Isomorphisms of Algebraic Number Fields

Mark van Hoeij and Vivek Pal[*]

Abstract

Let $\mathbb{Q}(\alpha)$ and $\mathbb{Q}(\beta)$ be algebraic number fields. We describe a new method to find (if they exist) all isomorphisms, $\mathbb{Q}(\beta) \to \mathbb{Q}(\alpha)$. The algorithm is particularly efficient if the number of isomorphisms is one.

1 Introduction

Let $\mathbb{Q}(\alpha)$ and $\mathbb{Q}(\beta)$ be two number fields, given by the minimal polynomials $f(x) = \sum_{i=0}^{n} f_i x^i$ and $g(x) = \sum_{i=0}^{n} g_i x^i$ of α and β respectively. In this paper we give an algorithm to compute the isomorphisms $\mathbb{Q}(\beta) \to \mathbb{Q}(\alpha)$. Suppose there is an isomorphism then we have the following diagram of field extensions:

$$\begin{tikzcd}
\mathbb{Q}(\beta) \arrow[r, "\cong"] \arrow[d, dash, "g(x)"] & \mathbb{Q}(\alpha) \arrow[d, dash, "f(x)"] \\
\mathbb{Q} & \mathbb{Q}
\end{tikzcd}$$

To represent such an isomorphism we need to give the image of β in $\mathbb{Q}(\alpha)$, in other words, we need to give a root of $g(x)$ in $\mathbb{Q}(\alpha)$.

We now describe two common methods of computing isomorphisms of number fields.

Method I. Field Isomorphism Using Polynomial Factorization [11, Algorithm 4.5.6]

- Find all roots of g in $\mathbb{Q}(\alpha)$. Each corresponds to an isomorphism $\mathbb{Q}(\beta) \to \mathbb{Q}(\alpha)$. The roots can be found by factoring g over $\mathbb{Q}(\alpha)$.
  (a) If done with Trager's method, one ends up factoring a polynomial in $\mathbb{Q}[x]$ of degree n^2.

[*]Florida State University

(b) An alternative is Belabas' algorithm for factoring in $\mathbb{Q}(\alpha)$.

Method II. Field Isomorphism Using Linear Algebra [11, Algorithm 4.5.1/4.5.5]

(a) Let $\alpha_1, \ldots, \alpha_d$ be the roots of f in $\mathbb{Q}_p$ (choose p with $d > 0$).

(b) Let $\beta_1, \ldots, \beta_d$ be the roots of g in $\mathbb{Q}_p$.

(c) If $\beta \mapsto h(\alpha)$ is an isomorphism, then $h(\alpha_1) = \beta_i$ for some $i \in \{1, \ldots, d\}$.

(d) Run a loop $i = 1, \ldots, d$ and for each i, use LLL[9] techniques to check if there exists a polynomial $h(x) \in \mathbb{Q}[x]_{<n}$ for which $h(\alpha_1) = \beta_i$.

Method I is often fast, but one can give examples where it becomes slow, e.g. for so-called Swinnerton-Dyer polynomials, the degree n^2 factoring leads to a lattice reduction in [VH 2002] of dimension approximately $n^2/2$. Method I(b) can be faster, but one can still produce examples where it becomes slow, e.g. [7]. For such examples method II is faster because the lattice reduction there has dimension approximately n.

Our algorithm is similar to Method II. There can be n distinct isomorphisms (in the Galois case) and in this case our algorithm is the same as Algorithm II. However, if there is only one isomorphism then we can save roughly a factor d. This is because we can do the LLL computation for all β_i simultaneously.

2 Preliminaries

We let $\mathbb{Q}[x]_{<n}$ denote the polynomials over $\mathbb{Q}$ with degree less than n.

Definition 2.1. If $h(\alpha) \in \mathbb{Q}(\alpha)$ then the notation $h(x)$ is the element of $\mathbb{Q}[x]_{<n}$ that corresponds to $h(\alpha)$ under $x \mapsto \alpha$.

Under an isomorphism β will map to some $h(\alpha) \in \mathbb{Q}(\alpha)$,

$$\beta \mapsto h(\alpha) = \sum_{i=0}^{n-1} h_i \alpha^i \tag{1}$$

.

A polynomial $h(x) \in \mathbb{Q}[x]_{<n}$ represents an isomorphism if and only if $h(\alpha)$ is a root of g, i.e. $g(h(\alpha)) = 0$.

If $\mathbb{Q}(\beta)$ is isomorphic to $\mathbb{Q}(\alpha)$ then g and f have the same factorization pattern in $\mathbb{Q}_p[x]$ for every prime p. To simplify the factoring in $\mathbb{Q}_p[x]$ we restrict to good primes p, defined as:

Definition 2.2. A good prime p is one that does not divide the leading coefficient of f or g and does not divide the discriminant of either f or g.

Remark 2.3. Both f and g are taken to be in $\mathbb{Z}[x]$.

