IMAPP MATHEMATICS
RADBOUD UNIVERSITY NIJMEGEN
The Netherlands

# Better paths for elliptic curve primality proofs

**Wieb Bosma**
**Antal Járai**
**Gyöngyvér Kiss**

**Report No. 0902 (July 2009)**

IMAPP MATHEMATICS
RADBOUD UNIVERSITY NIJMEGEN
Toernooiveld
6525 ED Nijmegen
The Netherlands

**Abstract**

An important part of the Elliptic Curve Primality Proving algorithm consists of finding a sequence of elliptic curves with appropriate properties. In this report we consider and test some strategies to search for an improved sequence, as part of a programme to obtain improved heuristics and running time analysis of the whole algorithm.

# 1 Introduction

Although mathematicians have been interested in prime numbers since ancient times, there is still no general, deterministic, unconditional, practical, polynomial time algorithm for primality proving. If we are willing to drop some of these adjectives, the situation becomes different. There exist tests of Lucas-Lehmer type that can certify primes of very large size but only of a special form. The Miller-Rabin test has a version that is practical and runs in polynomial time but only provides primality proofs conditional on a generalized version of the Riemann hypothesis; the variant commonly used only produces *probable primes*, in the sense that with small probability a composite number will pass the tests. The now famous AKS test [1], on the other hand, is deterministic and proves primality in polynomial time, but has yet to be proven practical; for an improved randomized version see Bernstein [3]. Somewhere in between there are two algorithms that can prove primality in situations of practical importance (primes of hundreds or several thousands of decimal digits), of which the complexity analysis shows sub-exponential dependency on the size of the prime, but for which polynomial time bounds have not been proven. The significance of such primality tests has increased with the widespread use of primes for cryptographic purposes.

This paper aims to contribute towards the rigorous analysis of one of the two successful practical tests for primality proving, ECPP see [2], based on elliptic curve arithmetic, by looking at heuristics for an optimal choice of parameters.

In what follows, we will always assume that $n$ is the input of our algorithm, for which we want to construct a primality proof; also, we assume that $n$ is a probable prime in the sense that it has passed some compositeness tests, and that it is free of small divisors. In particular, $\gcd(n, 6) = 1$. However, we should not *assume* that $n$ is prime.

# 2 Elliptic curves

The main objective in the Elliptic Curve Primality Proving (ECPP for short) algorithm, which will be described in detail in the next section, is to construct a sequence of integers $n_0, n_1, \ldots, n_k$ that will be proved prime in reversed order, ending at $n_0 = n$. When the proof is completed, these numbers $n_i$ will be (divisors of) orders of groups of points of elliptic curves over finite fields, as they are defined modulo $n_{i-1}$. However, during the construction we can not use yet that $n_{i-1}$ is prime, and this means that we will have to be careful in defining elliptic curves modulo $n$, and their arithmetic; see [5].

**Definition 2.1.** The *projective plane modulo* $m$, denoted $\mathbb{P}^2(\mathbb{Z}/m\mathbb{Z})$, for a positive integer $m$, consists of equivalence classes $(x : y : z)$ of triples $(x, y, z) \in$

$(\mathbb{Z}/m\mathbb{Z})^3$ satisfying $\gcd(x, y, z, m) = 1$, under the equivalence $(x, y, z) \sim (\lambda x, \lambda y, \lambda z)$ for any $\lambda \in (\mathbb{Z}/m\mathbb{Z})^*$.

**Definition 2.2.** Let $m$ be an integer with $\gcd(m, 6) = 1$. An *elliptic $E$ curve modulo $m$* is a pair $(a, b) \in (\mathbb{Z}/m\mathbb{Z})^2$ for which $\gcd(4a^3 + 27b^2, m) = 1$. *The set of points $E[\mathbb{Z}/m\mathbb{Z}]$* on an elliptic curve $E$ modulo $m$ consists of $(x : y : z) \in \mathbb{P}^2(\mathbb{Z}/m\mathbb{Z})$ for which

$$y^2z = x^3 + axz^2 + bz^3.$$

**Definition 2.3.** Let $m$ be an integer with $\gcd(m, 6) = 1$, and $a \in \mathbb{Z}/m\mathbb{Z}$. Define $V = V[\mathbb{Z}/m\mathbb{Z}]$ as the set of all $(x : y : 1) \in \mathbb{P}^2(\mathbb{Z}/m\mathbb{Z})$ together with $O = (0 : 1 : 0) \in \mathbb{P}^2(\mathbb{Z}/m\mathbb{Z})$. Given $(V, a)$, the *partial addition algorithm* computes for any pair $P = (x_p : y_p : z_p)$, $Q = (x_q : y_q : z_q) \in V$ either an element $R = (x_r, y_r, z_r) \in V$ called the *sum* $P + Q$ of $P$ and $Q$, or a non-trivial divisor $d$ of $m$, as follows.

