( \pi(B) = O(B/\log B) \) is the number of primes below \( B \) and the other term being the time needed to divide a multi-digit number by a single digit number.

Then, sieving both \( m = N + 1 - U \) and \( m' = N + 1 + U \) is done by computing \( u_i = U \mod p_i \) and comparing it to \( \pm r_i \) for primes \( p_i \) such that \( (-D/p_i) \not\equiv -1 \). See [3, 32] for more details and tricks.

The cost of this algorithm for \( t \) values of \( m \) is

\[ O(\pi(B)(\log N)^{1+\varepsilon}) + O(t \cdot \pi(B)(\log N)^{1+\varepsilon}) \]

where the second term is that for computing \( U \mod p_i \), which is slightly half that of \( (N + 1) \mod p_i \), since \( U = O(\sqrt{N}) \). Since we need to perform also \( t \) probable prime tests (say, a plain Fermat one), then the cost is

\[ O(tBL) + O(tL^{2+\mu}) = O(BL^2) + O(L^{3+\mu}) \]

and therefore the optimal value for \( B \) is \( B = O(L) \).

For larger numbers, it is better to use the stripping factor algorithm in [34], for a cost of \( O(B(\log B)^2) \), the optimal value of \( B \) being \( B = O((\log N)^3) \).
5.3.1. Remark. Suppose now that we have found $N'$ and that $m = N + 1 - U = cN'$. Then we will have to compute

$$r'_i = (N' + 1) \mod p_i$$

which may be computed as:

$$r'_i = (r_i - u_i)/c + 1 \mod p_i.$$ 

Computing the right hand side is faster, since $c$ is ordinarily small compared to $N'$.

5.3.2. Using an early abort strategy. This idea is presented in [20]. We would like to go down as fast as possible. So why not impose $N/N'$ greater than some given bound? Candidates $N'$ need be tested for probable primality only if this bound is met. From what has been written above, we can insist on $N/N' \approx \log B$. In practice, we used a bound $\delta$ and used $N/N' \geq 2^\delta$.

5.3.3. Using new invariants. Proving larger and larger numbers forces us to use larger and larger $D$’s, leading to larger and larger polynomials $H_D$. For this to be doable, new invariants had to be used, so as to minimize the size of the minimal polynomials. This task was done using Schertz’s formulation of Shimura’s reciprocity law [35], with the invariants of [18] as demonstrated in [16] (alternatively see [24, 23]). Note that replacing $j$ by other functions does not change the complexity of the algorithm, though it is crucial in practice.

5.3.4. Step 3 in practice. We already noted that this step is the theoretically dominating one in fastECPP, with a cost of $O((\log L)L^{3+\mu+\nu})$. In practice, even for small values of $h$, we can assume $\nu \approx 0$ (using for instance the algorithm of [30] for polynomial multiplication).

Galois theory comes in handy for reducing the $\log L$ term to a $\log \log L$ one, if we insist on $h$ being smooth. Then, we replace the time needed to factor a degree $h$ polynomial by a list of smaller ones, the largest prime factor of $h$ being $\log h$. We already used that in ECPP, using [27, 17]. Typical values of $h$ are now routinely in the 10000 zone.

It could be argued that keeping only smooth class numbers is too restrictive. Note however, that class numbers tend to be smoother than ordinary numbers [10].

5.3.5. Improving the program. The new implementation uses GMP† for the basic arithmetic, which enables one to use mpfr [26] and mpc [19], thus leading to a complete program that can compute polynomial $H_D$’s on the fly, contrary to the author’s implementation of ECPP, prior to version 11.0.5. This turned out to be the key for the new-born program to compete with the old one.