For a good prime p we can factor f in $\mathbb{Q}_p[x]$ up to any desired p-adic precision by factoring in $\mathbb{F}_p[x]$, followed by Hensel Lifting [11, p. 137]. Likewise we can distinct-degree factor f as:

$$f = F_1 F_2 \dots F_m \text{ in } \mathbb{Q}_p[x] \tag{2}$$

where F_d is the product of all irreducible factors of f in $\mathbb{Q}_p[x]$ of degree d [11, Section 3.4.3].

Definition 2.4. Sub-traces Let p be a prime and d a positive integer. Then we define the $\mathbb{Q}$-linear map:

$$Tr_p^d(f, *) : \mathbb{Q}(\alpha) \to \mathbb{Q}_p$$

as follows. Let $h(x) \in \mathbb{Q}[x]_{<n}$, $h(\alpha) \in \mathbb{Q}(\alpha)$ and F_d as above, then:

$$Tr_p^d(f, h(\alpha)) := \sum_{\substack{\gamma \in \overline{\mathbb{Q}}_p \\ F_d(\gamma)=0}} h(\gamma)$$

We call these maps *sub-traces* because the sum is taken over a subset of the roots of f.
Likewise we define $Tr_p^d(g, *) : \mathbb{Q}(\beta) \to \mathbb{Q}_p$.

Remark 2.5. The map Tr_p^d does not depend on the choice of the minimal polynomial f that is used to represent the number field. In particular if $\beta \mapsto h(\alpha)$ is an isomorphism $\mathbb{Q}(\beta) \to \mathbb{Q}(\alpha)$ then

$$Tr_p^d(g, \beta) = Tr_p^d(f, h(\alpha)) \text{ for every } p, d$$

Definition 2.6. We will now define two bases of $\mathbb{Q}(\alpha)$ that we will need. The first one is the standard basis, which is $\{1, \alpha, \alpha^2, \dots, \alpha^{n-1}\}$. The second will be called the *rational representation basis*, which is $\{1/f'(\alpha), \alpha/f'(\alpha), \dots, \alpha^{n-1}/f'(\alpha)\}$.

Rational representation can improve running time and complexity results, see [4]. This representation has also been used under various names, see [2, 4], and occurs naturally in algebraic number theory as a dual basis under the trace operator, see [2].

A basis for $\mathbb{Q}(\alpha)$ corresponds to a map $\rho : \mathbb{Q}^n \to \mathbb{Q}(\alpha)$. We use the rational representation basis, therefore

$$\rho : (a_0, a_1, \ldots, a_{n-1}) \mapsto \frac{1}{f'(\alpha)} \sum_{i=0}^{n-1} a_i \alpha^i.$$

Definition 2.7. The inverse linear map $h(\alpha) \mapsto \vec{h}$, from $\mathbb{Q}(\alpha)$ to $\mathbb{Q}^n$ is as follows. Let $h(\alpha) = \sum_{i=0}^{n-1} a_i \alpha^i \in \mathbb{Q}(\alpha)$ and write $f'(\alpha) \cdot h(\alpha)$ as $\sum_{i=0}^{n-1} b_i \alpha^i$. Then define $\vec{h} := (b_0, b_1, \ldots, b_{n-1}) \in \mathbb{Q}^n$.

Remark 2.8. One of the advantages of rational representation is: by using the b_i in $\vec{h}$ instead of the a_i, we have $\vec{h} \in \mathbb{Z}^n$ for every algebraic integer $h(\alpha)$, see lemma 4.2. Moreover, as in [4] this also improves bounds (section 4). It is also better to use $g_n \vec{h}$ than simply using $h(\alpha)$ since $g_n \vec{h}$ will have integer components, by Corollary 4.3, which are easier to bound and are heuristically of smaller size [4, Section 6].

For a polynomial $f(x) = \sum_{i=0}^{n} f_i x^i$ denote

$$\|f(x)\| := \left(\sum_{i=0}^{n} |f_i|^2 \right)^{1/2}.$$

Let $M(f)$ be the Mahler measure of f,

$$M(f) := f_n \cdot \prod_{\substack{f(\gamma)=0 \\ \gamma \in \mathbb{C}}} \max\{1, |\gamma|\}.$$

3 Overview of the Algorithm

Goal: To find all $g_n \vec{h} \in \mathbb{Z}^n$ for which $\beta \mapsto h(\alpha)$ defines an isomorphism $\mathbb{Q}(\beta) \to \mathbb{Q}(\alpha)$.
Idea: The aim of the pre-processing algorithm in Section 5 is to find a sequence

$$\mathbb{Z}^n = L_0 \supseteq L_1 \supseteq L_2 \supseteq \cdots \supseteq L_k$$

such that all $g_n \vec{h}$ are in each L_i. We can then use L_k to speed up the computation of the isomorphism(s), especially when $\dim(L_k)$ is small. The cost of computing L_k is comparable to one iteration in Method II.

