UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL
INSTITUTO DE MATEMÁTICA E ESTATÍSTICA
PROGRAMA DE PÓS-GRADUAÇÃO EM MATEMÁTICA APLICADA

Computing Subfields

por

Jonas Szutkoski

Tese submetida como requisito parcial
para a obtenção do grau de

Doutor em Matemática Aplicada

Prof. Dr. Vilmar Trevisan
Orientador

Prof. Dr. Luiz Emilio Allem
Coorientador

Porto Alegre, Dezembro de 2017.

CIP - CATALOGAÇÃO NA PUBLICAÇÃO

Szutkoski, Jonas

Computing Subfields / Jonas Szutkoski.—Porto Alegre: PPGMAp da UFRGS, 2017.

142 p.: il.

Tese (doutorado) —Universidade Federal do Rio Grande do Sul, Programa de Pós-Graduação em Matemática Aplicada, Porto Alegre, 2017.
Orientador: Trevisan, Vilmar; Coorientador: Allem, Luiz Emilio

Tese: Álgebra Computacional
Subcorpos, Reticulado de Subcorpos, Partições, Decomposição de Funções Racionais

Computing Subfields

por

Jonas Szutkoski

Tese submetida ao Programa de Pós-Graduação em Matemática Aplicada do Instituto de Matemática e Estatística da Universidade Federal do Rio Grande do Sul, como requisito parcial para a obtenção do grau de

Doutor em Matemática Aplicada

Linha de Pesquisa: Álgebra Computacional

Orientador: Prof. Dr. Vilmar Trevisan

Coorientador: Prof. Dr. Luiz Emilio Allem

Banca Examinadora:

Prof. Dr. Cícero Fernandes de Carvalho
PPMAT/UFU

Prof. Dr. Marcelo Escudeiro Hernandes
PMA/UEM

Prof. Dr. Carlos Hoppen
PPGMAp/IME/UFRGS

Tese Apresentada em
01/12/2017.

Prof. Dr. Carlos Hoppen
Coordenador

Sumário

Lista de Tabelas

Lista de Figuras

Lista de Algoritmos

LISTA DE SÍMBOLOS

$\mathbb{Z}, \mathbb{Q}, \mathbb{R}, \mathbb{C}$	set of integer, rational, real and complex numbers.
$\lceil a \rceil$	for a real number a, denotes the least suceeding integer.
$\mathbb{Z}_p, \mathbb{Q}_p$	ring of p-adic integers and field of p-adic numbers.
$m\mathbb{Z}$	ring of integers modulo m.
$\mathbb{F}_q$	field of characteristic q.
$R/\mathfrak{m}$	for a ring R and an ideal $\mathfrak{m} \subset R$, denotes the residue class ring.
$f \bmod \mathfrak{m}$	for a polynomial f defined over R, denotes the polynomial defined over $R/\mathfrak{m}$ by reducing the coefficients of f modulo the ideal $\mathfrak{m} \subset R$.
$\deg(f)$	for a polynomial f, denotes the degree of f.
$\text{coeffs}(f)$	for a polynomial f, denotes the set of coefficients of f.
$\text{lc}(f)$	for a polynomial f, denotes the coefficient of the leading term of f.
$\deg_x(f)$	for a multivariate polynomial f, denotes the degree of f with respect to x.
$\text{disc}(f)$	for a polynomial f, denotes the discriminant of f.
$\|f\|$	for a polynomial f, is the norm of the vector of coefficients of f.
$g \mid f$	for polynomials f, g, there exists a polynomial h such that $f = g \cdot h$.
$\gcd(g, h)$	for polynomials g, h, denotes the greatest common divisor of g and h.
$\text{Res}_x(g, h)$	for polynomials g, h, denotes the resultant of g and h with respect to the variable x.

$g \circ h$	composition of the functions g and h.
K/k	for fields K, k, means that $k \subseteq K$.
$\mathcal{O}_K$	for a field K, denotes the ring of integers of K.
$k[x]$	ring of polynomials with coefficients in k.
$k[x]_{<n}$	set of polynomials over k with degree less than n.
$k(t)$	field of rational functions with coefficients in k in the indeterminate t.
$\{f_1, \ldots, f_r\}^\pi$	for polynomials $f_1, \ldots, f_r$, denotes the set of all products of $f_1, \ldots, f_r$.
$\lvert P \rvert$	for a partition P, denotes the number of elements in P.
$P \vee P'$	for partitions P, P', denotes the *join* of P and P'.

ABSTRACT

In this work, we consider the problem of computing the subfield lattice of a separable and finite degree field extension $k(\alpha)/k$. That is, we wish to find all fields L such that $k \subseteq L \subseteq k(\alpha)$. Until recently, the algorithm used by most Computer Algebraic Systems relied on a combinatorial problem on the roots of the minimal polynomial f of α over k, which can be a computationally expensive task.

In 2013, another algorithm was presented to find the subfield lattice of $k(\alpha)/k$. This method computes a *small* set of subfields, called *principal subfields*, with the property that any other subfield of $k(\alpha)/k$ is the intersection of some of these principal subfields. Thus, the problem of computing the subfield lattice can be split into 2 steps: 1) Find the principal subfields of $k(\alpha)/k$ and 2) Compute all intersections of these subfields. The first step can be executed in polynomial time however, the second step can not and thus, dominates the algorithm complexity.

Our main goal is to improve the second step, both theoretically and practically. More specifically, we develop a method to quickly compute all intersections of principal subfields. While the complexity is still not polynomially bounded (in fact, it can not be for the total number of subfields is not polynomially bounded), this new method helps to improve the non-polynomial part of the complexity. Practical performance is also improved when the number of intersections is large.

We also focus on two special cases: number fields and rational function fields. For the number field case (*i.e.*, when $k = \mathbb{Q}$), we also present an improvement for the first step. For the rational function field case, computing the subfield lattice of the extension $K(t)/K(f(t))$ defined by $f(t) \in K(t)$ yields all decompositions of the rational function $f(t)$. Our algorithm outperforms previous algorithms for computing rational function decompositions.

RESUMO

Neste trabalho, consideramos o problema de calcular o reticulado de subcorpos de uma extensão separável e de grau finito $k(\alpha)/k$. Isto é, queremos encontrar todos os corpos L tais que $k \subseteq L \subseteq k(\alpha)$. Até recentemente, o algoritmo utilizado pela maioria dos Sistemas Algébricos Computacionais baseava-se em um problema combinatorial nas raízes do polinômio minimal f de α sobre k.

Em 2013, um algoritmo foi apresentado para encontrar tais subcorpos. Este método calcula um pequeno conjunto de subcorpos, chamados de *subcorpos principais*, com a propriedade de que todo subcorpo de $k(\alpha)/k$ é a interseção de alguns destes subcorpos. Assim, calcular o reticulado de subcorpos é dividido em duas etapas: 1) Encontrar os subcorpos principais de $k(\alpha)/k$ e 2) Calcular todas as interseções destes subcorpos. A primeira etapa pode ser feita em tempo polinomial. Entretanto, a segunda etapa não pode e assim, domina a complexidade do algoritmo.

Nosso objetivo é melhorar a segunda etapa, tanto em teoria quanto na prática. Para isso, mostramos como rapidamente calcular todas as interseções entre os subcorpos principais. Embora a complexidade continue não sendo limitada polinomialmente (e também não poderia ser, pois o número total de subcorpos não o é), conseguimos melhorar a complexidade do algoritmo. Também notamos um melhoramento na prática, principalmente quando o número de subcorpos é grande.

Além disso, estudamos dois casos especiais: corpos numéricos e o corpo das funções racionais. Para corpos numéricos (*i.e.*, quando $k = \mathbb{Q}$), também apresentamos um melhoramento para a primeira etapa. No segundo caso, os subcorpos da extensão $k(t)/k(f(t))$, definida por $f(t) \in k(t)$, nos fornecem decomposições da função racional $f(t)$. Nosso algoritmo tem uma performance melhor que algoritmos anteriores para calcular as decomposições de funções racionais.

RESUMO EXPANDIDO

Título: Computando Subcorpos

Este trabalho trata do problema de calcular subcorpos de uma extensão de corpos separável e de grau finito. Além de ser um problema interessante por si só, o problema de calcular subcorpos possui diversas aplicações, tais como o cálculo do grupo de Galois de um polinômio [17, 18], expressar raízes de um polinômio em termos de radicais [30] (quando isto for possível), simplificação de expressões algébricas, decomposição polinomial [10] e de funções racionais [60], entre outros.

O principal resultado da Teoria de Galois afirma que calcular o conjunto de todos os subcorpos de uma extensão K/k, separável e de grau finito, definida por um polinômio $f \in k[x]$, é equivalente ao problema de calcular certos subgrupos do grupo de Galois do polinômio f. Por se tratar de um problema clássico, já existem diversos algoritmos que calculam subcorpos de tais extensões, como por exemplo [13, 17, 25, 30, 32]. Até recentemente, o algoritmo utilizado na maioria dos Sistemas Algébricos Computacionais era baseado no trabalho de Klüners & Pohst [28]. Tal algoritmo funciona bem em certos caso, mas sua complexidade é exponencial no grau da extensão. Uma ideia deste algoritmo é apresentada no Capítulo 2.

Mais recentemente, van Hoeij *et al.* [51] também apresentaram um algoritmo para calcular os subcorpos de uma extensão finita e separável K/k. Neste trabalho iremos melhorar este algoritmo, que se baseia no seguinte resultado de [51].

Teorema *Seja K/k uma extensão separável e de grau n. Seja α um elemento primitivo de K e seja $f \in k[x]$ seu polinômio minimal. Sejam $f_1, \ldots, f_r$ os fatores irredutíveis de f sobre K e sejam $L_i := \{h(\alpha) \in K : h(x) \equiv h(\alpha) \bmod f_i\}$, $i = 1, \ldots, r$. Para todo subcorpo L de K/k, existe $I \subseteq \{1, \ldots, r\}$ tal que $L = \cap_{i \in I} L_i$.*

Os subcorpos $L_1, \ldots, L_r$ são chamados de *subcorpos principais* da extensão K/k. Aqui, também podemos utilizar uma fatoração $\hat{f}_1, \ldots, \hat{f}_{\hat{r}}$ de f sobre qualquer extensão algébrica $\hat{K}$ de K. Fazendo isso, podemos obter mais fatores (isto é, $\hat{r} \geq r$) porém, o conjunto de subcorpos principais permanece o mesmo. Mais detalhes sobre a escolha de $\hat{K}$ quando $k = \mathbb{Q}$ são dados no Capítulo 4.

Mais geralmente, se $L_1, \ldots, L_r$ são subcorpos quaisquer de K/k e se todo subcorpo L de K/k é a interseção de alguns destes subcorpos, dizemos que $\{L_1, \ldots, L_r\}$ é um *conjunto gerador* para K/k. Assim, o teorema anterior afirma que os subcorpos principais de K/k formam um conjunto gerador para K/k. Se g é um fator qualquer de f, também podemos definir o conjunto

$$L_g = \{h(\alpha) \in K : h(x) \equiv h(\alpha) \bmod g(x)\}.$$

O conjunto L_g é um subcorpo de K/k. Seja $g_1, \ldots, g_r$ uma fatoração qualquer de f sobre K (isto é, g_i não necessariamente irredutível sobre K). Se $g_1 = x - \alpha$ e se $L_{g_1}, \ldots, L_{g_r}$ formam um conjunto gerador para K/k, dizemos que $g_1, \ldots, g_r$ é uma *subfield factorization* para K/k. Com estas definições, podemos encontrar todos os subcorpos de K/k em 3 passos.

1. Encontrar uma *subfield factorization* $g_1, \ldots, g_r$ para K/k.
2. Calcular os correspondentes subcorpos $L_{g_1}, \ldots, L_{g_r}$ de K/k.
3. Calcular todas as interseções entre $L_{g_1}, \ldots, L_{g_r}$.

Os passos 1 e 2 podem ser executados em tempo polinomial (desde que fatoração em $K[x]$ possa ser feita em tempo polinomial). Porém, o passo 3 não pode ser feito em tempo polinomial. O número de interseções feita no passo 3 pode ser limitado por rm, onde m é o número total de subcorpos de K/k. Entretanto, o número m não é limitado polinomialmente no grau n da extensão K/k. Assim,

o custo do passo 3 é dado por rm vezes o custo de cada inteseção, que é feito utilizando-se Álgebra Linear sobre k (conforme [51]).

Por simplicidade, vamos supor que a *subfield factorization* é dada pela fatoração de f sobre K em fatores irredutíveis. Neste trabalho, iremos melhorar a complexidade do passo 3. Para tanto, vamos representar cada subcorpo L de K/k através de uma partição P_L do conjunto $\{1, \ldots, r\}$, onde cada i corresponde ao fator irredutível f_i de f sobre K. A partição P_L é definida da seguinte forma: sobre L, f possui uma fatoração em fatores irredutíveis, digamos $g_1, \ldots, g_s$. Cada g_j é o produto de alguns f_i's. Assim, definimos $P_L = \{\{i : f_i \mid g_j\}, j = 1, \ldots, s\}$. No exemplo abaixo, f possui cinco fatores irredutíveis sobre $k(\alpha)$ e dois fatores irredutíveis sobre L, definindo assim a partição P_L.

$$
\begin{array}{ccc}
k(\alpha) & f = f_1 \cdot f_2 \cdot f_3 \cdot f_4 \cdot f_5 \longrightarrow P_K = \{\{1\},\{2\},\{3\},\{4\},\{5\}\} \\
\vert & \uparrow & \\
L & f = (f_1 f_2 f_3) \cdot (f_4 f_5) \longrightarrow P_L = \{\{1,2,3\},\{4,5\}\} \\
\vert & \uparrow & \\
k & f = (f_1 f_2 f_3 f_4 f_5) \longrightarrow P_k = \{\{1,2,3,4,5\}\}
\end{array}
$$

A partir deste exemplo, é fácil notar que $L \subseteq L'$ se, e somente se, $P_{L'}$ refina P_L. O principal resultado que demonstramos neste trabalho é dado a seguir.

Teorema 3.26. *Sejam L, L' subcorpos de K/k e sejam P_L e $P_{L'}$ suas correspondentes partições. Então $P_{L \cap L'} = P_L \vee P_{L'}$.*

A partição $P_L \vee P_{L'}$ chama-se *junção* de P_L e $P_{L'}$ e é definida como a partição mais fina que é refinada tanto por P_L quanto por $P_{L'}$. Além disso, essa partição pode ser calculada utilizando-se o algoritmo dado por Freese [19, 20]. Assim, dados dois subcorpos principais L_i, L_j, ao invés de calcularmos diretamente a interseção $L_i \cap L_j$ utilizando-se Álgebra Linear (passo 1 no esquema abaixo), iremos primeiro calcular as respectivas partições P_{L_i}, P_{L_j} (passo 2), calcular a junção

$P_{L_i} \vee P_{L_j}$ (passo 3) e finalmente, calcular um gerador para o subcorpo $L_i \cap L_j$ a partir da partição $P_{L_i} \vee P_{L_j}$ (passo 4).

$$\begin{array}{ccc} L_i, L_j & \xrightarrow{(2)} & P_{L_i}, P_{L_j} \\ \downarrow{\scriptstyle (1)} & & \downarrow{\scriptstyle (3)} \\ L_i \cap L_j & \xleftarrow[(4)]{} & P_{L_i} \vee P_{L_j} \end{array}$$

Mais detalhes sobre os passos (2), (3) e (4) deste esquema são apresentados no Capítulo 3. Para $k = \mathbb{Q}$, este esquema nos permite demonstrar o seguinte resultado.

Teorema 3.47. *Seja m o número total de subcorpos de K/k. Quando $k = \mathbb{Q}$, podemos calcular todos os subcorpos de K/k (em termos de partições) com um custo esperado de $\tilde{\mathcal{O}}(rn^7 + rn^5 \log^2 \|f\|_2 + mr^2)$ operações de bit, onde r é o número de fatores da subfield factorization e n é o grau da extensão.*

Utilizar partições para calcular interseções também melhora o tempo computacional, especialmente nos casos onde há um grande número de subcorpos (veja Tabela 4.2). Quando $k = \mathbb{Q}$, van Hoeij *et al.* [51] também apresentam um método para calcular os subcorpos principais utilizando o algoritmo LLL [35], evitando assim calcular a fatoração de f sobre $\mathbb{Q}(\alpha)$. Utilizando estas ideias, podemos calcular uma *subfield factorization* para $\mathbb{Q}(\alpha)/\mathbb{Q}$ e também mostramos como podemos melhorar este passo (Remark 4.6), reduzindo o número de chamadas do algoritmo LLL. Para mais detalhes, veja o Capítulo 4.

Finalmente, passamos a nos concentrar em extensões definidas por funções racionais. Isto é, se $f(t) \in K(t)$, com K um corpo qualquer, é uma função racional, então $K(f(t))$ é um subcorpo de $K(t)$. Assim, temos definida uma extensão $K(f(t)) \subset K(t)$ de grau finito. Subcorpos desta extensão estão em bijeção com as decomposições de f, ou seja, L é um subcorpo de $K(t)/K(f(t))$ se, e somente se, existe $h(t) \in K(t)$ tal que $L = K(h(t))$ e $f = g \circ h$, para algum $g(t) \in K(t)$.

Assim, para encontrar todas as decomposições de $f(t) \in K(t)$, calcularemos todos os subcorpos de $K(t)/K(f(t))$. Para $K(t)$, o elemento $t \in K(t)$ é primitivo e seu polinômio minimal sobre $K(f(t))$ é dado por[1] $\Phi_f := f_n(x) - f(t)f_d(x) \in K(f(t))[x]$, onde $f_n(x), f_d(x) \in K[x]$ são coprimos e $f(t) = f_n(t)/f_d(t)$. Sejam $F_1, \ldots, F_r$ os fatores irredutíveis de Φ_f sobre $K(t)$ e defina

$$L_i := \{g(t) \in K(t) : F_i \mid \Phi_g\},\ i = 1, \ldots, r. \tag{1}$$

Teoremas 5.11 e 5.12. *Seja $f(t) \in K(t)$ e sejam $F_1, \ldots, F_r$ os fatores irredutíveis de Φ_f sobre $K(t)$. Então o conjunto $\{L_1, \ldots, L_r\}$, com L_i definido em* (5.17)*, é o conjunto dos subcorpos principais de $K(t)/K(f(t))$.*

Dada esta descrição dos subcorpos principais, utilizamos partições para calcular todas as interseções entre os subcorpos principais, o que simplifica significativamente o cálculo destas interseções. Para calcular um gerador para cada subcorpo L de $K(t)/K(f(t))$, dado pela partição P_L, demonstramos o seguinte resultado.

Teorema 5.26. *Seja $f(t) \in K(t)$ e sejam $F_1, \ldots, F_r$ os fatores irredutíveis de Φ_f sobre $K(t)$, com $F_1 = x - t$. Seja $P_L = \{P^{(1)}, \ldots, P^{(s)}\}$ a partição correspondente ao subcorpo L. Seja $g := \prod_{i \in P^{(1)}} F_i$, com $1 \in P^{(1)}$ e seja $c(t) \in K(t)$ um coeficiente não constante de g. Então $L = K(c(t))$.*

Estes resultados permitem encontrar todas as decomposições de uma função racional $f(t) \in K(t)$ de forma mais eficiente e com uma melhor complexidade (veja Corolário 5.25 e Tabelas 5.1 e 5.2). Além disso, no caso de $f(t) \in K[t]$, nosso algoritmo possui uma melhor complexidade do que o algoritmo apresentado por Blankertz [10]. A implementação deste algoritmo de decomposição de funções racionais foi incluida no Sistema Algébrico Computacional *Magma*. Para mais detalhes, veja o Capítulo 5.

[1] Aqui podemos sempre assumir que $\deg(f_n(x)) > \deg(f_d(x))$, o que garante que Φ_f é mônico.

AGRADECIMENTOS

First of all, I must thank my family, for all their support and unconditional love. Being away from home is not always easy, but you understood and supported my choice. I must also thank my girlfriend, Carla, for her patience and for always being by my side. I would not have come this far without your support.

Secondly, I must thank my Ph.D. advisor, Prof. Vilmar Trevisan. Since my undergraduate days, Prof. Trevisan has been encouraging and advising me on my academic path. It was also thanks to Prof. Trevisan that I came in contact with the field of Computer Algebra. I must also thank my co-advisor Prof. Luiz Emilio Allem, for giving good advice and for always pointing out opportunities, such as a Travel Award from my university, that allowed me to get in contact and to work with Prof. Mark van Hoeij, at Florida State University (FSU).

Speaking of which, I must heartily thank Prof. Mark van Hoeij. Prof. van Hoeij kindly received me twice at Florida State University, for a short visit in 2013 and later, a whole year in 2015, as part of my Ph.D. program. It was Prof. van Hoeij who suggested this topic and much of what is presented in this dissertation was done under his supervision. I am confident to say that this dissertation would not exist without Prof. van Hoeij's generous contribution.

Last, but not least, I thank my friends, who made this journey a little easier and more joyful. I am also grateful for my university, for providing free and high quality education and the Brazilian funding agencies, CAPES and CNPq, for funding my studies as a graduate student and my year-abroad at FSU.

1 INTRODUCTION

In this chapter we give a basic introduction to the subject, explain what this dissertation is about and mention our contributions to this field.

1.1 Computer Algebra - Let the computer do the work!

Computer Algebra, also referred to as *Symbolic Computation* or *Algebraic Computation*, is an area of study concerning the development of algorithms and software for manipulating mathematical objects. Using the aid of computers, these algorithms have become an important tool in a researcher's repertoire, especially if one has to perform lengthy algebraic computations, which could not be easily done using pencil and paper.

Unlike numerical computation, symbolic computation is exact (no rounding occurs) and one can manipulate mathematical objects involving symbols and variables without attributing numerical values to them. Furthermore, algebraic solutions are usually more "compact" than numerical solutions and one might obtain more (accurate) information from an algebraic solution than from a set of (approximate) numerical solutions. However, it has to be noted that this is not always the case. An algebraic solution to a certain problem might be so complex that not much information can be obtained. Besides, there are cases where one has to solve problems that are so complex that they can not be solved through algebraic or analytic methods (even on a computer), and what remains is - hopefully - a numerical approach.

A piece of software that performs algebraic manipulations is called a *Computer Algebra System*. One of the goals of a Computer Algebra System is to automate long, tedious and (more often than not) difficult algebraic manipulation

tasks. *Maple* and *Mathematica* are two well-known examples of Computer Algebra Systems, with a friendly graphical user interface and capable of solving a wide variety of tasks. Other Computer Algebra Systems can perform more specialized tasks. One such example is *Magma* [11], which focuses on Algebra, Number Theory, Algebraic Geometry and Algebraic Combinatorics, and on which all the algorithms presented in this dissertation were implemented. Another example is *SageMath* [46], an open-source Computer Algebra System which encompasses many existing open source packages for several areas of research, and whose mission is to create "a viable free open source alternative to Magma, Maple, Mathematica and Matlab".

It is not difficult to see that a Computer Algebra System is incredibly helpful for someone who has to perform symbolic computations. In fact, this research field was born by the need of theoretical physicists to perform such operations. In 1967, the Dutch theoretical physicist Martinus J. G. Veltman created the *SchoonSchip*, which is considered one of the first Computer Algebra System for use in particle physics (see [59] for more details).

From solving a quadratic polynomial equation to very specific algebraic computations, Computer Algebra Systems have evolved to solve a wide variety of problems and are now being used by researchers from several distinct fields. Teaching in universities also benefits from Computer Algebra Systems. For instance, some concepts and geometrical properties from Calculus can be easily visualized in a computer, helping students better understand the inner workings of Mathematics.

Moreover, as we shall see in Section 1.2.2, Computer Algebra Systems can not only find solutions to a certain problem, but also present them in a simpler form, which may lead to a better understanding of the whole theory. That is, Computer Algebra Systems can aid in research, teaching as well as in the real world applications. Thus, improving the algorithms used by a Computer Algebra System is of great importance.

1.2 The Subfield Lattice Problem

1.2.1 What is this Dissertation about?

Computer Algebra Systems can solve a wide variety of problems. One such problem is to find subfields of a field extension (precise definitions are given in the next chapter). That is, given a field extension K/k, we want to find fields L such that $k \subseteq L \subseteq K$. As we shall see in Subsection 1.2.2, finding such subfields has several applications.

In this dissertation we are interested in finding *all* subfields L of K/k, as well as their inclusion relations. This is often referred to as the *Subfield Lattice problem*. Moreover, we are interested in devising an algorithm to perform this task. In addition to giving a general algorithm, we shall consider two particular cases: number fields and rational function fields.

As we shall see in Chapter 2, finding subfields of a field extension is equivalent to finding certain subgroups of the Galois Group of this extension. Finding the Galois Group, and hence, the subfield lattice, is a classical problem and as such, there are already several results on this matter. In Chapter 2 we give some basic definitions that will be used throughout this work. We also briefly explain previous methods to compute the subfield lattice via Galois Theory.

Another method to find the subfield lattice was presented by [51]. This method focuses on finding a set of $r \leq n$ *principal* subfields, where n is the degree of the extension. These special subfields have the property that any other subfield of K/k is the intersection of some of them. In Chapter 3 we propose an improvement for this algorithm and in Chapter 4 we pay close attention to the number field case.

In Chapter 5 we focus on the case where the field extension is a rational function field extension $K(t)/K(f(t))$, for some rational function $f(t) \in K(t)$. As

we shall see, the subfield lattice of $K(t)/K(f(t))$ is closely related to the rational decompositions of $f(t) \in K(t)$. Hence, as a byproduct, we get an algorithm to compute all decompositions of a rational function.

1.2.2 Applications

Finding the subfield lattice is an interesting problem on its own right. However, there are several cases where finding subfields of a field extension can give us additional information or can even simplify our computations. In what follows, we briefly mention two problems that can be solved/simplified by computing subfields.

The first application we mention is the computation of rational function decompositions. Given a rational function $f(t) := f_n(t)/f_d(t) \in K(t)$, where $f_n(t), f_d(t) \in K[t]$ are coprime, we wish to find rational functions $g(t) = g_n(t)/g_d(t) \in K(t)$ and $h(t) = h_n(t)/h_d(t) \in K(t)$, where $g_n, g_d, h_n, h_d \in K[t]$, such that

$$f(t) = g \circ h = \frac{g_n\left(h(t)\right)}{g_d\left(h(t)\right)}.$$

In this case, we say that $g \circ h$ is a decomposition of f. In order to find g and h as above, consider the (rational function) field extension defined by f, denoted by $K(t)/K(f(t))$. Let L be a subfield of $K(t)/K(f(t))$. By Lüroth's Theorem (see [41] or [48] for more details), there exists a rational function $h(t) \in K(t)$ such that $L = K(h(t))$. Furthermore, since $K(f(t)) \subseteq L = K(h(t))$ and hence, $f(t) \in K(h(t))$, there exists $g(t) \in K(t)$ such that $f = g \circ h$. That is, we have found a decomposition of $f(t)$. Conversely, if $f = g \circ h$, with $g, h \in K(t)$, then $L := K(h(t))$ is such that $K(f(t)) \subseteq L \subseteq K(t)$. Thus, we see that there exists a relation between the decompositions of $f(t) \in K(t)$ and the subfields of the extension $K(t)/K(f(t))$, and that finding these subfields gives us decompositions of f.

Notice that polynomial decomposition is just a particular case of rational function decomposition. Polynomial decomposition allows us to express roots

of simple polynomials in terms of radicals. For instance, consider the polynomial $f = x^4 - 8x^3 + 18x^2 - 8x + 2$. The polynomial f has the following decomposition

$$f = (x^2 + 1) \circ (x^2 - 4x + 1).$$

If a is a zero of $x^2 + 1$ and if b is a root of $x^2 - 4x + 1 = a$, then b is a zero of f. The zeros of $x^2 + 1$ are $\pm i$. By solving $x^2 - 4x + 1 = \pm i$, we get the 4 zeros of f

$$2 + \sqrt{3 + i}, \;\; 2 + \sqrt{3 - i}, \;\; 2 - \sqrt{3 + i} \quad \text{and} \quad 2 - \sqrt{3 - i}.$$

The second application we mention is the simplification of algebraic numbers, which in turn can be used to simplify the solutions of a polynomial system. Consider the following polynomial system

$$\begin{cases} x^2 - 2xy + y^2 - 8 = 0, \\ x^2y^2 - (x^2 + 2x + 5)y + x^3 - 3x + 3 = 0. \end{cases}$$

Many Computer Algebra Systems, such as *Maple* and *Mathematica*, will reduce this problem to a univariate problem, by giving the solutions in terms of the roots of some univariate polynomial of higher degree. For instance, the command `solve` in *Maple* returns the solution $\{x = a, y = b\}$, where a is a root of

$$f = x^8 - 20x^6 + 16x^5 + 98x^4 + 32x^3 - 12x^2 - 208x - 191 \tag{1.1}$$

and b is given in terms of a by

$$b = -\frac{17}{1809}a^7 + \frac{61}{3618}a^6 + \frac{371}{1809}a^5 - \frac{1757}{3618}a^4 - \frac{563}{603}a^3 + \frac{6013}{3618}a^2 + \frac{3184}{1809}a + \frac{7175}{3618}. \tag{1.2}$$

Computationally speaking, this solution is "as good as any". However, a much simpler solution exists, given by

$$a = \sqrt{3} + \sqrt[4]{2} - \sqrt{2} \quad \text{and} \quad b = \sqrt{3} + \sqrt[4]{2} + \sqrt{2}. \tag{1.3}$$

Obviously, the solution in (1.3) is more pleasing to the eye than the solution presented in (1.1) and (1.2). So the question is, how do we find the solution in (1.3)?

First, consider the field extension $\mathbb{Q}(a)/\mathbb{Q}$, where a is a root of f. This extension has subfields $\mathbb{Q}(\alpha)$ and $\mathbb{Q}(\beta)$, where α satisfies $\alpha^2 - 3 = 0$ and β satisfies $\beta^4 - 2 = 0$ (we might consider $\alpha = \sqrt{3}$ and $\beta = \sqrt[4]{2}$). Moreover, one can show that $\mathbb{Q}(a) = \mathbb{Q}(\alpha, \beta)$. That is, $a \in \mathbb{Q}(\alpha, \beta)$ and we can write a in terms of α and β. By substituting this expression for a in (1.2) and simplifying, we get the expression for b in (1.3).

Simplifying an algebraic expression, as in the previous example, can be useful, for instance, for teaching purposes or if one has to display these results in an article. Thus, computing subfields of a field extension is an important task for solving/simplifying other problems.

1.2.3 Our Contribution

In order for computers to solve our algebraic problems efficiently and correctly, one has to devise (efficient and correct) algorithms that perform such tasks. As mentioned above, this dissertation focuses on the problem of computing all subfields of a field extension. Several algorithms already exist for finding subfields of a field extension. In this dissertation we make improvements on an algorithm presented by van Hoeij *et al.* [51].

The main improvement relies on the representation of each subfield. While a subfield can be viewed as a vector space, we represent it by a partition of $\{1, \ldots, r\}$, where $r \leq n$ and n is the degree of the extension. This has several advantages, as we shall see in Section 3.1.3 (more specifically, Remark 3.14). In the worst case, our algorithm performs similarly as the original. However, when the number of subfields is high, we see a big improvement. This leads to a faster algorithm in general, with significant improvement when the field extension has a high number of subfields. Since the number of subfields is not polynomially bounded, the complexity of such algorithm can not be polynomially bounded. Nonetheless, by

using partitions, we are able to improve the non-polynomially bounded complexity term (see Theorem 3.47).

In Chapter 3, we give a general algorithm to compute the subfield lattice of a generic extension K/k. We also analyze two particular cases: number fields (Chapter 4) and rational function fields (Chapter 5). In the number field case, we are able estimate the complexity in terms of CPU operations (see Theorem 3.47). In practice, the bottleneck of the algorithm is the LLL computations. We also present a nearly trivial result (Remark 4.6) that helps reduce the number of LLL calls. This further improves the performance of our algorithm (see Table 4.2).

In the rational function field case, our algorithm is the first[1] to use *principal subfields* to find the subfield lattice of this kind of extension field. Previous methods relied on a combinatorial approach on the factors of a certain bivariate polynomial, rendering them inefficient as the number of factors grew. Our algorithm is able to avoid this problem, being capable of solving examples in seconds while previous algorithms would take hours. Moreover, the subfield lattice of a rational function field extension $K(t)/K(f(t))$, where $f(t) \in K(t)$ is a rational function, is closely related to the set of non-equivalent complete decompositions of $f(t)$. When $f(t) \in K(t)$ is a rational function, finding a decomposition of $f(t)$ means finding $g(t), h(t) \in K(t)$ such that $f = g \circ h$. Hence, by finding the subfield lattice of $K(t)/K(f(t))$, we are able to give an efficient algorithm to find all rational decompositions of $f(t)$ (see Table 5.1 for a comparison with a previous method).

Polynomial decomposition is another classical topic in Computer Algebra, with contributions from several authors. Given $f \in K[t]$, one wants to find $g, h \in K[t]$ such that $f = g \circ h$. When $f(t) \in K[t]$ is a polynomial, our algorithm gives all (polynomial) decompositions of $f(t)$. Polynomial decomposition is often split into two cases: the *tame* case (when the characteristic of the field K does not

[1] to the best of our knowledge.

divide $\deg(g)$) and the *wild* case (when the characteristic of K does divide $\deg(g)$). In the tame case there are already very efficient algorithms to solve this problem, e.g. [55], which finds a (single) decomposition with $\mathcal{O}(n \log^2 n \log\log n)$ field operations. In the wild case, however, our algorithm has better complexity than a previous method from Blankertz [10].

