Annales Univ. Sci. Budapest., Sect. Comp. 44 (2015) 197–210

PRIMALITY PROOFS WITH ELLIPTIC CURVES: EXPERIMENTAL DATA

Gyöngyvér Kiss (Budapest, Hungary)

Communicated by Antal Járai

(Received October 10, 2015; accepted December 15, 2015)

Abstract. In this paper we present experimental data to confirm the heuristics that allowed us to use refinements to reduce the heuristic running time of the elliptic curve primality proving algorithm to $o(\ln^4 n)$. We described our implementation of the algorithm of Atkin and Morain in an earlier paper [3], where the fastest known algorithms are used for various parts and we applied heuristics and refinements to obtain the improved running time. Here we justify the heuristics with practical data.

1. Introduction

The Elliptic Curve Primality Proving algorithm (ECPP) of Atkin and Morain is a widely prevalent algorithm for primality testing. Starting from a probable prime n_0, it is called recursively on a sequence of probable primes $n_1, \ldots, n_l$ of decreasing size, until n_l is small enough to be easily deterministically verified prime. The original implementation of the algorithm is described in the paper of Atkin and Morain [1], and several modifications were made in order to decrease the heuristic running time of the original implementation $O(\ln^{4+\varepsilon} n)$ (see [7] [8]). In our work [3] it is proven that the heuristic running time can be reduced to $o(\ln^4 n)$ using the fastest possible algorithms under some heuristic assumptions. We ran several experiments in Magma [2] to confirm our statements in practice. In this paper we present some results of these experiments.

Although presenting all of them would be too long for this paper, it is possible to download them from the page `http://www.math.ru.nl/~gykiss`.

Key words and phrases: ECPP, elliptic curves, primality proving.
2010 Mathematics Subject Classification: 11Y11, 11A51, 14H52, 97R20.

2. The ECPP algorithm

As the ECPP algorithm is described in detail in [3], we only give a short outline here.

The input of the algorithm is a large positive odd integer n that has passed some compositeness tests (it is a *probable prime*).

Algorithm 2.0.1. ECPP

(D) *Starting with* $n_0 = n$, *form a sequence of probable primes* $n_0, \dots, n_l$, *such that* n_{i+1} *is a divisor of the order of an elliptic curve* $\mod n_i$, *such that* $n_{i+1} > (\sqrt[4]{n_i}+1)^2$, *and such that* n_l *is small enough to be easily recognized as a prime.*

(F) *For each of the integers* n_i, *with* $i = 0, 1, \dots, l-1$, *construct the curves and points for the proof.*

(V) *Verify that* n_l *is prime, and verify that* n_i *is prime, for* $i = l-1, l-2, \dots, 0$, *by showing that the points have the right order on the curves and that* $n_{i+1} > (\sqrt[4]{n_i}+1)^2$.

In this paper we only deal with (D), so we describe this part of the algorithm in some more detail.

The 'Downrun' part (D) of ECPP will be called recursively with input n_i; the aim of one iteration is to find n_{i+1}. We describe the i-th iteration here.

Algorithm 2.0.2. Downrun

(D) *Select a pair* (D, u) *of negative discriminant* D *and integer* u *such that* $m_i(u) = n_i + 1 - u$, *the order of an elliptic curve* $\mod n_i$, *is the product of small primes and a probable prime* n_{i+1} *that exceeds* $(\sqrt[4]{n_i}+1)^2$. *For simplicity those* $m_i(u)$ *that have this property we call* almost smooth.

This is done as follows:

(0) *Select discriminants* D *suitable for* n_i *from a list.*

(1) *Reduce the binary quadratic form* $n_i x^2 + Bxy + \frac{B^2-D}{4n_i} y^2$, *where* B *is such that* $B^2 \equiv D \mod 4n_i$. *If this provides a second order algebraic integer* $\nu \in Q(\sqrt{D})$ *with* $\nu \cdot \overline{\nu} = n_i$, *then* $u = \nu + \overline{\nu}$ *and the pair* (D, u) *is usable.*

(2) *For all pairs* $(D, \pm u)$ *of the previous step, determine* $n_i + 1 - u$ *and* $n_i + 1 + u$. *Select those which are* almost smooth.