In the algorithm we start with the lattice $\mathbb{Z}^n$ and then add the restrictions imposed by the condition that under an isomorphism, sub-traces

$\mathbb{Q}(\alpha) \to \mathbb{Q}_p$ must correspond to sub-traces $\mathbb{Q}(\beta) \to \mathbb{Q}_p$. By doing this for several primes we are able to narrow down the possible isomorphisms. If $\dim(L_k) \leq 1$, this directly gives the isomorphism or shows that there is no isomorphism. If $\dim(L_k) > 1$ then we switch to Method II, but starting with L_k. Thus we end up with d lattice reductions of dimension $\dim(L_k)$. In the worst case $\dim(L_k) \approx n$, this costs the same as Method II. In the best case, $\dim(L_k) \leq 1$ and we save a factor d.

4 Bounding the length of $g_n\vec{h}$

To effectively carry out this algorithm we will need a good upper bound on the size of $g_n\vec{h}$. In this section we aim to find such a bound.

Definition 4.1. Let $\alpha_1, \ldots, \alpha_n \in \mathbb{C}$ be the roots of f. Then using the basis $\{1, x, x^2, \ldots, x^{n-1}\}$ of $\mathbb{C}[x]_{<n}$ and the standard basis $\{e_1, e_2, \ldots, e_n\}$ for $\mathbb{C}^n$, the interpolation map $\mathbb{C}^n \to \mathbb{C}[x]_{<n}$ is given by:

$$e_i \mapsto \frac{f(x)/(x-\alpha_i)}{f'(\alpha_i)}$$

This polynomial takes value 1 at $x = \alpha_i$ and value 0 at $x = \alpha_j$ $(i \neq j)$. The inverse of the interpolation map is the evaluation map, which is given by the Vandermonde matrix:

$$\begin{bmatrix} 1 & \alpha_1 & \alpha_1^2 & \ldots & \alpha_1^{n-1} \\ 1 & \alpha_2 & \alpha_2^2 & \ldots & \alpha_2^{n-1} \\ 1 & \alpha_3 & \alpha_3^2 & \ldots & \alpha_3^{n-1} \\ \vdots & \vdots & \ddots & \ldots & \vdots \\ 1 & \alpha_n & \alpha_n^2 & \ldots & \alpha_n^{n-1} \end{bmatrix}$$

Lemma 4.2. *If $a \in \mathbb{Q}(\alpha)$ is an algebraic integer and $f(x)$ is the minimal polynomial for α, then $f'(\alpha) \cdot a \in \mathbb{Z}[\alpha]$.*

Proof. Denote by ${}^{(i)}$ the complex embeddings of $\mathbb{Q}(\alpha)$. Then define

$$m(x) := \sum_{i=1}^{n} a^{(i)} \frac{f(x)}{x - \alpha^{(i)}}.$$

The coefficients of $m(x)$ are in $\mathbb{Q}$ since the polynomial is symmetric in the $\alpha^{(i)}$. But $m(x)$ is also a sum of polynomials all of whose entries are algebraic integers. Hence $m(x) \in \mathbb{Z}[x]$. Note that for $\alpha = \alpha^{(1)}$ we get $m(\alpha) = af'(\alpha) \in \mathbb{Z}[\alpha]$. $\square$

Corollary 4.3. *Let $\beta \mapsto h(\alpha)$ be an isomorphism of $\mathbb{Q}(\beta)$ and $\mathbb{Q}(\alpha)$. Then $g_n h(\alpha)$ is an algebraic integer and hence $g_n \vec{h} \in \mathbb{Z}^n$.*

Proof. Apply lemma 4.2 by letting $a = g_n h(\alpha)$ and recall that $g_n \vec{h}$ is comprised of the coefficients of $g_n f'(\alpha)h(\alpha)$ in the standard basis, each of which will be integers by lemma 4.2. □

Lemma 4.4. *Let $P(x) = \sum_{i=1}^{n} \beta_i \frac{f(x)}{x-\alpha_i} \in \mathbb{Q}[x]_{<n}$, then $P(\alpha) = f'(\alpha)h(\alpha)$.*

Proof. If we evaluate $f'(x)h(x)$ at the roots of $f(x)$ and then interpolate we get:

$$\sum_{i=1}^{n} \beta_i f'(\alpha_i) \frac{f(x)/(x-\alpha_i)}{f'(\alpha_i)} = \sum_{i=1}^{n} \beta_i \frac{f(x)}{x-\alpha_i}.$$

Therefore $\sum_{i=1}^{n} \beta_i \frac{f(x)}{x-\alpha_i}$ will be the remainder of $f'(x)h(x)$ divided by $f(x)$, because they are of the same degree and coincide on the n roots of $f(x)$. The lemma then follows from the fact that α is a root of $f(x)$. □

In order to bound $f'(\alpha)h(x)$ we will have to bound both $\frac{f(x)}{x-\alpha_i}$ and also $|\beta_i|$. We will use Corollary 4.7 to bound $\frac{f(x)}{x-\alpha_i}$ and since we know the β_i up to a permutation (they are roots of $g(x)$), we can bound $\sum |\beta_i|$.