The results in this dissertation are available in the article [44], accepted for publication in the *Journal of Symbolic Computation*, and in the conference paper [3], presented at the 42*nd* International Symposium on Symbolic and Algebraic Computations (*ISSAC*'17). The first article encompasses Chapters 3 and 4, while the results in Chapter 5 can be seen in the conference paper. Another contribution of our work is that the implementation (in *Magma* [11]) of both algorithms, for computing the subfield lattice of a number field and the decompositions of rational functions, which are freely available to the scientific community at `http://www.math.fsu.edu/~jszutkos/MySubfields` and `http://www.math.fsu.edu/~jszutkos/Decompose`, respectively. Moreover, the implementation of the function decomposition algorithm was added to Magma, and is available in all releases beginning at v2.23.1.

1.3 Further Remarks

We make some further remarks before jumping into the fun part. We shall mention some techniques that will be used throughout this work. We also give some details about the complexity we use, that is, we give the complexity of the operations we will be using in our algorithms.

1.3.1 Techniques Used

In order to compute the subfield lattice, several techniques from Computer Algebra are used. We do not intend to explain all these techniques in great detail. However, we do feel obligated to (at least) briefly mention them.

Let K/k be a field extension. If this extension is finite and separable, we always have an associated polynomial $f \in k[x]$ (the minimal polynomial of a primitive element of K) and more often than not, we will be interested in the *factorization* of $f \in k[x]$ over K (or even an extension of K). That is, given $f \in k[x] \subseteq K[x]$, we want to find polynomials $f_1, \ldots, f_r \in K[x]$ such that

$$f = f_1 \cdots f_r. \tag{1.4}$$

The polynomials f_i, $1 \leq i \leq r$, are called factors of f. A polynomial f is said to be *irreducible* over K if f is not a constant and its only (non-constant) factor over K is f. The factorization in (1.4) is an *irreducible factorization* if all f_i are irreducible.

When $k = \mathbb{Q}$, factorization over a finite extension of k can be done using Trager's algorithm [47] or the algorithm proposed by Belabas [8], which is a generalization of van Hoeij's factorization algorithm [49] to the number field case. Factorization over $\mathbb{Q}$ can be computed using van Hoeij's algorithm [49] (see also [27]). For factorization algorithms over finite fields, see [58] and [57, Chapter 14].

Let $f \in R[x]$, where R is a commutative ring with unity. Let $\mathfrak{m} \subset R$ be an ideal and let $f_1, f_2 \in R[x]$ such that $f \equiv f_1 f_2 \bmod \mathfrak{m}$. We might be interested in the "lifted" factorization of f modulo $\mathfrak{m}^2$. That is, we want to find $\hat{f}_1, \hat{f}_2$ such that $f \equiv \hat{f}_1 \hat{f}_2 \bmod \mathfrak{m}^2$, $\hat{f}_1 \equiv f_1 \bmod \mathfrak{m}$ and $\hat{f}_2 \equiv f_2 \bmod \mathfrak{m}$. This process is often referred to as *Hensel Lifting*. For further details, see [57, Chapter 15]. Hensel Lifting will be used in the following context. Let $f \in \mathbb{Q}[x]$ and let p be a prime number (not dividing the denominator of the coefficients of f). We will be interested in the factorization of f over $\mathbb{Q}_p$, the p-adic completion of $\mathbb{Q}$. That is, we want to find

$\hat{f}_1, \ldots, \hat{f}_r \in \mathbb{Q}_p[x]$ such that $f = \hat{f}_1 \cdots \hat{f}_r$. However, like factorization over $\mathbb{R}$ or $\mathbb{C}$, one cannot always compute the factors $\hat{f}_i$ with *infinite precision*. This is where Hensel Lifting is used. If $\bar{f}_1, \ldots, \bar{f}_r$ are the irreducible factors of f over $p\mathbb{Z}$, then $\hat{f}_i \equiv \bar{f}_i \bmod p$ and for a given $a \in \mathbb{N}$, using Hensel Lifting, we can compute factors $f_1^{(a)}, \ldots, f_r^{(a)} \in p^a\mathbb{Z}$ such that

$$f \equiv f_1^{(a)} \cdots f_r^{(a)} \bmod p^a \tag{1.5}$$

and $\hat{f}_i \equiv f_i^{(a)} \bmod p^a$. Moreover, the factorization in (1.5) is said to be an approximation with *accuracy* (or *precision*) a of the factorization $\hat{f}_1 \cdots \hat{f}_r$. Working with $f_i^{(a)}$, if the accuracy a is high enough, is often sufficient.

Another paramount technique from Computer Algebra we shall frequently use is the LLL algorithm, which computes a *reduced basis* for a given lattice. Originally, this algorithm was used to present the first polynomial time algorithm for polynomial factorization over $\mathbb{Q}$. However, the LLL algorithm found numerous applications. We shall briefly mention how this technique works. Let $v_1, \ldots, v_n \in \mathbb{R}^n$ be linearly independent. The *lattice* generated by $v_1, \ldots, v_n$ is the set ($\mathbb{Z}$-modulo)

$$\mathcal{L} = \{c_1 v_1 + \cdots + c_n v_n \in \mathbb{R}^n : c_1, \ldots, c_n \in \mathbb{Z}\},$$

and $v_1, \ldots, v_n$ are said to be a basis for $\mathcal{L}$. A natural question is to compute a *shortest vector* in a given lattice $\mathcal{L}$ (i.e., v_0 is a shortest vector in $\mathcal{L}$ if $v \in \mathcal{L}$ and $\|v\| < \|v_0\|$ then $v = \mathbf{0}$). This is also an important question, as many problems can be re-stated as finding a shortest vector inside some particular lattice. However, computing a shortest vector v_0 is $\mathcal{NP}$-hard (see Ajtai [1]).

Let $\mathcal{L}$ be a lattice generated by $v_1, \ldots, v_n$ and let $v_1^*, \ldots, v_n^*$ be its Gram-Schmidt orthogonal basis. Then one can show that $\|v\| \geq \min\{\|v_1^*\|, \ldots, \|v_n^*\|\}$, for any non-zero $v \in \mathcal{L}$ (see [57, Lemma 16.7]). That is, if the vectors $v_1^*, \ldots, v_n^*$ are inside $\mathcal{L}$, then one of the v_i^* is a shortest vector of $\mathcal{L}$. Unfortunately, the vectors $v_1^*, \ldots, v_n^*$ are not usually inside $\mathcal{L}$. We say that a basis $v_1, \ldots, v_n$ of $\mathcal{L}$ is *reduced* if

the vectors $v_1^*, \ldots, v_n^*$ satisfy

$$\|v_i^*\|^2 \leq 2\|v_{i+1}^*\|^2, \;\; i = 1, \ldots, n-1.$$

Thus, if $v_1, \ldots, v_n$ is a reduced basis, then $\|v_1\| \leq 2^{(n-1)/2}\|v\|$, where v is any non-zero vector of $\mathcal{L}$ (see [57, Theorem 16.9]). That is, if we can compute a reduced basis, then we have a vector $v_1 \in \mathcal{L}$ which is at most $2^{(n-1)/2}$ times larger than a shortest vector of $\mathcal{L}$. For many problems, we can design a lattice in such a way that the solution to our problem is encoded in a *relatively short vector* inside this lattice. Hence the importance of the LLL algorithm: it allows us to compute a reduced basis (and hence, *relatively short* vectors) in polynomial time. Fortunately, for most applications, this will suffice.

Let us see an example. Let $\phi \in \mathbb{R}$ be a real number. Suppose that we, somehow, have an approximation $r = 1.618034$ of ϕ. Moreover, suppose we know that ϕ is the root of a quadratic polynomial $f \in \mathbb{Z}[x]$ with coefficients bounded by 10 in absolute value. How do we find f, knowing only this little information? Consider the lattice $\mathcal{L}$ generated by $v_1 = [1, 0, 0, Cr^2]$, $v_2 = [0, 1, 0, Cr]$ and $v_3 = [0, 0, 1, C]$, where $C = 10000$. A random element in the lattice $\mathcal{L}$ has the form $[a, b, c, C(ar^2+br+c)]$, for $a, b, c \in \mathbb{Z}$. When a, b, c are such that $f = ax^2+bx+c$, then $C(ar^2+br+c) \approx 0$, and the vector $[a, b, c, C(ar^2+br+c)]$ is relatively short (i.e., has a relatively small norm compared to vectors which do not correspond to the solution). The LLL algorithm is able to find the vector $[a, b, c, C(ar^2 + br + c)]$, provided the constant C is appropriately chosen. By applying LLL to the vectors v_1, v_2 and v_3 above, we obtain the basis $b_1 = [1, -1, -1, 0.000250]$, $b_2 = [-7, 41, -48, 131.558249]$ and $b_3 = [-11, 66, -78, -81.302750]$. Among these vectors, b_1 gives a polynomial $f = x^2 - x - 1$, which satisfies the coefficient bound given above. In this case, because of the information given, we are able to prove that ϕ is a root of $x^2 - x - 1$.

The constant C in the previous example was used to separate vectors corresponding to solutions from other vectors, and depends on how good the approx-

imation r is. For the LLL algorithm to work, we need to ensure that the vectors we do not want are at least $2^{\frac{n-1}{2}}$ times bigger than the vectors we do want. If we can construct a lattice with this property, then the short vectors in a reduced basis will span the solutions of our problem. If v_0 is a shortest vector in $\mathcal{L}$, then $\|v_0\|^n \leq |\det(L)|$, where L is the $n \times n$ matrix whose columns are the vectors $v_1, \ldots, v_n$ forming a basis of $\mathcal{L}$ (when the lattice has rank $< n$, we use $\sqrt{|\det(L^T L)|}$). Hence, if we want $v_S \in \mathcal{L}$ to be a shortest vector, than we need to, at least, ensure that $\|v_S\|^n \leq \det(L)$. This can be used as a starting point for choosing the constant C.

The LLL algorithm was devised by Lenstra, Lenstra and Lovász [35], and it was used to give the first polynomial time algorithm for factorization of polynomials over $\mathbb{Q}$. Several improvements have since been made. We cite [53], who presented an algorithm `LLL_with_removals`, which returns a basis of a sublattice of the original lattice, which still contains the desired vectors.

1.3.2 On the Complexity of the Algorithms

Throughout this dissertation, we shall mention different types of algorithms. A *deterministic algorithm* will produce the same output every time it is executed (with the same input, of course). A deterministic algorithm can be compared to a mathematical function, which associates a unique value for every element of its domain. A *randomized algorithm*, on the other hand, uses randomness as part of its logic. Usually, randomized algorithms are faster (on average) than the corresponding deterministic versions (when they exist). However, they might not always produce a correct output or even terminate. It is important to distinguish between *Monte Carlo* algorithms, which are randomized algorithms where the output has a (usually small) probability of being wrong, and *Las Vegas* algorithms, which also use randomness, but whose output is always correct. We shall present deterministic al-

gorithms and, whenever possible, a randomized version. The randomized algorithms in this dissertation are all of the *Las Vegas* type.

Whenever we write an algorithm, we are also often interested in its time complexity, which quantifies the number of operations the algorithm executes in terms of the input size. This number is given using the "Big Oh" notation $\mathcal{O}$, which is more concerned with the "rate of growth" rather than the precise number of operations performed by the algorithm. That is, $f(n) \in \mathcal{O}(g(n))$ if there exists $N \in \mathbb{N}$ and $c \in \mathbb{N}$ such that $f(n) \leq cg(n)$, for all $n \geq N$. For the sake of simplicity, we will often use the *soft-*$\mathcal{O}$ notation $\tilde{\mathcal{O}}$, which ignores logarithmic factors. More precisely, $f(n) \in \tilde{\mathcal{O}}(g(n))$ if there exists $k \in \mathbb{N}$ such that $f(n) \in \mathcal{O}(g(n) \log^k g(n))$. This might be useful for simplifying results. For more details, see [57, Section 25.7].

In what follows we list the complexity of basic operations we shall use in our algorithms. This should help us when analyzing the complexity of our algorithms. When $a, b \in \mathbb{Z}$ are bounded by B in absolute value, arithmetic operations $(+, -, \times, \div)$ and the equality test can be computed with $\mathcal{O}(\log B)$ CPU operations. If $a, b \in \mathbb{Q}$, with numerators and denominators bounded by B, then the operations $(+, -, \times, \div)$ can also be computed with $\mathcal{O}(\log B)$ CPU operations. For a finite field $\mathbf{F}_p$ of characteristic p, any element $a \in \mathbf{F}_p$ is bounded by p in absolute value. Hence, any arithmetic operation in $\mathbf{F}_p$ costs $\mathcal{O}(\log p)$ CPU operations.

Let K be a field and let $M(n)$ be a *multiplication time* for $K[x]$, that is, given two polynomials $f, g \in K[x]_{<n}$, we can compute $f \cdot g$ with $M(n)$ arithmetic operations in K. Classical algorithms have $M(n) = 2n^2$ and using *Fast Fourier Transform* multiplication yields $M(n) = n \log n$ (see [24]). We recall that the function M is *super-additive*: $M(n_1) + M(n_2) \leq M(n_1 + n_2)$, see [57, Chapter 8.3]. Furthermore, we can compute $\gcd(f, g) \in K[x]$ with $\mathcal{O}(M(n) \log n)$ arithmetic operations in K, see [57, Chapter 10]. Moreover, if $f \in K[x]$ is irreducible with degree n, then arithmetic in $K[x]/\langle f(x) \rangle$ costs $\mathcal{O}(M(n))$ operations in K (see [57, Chapter

11]). To give the complexity of these operations in terms of CPU operations one would need a bound for all coefficients involved in these operations, which is usually a hard task. Finally, given a linear system $\mathcal{S}$, with m equations in r variables defined over K, we can compute a basis of solutions of $\mathcal{S}$ with $\mathcal{O}(mr^{\omega-1})$ field operations [9, Chapter 2], where $2 < \omega \leq 3$ is a *feasible matrix multiplication exponent*. A number ω is a *feasible matrix multiplication exponent* if we can multiply two $n \times n$ matrices with $\mathcal{O}(n^\omega)$ operations. The classical algorithm shows that $\omega = 3$ is feasible. The smallest known feasible matrix multiplication exponent is $\omega = 2.3728$ [33].

Given $f \in \mathbb{Z}[x]$ of degree n, $\|f\|_\infty \leq p^l$, and a factorization of $f \bmod p$ given by $f \equiv f_1 \cdots f_r \bmod p$, we can compute (see [57, Theorem 15.18]) the lifted factorization of $f \bmod p^l$ with complexity

$$\mathcal{O}((M(n)M(l\mu) + M(n)\log nM(\mu) + nM(\mu)\log\mu)\log r),\ \mu = \log p.$$

For the precision l given in Remark 2.16 and using fast arithmetic, this complexity can be restated as $\tilde{\mathcal{O}}(n^3 + n^2 \log \|f\|))$ CPU operations. We will also need the complexity of an LLL call. In [53, Theorem 6], the complexity of one call of the algorithm `LLL_with_removals`, is given by

$$\mathcal{O}((r+N)c^3(c+\log B)[\log X + (r+N)(c+\log B)]),$$

where r, c, N, B and X are parameters of the lattice. For a given factor f_i of f and the lattice we will be using in Chapter 3, these parameters have values $r = n = \deg(f), N = d_i = \deg(f_i), c = n + d_i$, X is a bound for the entries in the matrix that contains a basis of the lattice and is given by p^l (where l is the accuracy from Remark 2.16), and B is an upper bound for the vectors in the desired sublattice, which is given by $B = n^2\|f\|$ (see Theorem 2.15). By substituting these values and simplifying, we get that the complexity for one LLL call (for our particular case) is given by $\mathcal{O}(n^7 + n^5 \log^2 \|f\|)$ CPU operations.

2 BASIC DEFINITIONS AND KNOWN RESULTS

In this chapter we give some basic definitions. We also recall one of the main result of Galois Theory, which gives a bijection between the set of subfields of a field extension and subgroups of the Galois Group of this extension. More details regarding fields, groups and the Galois Theory can be found in most Abstract Algebra books (e.g., [42]). We also mention previous approaches to compute the subfield lattice of a field extension K/k of finite degree. We shall mainly focus on two approaches, the first for its direct connection with Galois Theory, and the second which is the method we shall improve in this dissertation.

The first approach explicitly uses one of the main results given by Galois Theory, that the subfield lattice is in bijection with subgroups of its Galois group. This is presented in Section 2.2.2. All results presented in this section are proved in Dixon [17], Klüners & Pohst [28] or in the references therein. The second approach is presented in Section 2.4. This approach searches for a set of *intersection-generating subfields* of K/k. These special subfields have the important property that any subfield of K/k is the intersection of some of these subfields. This was first presented by van Hoeij *et al.* [51]. Other approaches are briefly mentioned in Section 2.3.

Although we do not use the first approach (even though some ideas from both methods are recurrent throughout this work), its direct connection with Galois Theory is worth mentioning. The remaining chapters are devoted to improving the method from van Hoeij *et al.*[51].

2.1 Basic Definitions

In this section we recall some basic definitions regarding fields, field extensions and Group Theory that will be used throughout the entire work.

2.1.1 Fields

We begin by formally defining a *field*. In order to do so, let us first define a *ring*. A *ring* $(R, +, \cdot)$ is an algebraic structure composed of a nonempty set R, equipped with binary operations of *addition* $+ : R \times R \to R$ and *multiplication* $\cdot : R \times R \to R$, such that

1. $\forall\ a, b, c \in R,\ (a + b) + c = a + (b + c)$ and $(a \cdot b) \cdot c = a \cdot (b \cdot c)$.
2. $\forall\ a, b \in R,\ a + b = b + a$ and $a \cdot b = b \cdot a$.
3. $\exists\ 0 \in R$ and $\exists\ 1 \neq 0 \in R$ such that $a + 0 = a$ and $a \cdot 1 = a$, $\forall\ a \in R$.
4. $\forall\ a \in R,\ \exists\ a' \in R$ such that $a + (a') = 0$.
5. $\forall\ a, b, c \in R,\ a \cdot (b + c) = a \cdot b + a \cdot c$.

As usual, we denote the product of two elements $a, b \in R$ simply by ab. The element $a' \in R$ in property 4 is called the *additive inverse* of $a \in R$, and is commonly denoted by $-a$. When the operations of addition and multiplication are clear, we shall denote the ring $(R, +, \cdot)$ simply by R.

An *ideal* $\mathfrak{p}$ of R is a non-empty subset of R such that $x - y \in \mathfrak{p}$, for every $x, y \in \mathfrak{p}$ and $ab \in \mathfrak{p}$, for every $a \in \mathfrak{p}$ and $b \in R$. Moreover, for $s \subset R$, we define the ideal $\langle s \rangle$ *generated by* s as the set of all sums of products of elements of s with coefficients in R. An ideal $\mathfrak{p}$ is said to be *principal* if $\mathfrak{p} = \langle p \rangle$, for some $p \in R$. An ideal defines a equivalence relation $\sim$ on R defined by $a \sim b$ if an only if, $a - b \in \mathfrak{p}$.

The set of equivalence classes defined by this relation is called the *residue class ring*, or the *quotient ring*, of R by $\mathfrak{p}$, and is usually denoted by $R/\mathfrak{p}$.[1] Let $(R_1, +, \cdot)$ and $(R_2, \oplus, \odot)$ be rings. An *homomorphism* is a map $\phi : R_1 \to R_2$ that satisfies

a) $\forall\ a, b \in R_1,\ \phi(a \cdot b) = \phi(a) \odot \phi(b)$

b) $\forall\ a, b \in R_1,\ \phi(a + b) = \phi(a) \oplus \phi(b)$.

If the homomorphism $\phi : R_1 \to R_2$ is a bijection, then ϕ is said to be an *isomorphism*. Moreover, if $R_1 = R_2$, then $\phi : R_1 \to R_1$ is called an *endomorphism* and if $\phi : R_1 \to R_1$ is also bijective, then ϕ is an *automorphism*. If a ring $(R, +, \cdot)$ also satisfies the following property

6. $\forall\ a, b \in R$, $ab = 0$ implies $a = 0$ or $b = 0$.

then $(R, +, \cdot)$ is an *integral domain*. If $(R, +, \cdot)$ is not an integral domain, there are $a, b \in R$ such that $ab = 0$ and neither $a = 0$ nor $b = 0$. In this case, a and b are called *zero divisors* of $(R, +, \cdot)$. If an integral domain $(R, +, \cdot)$ satisfies

7. $\forall\ a \in R \setminus \{0\}$, $\exists\ a'' \in R$ such that $a \cdot a'' = 1$.

then $(R, +, \cdot)$ is a *field*. The element $a'' \in R$ in 7. is the *multiplicative inverse* of $a \in R$, and is denoted by $1/a$. Hence, a field is but a ring with no zero divisors and where every nonzero element has a multiplicative inverse. Rings, integral domains and fields are usually denoted by the letter representing their set of elements. For instance, a field $(K, +, \cdot)$ is denoted simply by K, provided the definition of the operations of addition and multiplication are clear.

Let K be a field. A *subfield* k of K is a subset of K containing the elements 0 and 1, closed under addition, multiplication and multiplicative inverses,

[1] The difference between a field extension K/k and a quotient ring $R/\mathfrak{p}$ should be clear.

and with its own operations defined by the restriction over k of the operations of addition and multiplication defined over K. A *field extension* K/k is composed of a field K and a subfield k of K. A field L is said to be a subfield of the extension K/k if $k \subseteq L \subseteq K$.

Let K/k be a field extension. The field K can be seen as a vector space over k with the induced operations. The *degree* of the extension K/k is denoted by $[K : k]$ and is defined as the dimension of the vector space K over the base field k. If $[K : k]$ is finite, then the extension K/k is said to be *algebraic*. Otherwise, K/k is *transcendental*. Furthermore, if L is a subfield of K/k, then the following *(Short) Tower Law* holds

$$[K : k] = [K : L] \cdot [L : k]. \tag{2.1}$$

Remark 2.1. *Since we will be implementing algorithms to compute subfields, it is worth mentioning how fields can be represented. We will pay close attention to algebraic number fields, that is, finite extensions of* $\mathbb{Q}$*. Every number field can be represented by* $\mathbb{Q}(\alpha)$*, for a suitable algebraic number* α*. The minimal polynomial of* α *over* $\mathbb{Q}$ *will be represented by* $f \in \mathbb{Q}[x]$*. It is well known that the field* $\mathbb{Q}(\alpha)$ *is isomorphic to* $\mathbb{Q}[x]/\langle f \rangle$*. If* $n = \deg(f)$*, then any element* $\beta \in \mathbb{Q}(\alpha)$ *can be represented as* $\beta = c_0 + c_1\alpha + \cdots c_{n-1}\alpha^{n-1}$*, for some* $c_0, c_1, \ldots, c_{n-1} \in \mathbb{Q}$*. That is,* $\mathbb{Q}(\alpha)$ *can be seen as a* $\mathbb{Q}$*-vector space with basis* $\{1, \alpha, \alpha^2, \ldots, \alpha^{n-1}\}$*. A subfield* L *of* $\mathbb{Q}(\alpha)$ *is normally given by a generator* $\beta \in \mathbb{Q}(\alpha)$*, where* $\beta = h(\alpha)$*, for some* $h(x) \in \mathbb{Q}[x]$ *(the polynomial* h *is called the embedding of* L *into* $\mathbb{Q}(\alpha)$*), and the minimal polynomial* $g \in \mathbb{Q}[x]$ *of* β *over* $\mathbb{Q}$*. We notice that two subfields might have the same minimal polynomial, hence computing the embedding is necessary to distinguish subfields. Alternatively, one can uniquely represent a subfield* L *of* $\mathbb{Q}(\alpha)$ *using the minimal polynomial* $g \in L[x]$ *of* α *over* L*. Moreover, the coefficients of the polynomial* g *generate* L *(see Theorem 3.4).*

Let K be an algebraic number field and let $\alpha \in K$. We say that α is an *algebraic integer* if α satisfies a polynomial equation $f(\alpha) = 0$, with $f \in \mathbb{Z}[x]$ monic. The *ring of integers* $\mathcal{O}_K$ of K is defined as the set of all algebraic integers of K. Moreover, if $f \in \mathcal{O}_K[x]$ is monic and g is a monic factor of f over K, then $g \in \mathcal{O}_K[x]$ (see [8, Lemma 3.1]).

Given $f(x) \in k[x]$, the *splitting field* of f is the smallest extension K of k such that $f(x)$ splits into linear factors in $K[x]$. A polynomial $f \in k[x]$ is said to be *squarefree* if $g^2 \mid f$, with $g \in k[x]$, then $g \in k$. If $f(x)$ has no repeated roots over its splitting field, then $f(x)$ is said to be *separable* (over perfect fields, separability and "squarefreeness" are equivalent). Moreover, an extension K/k is said to be *separable* if for every $\alpha \in K$, its minimal polynomial $m_\alpha(x) \in k[x]$ is separable.

For a field extension K/k, a k-automorphism of K is an automorphism $\phi : K \to K$ such that $\phi(a) = a$, for every $a \in k$. We say that ϕ fixes k. The set $\mathrm{Aut}(K/k)$ of k-automorphisms of K is very important, as we shall see below.

2.1.2 Groups

In this subsection we recall the basic definitions involving the notion of a *group*. A *group* $(G, *)$ is an algebraic structure composed of a nonempty set G and a binary operation $* : G \times G \to G$ such that

1. $\forall\, a, b, c \in G$ such that $(a * b) * c = a * (b * c)$.
2. $\forall\, a \in G$, $\exists\, e \in G$ such that $a * e = e * a = a$.
3. $\forall\, a \in G$, $\exists\, b \in G$ such that $a * b = b * a = e$.

The element e in 2. is called the *identity* of the group G. The element b in 3. is called the *inverse* of a with respect to the operation $*$ and is denoted by a^{-1}.

Furthermore, if commutativity holds, i.e., $a * b = b * a$, $\forall\ a, b \in G$, then $(G, *)$ is called an *abelian group*.

Given a subset H of G, we say that $(H, *)$ is a *subgroup* of $(G, *)$ if $(H, *)$ is itself a group. For simplicity, we also write ab instead of $a * b$ and as with fields, we usually denote a group $(G, *)$ simply by G. If H is a subgroup of G then it is common to write $H \leq G$. The *order* of a group G, denoted by $|G|$, is the number of elements of the set G. If $H \leq G$ with $|G| = n < \infty$ and $|H| = d$, then $d \mid n$ (Lagrange's Theorem, for a proof see [31]).

A classical example of a group is the *Symmetric Group* S_n of the permutations of n symbols, with *composition* as the group operation. For instance,

$$\pi = \begin{pmatrix} 1 & 2 & \cdots & n \\ s_1 & s_2 & \cdots & s_n \end{pmatrix}$$

is a representation of a permutation $\pi : \{1, \ldots, n\} \to \{1, \ldots, n\}$ such that $i \mapsto s_i$. One can also represent a permutation as a product of *cycles*. A *cycle* is a permutation that maps a subset of $\{1, \ldots, n\}$ onto itself in a cyclic fashion. That is, a cycle $r = (r_1 r_2 \cdots r_t)$, with $r_i \neq r_j$ for $i \neq j$, is a permutation that maps $r_i \mapsto r_{i+1}$, $1 \leq i \leq t - 1$ and $r_t \mapsto r_1$. The cycle $r = (r_1 r_2 \cdots r_t)$ is also called a t-cycle or a cycle of size t. It is well known that every permutation can be written as a product of disjoint cycles. For simplicity, 1-cycles are omitted. For instance,

$$\pi = \begin{pmatrix} 1 & 2 & 3 & 4 & 5 & 6 \\ 4 & 1 & 5 & 2 & 3 & 6 \end{pmatrix} = (142)(35)(6) = (142)(35).$$

Moreover, if $\pi = \pi_1 \pi_2 \cdots \pi_r$, where π_i is a cycle of size n_i, then π is said to be a permutation of *cycle type* $[n_1, \ldots, n_r]$. Let G be a group and let X be a set. The (*left*) *action* of G on X is a function $\Phi : G \times X \to X$ that satisfies

1. $\forall\ x \in X,\ \Phi(e, x) = x$, where e is the identity of G.

2. $\forall\ g, h \in G$ and $x \in X$, $\Phi(gh, x) = \Phi(g, \Phi(h, x))$.

For instance, the Symmetric Group S_n acts on the set $\{1, \ldots, n\}$ by permuting its elements. Consider a group G, a set X and an action $\Phi : G \times X \to X$. Fix $x \in X$. Two notions involving group actions are the *orbit* and the *stabilizer* of an element (or subset) of X. The *orbit* of $x \in X$ is denoted by $\mathcal{O}_x$ and defined as $\{\Phi(g, x) \ : \ g \in G\} \subseteq X$. The *stabilizer* is denoted by Stab_x and is defined as $\{g \in G \ : \ \Phi(g, x) = x\} \subseteq G$. Furthermore, for every $x \in X$, $\text{Stab}_x \leq G$.

Moreover, a group action is *transitive* if it has only one group orbit. For transitive groups G, it is interesting to notice that the action of a subgroup of G defines a partition on the set X. For instance, if $G = S_4$, then $H = \{id, (12), (34), (12)(34)\}$ is a subgroup of G, where id is the identity permutation. Moreover, the only orbits under H are $\mathcal{O}_1 = \mathcal{O}_2 = \{1, 2\}$ and $\mathcal{O}_3 = \mathcal{O}_4 = \{3, 4\}$, which determine a partition of $X = \{1, 2, 3, 4\}$.

2.2 Galois Theory

One dare not talk about subfields of a field extension or the subfield lattice of a field extension without mentioning Galois Theory. First explored by Évariste Galois in the early XIX century, this theory shows an intrinsic relation between subfields of a field extension and subgroups of the *Galois Group*.

2.2.1 The Correspondence

Let K/k be a field extension of finite degree and consider the set $G = \text{Aut}(K/k)$ of k-automorphisms of K. The set G, under composition, is a group. Furthermore, consider the group action $\Phi : G \times K \to K$ such that $\Phi(\sigma, x) = \sigma(x) \in$

K, for every $\sigma \in \mathrm{Aut}(K/k)$. Let $H \leq G$ and define the set

$$K^H := \{x \in K \ : \ \sigma(x) = x, \ \forall\, \sigma \in H\}.$$

It can be shown that the set K^H, with the induced operations, is a subfield of K/k. For $H \leq G$, the field K^H is called the *fixed field* of H. On the other hand, if L is a subfield of K/k, then the set

$$\mathrm{Aut}(K/L) = \{\sigma \in G \ : \ \sigma(x) = x, \ \forall\, x \in L\},$$

under composition, is a subgroup of G. The correspondences $H \rightsquigarrow K^H$ and $L \rightsquigarrow \mathrm{Aut}(K/L)$ are *inclusion-reversing*, that is, if $H_1 \leq H_2$ are subgroups of G, then $K^{H_2} \subseteq K^{H_1}$ and if $L_1 \subseteq L_2$ are subfields of K/k, then $\mathrm{Aut}(K/L_2) \leq \mathrm{Aut}(K/L_1)$.

An important class of field extensions are the *Galois Extensions*. A field extension K/k is said to be *Galois* or a *Galois extension* if K is the splitting field of some separable polynomial $f \in k[x]$. For instance, let $\alpha = \sqrt[4]{2} \in \mathbb{R}$ and consider the extension $\mathbb{Q}(\alpha, i)/\mathbb{Q}$, where $i^2 = -1$. Then this extension is Galois. Indeed, the polynomial $f = x^4 - 2 \in \mathbb{Q}[x]$ has roots $\alpha, -\alpha, \alpha i$ and $-\alpha i$, all of which are elements of $\mathbb{Q}(\alpha, i)$. In the case of Galois extensions, we have the following result.

Theorem 2.2 (Galois Correspondence)**.** *Let K/k be a Galois extension and let $G = Aut(K/k)$. The inclusion-reversing mappings $L \rightsquigarrow Aut(K/L)$ and $H \rightsquigarrow K^H$, between subfields of K and subgroups of G, are inverses of each other. Moreover,*

1. *$|H| = [K : K^H]$ and $[K^H : k] = [G : H]$.*
2. *K^H/k is a Galois extension if and only if $H \trianglelefteq G$.*[2]

Theorem 2.2 tells us that there is a bijection between the subfields of the extension K/k and the subgroups of $\mathrm{Aut}(K/k)$ provided, of course, that this extension is Galois. However, one often finds extensions K/k which are not Galois.

[2] *H is a normal subgroup of G, that is, $\forall g \in G, gH = Hg$.*

Let K/k be a separable field extension (not necessarily Galois) of finite degree. Let α be a primitive element, i.e., $K = k(\alpha)$, and let $f \in k[x]$ be the minimal polynomial of α over K. Let $\hat{K}$ be the splitting field of f and observe that $\hat{K}/K$ is a separable extension. By Theorem 2.2, there exists a bijection between subfields of $\hat{K}$ and subgroups of $\mathrm{Aut}(\hat{K}/k)$. For simplicity, let us denote the group $\mathrm{Aut}(\hat{K}/k)$ by $\mathrm{Gal}(f)$. Conversely, whenever we write $\mathrm{Gal}(f)$, we mean the group $\mathrm{Aut}(\hat{K}/k)$, where $\hat{K}$ is the splitting field of f.

Since $k \subset K \subset \hat{K}$, there exists a subgroup $G_\alpha \leq \mathrm{Gal}(f)$ such that $\hat{K}^{G_\alpha} = K$. Since $K = k(\alpha)$, it is easy to determine G_α. Indeed,

$$G_\alpha = \mathrm{Aut}(\hat{K}/K) = \{\sigma \in \mathrm{Gal}(f) \ : \ \sigma(\alpha) = \alpha\}.$$

Thus, if L is a subfield of K, then the corresponding subgroup via the Galois Correspondence is a group containing G_α. On the other hand, if $G_\alpha \leq H \leq \mathrm{Gal}(f)$, then the field $\hat{K}^H$ is a subfield of K. That is, there is a bijection between subfields of K and subgroups of $\mathrm{Gal}(f)$ containing G_α.