(1) If $x_p = x_q$ and $y_p = -y_q$ then output $R = (0 : 1 : 0)$.

(2) If $x_p \neq x_q$ and $y_p = -y_q$ let $v = x_p - x_q$, otherwise let $v = y_p + y_q$; then use the extended Euclidean algorithm to compute $s, t \in \mathbb{Z}/m\mathbb{Z}$ such that $sv + tm = d = \gcd(v, m)$. If $d > 1$ then output $d$.

(3) Let $\lambda = s(y_p - y_q)$ if $x_p \neq x_q$ and $\lambda = s(3x_p^2 + a)$ if $x_p = x_q$. Output $R = (\lambda^2 - x_p - x_q : \lambda(\lambda^2 - 2x_p - x_q) + y_p : 1)$.

**Remark 2.4.** *If $m = p$ is prime, the set $E[\mathbb{Z}/p\mathbb{Z}]$ forms an abelian group for any ellipic curve $E = E_{a,b}$, with unit element $O$. In this case, the partial addition algorithm, which will now always produce a sum of two points on $E$, is equivalent to the usual addition algorithm.*

*Moreover, it can be shown that for a prime divisor $p$ of arbitrary $m$ coprime to 6, the sum $R$ produced by the partial addition algorithm for any two points $P, Q$ on an elliptic cruve $E_{a,b}$ modulo $m$, has the property that $R_p$ (obtained by reducing the coordinates of $R$ modulo $p$) is the sum of (the similarly defined) points $P_p$ and $Q_p$ in the group $E_{\bar{a},\bar{b}}[\mathbb{Z}/p\mathbb{Z}]$, where $\bar{a} \equiv a \bmod p$, and $\bar{b} \equiv b \bmod p$.*

Using the partial addition algorithm repeatedly, it is of course possible to obtain a partial multiplication algorithm, which computes either $k \cdot P$ or finds a divisor of $m$, for any positive integer $k$, given any $P \in V$ and any $a$ as before. However, there are various ways to speed up this computation of $k \cdot P$, using partial doubling, and the fact that it is not necessary to keep track of the $y$-coordinate.

In the next sections we will occassionally be sloppy, and write about the sum and multiples of points on elliptic curves modulo $n$; we mean the result of application of the partial addition and multiplication algorithms, which in exceptional cases means that a divisor of $n$ is found, rather than a point.

# 3 ECPP

We give an outline of the ECPP algorithm; some of the necessary definitions and details will be given in the subsequent sections. The algorithm is based on the following theorem.

**Theorem 3.1.** *Let $n_0 \in \mathbb{N}$ with $\gcd(6, n_0) = 1$. Let $E$ be an elliptic curve modulo $n_0$, and let $m, n_1 \in \mathbb{N}$ with $n_1 \mid m$. Suppose that for every prime factor $q$ of $n_1$ there exist $P \in E$ such that $mP = 0_E$ and $\frac{m}{q}P \neq 0_E$. Then for all prime factors $p$ of $n_0$ holds $\#E[\mathbb{Z}/p\mathbb{Z}] \equiv 0 \bmod n_1$.*

**Corollary 3.2.** *Suppose that the hypotheses of Theorem (3.1) are satisfied. Then:*

$$n_1 > (n_0^{\frac{1}{4}} + 1)^2 \quad \Rightarrow \quad n_0 \text{ is prime.}$$

Note that the requirement is that $n_1$ exceeds a bound slightly larger than $\sqrt{n_0}$.

Essential in the proof of the Corollary is the Theorem of Hasse, stating that the number of points on any elliptic curve modulo a prime $p$ equals $p + 1 - t$ for some integer $t$ with $|t| \leq 2\sqrt{p}$. Theorem 3.1 easily follows from the observation that, modulo any prime divisor $p$ of $n_0$ the conditions imply that $\#E[\mathbb{Z}/p\mathbb{Z}]$ can not be a proper divisor of $n_1$.