Theorem 4.5. *If $f(x)$ and $\tilde{f}(x)$ are polynomials with complex coefficients, of degree n and d respectively, such that $\tilde{f}(x)$ divides $f(x)$ and $|f(0)| = |\tilde{f}(0)| \neq 0$, then*

$$\|\tilde{f}(x)\| \leq \left(\sum_{j=0}^{n-d} \binom{d}{j}^2 \right)^{1/2} \|f(x)\|. \qquad (3)$$

Proof. See Granville, [1]. □

Corollary 4.6. *If $f(x)$ and $\tilde{f}(x)$ have the same leading coefficient and $f(0), \tilde{f}(0) \neq 0$ and $\tilde{f}(x)$ divides $f(x)$ then equation (3) holds.*

Proof. Apply Theorem 4.5 to the reciprocals of f and $\tilde{f}$. □

Corollary 4.7. *Let $P(x)$ be an irreducible polynomial (over $\mathbb{Q}$) of degree $n \geq 1$ and let $\{\gamma_1, \gamma_2, \ldots, \gamma_n\}$ be its complex roots. Then*

$$\left\| \frac{P(x)}{x-\gamma_i} \right\| \leq n\|P(x)\|$$

Proof. Take $\tilde{f}(x) = \frac{P(x)}{x-\gamma_i}$ and $f(x) = P(x)$ and apply Corollary 4.6. Then

$$\left\| \frac{P(x)}{x-\gamma_i} \right\| \leq \left(\sum_{j=0}^{n-(n-1)} \binom{n-1}{j}^2 \right)^{1/2} \|P(x)\| = \left(\sum_{j=0}^{1} \binom{n-1}{j}^2 \right)^{1/2} \|P(x)\|$$

$$= \left(\binom{n-1}{0}^2 + \binom{n-1}{1}^2 \right)^{1/2} \|P(x)\| = (n^2-2n+2)^{1/2} \|P(x)\| \leq n\|P(x)\|.$$

□

Theorem 4.8. *Let*

$$S_{g(x)} := \sum_{\substack{g(\beta)=0 \\ \beta\in\mathbb{C}}} |\beta_i|$$

then:

$$g_n\|\vec{h}\| \leq g_n n \left(S_{g(x)}\right) \|f(x)\|. \tag{4}$$

There are several ways to bound $S_{g(x)}$:
1) $S_{g(x)} \leq$ The degree of $g(x)$ times the rootbound described in [3].
2) $S_{g(x)} \leq M(g)/lc(g) + (n-1)$, where the Mahler measure can be bounded by $\|g(x)\|$.

Proof. (of Equation (4))

$$\|\vec{h}\| = \|P\| = \| \sum_{i=1}^{n} \beta_i \frac{f(x)}{x-\alpha_i} \| \leq n\|f(x)\| \sum_{i=1}^{n} |\beta_i| = n\|f(x)\|S_{g(x)}.$$

The first equality is by the definition of $\vec{h}$, the second by Lemma 4.4 and the inequality by Corollary 4.7. □

5 The Algorithms

Here we give the algorithms for computing the isomorphisms between number fields. The *Pre-processing* algorithm reduces the lattice of possible isomorphisms and gives the explicit isomorphism if there is only one. The next algorithm, *FindIsomorphism*, calls the *Pre-processing* algorithm and uses the remaining lattice to check which maps on roots corresponds to an isomorphism.

Algorithm: LLL-with-removals[10]
Input *A matrix A and a bound b.*

Output *A set of LLL reduced row vectors where the last vector is removed if its Gram-Schmitt length is greater than b.*

Algorithm: FindSuitablePrime
Input $(f(x), g(x), x, \mathrm{bp}, b, e)$, *where* bp *is the first prime to test, b and e determine the level to Hensel Lift to.*
Output $p, p^a, m, [[F_{d_1}, G_{d_1}], [F_{d_2}, G_{d_2}], \dots [F_{d_m}, G_{d_m}]]$, *see equation (2) for notation.*
Procedure

```
1. $p := \mathrm{bp}$, counter$:= 0$.
2. Repeat (until the algorithm stops in Steps 2(d)ii, 2(f) or 2(j)).
   (a) $p :=$ nextprime$(p)$
   (b) if $p|$ discriminant$(f, x)$ or $p|f_n$ then go to Step 2(a)
   (c) if $p|$ discriminant$(g, x)$ or $p|g_n$ then go to Step 2(a)
   (d) Distinct Degree Factor $f$ as $f \equiv F_{d_1} F_{d_2} \dots F_{d_m} \mod p$.
       i. If $m = 1$ then counter := counter $+1$.
       ii. If counter $> 25$ then print "They appear to be Galois" and
           return 0,0,0,0.
       iii. Return to Step 2(a).
   (e) Distinct Degree Factor $g$ as $g \equiv G_{d'_1} G_{d'_2} \dots G_{d'_{m'}} \mod p$.
   (f) If $m \neq m'$ or if the degrees of $F_i$ and $G_i$ do not match then return
       "There is no isomorphism".
   (g) Let $a := \lceil b^{e/10} 2^{e/4} \rceil$.
   (h) Hensel lift $f \equiv F_{d_1} F_{d_2} \dots F_{d_m} \mod p^a$ and likewise for $g$.
   (i) If deg$(F_1) > 0$ then store $p$ for later use.
   (j) Return $p, p^a, m, [[F_{d_1}, G_{d_1}], [F_{d_2}, G_{d_2}], \dots [F_{d_m}, G_{d_m}]]$ as output and
       stop.
```