5.4. fastECPP. We give here the expanded algorithm corresponding to step 1’. Using a smoothness bound $B$, we need approximately $t = \exp(-\gamma) \log N/\log B$ values of $m$ and therefore roughly $t/2$ discriminants. The probability that $D$ is a splitting discriminant is $g(-D)/h(-D)$. Therefore we build discriminants until

$$\sum_D g(-D)/h(-D) \approx t/2.$$ 

One way of building these discriminants is the following: we let $r$ increase and build all or some of the subsets of $\{q_1^*, \ldots, q_r^*\}$ until the expected number of $D$’s is reached. After this, we sort the discriminants with respect to $(h(-D)/g(-D), h(-D), D)$ and treat them in this order.

†http://www.swox.com/gmp/
6. Benchmarks

First of all, it should be noted that ECPP is not a well defined algorithm, as long as one does not give the list of discriminants that are used, or the principles that generate them.

Since the first phase of ECPP requires a tree search, testing on a single number does not reveal too much. Averaging on more than 20 numbers is a good idea.

Our current implementation uses GMP\(^\text{‡}\) for the basic arithmetic, which enables one to use \texttt{mpfr} \cite{26} and \texttt{mpc} \cite{19}, thus leading to a complete program that can compute polynomial \(H_D\)'s on the fly, contrary to the author’s implementation of ECPP, prior to version 11.0.5. This turned out to be the key for the new-born program to compete with the old one.

We give below some timings obtained with this implementation, after a lot of trials. We used as prime candidates the first twenty primes of 1000, 1500, and 2000 decimal digits. Critical parameters are as follows: we used \(D \leq 10^7\), \(h \leq 10^3\), \(\delta = 12\) (see section 5.3.2). For 1000 and 1500 decimal digits, we limited the largest prime factor of \(h\) to be \(\leq 30\) and for 2000 dd, it was put to 100. This parameter has an influence in Step 3. For the extraction of small prime factors (used in the algorithm described in \cite{20} and denoted EXTRACT in the sequel), we used \(B = 8 \cdot 10^6\), \(10^7\), \(3 \cdot 10^7\) for the three respective sizes.

\texttt{SQR T} refers to the computation of \(\sqrt{q_i}\); \texttt{CORN} to Cornacchia, \texttt{PRP} to probable primality tests; \texttt{HD} is the time for computing polynomials \(H_D\) using the techniques described in \cite{16}, \texttt{jmod} the time to solve it modulo \(p\); then \texttt{1st} refers to the building phase (step 1), \texttt{2nd} to the other ones; total is the total time, check the time to verify the certificate. Follow some data concerning \(D\), \(h\) and the size of the certificates (in kbytes). All timings are cumulated CPU time on an AMD Athlon 64 3400+ running at 2.4GHz.

<table>
<thead>
<tr>
<th></th>
<th>min</th>
<th>max</th>
<th>avg</th>
<th>std</th>
</tr>
</thead>
<tbody>
<tr>
<td>\texttt{SQR T}</td>
<td>19</td>
<td>34</td>
<td>25</td>
<td>3</td>
</tr>
<tr>
<td>\texttt{CORN}</td>
<td>10</td>
<td>24</td>
<td>17</td>
<td>4</td>
</tr>
<tr>
<td>\texttt{EXTRACT}</td>
<td>60</td>
<td>84</td>
<td>74</td>
<td>5</td>
</tr>
<tr>
<td>\texttt{PRP}</td>
<td>74</td>
<td>124</td>
<td>102</td>
<td>14</td>
</tr>
<tr>
<td>\texttt{HD}</td>
<td>0</td>
<td>7</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>\texttt{jmod}</td>
<td>42</td>
<td>99</td>
<td>61</td>
<td>11</td>
</tr>
<tr>
<td>\texttt{1st}</td>
<td>178</td>
<td>276</td>
<td>234</td>
<td>27</td>
</tr>
<tr>
<td>\texttt{2nd}</td>
<td>79</td>
<td>136</td>
<td>99</td>
<td>12</td>
</tr>
<tr>
<td>\texttt{total}</td>
<td>260</td>
<td>387</td>
<td>334</td>
<td>34</td>
</tr>
<tr>
<td>\texttt{check}</td>
<td>18</td>
<td>22</td>
<td>20</td>
<td>0</td>
</tr>
<tr>
<td>\texttt{nsteps}</td>
<td>124</td>
<td>156</td>
<td>143</td>
<td>7</td>
</tr>
<tr>
<td>\texttt{certif}</td>
<td>396</td>
<td>456</td>
<td>435</td>
<td>13</td>
</tr>
<tr>
<td>\texttt{D}</td>
<td>8740947</td>
<td>120639</td>
<td>608050</td>
<td></td>
</tr>
<tr>
<td>\texttt{h}</td>
<td>1000</td>
<td>31</td>
<td>87</td>
<td></td>
</tr>
</tbody>
</table>