Starting point for the application of ECPP will always be a probable prime $n_0 = n$; it is assumed that $n$ will be free of small prime factors (in particular 2 and 3), and that $n$ has passed certain compositeness tests (of Miller-Rabin type). This will make it very likely that $n$ is indeed prime; the objective is to prove that.

Given such an integer $n$, the basic ECPP algorithm proceeds roughly in these three stages:

(D) starting with $n_0 = n$, find a sequence of probable primes $n_0, n_1, \ldots, n_k$, such that $n_{i+1}$ divides the order of some elliptic curve modulo $n_i$, such that $n_{i+1} > (\sqrt[4]{n_i} + 1)^2$, and such that $n_k$ is so small that primality can be verified by easy inspection (or trial division).

(F) For each of the integers $n_i$ with $i = 0, 1, \ldots, k-1$, construct an elliptic curve $E_i$ of order a multiple of $n_{i+1}$ modulo $n_i$, together with a point $P_i$ of order $n_{i+1}$ on the curve modulo $n_i$.

(P) Verify that the conditions of Theorem 3.1 hold for the given probable primes $n_i$, curves $E_i$ and points $P_i$, for $i = k-1, k-2, \ldots, 0$.

The difficulties in each of the three steps have been obscured here by lack of detail. In the following subsections we will fill in some important details.

## 3.1 Downrun

The first part of the algorithm will be called recursively with input $n_i$; the main objective is to find $n_{i+1}$. This is what happens at level $i$:

(D) select a pair $D, u$ of negative discriminant $D$ and integer $u$ such that $n_i + 1 + u$ is the product of small primes and a probable prime $n_{i+1}$ that exceeds $(\sqrt[4]{n_i} + 1)^2$.

In practice this is what happens:

($D_0$) Prepare a list of primes up to some bound $s = s(n_i)$, as well as list of negative fundamental discriminants up to a bound $d = d(n_i)$ that factor completely in a product of primes from the prime list, together with their full prime factorization.

($D_1$) For all discriminants $D$ in the list, find the reduction of the binary quadratic form $Ax^2+Bxy+Cy^2$ of discriminant $D$, where $A=n$, $B^2 \equiv -D \bmod n$, and $C=(B^2+D)/(4n)$. This requires the modular square root of $-D$ modulo $n$, which is obtained as a product of the square roots of the prime factors of $-D$. If this provides $\nu$ with $\nu \cdot \overline{\nu} = n$, then $u = \nu + \overline{\nu}$.

($D_2$) All pairs $D, u$, found in the previous step, for which a probably prime $q$ dividing $n_i + 1 - u$ can be found such that $(n_i+1-u)/q$ is the product of small primes only, are added to a list; similarly for $-u$ instead of $u$.

($D_3$) Select the best possible pair $D, u$ from the list, and let $n_{i+1}$ be the probable prime $q$ for which $(n_i+1-u)/q$ is the product of small primes.

Several comments are in order.

Usually a 'master-list' of primes up to some bound $B$ is prepared in advance; the bound $s(n_i)$ (and hence the list) in step $D_0$ may depend on $i$ (the level of the recursion arrived at), but should be at most $B$. Similarly for the list of discriminants, and the bound $d(n_i)$. This means that step $D_0$ will mainly consist of the selection of sub-lists, from precompiled lists that are computed once for all $n$ up to a fixed size $N$. In Step $D_2$ the probable factorization of possible curve orders $n_i + 1 - u$ has to be found; one uses a smoothness bound $b = b(n_i)$, that is, all prime factors smaller than $b$ are removed (and considered small).

Step $D_3$ is the main step we will focus on in what follows: in particular, we need to specify what *best* means here.

Note that backtracking may be necessary: it is possible that at some level the list of suitable $D, u, q$ becomes empty!

The output of the first phase of the algorithm will consist of a sequence of triples $(n_i, D_i, u_i)$ for $i = 0$ to $i = k-1$ such that $n_i + 1 + u_i$ is the product of small primes and a probable prime $n_{i+1}$ that exceeds $(\sqrt[4]{n_i}+1)^2$.

## 3.2 Finding elliptic curves

The second phase in the primality testing algorithm can be done as follows. Again, we describe the steps to be taken at level $i$.