Algorithm: Pre-Processing
Input *Polynomials* $f(x)$ *and* $g(x)$.
Output *Either "No isomorphism exists", a verified isomorphism, or a* $\mathbb{Z}$*-module which contains* $(g_n \vec{h}, g_n)$ *for every isomorphism h.*
Remark*: This lattice is given as the row space of a matrix C.*
Procedure

```
1. Initialize
   (a) $e := n + 1$.
   (b) $C :=$ $(n + 1)$ x $(n + 1)$ identity matrix.
   (c) $p := 3$.
   (d) $q := 0$.
   (e) Let $\{\mathrm{Base}_i\} \in \mathbb{Q}(\alpha)_{<n}, i = 1 \dots n$ be $\{\rho(1, 0, \dots, 0), \rho(0, 1, \dots, 0), \dots, \rho(0, 0, \dots, 1)\}$
       with $\rho$ defined in Section 2.
2. Let $S$ be an upper bound for $\sum_{\substack{g(\beta)=0\\ \beta\in\mathbb{C}}} |\beta_i|$, e.g. (4.8.1) or (4.8.2). Our
   implementation uses (4.8.1).
3. Let $b := nS\|f(x)\|$, be the bound described in Theorem 4.8.
4. Repeat (until the algorithm stops in 4(b), 4(e) or 4(i)).
   (a) $q := q + 1$.
   (b) $p, p^a, m, M_q := \mathrm{FindSuitablePrime}(f, g, x, p, b, e)$.
       i. If $p = 0$ then return $C$.
   (c) Find $Tr_p^d(f, \mathrm{Base}_i)$ for $i = 1 \dots n$ and $Tr_p^d(g, \beta)$ for each $d$ with
       deg($F_d$)$> 0$. The necessary $F_d, G_d$ are read from $M_q$.
   (d) $A := \begin{bmatrix} C & CT \\ 0 & P \end{bmatrix}$, where

       $$P := \begin{bmatrix} p^a & & \\ & \ddots & \\ & & p^a \end{bmatrix},$$

       $$T := \begin{bmatrix} Tr_{p_1}^{d_1}(f, \mathrm{Base}_1) & \dots & Tr_{p_1}^{d_m}(f, \mathrm{Base}_1) \\ Tr_{p_1}^{d_1}(f, \mathrm{Base}_2) & \dots & Tr_{p_1}^{d_m}(f, \mathrm{Base}_2) \\ \vdots & \dots & \vdots \\ Tr_{p_1}^{d_1}(f, \mathrm{Base}_n) & \dots & Tr_{p_1}^{d_m}(f, \mathrm{Base}_n) \\ Tr_{p_1}^{d_1}(g, \beta) & \dots & Tr_{p_1}^{d_m}(g, \beta) \end{bmatrix}$$

       the $d_1, \dots, d_m$ are as in Step 2(h) in Algorithm FindSuitablePrime.
       (Omitted entries are zero.)
   (e) If $CT \equiv 0 \mod p^a$ then
       i. counter := counter +1.
```

```
        ii. If counter $< 10$ then Go to Step 4(a) else return $C$ and stop.
    (f) $L :=$ LLL-with-removals($A$, $b$).
    (g) Let $C$ be the matrix with the first $n+1$ columns of $L$ and $B$ the
        remaining $m$ columns of $L$, so $L = [\ C \quad B\ ]$.
    (h) if $B \neq 0$ then
        i. $B := 10^{20} \cdot B$
        ii. $A := [\ C \quad B\ ]$
        iii. $L :=$ LLL-with-removals$(A, b)$, then go to Step 4(g).
    (i) Let $e :=$ number of rows of $C$.
        i. if $e = 0$ then output "There is no isomorphism."
        ii. if $e = 1$ then let $C$ be $[V, v]$ with $V$ an $n$ dimensional vector,
            and let $h$ be the polynomial corresponding to $V/v$.
            A. Let iso$:= \frac{h(\alpha)g_n}{f'(\alpha)}$.
            B. If iso satisfies $g$ then output "iso is the only isomor-
               phism."
            C. If not then output "There is no isomorphism."
        iii. Else, go to Step 4a.
```

Remark 5.1. If we let d be the number of isomorphisms $(\mathbb{Q}(\beta) \to \mathbb{Q}(\alpha))$ then just by looking at the input/output of the Pre-processing algorithm we see that:

$$\text{If } d \in \{0,1\} \text{ then the output is either } \begin{cases} \text{all isomorphisms} \\ \text{a lattice} \end{cases}$$

$$\text{If } d > 1 \text{ then the output is a lattice}$$

In the next algorithm we use the lattice outputted from *Pre-Processing* to check all possible maps on the roots to see which are actual isomorphisms. This will find all isomorphisms from $\mathbb{Q}(\beta) \to \mathbb{Q}(\alpha)$.

