# Generating Subfields

Mark van Hoeij, Jürgen Klüners, Andrew Novocin

ISSAC'2011

# Notations.

Let $k(\alpha)$ be a field extension of $k$ of degree $n$ with minpoly $f$.

**Goal:** Find all subfields of $k(\alpha)$ over $k$, hopefully efficient in practice as well as in theory.

**Theoretical issue:** There could be more than polynomially many subfields. To obtain polynomial time complexity, we compute only a "generating set" of subfields (every subfield of $k(\alpha)$ over $k$ will be an intersection of these "generating subfields").

**Practical issue:** For the number field case, we can use LLL, which requires a cut-off bound. To optimize CPU time, we need to minimize this bound. The hardest part of the paper is to prove a cut-off bound that is linear in $||f||_2$.

# Applications

**Example 1.** Use a CAS to solve this system of equations:

$$a^2 - 2ab + b^2 - 8 = 0, \quad a^2b^2 - (a^2 + 2a + 5)b + a^3 - 3a + 3 = 0$$

**Result:** $a = \alpha, \quad b =$

$$\frac{-17\alpha^7}{1809} + \frac{61\alpha^6}{3618} + \frac{371\alpha^5}{1809} - \frac{1757\alpha^4}{3618} - \frac{563\alpha^3}{603} + \frac{6013\alpha^2}{3618} + \frac{3184\alpha}{1809} + \frac{7175}{3618}$$

where $\alpha$ is a solution of

$$x^8 - 20x^6 + 16x^5 + 98x^4 + 32x^3 - 12x^2 - 208x - 191 = 0.$$

Example 1 has a simpler solution:

$$a = \sqrt{3} + \sqrt[4]{2} - \sqrt{2}, \quad b = \sqrt{3} + \sqrt[4]{2} + \sqrt{2} \tag{1}$$

To find it we first need subfields of $\mathbb{Q}(\alpha)$.

# Applications

Bostan and Kauers [Proc AMS 2010] gave an algebraic expression for the generating function for Gessel walks, using two minpoly's with a combined size of 172 Kb. By computing subfields, this expression could be reduced to just 300 bytes, a 99.8% reduction. The idea is:

When $\mathrm{char}(k) = 0$, then a tower of extensions

$$k \subseteq k(\alpha_1) \subseteq k(\alpha_2) \subseteq k(\alpha_3) = K$$

can be given by a single extension $K = k(\alpha)$.

In general, the **primitive element theorem** will produce an $\alpha$ with a minpoly $f(x)$ of large size. Thus we can expect to reduce expression sizes using the reverse process (computing subfields).

# The Subfield Polynomial

Let $k \subseteq k(\alpha)$ be an algebraic extension with minpoly $f$. Let $k \subseteq L \subseteq K$ be a subfield. Let $g \in L[x]$ be the minpoly of $\alpha$ over $L$.

**Definition:** We can this $g$ the *subfield-polynomial* of $L$.

**Remark:** The subfield $L$ is generated by the coefficients of $g$. The coefficients of $f/g$ form a spanning set of $L$ as a $k$-vector space.

**Note:** A subfield-poly is also a factor of $f$ in $k(\alpha)[x]$. So we could find all subfields by trying out every factor of $f$ in $k(\alpha)[x]$.

# Factors of $f$

Let $f = f_1 \cdot f_2 \cdots f_r$ be a factorization of $f$ in $k(\alpha)[x]$. We can assume that $f_1 = x - \alpha$.

**Finding Subfields, Exponential Complexity:**
For each of the $2^r$ monic factors of $f$ in $k(\alpha)[x]$, compute the field generated by the coefficients of that factor.

**Finding Subfields, Polynomial Complexity:**
We perform a computation for each polynomial $f_2$, $f_3$, ..., $f_r$.
**Problems:**

1. These $f_2$, $f_3$, ... are not subfield-polynomials (i.e. we do not get a subfield by simply looking at their coefficients).
2. We do not get all subfields in this way.

# Finding subfields

Let $f = f_1 \cdot f_2 \cdots f_r$ be a factorization of $f$ in $k(\alpha)[x]$, where $f_1 = x - \alpha$. For each $i = 2, \ldots, r$, even though $f_i$ is not a subfield-poly, we can still define a corresponding subfield, as follows:

$$L_i = \{h(\alpha) \mid h(x) \in k[x]_{<n} \text{ and } h(x) \equiv h(\alpha) \bmod f_i\}.$$

If $h(x) \in k[x]$ with degree $< n$, then the condition

$$h(x) \equiv h(\alpha) \bmod f_i$$

translates into $k$-linear equations for the coefficients of $h$. Thus, $L_i$ can be computed by solving a system of $k$-linear equations.

# A generating set

A set $S$ of subfields of $k(\alpha)$ over $k$ is called a **generating set** if every subfield of $k(\alpha)$ over $k$ is an intersection of members of $S$.

**Theorem**: Let $L_2, L_3, \ldots, L_r$ be the subfields from the previous slide. Then $\{L_2, L_3, \ldots, L_r\}$ is a generating set.

**Theorem**: If $k = \mathbb{Q}$ then a generating set can be computed in polynomial time.

By computing intersections, we find all subfields. The cost depends linearly on the number of subfields (this number can be more than polynomial in $n$).

# Factoring over a completion

Polynomial time $\not\Longrightarrow$ efficient in practice.

Let $k = \mathbb{Q}$ and $K = \mathbb{Q}(\alpha)$. To factor $f$ in $K[x]$, we could use Belabas' algorithm. It first factors $f$ over some completion $\tilde{K}$ of $K$, and then uses LLL techniques.