(3) *Select the best possible pair* (D, u) *from this list, and let* n_{i+1} *be the large probable prime factor of* $n_i + 1 - u$

3. Heuristics

The main point of our previous paper [3] was that if we want to decrease the heuristic running time of ECPP to $o(\ln^4 n)$ then, besides the fast algorithms, we need to apply certain refinements. The theoretical background is described in detail in [3], so here we only list the important facts and explanations.

Parameters

First we will describe a few parameters that play a crucial role in our implementation.

There are three main parameters that are used in the algorithm to control the Downrun, introduced in previous papers [1], [7], [8], but used in a different way. We refer to the steps of the description of this algorithm in the previous section.

d: In step (0) we select a set of discriminants D. In order to control the size of this set, we apply an upper bound on the size of the discriminants, which will be denoted by d. In steps (1) and (2) we need to perform a quadratic form reduction for essentially each discriminant that is suitable for the current input, as well as an integer factorization and a primality test for each successful discriminant; thus the number of selected discriminants has a huge impact on the running time, and it is controlled through d.

s: In step (1) we also have to extract the modular square roots of discriminants $D \mod n_i$. That can be done faster if we extract the square roots of the prime divisors of the D-s instead. An upper bound s on the size of the factors of the discriminants controls the size of the set of primes on which we have to perform the square root extraction, and this bound s will also have an effect on the number of the discriminants of course, as we discard discriminants that are not s-smooth.

b: One of the bottlenecks we encounter, is the need to factor the curve orders $m_i(u) = n_i + 1 - u$, as performed in step (2). There are two ways to control the running time of the factorizations. The first one, that was mentioned above, is to control the size of the discriminant set through d and s; but we can also restrict the set of primes that we use to factor the curve orders. Most of the factorization is done using a form of trial division, and we put a bound b on the size of the primes used.

The value of the parameters chosen will depend on the current n_i, and so they ought to be denoted by $d(n_i)$, $s(n_i)$ and $b(n_i)$. By choice the parameters will all be of the form

$$a \ln^{c_1} n_i \ln\ln^{c_2} n_i$$

for certain values of c_1 and c_2 that are not dependent on n_i.

Estimates

After describing the main parameters, we get to the assumptions and estimates that provide the core of our heuristics and refinements.

As mentioned before, we control the discriminant set that is used in the iterations; by $e\big(s(n_i), d(n_i)\big)$ we denote the number of the curve orders that we gain in step (2) after processing a set of $s(n_i)$-smooth discriminants up to $d(n_i)$. The expected value of $e\big(s(n_i), d(n_i)\big)$ is

$$\bar{e}\big(n_i, s(n_i), d(n_i)\big) = \sum_D \frac{2^t}{h(D)},$$

where t is the number of the prime factors of D. For simplicity $\bar{e}$ will denote $\bar{e}\big(n_i, s(n_i), d(n_i)\big)$ if not ambiguous. We expect

$$h(D) = O(\sqrt{D}).$$

The number l of the almost b smooth curve orders is expected to be

$$\lambda\big(s(n_i), d(n_i), b(n_i)\big) = e^{\gamma} \frac{\ln b(n_i)}{\ln n_i} e\big(s(n_i), d(n_i)\big).$$

If not ambiguous $\lambda\big(s(n_i), d(n_i), b(n_i)\big)$ will be denoted by λ.

The size difference between n_i and n_{i+1} is $G(n_i)$. The expected size difference is

$$\bar{G}(n_i) = \ln b(n_i);$$

and we denote it by $\bar{G}$ if it is not ambiguous.

4. Experiments

In Figure 1 and Figure 2 we show the relation between $h(D)$ and $\sqrt{D}$ as found. Figure 2 presents the same graphs as Figure 1 but on a much finer scale. The graphs on the left present the average value of $\sqrt{D}/h(D)$ as a function of D, while the graphs on the right present a cumulative average of the same function. As $h(D) = O(\sqrt{D})$ as we expect the curve is more or less a straight line. It is interesting that the cumulative average has a maximum at $3 \cdot 10^8$ (but this may be due to the implementation of the class number function in Magma).