(F) Find elliptic curves $E_i$ and points $P_i$ on $E_i[\mathbb{Z}/n_i\mathbb{Z}]$ with the property that if $n_i$ is prime, then the order of $P_i$ is $n_{i+1}$.

This is done as follows.

($F_0$) Compute an auxiliary polynomial $G_i \in \mathbb{Z}[x]$; see the comments below.

[$F_1$) Find a root $j_i$ of $G_i \bmod n_i$ in $\mathbb{Z}/n_i\mathbb{Z}$, as well as an integer $t_i$ such that the Jacobi symbol $\left(\frac{t_i}{n_i}\right)$ equals $-1$.

($F_2$) Define elliptic curves $E_i'$ and $E_i''$ by

$$E_i' : \quad y^2 = x^3 + 3kx + 2k$$

and

$$E_i'' : \quad y^2 = x^3 + 3kt_i^2 x + 2kt_i^3$$

where $k = \frac{j}{1728-j}$.

(F$_3$) Find (for example by randomly choosing) a point on $E_i'$ or $E_i''$ that has order $n_{i+1}$, if $n_i$ is prime.

**Remark 3.3.** *The auxiliary polynomial $G_i$ is the Hilbert (or Weber) polynomial or a variant of this. The two elliptic curves are the twists of the elliptic curve with $j$-invariant $j_i$. We refer to [2] and [5] for more details, as this part of the algorithm plays no major role in what follows.*

In the third phase of the algorithm one simply checks the requirements of Theorem 3.1.

# 4 Asymptotic running time

The following observation plays an important role in the running time analysis: if we are able to fully factor curve cardinalities $m = n_0 + 1 + s$ up to a bound $b(n_0)$, the probability that one such curve cardinality $m$ leads to a new node will be the probability that the second largest prime factor of $m$ is less then $b(n_0)$; this is approximately

$$e^{-\gamma}\frac{\log b(n_0)}{\log n_0}.$$

It is reasonable to suppose that, if we have $e(n_0)$ such curve orders $m$, the number of new nodes has a probability distribution with average approximately

$$\lambda = e^{-\gamma}\frac{\log b(n_0)}{\log n_0}e(n_0).$$

For each negative discriminant $D \leq -7$ the probability of success is

$$\frac{1}{2h(D),}$$

where $h(D)$ is the ideal class number of $\mathbb{Q}(\sqrt{D})$. As each successful case results in 2 $m$'s, we expect

$$e(n_0) \approx \sum_D \frac{1}{h(D)}.$$

Let us give a brief analysis of the important steps; this strongly depends on the time needed for multiplication. To keep the calculations as far as possible independent from the given multiplication method, the time that we need to multiply arbitrary $k$-bit numbers is denoted by $m(k)$.

(D$_1$) We need the square root of the $s(n_i)$-smooth discriminants modulo $n_i$ below $d(n_i)$. In our implementation $s(n_i) = \log {n_i}^{1.3}$ and $d(n_i)$ is below $\log {n_i}^2$. As a standard example, we may choose $d(n_i) \approx \log {n_i}^2$. From the residue classes mod 16 only the classes $3, 4, 7, 8, 11, 15$ are kept. The discriminants have to be free from the square of any odd prime. This way, asymptotically we obtain $\frac{9}{(4\pi^2-6)d(n_i)}$ discriminants below $d(n_i)$. Suppose we only keep the discriminants which are $s(n_i) \approx d(n_i)^c$ smooth with appropriate, $0 < c < 1$. We are calculating the square root of primes below $s(n_i)$ for which $\left(\frac{n}{p}\right) = 1$. This way we need $O(\log {n_i}^3 \log\log\log n_i)$ bit operations if $c = \frac{1}{2}$ and $o(\log {n_i}^3)$ bit operations if $c < \frac{1}{2}$.

To check the discriminants up to $d(n_i)$ to see whether they are $d(n_i)^c$ smooth or not, and to find their factorization, we need $O(d(n_i)^{1+c} \log d(n_i))$ bit operations using trial division. With an appropriate choice of $s(n_i)$ and $d(n_i)$ it can be neglected.