Example 2.3. *Let α be a root of $f = x^4 - 2 \in \mathbb{Q}[x]$ (say $\alpha = \sqrt[4]{2}$). The 4 roots of f are $\alpha_1 = \alpha, \alpha_2 = i\alpha, \alpha_3 = -\alpha$ and $\alpha_4 = -i\alpha$ and hence, the splitting field of f is $\mathbb{Q}(\alpha, i)$. Let us analyze the bijection between subfields of $\mathbb{Q}(\alpha, i)$ and subgroups of $Gal(f) = Aut(\mathbb{Q}(\alpha, i)/\mathbb{Q})$. Consider the automorphisms $r, s \in Gal(f)$, such that $r(\alpha) = i\alpha$, $r(i) = i$ and $s(\alpha) = \alpha$, $s(i) = -i$, and the subgroup $H = \{id, r, r^2, r^3, s, rs, r^2s, r^3s\}$ of $Gal(f)$. Since $[\mathbb{Q}(\sqrt[4]{2}, i) : \mathbb{Q}] = |H| = 8$, it follows from item 1. of Theorem 2.2 that*

$$Gal(f) = H = \{id, r, r^2, r^3, s, rs, r^2s, r^3s\}.$$

This group is isomorphic to the group D_4 of the eight symmetries of the square whose vertices are the 4 roots of f. The isomorphism r can be seen as a rotation counterclockwise of 90 degrees and s as a reflexion across one diagonal. The next figure shows the subgroup lattice of $Gal(f) = D_4$.

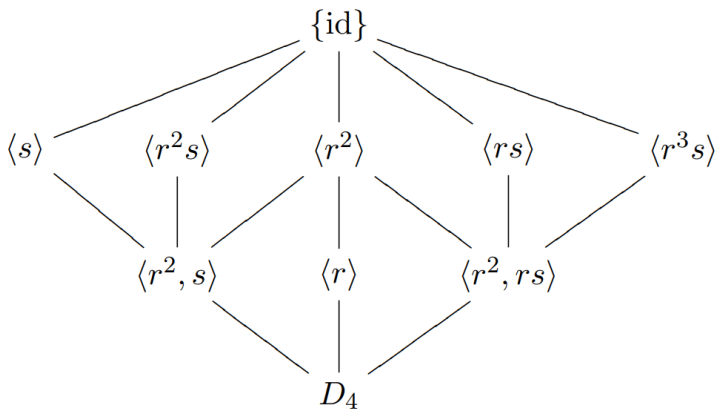

Figura 2.1: Subgroups Lattice

Since $\mathbb{Q}(\alpha, i)/\mathbb{Q}$ is a Galois extension, we already know that the subfield lattice has the same structure. Moreover, the field corresponding to $\{id\}$ is $\mathbb{Q}(\alpha, i)$ and the field corresponding to D_4 is $\mathbb{Q}$. The group $\langle r\rangle = \{id, r, r^2, r^3\}$, for instance, is a subgroup of $Gal(f)$ of index 2. Hence, it corresponds to a subfield of $\mathbb{Q}(\alpha, i)$ of degree 2 over $\mathbb{Q}$. Since $r(i) = i$ and $[\mathbb{Q}(i) : \mathbb{Q}] = 2$, this subfield is $\mathbb{Q}(i)$. Now suppose that we want to find all subfields of $\mathbb{Q}(\alpha)$. The group G_α that fixes $\mathbb{Q}(\alpha)$ is $\langle s\rangle$ and the only (proper) subgroup of D_4 that contains G_α is $\langle r^2, s\rangle$, which corresponds to the subfield $\mathbb{Q}(\sqrt{2})$. That is, the only (nontrivial) subfield of $\mathbb{Q}(\sqrt[4]{2})$ is $\mathbb{Q}(\sqrt{2})$. Below we show the subfield lattice.

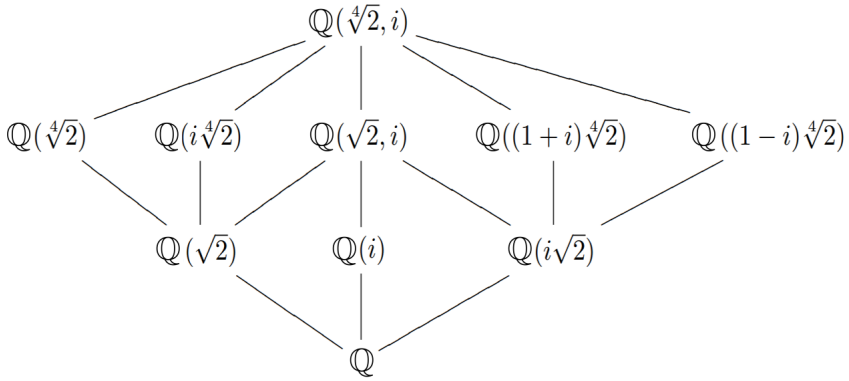

Figura 2.2: Subfields Lattice

2.2.2 Blocks of Imprimitivity

Let K/k be a separable field extension of degree n with primitive element $\alpha \in K$ and let $f \in k[x]$ be its minimal polynomial. Let $\hat{K}$ be the splitting field of f. By the relation given in Theorem 2.2, finding the subfield lattice of K/k is equivalent to finding the subgroups of $\mathrm{Gal}(f)$ that fix α, that is, subgroups containing G_α. In this subsection we study the action of $\mathrm{Gal}(f)$ on the roots of f.

Let $\Omega := \{\alpha_1 = \alpha, \alpha_2, \ldots, \alpha_n\}$ be the set of roots of f over $\hat{K}$ and let $\sigma \in \mathrm{Gal}(f)$. Then $f(\sigma(\alpha_i)) = \sigma(f(\alpha_i)) = 0$, for $i = 1, \ldots, r$, that is, σ maps roots of f into roots of f. Hence, $\mathrm{Gal}(f)$ acts permutationally on the set of roots Ω. Henceforth, we might call an element $\sigma \in \mathrm{Gal}(f)$ of a *permutation of* $\mathrm{Gal}(f)$. This "permutation" should be understood as a permutation of Ω.

Definition 2.4. *Let Δ be a subset of Ω. Then Δ is a block of imprimitivity if*

$$\sigma(\Delta) \cap \Delta \in \{\emptyset, \Delta\}, \forall\, \sigma \in \mathit{Gal}(f).$$

Let Δ be a block of imprimitivity. The set of all block conjugates, that is, $\{\sigma(\Delta) : \sigma \in \mathit{Gal}(f)\}$, is called a system of imprimitivity.

Notice that a system of imprimitivity yields a partition of the set Ω. Moreover, if $|\Delta| = d$, then $|\sigma(\Delta)| = d$, for any $\sigma \in \mathrm{Gal}(f)$. For simplicity, we might omit the word imprimitivity, so we have *blocks* and *system of blocks*.

Let L be a subfield of K of degree n/d and let $G_\alpha \leq H$ be the corresponding subgroup of $\mathrm{Gal}(f)$. Let $\Delta \subseteq \Omega$ be the orbit of α under H, that is, $\Delta = \{\sigma(\alpha) : \forall \sigma \in H\}$. Then Δ is a block and $H = \mathrm{Stab}_\Delta = \{\sigma \in \mathrm{Gal}(f) : \sigma(c) = c, \forall c \in \Delta\}$. On the other hand, if $\Delta \subseteq \Omega$ is a block of size d with $\alpha \in \Delta$, then $L := \hat{K}^{\mathrm{Stab}_\Delta}$ is a field such that $k \subseteq L \subseteq K$. That is, there is a bijection between the subfields L of K/k of degree n/d and the blocks Δ of Ω of size d which contain

α. In Section 2.2.3 we will show how one can determine the corresponding subfield from a given block of imprimitivity.

Example 2.5. *Consider* $f = x^4 - 2 \in \mathbb{Q}[x]$ *and* $\alpha = \sqrt[4]{2}$. *The set of roots of* f *is* $\Omega = \{\alpha, i\alpha, -\alpha, -i\alpha\}$. *Consider the subfield* $\mathbb{Q}(\sqrt{2})$ *of* $\mathbb{Q}(\alpha)$. *We wish to find the block of imprimitivity* Δ *corresponding to this subfield. As we have seen in Example 2.3, the only subgroup that contains* G_α *is* $\langle r^2, s\rangle = \{id, r^2, s, r^2s\}$. *The block of imprimitivity* Δ *is then given by the orbit of* α *under* $\langle r^2, s\rangle$, *that is,* $\Delta = \{id(\alpha), r^2(\alpha), s(\alpha), r^2s(\alpha)\} = \{\alpha, -\alpha, \alpha, -\alpha\} = \{\alpha, -\alpha\}$. *In this case, it is obvious that the block system is* $\Delta_1 = \Delta = \{\alpha, -\alpha\}$ *and* $\Delta_2 = \{i\alpha, -i\alpha\}$ *(the remaining roots). However, one can also compute* Δ_2 *as the image of* Δ_1 *over some automorphism of Gal*(f), *for instance,* $\Delta_2 = rs(\Delta_1) = \{rs(\alpha), rs(-\alpha)\} = \{i\alpha, -i\alpha\}$. *One can also check that this is the only non trivial block of imprimitivity, which confirms that* $\mathbb{Q}(\sqrt{2})$ *is the only proper subfield of* $\mathbb{Q}(\alpha)$.

However, we do not always have the set Ω of roots of f. If this was the case, the Galois group (and hence, the subfield lattice of our extension) could be easily computed (this is the case in Examples 2.3 and 2.5). Generally, directly computing the Galois group (and hence the blocks of imprimitivity) is a hard task, which we shall avoid.

A naïve approach to compute all subfields of degree n/d would be to consider all subsets of size d which contain α and verify which of these subsets corresponds to a subfield, that is, check which of these subsets is a block of imprimitivity. However, even for small values of n, this approach is unfeasible. In what follows, we see that not all subsets of size d have to be tested. This is achieved by defining *potential blocks of imprimitivity* and *potential systems of imprimitivity.*

Definition 2.6. *Let* π *be a permutation of Gal*(f). *A subset* $A \subseteq \Omega$ *is a potential block (for* π*) of size* d *if* $|A| = d$ *and* $\pi^j(A) \cap A \in \{\emptyset, A\}$, *for* $1 \leq j \leq |\langle \pi \rangle|$. *A system* $A_1, \ldots, A_m$ *of potential blocks of size* d *is a potential system of size* d *if*

$\Omega = \cup A_j$, $A_i \cap A_j = \emptyset$, *for* $i \neq j$ *and* $\pi^j(A_i) \in \{A_1, \ldots, A_m\}$, *for every* $1 \leq i \leq m$ *and* $1 \leq j \leq |\langle \pi \rangle|$.

The important property of this set of potential blocks is that a block of imprimitivity is always a potential block of imprimitivity and a block system is always a potential block system. Our objective now is to compute all potential block systems (for a permutation π).

Example 2.7. *Consider the automorphism* $r^2s \in Gal(f)$ *from Example 2.3, and let us enumerate the roots of* $f = x^4 - 2 \in \mathbb{Q}[x]$ *as* $\alpha_1 = \alpha, \alpha_2 = i\alpha, \alpha_3 = -\alpha$ *and* $\alpha_4 = -i\alpha$, *where* $\alpha = \sqrt[4]{2}$. *The automorphism* r^2s *can be seen as a permutation* π *on* $\{\alpha_1, \alpha_2, \alpha_3, \alpha_4\}$. *Indeed,* $r^2s(\alpha_1) = \alpha_3$, $r^2s(\alpha_2) = \alpha_4$, $r^2s(\alpha_3) = \alpha_1$ *and* $r^2s(\alpha_4) = \alpha_2$. *That is,* r^2s *acts as the permutation* $\pi = (\alpha_1\alpha_3)(\alpha_2\alpha_4)$ *on* $\Omega = \{\alpha_1, \alpha_2, \alpha_3, \alpha_4\}$. *For this permutation, we can compute all potential block systems of size* 2*: choosing* $A = \{\alpha_1, \alpha_2\}$ *we have* $\pi(\{\alpha_1, \alpha_2\}) = \{\alpha_3, \alpha_4\}$ *(notice that* $\pi^2 = id$*). So* A *is a potential block and the corresponding potential block system is* $A_1 = A = \{\alpha_1, \alpha_2\}, A_2 = \{\alpha_3, \alpha_4\}$. *Choosing different* A *gives us two more potential block systems, namely,* $A_1 = \{\alpha_1, \alpha_3\}, A_2 = \{\alpha_2, \alpha_4\}$ *and* $A_1 = \{\alpha_1, \alpha_4\}, A_2 = \{\alpha_2, \alpha_3\}$. *Notice that the second (of the three) potential block system is the block system we computed in Example 2.5.*

The potential block systems computed in the previous example depend on the choice of π. If we had chosen the automorphism rs, then the corresponding permutation would have been $\pi = (\alpha_1\alpha_4\alpha_3\alpha_2)$ and the only potential block system would have been $A_1 = \{\alpha_1, \alpha_3\}, A_2 = \{\alpha_2, \alpha_4\}$. However, in general, we do not always have an automorphism $\sigma \in \mathrm{Gal}(f)$. The following criterion allows us to compute a set of *potential blocks*, knowing only the cycle type information of a permutation π of $\mathrm{Gal}(f)$.

Theorem 2.8 (Dedekind Criterion)**.** *Let R be a unique factorization domain*[3]*, $\mathfrak{p}$ be a prime ideal in R, $\bar{R} := R/\mathfrak{p}$ its residue class ring, $f \in R[x]$ and $\bar{f} \in \bar{R}[x]$ with $f \equiv \bar{f} \bmod \mathfrak{p}$. If $\bar{f}$ is square-free, then Gal($\bar{f}$) is isomorphic to a subgroup of Gal(f).*

This criterion allows us to determine the cycle type of cyclic subgroups of $\mathrm{Gal}(f)$ generated by a permutation $\pi \in \mathrm{Gal}(f)$. That is, if $\bar{f}$ has a factorization $\bar{f}_1 \cdots \bar{f}_r$, with $n_i = \deg(\bar{f}_i)$, then $\mathrm{Gal}(f)$ has a permutation of cycle type $[n_1, \ldots, n_r]$.

Suppose we know a permutation π to be an element of $\mathrm{Gal}(f)$. Furthermore, suppose that $\pi = \pi_1 \cdots \pi_r$ is the product of π into disjoint cycles and $|\pi_i| = n_i$, for $1 \leq i \leq r$, i.e., the cycle π_i has size n_i (this information will be obtained using Theorem 2.8). Let Δ be a block of size d. Recall that $\sigma(\Delta) \cap \Delta = \Delta$ or $\sigma(\Delta) \cap \Delta = \emptyset$, for any $\sigma \in \mathrm{Gal}(f)$. Hence, there exists some $m \geq 1$ such that

$$\pi^j(\Delta) \cap \Delta = \emptyset, \text{ for } 1 \leq j < m, \text{ and } \pi^m(\Delta) \cap \Delta = \Delta. \tag{2.2}$$

Let α be an element of Δ. Let π_l, $1 \leq l \leq r$, be a cycle such that either $\pi_l = (\alpha)$ or that $\pi_l(\alpha) \neq \alpha$. In this case we say that $\alpha \in \pi_l$. Since $\pi^m(\Delta) = \Delta$, it follows that $\pi^{cm}(\alpha) \in \Delta$, for all $c \in \mathbb{N}$. Moreover, for any $\tilde{m}$ not divisible by m, we have $\pi^{\tilde{m}}(\Delta) \cap \Delta = \emptyset$. On the other hand, $\pi^{n_l}(\alpha) = \alpha$ (because $\alpha \in \pi_l$ and π_l is a cycle of length n_l). Since π^{n_l} fixes α and Δ satisfies Equation (2.2), it follows that π^{n_l} must fix Δ, that is, $\pi^{n_l}(\Delta) = \Delta$. Thus, $n_l = cm$, for some $c \in \mathbb{N}$ and hence, $m \mid n_l$. Furthermore, the cycle π_l contains n_l/m elements of Δ.

Since this holds for any block in a block system, it follows that there must be a partition $\{I_1, \ldots, I_s\}$ of $\{1, \ldots, r\}$ and integers $m_1, \ldots, m_s$, with $m_i \geq 1$ for $1 \leq i \leq s$, such that, for each $t = 1, \ldots, s$, we have

$$d = \sum_{i \in I_t} \frac{n_i}{m_t}, \text{ with } m_t \mid n_i, \text{ for all } i \in I_t. \tag{2.3}$$

[3] *A Unique Factorization Domain is an integral domain in which every non-zero non-unit element can be written as a product of irreducible elements,uniquely up to order and units.*

Sometimes, depending on d and the cycle structure (i.e., the values $n_i, 1 \leq i \leq r$), it is possible to affirm that there are no subfields of degree n/d because no partition of $\{1, \ldots, r\}$ exists that satisfy Equation (2.3). This equation might also severely reduce the number of ways in which the roots in Ω can be grouped to form blocks of imprimitivity of size d.

Example 2.9. *Let $\mathbb{Q}(\alpha)/\mathbb{Q}$ be a field extension of degree 15 and let $f \in \mathbb{Z}[x]$ be the minimal polynomial of α. We want some information about the existence of subfields of degree 3 (so the size of any potential block is $d = 5$). Suppose that for some prime p, the factorization of $f \bmod p$ is composed of 1 linear factor, 1 degree-2 factor and 3 degree-4 factors (let us call these factors $\bar{f}_1, \bar{f}_2, \bar{f}_3, \bar{f}_4$ and $\bar{f}_5$, respectively). By the Dedekind Criterion, this means that Gal(f) has a permutation of cycle type $[1, 2, 4, 4, 4]$. By looking at Equation (2.3), we see that there are only 3 ways in which the roots of f can fall into a potential block of size 5, namely: the root of the linear factor and the four roots of one of the degree-4 factors. Choosing the first degree-4 factor, i.e., $\bar{f}_3$, we have the partition $I_1 = \{1, 3\}$, $m_1 = 1$ and $I_2 = \{2, 4, 5\}$, $m_2 = 2$. The number of potential blocks is intimately connected to the cycle type. Suppose that p' is another prime number which gives a permutation of Gal(f) of cycle type $[1, 1, 1, 2, 2, 2, 2, 2, 2]$. In this case, there are 21 possible ways[4] into which the roots of f can be grouped to form potential blocks of size 5.*

Remark 2.10. *In Example 2.9, we mentioned that any block Δ of size $d = 5$ must contain roots of some factor $\bar{f}_i$ of $f \bmod p$. That is, the potential block Δ is given in terms of the roots of $f \bmod p$ and not on the roots of f. In the next section we show that this information is enough to compute the corresponding subfield.*

Remark 2.11. *One can always choose a prime p such that $p \nmid \text{disc}(f)$ (and hence, the factorization of $f \bmod p$ is square-free) and that if a potential block Δ contains a root of some factor $\bar{f}_i$ of $f \bmod p$, then Δ contains all roots of $\bar{f}_i$. This is achieved*

[4] *We are only counting blocks which contain the root of a fixed linear factor of $f \bmod p'$, say $\bar{f}_1$.*

by choosing a prime p such that $f \bmod p$ has a linear factor $x - \alpha$ (and this factor is fixed, i.e., we are only looking for blocks which contain α). This is the case in Example 2.9. If, for instance, f is some polynomial for which Gal(f) has a permutation of cycle type $[2, 2, 2, 2, 2]$, then any block Δ of size 5 would be composed of a single root of each of the degree-2 factors.

2.2.3 From Blocks to Subfields

In the previous section we showed how one could compute a set of *potential* blocks of imprimitivity. Recall that a set of potential blocks contains all blocks of imprimitivity. Therefore, we need a method to discard potential blocks which do not correspond to a subfield. One way to verify if a potential block corresponds to a subfield is attempting to compute this subfield. If the computation fails, then the potential block was not an actual block of imprimitivity.

For simplicity, let us consider the finite extension $\mathbb{Q}(\alpha)/\mathbb{Q}$ and let $f \in \mathbb{Q}[x]$ be the minimal polynomial of α over $\mathbb{Q}$. Furthermore, let $\Omega = \{\alpha_1 = \alpha, \alpha_2, \ldots, \alpha_n\}$ be the set of roots of f in its splitting field $\tilde{\mathbb{Q}}$. Given a block Δ corresponding to a subfield L of $\mathbb{Q}(\alpha)/\mathbb{Q}$, we want to find δ such that $L = \mathbb{Q}(\delta)$, given by $\delta = h(\alpha)$, for $h \in \mathbb{Q}[x]_{<n}$, and the minimal polynomial $g \in \mathbb{Q}[x]$ of δ over $\mathbb{Q}$. Let $\Delta_1, \Delta_2, \ldots, \Delta_m$ be the block system and define

$$\delta_i = \prod_{\gamma_j \in \Delta_i} \gamma_j, \text{ for } 1 \leq i \leq m, \text{ and let } g = \prod_{i=1}^{m} (x - \delta_i) \in \tilde{\mathbb{Q}}[x]. \tag{2.4}$$

Since Gal(f) acts permutationally on the roots of f and $\Delta_1, \ldots, \Delta_m$ is a block system, it follows that $\delta_i, 1 \leq i \leq m$, are conjugates over L and hence, $g \in \mathbb{Q}[x]$. Furthermore, if $\alpha \in \Delta_1$, then $L = \mathbb{Q}(\delta_1)$ and g is the minimal polynomial of δ_1 over $\mathbb{Q}$. We may suppose that all δ_i are distinct (see [17, Appendix A]). Furthermore, we may also suppose that the δ_i are algebraic integers (that is, the minimal polynomial of δ_i is monic and has integer coefficients, see [17]). Thus, we may assume $g \in \mathbb{Z}[x]$.

However, as we have mentioned before, we do not have the set Ω. Let us now show that it is enough to have the potential blocks in terms of the roots of $f \bmod p$. Let p be a prime number such that $p \nmid \operatorname{disc}(f)$. Let $\mathcal{O}_N$ be the ring of integers of $N = \tilde{\mathbb{Q}}$ and let $\mathfrak{p}$ be a prime ideal of $\mathcal{O}_N$ lying over p (that is, $\mathfrak{p} \cap \mathbb{Z} = p\mathbb{Z}$). Furthermore, let $E = \tilde{\mathbb{Q}}_{\mathfrak{p}}$ be the p-adic completion of $\tilde{\mathbb{Q}}$ and let Φ be the canonical embedding from $\tilde{\mathbb{Q}}$ to $\tilde{\mathbb{Q}}_{\mathfrak{p}}$. If $\tilde{f} = \Phi(f)$, let $\{\tilde{\alpha}_1, \ldots, \tilde{\alpha}_n\}$ be the roots of $\tilde{f}$ over $\tilde{\mathbb{Q}}_{\mathfrak{p}}$, with $\tilde{\alpha}_i = \Phi(\alpha_i)$. It can be shown that, if

$$\tilde{\delta}_i = \prod_{\gamma_j \in \Delta_i} \Phi(\gamma_j), \text{ for } 1 \leq i \leq m, \text{ and } \tilde{g} = \prod_{i=1}^{m} (x - \tilde{\delta}_i) \in \tilde{\mathbb{Q}}_{\mathfrak{p}}[x], \tag{2.5}$$

then $\tilde{g} = \Phi(g)$. Furthermore, let $a \in \mathbb{N}$ and let $\mathcal{O}_E$ be the ring of integers of $E = \tilde{\mathbb{Q}}_{\mathfrak{p}}$ with maximal ideal $\mathfrak{P}$. If $\hat{\delta}_i$ is an approximation of $\tilde{\delta}_i \in \tilde{\mathbb{Q}}_{\mathfrak{p}}$ with accuracy a, that is, $\hat{\delta}_i \equiv \tilde{\delta}_i \bmod \mathfrak{P}^a$, and if $\hat{g} = \prod_{i=1}^{m} (x - \hat{\delta}_i)$, then we have $\hat{g} \equiv \tilde{g} \bmod \mathfrak{P}^a$. Since we are assuming $g \in \mathbb{Z}[x]$, it follows that $\tilde{g} \in \mathbb{Z}_p[x]$ and hence, $\hat{g} \equiv \tilde{g} \bmod p^a$. Moreover, $\tilde{g} = g$ over $\mathbb{Z}_p$ (provided g is embedded into $\mathbb{Z}_p[x]$ canonically), and hence $\hat{g} \equiv g \bmod p^a$, provided a is high enough (see [26, Lemma 39]). The following picture might help elucidate the situation.

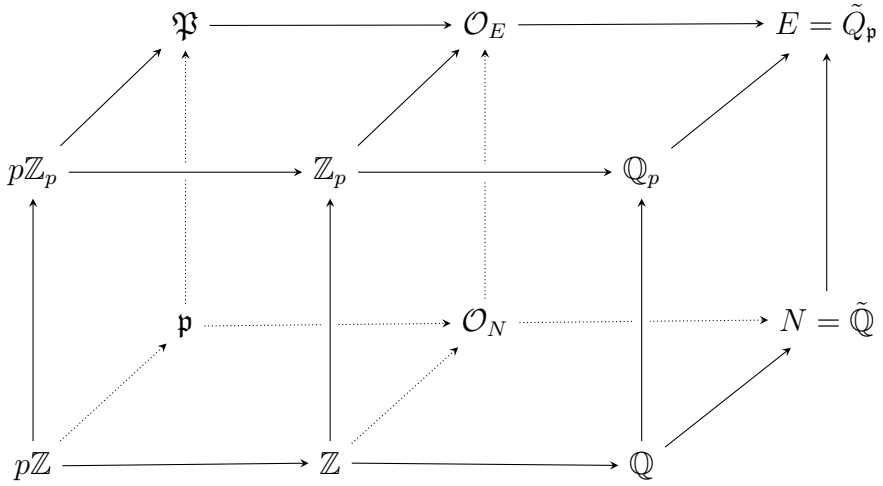

Figura 2.3: Inclusion Diagram

If we have a bound for the coefficients of $g \in \mathbb{Z}[x]$, then choosing a high enough allows us to compute g from $\hat{g}$. That is, we want to find $g \in \mathbb{Z}[x]$ given by $\prod_{i=1}^{m}(x-\delta_i)$, where $\delta_i = \prod_{\gamma\in\Delta_i}\gamma \in \tilde{\mathbb{Q}}$, $1 \le i \le m$. However, we do not have the elements $\delta_i \in \tilde{\mathbb{Q}}$. Hence, we look at the p-adic completion of $\tilde{\mathbb{Q}}$ and define $\tilde{g} \in \tilde{\mathbb{Q}}_{\mathfrak{p}}[x]$. Again, we cannot compute $\tilde{g}$ with infinite accuracy. However, we can compute approximations $\hat{\delta}_i$ of the $\tilde{\delta}_i$ (by Hensel Lifting the factorization of $f \bmod p$) and hence, approximations $\hat{g}$ of $\tilde{g}$. Since $g \in \mathbb{Z}[x]$ and $\tilde{g} \in \mathbb{Z}_p[x]$, and if the accuracy a is high enough, we can compute $g \in \mathbb{Z}[x]$ by the approximations $\hat{g}$ of $\tilde{g} \in \mathbb{Z}_p[x]$. Obviously, we need an *a priori* bound on the coefficients of $g \in Z[x]$ to be able to compute g. For this, we use the Mignotte's bound [37] for factors of f.

Example 2.12. *Again, consider $f = x^4 - 2 \in \mathbb{Q}[x]$. Modulo $p = 7$, f factors as*

$$f \bmod 7 = (x+2)(x+5)(x^2+4).$$

Let $\alpha_1 = 5$, $\alpha_3 = 2$ and α_2 and α_4 be the two roots of x^2+4 (this choice of order does not influence on the computations). Hence, by Dedekind's Criterion, Gal(f) has a permutation of cycle type $[1,1,2]$ (indeed, the permutation s in Example 2.3 has this cycle type). Let us find all potential blocks of size 2 which contain α_1 and satisfy Equation (2.3). For this choice of prime, the only potential block is $\{\alpha_1,\alpha_3\}$ and the corresponding potential block system is $A_1 = \{\alpha_1,\alpha_3\}$, $A_2 = \{\alpha_2,\alpha_4\}$. We will now try to compute the subfield L corresponding to this potential block system. First, let us compute $g \in \mathbb{Z}[x]$, the minimal polynomial of a generator of L. Since $g \mid f$, the Mignotte bound tells us that $\|g\|_\infty \le 2^{n-1}\|f\|_2 = 8\sqrt{5} < 18$. Hence, we should Hensel Lift the factorization of $f \bmod 7$ to a factorization $\bmod 7^2$ (since $7^2 > 2\cdot 18$). By doing this, we get the factorization

$$f \bmod 7^2 = (x+23)(x+26)(x^2+39).$$

Now, let us compute an approximation $\hat{g}$ of g modulo 7^2. Let $\hat{\delta}_1 = (-23)(-26) = 10 \bmod 7^2$ and $\hat{\delta}_2 = 39 \bmod 7^2$. Finally, define $\hat{g} = (x-\hat{\delta}_1)(x-\hat{\delta}_2) = x^2+47 \bmod 7^2$.

Rewriting $\hat{g}$ in the symmetric representation, that is, with coefficients in $\left(-\frac{7^2}{2}, \frac{7^2}{2}\right)$, we get $\hat{g} = x^2 - 2$. Notice that the coefficients of $\hat{g}$ satisfy the bound found above. If this was not the case, then we would have a proof that this potential block was not an actual block of imprimitivity. It remains to compute the embedding of L in $\mathbb{Q}(\alpha)$, which we explain below.

While the previous method requires that we have a block system, it is enough to only have the block $\Delta = \Delta_1$ such that $\alpha_1 \in \Delta$. For simplicity, let us consider $f \in \mathbb{Z}[x]$. Let p be a prime such that $p \nmid \operatorname{disc}(f)$ and such that $f \bmod p$ has a linear factor. Furthermore, suppose that $f \bmod p$ has the following factorization

$$f \equiv \bar{f}_1 \bar{f}_2 \cdots \bar{f}_r \bmod p, \tag{2.6}$$

where $\bar{f}_1$ is linear. As mentioned in the last subsection, if $n_i = \deg(\bar{f}_i)$, then $\mathrm{Gal}(f)$ has a permutation of cycle type $[n_1, \ldots, n_r]$. Let Δ be a potential block of imprimitivity and suppose that Δ contains the root of $\bar{f}_1$ (here Δ is a subset of the roots of $f \bmod p$). Then Δ must contain all the roots of some factors $\bar{f}_i$ of $f \bmod p$ (see Remark 2.11). This allows us to compute the product $\delta \in p\mathbb{Z}$ of the roots in Δ. Furthermore, we can compute δ with *accuracy* a, that is, we can compute $\tilde{\delta} \in p^a\mathbb{Z}$ such that $\tilde{\delta} \equiv \delta \bmod p^a$, for some integer a, by Hensel Lifting the factorization in (2.6) to a factorization

$$f \equiv f_1^{(a)} f_2^{(a)} \cdots f_r^{(a)} \bmod p^a, \tag{2.7}$$

where $f_i^{(a)} \in p^a\mathbb{Z}[x]$ and $f_i^{(a)} \equiv \bar{f}_i \bmod p$. If the accuracy a is high enough, we can use LLL to compute the minimal polynomial $g \in \mathbb{Z}[x]$ of δ (recall Section 1.3.1).

At this point we have the minimal polynomial $g \in \mathbb{Q}[x]$ of δ over $\mathbb{Q}$ (computed from the image $\tilde{\delta}$ of δ over $p^a\mathbb{Z}$). The next step is to express δ as an element of $\mathbb{Q}(\alpha)$, where α is a root of f (notice that so far we only have an approximation of δ). That is, we want to find $h \in \mathbb{Q}[x]$ such that $\delta = h(\alpha)$. We may suppose that $\deg(h) < n$, where $n = \deg(f)$. The polynomial h must satisfy

$h(\alpha_i) = \delta_j$, for all $\alpha_i \in \Delta_j$, and for $1 \leq j \leq m$, where δ_j is as in (2.4). At this step, we can change the prime p (or we can extend the field $\mathbb{Z}/\langle p \rangle$) such that f splits into linear factors over this new field. Hence, we know the value of h at n distinct points. Since $\deg(h) \leq n-1$, h is uniquely determined. Again, we can use p-adic techniques to compute an approximation modulo p of h from the approximations of α_i and δ_j in a p-adic extension.

If we start with a potential block, then the construction of g or h might fail, for instance, the coefficients of g (or h) might not satisfy the (a priori) bound for its coefficients. If this happens, we have proof that the potential block does not correspond to a subfield of $\mathbb{Q}(\alpha)/\mathbb{Q}$.

2.3 Other Approaches

There are several other algorithms to compute subfields of a field extension K/k. In this section we briefly mention some of them. The method presented in the previous section is mostly based on Dixon's work [17]. Klüners [26] presents some improvements on Dixon's algorithm, including the intersection of potential blocks, which helps to reduce the number of potential blocks we have to test.

In 1983, Landau and Miller [30] gave a different method to compute the blocks of imprimitivity. However, this method requires the computation of factorization over a number field, and computation of gcd's over $\mathbb{Q}(\alpha, \beta)$, where α, β are roots of f, which tends to be computationally expensive.

If L is a subfield of K/k and if α is a primitive element with minimal polynomial f, then $g \mid f$, where g is the minimal polynomial of α over L. Moreover, the coefficients of g generate L as a k-algebra (see Lemma 3.1). Hulpke [25] proposes to factor f over K and test which combination of these factors yields a subfield. Although these tests can be done with p-adic approximations, the method involves

the factorization of f over K. Moreover, if the number of factors is high, the algorithm might require a large amount of time due to its combinatorial nature.

Cohen and Diaz y Diaz [14] present an algorithm, known as `POLRED`, whose objective is, given an extension $\mathbb{Q}(\alpha)$ with minimal polynomial f, return a polynomial g *as simple as possible*[5] such that, if β is a root of g, then $\mathbb{Q}(\alpha) \cong \mathbb{Q}(\beta)$. This algorithm might return several candidates g. However, if $\deg(g) < \deg(f)$, then g generates a subfield of $\mathbb{Q}(\alpha)$. The algorithm `POLRED` needs to compute an integral basis of the ring of integers of $\mathbb{Q}(\alpha)$. This operation involves the factorization of the discriminant of f, which might be computationally hard. Moreover, this algorithm is not guaranteed to give a proper subfield, and even if it does, we are not guaranteed to obtain all subfields of the extension $\mathbb{Q}(\alpha)/\mathbb{Q}$.

