

ANTS VI: Algorithmic Number Theory Symposium Poster Abstracts

Communicated by Ilias Kotsireas & Emil Volcheck
Wilfrid Laurier University, Canada; National Security Agency, USA
ikotsire@wlu.ca, volcheck@acm.org

The Algorithmic Number Theory Symposium (ANTS) meetings are held biannually since 1994, and have become the premier international forum for the presentation of new research in Computational Number Theory. Previous ANTS conferences have been held as follows: ANTS I, 1994, Cornell University, USA, ANTS II, 1996, University of Bordeaux, France, ANTS III, 1998 Reed College, USA, ANTS IV, 2000, University of Leiden, Netherlands, ANTS V, 2002, University of Sydney, Australia. ANTS VI was held on June 13-18, 2004 at the University of Vermont in the United States, <http://web.ev.usna.edu/~ants>. ANTS VII is scheduled to be held in Berlin in the summer of 2006. Below are six poster abstracts presented during ANTS VI. We thank the organizers for the opportunity to publish the ANTS VI posters. We hope to help bring this work to a wider audience, not only to computational number theorists, but throughout the Symbolic Computation community.

Computing Galois groups by specialisation

Martin Bright¹

Department of Mathematics
University of Liverpool
Liverpool L69 7ZL, England
mjbright@liv.ac.uk

1 Introduction

What is the Galois group of the field extension

$$\mathbb{Q}(\epsilon, \sqrt[4]{a_1}, \sqrt[4]{a_2}, \sqrt[4]{a_3})/\mathbb{Q}, \quad (1)$$

where ϵ is a primitive eighth root of unity and a_i are given non-zero rational numbers?

Kummer theory tells us that, after adjoining a fourth root of unity i to \mathbb{Q} , the rest of the extension is Abelian. Indeed, when the a_i are “sufficiently general” (meaning that they are independent elements of $\mathbb{Q}^\times/(\mathbb{Q}^\times)^4$), the answer is totally straightforward. However, to use a standard number theory package to compute anew the Galois group of this extension for each specific choice of the a_i is extremely inefficient. The object of this note is to demonstrate a simple algorithm, essentially linear algebra, for finding the

¹The author was supported by the Engineering and Physical Sciences Research Council, Grant GR/R82975/01.

Galois group, and its action on the generators of the extension, for arbitrary a_i . We further show how to find the decomposition group at any prime.

We begin by fixing some notation. The map

$$\pi : A = \mathbb{Q}[s_1^\pm, s_2^\pm, s_3^\pm] \rightarrow B = \mathbb{Q}(\epsilon)[t_1^\pm, t_2^\pm, t_3^\pm]$$

defined by $t_i = s_i^4$ is étale. The \mathbb{Q} -valued points of $U = \text{Spec } A$ will correspond to triples (a_1, a_2, a_3) of parameters in (1). The automorphism group G of π is the same as that of the extension of fields of fractions, which by Kummer theory is of order 256, generated by five elements:

$$\begin{aligned} \sigma_1 : t_1 &\mapsto it_1 & \sigma_2 : t_2 &\mapsto it_2 & \sigma_3 : t_3 &\mapsto it_3 \\ \sigma_4 : \sqrt{2} &\mapsto -\sqrt{2} & \tau : i &\mapsto -i \end{aligned} \tag{2}$$

where each generator fixes those of $\{t_1, t_2, t_3, \sqrt{2}, i\}$ not mentioned. The multiplication in G is as follows: all the σ_i commute with each other, and $\tau\sigma_i = \sigma_i^{-1}\tau$.

Let $a = (a_1, a_2, a_3)$ be a \mathbb{Q} -valued point of U . Then $\pi^{-1}a$ consists of finitely many points of $V = \text{Spec } B$, which are permuted by the action of G ; let b be one such point. The *decomposition group* G_b is by definition the stabiliser of b in G ; it is isomorphic in an obvious way to $\text{Gal}(k(b)/k(a))$, which is the Galois group of the extension (1). A different choice of b gives a conjugate decomposition group; this corresponds to a different choice of the various roots in the field extension. The problem is now, given a , to find G_b .

Let X be the following subgroup of B^\times :

$$X = \langle (1+i), \sqrt{2}, t_1, t_2, t_3 \rangle.$$

X has the property that, for any subgroup H of G , the fixed space B^H is generated as an A -module by X^H . We write Y for $X/(X \cap A^\times)$, an Abelian group of order 512. Let μ_4 denote the group of fourth roots of unity in $\bar{\mathbb{Q}}$. The action of G on X gives us a map

$$\phi : G \rightarrow \text{Hom}(Y, \mu_4), \quad g \mapsto (x \mapsto gx/x).$$

Note that ϕ is not a homomorphism, but rather a crossed homomorphism. Now let $\beta : B \rightarrow \bar{\mathbb{Q}}$ denote the evaluation map of the point b ; then β induces maps $Y \rightarrow \bar{\mathbb{Q}}^\times/\mathbb{Q}^\times$ and hence

$$\beta^* : \text{Hom}(\bar{\mathbb{Q}}^\times/\mathbb{Q}^\times, \mu_4) \rightarrow \text{Hom}(Y, \mu_4).$$

Proposition 1. *The decomposition group G_b is described as*

$$G_b = \phi^{-1}(\text{im } \beta^*).$$

2 Implementation

If k is a global field and S a finite set of primes of k including all of the infinite primes, let k_S denote the group of S -units of k .