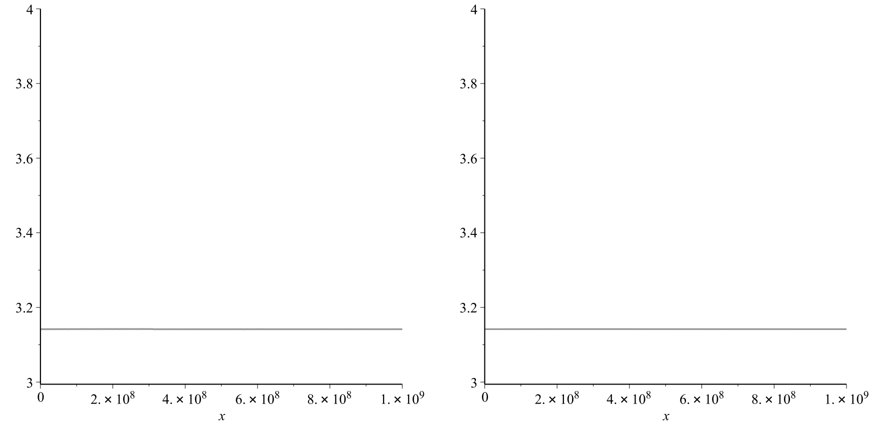

Figure 1. $\sqrt{D}/h(D)$ as a function of D

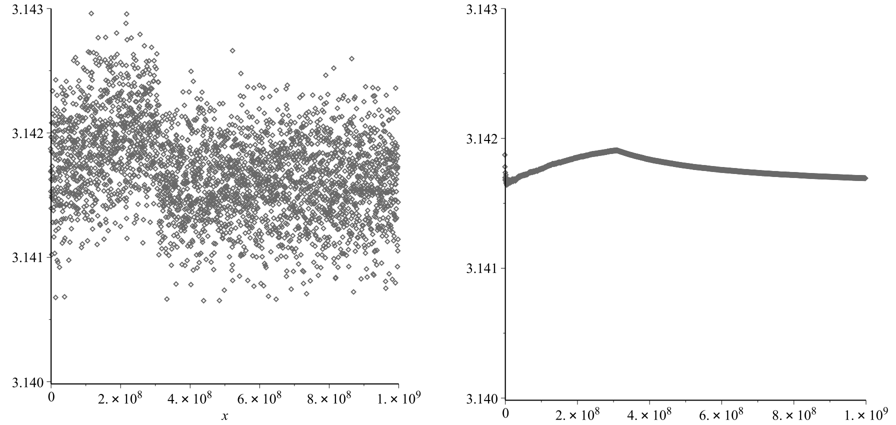

Figure 2. $\sqrt{D}/h(D)$ as a function of D on finer scale

Figure 3 displays the relation between $\bar{e}$ and $e\big(s(n_i), d(n_i)\big)$ for numbers with 3400 digits (around 10000 bits), that is, the number of curve orders found versus the expected number. On the abscissa the $\bar{e}$ values are marked and on the ordinate the actual number of curve orders found in all experiments that we did for numbers up to 3400 digits (with varying values of s and d). We see that all points are close to the identity function, indicating that $\bar{e}$ is a good estimator for e.

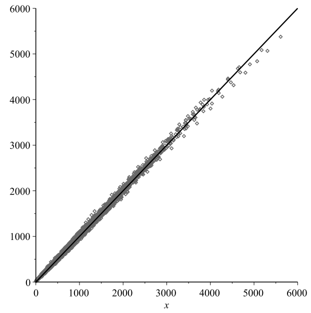

Figure 3. Precision of $\bar{e}$

As was mentioned before, the parameters $d(n_i)$ and $s(n_i)$ have an effect on the value $e\big(s(n_i), d(n_i)\big)$, the number of the curve orders that will be found. In our previous paper [3] we state that if we decrease $d(n_i)$ below $\ln^2 n_i$, the number of the curve orders will drop considerably. On the other hand, for the value of $s(n_i)$ it was stated that it is useful to keep it below $\sqrt{d(n_i)}$, because in practice we do not lose too many curve orders in this case, but the running time of extracting the modular square roots will drop from $O(\ln^3 n \ln\ln\ln n)$ to $o(\ln^3 n)$. We wanted to see this behavior in practice.