Let f be the minimal polynomial of α over $\mathbb{Q}$ with roots $\alpha_1 = \alpha, \alpha_2, \ldots,$ α_n, and let L be a subfield of $\mathbb{Q}(\alpha)$ of index d, that is, $[\mathbb{Q}(\alpha) : L] = d$. Let $s \in \mathbb{Q}(x_1, \ldots, x_d)$ and let H be a subgroup of the Symmetric group S_n. Denote by $H(s)$ the set of all functions $s(x_{\sigma(1)}, \ldots, x_{\sigma(d)})$, for any $\sigma \in H$. If $s \in \mathbb{Q}(x_1, \ldots, x_d)$ is a symmetric function, we define the d-symmetric resolvent $s_*(f)$ as

$$s_*(f) = \prod_{g \in S_n(s)} (x - g(\alpha_1, \ldots, \alpha_d)) .$$

Lazard and Valibouze [32] noticed that any d-symmetric resolvent $s_*(f) \in \mathbb{Q}[x]$ has a factor of degree n/d which is a power of an irreducible polynomial h and the roots of h are in L. However, the degree of $s_*(f)$ is $\binom{n}{d}$, which, depending on d, is much bigger then the degree of the extension.

Casperson and Mckay [13] recall that if $\mathbb{Q}(\alpha)$ has a non-trivial subfield, then there exists a polynomial $h \in \mathbb{Q}[x]$ such that $h(\alpha_i) = h(\alpha_j)$, for some α_i, α_j distinct roots of f (the minimal polynomial of α). By using approximations of the

[5] As defined in their paper.

roots of f and LLL, the authors are able to determine the polynomial h and hence, $L = \mathbb{Q}(h(\alpha))$ is a subfield of $\mathbb{Q}(\alpha)$.

2.4 Computing Subfields using Principal Subfields

Let K/k be a field extension of degree n. In this section we present a method to compute the subfield lattice K/k by searching for a special subset of (at most) $n-1$ subfields of K/k. This special subfields are called *intersection-generating subfields* and any subfield of the extension K/k is the intersection of some of these principal subfields. This method was presented by van Hoeij *et al.* [51].

The number of subfields of a finite extension field is not polynomially bounded in general. The importance of this method is that we essentially need to find at most $n-1$ subfields, that is, a polynomial time task. Furthermore, all subfields can be computed by intersecting subsets of these subfields. While the latter step is still not polynomially bounded, our main contribution (which we present in the next chapter) resides on how these intersections are computed. The remaining of this chapter is a brief description of the algorithm from [51].

2.4.1 Principal Subfields

Let K/k be a field extension of degree n. Throughout this work, K/k will be a separable extension. Let α be a primitive element with minimal polynomial $f \in k[x]$. Furthermore, let $\tilde{K}$ be a finite extension of K, so that we have $k \subseteq K \subseteq \tilde{K}$. Consider the factorization $f = f_1 \cdots f_r$ of f over $\tilde{K}$, where $f_i \in \tilde{K}[x]$ is an irreducible polynomial. We may suppose that $f_1 = x - \alpha$. Define the fields $\tilde{K}_i := \tilde{K}[x]/\langle f_i(x) \rangle$, $1 \leq i \leq r$. If the elements of K are denoted by $g(\alpha)$, for $g \in k[x]_{<n}$, define

$$\Phi_i := K \to \tilde{K}_i, \quad \Phi(g(\alpha)) := g(x) \bmod f_i, \tag{2.8}$$

for $1 \leq i \leq r$. Furthermore, let

$$L_i := \text{Kernel}(\Phi_i - id) = \{g(\alpha) \in K : g(x) \equiv g(\alpha) \bmod f_i\}. \tag{2.9}$$

The set L_i is a subfield of K/k and the subfields L_i, $1 \leq i \leq r$, are called *principal subfields* of the extension K/k. Moreover, the following is true.

Theorem 2.13 ([51], Theorem 1)**.** *Let L_i, $1 \leq i \leq r$, be as in* (2.9)*. If L is a subfield of K/k, then there exists a subset $I \subseteq \{1, \ldots, r\}$ such that*

$$L = \{g(\alpha) \in K : g(x) \equiv g(\alpha) \bmod g_L\} = \bigcap_{i \in I} L_i, \tag{2.10}$$

where $g_L \in L[x]$ is the minimal polynomial of α over L.

Proof. Let $g_L \in L[x]$ be the minimal polynomial of α over L. It follows that $g_L \mid f$, and hence, there exists a set $I \subseteq \{1, \ldots, r\}$ such that $g_L = \prod_{i \in I} f_i$. We shall now prove the first equality of (2.10). Let $g(\alpha) \in L$ and consider $h(x) := g(x) - g(\alpha) \in L[x]$. Then $h(x)$ is divisible by $x - \alpha$. However, the set of polynomials over L with α as a root is an ideal given by $\langle g_L \rangle$, and hence, $h(x) \in \langle g_L \rangle$. That is, $g_L \mid h$, or equivalently, $g(x) \equiv g(\alpha) \bmod g_L$. Conversely, suppose that $g(\alpha) \in K$ satisfies $g(\alpha) = g(x) \bmod g_L$. Since $g \in k[x]$ and $g_L \in L[x]$, it follows that $g(x) \bmod g_L \in L[x]$ and hence, $g(\alpha) \in L[x] \cap K = L$.

To show the second equality, let $g(\alpha) \in L$, that is, $g(x) \equiv g(\alpha) \bmod g_L$ or, equivalently, $g_L \mid g(x) - g(\alpha)$. Since we are assuming K/k separable (and hence, $f_i \neq f_j$, for $i \neq j$), it follows that $g_L \mid g(x) - g(\alpha)$ if and only if, $f_i \mid g(x) - g(\alpha)$, for every $i \in I$. That is, $g(\alpha) \in L$ if and only if, $g(\alpha) \in L_i$, for every $i \in I$. □

Notice that if $f_1 = x - \alpha$, then $L_1 = K$. The fields $L_i, 1 \leq i \leq r$ are called *principal subfields* of the extension K/k and Theorem 2.13 shows that $\{L_1, \ldots, L_r\}$ is a set of intersection-generating subfields. Another important property is that the set $\{L_1, \ldots, L_r\}$ is independent of the extension $\tilde{K}$. We might take

$\tilde{K} = K$, however, in some cases, it might be interesting to choose a different extension. For instance, in the number field case, computing the factorization of f over $\mathbb{Q}(\alpha)$ is a hard task. By choosing an appropriate prime p, we can embed $\mathbb{Q}(\alpha)$ into $\tilde{K} := \mathbb{Q}_p$, the p-adic completion of $\mathbb{Q}$. The factorization of f over $\tilde{K} = \mathbb{Q}_p$ can then be approximated by Hensel lifting a factorization of $f \bmod p$. Further discussion on how to choose $\tilde{K}$ is presented at the beginning of Chapter 4.

To find all subfields of the extension K/k, it suffices to compute the intersection of all subsets of $\{L_1, \ldots, L_r\}$. If we naïvely try to compute the intersection of all subsets of $\{L_1, \ldots, L_r\}$, we might compute the same subfield several times. In [51], an algorithm is presented that avoids computing the same subfield several times. This is done by associating a vector $e = (e_1, \ldots, e_r)$ to every subfield of L of K/k, where $e_i = 1$ if $L \subseteq L_i$ and $e_i = 0$ otherwise.

To actually compute the intersections, the authors of [51] represented every subfield L of K/k as a k-vector subspace of K. The intersection of subfields L_i and L_j is then given by intersecting the corresponding subspaces. Here lies our improvement, which we present in the next chapter. We propose a different representation for the subfields, namely, a partition of the set $\{1, \ldots, r\}$, such that the intersections can now be computed efficiently.

Example 2.14. *Let $f = x^8 - 5 \in \mathbb{Q}[x]$. Let α be a root of f and consider the extension $\mathbb{Q}(\alpha)/\mathbb{Q}$. Over $\mathbb{Q}(\alpha)$, f has the following factorization*

$$f = (x - \alpha)(x + \alpha)(x^2 + \alpha^2)(x^4 + \alpha^4).$$

Hence, this tells us that we have 4 principal subfields. Let $g(\alpha) = g_0 + g_1\alpha + \cdots + g_7\alpha^7$, where $g_0, \ldots, g_7 \in \mathbb{Q}$, be an arbitrary element in $\mathbb{Q}(\alpha)$. Let us compute the principal subfield L_2 corresponding to $f_2 = x + \alpha$ (the subfield L_1 corresponding to $f_1 = x - \alpha$ we already know to be $\mathbb{Q}(\alpha)$). The element $g(\alpha)$ is in L_2 if, and only if,

$$g(x) \bmod f_2 = g(\alpha).$$

The left-hand side of this equation is $g_0 - g_1\alpha + \cdots - g_7\alpha^7$ *and the whole equation is equivalent to* $2g_1\alpha + 2g_3\alpha^3 + \cdots + 2g_7\alpha^7 = 0$. *This equation gives us a set of linear conditions on* $g_0, \ldots, g_7$ *such that* $g(\alpha) = g_0 + g_1\alpha + \cdots + g_7\alpha^7$ *is an element of* L_2. *In this case, the conditions are* $g_1 = g_3 = g_5 = g_7 = 0$, *while* g_0, g_2, g_4, g_6 *can be arbitrary elements. Therefore,*

$$L_2 = \{g_0 + g_2\alpha^2 + g_4\alpha^4 + g_6\alpha^6 \ : \ g_0, g_2, g_4, g_6 \in \mathbb{Q}\}.$$

Similarly, the subfield L_3, *corresponding to the factor* $f_3 = x^2 + \alpha^2$, *is given by*

$$L_3 = \{g_0 + g_4\alpha^4 \ : \ g_0, g_4 \in \mathbb{Q}\}$$

and the linear conditions are $g_1 = g_2 = g_3 = g_5 = g_6 = g_7 = 0$ *and* g_0, g_4 *arbitrary. The subfield* L_4 *corresponding to* $f_4 = x^4 + \alpha^4$ *is* $\mathbb{Q}$. *In order to find all subfields, it remains to compute the intersection of all combinations of* L_1, L_2, L_3 *and* L_4, *which can be done by solving the linear conditions of every subfield in this combination simultaneously. In this case, it is easy to see that* $L_4 \subseteq L_3 \subseteq L_2 \subseteq L_1$ *and hence, these are the only subfields of* $\mathbb{Q}(\alpha)/\mathbb{Q}$.

Unfortunately, not all subfields of an arbitrary extension K/k have such nice expressions as in Example 2.14. The linear conditions on $g_0, \ldots, g_{n-1}$ can be very complex and if we have a large number of subfields (recall that this number is not polynomially bounded), computing all intersections can be time consuming.

2.4.2 The Number Field Case

We will now focus on the number field case. Let $\mathbb{Q}(\alpha)/\mathbb{Q}$ be a separable extension of finite degree n and let $f \in \mathbb{Q}[x]$ be the minimal polynomial of α over $\mathbb{Q}$. One can compute the principal subfields by factoring $f \in \mathbb{Q}[x]$ over $\mathbb{Q}(\alpha)$. However, this is not always a simple task. To avoid factorization over $\mathbb{Q}(\alpha)$, we embed the field $\mathbb{Q}(\alpha)$ into a p-adic completion of $\mathbb{Q}$. Comparison between these two approaches (factorization over $\mathbb{Q}(\alpha)$ and p-adic embedding) is further discussed in Chapter 4.

Let p be a prime number such that $f \bmod p$ is separable (i.e., $p \nmid \operatorname{disc}(f)$) and such that $f \bmod p$ has a linear factor. Let $f \equiv \bar{f}_1 \cdots \bar{f}_r \bmod p$ be the factorization of $f \bmod p$, with $\bar{f}_1 = x - \bar{\alpha}$. By Hensel Lifting, this factorization corresponds to a factorization $\hat{f}_1 \cdots \hat{f}_r$ over $\mathbb{Q}_p$, where $\hat{f}_1 = x - \hat{\alpha}$ and $\hat{\alpha} \equiv \bar{\alpha} \bmod p$. By mapping $\alpha \mapsto \hat{\alpha}$, we get an embedding of $\mathbb{Q}(\alpha)$ into $\mathbb{Q}_p$.

We cannot compute the factorization $\hat{f}_1 \cdots \hat{f}_r$ of f over $\mathbb{Q}_p$, however, by Hensel Lifting, we can compute an approximation of this factorization with accuracy a, for any $a \in \mathbb{N}$. That is, we can compute polynomials $f_1^{(a)}, \ldots, f_r^{(a)} \in p^a\mathbb{Z}[x]$ such that $f \equiv f_1^{(a)} \cdots f_r^{(a)} \bmod p^a$ and $\hat{f}_i \equiv f_i^{(a)} \bmod p^a$. For each of the factors $\hat{f}_i$, we need to compute the principal subfield L_i, that is, we want to find a basis of the $\mathbb{Q}$-subspace L_i of $\mathbb{Q}(\alpha)$. The idea is to design a lattice where elements of L_i correspond to short vectors in this lattice. The LLL algorithm will then determine these vectors.

First, we need a basis of $\mathbb{Q}(\alpha)$ as a $\mathbb{Q}$-vector space. The could use the *canonical basis* $\{1, \alpha, \ldots, \alpha^{n-1}\}$, as every element $g(\alpha) \in \mathbb{Q}(\alpha)$ can be written as $g_0 + g_1\alpha + \cdots + g_{n-1}\alpha^{n-1}$, for some $g_0, \ldots, g_{n-1} \in \mathbb{Q}$. However, we choose the basis

$$\left\{ \frac{1}{f'(\alpha)}, \frac{\alpha}{f'(\alpha)}, \ldots, \frac{\alpha^{n-1}}{f'(\alpha)} \right\},$$

which is called the *rational representation basis*. This basis allows to prove better bounds for elements of L_i inside our lattice. In other words, the following is true.

Theorem 2.15 ([51], Theorem 12)**.** *Let L_i be a principal subfield of $\mathbb{Q}(\alpha)/\mathbb{Q}$ of degree m_i over $\mathbb{Q}$. For $\beta \in \mathbb{Q}(\alpha)$ with $\beta = \sum b_i \frac{\alpha^i}{f'(\alpha)}$, define $\boldsymbol{v}_\beta := (b_0, b_1, \ldots, b_{n-1})$. There exists m_i linearly independent algebraic numbers $\beta_1, \ldots, \beta_{m_i} \in L_i$ such that $\|\boldsymbol{v}_{\beta_j}\|_2 \leq n^2\|f\|_2$, for every $j = 1, \ldots, m_i$.*

The lattice B_i, which contains the elements $\beta_1, \ldots, \beta_{m_i}$, is generated by the columns of the following $(n + d_i) \times (n + d_i)$ matrix

$$B_i := \begin{pmatrix} 1 & & & & & \\ & \ddots & & & & \\ & & 1 & & & \\ c_{0,0} & \cdots & c_{0,n-1} & p^a & & \\ \vdots & \ddots & \vdots & & \ddots & \\ c_{d_i-1,0} & \cdots & c_{d_i-1,n-1} & & & p^a \end{pmatrix}$$

where $d_i = \deg(\bar{f}_i)$ and $c_{k,j}$ is the k-th coefficient of the image of $\frac{\alpha^j}{f'(\alpha)}$ under $\Phi_i - \mathrm{id}$ reduced modulo p^a. Notice that, if $\beta = \sum b_i \frac{\alpha^i}{f'(\alpha)} \in L_i$, its image under $\Phi_i - \mathrm{id}$ is 0. Furthermore, the vector $(b_0, b_1, \ldots, b_{n-1}, 0, \ldots, 0)$ is a *short vector* of the lattice B_i. If the accuracy a is appropriate, the LLL algorithm is then able to detect these elements, returning m_i algebraic independent elements $\beta_1, \ldots, \beta_{m_i}$ comprising a basis of L_i as a $\mathbb{Q}$-vector space.

Remark 2.16. *An initial guess for the Hensel lifting accuracy a can be computed using the ideas of Section 1.3.1. A short vector has norm less than $n^2\|f\|_2$, according to Theorem 2.15. Hence, the determinant of B_i (and the number a) should satisfy*

$$(2^{(n+d_i-1)/2} n^2 \|f\|_2)^n \leq \det(B_i) = p^{ad_i}.$$

Let V be the vector space generated by the elements $\beta_1, \ldots, \beta_{m_i}$. Since we are only using approximations $f_i^{(a)}$ of $\hat{f}_i$, for some $a \in \mathbb{N}$, it is possible that $L_i \neq V$ (for instance, if the accuracy a is not high enough). However, we always have $L_i \subseteq V$. For the algorithm to work, we need to ensure that equality holds. Let us assume for a moment that $L_i = V$. The authors of [51] propose to attempt to construct g_i, the minimal polynomial of α over L_i. Since $g_i \mid f$, it follows that the image $\hat{g}_i$ of g_i over $\mathbb{Q}_p$ is given by $\prod_{j \in J} \hat{f}_j \in \mathbb{Q}_p[x]$, where $J \subseteq \{1, \ldots, r\}$ is given by

$$J = \{j \in \{1, \ldots, r\} : (\Phi_j - \mathrm{id})(\beta_k) = 0, \text{ for all } k = 1, \ldots, m_i\}$$

and correspond to the principal subfields that contain L_i. Notice that $\{1, i\} \subseteq J$. Furthermore, since the coefficients of g_i are elements of L_i, each coefficient can be written as a $\mathbb{Q}$-linear combination of $\beta_1, \ldots, \beta_{m_i}$. We can then use an approximation of the image of this coefficient over $\mathbb{Q}_p$ (that we have from an approximation of $\hat{g}_i$) and LLL to compute this combination. This allows us to compute $g_i \in L_i[x]$.

When $\beta_1, \ldots, \beta_{m_i}$ is not a $\mathbb{Q}$-basis of L_i, that is, $L_i \subsetneq V$ (which we do not know *a priori*), the construction of g_i might fail. If this happens, we know for sure that $L_i \neq V$ and we need to increase the precision a in the Hensel Lifting. Suppose we are able to construct $g_i \in \mathbb{Q}(\alpha)[x]$. Next, we check whether $g_i \mid f$ over $\mathbb{Q}(\alpha)$. If this fails, then again we need to increase the precision a and compute g_i again. Otherwise, we might consider the subfield L_{g_i} of $\mathbb{Q}(\alpha)/\mathbb{Q}$ (see Equation 3.1). Since $f_i \mid g_i$, it follows that $L_{g_i} \subseteq L_i$ (see Equation 3.2). Finally, we need to verify that $\beta_j \in L_{g_i}$, for $j = 1, \ldots, m_i$. This ensures that $V \subseteq L_{g_i}$. Thus, if all tests pass, we have $L_i = L_{g_i} = V$ and $g_i \in L_i[x]$ is the minimal polynomial of α over L_i.

3 COMPUTING INTERSECTIONS EFFICIENTLY

Let k be a field and let $K = k(\alpha)$ be a separable field extension of degree n. Let $f \in k[x]$ be the minimal polynomial of α over k. As we have seen in the previous chapter, there are several methods to compute the subfields of a field extension. Many of these methods take advantage of the connection between subfields of a field extension and subgroups of the Galois group $\mathrm{Gal}(f)$, see [17, 25, 26]. Other methods involve resolvents, such as [32], and symmetric functions, [13]. The `POLRED` algorithm [14] may also find subfields, but it is not guaranteed to work. According to [51], there exists a set of so called *intersection-generating subfields* $\{L_1, \ldots, L_r\}$, with $r \leq n$, such that every subfield of K/k is the intersection of a subset of $\{L_1, \ldots, L_r\}$ (recall Theorem 2.13). Thus, the problem of computing all subfields of K/k can be done in two phases:

Phase I: Compute the principal subfields $L_1, \ldots, L_r$ of K/k.

Phase II: Compute all subfields by computing intersections of $L_1, \ldots, L_r$.

In practice, phase I usually dominates the CPU time. However, in the theoretical complexity, the reverse is true: for $k = \mathbb{Q}$, phase I is polynomial time but phase II depends on the number of subfields, which is not polynomially bounded. Our objective is to speed up phase II. This improves the theoretical complexity (see Theorem 3.47). It also improves practical performance, although the improvement is only significant when the number of subfields is large (see Section 4.2).

When computing all subfields, we might compute the same subfields several times. An algorithm to compute the intersection of all subsets of $\{L_1, \ldots, L_r\}$ which avoids computing the same subfield several times is given in [51]. We will use the same algorithm, however, our improvement relies on how each intersection is

computed. While in [51], every intersection $L_i \cap L_j$ is computed as the intersection of k-vector subspaces of K, we first associate a partition to every principal subfield and then compute the partition corresponding to the intersection of the subfields. An important property of this new intersection algorithm is that it does not use computations in k. This will be made clearer in the next sections.

3.1 Representing Subfields with Partitions

As usual, let K/k be a separable field extension of finite degree n. Furthermore, let $\alpha \in K$ be a primitive element and let $f \in k[x]$ be the minimal polynomial of α over k. In this Section, we will show how one can uniquely associate a partition of a set of indexes to every subfield L of K/k.

3.1.1 Subfield Polynomial

Let $\tilde{K}$ be an extension of K and let $f = \tilde{f}_1 \cdots \tilde{f}_{\tilde{r}}$, where $\tilde{f}_i \in \tilde{K}[x]$, be the factorization of f in $\tilde{K}[x]$. Let $g \in \tilde{K}[x]$ with $g \mid f$. Since K/k is separable, g is separable as well. The following set is a subfield of K

$$L_g := \{h(\alpha) : h \in k[x]_{<n},\ h(x) \equiv h(\alpha) \bmod g\} \subseteq K, \tag{3.1}$$

where $k[x]_{<n}$ denotes the set of polynomials over k with degree at most $n-1$. This follows from the fact that f is separable and thus,

$$\text{if } g = g_1 g_2 \mid f \text{ then } L_g = L_{g_1} \cap L_{g_2}. \tag{3.2}$$

According to [51], the set $\{L_{\tilde{f}_1}, \ldots, L_{\tilde{f}_{\tilde{r}}}\}$ is independent of the choice of the extension $\tilde{K}$ and is called the set of *principal subfields* of K/k (recall Theorem 2.13). Notice that we can always take $\tilde{K} = K$, however, it can be more fruitful to choose a non-trivial extension of K (this is further discussed in Chapter 4). All results in this

chapter hold if we consider $\tilde{f}_1, \ldots, \tilde{f}_r$ irreducible factors of f over an extension $\tilde{K}$ of K however, for simplicity, we will henceforth use $\tilde{K} = K$.

Lemma 3.1. *If $g \in L[x]$ is the minimal polynomial of α over L, for some subfield L of K/k, then*

(*i*) $\deg(g) \cdot [L : k] = n$.

(*ii*) $L = \{h(\alpha) \in K : h(x) \equiv h(\alpha) \bmod g\} = L_g$.

(*iii*) *The coefficients of g generate L as a k-algebra.*

Proof. Item (*i*) follows directly from the Short Tower Law $[K : k] = [K : L] \cdot [L : k]$. Item (*ii*) follows from Theorem 2.13. Finally, the minimal polynomial of α over L has degree $[L(\alpha) : L] = [K : L]$. Let $\tilde{L}$ to be the field generated by the coefficients of g. Since $\tilde{L} \subseteq L$, the minimal polynomial $\tilde{g}$ of α over $\tilde{L}$ has degree $\deg(\tilde{g}) = [K : \tilde{L}] \geq [K : L] = deg(g)$. On the other hand, $g \in \tilde{L}[x]$ and $g(\alpha) = 0$ and therefore, $\tilde{g} \mid g$. Thus, $g = \tilde{g}$ and therefore $[K : \tilde{L}] = [K : L]$. That is, $L = \tilde{L}$. □

Lemma 3.2. *Let $h(x) \in k[x]$ and let $L = k(h(\alpha))$ be a subfield of K/k. The minimal polynomial of α over L is the* gcd *of f and $h(x) - h(\alpha)$.*

Proof. Let g be the gcd, d its degree, and let $\tilde{g}$ be the minimal polynomial of α over L. The polynomials f, $h(x) - h(\alpha)$, and g, are elements of $L[x]$ and have α as a root, and are thus divisible by $\tilde{g}$. It remains to show that g and $\tilde{g}$ have the same degree. If $\alpha_1, \ldots, \alpha_n$ are the roots of f in a splitting field, then the roots of g are those α_i for which $h(\alpha_i) = h(\alpha)$. So d is the number of i for which $h(\alpha_i) = h(\alpha)$. The degree $[L : k]$ is the number of distinct $h(\alpha_i)$, which is n/d. The degree of $\tilde{g}$ is $[K : L] = n/[L : k] = d$. □

Lemma 3.3. *Let L_g be a subfield of K/k, for some $g \mid f$ and let $\tilde{g}$ be the minimal polynomial of α over L_g. Then $g \mid \tilde{g}$.*

Proof. Let $h \in k[x]$ be such that $L_g = k(h(\alpha))$. By the previous lemma, $\tilde{g} = \gcd(f, h(x) - h(\alpha))$ which is divisible by g. □

Finally, we show the main theorem of this section, regarding the minimal polynomial of α over a subfield L of K/k.

Theorem 3.4. *Let $g \in K[x]$ be a monic polynomial such that $x - \alpha \mid g \mid f$. The following are equivalent*

(1) *g is the minimal polynomial of α over L, for some subfield L of K/k.*

(2) *$deg(g) \cdot [k(coeffs(g)) : k] \leq n$.*

(3) *$deg(g) \cdot [L_g : k] = n$.*

(4) *The coefficients of g generate L_g as a k-algebra.*

(5) *$g \in L_g[x]$.*

(6) *$g = \gcd(f, h(x) - h(\alpha))$, for some $h(x) \in k[x]_{<n}$.*

Proof. The equivalence 1) $\Leftrightarrow$ 6) follows from Lemma 3.2. We shall prove that 1) $\Rightarrow$ 2) $\Rightarrow$ 3) $\Rightarrow$ 4) $\Rightarrow$ 5) $\Rightarrow$ 1). Suppose 1), then 2) follows from Lemma 3.1. Now, suppose 2). Let $\tilde{g}$ be the minimal polynomial of α over $L := k(\text{coeffs}(g))$. Thus, $L = L_{\tilde{g}}$. Moreover, since $g, \tilde{g} \in L[x]$ and $g(\alpha) = 0$, we have $\tilde{g} \mid g$. Hence,

$$n = \deg(\tilde{g}) \cdot [L_{\tilde{g}} : k] = \deg(\tilde{g}) \cdot [L : k] \leq \deg(g) \cdot [L : k] \leq n.$$

Thus, $g = \tilde{g}$. Item 3) then follows from Lemma 3.1 (i). If 3) holds, let $\tilde{g}$ be the minimal polynomial of α over L_g. Thus, $L_{\tilde{g}} = L_g$. By Lemma 3.3, $g \mid \tilde{g}$ and thus,

$$n = \deg(g) \cdot [L_g : k] = \deg(g) \cdot [L_{\tilde{g}} : k] \leq \deg(\tilde{g}) \cdot [L_{\tilde{g}} : k] = n.$$

Thus, $g = \tilde{g}$. Item 4) then follows from Lemma 3.1 (iii). If 4) holds, then 5) holds trivially. Finally, suppose that 5) holds. Let $\tilde{g}$ be the minimal polynomial of α over

L_g. By Lemma 3.3 it follows that $g \mid \tilde{g}$. On the other hand, since $g \in L_g[x]$ and $g(\alpha) = 0$, we have $\tilde{g} \mid g$. Therefore, $g = \tilde{g}$ and item 1) follows. □

Definition 3.5. *If any of the conditions in Theorem 3.4 holds, then g is called a* subfield polynomial*. Furthermore, we call g the* subfield polynomial *of the subfield L, which coincides with $k(coeffs(g))$ in* (2)*, L_g in* (3)*,* (4)*,* (5) *and $k(h(\alpha))$ in* (6)*.*

Notice that the subfield polynomial of K is $x - \alpha$ and the subfield polynomial of k is f. In what follows we shall frequently use conditions 1, 4 and 6.

3.1.2 From a Subfield to a Partition

Let $f = f_1 \cdots f_r$ be a partial factorization of f over K (f_i not necessarily irreducible). In this subsection we define a partition P_L of $\{1, \ldots, r\}$ corresponding to a given subfield L of K/k. Recall that a partition $P = \{P^{(1)}, \ldots, P^{(t)}\}$ of $\{1, \ldots, r\}$ satisfies

1. $\bigcup P^{(i)} = \{1, \ldots, r\}$.
2. $P^{(i)} \neq \emptyset,\ 1 \leq i \leq t$.
3. $P^{(i)} \bigcap P^{(j)} = \emptyset$, for every $i \neq j$.

Definition 3.6. *Let $P = \{P^{(1)}, \ldots, P^{(t)}\}$ be a partition of $\{1, \ldots, r\}$. We call P-*products *(with respect to the factorization $f_1 \cdots f_r$ of f) the polynomials defined by $\prod_{i \in P^{(j)}} f_i$, $1 \leq j \leq t$.*

For instance, if $P = \{\{1, 2, 3\}, \{4, 7\}, \{5, 6\}\}$ is a partition of $\{1, \ldots, 7\}$, then the P-products (w.r.t. the factorization $f_1 \cdots f_7$ of f) are the polynomials $f_1 f_2 f_3$, $f_4 f_7$ and $f_5 f_6$. We might also mention the *size* $|P| \in \mathbb{N}$ of a partition P, which is the number of elements of P.

Definition 3.7. *For every subfield L of K/k, let $P_L = \{P_L^{(1)}, \ldots, P_L^{(t)}\}$ be the partition of $\{1, \ldots, r\}$ satisfying*

1. *The P_L-products are in $L[x]$.*
2. *$|P_L|$ is maximal satisfying 1.*

We say that P_L is the *partition defined* by L. Notice that this partition depends on the factorization $f_1, \ldots, f_r$ of f. We now prove that P_L is well defined.

Notation 3.8. *Denote by $\{f_1, \ldots, f_r\}^\pi$ the set of all products $\{\prod f_i^{e_i} : e_i \in \{0, 1\}\}$.*

Lemma 3.9. *Let P be a partition of $\{1, \ldots, r\}$ and let F_i, $1 \leq i \leq t$, be the P-products. Let L be a subfield of K/k. Then $P = P_L$ if and only if*

$$\{f_1, \ldots, f_r\}^\pi \bigcap L[x] = \{F_1, \ldots, F_t\}^\pi. \tag{3.3}$$

Proof. Suppose that $P = P_L$, that is, P satisfies the properties of Definition 3.7 for a subfield L. Then $F_i \in L[x]$ and hence

$$\{F_1, \ldots, F_t\}^\pi \subseteq \{f_1, \ldots, f_r\}^\pi \bigcap L[x].$$

Conversely, let $F \in \{f_1, \ldots, f_r\}^\pi \bigcap L[x]$. Then $\gcd(F, F_i) \in L[x]$, for every $1 \leq i \leq t$. Furthermore, $\gcd(F, F_i) \in \{1, F_i\}$ (otherwise, we could replace $P_L^{(i)}$ in P_L by two non-empty sets, which contradicts the maximality of t). Therefore, $F \in \{F_1, \ldots, F_t\}^\pi$ and Equation (3.3) follows.

Now let P be a partition of $\{1, \ldots, r\}$ and assume Equation (3.3). We need to prove that P satisfies conditions (1) and (2) of Definition 3.7. From Equation (3.3), it follows that $F_i \in L[x]$. Condition (2) follows from the fact that f is separable and any partition P satisfying Definition 3.7 (1) defines $|P|$ multiplicatively independent elements of $\{f_1, \ldots, f_r\}^\pi \bigcap L[x]$ (*i.e.*, the gcd of any two of them equals 1). By Equation (3.3), the maximal number of multiplicatively independent elements of $\{f_1, \ldots, f_r\}^\pi \bigcap L[x]$ is t. □

This means that if P'_L is any other partition that satisfies Definition 3.7, then the P'_L-products are in $\{F_1, \ldots, F_t\}^\pi$. Hence $|P'_L| \leq |P_L|$ and therefore, $|P'_L| = |P_L|$, since both P_L and P'_L satisfy Definition 3.7. Now if $P_L \neq P'_L$, then using the same argument as in the first part of the proof would allow us to construct a partition $\tilde{P}$ satisfying Definition 3.7 (1) and such that $|\tilde{P}| > |P_L|$, which contradicts the maximality of $|P_L|$. That is, the partition P_L is well defined.

Example 3.10. *Consider $f = x^8 - 5 \in \mathbb{Q}[x]$ and α a root of f. The factorization of f over $\mathbb{Q}(\alpha)$ is given by $f = (x-\alpha)(x+\alpha)(x^2+\alpha^2)(x^4+\alpha^4)$. From Example 2.14, the subfield corresponding to $f_2 = x + \alpha$ is given by*

$$L_2 = \{g_0 + g_2\alpha^2 + g_4\alpha^4 + g_6\alpha^6 \ : \ g_0, g_2, g_4, g_6 \in \mathbb{Q}\}.$$

It is not difficult to see that $L_2 = \mathbb{Q}(\alpha^2)$. Let us find the partition of $\{1,2,3,4\}$ corresponding to this subfield. The factor $f_1 = x - \alpha$ is not in $L_2[x]$. Hence, we need to multiply f_1 with some other factors of f such that the product lies in $L_2[x]$. By considering $f_1 f_2 = x^2 - \alpha^2$, we see that $f_1 f_2 \in L_2[x]$. Since $f_3, f_4 \in L_2[x]$, we have a partition $\{\{1,2\},\{3\},\{4\}\}$ of $\{1,2,3,4\}$ Notice that this partition also satisfies item 2. of Definition 3.7 and hence, is the partition defined by L_2. Similarly, the partition of $L_3 = \mathbb{Q}(\alpha^4)$ is $\{\{1,2,3\},\{4\}\}$.