Definition 2. For any field k , define

$$\sqrt[4]{k} := \{x \in \bar{k}^\times \mid x^4 \in k\}.$$

If k is a global field and S a finite set of primes in k , define

$$\sqrt[4]{k_S} := \{x \in \bar{k}^\times \mid x^4 \in k_S\}.$$

It is clear that the image of $\beta : Y \rightarrow \bar{\mathbb{Q}}^\times / \mathbb{Q}^\times$ lies in $\sqrt[4]{\mathbb{Q}_S} / \mathbb{Q}_S$, where S is the set of primes dividing the a_i together with 2 and ∞ .

The structure of $\sqrt[4]{k} / k^\times$ is straightforward:

Lemma 3. *Let k be a field.*

- *If k contains a primitive fourth root of unity, then $\sqrt[4]{k} / k^\times$ is isomorphic to $k^\times / (k^\times)^4$.*
- *If k does not contain a primitive fourth root of unity, then there is an exact sequence*

$$0 \rightarrow \mu_4 / \mu_2 \rightarrow \sqrt[4]{k} / k^\times \rightarrow k^\times / (k^\times)^4 \rightarrow 0.$$

If k is a global field and S a finite set of primes of k including 2 and all the infinite primes, then the statements hold if we replace k^\times by k_S .

The algorithm for computing the Galois group of the extension (1) is now as follows.

1. Factorise the a_i and let S be the set of primes dividing any of them, together with 2 and ∞ .
2. Construct the finite Abelian groups Y and $\sqrt[4]{\mathbb{Q}_S} / \mathbb{Q}_S$, together with the map β between them defined by $t_i \mapsto \sqrt[4]{a_i}$ (with some arbitrary choice of fourth roots).
3. Apply the functor $\text{Hom}(\cdot, \mu_4)$ to get

$$\text{Hom}(\sqrt[4]{\mathbb{Q}_S} / \mathbb{Q}_S, \mu_4) \xrightarrow{\beta^*} \text{Hom}(Y, \mu_4).$$

4. Compute (generators for) the image of β^* .
5. Compute generators for $\phi^{-1}(\text{im } \beta^*)$.

The algorithm may be extended to compute the decomposition group of the extension (1) at a prime p , as a subgroup of the global Galois group. Note that some care needs to be taken to use the same choice of fourth roots as when computing the global Galois group. We simply replace β by the composite map

$$Y \xrightarrow{\beta} \sqrt[4]{\mathbb{Q}_S} / \mathbb{Q}_S \rightarrow \sqrt[4]{\mathbb{Q}_p} / \mathbb{Q}_p^\times.$$

Here the last group is again finite and described by Lemma 3..

All these steps are easily accomplished using a computer algebra system such as MAGMA [1]. An implementation is available on the author's web site [2].

References

- [1] W. Bosma, J. Cannon, and C. Playoust. The MAGMA algebra system I: The user language. *Journal of Symbolic Computation*, 3/4(24):235–265, 1997.
- [2] M. J. Bright. Implementation of algorithm for computing galois groups by specialisation, 2003. <http://www.boojum.org.uk/maths/quartic-surfaces/galspec.magma>.

Deciding whether the p -torsion group of the \mathbb{Q}_p -rational points of an elliptic curve is non-trivial

Iftikhar Burhanuddin, Ming-Deh Huang

Department of Computer Science
University of Southern California
Los Angeles, CA 90007 USA.
{burhanud, huang}@usc.edu

Abstract

This note describes an algorithm to decide whether an elliptic curve over \mathbb{Q}_p has a non-trivial p -torsion part ($\#E(\mathbb{Q}_p)[p] \neq 1$) under certain assumptions.

We present a polynomial (in $\log p$) time algorithm that decides whether a given elliptic curve over \mathbb{Q}_p has a non-trivial p -torsion part under certain assumptions. We do not attempt a self-contained exposition and in particular the proofs are merely sketches. The background material on elliptic curves can be found in [8]. We borrow freely (but with due acknowledgement and gratitude) from the literature mentioned in the References section. We would like to thank Sheldon Kamienny, Qing Luo and Wayne Raskind for the helpful discussions. The authors were supported in part by the following NSF grants CCR-9820778 and CCR-0306393.

Notation. Let E be an elliptic curve and R a point on it, $x(R)$ and $y(R)$ will denote the x and y coordinates of R respectively. The reduced curve $E \bmod p$ will be denoted by \overline{E} and a point on it by \overline{R} .

In what follows we will assume that $p > 2$, is a prime. $E : y^2 = x^3 + ax + b$, an elliptic curve over \mathbb{Q}_p is said to have good (bad) reduction at p if $v_p(\Delta) = 0$ ($v_p(\Delta) > 0$), where $\Delta = -16(4a^3 + 27b^2)$ is called the discriminant of E and v_p is the discrete valuation of \mathbb{Q}_p [8](Exercise VII.7.1(b)). Let E_0 denote points on E , which reduce to \overline{E}_{ns} , the non-singular points on \overline{E} .

Algorithm 1. Let E be an elliptic curve over \mathbb{Q}_p given by a minimal Weierstrass equation $y^2 = x^3 + ax + b$, where $p > 2$.