In Figure 4 and 5 we can see the relation between $s(n_i) = \ln^\sigma(n_i)$ and $e\big(s(n_i), d(n_i)\big)$. The value of d is fixed, namely $d(n_i) = \ln^2(n_i)$. In each case, on the abscissa there are values for σ in the range $0.5 \ldots 1.1$. The ordinate of Figure 4 gives the values of $e\big(s(n_i), d(n_i)\big)$ that occurred for 3400 digit numbers on the left, and the average value of the $e\big(s(n_i), d(n_i)\big)$-s for the same numbers on the right. On the ordinate of Figure 5 we see the average value of the $e\big(s(n_i), d(n_i)\big)$-s for 500–3400 digit numbers.

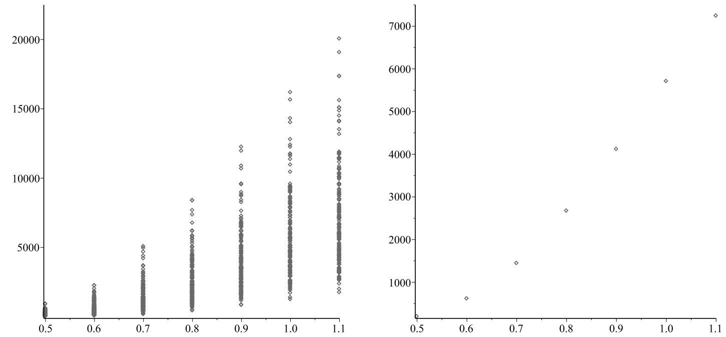

Figure 4. $e\big(s(n_i), d(n_i)\big)$ and its mean as a function of S for 3400 digit numbers

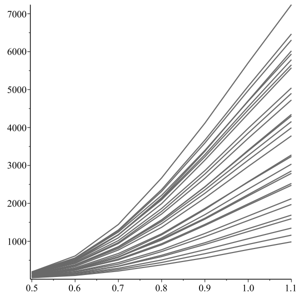

Figure 5. The mean of $e\big(s(n_i), d(n_i)\big)$ as a function of σ for 3400 digit numbers

In Figure 6 and 7 we can see the same type of graphs, only in this case the value of s is fixed to $s(n_i) = \ln(n_i)$, and $d(n_i) = \ln^{\delta}(n_i)$, where δ ranges over 0.5–2.

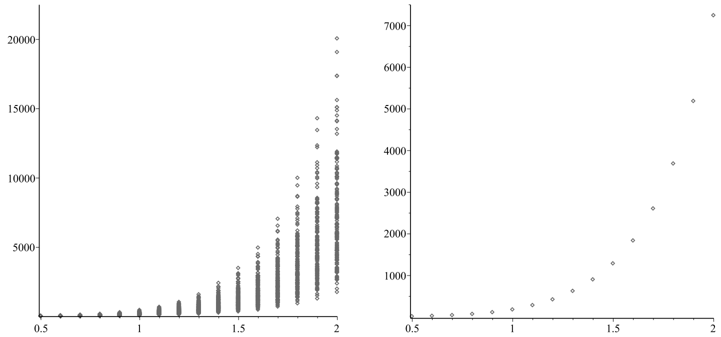

Figure 6. $e\big(s(n_i), d(n_i)\big)$ and its mean as a function of δ for 3400 digit numbers

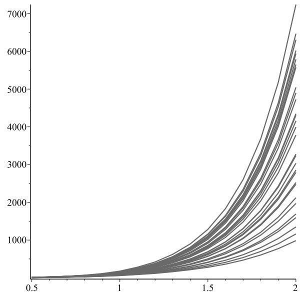

Figure 7. The mean of $e\big(s(n_i), d(n_i)\big)$ as a function of δ for 3400 digit numbers

It is clear that decreasing the value of $d(n_i)$ has a more drastic effect on the value of $e\big(s(n_i), d(n_i)\big)$: if we decrease the value of δ from 2 to 1.9, the value of $e\big(s(n_i), d(n_i)\big)$ drops from slightly more than 7000 to just over 5000, but if we decrease the value of σ from 1 to 0.9 we still have 6500 curve orders.