For a given discriminant $D$ we check whether the conditions $\left(\frac{D}{n}\right)$ and $\left(\frac{n}{p}\right)$ for all odd prime factors of $D$ are satisfied and extract a square root from $D \bmod n_i$. This takes $O(\frac{m(\log n_i)d(n_i)}{\log d(n_i)})$.

For the remaining $D$'s, the reduction algorithm of quadratic forms (or, alternatively, the Cornacchia algorithm) is applied. The running time (in both cases) can be decreased down to $O(m(\log n_i) \log\log n_i)$. Thus the total running time in this case is

$$O(\frac{m(\log n_i)d(n_i) \log\log n_i}{\log d(n_i)}).$$

($D_2$) Twice the number of the remaining good discriminants is the number of the elliptic curve cardinalities $m_{i,j}$, and this can be approximated by $e(n_i)$. To remove the small factors of them, various factorization methods are used up to a smoothness bound $b(n_i)$. This takes $O(e(n_i)b(n_i) \log n)$ altogether.

($D_3$) The number of the possibilities for the next step is predicted by the $\lambda$ function. Applying the factorization method and given that the curve cardinality is completely factored, the expected gain is the log of the smooth part of the cardinality, $\log(b(n_i))$. Then $\lambda$ remainders have to be tested with Miller-Rabin test. This takes $O(\lambda \log n_i m(\log n_i))$ time. It is not clear yet what is the best tactic of the downrun process, and what probability of success can be obtained. On the one hand, we would like to use minimal time, and on the other hand we would like to obtain as large a gain in the size of the prime as possible; see the discussion of strategies below.

($F_1$) In finding the proof, the first step is the calculation of the Hilbert polynomial. The time can be neglected. (These polynomials are often precomputed.)

($F_2$) The second step is the determination of modular roots of the Hilbert polynomial, which has degree $h(D)$. The degree is expected to stay below $2\sqrt{d(n_i)} \log d(n_i)$ and we may hope $\approx \sqrt{d(n_i)}$. In a simpler version the running time of the step is $O(m(h(D) \log n_i) \log n_i)$; in a more sophisticated version the running time can be $O(m(g(D) \log n_i) \log n_i)$, where $g(D)$ is the number of genera.

($F_3$) The third step is to find an appropriate elliptic curve. The number of the elliptic curves to try is only two, hence this step requires time $O(m(\log n_i) \log n_i)$.

# 5 The tree structure

In every (recursive) call of step (D) of the ECPP algorithm, on input $n_i$, a list of probable primes $n_{i,1}, \ldots, n_{i,t_i}$ is computed, from which the input $n_{i+1}$ for the

next call is to be selected.

This process may be envisaged as choosing a path through a (directed) tree of which the nodes represent probable primes. (Note that strictly speaking the graph is not a tree, as it is possible that different paths lead to the same node!) The root of this tree is $n_0$, the leaves correspond to probable primes that are small enough to be recognized as primes by some direct method. The aim is to find a relatively short path from the root $n_0$ to a leaf $n_k$ as fast as possible; in particular, one would like to avoid computing too many nodes explicitly.

By the latter we mean that we store certain information with the nodes that we compute explicitly: for possible descendants $n_{i,j}$, $j = 1, \ldots, t_i$ of $n_i$, we store some information to base our choice of $n_{i+1}$ on. This includes the value $n_{i,j}$, as well as $s(n_{i,j})$, $d(n_{i,j})$ and $b(n_{i,j})$, (respectively: the smoothness bound for the discriminants, the bound on the size of the discriminants and the smoothnes bound for the curve order), the level $i$ in the tree and a parameter measuring the *suitability*. The choice of $n_{i+1}$ is based on this *suitability* parameter, which is determined during the call of the algorithm with input $n_i$. The value of the *suitability* depends on the strategy that is being used; see below.

The strategies select a certain value based on the information $s(n_{i,j})$, $d(n_{i,j})$, $b(n_{i,j})$, $n_{i,j}$ stored with the nodes, to initialize the *suitability*. Backtracking is unfavorable as, besides the useless work decreasing the level in the tree, the size of the primes in the nodes is likely to increas; thus there is a fixed penalty value $p$ added to the *suitability* of each node when a new level starts up, except for the nodes of the new level, or in the case $n_{i,j}$ is picked up as significant field then the field itself contains the penalty, as the primes in the nodes with lower level are likely bigger.

Currently two strategies have been used, one of which uses the discriminant bound $d(n_{i,j})$ as significant value, the other using the size of the probable primes.