Input. We are given the coefficients of E , modulo p^2

Output. TRUE if $\#E_0(\mathbb{Q}_p)[p] = p$ and FALSE if $\#E_0(\mathbb{Q}_p)[p] = 1$

1. $n \leftarrow \#\overline{E}_{ns}(\mathbb{F}_p)$
2. If $\gcd(n, p) = 1$ return FALSE
3. Pick a non-trivial point $\overline{P} \in \overline{E}_{ns}(\mathbb{F}_p)[p]$
4. Lift \overline{P} to $P \in E_0(\mathbb{Q}_p) \setminus E_1(\mathbb{Q}_p)$ using Hensel's lemma [6](Lemma 2.8) such that $P \equiv \overline{P} \pmod{p}$. We only need to determine $x(P)$ modulo p^2
5. Compute $x([p-1]P) \bmod p^2$ using the repeated squaring trick [4](page 23) and the elliptic curve group law formulae [8](Algorithm 2.3)
6. If $x([p-1]P) \equiv x(P) \pmod{p^2}$ return TRUE. Otherwise return FALSE

Theorem 1. *The above algorithm works as desired.*

Proof Note that $E_1(\mathbb{Q}_p) \cong \hat{E}(p\mathbb{Z}_p) \cong \hat{G}_a(p\mathbb{Z}_p)$ is torsion-free [8](Proposition VII.2.2, IV.6.4(b)). Now the short exact sequence $0 \rightarrow E_1(\mathbb{Q}_p) \rightarrow E_0(\mathbb{Q}_p) \rightarrow \overline{E}_{ns}(\mathbb{F}_p) \rightarrow 0$ [8](Proposition VII.2.1) gives rise to the following long exact sequence via the extended snake lemma [5](Lemma II.4.1):

$$0 \rightarrow E_0(\mathbb{Q}_p)[p] \rightarrow \overline{E}_{ns}(\mathbb{F}_p)[p] \xrightarrow{\phi} \hat{G}_a(p\mathbb{Z}_p)/p\hat{G}_a(p\mathbb{Z}_p) \rightarrow E_0(\mathbb{Q}_p)/pE_0(\mathbb{Q}_p) \rightarrow \overline{E}_{ns}(\mathbb{F}_p)/p\overline{E}_{ns}(\mathbb{F}_p) \rightarrow 0$$

If $\gcd(n, p) = 1$ then $\overline{E}_{ns}(\mathbb{F}_p)[p] = 0$ which implies that $E_0(\mathbb{Q}_p)[p] = 0$.

Next we recall the following lemma [1]: Let E be an elliptic curve over \mathbb{Q}_p given by a minimal Weierstrass equation of the form $y^2 = x^3 + ax + b$, where $p > 2$. If $Q \in E_0(\mathbb{Q}_p) \setminus E_1(\mathbb{Q}_p)$ and $\overline{Q} \in \overline{E}_{ns}(\mathbb{F}_p)[p]$ then $x([p-1]Q) - x(Q) \equiv 0 \pmod{p^2} \Leftrightarrow \phi = 0$, where $\phi : \overline{E}_{ns}(\mathbb{F}_p)[p] \rightarrow \hat{G}_a(p\mathbb{Z}_p)/p\hat{G}_a(p\mathbb{Z}_p)$

If $\gcd(n, p) \neq 1$ (due to good reduction or additive reduction) and we pick a point P in $\overline{E}_{ns}(\mathbb{F}_p)[p]$, then appealing to the above lemma tells us that $E_0(\mathbb{Q}_p)[p]$ being non-trivial is equivalent to $x([p-1]P) \equiv x(P) \pmod{p^2}$.

In step 3 we merely need to pick $x_0 \in \mathbb{F}_p$ such that $(\frac{x_0^3+ax_0+b}{p}) = 1$ and computing the Legendre symbol can be done in $O(\log^2 p)$ [2](Algorithm 1.4.12). Hensel lifting can be performed in almost linear time [3](Lemma 2.1). Step 5 would consume $O(\log^{2+\epsilon} p)$ bit operations and therefore the overall time complexity of the algorithm is dominated by the point counting routine [7] which takes time $O(\log^{4+\epsilon} p)$ for all $\epsilon > 0$.

Next we will use the output of the above algorithm to decide whether $E(\mathbb{Q}_p)[p]$ is non-trivial with the help of Corollary 15.2.1 [8](Appendix C) and in order to weed out the spurious cases we impose some conditions.

Theorem 2. *Algorithm 1 correctly computes $\#E(\mathbb{Q}_p)[p]$ provided either*

- *E has good reduction*
- *E has split multiplicative reduction and $\gcd(v_p(\Delta), p) = 1$, or*
- *E has additive reduction and $p > 3$, or*
- *E has additive reduction, $p = 3$ and $\gcd(\#G, 3) = 1$, or*
- *E has non-split multiplicative reduction*

Proof In the case of good reduction $E_0 = E$ and $\overline{E}_{ns} = \overline{E}$ and the theorem follows. On the other hand, the exact sequence $0 \rightarrow E_0(\mathbb{Q}_p) \rightarrow E(\mathbb{Q}_p) \rightarrow G \rightarrow 0$ leads to the following long exact sequence:

$$0 \rightarrow E_0(\mathbb{Q}_p)[p] \rightarrow E(\mathbb{Q}_p)[p] \rightarrow G[p] \rightarrow E_0(\mathbb{Q}_p)/pE_0(\mathbb{Q}_p) \rightarrow E(\mathbb{Q}_p)/pE(\mathbb{Q}_p) \rightarrow G/pG \rightarrow 0$$

Under the assumptions of the theorem $G[p] = 0$ and therefore $E_0(\mathbb{Q}_p)[p] \cong E(\mathbb{Q}_p)[p]$.