Hence, every subfield defines a single partition of $\{1, \ldots, r\}$. However, if we consider any factorization $f_1, \ldots, f_r$ of f over K, then different subfields might correspond to the same partition. For instance, if we had chosen the factorization $f = (x^4 - \alpha^4)(x^4 + \alpha^4)$ in Example 3.10, then the partitions of L_2 and L_3 would have been $\{\{1\},\{2\}\}$. That is, we need some conditions on the factorization that guarantee that different subfields define different partitions. As we shall see, the irreducible factorization has this property, but it is not the only one.

3.1.3 Subfield Factorization

Let $f = f_1 \cdots f_r$ be a partial factorization of f, that is, $f_i, 1 \leq i \leq r$, is not necessarily irreducible. In Section 3.1.2 we defined a partition P_L of $\{1, \ldots, r\}$ for each subfield L of K/k. In this subsection we will define the concept of a *subfield factorization*. This factorization has the property that different subfields define different partitions.

Definition 3.11. *Let $f_1, \ldots, f_r \in K[x]$, not necessarily irreducible, be such that $f = f_1 \cdots f_r$. Then $f_1, \ldots, f_r$ is called a* subfield factorization *of f if $f_1 = x - \alpha$ and $\{f_1, \ldots, f_r\}^\pi$ contains the subfield polynomial of every principal subfield of K/k.*

The full factorization of f into irreducible factors over K is always a subfield factorization of f, but the converse need not be true. For instance, if K/k has no nontrivial subfields, then $\{x - \alpha, f/(x - \alpha)\}$ is a subfield factorization of f, even if $f/(x - \alpha)$ is reducible. The reason for defining a subfield factorization is that, in some cases, different irreducible factors might give the same principal subfield. That is, let $f_1, \ldots, f_r$ be the irreducible factors of f and suppose that f_1, f_2 are such that $L_{f_1} = L_{f_2} = L$. Hence, if $g = f_1 f_2$, then (recall Equation 3.2) $L_g = L_{f_1} \cap L_{f_2} = L$. That is, if we replace f_1, f_2 by g, then the (partial) factorization $g, f_3, \ldots, f_r$ of f still yields all principal subfields of K/k. That is, the set $\{L_g, L_{f_3}, \ldots, L_{f_r}\}$ is still an intersection-generating set for K/k. Another advantage is that we do not necessarily need to compute the irreducible factorization of f over K, giving us some room for improvement (see Chapter 4).

Lemma 3.12. *If $f_1, \ldots, f_r$ is a subfield factorization of f, then $\{L_{f_1}, \ldots, L_{f_r}\}$ is a set of intersection-generating subfields of K/k. Moreover, if g is a subfield polynomial (of any subfield of K/k), then $g \in \{f_1, \ldots, f_r\}^\pi$.*

Proof. Let L be a principal subfield and let g be the subfield polynomial of L. By Lemma 3.1, we have $L = L_g$. Since $f_1, \ldots, f_r$ is a subfield factorization, then

$g \in \{f_1, \ldots, f_r\}^\pi$, that is, $g = f_{i_1} \cdots f_{i_s}$, for polynomials $f_{i_1}, \ldots, f_{i_s}$ in $\{f_1, \ldots, f_r\}$. Hence, $L = L_g = L_{f_{i_1}} \cap \cdots \cap L_{f_{i_s}}$. Thus, we showed that every principal subfield is the intersection of some L_{f_i}, $1 \leq i \leq r$. Since every subfield of K/k is the intersection of principal subfields, it follows that $\{L_{f_1}, \ldots, L_{f_r}\}$ is an intersection-generating set.

For the second claim, let g be a subfield polynomial and let $\tilde{g} = \prod_{f_i | g} f_i$. Hence, $\tilde{g} \mid g$. We want to show that $g = \tilde{g}$ and hence, it suffices to prove that $g \mid \tilde{g}$. Let $h \in K[x]$ be an irreducible polynomial such that $h \mid g$. Let $\tilde{h}$ be the subfield polynomial of L_h. Since $h \mid g$, it follows that $L_g \subseteq L_h$ (see Equation 3.2) and hence $h \mid \tilde{h} \mid g$ (see Lemma 3.3). On the other hand, since h is irreducible, L_h is a principal subfield and hence, $\tilde{h} \in \{f_1, \ldots, f_r\}^\pi$. Therefore, $\tilde{h} \mid \tilde{g}$ and hence $h \mid \tilde{g}$. □

Lemma 3.12 shows that $f_1, \ldots, f_r$ is a subfield factorization if, and only if, $\{L_{f_1}, \ldots, L_{f_r}\}$ is an intersection-generating set for K/k. Let P_L be the partition corresponding to a subfield L of K/k, as defined in Definition 3.7. We shall number the $P_L^{(i)}$ in such a way that $1 \in P_L^{(1)}$ so that $f_1 = x - \alpha$ divides the *first P_L-product* $\prod_{i \in P_L^{(1)}} f_i$. Another important property of a subfield factorization is that it allows us to prove that $P_L = P_{L'}$ if and only if $L = L'$.

Lemma 3.13. *Let $f_1, \ldots, f_r$ be a subfield factorization of f, let L be a subfield of K/k and P_L its partition. Then the* first P_L-product *is the subfield polynomial of L. In particular, $L = L'$ if and only if, $P_L = P_{L'}$.*

Proof. Let h be the first P_L-product and g be the subfield polynomial of L. By Definition 3.7 (1), it follows that $h \in L[x]$. Furthermore, since $1 \in P_L^{(1)}$, we have $x - \alpha \mid h$, that is, $h(\alpha) = 0$. Since g is the minimal polynomial of α over L, we have $g \mid h$. If $g \neq h$, then there exists $\tilde{h} \in L[x]$ such that $h = g\tilde{h}$. This means that we can replace $P_L^{(1)}$ by two non-empty sets (one corresponding to g and the other, to $\tilde{h}$). The resulting partition would also satisfy Definition 3.7 (1), which contradicts the maximality of $|P_L|$. Hence, $g = h$. Therefore, if $f_1, \ldots, f_r$ is

a subfield factorization, then for every partition P_L, the subset $P_L^{(1)} \subseteq \{1, \ldots, r\}$ encodes the subfield polynomial of L. In particular, if $L \neq L'$ then $P_L^{(1)} \neq P_{L'}^{(1)}$ and hence $P_L \neq P_{L'}$. On the other hand, if $L = L'$, then $P_L = P_{L'}$, because P_L is well defined. Hence, every subfield L is uniquely represented by P_L. □

Remark 3.14. *Representing subfields using partitions has many advantages:*

1. *Given P_L, one can quickly find elements of L, for instance, by computing a coefficient of a P_L-product, or by computing a P_L-product evaluated at $x = c$, for some $c \in k$. Section 3.4.2 gives a quick test to see if the elements obtained in this way generate L as a k-algebra.*

2. *$P_L^{(1)}$ immediately gives the subfield polynomial in partially factored form.*

3. *Given P_L and $P_{L'}$, it is trivial (see Lemma 3.16) to check whether $L \subseteq L'$. Section 3.2 shows that one can quickly compute the partition for $L \bigcap L'$. The degree $[L : k]$ can be read from $P_L^{(1)}$ with Theorem 3.4 (3).*

4. *P_L only requires $\mathcal{O}(r \log r)$ bits of storage. This means that when a subfield factorization $f_1, \ldots, f_r$ of f is given, one only needs $\mathcal{O}(mr \log r)$ additional bits to represent the complete subfield lattice, where m is the number of subfields.*

Hence, provided that $f_1 \cdots f_r$ is a subfield factorization of f, every subfield of K/k defines a unique partition of $\{1, \ldots, r\}$ and different subfields define different partitions. Notice that the converse is not true, that is, not every partition of $\{1, \ldots, r\}$ defines a subfield of K/k.

3.2 Intersecting Subfields represented by Partitions

Since we are interested in the intersection of principal subfields, and we have just found a representation of subfields in terms of partitions, we would like to determine the partition corresponding to the intersection of two subfields L and L'. That is, given P_L and $P_{L'}$, we want to find $P_{L\cap L'}$. In this section we give a description of $P_{L\cap L'}$, as well as an algorithm to compute this partition.

3.2.1 The partition of $L \cap L'$

Definition 3.15. *A partition $P = \{P^{(1)}, \ldots, P^{(s)}\}$ is a* refinement *of a partition $Q = \{Q^{(1)}, \ldots, Q^{(t)}\}$ (or simply P* refines *Q) if every $Q^{(i)}$, $1 \leq i \leq t$, can be written as a union of some of the $P^{(j)}$, $1 \leq j \leq s$.*

Lemma 3.16. *Let L, L' be two subfields of K/k and let P_L and $P_{L'}$ be their corresponding partitions of $\{1, \ldots, r\}$. Then $L \subseteq L'$ if, and only if, $P_{L'}$ refines P_L.*

Proof. If $P_{L'}$ refines P_L, then $P_{L'}^{(1)} \subseteq P_L^{(1)}$. This means that the subfield polynomial of L is divisible by the subfield polynomial of L'. Equation (3.2) implies that $L \subseteq L'$. The converse follows from Lemma 3.9, that is, if $L \subseteq L'$, then the P_L-products are contained in $\{P_{L'}\text{-products}\}^{\pi}$. This only happens when $P_{L'}$ refines P_L. □

Definition 3.17. *Let P be a partition of $\{1, \ldots, r\}$. We say that P is the finest partition satisfying property X if P satisfies X and, for every partition Q satisfying X, P refines Q.*

Theorem 3.18. *Let L, L' be two subfields of K/k and let P_L and $P_{L'}$ be their corresponding partitions. Then the partition corresponding to $L \cap L'$ is the finest partition P for which both P_L and $P_{L'}$ refine P.*

Proof. Let $P = P_{L\cap L'} = \{P^{(1)}, \ldots, P^{(t)}\}$ satisfy items (1) and (2) of Definition 3.7. We need to prove that P is the finest partition such that $P_L = \{P_L^{(1)}, \ldots, P_L^{(s)}\}$ and

$P_{L'} = \{P_{L'}^{(1)}, \ldots, P_{L'}^{(s')}\}$ refine P. The fact that P_L and $P_{L'}$ refine P follows from Lemma 3.16. To prove that P is the finest partition with this property, let Q be a partition refined by both P_L and $P_{L'}$. We need to prove that P refines Q. Pick $Q^{(i)}$ and let $P^{(j)}$ be such that $R := Q^{(i)} \cap P^{(j)} \neq \emptyset$. We need to prove that $P^{(j)} \subseteq Q^{(i)}$ or, equivalently, $R = P^{(j)}$. Since P_L and $P_{L'}$ refine P, there exist subsets $J_1 \subseteq \{1, \ldots, s\}$ and $J_2 \subseteq \{1, \ldots, s'\}$ such that

$$P^{(j)} = \bigcup_{k \in J_1} P_L^{(k)} = \bigcup_{k \in J_2} P_{L'}^{(k)}.$$

Likewise, there exist $I_1 \subseteq \{1, \ldots, s\}$ and $I_2 \subseteq \{1, \ldots, s'\}$ such that

$$Q^{(i)} = \bigcup_{k \in I_1} P_L^{(k)} = \bigcup_{k \in I_2} P_{L'}^{(k)}.$$

Therefore,

$$R = Q^{(i)} \cap P^{(j)} = \bigcup_{k \in I_1 \cap J_1} P_L^{(k)} = \bigcup_{k \in I_2 \cap J_2} P_{L'}^{(k)} \tag{3.4}$$

and

$$P^{(j)} \setminus R = \bigcup_{k \in J_1 \setminus I_1} P_L^{(k)} = \bigcup_{k \in J_2 \setminus I_2} P_{L'}^{(k)}. \tag{3.5}$$

If $R \neq P^{(j)}$, then we can replace $P^{(j)}$ by the non-empty sets R and $P^{(j)} \setminus R$. Equations (3.4) and (3.5) imply that the resulting partition is refined by both P_L and $P_{L'}$ and therefore, satisfies item (1) of Definition 3.7 for $L \cap L'$. This contradicts the maximality of $|P_{L \cap L'}|$. Hence, $R = P^{(j)} \subseteq Q^{(i)}$ and P refines Q. $\square$

3.2.2 Partition-vectors

In the previous subsection we gave a description of the partition $P_{L \cap L'}$. In this and in the next subsection, we consider the problem of effectively computing $P_{L \cap L'}$, given just P_L and $P_{L'}$. In order to compute $P_{L \cap L'}$ efficiently, we will to work with vectors instead of sets.

Definition 3.19. *A* partition-vector *is a vector* $\mathbf{v} = (v_1, \ldots, v_r)$ *with* $v_i \in \{1, \ldots, i\}$, *for each* $1 \leq i \leq r$. *If* $\mathbf{v}$ *is a partition-vector then its* normalization $\mathbf{v}^\infty$ *is the partition-vector* $(v_1^\infty, \ldots, v_r^\infty)$ *defined recursively as follows*

$$v_i^\infty = \begin{cases} i, & \text{if } v_i = i, \\ v_{v_i}^\infty, & \text{if } v_i < i. \end{cases}$$

This definition gives a procedure `Normalize` with complexity $\mathcal{O}(r)$ CPU operations, which finds the normalization $\mathbf{v}^\infty$ of a partition-vector $\mathbf{v}$.

Definition 3.20. *A partition-vector* $\mathbf{v}$ *is* normalized *if* $\mathbf{v} = \mathbf{v}^\infty$. *If* P *is a partition then the* vector of P *is the normalized partition-vector* $\mathbf{v} = (v_1, \ldots, v_r)$ *given by:*

$$v_i = \min(P^{(j)}), \text{ where } P^{(j)} \text{ is the part of } P \text{ that contains } i.$$

Conversely, if $\mathbf{v}$ *is a partition-vector, then the partition* $P^{\mathbf{v}}$ *defined by* $\mathbf{v}$ *is the partition whose vector is* $\mathbf{v}^\infty$.

Example 3.21. *Let* $\mathbf{v} = (1, 1, 3, 2, 3, 6, 6, 7)$. *Then* $\mathbf{v}$ *is a partition-vector. The normalization* $\mathbf{v}^\infty$ *is given by* $\mathbf{v}^\infty = (1, 1, 3, 1, 3, 6, 6, 6)$ *and the partition* $P^{\mathbf{v}}$ *of* $\{1, \ldots, 8\}$ *is*

$$P^{\mathbf{v}} = \{\{1, 2, 4\}, \{3, 5\}, \{6, 7, 8\}\}.$$

Remark 3.22. *Let* $p, q \in \{1, \ldots, r\}$. *For simplicity's sake, if* p, q *are in the same part in a partition* P, *then we say that* p, q *are* P*-equivalent. Moreover, if* $\mathbf{v}$ *is the vector of* P, *then* p, q *are* P*-equivalent if, and only if,* $v_p^\infty = v_q^\infty$.

Definition 3.23. *Let* P, P' *be partitions of* $\{1, \ldots, r\}$ *and let* $p, q \in \{1, \ldots, r\}$. *We say that there is a* P, P'-path *from* p *to* q *if there exist* $p_0, p_1, \ldots, p_t \in \{1, \ldots, r\}$ *such that*

1. $p_0 = p$ *and* $p_t = q$.
2. p_{2n}, p_{2n+1} *are* P*-equivalent, for each* $0 \leq n \leq (t-1)/2$.

3. p_{2n+1}, p_{2n+2} *are* P'*-equivalent, for each* $0 \leq n \leq (t-2)/2$.

Notice that, if $p_0, p_1, \ldots, p_t$ is a P, P'-path, then $p_0, p_0, p_1, \ldots, p_t$ is a P', P-path. That is, there is a P, P'-path from p to q if, and only if, there is a P', P-path from p to q. The definition of P, P'-path defines an equivalence relation on $\{1, \ldots, r\}$ (paths can be concatenated and inverted). This equivalence relation defines a partition of $\{1, \ldots, r\}$, where p, q are in the same part if and only if there is a P, P'-path from p to q.

Definition 3.24. *Let* P, P' *be partitions of* $\{1, \ldots, r\}$. *The partition defined by the* P, P'*-path equivalence relation is called the* join *of* P *and* P' *and is denoted by* $P \vee P'$. *Moreover, if* $\mathbf{v}, \mathbf{w}$ *are partition-vectors, then* $\mathbf{v} \vee \mathbf{w}$ *denotes the vector of* $P^{\mathbf{v}} \vee P^{\mathbf{w}}$.

One can also see the partition $P \vee P'$ as the partition given by the transitive closure of the union of the equivalence relations defined by P and P'.

Example 3.25. *Let* $P = \{\{1,2\},\{3,4\},\{5,6\}\}$ *and* $P' = \{\{1,3\},\{2,4\},\{5\},\{6\}\}$ *be partitions. Then there is* P, P'*-path from* 1 *to* 4. *Indeed, consider* $p_0 = 1, p_1 = 2$ *and* $p_2 = 4$. *Then* p_0 *and* p_1 *are* P*-equivalent and* p_1 *and* p_2 *are* P'*-equivalent (and hence,* 1 *and* 4 *are* $P \vee P'$*-equivalent). On the other hand, there is no* P, P'*-path from* 1 *to* 5 *(and hence,* 1 *and* 5 *are not* $P \vee P'$*-equivalent). Considering all* P, P'*-paths, we have*

$$P \vee P' = \{\{1,2,3,4\},\{5,6\}\}.$$

Theorem 3.26. *The partition* $P \vee P'$ *is the finest partition that is refined by both* P *and* P'. *Hence, if* L *and* L' *are subfields of* K/k, *then* $P_{L \cap L'} = P_L \vee P_{L'}$.

Proof. Clearly, P and P' refine $P \vee P'$, since for every p, q in the same part in P (or P'), $p_0 = p, p_1 = q$ is a P, P'-path (if p, q are P'-equivalent, then $p_0 = p, p_1 = p, p_2 = q$ is a P, P'-path). Let Q be the finest partition refined by P and P'. Then Q refines $P \vee P'$. Let us suppose that $Q \neq P \vee P'$, then there exists a part A in $P \vee P'$ such

that $A = B_1 \cup B_2$, where B_1, B_2 are disjoint parts of Q. Let $p, q \in A$ such that $p \in B_1$ and $q \in B_2$. Since P and P' refine Q, there must be sets $J_1, J_1', J_2, J_2' \subseteq \{1, \ldots, r\}$ such that

$$p \in B_1 = \bigcup_{i \in J_1} P^{(i)} = \bigcup_{i \in J_1'} P'^{(i)} \tag{3.6}$$

and

$$q \in B_2 = \bigcup_{i \in J_2} P^{(i)} = \bigcup_{i \in J_2'} P'^{(i)}. \tag{3.7}$$

Moreover, since $B_1 \cap B_2 = \emptyset$, it follows that $J_1 \cap J_2 = \emptyset$ and $J_1' \cap J_2' = \emptyset$. Now, let $p_0, p_1, \ldots, p_t$ be a P, P'-path starting at p (that is, $p_0 = p$). Since $p_0 \in B_1 = \cup_{i \in J_1} P^{(i)}$ and, by definition, p_0, p_1 must be P-equivalent, it follows that also $p_1 \in B_1$. Now p_1, p_2 must be P'-equivalent. Since $p_1 \in B_1 = \cup_{i \in J_1'} P'^{(i)}$, it follows that $p_2 \in B_1$. By continuing with this argument, we see that this path is entirely contained in B_1. Since this path is arbitrary, this shows that there is no P, P'-path from p to q, which is an absurd, since for every p, q in the same part in $P \vee P'$, there must be a P, P'-path from p to q. Hence, $Q = P \vee P'$. Finally, Theorem 3.18 implies that $P_{L \cap L'} = P_L \vee P_{L'}$. □

3.2.3 The `Join` Algorithm

What we need now is an algorithm that, given two partitions P, P', returns the partition $P \vee P'$. The following algorithm does this using partition-vectors. It has input partition-vectors $\mathbf{v}, \mathbf{w}$ and the output is a partition-vector $\mathbf{u}$ with $\mathbf{u} = \mathbf{v} \vee \mathbf{w}$.

Algorithm 3.1 `Join`

Input: Partition-vectors $\mathbf{v}, \mathbf{w} \in \{1, \ldots, r\}^r$.

Output: The normalized partition-vector $\mathbf{v} \vee \mathbf{w}$.

1. Let $\mathbf{u} := \mathbf{w}$.
2. **for** $a = 1, 2, \ldots, r$ **do**
3. $\quad b := v_a$.
4. $\quad$ Compute $A := u_a^\infty$.
5. $\quad$ Compute $B := u_b^\infty$.
6. $\quad$ **if** $A < B$ **then** $u_B := A$.
7. $\quad$ **if** $B < A$ **then** $u_A := B$.
8. **return** `Normalize`$(\mathbf{u})$.

In what follows we present a proof of the correctness of the algorithm. The proof will work by induction. Let $\mathbf{u} = (u_1, \ldots, u_r)$ be a partition-vector. Define

$$\mathrm{Cut}(\mathbf{u}, s) := (u_1, \ldots, u_s, s+1, \ldots, r), \text{ for any } 1 \leq s \leq r.$$

Notice that $\mathrm{Cut}(\mathbf{u}, 1) = (1, 2, \ldots, r)$ and $\mathrm{Cut}(\mathbf{u}, r) = \mathbf{u}$. Hence, if for every $a = 1, \ldots, r$ in Step 2 (after Steps 3-7) we show that $\mathbf{u} = \mathbf{w} \vee \mathrm{Cut}(\mathbf{v}, a)$, the algorithm will follow. In Lemma 3.28 we will show what happens to the partition-vector $\mathbf{u}$ after each iteration of Step 2. Lemma 3.30 then shows the induction step.

Definition 3.27. *Let P, Q be partitions of $\{1, \ldots, r\}$. We say that P is a* simple refinement *of Q (or that Q is a* simple merge *of P) if P refines Q and $|P| = |Q| + 1$.*

Lemma 3.28. *Let $\mathbf{u}$ be a partition-vector such that $u_A^\infty = A$ and $u_B^\infty = B$, for $A, B \in \{1, \ldots, r\}$ and $A < B$. Let $\mathbf{u}'$ be such that $u'_i = u_i$, for $i \neq B$, and $u'_B = A$. Then $P^{\mathbf{u}'}$ is a simple merge of $P^{\mathbf{u}}$. Moreover, A and B are $P^{\mathbf{u}'}$-equivalent.*

Proof. Since $u'_B = A$, it follows that every element in the same part as B in $P^{\mathbf{u}}$ is now in the same part as A in $P^{\mathbf{u}'}$. Since this is the only difference between $\mathbf{u}$ and

$\mathbf{u}'$, it follows that $P^{\mathbf{u}}$ is a simple refinement of $P^{\mathbf{u}'}$ and that A and B are in the same part in $P^{\mathbf{u}'}$. □

Example 3.29. *For instance, let* $\mathbf{u} = (1,1,3,3,1,3,7)$ *be a partition-vector corresponding to the partition* $P = \{\{1,2,5\},\{3,4,6\},\{7\}\}$. *Let* $A = 1$ *and* $B = 3$. *Notice that* $u^{\infty}_A = 1 = A$ *and* $u^{\infty}_B = 3 = B$. *Let us define* $\mathbf{u}'$ *as in the previous lemma, that is,* $\mathbf{u}' = (1,1,1,3,1,3,7)$. *This partition-vector corresponds to the partition* $P' = \{\{1,2,3,4,5,6\},\{7\}\}$, *which is a simple merge of* P. *Moreover,* $A = 1$ *and* $B = 3$ *are in the same part in* P'.

Lemma 3.30. *Let* $\mathbf{v}, \mathbf{w}$ *be partition-vectors and let* $\mathbf{u}$ *be such that* $\mathbf{u} = \mathbf{w} \vee \mathit{Cut}(\mathbf{v}, s)$. *Let* $\mathbf{u}'$ *be such that*

(1) *If* w_{s+1}, v_{s+1} *are* $P^{\mathbf{u}}$*-equivalent, then* $\mathbf{u}' = \mathbf{u}$;

(2) *Otherwise, define* $\mathbf{u}'$ *such that* $P^{\mathbf{u}'}$ *is a simple merge of* $P^{\mathbf{u}}$ *and* w_{s+1}, v_{s+1} *are* $P^{\mathbf{u}'}$*-equivalent.*

Then $\mathbf{u}' = \mathbf{w} \vee \mathit{Cut}(\mathbf{v}, s+1)$.

Proof. For simplicity, let us denote $\mathrm{Cut}(\mathbf{v}, s)$ simply by $\mathbf{v}_{|s}$. We need to show that $\mathbf{u}' = \mathbf{w} \vee \mathbf{v}_{|s+1}$ or, equivalently, that $P^{\mathbf{u}'} = P^{\mathbf{w}} \vee P^{\mathbf{v}_{|s+1}}$. That is, for $p, q \in \{1, \ldots, r\}$, we need to show that p, q are $P^{\mathbf{u}'}$-equivalent if and only if, there is a $P^{\mathbf{w}}, P^{\mathbf{v}_{|s+1}}$-path from p to q.

Let p, q be $P^{\mathbf{u}'}$-equivalent. If p, q are also $P^{\mathbf{u}}$-equivalent, then (since $\mathbf{u} = \mathbf{w} \vee \mathbf{v}_{|s}$) there exists a $P^{\mathbf{w}}, P^{\mathbf{v}_{|s}}$-path from p to q, and hence, there exists a $P^{\mathbf{w}}, P^{\mathbf{v}_{|s+1}}$-path from p to q.[1] Now suppose that p, q are not $P^{\mathbf{u}}$-equivalent. Since p, q are $P^{\mathbf{u}'}$-equivalent, we can assume, w.l.o.g., that both p, w_{s+1} and v_{s+1}, q are $P^{\mathbf{u}}$-equivalent (and hence, there are $P^{\mathbf{w}}, P^{\mathbf{v}_{|s+1}}$-paths from p to w_{s+1} and from v_{s+1} to q). On

[1] Every $P^{\mathbf{w}}, P^{\mathbf{v}_{|s}}$-path is also a $P^{\mathbf{w}}, P^{\mathbf{v}_{|s+1}}$-path.

the other hand, $w_{s+1}, s+1, v_{s+1}$ is also a $P^{\mathbf{w}}, P^{\mathbf{v}_{|s+1}}$-path. Since paths can be concatenated, this yields a $P^{\mathbf{w}}, P^{\mathbf{v}_{|s+1}}$-path from p to q.

Conversely, suppose that $p_0, \ldots, p_t$ is a $P^{\mathbf{w}}, P^{\mathbf{v}_{|s+1}}$-path, with $p_0 = p$ and $p_t = q$. We need to show that p, q are $P^{\mathbf{u}'}$-equivalent. If $p_i \neq s+1$, for every $i = 0, \ldots, t$, then $p_0, \ldots, p_t$ is also a $P^{\mathbf{w}}, P^{\mathbf{v}_{|s}}$-path and since $\mathbf{u} = \mathbf{w} \vee \mathbf{v}_{|s}$, it follows that p_0, p_t are $P^{\mathbf{u}}$-equivalent (and thus, $P^{\mathbf{u}'}$-equivalent, since $P^{\mathbf{u}'}$ is a simple merge of $P^{\mathbf{u}}$). Now suppose that $p_i = s+1$, for some i (we may assume i is unique). Then, for the same reason as above, $p_0, \ldots, p_{i-1}$ and $p_{i+1}, \ldots, p_t$ are $P^{\mathbf{w}}, P^{\mathbf{v}_{|s}}$-paths. Thus, p_0, p_{i-1} and p_{i+1}, p_t are $P^{\mathbf{u}}$-equivalent (and thus, $P^{\mathbf{u}'}$-equivalent). W.l.o.g., assume that $p_{i-1}, s+1$ are $P^{\mathbf{w}}$-equivalent and $s+1, p_{i+1}$ are $P^{\mathbf{v}_{|s+1}}$-equivalent. In $P^{\mathbf{w}}$, $s+1$ is in the same part as w_{s+1} and hence, p_{i-1}, w_{s+1} are $P^{\mathbf{w}}$-equivalent (and thus, $P^{\mathbf{u}'}$-equivalent). Likewise, v_{s+1}, p_{i+1} are $P^{\mathbf{v}_{|s+1}}$-equivalent. If $v_{s+1} = s+1$, then $\mathbf{v}_{|s} = \mathbf{v}_{|s+1}$ and hence, v_{s+1}, p_{i+1} are $P^{\mathbf{v}_{|s}}$-equivalent. If $v_{s+1} < s+1$, then again, v_{s+1}, p_{i+1} are $P^{\mathbf{v}_{|s}}$-equivalent. Either way, v_{s+1}, p_{i+1} are $P^{\mathbf{u}'}$-equivalent. Finally, by construction, we have that v_{s+1}, w_{s+1} are $P^{\mathbf{u}'}$-equivalent. □

Theorem 3.31. *Given two partition-vectors* $\mathbf{v}$ *and* $\mathbf{w}$*, Algorithm* `Join` *returns a partition-vector* $\mathbf{u}$ *such that* $\mathbf{u} = \mathbf{v} \vee \mathbf{w}$*. Moreover, we can compute the join of two partitions with* $\tilde{\mathcal{O}}(r^{3/2})$ *CPU operations.*

Proof. We will prove that for every $a = 1, \ldots, r$ in step 2, after computing steps 3-8, the partition-vector $\mathbf{u}$ becomes $\mathbf{w} \vee \mathrm{Cut}(\mathbf{v}, a)$. Hence, at the end of the algorithm, $\mathbf{u} = \mathbf{w} \vee \mathrm{Cut}(\mathbf{v}, r) = \mathbf{w} \vee \mathbf{v}$.

At Step 1, $\mathbf{u} = \mathbf{w} = \mathbf{w} \vee \mathrm{Cut}(\mathbf{v}, 1)$. Suppose that after computing steps 3-8 for some $a = s$ in Step 2, we have that $\mathbf{u} = \mathbf{w} \vee \mathrm{Cut}(\mathbf{v}, s)$. Let $\mathbf{u}'$ be the partition computed after steps 3-8 for $a = s+1$. We want to prove that $\mathbf{u}' = \mathbf{w} \vee \mathrm{Cut}(\mathbf{v}, s+1)$. According to Lemma 3.30, we need to show that $\mathbf{u}'$ satisfies conditions (1) or (2) of Lemma 3.30. Suppose that w_{s+1}, v_{s+1} are $P^{\mathbf{u}}$-equivalent.

This means that $u^{\infty}_{w_{s+1}} = u^{\infty}_{v_{s+1}}$. Since $\mathbf{u} = \mathbf{w}$ at Step 1, it follows that $u^{\infty}_{w_{s+1}} = u^{\infty}_{s+1}$. Hence,

$$A := u^{\infty}_{s+1} = u^{\infty}_{w_{s+1}} = u^{\infty}_{v_{s+1}} =: B.$$

The algorithm sets $\mathbf{u}' := \mathbf{u}$ and hence, by Lemma 3.30, $\mathbf{u}' = \mathbf{w} \vee \mathrm{Cut}(\mathbf{v}, s+1)$. On the other hand, suppose that w_{s+1}, v_{s+1} are not $P^{\mathbf{u}}$-equivalent. This means that we have either $A < B$ or $B < A$ in steps 5-6 for $a = s+1$. In either case, Lemma 3.28 tells us that the partition $P^{\mathbf{u}'}$ is a simple merge of $P^{\mathbf{u}}$ and that w_{s+1}, v_{s+1} are $P^{\mathbf{u}'}$-equivalent. Again, by Lemma 3.30, $\mathbf{u}' = \mathbf{w} \vee \mathrm{Cut}(\mathbf{v}, s+1)$.

For each $1 \leq a \leq r$ in step 2, denote by l_a the length of the loop in steps 4 and 5 of the algorithm. If we normalize $\mathbf{u}$ every $\lceil \sqrt{r} \rceil$ iterations of the loop in step 2, then it follows that $l_a \in \mathcal{O}(r^{1/2})$. The total number of normalizations is $\mathcal{O}(r^{1/2})$, and the cost of each normalization is $\mathcal{O}(r)$. Therefore, the total cost of the algorithm is the cost of $\mathcal{O}(r^{1/2})$ normalizations plus r times the cost of finding u^{∞}_a (and u^{∞}_b), which is given by $\mathcal{O}(r^{1/2})$. Hence, we can compute $\mathbf{v} \vee \mathbf{w}$ with at most $\mathcal{O}(r^{3/2})$ CPU operations. □

In general, if the *depth* of $\mathbf{u}$ (*i.e.*, the length of the loops in Steps 4 and 5) is bounded by d, then the complexity for computing the join of two partitions using Algorithm `Join` is $\tilde{\mathcal{O}}(rd)$.

A similar algorithm is presented by Freese [19] (see also [20]), which we found *after* we devised and proved algorithm `Join` above. However, [19] uses a different "representation vector" for a partition. This (and other clever tricks) allows the author to show that the join of two partitions can be computed with $\mathcal{O}(r \log r)$ CPU operations. We have implemented Freese's algorithm, but it performed slightly worse than the simpler algorithm `Join` given above. For this reason, we decided to keep this section. Moreover, we shall use the Algorithm `Join` given above in our implementations. To estimate the complexity, we use the complexity stated in [19].