## 5.1 The discriminant size strategy

The first strategy prefers smaller discriminants, and is therefore called the *discriminant size* strategy. Here the idea is that the computations for smaller discriminants will be cheaper, and hence the algorithm will be completed faster.

In this case, the field $d(n_{i,j})$ gives the initial value of *suitability* of the node. $d(n_{i,j})$ is determined based on the function $\lambda$, searching for the minimal power of $d(n_{i,j}) = \log(n_{i,j})^{\delta_{i,j}}$ allowing the value of $\lambda$ to exceed a certain bound given $s(n_{i,j})$, $b(n_{i,j})$. The power $\delta$ is stored as the initial value of *suitability*. Store the nodes in an array, then a certain penalty $p$ is added if necessary. The value of $p$ depends on how hard we want to punish backtrack steps. Order by *suitability* and select the smallest one and call algorithm (T) recursively with that node as input. If (T) is successful the new nodes are added to the array and the sorting process starts again. If there is no new node found the value of *suitability* and the field $d(n_{i,j})$ of the chosen node is increased. The power $d(n_{i,j})$ can reach a given bound where the node falls out from the array of possible nodes. The nodes are reordered. Repeat this procedure, until the size of the nodes reaches a limit which is small enough to recognize the prime.

### 5.2 The size reduction strategy

In the second strategy, larger gain in size reduction is preferred, hence it will be called the *size reduction* strategy. The idea is, of course, that large size reduction will lead to a short path, and hence to faster completion of the algorithm.

The nodes are stored in the same array as previously and the determination of the initial value of $d(n)$ goes in the same way by use of the function $\lambda$, but in this case the significant field of the nodes is $n_{i,j}$. Order the nodes by size of $n_{i,j}$, select the smallest one again and call (T) with this input. If (T) is not successful and there is no new node, first the value of $d(n)$, and after that the value of $b(n)$ will be increased. If, up to a certain limit, this does not help, the node will be deleted from the array of possible nodes, and the second smallest will be selected. Otherwise, if new nodes are found in (T), they are placed in the array and the ordering starts again.

### 5.3 The path finding algorithm

The path finding algorithm (T) in the 'tree' then has three main stages:

($T_0$) Step (D) is being applied for $n_0$ looking for the minimal choice of $d(n_0) = \log(n_0)^\delta$ for which $D$ is successful with a kind of brute force strategy, using a loop in which the value of $d(n_0)$ is incremented until there exists at least one descendant $n_{1,j}$, as the next step requires a non-empty list of $n_{1,j}$'s. This part is called just once at the beginning and is the same in both strategies.

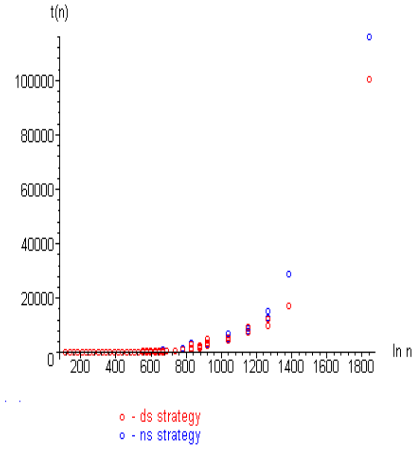
The next steps are repeated for $i = 1, 2, \ldots, k$, where $n_k$ is the first probable prime that can be proven prime directly.

($T_1$) Keeping the value of $\lambda = \lambda_{i_j}$ below 1.5, determine the value of $d(n_{i,j})$ for given $s(n_{i,j})$, $b(n_{i,j})$ for each newly found $n_{i,j}$, and store a list of (at most 100 of) the best ivalues according to suitability.

($T_2$) Sort the list of the best hundred nodes and select the best as $n_{i+1}$. Apply Step (D) again, with the parameter $d(n_{i+1})$; if no new node is found increase the $d(n_{i+1})$ by 0.1 and repeat ordering and selection until at least one new $n_{i+1,j}$ is found. Once Step (D) is successful, go back to ($T_1$) with the new list of nodes as input.