The instances which are left out in the above theorem can be further analyzed using the fact that $\frac{\#E(\mathbb{Q}_p)[p]}{\#E_0(\mathbb{Q}_p)[p]} \mid \#G[p]$. Computing $E(\mathbb{Q}_p)[p]$ efficiently seems to be an interesting problem. Finally we would like to point out that the techniques discussed here can be used to elicit information about rational torsion on elliptic curves [1].

References

- [1] Burhanuddin, I., Huang, M.-D. *Computing rational torsion on elliptic curves in linear time*. Preprint.
- [2] Cohen, H. *A Course in Computational Algebraic Number Theory*. GTM 138, Springer-Verlag, 1993.
- [3] Fouquet, M., Gaudry, P., and Harley, R. *An extension of Satoh’s algorithm and its implementation*, J. Ramanujan Math. Soc., 15:281-318, 2000.
- [4] Koblitz, N. *A Course in Number Theory and Cryptography*. GTM 114, Springer-Verlag, 1994.

- [5] Milne, J. S. *Class Field Theory*. Available at <http://jmilne.org/math/>
- [6] Milne, J. S. *Elliptic Curves*. Available at <http://jmilne.org/math/>
- [7] Schoof, R. *Counting points on elliptic curves over finite fields*. Les Dix-huitièmes Journées Arithmétiques (Bordeaux, 1993). J. Théor. Nombres Bordeaux 7 (1995), no. 1, 219–254.
- [8] Silverman, J. H. *The Arithmetic of Elliptic Curves*. GTM 106, Springer-Verlag, 1986.

Finding a Divisible Pair²

Stelian Ciurea³, Erik D. Demaine⁴, Corina E. Pătrașcu⁵, Mihai Pătrașcu⁶

Abstract

Our problem is the natural algorithmic version of a classic mathematical result: any $(n + 1)$ -subset of $\{1, \dots, 2n\}$ contains a pair of divisible numbers. How do we actually find such a pair? If the subset is accessible only through a membership oracle, we show a lower bound of $\frac{4}{3}n - O(1)$ and an almost matching upper bound of $(\frac{4}{3} + \frac{1}{24})n + O(1)$ on the number of queries necessary in the worst case.

1 Introduction

A natural formalization of our problem is in an oracle model: the $(n + 1)$ -subset is not known to the algorithm, and is only accessible through membership queries asked to the oracle. The main results of this paper will be a lower bound of $\frac{4}{3}n - O(1)$ on the number of queries necessary in the worst case, and an almost matching upper bound of $(\frac{4}{3} + \frac{1}{24})n + O(1)$. We believe the upper bound can be improved to match the lower bound, but we were unable to do that.

We begin with the folklore proof of the existence of a divisible pair, which immediately implies an algorithm making $O(n)$ queries.

Theorem 1. For any $S \subset \{1, \dots, 2n\}$, $|S| = n + 1$, there exist $a, b \in S$ such that a divides b .

Proof. For every odd number $q \in \{1, 3, 5, \dots, 2n - 1\}$, let $B_q = \{q \cdot 2^i \mid 0 \leq i \leq \log_2 \lfloor n/q \rfloor\}$ – that is, a bin with all power-of-two multiples of q . Since every number is in the bin generated by its largest odd divisor, we have $B_q \cap B_r = \emptyset$ and $\bigcup B_q = \{1, \dots, 2n\}$. There are exactly n such bins, so at least one bin must contain two elements $a, b \in S$. By construction of the bins, a and b are a divisible pair. \square

This proof is quite strong: it not only tells us that a divisible pair exists, but that one exists in which one number is a power-of-two multiple of the other! We note that this immediately gives an algorithm which makes at most $\frac{3}{2}n + O(1)$ queries. This is because odd numbers greater than n need never be considered. Indeed, these numbers have no multiple below $2n$, so they cannot be part of a pair where the two numbers differ by a power of two factor.

²This material originally appeared as part of the paper “Finding a Divisible Pair and a Good Wooden Fence” by the same authors, published in the Proceedings of the 3rd International Conference on Fun with Algorithms (FUN 2004).

³Universitatea Lucian Blaga, Sibiu, Romania; stelian.ciurea@ulbsibiu.ro

⁴MIT, Computer Science and Artificial Intelligence Laboratory; edemaine@mit.edu

⁵Harvard University, Department of Mathematics; patrascu@fas.harvard.edu

⁶MIT, Computer Science and Artificial Intelligence Laboratory;

2 A Good Lower Bound

In this section, we prove a lower bound of $4n/3 - O(1)$ on the number of queries necessary in the worst case. The strategy of the adversary is simple. The first query to every pair $(x, 2x)$, with $x \in \{2n/3 + 1, \dots, n\}$, receives a positive answer. As long as the adversary has a choice, i.e. it has not already declared $n - 1$ numbers to be out of the set, the second query made to a pair receives a negative answer. Numbers greater than n which are not part of such a pair are always included in the set, so an algorithm which knows the adversary need not ask about these. Numbers below $2n/3$ are *in principle* not included in the set. However, the very last one probed might be included in the set, if the adversary has already declared $n - 1$ values to be out of the set.

The lower bound comes from analyzing the number of queries needed to fix the divisible pair. Once the pair is fixed, an optimal algorithm need not actually query the two numbers in the pair. First note that our adversary never generates a divisible pair unless it has already declared $n - 1$ numbers to be out of the set. Indeed, before this inevitable moment, no number smaller than $2n/3$ is included in the set (so numbers above n which are always included have no divisors in the set), and only one number from each special pair is declared to be in.