Theorem 3.32. *Given two partition-vectors* $\mathbf{v}$ *and* $\mathbf{w}$*, there exists an algorithm that returns a partition-vector* $\mathbf{u}$ *such that* $\mathbf{u} = \mathbf{v} \vee \mathbf{w}$*. Moreover, the partition-vector* $\mathbf{u}$ *can be computed with* $\tilde{\mathcal{O}}(r)$ *CPU operations.*

Proof. See [19] (see also [20]). □

3.3 Computing the Partition of a Principal Subfield

Let $f_1, \ldots, f_r$ be a subfield factorization of f. In general, one can compute a subfield factorization of f by factoring f over K. For $k = \mathbb{Q}$ we will give an alternative in Chapter 4. We already know how to intersect partitions to find the partition of the intersection of subfields. However, it remains to find the partition of the principal subfields. We already mentioned how one can compute this partition in Example 3.10 by, basically, combining the factors of the subfield factorization and checking which combination has coefficients in the subfield. In this section we present a polynomial time algorithm to compute the partition P_i of $\{1, \ldots, r\}$ defined by a principal subfield L_i of K/k. First of all, notice that, in order to find P_i, it suffices to find a basis of the vectors $(e_1, \ldots, e_r) \in \{0, 1\}^r$ for which

$$\prod_{j=1}^{r} f_j^{e_j} \in L_i[x]. \tag{3.8}$$

The numbers $e_1, \ldots, e_r$ appear as exponents in Equation (3.8). A way to linearize this problem is to use the logarithmic derivative h_j of f_j (this technique has been used in several other algorithms, mainly for polynomial factorization, such as [38] and [21]). That is, let $h_j = f_j'/f_j \in K(x)$ and let $H(x) = \sum_{j=1}^{r} e_j h_j$. If $g = \prod_{j=1}^{r} f_j^{e_j}$, then $g'/g = H$. We now have to find sufficient conditions for H such that if H satisfies these conditions for a certain $(e_1, \ldots, e_r) \in \{0, 1\}^r$, then Equation (3.8) holds for the same vector $(e_1, \ldots, e_r) \in \{0, 1\}^r$.

Definition 3.33. *Let $f \in k[x]$. Then f is* semi-separable *if $char(k) = 0$ or $char(k) = p$ and f has no roots with multiplicity larger than or equal to p.*

Lemma 3.34. *Let $g \in K[x]$ monic and semi-separable, and let L be a subfield of K/k. If $g'/g \in L(x)$, then $g \in L[x]$.*

Proof. Consider the groups $(K(x)^*, \cdot)$ and $(K(x), +)$ and let $\phi : K(x)^* \to K(x)$ be the group homomorphism defined by $\phi(g) = g'/g$. The kernel of ϕ is K^* in characteristic 0 and $K(x^p)^*$ in characteristic p. So, if we restrict ϕ to monic semi-separable polynomials, then ϕ becomes injective. Let $g \in K[x]$ be a monic semi-separable polynomial such that $g'/g \in L(x)$. Let $\overline{g} \in \bar{L}[x] = \bar{K}[x]$ be a conjugate of g over L. Since $g'/g \in L(x)$, it follows that

$$\phi(\overline{g}) = \overline{g}'/\overline{g} = \overline{(g'/g)} = g'/g = \phi(g),$$

By the injectivity of ϕ on monic semi-separable polynomials, $\overline{g} = g$ for any conjugate of g over L in $\overline{K}[x]$. Therefore, $g \in L[x]$ (recall that K/k and hence K/L are assumed to be separable extensions). □

Lemma 3.35. *Let $g \in K[x]$ monic, $deg(g) = n$, and let L be a subfield of K. Let $p_1, \ldots, p_{2n} \in k$ be distinct elements. If $g'(p_i)/g(p_i) \in L$, $1 \leq i \leq 2n$, then $g'/g \in L(x)$.*

Proof. Let $h = g'/g \in K(x)$ and suppose that $h(p_i) \in L$, $1 \leq i \leq 2n$. Let $\overline{h} = \overline{g}'/\overline{g}$ be a conjugate of h over L. Then

$$h(p_i) = \overline{h(p_i)} = \overline{h}(\overline{p_i}) = \overline{h}(p_i), \;\; 1 \leq i \leq 2n.$$

This means that the polynomial $g'\overline{g} - \overline{g}'g$ of degree $< 2n$ has $2n$ distinct roots. Hence, $g'\overline{g} - \overline{g}'g = 0$ and therefore $h = \overline{h}$, for every conjugate $\overline{h}$ of h over L. That is, $h \in L(x)$. □

The idea is to use Lemmas 3.34 and 3.35 to find the desired vectors $(e_1, \ldots, e_r)$ by solving a linear system on $e_1, \ldots, e_r$. Consider the following subroutine `System`.

Algorithm 3.2 Subroutine `System`.

Input: Subfield factorization $f_1, \ldots, f_r$, an index i and indeterminates $e_1, \ldots, e_r$.

Output: Set of equations $\mathcal{S}_i$ on $e_1, \ldots, e_r$, whose solutions give the partition P_i.

1. Choose distinct elements $p_1, \ldots, p_{2n}$ of k.
2. Let $q_j(\alpha) := \sum e_l \frac{f_l'(p_j)}{f_l(p_j)}$, where $q_j(x) \in e_1 \cdot k[x] + \cdots + e_r \cdot k[x]$, $j = 1, \ldots, 2n$.
3. Let $\mathcal{S}_i$ be the system of k-linear equations obtained by taking the coefficients of x and α of $q_j(x) \bmod f_i = q_j(\alpha)$, for $j = 1, \ldots, 2n$,.
4. **return** $\mathcal{S}_i$.

In Step 3 of the algorithm, $q_j(x) \bmod f_i$ represents the remainder of the division of $q_j(x)$ by f_i over K. Notice that the field k should contain at least $2n$ elements for the algorithm to work. However, we should not worry about this, as we will present a probabilistic version of this algorithm which requires much less than $2n$ elements. If $(e_1, \ldots, e_r)$ is a solution of the system $\mathcal{S}_i$ given by Subroutine `System`, then by Lemmas 3.34 and 3.35, it follows that $\prod f_j^{e_j} \in L_i[x]$. By construction, $\mathcal{S}_i$ has a basis of solutions in $\{0,1\}$-*echelon form*:

Definition 3.36. *A basis of solutions $\{s_1, \ldots, s_t\}$ of $\mathcal{S}_i$ is called a* $\{0,1\}$-echelon basis of $\mathcal{S}_i$ *if*

1. $s_i = (s_{i,1}, \ldots, s_{i,r}) \in \{0,1\}^r \subset \mathbb{Z}^r$, $1 \leq i \leq t$.
2. $\sum_{i=1}^{t} s_i = (1, \ldots, 1)$.

Remark 3.37. *If a $\{0,1\}$-echelon basis of $\mathcal{S}_i$ exists, then any reduced echelon basis of $\mathcal{S}_i$ is automatically a $\{0,1\}$-echelon basis due to the uniqueness of the reduced echelon basis.*

Corollary 3.38. *Let* $\{s_1, \ldots, s_t\}$ *be a* $\{0,1\}$*-echelon basis of* $\mathcal{S}_i$ *and define* $P_i = \{P^{(1)}, \ldots, P^{(t)}\}$*, where* $P^{(l)} = \{j : s_{l,j} = 1\}$*. Then* P_i *is the partition defined by* L_i*.*

Proof. If $(e_1, \ldots, e_r) \in \{0,1\}^r$ is a solution of $\mathcal{S}_i$ then, by Lemmas 3.34 and 3.35, it follows that $g = \prod_{j=1}^{r} f_j^{e_j} \in L_i[x]$. Thus, the P_i-products are in $L_i[x]$. The maximality of $|P_i|$ follows from the fact that $s_1, \ldots, s_t$ form a basis for the solution space of $\mathcal{E}$ and that any vector $(e_1, \ldots, e_r) \in \{0,1\}^r$ such that $\prod f_i^{e_i} \in L_i[x]$ is a solution of $\mathcal{E}$. Hence, the partition P_i is the partition defined by L_i. □

Therefore, the partition P_i of $\{1, \ldots, r\}$ corresponding to the subfield L_i can be found using the following algorithm.

Algorithm 3.3 `Partition` (Slow version).

Input: Subfield factorization $f_1, \ldots, f_r$ of f and an index i.

Output: The partition P_i of $\{1, \ldots, r\}$ defined by L_i.

1. Compute $\mathcal{S}_i := \texttt{System}(\{f_1, \ldots, f_r\}, i)$.
2. Compute a $\{0,1\}$-echelon basis $\{s_1, \ldots, s_t\}$ of $\mathcal{S}_i$.
3. **return** $P_i := \{P^{(1)}, \ldots, P^{(t)}\}$, where $P^{(l)}$ is as in Corollary 3.38.

This algorithm, however, does not perform very well in practice. Apart from the (costly) $2n$ polynomial divisions over K in Step 3 of `System`, the system $\mathcal{S}_i$ is over-determined. The number of linear equations in $\mathcal{S}_i$ is bounded by $2n^2 d_i$, where $d_i = \deg(f_i)$, while the number of variables is $r \leq n$. Furthermore, the coefficients are in k and can be potentially large, while the solutions are 0-1 vectors (that could have been recovered from its images modulo a prime number). We address these problems by computing a subset of $\mathcal{S}_i$ modulo a prime ideal $\mathfrak{p}$.

Definition 3.39. *A* good k-valuation w.r.t. f *is a valuation* $v : k \to \mathbb{Z} \cup \{\infty\}$ *such that if* $R_v = \{a \in k \ : \ v(a) \geq 0\}$ *and* $p_v = \{a \in k \ : \ v(a) > 0\}$*, then* $f \in R_v[x]$*, the residue field* $\boldsymbol{F} := R_v/p_v$ *is finite, the image* $\bar{f}$ *of* f *in* $\boldsymbol{F}[x]$ *is separable and*

$\deg(\bar{f}) = \deg(f)$. *Furthermore, we call an ideal* $\mathfrak{p}$ *a* good k-ideal *if* $\mathfrak{p} = p_v$, *for some good* k*-valuation* v.

If $k = \mathbb{Q}$, then a good k-ideal $\mathfrak{p}$ is of the form $\langle p \rangle$, for some prime number p such that $f \bmod p$ is separable and has the same degree as f. The following subroutine returns $\tilde{\mathcal{S}}_{i,c}$: a subset of $\mathcal{S}_i$ modulo a good k-ideal $\mathfrak{p}$.

Algorithm 3.4 Subroutine `SystemModP`.

Input: Subfield factorization $f_1, \ldots, f_r$, an index i and a good k-ideal $\mathfrak{p}$.

Output: $\tilde{\mathcal{S}}_{i,c}$: necessary equations modulo $\mathfrak{p}$ for $e_1, \ldots, e_r$.

1. Choose $c \in \mathbf{F}$ at random.
2. If $f_j(c) \bmod \mathfrak{p}$ has no inverse, for some $1 \leq j \leq r$, go to Step 1.
3. Let $q(\alpha) := \sum e_j f_j'(c)/f_j(c) \bmod \mathfrak{p}$, where $q(x) \in e_1 \cdot \mathbf{F}[x]_{<n} + \cdots + e_r \cdot \mathbf{F}[x]_{<n}$.
4. Let $\tilde{\mathcal{S}}_{i,c}$ be the system of $\mathbf{F}$-linear equations obtained by taking the coefficients of x and α of $q(x) \bmod f_i = q(\alpha)$.
5. **return** $\tilde{\mathcal{S}}_{i,c}$.

The element $f_j(c) \bmod \mathfrak{p}$ from Step 2 is in the finite ring $\mathbf{F}[\alpha]$, where $\mathbf{F}$ is as in Definition 3.39, for a good k-valuation v such that $\mathfrak{p} = p_v$, and hence, may not have an inverse. In this algorithm, we also need $\mathbf{F}$ to have sufficiently many elements for step 2. If this is not the case, we can compute a finite extension $\tilde{\mathbf{F}}$ of $\mathbf{F}$ and compute/solve the system $\tilde{\mathcal{S}}_{i,c}$ over this extension.

The system $\tilde{\mathcal{S}}_{i,c}$ is a subset of $\mathcal{S}_i$ reduced modulo a prime ideal $\mathfrak{p}$. Therefore, a basis of solutions for $\tilde{\mathcal{S}}_{i,c}$ may not represent the partition P_i. In fact, $\tilde{\mathcal{S}}_{i,c}$ may not even have a $\{0,1\}$-echelon basis. This means that we have to devise a test that verifies that the solution basis of $\tilde{\mathcal{S}}_{i,c}$ represents the partition P_i. This test is given in Theorem 3.43 below.

If we find out that the solution basis for $\tilde{\mathcal{S}}_{i,c}$ does not represent the partition P_i (for instance, it contains elements other than 0's and 1's), then we choose $c' \neq c$, compute $\tilde{\mathcal{S}}_{i,c'}$ and solve $\tilde{\mathcal{S}}_i := \tilde{\mathcal{S}}_{i,c} \cup \tilde{\mathcal{S}}_{i,c'}$. We continue this process until we find the correct partition P_i.

Example 3.40. *Let $f = x^8 - 5 \in \mathbb{Q}[x]$ and let α be a root of f. As we have seen, the factorization of f over $\mathbb{Q}(\alpha)$ is given by $f = (x-\alpha)(x+\alpha)(x^2+\alpha^2)(x^4+\alpha^4)$, whose factors we call f_1, f_2, f_3 and f_4, respectively. Let us compute the partition P_3, corresponding to the subfield $L_3 = \mathbb{Q}(\alpha^4)$ using the above method. First of all, we need an appropriate prime p. For $p = 3$, one can check that $f \bmod p$ is separable and that $\deg(f) = \deg(f \bmod p)$. Next, let us choose a random element in $\boldsymbol{F} = 3\mathbb{Z}$. For instance, $c = 2$. We now need to verify that $f_j(c) \bmod 3$ has an inverse in $3\mathbb{Z}[\alpha]$, for $j = 1, \ldots, 4$. These elements are*

$$\begin{aligned} f_1(2) \bmod 3 &= 2 - \alpha = 2 + 2\alpha, \\ f_2(2) \bmod 3 &= 2 + \alpha, \\ f_3(2) \bmod 3 &= 2^2 + \alpha^2 = 1 + \alpha^2, \\ f_4(2) \bmod 3 &= 2^4 + \alpha^4 = 1 + \alpha^4. \end{aligned}$$

To compute the inverse of $f_1(c) \bmod 3$, for instance, we use the Extended Euclidean Algorithm. Consider $g = 2x + 2$. Since $\deg(g) < \deg(f)$ and f is irreducible, it follows that $\gcd(f, g) = 1$. Hence, there exists $a, b \in \mathbb{Z}[x]$ such that $1 = af + bg$. By evaluating this equation at $x = \alpha$, we get $1 = a(\alpha)f(\alpha) + b(\alpha)g(\alpha) = 0 + b(\alpha)g(\alpha)$. Hence, $b(\alpha)g(\alpha) = 1$ and $b(\alpha)$ is the inverse of $g(\alpha)$ (provided $b(\alpha)$ is not 0 in $3\mathbb{Z}[\alpha]$). In this case, the inverses are

$$\begin{aligned} 1/f_1(2) \bmod 3 &= 2\alpha^7 + \alpha^6 + 2\alpha^5 + \alpha^4 + 2\alpha^3 + \alpha^2 + 2\alpha + 1, \\ 1/f_2(2) \bmod 3 &= \alpha^7 + \alpha^6 + \alpha^5 + \alpha^4 + \alpha^3 + \alpha^2 + \alpha + 1, \\ 1/f_3(2) \bmod 3 &= \alpha^6 + 2\alpha^4 + \alpha^2 + 2, \\ 1/f_4(2) \bmod 3 &= \alpha^4 + 2. \end{aligned}$$

This allows us to compute the elements

$$f_1'(2)/f_1(2) \bmod 3 = 2\alpha^7 + \alpha^6 + 2\alpha^5 + \alpha^4 + 2\alpha^3 + \alpha^2 + 2\alpha + 1 =: A(\alpha),$$
$$f_2'(2)/f_2(2) \bmod 3 = \alpha^7 + \alpha^6 + \alpha^5 + \alpha^4 + \alpha^3 + \alpha^2 + \alpha + 1 =: B(\alpha),$$
$$f_3'(2)/f_3(2) \bmod 3 = \alpha^6 + 2\alpha^4 + \alpha^2 + 2 =: C(\alpha),$$
$$f_4'(2)/f_4(2) \bmod 3 = 2\alpha^4 + 1 =: D(\alpha).$$

Let us now define $q(\alpha)$. This element is given by

$$q(\alpha) := \sum e_i f_i'(2)/f_i(2) = A(\alpha)e_1 + B(\alpha)e_2 + C(\alpha)e_3 + D(\alpha)e_4,$$

and hence, $q(x) \bmod f_3$ is given by

$$q(x) \bmod f_3 = (A(x) \bmod f_3)e_1 + (B(x) \bmod f_3)e_2 + (C(x) \bmod f_3)e_3 + (D(x) \bmod f_3)e_4.$$

By carrying out these divisions, we have

$$\begin{aligned} q(x) \bmod f_3 = &\left((\alpha^6 + 2\alpha^4 + \alpha^2 + 2)x + (2\alpha^6 + \alpha^4 + 2\alpha^2 + 1)\right) e_1 + \\ &+ \left((2\alpha^6 + \alpha^4 + 2\alpha^2 + 1)x + (2\alpha^6 + \alpha^4 + 2\alpha^2 + 1)\right) e_2 + \\ &+ (2\alpha^6 + 2\alpha^4 + 2\alpha^2 + 2)e_3 + (\alpha^4 + 2)e_4. \end{aligned}$$

Hence, by looking at the coefficients of x of $q(x) \bmod f_3 = q(\alpha)$, we have the following system

$$\begin{cases} (\alpha^6 + 2\alpha^4 + \alpha^2 + 2)e_1 + (2\alpha^6 + \alpha^4 + 2\alpha^2 + 1)e_2 = 0 \\ (2\alpha^6 + \alpha^4 + 2\alpha^2 + 1)e_1 + (2\alpha^6 + \alpha^4 + 2\alpha^2 + 1)e_2 + \\ \quad + (2\alpha^6 + 2\alpha^4 + 2\alpha^2 + 2)e_3 + (2\alpha^4 + 1)e_4 = q(\alpha) \end{cases}$$

Or, by looking at the coefficients of α, we have the system

$$\tilde{\mathcal{S}}_{3,2} = \begin{cases} e_1 + 2e_2 = 0 \\ e_1 + e_2 + e_3 = 0 \end{cases}$$

By solving this system over $3\mathbb{Z}$, we obtain the basis $\{(1,1,1,0),(0,0,0,1)\}$. The next step is to verify that this basis of solutions is a $\{0,1\}$-echelon basis and, if so,

that the corresponding partition is indeed P_3. The first part is an easy check. The second part is shown in Theorem 3.43 below. If one of these checks fail, then we need to choose a different $c' \in 3\mathbb{Z}$, construct the system $\tilde{\mathcal{S}}_{3,c'}$ and solve $\tilde{\mathcal{S}}_{3,c} \cup \tilde{\mathcal{S}}_{3,c'}$. In this case, $\{(1,1,1,0),(0,0,0,1)\}$ is a $\{0,1\}$-echelon basis and it corresponds to the partition $\tilde{P}_3 := \{\{1,2,3\},\{4\}\}$. As mentioned above, it remains to show $\tilde{P}_3 = P_3$.

Now we need to devise a test that proves that a given partition corresponds to a certain principal subfield. That is, let $f_1, \ldots, f_r$ be a subfield factorization and let L_i be the principal subfield corresponding to f_i and P_i be the corresponding partition of $\{1, \ldots, r\}$ (which we wish to find). Given a partition P, we wish to show that $P = P_i$. In order to do so, we will need the following lemma,

Lemma 3.41. *Let K be a field and $f \in K[x]$ monic separable such that $f = g_1 \cdots g_t = h_1 \cdots h_t$, where $g_j, h_j \in \mathcal{O}_K[x]$ are monic but not necessarily irreducible. Let $\mathfrak{q} \subseteq \mathcal{O}_K$ be an ideal such that $f \bmod \mathfrak{q}$ is separable. If $g_j \equiv h_j \bmod \mathfrak{q}$, for every $1 \leq j \leq t$, then $g_j = h_j$, $1 \leq j \leq t$.*

Proof. It suffices to show that for every irreducible factor q of f in $K[x]$, $q \mid g_j$ if and only if, $q \mid h_j$. Suppose that $q \mid g_j$. Then $q \nmid g_l$, for any $l \neq j$, because f is separable. Moreover, q also does not divide $g_l \bmod \mathfrak{q}$, $l \neq j$, because f is separable modulo $\mathfrak{q}$. Since $g_l \equiv h_l \bmod \mathfrak{q}$, it follows that $q \nmid h_l \bmod \mathfrak{q}$ and hence, $q \nmid h_l$ over K, for all $l \neq j$. But q divides $f = h_1 \cdots h_t$ and since $K[x]$ is a unique factorization domain, it follows that $q \mid h_j$. The converse follows similarly. Hence $q \mid g_j$ if and only if, $q \mid h_j$. Since this holds for any irreducible factor q of f in $K[x]$ and g_j, h_j are monic, the equality follows. □

Remark 3.42. *When choosing the ideal $\mathfrak{q}$ we have to make sure that denominators of coefficients of g_j and h_j are not elements of $\mathfrak{q}$, otherwise the equation $g_j \equiv h_j \bmod \mathfrak{q}$ would return an error message. For $k = \mathbb{Q}$, and assuming f monic, we assert that it is enough to choose $\mathfrak{q}$ such that $disc(f) \not\equiv 0 \bmod \mathfrak{q}$. This follows from the following*

inclusions

$$\mathbb{Z}[\alpha] \subseteq \mathcal{O}_K \subseteq \frac{1}{f'(\alpha)} \cdot \mathbb{Z}[\alpha] \subseteq \frac{1}{disc(f)} \cdot \mathbb{Z}[\alpha],$$

where $\mathcal{O}_K$ is the ring of integers of K and $disc(f)$ is the discriminant of f, and the fact that any factor of f over K is in $\mathcal{O}_K[x]$.

Let $f_1, \ldots, f_r$ be a subfield factorization of f and let L_i be the principal subfield corresponding to f_i. Let $K_i := K[y]/\langle f_i(y)\rangle$ and define

$$\sigma_i : K \to K_i, \;\; h(\alpha) \mapsto h(y) \bmod f_i(y).$$

By the definition of σ_i and since f is separable, we can rewrite the definition of the subfield L_i in the following way (set $g = f_i$ in Equation 3.1)

$$L_i = L_{f_i} = \{h(\alpha) : h \in k[x]_{<n},\ \sigma_i(h(\alpha)) = h(\alpha)\}.$$

Theorem 3.43. *Let $f_1, \ldots, f_r$ be a subfield factorization of f. Let L_i be the principal subfield corresponding to f_i and let P_i be the corresponding partition. Let $\mathfrak{q} \subseteq \mathcal{O}_{K_i}$ be an ideal such that $f \bmod \mathfrak{q}$ is separable. Moreover, let P be a partition of $\{1, \ldots, r\}$ such that P refines P_i. If*

$$\sigma_i(g_j) \equiv g_j \bmod \mathfrak{q}, \tag{3.9}$$

for $j = 1, \ldots, t$, where $g_1, \ldots, g_t$ are the P-products and σ_i acts on g_j coefficient-wise, then $P = P_i$.

Proof. Since P refines P_i, it suffices to prove that the P-products $g_1, \ldots, g_t$ are elements of $L_i[x]$ (the maximality of $|P|$ will follow from the fact that P refines P_i). By the definition of σ_i, $g_j \in L_i[x]$, if and only if, $\sigma_i(g_j) = g_j$. Since

$$g_1 \cdots g_t = f = \sigma_i(f) = \sigma_i(g_1) \cdots \sigma_i(g_t)$$

over K_i, $f \bmod \mathfrak{q}$ is separable and $\sigma(g_j) \equiv g_j \bmod \mathfrak{q}$, it follows by Lemma 3.41 that $\sigma(g_j) = g_j$, that is, $g_j \in L_i[x]$ and hence, $P = P_i$. If K_i is not a field, we

cannot directly apply Lemma 3.41. Let $f_i = f_{i_1} \cdots f_{i_s}$, with $f_{i_m} \in K[x]$ irreducible, $m = 1, \ldots, s$. Let $K_{i_m} := K[y]/\langle f_{i_m}(y) \rangle$ and define $\sigma_{i_m} : K \to K_{i_m}$ as above. Since f is separable, it follows that $\tilde{g}_j \in L_i[x]$ if and only if, $\sigma_{i_m}(\tilde{g}_j) = \tilde{g}_j$, $m = 1, \ldots, s$. To use Lemma 3.41, we would need $\mathfrak{q}$ to be a good K_{i_m}-ideal. However, we can view $K_{i_m} = K[\alpha_{i_m}]$, where α_{i_m} is a root of f_{i_m}, and choose $\mathfrak{q}$ to be a good K-ideal. Thus, by Lemma 3.41 (with σ_{i_m} instead of σ_i in the argument above and $\mathfrak{q}$ a good K-ideal), it follows that $\sigma_{i_m}(\tilde{g}_j) = \tilde{g}_j$, if and only if, $\sigma_{i_m}(\tilde{g}_j) \equiv \tilde{g}_j \bmod \mathfrak{q}$, $m = 1, \ldots, s$. Since $f \bmod \mathfrak{q}$ is separable, this is equivalent to $\sigma_i(\tilde{g}_j) \equiv \tilde{g}_j \bmod \mathfrak{q}$. That is, if the $\tilde{P}_i$-products satisfy Equation (3.9), then $\tilde{P}_i$ is the partition of L_i. □

The partition P_i of L_i can be computed with the following algorithm.

Algorithm 3.5 `Partition` (Fast version)

Input: Subfield factorization $f_1, \ldots, f_r$, an index i and a good k-ideal $\mathfrak{p}$.

Output: The partition P_i of $\{1, \ldots, r\}$ defined by L_{f_i}.

1. Compute $\tilde{\mathcal{S}}_i$ using subroutine `SystemModP`.
2. Compute a $\{0,1\}$-echelon basis $\{s_1, \ldots, s_t\}$ of $\tilde{\mathcal{S}}_i$ (see Remark 3.37).
3. **if** Step 2 fails **then**
4. Compute more equations with `SystemModP`.
5. Go to Step 2.
6. Let $\tilde{P}_i := \{P^{(1)}, \ldots, P^{(t)}\}$, where $P^{(l)}$ is as in Corollary 3.38.
7. Let $\hat{g}_1, \ldots, \hat{g}_t$ be the $\tilde{P}_i$-products. //
8. Let $\mathfrak{q}$ be an ideal as in Lemma 3.41. //
9. **for** $j = 1, \ldots, t$ **do** // Correctness check (Theorem 3.43).
10. **if** $\sigma_i(\tilde{g}_j) \not\equiv \tilde{g}_j \bmod \mathfrak{q}$ **then** //
11. Go to Step 4. //
12. **return** $\tilde{P}_i$.

We were not able to bound the number of calls to `EquationsModP` when computing the partition P_i. However, based on our experiments for $k = \mathbb{Q}$, the average number of calls to `EquationsModP` appears to be bounded by a constant (in fact, this number never exceeded 3 in our examples). For this reason, we shall assume that the number of calls to `EquationsModP` is $\mathcal{O}(1)$.

Example 3.44. *Let $f = x^8 - 5 \in \mathbb{Q}[x]$ and let $f = (x-\alpha)(x+\alpha)(x^2+\alpha^2)(x^4+\alpha^4)$ be its factorization over $\mathbb{Q}(\alpha)$, where α is a root of f. In Example 3.40 we have found a candidate $\tilde{P}_3 = \{\{1,2,3\},\{4\}\}$ for the partition of the subfield L_3. According to Theorem 3.43, we need to show that the $\tilde{P}_3$-products $\tilde{g}_1 = f_1 f_2 f_3 = x^4 - \alpha^4$ and $\tilde{g}_2 = f_4 = x^4 + \alpha^4$ satisfy Equation (3.9), for some ideal $\mathfrak{q}$ of $\mathcal{O}_{\mathbb{Q}(\alpha)}$ such that $f \bmod \mathfrak{q}$ is separable. In this case we choose $\mathfrak{q} = \langle \alpha - 3 \rangle$. Hence, for any $\beta \in \mathcal{O}_{\mathbb{Q}(\alpha)}$, $\beta \bmod \mathfrak{q} \in \mathbb{Q}$. The only coefficient of $\tilde{g}_1$ and $\tilde{g}_2$ not in $\mathbb{Q}$ is α^4. Hence, we have to show that $\sigma_3(\alpha^4) \equiv \alpha^4 \bmod \mathfrak{q}$. This is equivalent to*

$$(x^4 \bmod \mathfrak{q}) \bmod (f_3 \bmod \mathfrak{q}) = \alpha^4 \bmod \mathfrak{q}. \tag{3.10}$$

Notice that the left-hand side is computed over $\mathbb{Q}$. Moreover, if we choose a prime number p according to Remark 3.42, then we can also check the equality in (3.10) modulo the prime number p. In this case, we choose $p = 11$. We have $f_3 \bmod \mathfrak{q} = x^2 + 3^2 = x^2 + 9$ and hence,

$$l.h.s. = x^4 \bmod x^2 + 9 = 4 \,(\text{mod } 11).$$

On the other hand, the right-hand side of Equation (3.10) *becomes*

$$r.h.s. = \alpha^4 \bmod \mathfrak{q} = 3^4 = 4 \,(\text{mod } 11).$$

This is enough to conclude that $\tilde{P}_3$ is the correct partition of L_3. Let us suppose that we had erroneously guessed the partition for L_3 (given by $P = \{\{1,2\},\{3\},\{4\}\}$). In this case, the P-products are $x^2 - \alpha^2$, $x^2 + \alpha^2$ and $x^4 + \alpha^4$. The only coefficients not in $\mathbb{Q}$ are α^2 and α^4. We already showed that $\sigma_3(\alpha^4) \equiv \alpha^4 \bmod \mathfrak{q}$. However,

$\sigma_3(\alpha^2) \bmod \mathfrak{q} = x^2 \bmod x^2 - 9 = 2 \pmod{11}$ *and* $\alpha^2 \bmod \mathfrak{q} = 3^2 = 9 \pmod{11}$. *This proves that the partition* $\{\{1,2\},\{3\},\{4\}\}$ *is not the partition of* L_3.

Theorem 3.45. *If the Algorithm* `Join` *finishes, the output is the partition* P_i *of the principal subfield* L_i. *Moreover, assuming that the number of calls to* `SystemModP` *is bounded by a constant, when* $k = \mathbb{Q}$, *the number of CPU operations for computing the partition* P_i *is*

$$\tilde{\mathcal{O}}(n(n^2 + d_i r^{\omega-1} + n \log \|f\|)),$$

where we omit $\log p$ *factors in* $\tilde{\mathcal{O}}$ *notation (see Remark 3.46).*

Proof. Let $\tilde{P}_i$ be the output of Algorithm `Partition`, that is, $\tilde{P}_i$ is the partition at step 6. Since $\tilde{P}_i$ comes from the $\{0,1\}$-echelon basis of $\tilde{\mathcal{S}}_i$, and $\tilde{\mathcal{S}}_i$ is a subset of $\mathcal{S}_i$ reduced modulo a prime ideal $\mathfrak{p}$, it follows that $\tilde{P}_i$ is a refinement of P_i. Moreover, if the P-products $\tilde{g}_1, \ldots, \tilde{g}_t$ satisfy Equation 3.9, then $\tilde{P}_i = P_i$, by Theorem 3.43. To prove the complexity bound, we first bound the cost of calling algorithm `SystemModP`. The integer coefficients of $f'(\alpha) f_j \in \mathbb{Z}[\alpha][x]$ can be bounded by $n4^n \|f\|^2$ (see Lemma 4.18, Appendix 4.4). Hence, computing $f'(\alpha) f_j(c)$ modulo p, for $1 \leq j \leq r$, has a cost of $\tilde{\mathcal{O}}(n^2(n + \log \|f\|))$ CPU operations. The divisions $f_j'(c)/f_j(c) \bmod p = f'(\alpha) f_j'(c)/f'(\alpha) f_j(c) \bmod p$ in step 3 of `SystemModP` can be executed with $\tilde{\mathcal{O}}(rn \log p)$ CPU operations and step 4 has a cost of $\tilde{\mathcal{O}}(rn^2 \log p)$ CPU operations. Hence, by omitting $\log p$ factors, one call of `SystemModP` has a cost of $\tilde{\mathcal{O}}(n^2(n + \log \|f\|))$ CPU operations. In our experiments, the number of calls to algorithm `SystemModP` from algorithm `Partition` was never more than 3. Usually 1 call sufficed to find the partition P_i. In this case, the system $\tilde{\mathcal{S}}_i$ has at most nd_i equations in r variables and hence, a solution basis can be found with $\mathcal{O}(nd_i r^{\omega-1})$ field operations in $\mathbb{F}_p$ or $\tilde{\mathcal{O}}(nd_i r^{\omega-1} \log p)$ CPU operations. The cost of steps 7-11 in algorithm `Partition` is given by the cost of computing the polynomials $\tilde{g}_j, 1 \leq j \leq t$, which can be done with at most $r - 1$ polynomial multiplications in

$\mathbb{F}_p(\alpha)[x]$, and the cost of nt divisions in $\mathbb{F}_p[x]$. The result follows by omitting $\log p$ factors. □

Remark 3.46. *One can design the algorithm to work with any prime p for which $f \bmod p$ is separable and p does not divide the leading coefficient of f. Then $\log p$ can be bounded by $\mathcal{O}(\log(n + \|f\|))$ by Equation (3.9) in [35]. However, it is best to choose p for which f has a root* mod$p$*. The probability that f has a root* mod$p$ *for a random prime p is asymptotically at least $1/n$, by Chebotarev's density theorem. With the (unproven, but true in experiments) assumption that this probability is not much smaller for small p, the expected size for $\log p$ is still bounded by $\mathcal{O}(\log(n + \|f\|))$.*

3.4 General Algorithm and Generators

In this section we combine the ideas of Sections 3.1, 3.2 and 3.3 and give a general algorithm for computing the subfields of K/k.