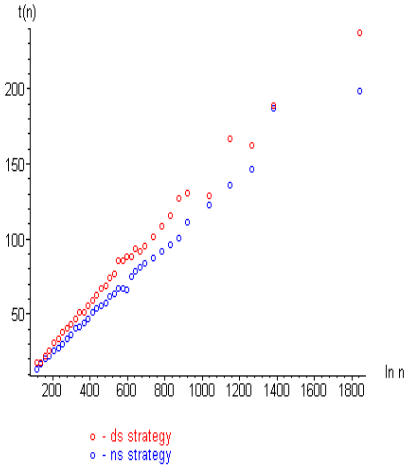
## 6 Experiments

We ran our implementation of these algorithms in Magma [4] for primes $n_0$ ranging from 50 up to 800 decimal digits. The running time and the number of nodes in the tree they produced were checked for algorithms using both strategies The running times are plotted below.

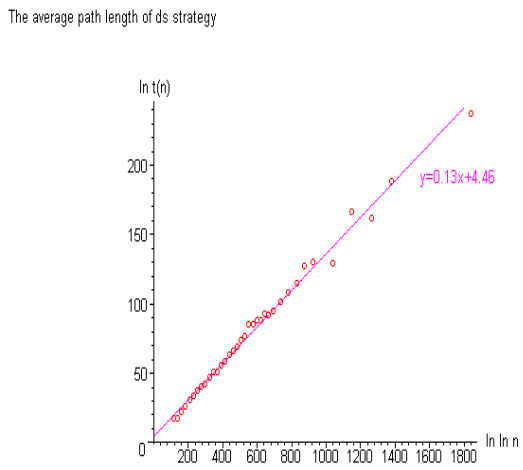



Using the Least Squares method, for the *discriminant size* strategy the best fit was found with $\log {n_0}^{4.362} - 21.835$, and for the *size reduction strategy*, with $\log {n_0}^{4.417} - 22.137$.

We also plotted the total number of nodes.




The level is linear in $\log n_0$ in both cases. In the first case:




One call of Step (D) seems to take about $\log(n_0)^{3+\epsilon}+c$, where $c$ is a constant and $\epsilon > 0$. The most time consuming step is extracting modular square roots.

## 6.1 Interpretation

As we can see in the plots, some numbers require more time than other numbers of the same size. We briefly give two 110 digit examples to show the difficulties that might slow the algorithm down.

**Example 6.1.** Call the algorithm (T) with input $n_0$ of 110 decimal digits, that is one of the points standing out in the plots. The value of the penalty is 0.8 and the initial value of $\sigma_0$ and $\beta_0$ is 1.3 and 1.

In step (0) two new values $n_{0,1}$ and $n_{0,2}$ turn up, for $\delta_0 = 1.9$, both is with 108 digits. Step (1) results in $\delta_{0,1} = 1.2 = \delta_{0,2}$. As the powers are the same both strategies choose the smaller number. The prediction of the function $\lambda$ held, Step (D) found three new nodes for which the prediction starts again and found $\delta_{1,1} = 1.3$, $\delta_{1,2} = 1$, $\delta_{1,3} = 1.1$, so two algorithms differ. The discriminant size strategy chooses $n_{1,2}$, the size reduction selects $n_{1,1}$ for $n_2$, and this way the algorithm (T) can be called again with input the $n_2, \ldots, n_k$ down to the small primes. The running time of the discriminant size strategy is 17.790 seconds and of the size reduction strategy is 20.050 seconds; the length of the path is 39 in the first and 35 in the second case.

**Example 6.2.** Again, the algorithm is called with input $n_0$ of 110 decimal digits, the initial value of $\sigma_0$ and $\beta_0$ being 1.3 and 1. In this case, the in Step $G_0$ two new values $n_{0,1} =$ and $n_{0,2} =$ turn up, with $\delta_0 = 1$; $n_{0,1} =$ has 104 and $n_{0,2} =$ has 108 digits. Step (1) results in $\delta_{0,1} = 1.4$ and $\delta_{0,2} = 1$. The two strategies differ immediately: the discriminant size strategy chooses $n_{0,1}$ and the size reduction strategy selects the other as $n_1$. The prediction of the function $\lambda$ is correct, the $D$ found two new nodes in both cases. With these the steps are repeated again, giving $\delta_{1,1} = 1.4$, $\delta_{1,2} = 1.1$ in the discriminant size strategy, $\delta_{1,1} = 1.4$, $\delta_{1,2} = 1$ in the size reduction strategy. According to the strategy $n_2$ is chosen, and the recursive steps repeated down to the small primes. The running time of the discriminant size strategy is 9.890 seconds and

of the size reduction strategy 14.400 seconds; the length of the path is 41 in the first and 26 in the second case.