Therefore, a divisible pair is not fixed, unless $n - 1$ queries have already received a negative answer. However, at most $2n/3$ of these can come from positions $1, \dots, 2n/3$. Therefore, at least $n/3 - 1$ must come from one of the special pairs. Now remember that the first query touching a special pair always receives a positive answer. So in addition to the $n - 1$ negative answers, the adversary must also have given $n/3 - 1$ positive answers.

Theorem 2. Finding a divisible pair requires at least $\frac{4}{3}n - O(1)$ queries in the worst case.

3 A Good Upper Bound

It can be seen that we cannot hope to beat the $3n/2$ barrier by searching only for power-of-two multiples. However, it might seem that a strategy that probes all multiples of an element can do even worse than $3n/2$, since it foregoes even the basic guarantee of not touching odd numbers greater than n . Surprisingly, a naive algorithm which probes all i 's in increasing order, and for every i in the set probes all its multiples, has relatively good performance. We can prove that this algorithm does at most $(\frac{4}{3} + \frac{1}{12})n + O(1)$ queries, and we can exhibit a family of inputs on which this bound is tight. All the ideas necessary to prove this will also appear in the proofs of this section.

In this section, we analyze a simple improvement to this strategy. The algorithm considers all numbers i in increasing order. Usually, i is probed, and if it is in the set, its multiples are also probed. However, if $i > 3n/2$ and its single multiple $2i$ has previously been probed and received a negative answer, there is no point in querying i . We will show that the performance of this improved algorithm is $(\frac{4}{3} + \frac{1}{24})n + O(1)$, which almost matches our lower bound.

For the purpose of the analysis, we break the execution of this algorithm into two stages. The first stage lasts as long as $i \leq 2n/3$. We begin with the case when the algorithm finishes during the first stage, i.e. with $i \leq 2n/3$. In this case, we show the algorithm does at most $4n/3 + O(1)$ queries. The algorithm can ask at most $n - 1$ queries which receive a negative answer, so we must prove that at most $n/3 + O(1)$ queries can receive a positive answer. At most one query for a number greater than $2n/3$ can receive a positive answer, because such a number is only probed when one of its divisors is in the set, so we stop after the first positive answer. On the other hand, at most $n/3 + 1$ queries from the range $\{1, \dots, 2n/3\}$ can receive a positive answer. Indeed, by fact 1 a $(n/3 + 1)$ -subset of $\{1, \dots, 2n/3\}$ contains a divisible pair, so if we have received $n/3 + 1$ positive answers, we have also found a divisible pair.

Now assume that the algorithm finishes only in the second stage. In this case, all numbers between 1 and $2n/3$ are probed. Let T be the set of such numbers which received a positive answer, and let N

be the set of probed numbers which received a negative answer during the first stage. Given this, exactly $n - 1 - |N|$ numbers are outside the set and not yet discovered. For every $i > 2n/3$, we only probe i if $2i$ has not been probed already. Thus, either i and $2i$ are both in the set and the algorithm stops, or we discover one new number that is outside the set (and possibly also one that is in the set). Thus, the number of queries done in the second stage is at most $2 + 2(n - 1 - |N|) = 2n - 2|N|$. The total number of queries must then be at most $|T| + |N| + 2n - 2|N| = 2n - (|N| - |T|)$. We will show below that $|N| \geq (\frac{2}{3} - \frac{1}{24})n + |T|$, which immediately implies our desired bound.

To prove our bound on $|N|$, first note that N contains exactly $2n/3 - |T|$ numbers from $\{1, \dots, 2n/3\}$. Let $M = N \cap \{2n/3 + 1, \dots, 2n\}$. Our bound is equivalent to $|M| \geq 2|T| - \frac{n}{24}$. First note that M contains all multiples greater than $2n/3$ of numbers from T , and in particular all power-of-two multiples. There are 2 power-of-two multiples for every number in $T \cap \{1, \dots, n/2\}$ and one for the rest of T . However, for every number in T between $n/2 + 1$ and $2n/3$, M must also contain its triple. Thus, we have identified two multiples belonging to M for every number from T . No two numbers from T can have a power-of-two multiple in common, since they would be divisible, and the algorithm would stop during the first stage. Thus, the only double counting can come from triples, namely when $3x = 2^i y$ with some $y \in T, x \in T \cap \{n/2 + 1, \dots, 2n/3\}$. Clearly, $3x \in \{3n/2 + 3, \dots, 2n\}$, so $2^{i-1}y \in \{3n/4 + 1, \dots, n\}$. In addition, $2^i y$ must be a multiple of three, so $2^{i-1}y$ must also be a multiple of three. Finally, $2^{i-1}y$ must be even, because $y \leq 2n/3$, so $i \geq 2$. Thus, $2^{i-1}y$ must be a multiple of 6 in the range $\{3n/4 + 1, \dots, n\}$. Since there are only $n/24$ such possibilities, and each one defines at most one double-counted multiple, we obtain $|M| \geq 2|T| - n/24$, which completes our proof.

Theorem 3. In the worst case, $(\frac{4}{3} + \frac{1}{24})n + O(1)$ queries are sufficient to find a divisible pair.

Computing with overconvergent modular forms

L. J. P. Kilford

California Institute of Technology

ljpk@its.caltech.edu

1 Some results on the slopes of overconvergent modular forms

Let τ be the nontrivial Dirichlet character of conductor 4 and let k be an odd natural number. If p is a prime number, we define the p -slope of a normalised modular eigenform f to be the p -valuation of the eigenvalue of the Hecke operator (either T_p , if p is odd, or U_2) acting on f .