The following algorithm is described for (linear) roots of f and g in $\mathbb{Q}_p$ and can be extended to the roots of F_i and G_i instead.

Remark 5.2. It should be noted that even if the Pre-processing Algorithm does not find the isomorphism(s), the LLL switches it performs will still contribute to the FindIsomorphism Algorithm. This is true for the same reason as in [11, pg 175].

Algorithm: FindIsomorphism

Input *Two polynomials, $f, g \in \mathbb{Z}[x]$ which are irreducible and of the same degree.*

Output *The set of all isomorphisms from $\mathbb{Q}[x]/(f)$ to $\mathbb{Q}[x]/(g)$.*

Procedure

```
1. $C :=$ Pre-Processing($f(x)$ , $g(x)$, $x$).
2. If Step 2(i) in Algorithm FindSuitablePrime (called from Pre-Processing)
   stored at least one prime, then choose one with smallest deg($F_1$). Oth-
   erwise keep calling Algorithm FindSuitablePrime until such a prime is
   found.
3. Let $\alpha_1, \ldots, \alpha_d$ be the roots of $F_1$ and Hensel lift them to $\mathbb{Z}/(p^a)$ with a
   as in Algorithm FindSuitablePrime. Likewise let $\beta_1, \ldots, \beta_d \in \mathbb{Z}/(p^a)$
   be the roots of $G_1$.
4. For j from 1 to d do:
   (a) Apply steps 4(d) through 4(i)ii of Pre-Processing using
```

$$T := \begin{bmatrix} \mathrm{Base}_1|_{\alpha=\alpha_j} \\ \mathrm{Base}_2|_{\alpha=\alpha_j} \\ \vdots \\ \mathrm{Base}_n|_{\alpha=\alpha_j} \\ \beta_1 \end{bmatrix}$$

```
   (b) If $e > 1$ then
       i. Hensel Lift the roots of f and g to twice the current p-adic
          precision, i.e. $p^{2a}$.
       ii. Apply Step 4(a) with the more precise roots.
```

5.1 Proofs of Termination and Validity

In this section we prove that the algorithms terminate and show that the algorithm does indeed produce all isomorphisms of the number fields $\mathbb{Q}(\beta)$ and $\mathbb{Q}(\alpha)$.

First we cite a lemma which shows why we can use LLL with removal in our algorithm.

Lemma 5.3. *Let $\{b_1, \ldots, b_k\}$ be a basis for a lattice, C, and $\{b_1^*, \ldots, b_k^*\}$ the corresponding Gram-Schmitt orthogonalized basis for C. If $\|b_k^*\| > B$ then a vector in C with norm less than B will be a $\mathbb{Z}$-linear combination of $\{b_1, \ldots, b_{k-1}\}$.*

Proof. This follows from the proof of Proposition 1.11 in [9], it is also stated as Lemma 2 in [10]. □

Corollary 5.4. *Using LLL-with-removals on a lattice containing $g_n\vec{h}$ with the bound b, computed in Step 3 of* Pre-Processing*, does not remove $g_n\vec{h}$ from the lattice.*

Proof. Using Lemma 5.3 and Theorem 4.8 we know that removing final vectors with Gram-Schmitt length bigger than b does not remove any of the $g_n\vec{h}$. □

Lemma 5.5. *The* Pre-Processing *Algorithm terminates.*

Proof. The steps of the *Pre-Processing* algorithm are known algorithms that terminate, the only one that is not immediate is Step 4(h). Step 4(h) terminates because each run increases the determinant of the lattice (Step 4(h)i) and any final (see Lemma 5.3) vector with Gram-Schmitt length bigger than b is removed, thus the number of vectors is monotonically decreasing and hence it can only be run a finite number of times. □

Lemma 5.6. *The* FindIsomorphism *Algorithm described above terminates.*

Proof. For Steps 1-3 it is clear why each will terminate. We show that Step 4 terminates by contradiction.
Suppose Step 4 never terminates (i.e. the lattice always has dimension $>$ 1) then it contains at least two vectors: (h_1, e_1) and (h_2, e_2). Let $H = h_1$ if $e_1 = 0$ or $H = e_1h_2 - e_2h_1$ otherwise. Then $H(\alpha) \equiv 0 \mod p^a$. We get a contradiction when p^a is larger than an upper bound for $\text{Res}_x(H, f)$. An upper bound for H can be obtained from equation 1.7 in [9] and the fact that the last vector after LLL-with-removals has Gram-Schmitt length $\leq b$. □

6 Heuristic estimate on the rank of C

Let $C \subseteq \mathbb{Z}^{n+1}$ be the output of the Pre-Processing Algorithm.