\begin{table}[h]
\centering
\begin{tabular}{|c|c|c|c|c|}
\hline
        & min & max & avg & std \\
\hline
\texttt{SQR T} & 19  & 34  & 25  & 3   \\
\texttt{CORN}   & 10  & 24  & 17  & 4   \\
\texttt{EXTRACT} & 60  & 84  & 74  & 5   \\
\texttt{PRP}    & 74  & 124 | 102 | 14  \\
\texttt{HD}     & 0   & 7   & 2   & 2   \\
\texttt{jmod}   & 42  & 99  & 61  & 11  \\
\texttt{1st}    & 178 | 276 | 234 | 27  \\
\texttt{2nd}    & 79  | 136 | 99  | 12  \\
\texttt{total}  & 260 | 387 | 334 | 34  \\
\texttt{check}  & 18  | 22  | 20  | 0   \\
\texttt{nsteps} | 124 | 156 | 143 | 7   \\
\texttt{certif} | 396 | 456 | 435 | 13  \\
\texttt{D}      | 8740947 | 120639 | 608050 \\
\texttt{h}      | 1000 | 31  | 87  |
\hline
\end{tabular}
\caption{1000 decimal digits}
\end{table}

Looking at the average total time, we see that it follows very closely the \(O((\log N)^4)\) prediction. Note also that the dominant time is that of the PRP tests, and that all phases have time close to what was predicted.

\(^\text{‡}\)http://www.swox.com/gmp/
We have described in greater details the fast version of ECPP. We have demonstrated its efficiency. As for ECPP, it is obvious that the computations can be distributed over a network of computers. We refer the reader to [20] for more details. Note that the current record of 15041 decimal digits (with the number $4405^{2638} + 2638^{4405}$ see transaction in the NMBRTHRY mailing list), was settled using this approach. Many more numbers were proven prime using either the monoprocessor version or the distributed one, most of them from the tables of numbers of the form $x^y + y^x$ made by P. Leyland\textsuperscript{§}.

Cheng [9] has suggested to use ECPP to help his improvement of the AKS algorithm, forcing $m = cN'$ to have $N' - 1$ divisible by a given prime large prime of size $O((\log N)^2)$. The same idea can be used to speed up the Jacobi sums algorithm, and this will be detailed elsewhere.

\textsuperscript{§}http://www.leyland.vispa.com/numth/primes/xyyx.htm
Acknowledgments. The author wants to thank N. Bourbaki for making him dive once again in the field of primality proving and D. Bernstein for stimulating emails on the existence and analysis of fastECPP. My co-authors of [20] were a source of stimulation through their records. Thanks also to P. Gaudry for never ending discussions on how close we are to infinity, as far as fast algorithms are concerned. D. Stehlé and P. Zimmermann for useful discussions around Cornacchia and fast sieving. Thanks to A. Enge for his help in improving the exposition, and to D. Bernardi for his remarks that helped clarify the exposition.

References


(F. Morain) LIX ÉCOLE POLYTECHNIQUE, CNRS/UMR 7161, INRIA/FUTURS, F-91128 PALAISEAU CEDEX, FRANCE

E-mail address, F. Morain: morain@lix.polytechnique.fr