To compute λ, the expected number of curve orders that will factor in the required way, we need $e(s(n_i), d(n_i))$, as we saw in the previous chapter. It is possible to use the actual $e(s(n_i), d(n_i))$, but for that we need to determine the curve orders. In practice we use $\bar{e}$ to predict this value, and for this we only need the appropriate $s(n_i)$-smooth discriminants up to $d(n_i)$, which requires much less computation; the estimation will be less accurate. We ran experiment for both ways.

Figure 8 and 9 show the actual number l of the almost smooth curve orders, as a function of the value of λ. In both cases we included the identity function to see how much the graphs diverse from it.

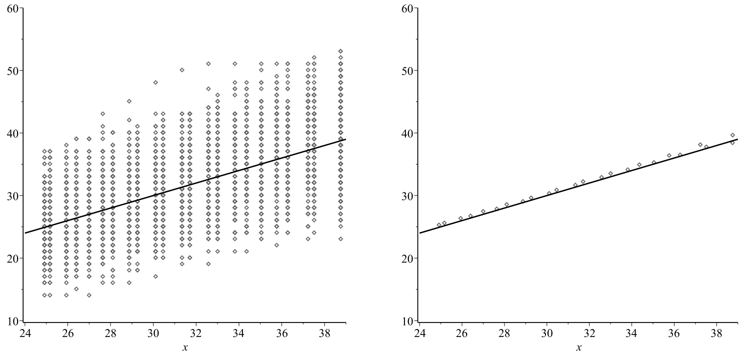

Figure 8. l as a function of λ for 3400 digit numbers using actual values e

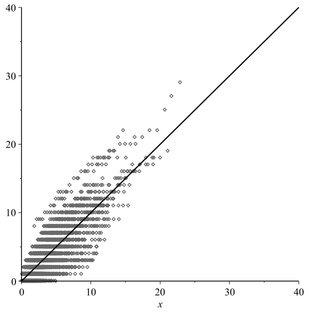

Figure 9. l as a function of λ for 3400 digit numbers using estimated values $\bar{e}$

In the first case we generated e random numbers and factored them with bound b, for several different values of b, and in each situation we determined the value of λ. We use e instead of $e\big(s(n_i), d(n_i)\big)$ as this experiment was carried out with a set of random numbers, not curve orders produced by the algorithm. In this case we have an actual value e. On the abscissa of the first graph of Figure 8 are the different values of λ, the ordinate shows the values l. The abscissa of the second graph is the same as the first, but on the ordinate shows the average values of the l-s.

Figure 9 shows the situation as it is in practice. Instead of using an actual value e, we use the estimation $\bar{e}$ while estimating the value of λ. The abscissa and ordinate are the same as in Figure 8. It does not make to much sense to determine the average of l-s in this case; while in the first case we have the same set of values λ as the e-s are the same for all the experiments and the b-s are increased in the same way, here we only have different values λ.

It is important to determine whether, in order to increase the number l of the almost smooth curve orders, it might be better to increase $b(n_i)$ or $e\big(s(n_i), d(n_i)\big)$. In the picture of Figure 10 and 11 we see the relation between l, b and e and the identity function. Again we have to mention that we are not using $b(n_i)$ or $e(s(n_i), d(n_i))$ here as we are talking about factoring random numbers, not generated from an n_i. Here b is of the form $2^T \cdot s \ln 2$, with $s = 10^6$ and $T = 0, \ldots, 12$.

Figure 10 shows l as a function of T. The left hand graph includes all the values that occurred as l while increasing T, while the right hand graph gives the average values of l on the ordinate. In Figure 11 we present l as a function of e. We have the same situation here as in Figure 10: the left hand graph gives all the values of l, and the second only shows the average value of the l-s.