Observation 6.1. *In most (but not all) examples, dim(C) is equal to $n - n/d + 1$.*

This means that Pre-Processing is most effective when $d = 1$. Though as pointed out in Remark 5.2 the work done in Pre-Processing reduces the amount left to do.

Let G be the Galois group of $f(x)$ and let H_i be the stabilizer of α_i for $i \in \{1, 2, \ldots, n\}$, where the α_i are the roots of $f(x)$.

Let d be the number of j such that $H_1 = H_j$, then d is the number of automorphisms of $\mathbb{Q}(\alpha)$. If $\mathbb{Q}(\alpha)$ and $\mathbb{Q}(\beta)$ are isomorphic then d will also be the number of isomorphisms from $\mathbb{Q}(\beta)$ to $\mathbb{Q}(\alpha)$.

Remark 6.2. We view G, which as the Galois group acts on $\{\alpha_1, \alpha_2, \ldots, \alpha_n\}$, as acting on the set $\{1, 2, \ldots, n\}$ in the most natural way. Hence we view G as a subgroup of S_n, the symmetric group.

We will construct a partition matrix as follows. For each $\sigma \in G$, group together the cycles of the same length. Different group elements and cycle lengths will correspond to different rows. For each element of G and for each cycle length in σ, construct one row of P as follows: place a 1 in the i^{th} entry if α_i is in a cycle of that length. We call the resulting matrix P.

For example for $\sigma_1 = (1)(2)(3)(456)$ and $\sigma_2 = (12)(3456)$ we would get the following partition matrix :

$$P = \begin{array}{r} \sigma_1\ l = 1 \\ \sigma_1\ l = 3 \\ \sigma_2\ l = 2 \\ \sigma_2\ l = 4 \\ \vdots \end{array} \begin{bmatrix} 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 \\ 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 & 1 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \end{bmatrix}$$

Since there are d automorphisms the number of distinct columns of P will be $\leq n/d$, hence rank$(P) \leq n/d$ and thus Nullspace$(P) \geq n - n/d$.

This translates into an estimate on the rank of the lattice C since it helps us bound

$$V = \bigcap_{p,d} Ker(Tr_p^d(f, *)).$$

Nullspace(P) corresponds to elements for which all sub-traces are zero, so dim(Nullspace$(P)) \leq$ dim(V).

Since we used LLL-with-removals with cut off point b, if V admits a basis whose norms are all smaller than b then $V \subseteq \pi_{1\ldots n}(C)$, where $\pi_{1\ldots n}$ is the projection on the first n coordinates.

Therefore under that assumption

$$\dim(\pi_{1\ldots n}(C)) \geq \dim(\text{Nullspace}(P)) \geq n - n/d.$$

This leads to our estimate:

$$\dim(C) \approx n - n/d + 1. \tag{5}$$

For most polynomials taken from the database [5] our estimate is an equality. Peter Muller provided an infinite sequence of counter-examples for the case we were most interested in ($d = 1$). For the first group in this sequence, the database [5] provides the following example:
$f := x^{14} + 2x^{13} - 5x^{12} - 184x^{11} - 314x^{10} + 474x^9 + 1760x^8 + 1504x^7 - 400x^6 - 1478x^5 - 818x^4 + 73x^3 + 260x^2 + 121x + 23$,
which has one automorphism but the Pre-processing algorithm outputs a dimension 2 lattice.