3.4.1 The `Subfields` Algorithm

The following algorithm returns the partition-vectors corresponding to all subfields of K/k. Step 1 asks for a subfield factorization of f, which can be computed by fully factoring f over K (or an extension of). When $k = \mathbb{Q}$, one can also use p-adic factorization and LLL (see next Chapter). Steps 2-4 involve computing the partition of the principal subfields, according to Section 3.3. Finally, Steps 5-8 compute the partition-vectors of all subfields of K/k.

Algorithm 3.6 `Subfields`

Input: An irreducible squarefree polynomial $f \in k[x]$.

Output: A data structure that lists all subfields of K/k (given by partition-vectors).

1. Compute a subfield factorization $f_1 \cdots f_r$ of f in $K[x]$.
2. **for** $i = 1, \ldots, r$ **do**
3. Compute the partition P_i using algorithm `Partition`.
4. $\mathbf{v}_i :=$ the vector of P_i (see Definition 3.20).
5. $S_0 := \{\mathbf{v}_1, \ldots, \mathbf{v}_r\}$.
6. $S := S_0$.
7. **for** $\mathbf{v}$ in S_0 **do**
8. $S := S \cup \{\mathbf{v} \vee \mathbf{w} : \mathbf{w} \text{ in } S\}$.
9. **return** S and $[f_1, \ldots, f_r]$.

The output of algorithm `Subfields` is a set S which contains the partition-vector for every subfield of K/k. This output is particularly useful if one wants the subfield lattice of the extension K/k. On the other hand, the set S and the subfield factorization of f allow us to give the subfield polynomial of each subfield of K/k in (partially) factored form. One can also compute generators for each subfield, see Section 3.4.2. Next, we analyze the complexity of algorithm `Subfields` for the case $k = \mathbb{Q}$.

Theorem 3.47. *Let m be the number of subfields of K/k. When $k = \mathbb{Q}$, algorithm* `Subfields` *has an expected cost of $\tilde{\mathcal{O}}(rn^7 + rn^5 \log^2 \|f\|_2 + mr^2)$ CPU operations, where n is the degree of the extension K/k and r is the number of factors in the subfield factorization.*

Proof. In Step 1 we have to compute a subfield factorization of f over K. To find such factorization one can compute the irreducible factorization of f over $\mathbb{Q}(\alpha)$

(see [47, 8, 40], complexities not stated). Alternatively, one can use Algorithm `SubfFact`, presented in Chapter 4, which finds a subfield factorization with $\tilde{\mathcal{O}}(rn^7 + rn^5 \log^2 \|f\|)$ CPU operations. In Steps 2-4 we have to compute r partitions, where each partition can be computed with an expected number of $\tilde{\mathcal{O}}(n(n^2 + d_i r^{\omega-1} + n \log \|f\|))$ CPU operations, where d_i is the degree of f_i. Finally, the set S never has more than m elements, and the set S_0 has at most r elements. Therefore, the number of times we compute $\mathbf{v} \vee \mathbf{w}$ is bounded by rm. Since the cost of each partition join is $\tilde{\mathcal{O}}(r)$, the cost of steps 7-8 is given by $\tilde{\mathcal{O}}(mr^2)$ CPU operations. □

Steps 7-8 compute all intersections of the principal subfields, but (this simple implementation) may compute the same subfield several times. Although the number of intersections is bounded by rm, this part can be improved by using the Algorithm `AllSubfields` from [51]. While the bound for the number of intersections in `AllSubfields` is still rm, this algorithm avoids computing a subfield already computed, which can be a big improvement when we have several subfields. Since the number of subfields m is not polynomially bounded, the theoretical worst-case complexity is dominated by the cost of all intersections of the principal subfields $L_1, \ldots, L_r$. Since each subfield is represented by a partition and the intersection of subfields can be computed by joining partitions, we were able to improve the theoretical complexity. Moreover, computing all subfields using partitions only contributes to a small percentage of the total CPU time.

3.4.2 From a Partition to a Subfield

In addition to returning the subfield lattice (in terms of partition-vectors), one can also compute generators for any subfield of K/k. Let $f_1, \ldots, f_r$ be a subfield factorization and let $L_1, \ldots, L_r$ be the principal subfields. Given a partition P_L, corresponding to a subfield L of K/k, one can find a set of generators

of L by computing the subfield polynomial g_L of L (given by the product of f_j, for $j \in P_L^{(1)}$) and taking its coefficients (see Theorem 3.4).

Algorithm 3.7 `Generators` (Slow version).

Input: Subfield factorization $f_1, \ldots, f_r$ of f and the partition P_L.

Output: A set of generators of the subfield L of K/k.

1. Compute $g_L := \prod_{j \in P_L^{(1)}} f_j$.
2. **return** the set of coefficients of g_L.

However, expanding the subfield polynomial can be an expensive task, especially when g_L has high degree. Alternatively, one can compute only a few (easy to compute) coefficients of g_L (for example, if $d = \deg(g_L)$, then the coefficients of x^{d-1} and x^0 are easy to compute from the partial factorization of g_L) or one can compute $g_L(c) = \prod_{i \in P_L^{(1)}} f_i(c)$, for $c \in k$, for as many c as we want. Let us denote by `NextElem`( ) a procedure that returns elements of L. What we need now is a practical criterion that tells us when a set of elements of L generates L.

Theorem 3.48. *Let $\beta_1, \ldots, \beta_s \in L$ and let P_L be the partition defined by L. Then $L = k(\beta_1, \ldots, \beta_s)$ if and only if, for any $j \notin P_L^{(1)}$ there exists $l \in \{1, \ldots, s\}$ with $\beta_l \notin L_j$.*

Proof. Notice that $L \cap L_j \subsetneq L$, for any $j \notin P_L^{(1)}$. Hence, if there exists some $j \notin P_L^{(1)}$ such that $\beta_i \in L_j$, for every β_i, then $k(\beta_1 \ldots, \beta_s) \subseteq L \cap L_j \subsetneq L$. Conversely, let $\beta_1 \ldots, \beta_s \in L$ be such that for any $j \notin P_L^{(1)}$, there exists β_i such that $\beta_i \notin L_j$. Let $\tilde{L} := k(\beta_1 \ldots, \beta_s)$ and suppose that $\tilde{L} \subsetneq L$. Let $P_{\tilde{L}}$ be the partition defined by $\tilde{L}$. By Lemma 3.16 we have $P_L^{(1)} \subsetneq P_{\tilde{L}}^{(1)}$ and hence, there exists $j \in P_{\tilde{L}}^{(1)}$ such that $j \notin P_L^{(1)}$ and $\beta_i \in L_j$, for any $i \in P_L^{(1)}$, which is a contradiction. Therefore, $L = k(\beta_1 \ldots, \beta_s)$. □

Recall that for any element $\beta \in K$, there exists $g(x) \in k[x]_{<n}$ such that $\beta = g(\alpha)$ and that $\beta \in L_j$ if and only if, $g(x) \equiv g(\alpha) \bmod f_j$. To show that $\beta \notin L_j$, it is enough to show that $g(x) \not\equiv g(\alpha) \bmod (f_j, \mathfrak{p})$, where $\mathfrak{p}$ is as in Definition 3.39. Theorem 3.48 allows us to write an algorithm for finding a set of generators of L.

Algorithm 3.8 `Generators` (Fast version)

Input: Subfield factorization $f_1, \ldots, f_r$ of f and the partition P_L.

Output: A set of generators of the subfield L of K/k.

1. $S := \emptyset$.
2. $J := \{1, \ldots, r\} - P_L^{(1)}$.
3. $\beta :=$ `NextElem`( ), where $\beta = g(\alpha)$, for some $g(x) \in k[x]_{<n}$.
4. $S := S \cup \{\beta\}$.
5. **for** $j \in J$ **do**
6. **if** $g(x) \not\equiv g(\alpha) \bmod (f_j, \mathfrak{p})$ **then** $J := J - \{j\}$.
7. **if** $J \neq \emptyset$ **then** Go to Step 3 **else return** S.

Theorem 3.49. *The output of Algorithm 3.8 is a set $S \subseteq L$ which generates L.*

Proof. If $g(x) \not\equiv g(\alpha) \bmod (f_j, \mathfrak{p})$ in Step 6, then $g(x) \not\equiv g(\alpha) \bmod f_j$ and hence, $g(\alpha) \notin L_j$. If S is the output of Algorithm `Generators`, then for any $j \notin P_L^{(1)}$, there exists $\beta \in S$ such that $\beta \notin L_j$. By Theorem 3.48, S generates L. □

This algorithm can also be used to decide if $\beta \in \mathbb{Q}(\alpha)$ is a generator for a subfield L. Algorithm `Generators`, as it is stated, is not guaranteed to finish. However, if the algorithm has not found a generating set after a certain number of elements computed, one could compute the subfield polynomial and return its coefficients.

4 THE NUMBER FIELD CASE

Let K/k be a finite and separable field extension, let α be a primitive element and let $f \in k[x]$ be the minimal polynomial of α over k. In light of the previous chapter, given a subfield factorization $f_1 \cdots f_r$ of f, we can use Algorithm `Partitions` to compute the partitions $P_1, \ldots, P_r$ of the principal subfields $L_1, \ldots, L_r$ and compute their intersections using Algorithm `Join`. This gives us the partition of every subfield of K/k and we can compute generators for each of them using Algorithm `Generators`. When $k = \mathbb{Q}$, we have two methods to find a subfield factorization:

Method 1. Fully factoring f over $K = \mathbb{Q}(\alpha)$. One can use Trager's method [47] or Belabas's Algorithm [8] (a generalization of van Hoeij's factorization algorithm for number fields).

Method 2. Use a p-adic factorization of f, for appropriate p, and LLL to find the principal subfields (as in van Hoeij *et al.* [51]).

We compared both methods to see which works best when adjusted to our new approach for the intersections. Based on timings in Magma (see the columns `SubfFact` and (Factorization) in Table 4.1) and since the cut-off bound for the LLL given in [51] is essentially optimal, we expected Method 2 to be faster. However, we noticed that this was not the case after trying to factor f over $\mathbb{Q}(\alpha)$ using the algorithm from Belabas [8] in Pari/GP [45]. Method 2 and Belabas' factorization algorithm both use LLL, but Method 2 does this for every p-adic factor separately, introducing a factor $\tilde{r}$. Without additional results, (could the LLL-work for each p-adic factor in Method 2 be shared?), Method 1 with Belabas' factorization algorithm gives the best timings. However, we still develop Method 2 to get a complexity bound (as there is no complexity bound for Belabas' factorization algorithm).

4.1 Computing a Subfield Factorization (Method 2)

In this section, we show how one can use p-adic factorization and LLL to directly compute a subfield factorization. The idea is to find a primitive element of L which, in turn, gives us the subfield polynomial of L (recall Theorem 3.4, item 6.). Once we have the subfield polynomial of all principal subfields, we construct a subfield factorization.

As usual, we start by choosing a prime p such that p does not divide the leading coefficient of $f \in \mathbb{Z}[x]$, $f \bmod p$ is separable and has at least one linear factor in $\mathbb{F}_p[x]$, which we denote by $\bar{f}_1$. Let $\hat{K} = \mathbb{Q}_p$ be the field of p-adic numbers. The factorization $\bar{f}_1, \ldots, \bar{f}_{\hat{r}}$ of $f \bmod p$ lifts to a factorization $\hat{f}_1 \cdots \hat{f}_{\hat{r}}$ of f into irreducible factors over $\mathbb{Q}_p$, with $\hat{f}_1$ linear. Computationally, we can only compute p-adic factors with finite accuracy. For $i = 1, \ldots, \hat{r}$ and a positive integer a, let $f_i^{(a)} \in p^a\mathbb{Z}[x]$ be an approximation of $\hat{f}_i$ with accuracy a, that is, $\hat{f}_i \equiv f_i^{(a)} \bmod p^a$ (Hensel lifting). By mapping $\alpha \in \mathbb{Q}(\alpha)$ to the root $\hat{\alpha}$ of $\hat{f}_1$ in $\mathbb{Q}_p$, we can view $\mathbb{Q}(\alpha)$ as a subfield of $\mathbb{Q}_p$.

For $g \in \mathbb{Q}(\alpha)[x]$, we will denote by $\bar{g} \in \mathbb{F}_p[x]$, the image of g under the map $\alpha \to \bar{\alpha}$, where $\bar{\alpha}$ is the root of $\bar{f}_1$, and by $\hat{g} \in \mathbb{Q}_p[x]$, the image of g under the map $\alpha \to \hat{\alpha}$, where $\hat{\alpha}$ is the root of $\hat{f}_1$. Furthermore, for $g, h \in \mathbb{Q}(\alpha)[x]$, we denote by $\gcd_p(g, h)$ the gcd of the images $\bar{g}$ and $\bar{h}$ over $\mathbb{F}_p$.

As shown in [51], one can use LLL to compute linearly independent algebraic numbers $\beta_1, \ldots, \beta_{m_i} \in \mathbb{Q}(\alpha)$ which (likely) form a $\mathbb{Q}$-basis of L_i (it is only guaranteed that $L_i \subseteq \mathbb{Q}\cdot\beta_1 + \cdots + \mathbb{Q}\cdot\beta_{m_i}$ as vector spaces). Recall that $f_1, \ldots, f_r$ is a subfield factorization if for every principal subfield L_i with subfield polynomial g, we have $g \in \{f_1, \ldots, f_r\}^\pi$ (recall Notation 3.8). The idea of the algorithm below comes from the following fact: let $g_1, \ldots, g_s$ be any factorization of f. If, for every j such

that $G := \gcd(g_j, g) \neq 1$, we replace g_j by G and g_j/G, then the new factorization $G_1, \ldots, G_S$ is such that $g \in \{G_1, \ldots, G_S\}^\pi$.

Algorithm 4.1 `PartialSubfFact`

Input: A $\mathbb{Q}$-basis $\beta_1, \ldots, \beta_{m_i}$ of some V such that $L_i \subseteq V$ and a factorization $g_1, \ldots, g_s$ of f.

Output: A partial factorization $G_1, \ldots, G_S$ of f over $\mathbb{Q}(\alpha)$, with $s \leq S$, and such that $g_{L_i} \in \{G_1, \ldots, G_S\}^\pi$ or **Error**.

1. Let $SF := \{g_1, \ldots, g_s\}$ and let $T \subseteq k$ finite.
2. Let β be a random T-combination of $\beta_1, \ldots, \beta_{m_i}$.
3. Let $H := h(x) - h(\alpha)$, where $h(x) \in \mathbb{Z}[x]_{<n}$ and $h(\alpha) = \beta$.
4. Compute $g_0 := \gcd_p(f, H)$ in $\mathbb{F}_p[x]$.
5. **if** $\deg(g_0) \cdot m_i \neq n$ **then** go to Step 2.
6. **for** $j = 1, \ldots, s$ **do**
7. Compute $g := \gcd_p(g_j, H)$ in $\mathbb{F}_p[x]$.
8. **if** $0 < \deg(g) < \deg(g_j)$ **then**
9. Compute $G := \gcd(g_j, H)$ in $\mathbb{Q}(\alpha)[x]$.
10. **if** $\bar{f}_i \mid \bar{g}_j$ but $\bar{f}_i \nmid \bar{G}$ **then return Error**.
11. $SF := (SF - \{g_j\}) \cup \{G, g_j/G\}$.
12. **return** SF

The purpose of the gcd computations mod p in Steps 4 and 7 is to avoid expensive gcd computations over $\mathbb{Q}(\alpha)$. Since $\beta_1, \ldots, \beta_{m_i}$ is not guaranteed to be a $\mathbb{Q}$-basis for L_i, we might run into some problems. For instance, if $\beta_1, \ldots, \beta_{m_i}$ is not a $\mathbb{Q}$-basis of L_i, the element β in step 2 might never be a primitive element and hence, Step 5 sends the algorithm into an infinite loop. Otherwise, $\deg(g_0) \cdot m_i \neq n$ when the random element β is not a generator of L_i, which happens with probability at most $(m_i - 1)|T|^{m_i(1-q)/q}$, where q is the smallest prime that divides m_i (see Appendix 4.3). To prove the correctness of the algorithm, we use the following remark.

Remark 4.1. *As a consequence of Lemma 3.41, if $g, h \in \mathbb{Q}(\alpha)[x]$ are factors of f then one can quickly verify whether or not $h \mid g$ by checking whether the image of h in $\mathbb{F}_p[x]$ divides the image of g in $\mathbb{F}_p[x]$. The same holds for deciding when $\gcd(g, h) \in \mathbb{Q}(\alpha)[x]$ is trivial or not.*

Lemma 4.2. *If algorithm* `PartialSubfFact` *does not end in an error message, then the input $\beta_1, \ldots, \beta_{m_i}$ is a basis of L_i, and moreover, $L_i = \mathbb{Q}(\beta)$, with β from Step 2. If Step 10 returns an error message, then $\beta_1, \ldots, \beta_{m_i}$ is not a basis of L_i.*

Proof. Let g_{L_i} be the subfield polynomial of L_i and let $g_\beta := \gcd(f, h(x) - h(\alpha))$ be the subfield polynomial of $\mathbb{Q}(\beta)$ (see Theorem 3.4, item 6.). Let $g_0 \in \mathbb{F}_p[x]$ be as in Step 4. It follows that

$$\deg(g_0) \geq \deg(\bar{g}_\beta) = \deg(g_\beta). \tag{4.1}$$

Furthermore, since $L_i \subseteq V$ as $\mathbb{Q}$-vector spaces, we have $\dim(V) \geq \dim(L_i)$. But $\dim(L_i) = n/\deg(g_{L_i})$ and $\dim(V) = m_i$. Hence,

$$\deg(g_{L_i}) \geq n/m_i. \tag{4.2}$$

If Step 5 does not generate an infinite loop (in which case the algorithm should return an error message), then $\deg(g_0) \cdot m_i = n$ and hence, Equations (4.1) and (4.2) tell us that

$$\deg(g_{L_i}) \geq n/m_i = \deg(g_0) \geq \deg(g_\beta). \tag{4.3}$$

Now suppose that Step 10 did not return an error message. Since f is separable modulo p, there is only one index I, $1 \leq I \leq s$, such that $\bar{f}_i \mid \bar{G}$, where $G = \gcd(g_I, H)$. If F is the irreducible factor of f over $\mathbb{Q}(\alpha)$ such that $\bar{f}_i \mid \bar{F}$, then using Remark 4.1 one can show that $F \mid G \mid g_\beta$ and hence, $\mathbb{Q}(\beta) = L_{g_\beta} \subseteq L_F$. On the other hand, if $\hat{f}_i$ is the p-adic factor of f which reduces to $\bar{f}_i$ modulo p, then $\hat{f}_i \mid \hat{F}$ and hence, $L_F \subseteq L_{\hat{F}} \subseteq L_{\hat{f}_i} = L_i$. Therefore $\mathbb{Q}(\beta) \subseteq L_i$ and thus, $g_{L_i} \mid g_\beta$,

by Lemmas 3.13 and 3.16. By Equation (4.3), we have $g_{L_i} = g_\beta = \gcd(f, H)$ and hence,

$$L_i = \mathbb{Q}(\beta) = V.$$

Notice that this also shows that the polynomials g in Step 7 and G in Step 9 have the same degree. If the algorithm does return an error message in Step 10, then $\bar{f}_i \mid \bar{g}_I$ but $\bar{f}_i \nmid \bar{G}$ (and hence, $\bar{f}_i \nmid \bar{H}$). By looking at the images over the p-adic numbers, we also must have $\hat{f}_i \nmid \hat{H}$, which means that $h(\alpha) = \beta \notin L_i$. Hence, $\beta_1, \ldots, \beta_{m_i}$ is not a basis of L_i. □

Theorem 4.3. *Let g_{L_i} be the subfield polynomial of L_i. Given a $\mathbb{Q}$-basis of $V \supseteq L_i$ and a (partial) factorization of f, Algorithm* `PartialSubfFact` *returns a (partial) factorization $G_1, \ldots, G_S$ of f such that $g_{L_i} \in \{G_1, \ldots, G_S\}^\pi$ or an error message.*

Proof. If the algorithm does not return an error message, then by Lemma 4.2 it follows that $g_{L_i} = \gcd(f, H)$. Hence, by computing the gcd of H with the partial factorization of f and updating the set SF (Step 11), it follows that the output SF in Step 12 is such that $g_{L_i} \in SF^\pi$. □

Different bases for $\mathbb{Q}(\alpha)$ give different bounds on the bit-size of $\beta_1, \ldots, \beta_{m_i}$. While the standard basis $\{1, \alpha, \ldots, \alpha^{n-1}\}$ simplifies implementation, the *rational univariate representation basis* $\{1/f'(\alpha), \ldots, \alpha^{n-1}/f'(\alpha)\}$ can improve running times and provide better complexity results, see [5] and [15].

Besides giving better bounds, there are more advantages in using the rational univariate representation basis. For example, if g is a monic factor of f in $\mathbb{Q}(\alpha)[x]$, then $f'(\alpha)g \in \mathbb{Z}[\alpha][x]$ (see [54]). This allows us to make simplifications in a general algorithm for computing gcd's in $\mathbb{Q}(\alpha)[x]$, giving better complexity results. See Appendix 4.4.

Remark 4.4. *Suppose that $\beta_1, \ldots, \beta_{m_i}$ is a $\mathbb{Q}$-basis of $V \supseteq L_i$. Let β be a random T-combination of $\beta_1, \ldots, \beta_{m_i}$ and let $b_0, \ldots, b_{n-1} \in \mathbb{Z}$ be such that $\beta = \sum b_j \frac{\alpha^j}{f'(\alpha)}$. If $\tilde{h}(x) = \sum b_j x^j \in \mathbb{Z}[x]$, then one should define $H(x)$ as $\tilde{h}(x)f'(\alpha) - \tilde{h}(\alpha)f'(x) \in \mathbb{Z}[\alpha][x]$ in Step 3 of Algorithm* `PartialSubfFact`.

Lemma 4.5. *Given a $\mathbb{Q}$-basis of $V \supseteq L_i$ (computed in the rational univariate representation basis) and a partial factorization $g_1, \ldots, g_s$ of f, the number of CPU operations for running Algorithm* `PartialSubfFact` is bounded by $\tilde{\mathcal{O}}(n^3(r + \log \|f\|_2))$.

Proof. The cost of Steps 4 and 7 is less than the cost of Step 9. The cost of the division g_j/G in Step 11 is similar to the cost of the gcd in Step 9 (this division can be computed by dividing the images of g_j and G in $\mathbb{F}_p(\alpha)[x]$ and then Chinese remaindering). Since f is separable modulo p, there is only one g_I such that $\bar{f}_i \mid \bar{g}_I$ and if

$$\bar{f}_i \mid \bar{G}, \text{ where } G = \gcd(g_I, H), \tag{4.4}$$

then, by the proof of Lemma 4.2, we have

$$\deg(\gcd(g_j, H)) = \deg(\gcd_p(g_j, H)), \text{ for any } 1 \leq j \leq s$$

and hence, when computing $\gcd(g_j, H)$, $j \neq I$, we can skip the trial divisions in the modular gcd algorithm (see [52] and Appendix 4.4). That is, we have one gcd computation with trial divisions, which costs $\tilde{\mathcal{O}}(n^3 \log \|f\|_2)$ CPU operations, and if (4.4) holds, then we can skip the trial divisions in the remaining gcd's, where each such gcd costs $\tilde{\mathcal{O}}(n^2(n+\log \|f\|_2))$ CPU operations (see Appendix 4.4). Furthermore, each division test in step 10 costs $\tilde{\mathcal{O}}(n \log p)$ CPU operations. The result follows by omitting $\log p$ terms and using the fact that $s \leq n$. □

The algorithm `PartialSubfFact` creates a factorization that contains the subfield polynomial of a single principal subfield. By iterating through all principal subfields we get a subfield factorization. A general description of the algorithm to compute a subfield factorization of f over $\mathbb{Q}(\alpha)$ is given below.

Algorithm 4.2 `SubfFact`.

Input: A squarefree irreducible polynomial $f \in \mathbb{Z}[x]$.

Output: A subfield factorization of f over $\mathbb{Q}(\alpha)$.

1. Let p prime for which $\bar{f}$ is separable, has a linear factor and $\deg(f) = \deg(\bar{f})$.
2. Compute the irreducible factorization $\bar{f}_1, \ldots, \bar{f}_{\hat{r}}$ of $\bar{f} \in \mathbb{F}_p[x]$.
3. $SF_0 := \{x - \alpha, f/(x - \alpha)\}$.
4. **for** $i = 1, \ldots, \hat{r}$ **do**
5. Hensel lift $\bar{f}_1, \ldots, \bar{f}_{\hat{r}}$ to a factorization $f_1^{(a)}, \ldots, f_{\hat{r}}^{(a)}$ of $f \bmod p^a$, for some a.
6. Use LLL to compute a basis $\beta_1, \ldots, \beta_{m_i}$ of some $V \supseteq L_i$.
7. SF_i:=`PartialSubfFact`$(\{\beta_1, \ldots, \beta_{m_i}\}, SF_{i-1})$.
8. If $SF_i =$ **Error**, increase the lifting precision a, go to Step 5.
9. **return** $SF_{\hat{r}}$.

The starting precision a from Step 5 is the same as that from [51]. Likewise, the basis $\beta_1, \ldots, \beta_{m_i}$ is computed as explained in [51].

Remark 4.6. *While computing the subfield factorization, whenever we find a linear factor $x - h_1(\alpha) \in \mathbb{Q}[\alpha][x]$ of f, we can use it to find new linear factors in the following way: if $x - h_2(\alpha)$ is another linear factor, then $h_1(h_2(\alpha))$ is also a root of f. This follows from the fact that $\phi : K \to K$, $\alpha \mapsto h_1(\alpha)$ is an automorphism of K/k (which permutes the roots of f). This is particularly helpful when f has several roots in K, since the number of LLL calls can be reduced significantly.*

Example 4.7. *Let $f = S_3(x) = x^8 - 40x^6 + 352x^4 - 960x^2 + 576 \in \mathbb{Q}[x]$ be the Swinnerton-Dyer polynomial of index 3 and let α be a root of f. Let us compute a subfield factorization for $\mathbb{Q}(\alpha)/\mathbb{Q}$. First of all, we need to choose a prime p such that $f \bmod p$ is separable, has a linear factor and the same degree as f. One such prime is $p = 1009$ since $f \bmod p$ factors as*

$$f \bmod p = (x+46)(x+177)(x+344)(x+475)(x+534)(x+665)(x+832)(x+963),$$

whose factors we call $\bar{f}_1, \ldots, \bar{f}_8$*, respectively. This prime is chosen as the first prime greater than 1000 with the required properties*[1]*. Now, let* $\bar{\alpha} = -46 = 963$ *be the root of* $\bar{f}_1$ *and let* $\hat{\alpha}$ *be the* p*-adic root of* f *such that* $\hat{\alpha} \equiv \bar{\alpha} \bmod p$*. The principal subfield corresponding to* $\bar{f}_1 = x + 46$ *we already know to be* $\mathbb{Q}(\alpha)$*. So we start with* $r = 2$ *and* $\bar{f}_2 = x + 177$*. By Hensel Lifting the factorization above and using LLL, we get a* $\mathbb{Q}$*-basis for some* V *such that* $L_2 \subseteq V$*, given by*

$$\begin{aligned}
\beta_1 &= (\alpha^4 - 4\alpha^2 - 24)/f'(\alpha), \\
\beta_2 &= (\alpha^5 - 12\alpha^3 + 24\alpha)/f'(\alpha), \\
\beta_3 &= (\alpha^7 - 19\alpha^5 + 44\alpha^3 + 24\alpha)/f'(\alpha), \\
\beta_4 &= (\alpha^6 - 19\alpha^4 + 68\alpha^2 - 24)/f'(\alpha).
\end{aligned}$$

These elements are computed in the rational representation basis. Notice that if we rewrite, for instance, the element β_1 *in the canonical representation (i.e., in the basis* $1, \alpha, \alpha^2, \ldots, \alpha^7$*), we get*

$$\beta_1 = -\frac{1}{6144}\alpha^7 + \frac{47}{7680}\alpha^5 - \frac{11}{256}\alpha^3 + \frac{1}{15}\alpha,$$

which have much larger coefficients[2]*. Now we need to call Algorithm* `PartialSubfFact`*. Let* $T = \{-10, \ldots, 10\}$*. The first step is to compute a random* T*-combination of* $\beta_1, \ldots, \beta_4$*, say*

$$\beta := 10\beta_1 + 5\beta_2 - \beta_3 - 6\beta_4 = (-\alpha^7 - 6\alpha^6 + 24\alpha^5 + 124\alpha^4 - 104\alpha^3 - 448\alpha^2 + 96\alpha - 96)/f'(\alpha).$$

According to Remark 4.4, let $H(x) = \tilde{h}(x)f'(\alpha) - \tilde{h}(\alpha)f'(x)$ *(too long to display), where*

$$\tilde{h}(x) = -x^7 - 6x^6 + 24x^5 + 124x^4 - 104x^3 - 448x^2 + 96x - 96.$$

[1] *If the prime* p *is very small, we may not have enough elements to compute the partitions (though the primes used to compute the partitions and the subfield factorization could be different, we shall use the same one), recall Example 3.40. This is why* p *is chosen this way.*

[2] *Larger in the sense that more bits of information are needed to represent* β_1*.*

The next step is to compute the gcd *of the images* $\bar{f}$ *and* $\bar{H}$ *of* f *and* H *in* $\mathbb{F}_p[x]$*, that is, replacing* α *by* $\bar{\alpha}$ *and considering the resulting polynomial over* $\mathbb{F}_p$*. By doing so we have* $\bar{H} = 539x^7 + 32x^6 + 6x^5 + 684x^4 + 643x^3 + 35x^2 + 560x + 512$ *and hence*

$$g_0 := gcd_p(f, H) = \gcd(\bar{f}, \bar{H}) = x^2 + 223x + 70.$$

Notice that $\deg(g_0) \cdot m_i = 2 \cdot 4 = 8 = n$ *(where* m_i *is the dimension of* V*, defined by* $\beta_1, \ldots, \beta_4$*). At this point, our partial subfield factorization is just* $SF_0 = \{x - \alpha, f/(x-\alpha)\} =: \{g_1, g_2\}$*. Now we compute the* gcd *of the images of* H *and the* g_i*'s. Since* $g_1 = x - \alpha$*, we can skip this factor. For* $g_2 = f/(x-\alpha)$*, we have*

$$g := gcd_p(g_2, H) = x + 177.$$

Since $0 < \deg(g) < \deg(g_2)$*, we compute the* $G := \gcd(g_2, H)$ *(now over* $\mathbb{Q}(\alpha)$*), which yields*

$$G := \gcd(g_2, H) = x - \left(-\frac{1}{288}\alpha^7 + \frac{7}{72}\alpha^5 + \frac{7}{36}\alpha^3 - \frac{7}{3}\alpha\right) =: x - h_1(\alpha).$$

Notice that $\bar{f}_2 = x + 177 \mid \bar{g}_2$ *and* $\bar{f}_2 \mid \bar{G} = x + 177$*, which proves that* $\gcd(f, H)$ *is indeed the subfield polynomial of* L_2 *(recall Lemma 4.2). Hence, the partial subfield factorization is now*

$$SF_2 := \{x - \alpha, G = x - h_1(\alpha), g_2/G\} =: \{g_1, g_2, g_3\}.$$

We now use $\bar{f}_3 = x + 344$*. Hensel lifting and LLL return the* $\mathbb{Q}$*-basis*

$$\begin{aligned}
\beta_1 &= (\alpha^4 + 4\alpha^2 - 24)/f'(\alpha),\\
\beta_2 &= (\alpha^5 - 8\alpha^3 + 24\alpha)/f'(\alpha),\\
\beta_3 &= (\alpha^6 - 20\alpha^4 + 72\alpha^2)/f'(\alpha),\\
\beta_4 &= (\alpha^7 - 19\alpha^5 + 88\alpha^3 + 24\alpha)/f'(\alpha).
\end{aligned}$$

of some V *such that* $L_3 \subseteq V$*. Again, we call Algorithm* `PartialSubfFact`*. The random* T*-combination we choose this time is*

$$\beta := 3\beta_1 + 10\beta_2 + \beta_3 - 9\beta_4 = (-9\alpha^7 + \alpha^6 + 181\alpha^5 - 17\alpha^4 - 872\alpha^3 + 84\alpha^2 + 24\alpha - 72)/f'(\alpha).$$

Again, we define $H(x) = \tilde{h}(x)f'(\alpha) - \tilde{h}(\alpha)f'(x)$, *where*

$$\tilde{h}(x) = -9x^7 + x^6 + 181x^5 - 17x^4 - 872x^3 + 84x^2 + 24x - 72$$

and compute $g_0 := gcd_p(f, H) = \gcd(\bar{f}, \bar{H}) = x^2 + 390x + 689$. *Again, we have* $\deg(g_0) \cdot m_i = 8 = n$, *and hence, we may continue. Now we compute* $gcd_p(g_j, H)$, *for every* $g_j \in SF_2 = \{g_1, g_2, g_3\}$. *The only* g_j *for which* $gcd_p(g_j, H)$ *is not trivial is* g_3, *yielding* $g := gcd_p(g_3, H) = x + 344$ *and*

$$G := \gcd(g_3, H) = x - \left(\frac{1}{48}\alpha^7 - \frac{37}{48}\alpha^5 + \frac{61}{12}\alpha^3 - \frac{13}{2}\alpha\right) =: x - h_2(\alpha).$$

Again, we have $\bar{f}_3 = x + 344 \mid \bar{g}_3$ *and* $\bar{f}_3 \mid \bar{G} = x + 344$, *so we may continue. We now update the partial subfield factorization*

$$SF_3 := \{x - \alpha, x - h_1(\alpha), G = x - h_2(\alpha), g_3/G\} =: \{g_1, g_2, g_3, g_4\}.$$

Since we now have 2 linear factors (distinct from $x - \alpha$*), we may try to find a distinct linear factor using Remark 4.6. To do so, consider*

$$x - h_1(h_2(\alpha)) = x - \left(\frac{5}{288}\alpha^7 - \frac{97}{144}\alpha^5 + \frac{95}{18}\alpha^3 - \frac{59}{6}\alpha\right) =: x - h_3(\alpha).$$

We can check that $x - h_3(\alpha)$ *is indeed a distinct linear factor of* f *(not already computed) and we may update the partial subfield factorization to*

$$SF_3 := \{x - \alpha, x - h_1(\alpha), x - h_2(\alpha), x - h_3(\alpha), g_4/(x - h_3(\alpha))\}.$$

For $x - h_3(\alpha)$, *there exists only one factor* $\bar{f}_j$ *such that* $\bar{f}_j = x - h_3(\bar{\alpha}) = x + 475$. *That is, we may skip the factor* $\bar{f}_4 = x + 475$ *(after all, this is the reason of Remark 4.6, to skip expensive LLL calls). For the next factor* $\bar{f}_5 = x + 534$, *we find yet another linear factor* $x - h_4(\alpha)$ *and using Remark* 4.6, *we find* 3 *more linear factors. That is, we have found* 8 *linear factors of the degree-*8 *polynomial* f. *Thus, we have found a subfield factorization (which, in this case, coincides with the factorization of* f *over* $\mathbb{Q}(\alpha)$ *into irreducible factors).*

Theorem 4.8. *Assuming a prime p of suitable size (see Remark 3.46) is found, and assuming the Hensel Lifting accuracy a from [51] is large enough (see Remark 4.9 below), the number of CPU operations executed by Algorithm* `SubfFact` *can be bounded by $\tilde{\mathcal{O}}(rn^7 + rn^5 \log^2 \|f\|)$.*

Proof. Steps 1 and 2 involve factoring f modulo a few primes p until we find a prime that satisfies the conditions from Step 1. Factoring f over $\mathbb{F}_p$ can be executed with $\tilde{\mathcal{O}}(n^2+n \log p)$ operations in $\mathbb{F}_p$ (see [57], Corollary 14.30). Multifactor Hensel lifting takes $\tilde{\mathcal{O}}(n^2(n + \log \|f\|_2))$ CPU operations (see [57], Theorem 15.18). For each i in Step 4 we have one LLL call, costing $\tilde{\mathcal{O}}(n^7+n^5 \log^2 \|f\|_2)$ CPU operations (see [53]), and one `PartialSubfFact` call, which costs $\tilde{\mathcal{O}}(n^3(r + \log \|f\|_2))$ CPU operations according to Lemma 4.5. The theorem follows by omitting $\log p$ factors. □

Remark 4.9. *If the initial value of a is low, our implementation increases a. However, this has little impact on CPU timings or complexity. The highest degree term in the complexity comes from the LLL reduction. To bound the LLL cost, one must bound the vector lengths that can occur during LLL, and the total number of LLL switches. Gradual sublattice reduction (such as [53]) makes those bounds independent of a. More details can also be found in [50], which explains why the highest degree term in the complexity of factoring in $\mathbb{Q}[x]$ depends only on r. To prove an upper bound for a, we need to bound the coefficients of a basis element $\beta_j \in V - L_i$ by multiplying the LLL cut-off bound $n^2\|f\|$ from [51] with the LLL fudge factor $2^{\mathcal{O}(n)}$. Then bound the norm of the resultant of $f(x)$ and $H(x)$ from Remark 4.4, and use the fact that it must be divisible by p^a because $\hat{f}_i$ is a common factor* $\bmod p^a$ *but not* $\bmod p^\infty$ *if $\beta_j \in V - L_i$.*

4.2 CPU Time Comparison

In this last section we give a few timings comparing Algorithm `SubfFact` and factorization algorithms over $\mathbb{Q}(\alpha)$ (recall that both algorithms yield a subfield factorization). We also compare our algorithm `Subfields` with that from [51]. Our algorithm was implemented in the computer algebra system *Magma*, since there exists an implementation of [51] in *Magma* as well. All timings displayed in this and in the next chapter were obtained on an Intel® Core i7-3770 CPU @ 3.40GHz with 32GB of RAM. All examples were computed only once, as timings do not vary significantly at each run.