We observe that in the first case both algorithms were slower.

The main part of the difference between the running times in the first case is $\delta_0 = 1.9$, in the other case it is 1 in the step (0). Besides increasing $\delta_0$ up to 1.9 we have to make a great effort to find new nodes as the initial value of $\delta_0$ in step (0) is 1 and the brute force technique is applied. This phenomenon can be found in the first example many times during step (1), and although the brute force method is replaced, they are still slowing the algorithm down.

There are three occurrences worth mentioning. The first is in level 20 in the discriminant size strategy, where the expected minimal value of $\delta_{i,j} = 1$ and new nodes are found with $\delta_{i,j} = 1.4$. The other two are during the size reduction strategy: the first is in level 6, where the initial value of $\delta_{i,j} = 1.3$ and Step (D) has to be called five times to find new nodes. The last one can be found in level 14, where we expected $\delta_{i,j} = 1.2$ and the successful value was finally $\delta_{i,j} = 1.7$ In the other occurrences the number of attempts was at most three. The difference between the running times of the two strategies is significant, as in the discriminant size strategy there is just one occurrence, in the size reduction strategy there are two and the size reduction strategy was slower despite the fact that its path was shorter. In general applying the size reduction strategy we will face this situation more pften, as in this case we are forcing one node until a certain limit and give it up just afterwards; the discriminant size strategy allows us to backtrack easier.

Backtracking is when $n_i = n_{j_e}$, $n_{i+1} = n_{l_r}$ and $l \leq j$ in this situation. This can make the algorithm slower because we execute useless operation in the dead end. We can find one backtrack in the second example, applying the discriminant size strategy, at level 15 $n_{15}$ had two new possibilities $n_{15,1} =$ and $n_{15,2} =$. The $n_{15,1}$ was preferred at first, as $\delta_{15,1} = 1.1$, $\delta_{15,2} = 1.2$, but the Step (D) failed. Now the $\delta_{15,1}$ is increased by 0.1; this way the next point chosen is $n_{15,2}$ as they both have the same $\delta = 1.2$ and $n_{15,2}$ is smaller. Step (D) finds a node with input $n_{15,2}$ and continues the recursion.

The main difference between the two strategies is that the discriminant size strategy can backtrack easily; this way it works with low values of $\delta$ if it is possible, but does not put emphasis on the length of the path. Accordingly, in general the path generated in this strategy is longer than the one belonging to the size reduction strategy, but each step takes shorter. The size reduction strategy concentrates on the length of the path, as it always choses the smallest; besides, it tries to find the smallest $\delta$ also, but as it cannot backtrack so easily, the limits can grow higher than in the discriminant size strategy.

# 7 Conclusions

As we saw, the running time of the discriminant size strategy was a little bit smaller, but the path of the size reduction strategy was shorter. Moreover, in the first example we increased the value of $d(n)$ for nothing, which consumes a lot of time in the beginning. To combine the two strategies and avoid useless computations, the next step would be an algorithm, predicting which limit should we increase from $s(n)$, $d(n)$ and $b(n)$ for the best efficiency, that means

the algorithm will be aware of all the three limits and the gain in the length too. Our hope is smoothing away the differences between the running time with input $n_0$-s with the same size, and finding a relatively short path with minimal effort.

# References


[1] M. Agrawal, N. Kayal, N, Saxena, *PRIMES is in P*, Annals Math. **160** (2004), 781–793.

[2] A. O. L. Atkin, F. Morain, *Elliptic Curves And Primality Proving*, Math. Comp. **61** (1993), 29–68.

[3] D. Bernstein, *Proving primality in essentially quartic random time*, Math. Comp. **76** (2007), 389–403.

[4] Wieb Bosma, John Cannon, Catherine Playoust, *The Magma algebra system. I. The user language*, J. Symbolic Comput., **24**(3-4), (1997), 235–265.

[5] A. K. Lenstra, H. W. Lenstra, Jr., *Algorithms in number theory*, in: J. van Leeuwen (ed.), *Handbook of theoretical computer science* **A***: algorithms and complexity*, pp. 673–715, Cambridge (MA): MIT Press, 1991.