7 Computational Efficiency

We compare our algorithm implemented in Maple with other methods of finding isomorphisms. The best algorithm we know for factoring over number fields is give by Belabas in [6], which is implemented in Pari/Gp. Both implementations were run on a standard 2.2GHz processor. We tested them on the field extensions given by the following two degree 25 polynomials:
$f_1 := 2174026154062500000\,x^{25} - 12927273797812500000\,x^{24} + 44254465332187500000\,x^{23} - 102418940816662500000\,x^{22} + 180537842164766250000\,x^{21} - 249634002590534050000\,x^{20} + 292282923494920350000\,x^{19} - 384197583430502150000\,x^{18} + 815826517614521346000\,x^{17} - 2131245874043847615600\,x^{16} + 4352260622811059705104\,x^{15} - 6463590834754261173232\,x^{14} + 6920777688226436002712\,x^{13} - 4525061881234027826296\,x^{12} + 528408698276686662696\,x^{11} + 2762117617850418790424\,x^{10} - 4343360968383689825174\,x^{9} + 4191186502263628451150\,x^{8} - 2802452375464033976482\,x^{7} + 1332292171242725153638\,x^{6} - 161285249796825311495\,x^{5} - 429207332210687640181\,x^{4} + 264147194777000152867\,x^{3} + 60321986329616997729\,x^{2} - 42885793067858008650\,x + 13774402803823804220$ and
$f_2 := -42885793067858008650\,x - 13774402803823804220 - 161285249796825311495\,x^{5} + 429207332210687640181\,x^{4} + 264147194777000152867\,x^{3} - 60321986329616997729\,x^{2} - 1332292171242725153638\,x^{6} - 2802452375464033976482\,x^{7} - 4191186502263628451150\,x^{8} - 4343360968383689825174\,x^{9} - 2762117617850418790424\,x^{10} + 528408698276686662696\,x^{11} + 4525061881234027826296\,x^{12} + 6920777688226436002712\,x^{13} + 6463590834754261173232\,x^{14} + 4352260622811059705104\,x^{15} + 815826517614521346000\,x^{17} + 384197583430502150000\,x^{18} + 292282923494920350000\,x^{19} + 249634002590534050000\,x^{20} + 180537842164766250000\,x^{21} + 102418940816662500000\,x^{22} + 44254465332187500000\,x^{23} + 2131245874043847615600\,x^{16} + 12927273797812500000\,x^{24} + 2174026154062500000\,x^{25}$.

These are field extensions with one isomorphism between them. Using Belabas' method we have a runtime of 11.69 seconds, which includes the indispensable operation of defining the number field, and with our algorithm we have a runtime of 2.97 seconds.

We also tested this algorithm in the case where our heuristic estimate on the rank does not apply, namely on the first counter-example. Using Belabas' method we have a runtime of .091 seconds and with our algorithm we have a runtime of .797 seconds.

Bearing in mind that our implementation is not optimized and is coded in Maple we expect it to be much faster when using a better implementation of LLL.

We also tested them on a larger example, namely the degree 81 example located at [7], our algorithm found the isomorphism in 2326.581 seconds and the Belabas' implementation did not finish as it ran out of memory after trying for a few days. Therefore there are certainly advantages to using this algorithm, as there are examples where there is a significant reduction in the required runtime/resources.

8 Summary

Method II (from Section 1) can be described by the following procedure: first pick p such that f and g have roots in $\mathbb{Q}_p$. Fix one root $\beta \in \mathbb{Q}_p$ of g, take all roots $\alpha_1, \ldots, \alpha_d \in \mathbb{Q}_p$ of f. Then for each α_i use LLL to find $h_i \in \mathbb{Q}[x]$ (if it exists) with $h_i(\alpha_i) = \beta_i$.

Our approach is similar, the difference is that we start with LLL reductions (obtained from sub-traces) that are valid for all α_i. This way, a portion of the LLL computation to be done for each α_i is now shared. The time saved is then $(d-1)$ times the cost of the shared portion. This can be made rigorous by introducing a progress counter for LLL cost similar to [10].

9 References

1. Granville, A. "Bounding the coefficients of a divisor of a given polynomial", Monatsh. Math. 109 (1990), 271-277.

2. Conrad, Kieth. "The different ideal". Expository papers/Lecture notes. Available at: http://www.math.uconn.edu/∼kconrad/blurbs/gradnumthy/different.pdf

3. Monagan, M. B. "A Heuristic Irreducibility Test for Univariate Polynomials", J. of Symbolic Comp., 13, No. 1, Academic Press (1992)

47-57.

4. Dahan, X. and Schost, É. 2004. “Sharp estimates for triangular sets”. In Proceedings of the 2004 international Symposium on Symbolic and Algebraic Computation (Santander, Spain, July 04 - 07, 2004). ISSAC ’04. ACM, New York, NY, 103-110.

5. Database by Jürgen Klüners and Gunter Malle , located at: http://www.math.uni-duesseldorf.de/~klueners/minimum/minimum.html

6. Belabas, Karim. “A relative van Hoeij algorithm over number fields”. J. Symbolic Computation, Vol. 37 (2004), no. 5, pp. 641-668.

7. Website with implementations and Degree 81 examples: http://www.math.fsu.edu/ ~vpal/Iso/

8. van Hoeij, Mark. “Factoring Polynomials and the Knapsack Problem.” J. Number Th. 95, 167-189, 2002

9. Lenstra, A. K.; Lenstra, H. W., Jr.; Lovász, L. “Factoring polynomials with rational coefficients”. Mathematische Annalen 261 (4), 515-534, 1982.

10. M. van Hoeij and A. Novocin, “ Gradual sub-lattice reduction and a new complexity for factoring polynomials”, accepted for proceedings of LATIN 2010.

11. Cohen, Henri A Course in Computational Algebraic Number Theory, Graduate Texts in Mathematics 138, Springer-Verlag, 1993.

Florida State University 211 Love Building, Tallahassee, Fl 32306-3027, USA
E-mail address: hoeij@math.fsu.edu
E-mail address: vpal@math.fsu.edu