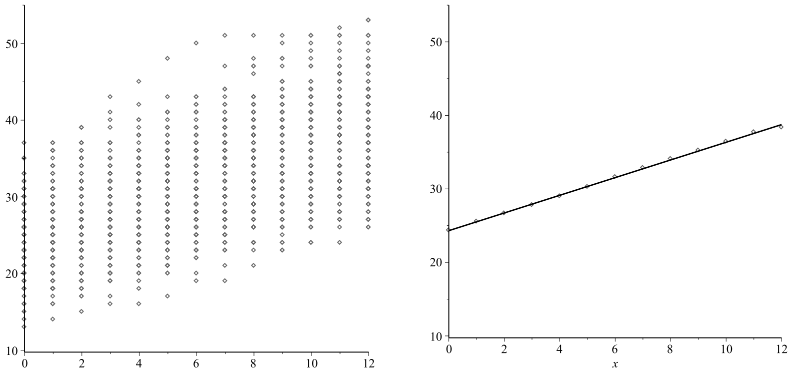

Figure 10. l as a function of T for 3400 digit numbers

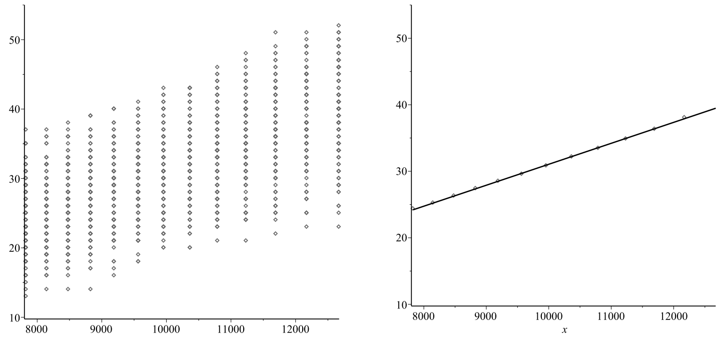

Figure 11. l as a function of e for 3400 digit numbers

We can see that the value of l increases similarly in both cases, but we need a bigger increase if we choose to increase b. If we consider the expected value $\lambda = e^{\gamma} \cdot \bar{e} \cdot \ln b(n)/\ln n$ of l, we see that a 1.04-fold increase of e increases the value of λ also 1.04-fold. But increasing the value of $b = 2^T \cdot s \cdot \ln 2$ to $2^{T+1} \cdot s \cdot \ln 2$ results in

$$\lambda(2 \cdot b) = e^{\gamma} \cdot \bar{e} \cdot \ln(2^{T+1} \cdot s \cdot \ln 2)/\ln n =$$

$$e^{\gamma} \cdot \bar{e} \cdot \big(\ln(2^T \cdot s \cdot \ln 2)/\ln n + \ln 2/\ln n\big) =$$

$$\lambda(b)\big(1 + \ln 2/\ln(2^T \cdot s \cdot \ln 2)\big),$$

thus on average also a 1.04-fold increase in the range of $T = 0, ..., 12$. If we compare the 1^{st} and 13^{th} iterations in both cases, the value of e has to be increased 1.619-fold, but the value of b increased 4096-fold. Of course there are situations in which it can still be worthwhile to increase b, for example if we have a huge amount of curve orders already, or if we have a very efficient factoring algorithm.

Increasing $b(n_i)$ has some pleasant effects on the length of the path from n_i, $I(n_i)$, and also on the gain, $G(n_i)$. We state in our previous paper that $\bar{G}(n_i) = \ln b(n_i)$ and $I(n_i) \asymp \ln n_i/G(n_i)$. Therefore increasing $b(n_i)$ will increase $G(n_i)$ and decrease $I(n_i)$.

The length $I(n_i)$ of the ECPP-path could refer to two different measures: on the one hand we simply have a path from n_i to the small primes, along which we verify the primality of n_i; on the other hand, there is a sequence of probable primes that we used to get to the small primes, including backtracks and repetitions. We will denote the first one by $I_b(n_i)$ and the second by $I_n(n_i)$; one would expect a constant factor between them.