Let $\tilde{K}$ be $\mathbb{Q}_p$, where $p$ is chosen such that $f$ has a root in $\mathbb{Q}_p$ (that way $\mathbb{Q}(\alpha) \subseteq \mathbb{Q}_p$).

**Idea:** Instead of:

factoring over $\tilde{K} \overset{\text{LLL}}{\Longrightarrow}$ factoring over $K \overset{\text{LinSolve}}{\Longrightarrow}$ subfields of $K$

we can do directly:

factoring over $\tilde{K} \overset{\text{LLL}}{\Longrightarrow}$ subfields.

# Factoring over a completion

**A nice result:** In general, $f$ will have more factors over $\tilde{K}$ than over $K$, in other words, if we factor $f = f_1 \cdots f_r$ then $r$ will increase if we use a completion $\tilde{K}$ instead of $K$ itself.
Nevertheless, the set $\{L_2, \ldots, L_r\}$ will stay the same!

If we use a factor $f_i$ in $K[x]$, where $K = k(\alpha)$, then the subfield $L_i$ is determined by solving $k$-linear equations.

But when $f_i \in \tilde{K}[x]$ then $f_i$ can only be computed with finite accuracy (mod $p^a$ for some finite $a$) (assume from now $k = \mathbb{Q}$).
Then the linear equations are also computed with finite accuracy.

To solve them we need LLL.

# Factoring over a completion, then LLL

Recall that the idea to improve efficiency was to replace:

factoring over $\tilde{K} \overset{\text{LLL}}{\Longrightarrow}$ factoring over $K \overset{\text{LinSolve}}{\Longrightarrow}$ subfields of $K$

by:

factoring over $\tilde{K} \overset{\text{LLL}}{\Longrightarrow}$ subfields.

We use **LLL-with-removals** which is a modification of LLL that removes the last vector whenever it has a Gram-Schmidt length that exceeds a specified bound $B$. This bound $B$ needs to be sharp because:

- If $B$ is too low, then the algorithm won't be correct.
- If $B$ is higher than necessary, then we do not get an optimal running time, which defeats the purpose of this approach.

# The cut-off bound $B$

Roughly speaking, if $B$ has twice as many digits as it needs to have, then our algorithm would be about 4 times slower than it should be. If the goal is practical efficiency instead of theoretical complexity, then such a constant factor matters.

**Necessary condition for** $B$: The lattice to which we apply LLL corresponds to a $\mathbb{Z}$-module $\Lambda$ inside $K = \mathbb{Q}(\alpha)$. Suppose that $L$ is the subfield that we are aiming to find. The number $B$ needs to be large enough so that inside $\Lambda$, there exists a basis $b_1, \ldots, b_d$ of $L$ for which each $b_i$ corresponds to a vector with length $\leq B$.

This condition ensures that, after calling LLL-with-removals with bound $B$, a basis of $L$ will still exist in the span of the remaining vectors. It allows us to prove correctness of the algorithm.

# The lattice that LLL works with

We could choose

- $\Lambda = \mathbb{Z} \cdot 1 + \mathbb{Z} \cdot \alpha + \cdots + \mathbb{Z} \cdot \alpha^{n-1}$. This would lead to a huge bound for $B$.
- $\Lambda = \mathbb{Z} \cdot 1/f'(\alpha) + \mathbb{Z} \cdot \alpha/f'(\alpha) + \cdots + \mathbb{Z} \cdot \alpha^{n-1}/f'(\alpha)$.
  See also Dahan and Schost ISSAC'2004. In this setting, a vector $(v_0, v_1, \cdots)$ in the LLL-output does not represent the algebraic number $\sum v_i \alpha^i$. Instead, this vector represents

$$\frac{v_0\alpha^0 + \cdots v_{n-1}\alpha^{n-1}}{f'(\alpha)}$$

  This leads fairly quickly to a bound $B$ that depends quadratically on $||f||_2$. We then have an algorithm that works well in practice.

# A problem remains

A bound $B$ that depends quadratically on $||f||_2$ is fairly easy to prove, using standard techniques from number theory. One considers the ring of algebraic integers $O_K$, which is a $\mathbb{Z}$-module. The subfield $L$ contains the $\mathbb{Z}$-module $O_L$, and $O_L = O_K \bigcap L$. The rings $O_K$ and $O_L$ have many nice properties that we can use to quickly prove a quadratic bound. End of story?

**Problem:** We wrote an implementation, and observed that in practice, a bound that is *linear* in $||f||_2$ always holds.

So the quadratic bound appears to be not sharp, we could cut $\log(B)$ by a factor 2, which will make the algorithm 4 times faster! How to prove this linear bound?

# Proving a linear bound

It does not suffice to bound elements of $O_L$, a linear bound can not be proven in that way. Instead, we have to work with the $\mathbb{Z}$-module $1/f'(\alpha) \cdot (\mathbb{Z}\alpha^0 + \cdots \mathbb{Z}\alpha^{n-1})$ intersected with $L$. Working with this $\mathbb{Z}$-module makes the proof much more technical.

We prove that this $\mathbb{Z}$-module contains the dual (under the trace bilinear form) of $O_L$, and then use Banaszcyk's transference theorem to translate a bound for elements of $O_L$ into a better bound for elements of this $\mathbb{Z}$-module. We then get a linear bound for the vectors that LLL works with.

The hardest two pages of the paper are devoted to proving this linear bound. The end result of this work is a speedup of a factor 4.