Theorem 1 ([7], Theorem 2). *Let k be an odd integer greater than or equal to 5.*

The slopes of the U_2 operator acting on cusp forms of level $\Gamma_0(4)$, weight k and character τ are the integers

$$2, 4, 6, \dots, k - 3.$$

As a corollary of this, the Fourier coefficients of a normalised eigenform of this level and weight are elements of \mathbf{Q}_2 .

This result about classical modular forms is a consequence of results about a generalization of classical modular forms; the (2-adic) overconvergent modular forms. These are defined by considering how classical modular forms are defined; as sections of a sheaf on all of the modular curve $X_0(N)$. The basic idea behind the definition of overconvergent modular forms is to define the sheaf only on a part of the modular curve (a “not-too supersingular” disc around the cusp ∞). For more details, see [2] and [7].

that the slopes are $1/2, 3/2, 5/2, \dots$). I hope to use methods similar to those of Jacobs to study classical modular forms of weight 1 via quaternionic modular forms.

References

- [1] W. Bosma, J. Cannon, and C. Playoust. The Magma algebra system I: The user language. *J. Symb. Comp.*, 24(3–4):235–265, 1997.
- [2] K. Buzzard and L. J. P. Kilford. The 2-adic eigencurve at the boundary of weight space. Accepted by *Compositio Mathematica*, 2004.
- [3] R. Coleman. p -adic Banach spaces and families of modular forms. *Invent. Math.*, 127:417–479, 1997.
- [4] Robert F. Coleman. Classical and overconvergent modular forms. *Invent. Math.*, 124(1-3):215–241, 1996.
- [5] Fernando Q. Gouvêa. Where the slopes are. *J. Ramanujan Math. Soc.*, 16(1):75–99, 2001.
- [6] D. Jacobs. *Slopes of compact operators*. PhD thesis, Imperial College, University of London, 2002.
- [7] L. J. P. Kilford. Slopes of 2-adic overconvergent modular forms with small level. Submitted to *Math Research Letters*, 2004.
- [8] K. McMurdy. Personal Communication, 2003.

Lehmer’s Algorithm for Very Large Numbers

Jonathan P. Sorenson
 Computer Science and Software Engineering
 Butler University
 Indianapolis, IN 46208 USA
 sorenson@butler.edu
<http://www.butler.edu/~jsorenson>

Computing the greatest common divisor of two integers is a fundamentally important problem in algorithmic and computational number theory and their applications such as cryptography [2, 6, 8].

The fastest known algorithm for this problem is due to Schönhage [9] (see also [1]) which takes $O(n \log^2 n \log \log n)$ time for n -bit inputs. However, this algorithm only has a chance to be practical if the inputs are sufficiently large for FFT-based multiplication to be optimal [10, 6].

In recent years, work has been done to improve the performance of GCD algorithms under the assumption that classical multiplication algorithms are used [11, 4, 15, 7, 12, 5, 4, 13]; the best of these take $O(n^2/\log n)$ time.

For single precision inputs, the binary algorithm [14, 6] or Euclid’s algorithm [2] perform very well, depending on your hardware.

We looked at designing a GCD algorithm under the assumption that the inputs are very large, but not large enough for FFT multiplication to be optimal in practice. We assume Karatsuba complexity multiplication [6]: $O(n^{\log_2 3}) \approx n^{1.585}$ time to multiply n -bit inputs. Our new algorithm is essentially a version of Lehmer’s algorithm where we extract many more leading bits. Instead of taking about $\log n$ bits

as was done in [12], we take $n^{1/(3-\log_2 3)} \approx n^{0.7067}$, resulting in an algorithm with a running time of roughly $n^{1.7067}$.

The main loop of algorithm LehmerLVN is essentially identical to the main loop of the modified Lehmer algorithm from [12].

```

w := machine word size (global constant) ;
function LehmerVLN(u,v) /* u, v > 0 */
  if(u < v) then (u, v) := (v, u); endif;
  k := optimize(u)  $\approx$  (log u)0.7067; /* number of leading words to use */
  while( $\lfloor \log_w v \rfloor > k$ )
    if( $\lfloor \log_w u \rfloor - \lfloor \log_w v \rfloor < k/2$ ) then
      (u, v) := Lehmer1(u,v);
    endif;
    (u, v) := (v, u mod v); /* Euclidean step */
  endwhile;
  return gcd(u, v); /* use any  $O(n^2/\log n)$  time algorithm */

```

This main loop will run $O(n/k)$ times, or about $n^{0.2933}$ times, where n is the number of bits in u .

The Lehmer1 function uses an extended version of Lehmer's algorithm to perform a half-gcd computation (Lehmer's algorithm to do Lehmer's algorithm), for an $O(k^2/\log n) \approx n^{1.4134}$ running time.

Thus, the overall running time is roughly $O(n/k \cdot k^2) = O(nk) = O(n^{1.707})$. In fact, if multiplication takes time $O(n^m)$, the optimal value for k is $n^{1/(3-m)}$ giving a time of $O(n^{(4-m)/(3-m)})$.

We implemented this algorithm and compared it Gnu-MP's built-in GCD algorithm based on the k -ary algorithms [4, 11, 15]. Below are the times we got (in milliseconds) for 100 pseudorandom inputs ranging in size from 1k to 32k 32-bit words in length. We also show the average number of main loop iterations and the value for k for LehmerVLN. Note that powers of 2 are a good choice for k because of the structure of the recursion in Karatsuba's algorithm.