In Figure 12 we see $G(n_i)$ as a function of $\bar{G}(n_i)$ and the identity function. We observe a constant difference between $G(n_i)$ and $\bar{G}(n_i)$, due to the relatively significant difference between the sum and the integral for small primes and

$$\bar{G}(n_i) = \sum_{p \in \mathbb{P}, p \leq b(n_i)} \frac{\ln p}{p} \sim \int_2^{b(n_i)} \frac{1}{x}\, dx = \ln b(n_i) - \ln 2.$$

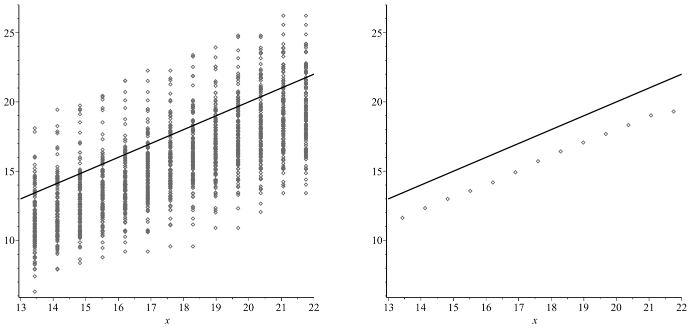

Figure 12. $G(n_i)$ as the function of $\ln b(n_i)$ for 3400 digit numbers

Figure 13 shows $I_b(n_0)$ on the left and $I_n(n_0)$ on the right. We can see that the graph of $I_b(n_0)$ is much steeper. Figure 14 shows the average $I_b(n_0)$ and $I_n(n_0)$ as a function of $\log_{10} n$ together with the identity function. Indeed, the graphs are consistent with a constant factor of around 3.95 between the two functions.

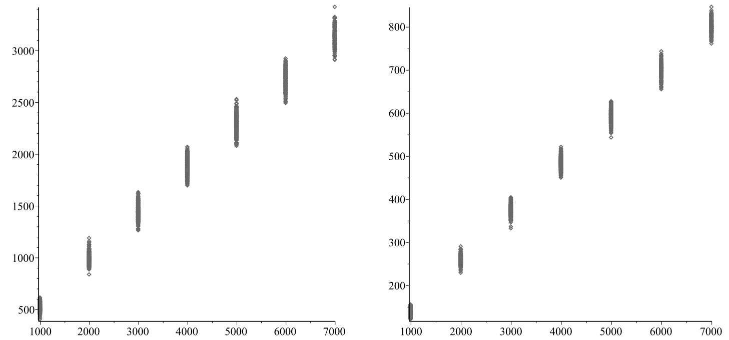

Figure 13. $I_b(n_0)$ and $I_n(n_0)$ as the function of $\log_{10} n_0$ up to 7000 digit numbers

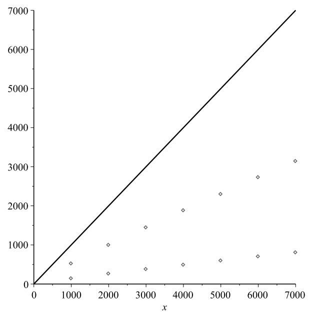

Figure 14. Average $I_b(n_0)$ and $I_n(n_0)$ as the function of $\log_{10} n_0$ up to 7000 digit numbers

5. Conclusion

The aim of this paper is to study the behavior of the heuristics from [3] on a manageable set of experimental data. In the previous chapter we reported on experiments for numbers ranging from 500 digits up to 3400 digits; we summarize the results here. The assumptions underlying the analysis in [3] were the following

(1)
$$h(D) = O(\sqrt{D}).$$

(2)
$$e\bigl(s(n_i), d(n_i)\bigr) \approx \sum_{D} \frac{2^t}{h(D)},$$

where $e\bigl(s(n_i), d(n_i)\bigr)$ is the number of curve orders that we gain after processing all the $s(n_i)$ smooth discriminants D up to $d(n_i)$.

(3) The heuristic running time of extracting the modular square roots will drop from $O(\ln^3 n \ln\ln\ln n)$ to $o(\ln^3 n)$ if we choose a value of $s(n_i)$ that is below $\sqrt{d(n_i)}$ while retaining a reasonable number of discriminants by not decreasing the value of $d(n_i)$ below $\ln^2 n_i$.

(4)
$$\lambda\bigl(s(n_i), d(n_i), b(n_i)\bigr) = e^{\gamma} \frac{\ln b(n_i)}{\ln n_i} e\bigl(n_i, s(n_i), d(n_i)\bigr),$$

where $\lambda\bigl(s(n_i), d(n_i), b(n_i)\bigr)$ is the expected number of curve orders that factor into the required form.

(5)

$$\bar{G}(n_i) = \ln b(n_i),$$

where $G(n_i)$ is the expected size of the difference between n_i and n_{i+1}.

(6)

$$I(n_i) \asymp \frac{\ln n_i}{G(n_i)},$$

where $I(n_i)$ is the length of the path from n_i to the small primes.

For (1) we presented a non cumulative and a cumulative average of $\frac{\sqrt{D}}{h(D)}$ and in the range we looked at it is close to a constant value of approximately 3.14.

For (2) we presented $e\big(s(n_i), d(n_i)\big)$ as a function of $\bar{e}$ and the graph is indeed close to the identity function.

To see (3) we looked at e as a function of $s(n_i)$ and $d(n_i)$ and the latter has a much bigger gradient, which means that if we want to decrease the running time by decreasing the number of discriminants, it is much more effective to decrease $s(n_i)$ below $\sqrt{d(n_i)}$ because the number of discriminants will not drop much, while the running time will drop from $O(\ln^3 n \ln\ln\ln n)$ to $o(\ln^3 n)$.

Regarding (4), we expected the estimation of l to be much more precise if we use the actual $e\big(s(n_i), d(n_i)\big)$ instead of $\bar{e}$, but practice shows ([6]) that it is not worthwhile to compute the curve orders for estimation purposes, as it takes too much time, and working with $\bar{e}$ seems to be appropriate.

The justification of (5) is given by Figure (12), apart from the constant difference between the functions $G(n_i)$ and $b(n_i)$ that was explained above, as the two lines are parallel; the same holds for (6).

The overall conclusion is that the experiments support our assumptions in practice for numbers in the range from 500 to 3400 digits. Running experiments on numbers beyond this range would take too much time; therefore in a real implementation we increment the parameters in much smaller steps. There is an implementation of ECPP within the confines of this project, that is based on the heuristics. Further information can be found in [6].

References

[1] **Atkin, A.O.L. and F. Morain,** Elliptic curves and primality proving, *Math. Comp.* **61(203)** (1993), 29–68.

[2] **Bosma, W., J. Cannon, and C. Playoust,** The Magma algebra system. I. The user language, *J. Symbolic Comput.*, **24(3–4)** (1997), 235–265.

[3] **Bosma, W., E. Cator, A. Járai and Gy. Kiss,** Primality proofs with elliptic curves: heuristics and analysis, *Annales. Univ. Sci. Budapest., Sect. Comp.*, **44** (2015),

[4] **Farkas, G., G. Kallós and Gy. Kiss,** Large primes in generalized pascal triangles, *Acta Univ. Sapientiae, Informatica* **3(2)** (2011), 158–171.

[5] **Járai, A. and Gy. Kiss,** Finding suitable paths for the elliptic curve primality proving algorithm, *Acta Univ. Sapientiae, Informatica* **5(1)** (2013), 35–52.

[6] **Kiss, Gy.,** A strategy for elliptic curve primality proving, *Acta Univ. Sapientiae*, in print.

[7] **Lenstra, A.K. and H.W. Jr. Lenstra,** Algorithms in number theory, in: *Handbook of Theoretical Computer Science, Vol. A. Algorithms and Complexity* (ed. J. van Leeuwen), Elsevier, 1990, 673–716.

[8] **Morain, F.,** Implementing the asymptotically fast version of the elliptic curve primality proving algorithm, *Math. Comp.* **76(257)** (2007), 493–505.

Gy. Kiss
Department of Computer Algebra
Eötvös Loránd University
H-1117 Budapest
Pázmány Péter sétány 1/C
Hungary
kissgyongyver@gmail.com