Input sz. (words)	Gnu-MP Time (ms)	Lehmer-LVN		
		Time (ms)	Iters.	k
1k	47	58	131.4	16
2k	178	195	63.0	64
4k	652	599	63.0	128
8k	2367	2024	63.0	256
16k	12637	6270	63.0	512
32k	50583	20764	127.0	512

The computing platform we used was a Dell Pentium IV, 1.0GHz running Red Hat Linux. We used Gnu-MP [3] for multiprecision arithmetic.

References

- [1] A. V. Aho, J. E. Hopcroft, and J. D. Ullman. *The Design and Analysis of Computer Algorithms*. Addison-Wesley, 1974.
- [2] Eric Bach and Jeffrey O. Shallit. *Algorithmic Number Theory*, volume 1. MIT Press, 1996.
- [3] Gnu MP version 4.1.2 online reference manual. <http://swox.com/gmp/manual/index.html>, 2002.

- [4] Tudor Jebelean. A generalization of the binary GCD algorithm. In M. Bronstein, editor, *1993 ACM International Symposium on Symbolic and Algebraic Computation*, pages 111–116, Kiev, Ukraine, 1993. ACM Press.
 - [5] Tudor Jebelean. A double-digit Lehmer-Euclid algorithm for finding the GCD of long integers. *Journal of Symbolic Computation*, 19:145–157, 1995.
 - [6] D. E. Knuth. *The Art of Computer Programming: Seminumerical Algorithms*, volume 2. Addison-Wesley, Reading, Mass., 3rd edition, 1998.
 - [7] D. H. Lehmer. Euclid’s algorithm for large numbers. *American Mathematical Monthly*, 45:227–233, 1938.
 - [8] A. J. Menezes, P. C. van Oorschot, and S. A. Vanstone. *Handbook of Applied Cryptography*. CRC Press, Boca Raton, 1997.
 - [9] A. Schönhage. Schnelle Berechnung von Kettenbruchentwicklungen. *Acta Informatica*, 1:139–144, 1971.
 - [10] A. Schönhage and V. Strassen. Schnelle Multiplikation großer Zahlen. *Computing*, 7:281–292, 1971.
 - [11] Jonathan P. Sorenson. Two fast GCD algorithms. *Journal of Algorithms*, 16:110–144, 1994.
 - [12] Jonathan P. Sorenson. An analysis of Lehmer’s Euclidean GCD algorithm. In A. H. M. Levelt, editor, *1995 ACM International Symposium on Symbolic and Algebraic Computation*, pages 254–258, Montreal, Canada, July 1995. ACM Press.
 - [13] Jonathan P. Sorenson. An analysis of the generalized binary GCD algorithm. In A. Stein and A. Van Der Poorten, editors, *Proceedings of the Conference in Number Theory in Honour of H. C. Williams*. The Fields Institute and the American Mathematical Society, 2004.
 - [14] J. Stein. Computational problems associated with Racah algebra. *Journal of Computational Physics*, 1:397–405, 1967.
 - [15] Ken Weber. The accelerated integer GCD algorithm. *ACM Transactions on Mathematical Software*, 21(1):111–122, 1995.
-

Using the Faugère F_4 algorithm in MAGMA for Gröbner Basis Problems including Polynomial-based Cryptography

Allan Steel

School of Mathematics and Statistics F07
University of Sydney, NSW 2006, Australia
allan@maths.usyd.edu.au

1 Overview

1.1 Introduction

This paper presents timings of various standard benchmarks for the computation of Gröbner bases.

MAGMA V2.11 (first released in May 2004) [1] has a new optimized implementation of the F_4 algorithm of Jean-Charles Faugère [4], which uses linear algebra to compute Gröbner bases, and is generally much faster than the well-established Buchberger algorithm [2, Chap. 2, §7].

The MAGMA implementation performs very well, both over finite fields and over the rationals, as the following timings show. Further special optimizations have been introduced to handle polynomial systems over $\text{GF}(2)$ which arise in polynomial-based cryptography.

Comparison is made with other publically available computer algebra systems for computing Gröbner bases. The other systems use the Buchberger algorithm, and, as seen below, the state-of-the-art for this algorithm has hitherto been represented by SINGULAR over small-prime finite fields and by MAGMA over the rational field. But the new timings show that the F_4 algorithm can be implemented very effectively and is much better in general.

See the webpage [7] for more details on the benchmarks and up-to-date information.

1.2 Machine and Software Versions

For an accurate comparison, all timings are performed on an AMD Athlon XP 2800+ PC (2.087GHz), with cache sizes L1 64K, D 64K, L2 512K, 1.5GB main memory and running Redhat Linux 9.0 (kernel version 2.4.20-8). The only exceptions are the Cyclic 9 system over the rationals and the HFE Challenge 1 system (noted below). The software used is:

- MAGMA [1], version V2.11-8, which uses the new F_4 algorithm.
- MAGMA, version V2.10, which uses the Buchberger algorithm.
- SINGULAR, version 2-0-5, which uses the Buchberger algorithm (downloaded April 2004).
- MACAULAY 2, version 0.9.2, which uses the Buchberger algorithm (downloaded April 2004).

All Gröbner bases are computed with respect to the *grevlex* (graded reverse lexicographical) order, and each software package is called with default options. MAGMA computes the unique sorted reduced Gröbner basis for the term order, but the other systems seem not to do this by default. However, this is done very quickly by MAGMA, so we can ignore the extra bit that MAGMA has to do.

Timings are in seconds unless indicated otherwise. $>T$ means that the computation was stopped after time T (and it was unclear how long there was to go), while $\gg T$ means that the computation was stopped after time T when there was clearly a very long way to go (because the queue of remaining pairs was very large and still growing).

2 Modular (small prime finite field)

All systems are over the prime finite field GF(32003). MAGMA actually handles prime finite fields up to GF(11863279) in the same time, but we restrict to this smaller prime for the other systems which handle small primes only up to 16 bits. SINGULAR clearly has a highly optimized Buchberger algorithm over small finite fields but the new MAGMA 2.11 F_4 algorithm generally beats it easily.

	Magma 2.11	Magma 2.10	Singular 2-0-5	Macaulay 2 0.9.2
Katsura 8	0.3	9.3	2.4	11.8
Katsura 9	1.7	100.9	22.5	123.9
Katsura 10	13.1	1259.3	186.2	1347.1
Cyclic 7 (Hom)	0.4	13.7	3.7	12.3
Cyclic 8 (Hom)	10.4	628.4	132.8	643.9
Cyclic 9	1453.6	$\gg 3d$	38298.8	?

The Katsura n system arises from problems involving magnetism in physics, while the Cyclic n -th roots system is a standard benchmark problem (which has n variables and n polynomials which are invariant under cyclic shifts).

3 Rational Field

All systems are over the rational field.

	Magma 2.11	Magma 2.10	Singular 2-0-5	Macaulay 0.9.2
Katsura 8	2.2	26.7	29.6	550.5
Katsura 9	12.2	291.2	800.0	6884.0
Katsura 10	130.7	4603.9	22972.2	?
Cyclic 7 (Hom)	2.2	51.1	5174.6	1649.5
Cyclic 8 (Hom)	83.7	4320.8	$\gg 3d$	$> 2d$
Cyclic 9	4.6d*			

* Computed in 4.6 days on a 750MHz Sunfire v880 (using 11GB of memory). As far I am aware, no-one else has successfully computed this Gröbner basis except for J.-C. Faugère in 1999 [4]: he took 18 days (sequential time) in 1999 on three 200MHz x86 processors and one 400MHz Alpha processor. So the MAGMA timing is comparable (one 750MHz processor).

4 GF(2) and HFE Cryptosystems

4.1 Smaller systems

All systems are over GF(2). First we have the standard cyclic-roots examples reduced modulo 2 and then several Hidden Field Equation (HFE) systems. These polynomial systems arise from algebraic attacks on the HFE cryptosystem [6] and are denoted by HFE n_d , where n is the number of variables and d is the degree of the underlying secret polynomial.

	Magma 2.11	Magma 2.10	Singular 2-0-5	Macaulay 0.9.2
Cyclic 8	1.0	141.8	10.3	125.8
Cyclic 9	222.6	?	15673.3	?
HFE 30 ₉	0.2	1255.0	25.8	2094.2
HFE 25 ₉₆	6.6	64370.0	49290.3	
HFE 30 ₉₆	28.9	> 1d	>> 10h*	
HFE 35 ₉₆	94.2			
HFE 40 ₉₆	230.5			
HFE 45 ₉₆	530.8			

* Killed after about 10 hours because it was swapping very badly and still growing with very many pairs to go.

4.2 HFE Challenge 1 Solved

Finally, I have solved the first HFE challenge of J. Patarin using the F_4 algorithm in MAGMA 2.11. This involves solving a system of 80 quadratic equations in 80 variables over $\text{GF}(2)$. The input system can be downloaded from [3].

MAGMA solves the challenge in 25.4 hours on a 750MHz Sunfire v880 (using 15GB of memory). There are 4 solutions to the original equations, and the bulk of the computation involves solving three fairly dense linear systems over $\text{GF}(2)$, having respective dimensions 244097×1666981 , 243493×1666981 and 293287×1666981 .

The challenge was first solved by J.-C. Faugère using his $F_{5/2}$ algorithm in May 2002 in 52.2 hours on a 1GHz Alpha EV68 [5]. Thus, allowing for processor differences, the MAGMA time is roughly 2.7 times faster.

5 Acknowledgments

I thank Geoff Bailey, Dan Bernstein and Alyson Reeves for several useful ideas and I thank Jean-Charles Faugère and Antoine Joux for providing me with HFE input systems.

References

- [1] Bosma, W., Cannon, J., Playoust, C. (1997). The Magma algebra system I: The user language, *J. Symb. Comput.*, **24**, 235–265. URL: <http://magma.maths.usyd.edu.au/>.
- [2] Cox, D., Little, J., O’Shea D., *Ideals, Varieties and Algorithms*, Springer, UTM, 1996.
- [3] Courtois, N. *The HFE public key encryption and signature*, URL: <http://www.minrank.org/hfe>.
- [4] Faugère, J.-C., A New Efficient Algorithm for Computing Gröbner Bases (F_4), *Journal of Pure and Applied Algebra*, 1999, **139** (1–3), 61–88.
- [5] Faugère, J.-C., Joux, A., Algebraic Cryptanalysis of Hidden Field Equation (HFE) Cryptosystems using Gröbner Bases. In CRYPTO 2003, 44–60, 2003.
- [6] Patarin, J., Hidden fields equations (HFE) and isomorphisms of polynomials (IP): Two new families of asymmetric algorithms. In *EUROCRYPT’96*, Springer, LCNS 1070, 33–48, 1996.
- [7] Steel, A., *Allan Steel’s Gröbner Basis Timings Page*, URL: <http://magma.maths.usyd.edu.au/users/allan/gb>.