4.2.1 `SubfFact` vs. Factoring over $\mathbb{Q}(\alpha)$

Algorithm `Subfields` is based on the definition of a subfield factorization of f. As noted before, the irreducible factorization of f over $\mathbb{Q}(\alpha)$ is a subfield factorization. In this section we compare the time necessary to find a subfield factorization of f_i, for several polynomials $f_i \in \mathbb{Z}[x]$, using algorithm `SubfFact`, presented above, with the time necessary to completely factor f_i over $\mathbb{Q}(\alpha)$ in *Magma* and in *Pari/GP*. We also list s, the number of irreducible factors of f_i and r, the number of factors in the subfield factorization obtained using `SubfFact`. The polynomials f_i used to construct this table can be found at `http://www.math.fsu.edu/~jszutkos/MySubfields`. Most of these polynomials can also be found in [51] and they are carefully chosen so as to give interesting Galois groups, as "random" polynomials will not have interesting Galois groups.

f_i	$\deg(f_i)$	s	r	Magma v2.21-3 `SubfFact`	Magma v2.21-3 (Factorization)	Pari/GP v2.9.2 (nffactor)
f_1	32	32	32	0.56s	4.71s	0.46s
f_2	36	24	16	3.76s	4.20s	0.63s
f_3	45	3	3	3.66s	20.01s	94.54s
f_4	48	20	16	21.10s	34.23s	3.30s
f_5	50	26	11	24.08s	20.51s	2.89s
f_6	56	14	6	50.26s	127.34s	26.48s
f_7	60	33	18	107.22s	1,836.80s	38.75s
f_8	60	60	32	117.43s	9,069.22s	40.70s
f_9	64	16	12	101.82s	190.99s	48.82s
f_{10}	72	3	3	77.76s	300.62s	133.54s
f_{11}	72	32	24	175.85s	130.40s	17.23s
f_{12}	75	20	6	542.30s	$> 24h$	518.40s
f_{13}	75	21	9	199.70s	180.06s	114.38s
f_{14}	80	3	3	117.03s	280.18s	136.21s
f_{15}	81	42	28	680.24s	13,661.89s	96.00s
f_{16}	90	24	7	921.53s	$> 24h$	516.14s
f_{17}	96	32	32	555.24s	622.33s	137.23s
f_{18}	96	96	56	2,227.06s	16,352.01s	91.43s

Tabela 4.1: Subfield Factorization vs. Factoring in $\mathbb{Q}(\alpha)[x]$.

In a few cases, factoring f_i over $\mathbb{Q}(\alpha)$ in Magma is faster than `SubfFact`. However, when it is not, using `SubfFact` to find a subfield factorization is usually much faster. Factoring f_i over $\mathbb{Q}(\alpha)$ in *Pari/GP* is usually faster still, except in cases where *Pari/GP* struggles to find an integral basis for K (a step that is not necessary because one can use rational univariate representation instead).

Remark 4.10. *The timing difference between the factorization algorithms from Magma and Pari/GP might be explained by the fact that Magma uses a version of the factorization algorithm from [47], while Pari/GP uses the factorization algorithm from [8]. The reader might be tempted to conclude that it would be best to implement our algorithms in Pari/GP, however, to the best of the author's knowledge, Pari/GP does not currently support multivariate polynomial factorization (which will be needed in the next chapter). Moreover, Pari/GP does not contain an implementation of the algorithm from [51], which we are improving and with which we wish to compare*[3].

Remark 4.11. *In Step 6 of Algorithm* `SubfFact`*, the subfield L_i (to be precise: a subspace V containing L_i, but these are practically always the same) is computed with LLL techniques. Factoring f in Pari/GP is done with LLL techniques as well [8]. We expect the computation of L_i to be faster than factoring f in Pari/GP, because the bound in [51, Theorem 12] used by* `SubfFact` *is very good. The above table shows that, compared with [8], the CPU time saved by this tight bound does not compensate for the fact that Step 6 in Algorithm* `SubfFact` *is done r times. In contrast, the cost of computing one factor with [8] is the same as the cost of computing all irreducible factors. This is why [8] is faster.*

4.2.2 Comparing Algorithms

Finally, we compare the running time of our algorithm `Subfields` (where the subfield factorization is computed using `SubfFact`) and the algorithm from [51] (currently built-in *Magma*). In order to give a better comparison of the running time for both algorithms, we also compute a generator for every subfield (according to Section 3.4.2). To compare the algorithms we need interesting examples (i.e., polynomials defining extensions with several subfields). Hence, these

[3] *Pari/GP does have a command for computing subfields, but it is based on different algorithms.*

polynomials have to be chosen carefully, as random polynomials will likely define an extension with no (non-trivial) subfields. Most of the polynomials used to construct the table below were taken from [51].

As noted before, the main contribution of our work is in the way the intersections of the principal subfields are computed. In the table below, $\hat{r}$ is the number of irreducible factors of f_i in $\mathbb{F}_p[x]$ and r is the number of principal subfields. We also list the number of LLL calls used by algorithm `Subfields`, $d_i = \deg(f_i)$ and m, the total number of subfields of the extension defined by f_i.

ex.	d_i	$\hat{r}$	r	LLL calls	m	m/r	Magma v2.21-3 (built-in)	Magma v2.21-3 (`Subfields`)
f_1	32	32	32	5	374	11.68	11.42s	1.15s
f_2	36	24	16	19	24	1.50	5.14s	3.84s
f_3	48	28	16	26	25	1.56	24.52s	21.21s
f_4	50	26	11	19	12	1.09	26.06s	24.16s
f_5	56	20	6	19	6	1.00	52.29s	50.31s
f_6	60	33	18	31	19	1.05	112.90s	107.53s
f_7	60	60	32	24	59	1.84	205.46s	118.50s
f_8	64	24	12	22	14	1.16	110.89s	101.99s
f_9	64	40	30	35	93	3.10	167.13s	122.24s
f_{10}	64	64	64	6	2,825	44.14	1,084.91s	43.62s
f_{11}	72	40	24	35	42	1.75	219.30s	176.65s
f_{12}	75	20	6	19	6	1.00	516.45s	542.60
f_{13}	75	21	9	19	10	1.11	200.42s	199.85s
f_{14}	80	48	27	37	57	2.11	1,021.22s	685.65s

ex.	d_i	$\hat{r}$	r	LLL calls	m	m/r	Magma v2.21-3 (built-in)	Magma v2.21-3 (`Subfields`)
f_{15}	81	42	28	40	56	2.00	715.70s	681.35s
f_{16}	81	45	25	40	36	1.44	746.33s	716.12s
f_{17}	90	24	7	23	7	1.00	923.74s	921.77s
f_{18}	96	32	32	20	134	4.18	1,159.04s	558.96s
f_{19}	96	96	56	68	208	3.71	4,026.65s	2,239.54s
f_{20}	100	100	57	62	100	1.75	7,902.09s	4,250.39s
f_{21}	128	128	128	7	29,211	228.21	306,591.68s	5,164.75s

Tabela 4.2: Comparison Table - Number Fields.

Notice that when m is close to r (i.e., when there are not many subfields other than the principal subfields and hence, very few intersections to be computed) our algorithm performs similarly as [51]. However, we see a noticeable improvement when m is very large compared to r, since in this case there are a large number of intersections being computed (see examples f_1, f_{10} and f_{21}).

It has to be noted that the time improvement in these particular cases (that is, examples[4] f_1, f_{10} and f_{21}) is not only due to the new intersection algorithm, but also to Remark 4.6. In these cases, f factors linearly over $\mathbb{Q}(\alpha)$ and hence, many of these factors can be found using Remark 4.6, which helps improve CPU timings (since most LLL calls can be skipped). For instance, without attempting to find new linear factors using Remark 4.6, example f_{10} calls LLL 62 times and the total time in this case is 322.13s.

[4]These particular cases are the Swinnerton-Dyer polynomials $S_5(x), S_6(x)$ and $S_7(x)$, respectively. These polynomials have interesting properties and are often the worst case for several algebraic algorithms.

Other examples also benefit from Remark 4.6, but not as much as examples f_1, f_{10} and f_{21}. To check the number of LLL's skipped in each example, we refer the reader to `http://www.math.fsu.edu/~jszutkos/Timings`. More details about these timings (for instance, the time to compute the subfield factorization, the partitions and the intersections given separately) can also be found there. The implementation of our algorithm, as well as the polynomials used in this comparison table, can be found at `http://www.math.fsu.edu/~jszutkos/MySubfields`.

4.3 (Appendix) Primitive Element Probability

Let L/k be a separable field extension and let $\beta_1, \ldots, \beta_m$ be a k-basis of L. Let $T \subseteq k$ finite and let $S = \{\sum a_i\beta_i : a_i \in T\}$. In this section we compute the probability that a random element $s \in S$ is a primitive element of L.

Lemma 4.12. *Let V be a k-vector space with basis $v_1, \ldots, v_m$. Let $W \subseteq V$ be a subspace of dimension d. Let $T \subseteq k$ be a finite set and let $S = \{\sum_{i=1}^m a_i v_i : a_i \in T\}$. Then*

$$|S \cap W| \leq |T|^d.$$

Proof. Let $w_1, \ldots, w_d$ be a basis of W. For every j there exist $c_{i,j} \in k$, $1 \leq i \leq m$, such that

$$w_j = \sum_{i=1}^{m} c_{i,j} v_i. \tag{4.5}$$

Let $w \in W$, then

$$w = \sum_{i=1}^{m} a_i v_i = \sum_{j=1}^{d} b_j w_j, \tag{4.6}$$

for some $a_i \in k$, $1 \leq i \leq m$ and some $b_j \in k$, $1 \leq j \leq d$. Combining equations (4.5) and (4.6), it follows that $a_i = \sum_{j=1}^{d} c_{i,j} b_j$, $1 \leq i \leq n$. That is, we have the following

equation

$$\begin{pmatrix} a_1 \\ \vdots \\ a_m \end{pmatrix} = \begin{pmatrix} c_{1,1} & \cdots & c_{1,d} \\ \vdots & & \vdots \\ c_{m,1} & \cdots & c_{m,d} \end{pmatrix} \begin{pmatrix} b_1 \\ \vdots \\ b_d \end{pmatrix}. \tag{4.7}$$

If C is the $m \times d$ matrix in (4.7), then C has d linearly independent rows. That is, only d of the values a_i suffice to determine w, while the remaining values are dependent. Therefore,

$$|S \cap W| \leq |T|^d.$$

□

Theorem 4.13. *Let L/k be a separable field extension and let $\beta_1, \ldots, \beta_m$ be a k-basis of L. If $T \subseteq k$ is a finite set and $S = \{\sum a_i\beta_i : a_i \in T\}$, then*

$$|\{s \in S : k(s) \subsetneq L\}| \leq (m-1) \cdot |T|^{m/p},$$

where p is the smallest prime that divides m.

Proof. Let $L_1, \ldots, L_r$ be the principal subfields of L/k. Since every subfield of L/k is an intersection of some of the principal subfields of L/k, it suffices to find $|\{s \in S : s \in L_i \subsetneq L, \text{ for some } 1 \leq i \leq r\}|$. The number of principal subfields (not equal to L) is at most $m-1$ and $[L_i : k] \leq m/p$, where p is the smallest prime that divides m. According to Lemma 4.12, $|S \cap L_i| \leq |T|^{m/p}$. Therefore,

$$|\{s \in S : k(s) \subsetneq L\}| \leq (m-1) \cdot |T|^{m/p}.$$

□

Corollary 4.14. *Let L/k be a separable field extension and let $\beta_1, \ldots, \beta_m$ be a k-basis of L. Let $T \subseteq k$ finite and let $S = \{\sum a_i\beta_i : a_i \in T\}$. If s is a random element of S and p is the smallest prime that divides m, then*

$$Prob(k(s) \subsetneq L) \leq (m-1) \cdot |T|^{m(1-p)/p}.$$

4.4 (Appendix) Bounding the coefficients of $H(x)$ and GCD's in $\mathbb{Q}(\alpha)[x]$

The bottleneck of Algorithm `PartialSubfFact` in Section 4.1 is the computation of the gcd's in $\mathbb{Q}(\alpha)[x]$. If p is a prime, $\mathbb{F}_p(\alpha) := \mathbb{F}_p[t]/(f(t))$ is a finite ring. Let $g_1, g_2 \in \mathbb{Q}(\alpha)[x]$. The modular gcd algorithm reconstructs $g := \gcd(g_1, g_2)$ from its images in $\mathbb{F}_p(\alpha)[x]$ for suitable primes. In other words, there are mainly four steps to be carried out (see [52])

1) Compute $g_1 \bmod p$, $g_2 \bmod p$, for several suitable primes p.

2) Compute $\gcd(g_1 \bmod p, g_2 \bmod p)$, for each prime p.

3) Chinese remaider the polynomials in 2) and use rational reconstruction to find a polynomial $g \in \mathbb{Q}(\alpha)[x]$.

4) Trial Division: check if $g|g_1$ and $g|g_2$.

The number of primes needed depends on the coefficient size of g. But the (bound for) coefficient size of $f'(\alpha)g \in \mathbb{Z}[\alpha][x]$ is much better than that of $g \in \mathbb{Q}(\alpha)[x]$. Hence, to get a good complexity/run time we choose to reconstruct $f'(\alpha)g$ from its modular images instead of g. Furthermore, if we have some information about g (for instance, its degree), then step 4 can be skipped.

In our case, we need to compute $\gcd(H, g_j)$, where $H = f'(x)\tilde{h}(\alpha) - f'(\alpha)\tilde{h}(x) \in \mathbb{Z}[\alpha][x]$ is as in Remark 4.4 and g_j is a factor of f over $\mathbb{Q}(\alpha)$. In what follows we compute a bound for the coefficients of H and a bound for the coefficients of a factor of H. Let $\alpha_1, \ldots, \alpha_n$ be the complex roots of f and let σ_i be the i-th embedding of $\mathbb{Q}(\alpha)$ into $\mathbb{C}$ such that $\sigma_i(\alpha) = \alpha_i$. For $\beta \in \mathbb{Q}(\alpha)$, we define the T-norm of the element β to be

$$T(\beta) := \sum_{i=1}^{n} |\sigma_i(\beta)|^2.$$

If $\beta \in \mathbb{Z}[\alpha]$, then there exists $b_0, \ldots, b_{n-1} \in \mathbb{Z}$ such that $\beta = \sum b_i \alpha^i$ and one can also define

$$\|\beta\|_2 := \|(b_0, \ldots, b_{n-1})\|_2.$$

The relation between $\|\cdot\|_2$ and the T-norm is the following

Lemma 4.15 (Lemma 18 of [51]). *Let $\beta \in \mathbb{Q}(\alpha)$. If $f'(\alpha)\beta = \sum b_i \alpha^i$, with $b_i \in \mathbb{Z}$, then*

$$\|f'(\alpha)\beta\|_2 = \|(b_0, \ldots, b_{n-1})\|_2 \leq n^{3/2}\|f\|_2\sqrt{T(\beta)}.$$

Let $\beta_1, \ldots, \beta_m$ be generators of L as a $\mathbb{Q}$-vector space, where

$$f'(\alpha)\beta_i = \sum b_{i,j}\alpha^j$$

with $b_{i,j} \in \mathbb{Z}$ and $\|(b_{i,0}, \ldots, b_{i,n-1})\|_2 \leq n^2\|f\|_2$ (See [51]). Let β be a random T-combination of $\beta_1, \ldots, \beta_m$. That is,

$$\beta = \sum t_i\beta_i \in \frac{1}{f'(\alpha)}\mathbb{Z}[\alpha]_{<n},$$

with $t_i \in T$. Let $h_0, \ldots, h_{n-1} \in \mathbb{Z}$ be such that $f'(\alpha)\beta = \sum h_i\alpha^i$ and let $\tilde{h}(x) = \sum h_i x^i \in \mathbb{Z}[x]$. By expanding the summations, one can show that

$$h_i = \sum t_j b_{j,i}$$

and hence

$$|h_i| \leq nT_B \max\{\|\beta_j\|_\infty\} \leq n^3 T_B\|f\|_2, \tag{4.8}$$

where T_B is a bound for the elements of T. Let us now bound the integer coefficients of $H(x) := f'(x)\tilde{h}(\alpha) - f'(\alpha)\tilde{h}(x) \in \mathbb{Z}[\alpha][x]$. If $c \in \mathbb{Z}$ is a coefficient of $f'(x)$, then $|c| \leq n\|f\|_\infty$. Now if $c\tilde{h}(\alpha) \in \mathbb{Z}[\alpha]$ is a coefficient of $f'(x)\tilde{h}(\alpha)$, then

$$\|c\tilde{h}(\alpha)\|_2 = \|\sum ch_i\alpha^i\|_2 \leq |c|\sqrt{\sum |h_i|^2} = |c|\sqrt{n}\max\{|h_i|\} \leq n^{9/2}T_B\|f\|_\infty\|f\|_2.$$

Likewise, if $h_i f'(\alpha)$ is a coefficient of $f'(\alpha)\tilde{h}(x)$, then one can show that

$$\|h_i f'(\alpha)\|_2 \leq n^{9/2}T_B\|f\|_2\|f\|_\infty.$$

This shows the following theorem.

Theorem 4.16. *If $c = \sum c_i \alpha^i \in \mathbb{Z}[\alpha]$ is a coefficient of $H(x)$, then $\|(c_0, \ldots, c_{n-1})\|_2 \leq 2n^{9/2} T_B \|f\|_2 \|f\|_\infty$.*

Now let us bound the coefficients of any factor of f and $H(x)$. We use the following result (Landau-Mignotte bound).

Theorem 4.17 (Theorem 6.32 of [57])**.** *If $h = \sum_{i=0}^{m} h_i x^i \in \mathbb{C}[x]$ divides $f = \sum_{i=0}^{n} f_i x^i \in \mathbb{C}[x]$, then $\|h\|_2 \leq \|h\|_1 \leq 2^m M(h) \leq \left|\frac{h_m}{f_n}\right| 2^m \|f\|_2$.*

Lemma 4.18. *Let $g \in \mathbb{Q}(\alpha)[x]$ be a factor of f and let c be a coefficient of g. Furthermore, let $b_0, \ldots, b_{n-1} \in \mathbb{Z}$ such that $f'(\alpha)c = \sum b_i \alpha^i \in \mathbb{Z}[\alpha]$. Then*

$$\|(b_0, \ldots, b_{n-1})\|_\infty \leq n 4^n \|f\|_2^2.$$

Proof. Let $c^{(1)}, \ldots, c^{(n)}$ be the evaluation of c in the complex roots $\alpha_1, \ldots, \alpha_n$ of f, respectively. Since g is a factor of f, the Landau-Mignotte bound [36] tells us that $|c^{(i)}| \leq 2^n \|f\|_2$. As shown in [54], Lemma 4.2, we can write $f'(\alpha)c = P(\alpha)$, where

$$P(x) = \sum_{i=1}^{n} c^{(i)} \frac{f(x)}{x - \alpha_i} \in \mathbb{Z}[x].$$

Hence

$$\|P(x)\|_\infty \leq \|P(x)\|_2 = \left\| \sum_{i=1}^{n} c^{(i)} \frac{f(x)}{x - \alpha_i} \right\|_2 \leq \sum_{i=1}^{n} |c^{(i)}| \left\| \frac{f(x)}{x - \alpha_i} \right\|_2.$$

Again using the Landau-Mignotte bound, we get $\|P(x)\|_\infty \leq n 4^n \|f\|_2^2$. □

To bound the integer coefficients of any factor of H we need some results.

Lemma 4.19. *Let c be a coefficient of $H^{\sigma_i} \in \mathbb{C}[x]$, the image of H under the embedding σ_i. Then $|c| \leq 2T_B n^5 \|f\|_2 \|f\|_\infty B_r^{n-1}$, where $1 \leq B_r$ is a bound for the complex roots of f.*

Proof. First of all, notice that

$$H^{\sigma_i} = \sigma_i(H(x)) = f'(x)\tilde{h}(\sigma_i(\alpha)) - f'(\sigma_i(\alpha))\tilde{h}(x) =$$

$$= f'(x)\tilde{h}(\alpha_i) - f'(\alpha_i)\tilde{h}(x).$$

Hence, if $c\tilde{h}(\alpha_i)$ is a coefficient of $f'(x)\tilde{h}(\alpha_i)$, then

$$|c\tilde{h}(\alpha_i)| = |c||\tilde{h}(\alpha_i)| \leq n\|f\|_\infty |\sum h_j \alpha_i^j| \leq n\|f\|_\infty \sum |h_j||\alpha_i|^j$$

If $1 \leq B_r$ is a bound for $|\alpha_i|$, for any i, and by Equation (4.8), it follows that

$$|c\tilde{h}(\alpha_i)| \leq T_B n^5 \|f\|_2 \|f\|_\infty B_r^{n-1}.$$

Likewise, if $h_j f'(\alpha_i)$ is a coefficient of $f'(\alpha_i)\tilde{h}(x)$, one can show that

$$|h_j f'(\alpha_i)| \leq T_B n^5 \|f\|_2 \|f\|_\infty B_r^{n-1}.$$

The result follows by the triangle inequality. □

Let us now bound the integer coefficients of any factor of H.

Theorem 4.20. *Let $G = \sum g_i x^i$ be a monic factor of $\frac{H}{lc(H)}$ and let $g_i = \sum \tilde{g}_i \frac{\alpha^i}{f'(\alpha)}$, with $\tilde{g}_i \in \mathbb{Z}$. Then $f'(\alpha) lc(H) g_i \in \mathbb{Z}[\alpha]$ and*

$$\|f'(\alpha) lc(H) g_i\|_2 \leq n^{7.5} 2 T_B \|f\|_2^2 \|f\|_\infty (2(1 + \|f\|_\infty))^n.$$

Proof. Notice that *lcoeff*$(H) \in \mathbb{Z}[\alpha]$ and that $f'(\alpha) g_i = \sum \tilde{g}_i \alpha^i \in \mathbb{Z}[\alpha]$. Hence, $f'(\alpha)\mathrm{lc}(H) g_i \in \mathbb{Z}[\alpha]$. By Lemma 4.15, it follows that

$$\|f'(\alpha)\mathrm{lc}(H) g_i\|_2 \leq n^{3/2} \|f\|_2 \sqrt{T(\mathrm{lc}(H) g_i)}. \tag{4.9}$$

Furthermore,

$$T(\mathrm{lc}(H) g_i) = \sum_\sigma |\mathrm{lc}(H^\sigma) g_i^\sigma|^2 = \sum_\sigma |\mathrm{lc}(H^\sigma)|^2 |g_i^\sigma|^2,$$

where g_i^σ is the i-th complex coefficient of the monic factor G^σ of H^σ. By Theorem 4.17, it follows that

$$|g_i^\sigma| \leq \left| \frac{1}{\mathrm{lc}(H^\sigma)} \right| 2^n \|H^\sigma\|_2.$$

Hence,

$$T(\mathrm{lc}(H)g_i) \leq \sum_{\sigma} |\mathrm{lc}(H^\sigma)|^2 \left(\left| \frac{1}{\mathrm{lc}(H^\sigma)} \right| 2^n \|H^\sigma\|_2 \right)^2 = \sum_{\sigma} \left(2^n \|H^\sigma\|_2\right)^2$$

Using Lemma 4.19, it follows that

$$T(\mathrm{lc}(H)g_i) \leq \sum_{\sigma} \left(2^n \|H^\sigma\|_2\right)^2 \leq n \left(\sqrt{n} 2^{n+1} T_B n^5 \|f\|_2 \|f\|_\infty B_r^{n-1}\right)^2 \tag{4.10}$$

By Equations (4.9) and (4.10), it follows that

$$\|f'(\alpha)\mathrm{lc}(H)g_i\|_2 \leq 2n^{7.5} T_B \|f\|_2^2 \|f\|_\infty (2B_r)^n.$$

The result follows by applying the Cauchy bound for the roots of a monic polynomial $B_r \leq 1 + \|f\|_\infty$. □

Let us now determine the cost (in CPU operations) for computing

$$f'(\alpha)\gcd(H, f'(\alpha)g_j) \in \mathbb{Z}[\alpha][x].$$

Let B be a bound for the integer coefficients of $f'(\alpha)\gcd(H, f'(\alpha)g_j)$ (Lemma 4.18).

- Step 1) First of all, we compute the images of H and $f'(\alpha)g_j$ in $\mathbb{F}_p(\alpha)[x]$, which can be done with $\mathcal{O}(n^2)$ integer reductions modulo several primes p. The number of primes is $\mathcal{O}(\log B) = \tilde{\mathcal{O}}(n + \log \|f\|_2)$. According to [57, Theorem 10.24], the complexity of this step is bounded by $\tilde{\mathcal{O}}(n^2(n + \log \|f\|_2))$.

- Step 2) Secondly, we have to compute one gcd in $\mathbb{F}_p(\alpha)[x]$, for $\mathcal{O}(\log B)$ primes p. Using the Extended Euclidean Algorithm (see [57, Corollary 11.6]), one gcd in $\mathbb{F}_p(\alpha)[x]$ can be computed with $\tilde{\mathcal{O}}(n)$ operations in $\mathbb{F}_p(\alpha)$ or $\tilde{\mathcal{O}}(n^2)$ operations in $\mathbb{F}_p$. Hence, this step can be bounded by $\tilde{\mathcal{O}}(n^2(n + \log \|f\|_2))$.

- Step 3) In this step we need to find a polynomial $f'(\alpha)G \in \mathbb{Z}[\alpha][x]$ whose images modulo several primes are given in Step 2. For this we use the Chinese Remainder Algorithm (CRA). There are $n(d+1)$ integers to be reconstructed, where $d = \deg(\gcd(H, g_j))$, and each CRA call costs $\tilde{\mathcal{O}}(\log P)$, where $P = \prod p$ (see [57, Theorem 10.25]). Since $P = \mathcal{O}(B)$, the total cost of this step is $\tilde{\mathcal{O}}(n^2(n + \log \|f\|_2))$.

- Step 4) Instead of computing the division H/G (and g_j/G, whose complexity is hard to bound), we can substitute this trial division by reconstruction from modular images followed by a trial multiplication. That is, we can compute the images of H and G modulo several primes p, compute H/G modulo p and then reconstruct $f'(\alpha)\mathrm{lc}(H)(\frac{H}{G}) \in \mathbb{Z}[\alpha][x]$ and verify that $f'(\alpha)\mathrm{lc}(H)(\frac{H}{G}) \cdot G = f'(\alpha)H$. The cost is similar to steps 1), 2) and 3) above, the only difference is the number of primes needed (since what we want to reconstruct is a factor of H, the bound B is given by Lemma 4.20) and the trial multiplication at the end (which can be executed with $\tilde{\mathcal{O}}(n^3 \log \|f\|_2)$ CPU operations). Hence, this step has complexity $\tilde{\mathcal{O}}(n^3 \log \|f\|_2)$.

5 RATIONAL FUNCTION DECOMPOSITIONS

Given rational functions (resp. polynomials) g, h, the composition $g \circ h$ of g and h is again a rational function (resp. polynomial). The inverse problem, i.e., given f, find g, h such that $f = g \circ h$, is called the *(univariate) rational functional decomposition* problem. Finding a decomposition of a rational function $f \in K(t)$ has been studied by several authors. We highlight the work of Zippel [60], which presents the first polynomial time algorithm that finds (if it exists) a nontrivial decomposition of f. In [4], Alonso *et al.* give an exponential time algorithm that computes all decompositions of f by generalizing the ideas for the polynomial case from Barton and Zippel [7]. More recently, Ayad & Fleischmann [6] presented improvements on [4], though the complexity is still exponential on the degree of f.

The particular case of polynomial decomposition has long been studied. As far as the author's knowledge goes, the first work on polynomial decomposition is from Ritt [39], who presented a strong structural property of polynomial decompositions over the complex numbers. Barton and Zippel [7] presented two (exponential time) algorithms for finding the decompositions of a polynomial over a field of characteristic zero. Some simplifications are suggested by Alagar and Thanh [2] and Alonso *et al.* [4]. Kozen and Landau [29] give the first polynomial time algorithm, which works over any commutative ring containing an inverse of $\deg(g)$. van zur Gathen [55, 56] further improves the work of [29]. More recently, Blankertz [10], following the ideas of Zippel [60], gives a polynomial time algorithm that finds all *minimal decompositions* of f, with no restrictions on $\deg(g)$.

Rational Functional Decomposition (of either a rational function or a polynomial) is closely related to the subfield lattice of the field extension $K(t)/K(f(t))$ (see Theorem 5.8 below). However, in general, the number of subfields is not poly-

nomially bounded and algorithms for finding all *complete decompositions*, such as Ayad & Fleischmann [6], can suffer from its combinatorial nature. In this last chapter, we use the algorithms presented in Chapter 3 to find the subfield lattice of $K(t)/K(f(t))$ and hence, all *complete decompositions* of f.

5.1 Basic Definitions

In this section we recall some basic definitions regarding rational function fields and rational function decompositions. Most of the results presented in this section can be found in [4], [6] or in the references therein. Let K be an arbitrary field and let $K(t)$ be the rational function field over K. As usual, let $\mathbb{S} = K(t)\backslash K$ be the set of non-constant rational functions and let $f = f_n/f_d \in \mathbb{S}$ be a rational function with $f_n, f_d \in K[t]$ coprime. The *degree* of f is defined as $\max\{\deg(f_n), \deg(f_d)\}$ and denoted by $\deg(f)$. The set $\mathbb{S}$, together with composition, is equipped with the structure of a monoid. The K-automorphisms of $K(t)$ are the fractional transformations $u = (ax + b)/(cx + d)$ such that $ad - bc \neq 0$. The group of automorphisms is isomorphic to the group $PGL_2(K)$ and also to the group of units of $\mathbb{S}$ under composition. An element $f \in K(t)$ is said to be *indecomposable* if 1) f is not a unit and 2) $f = g \circ h$ implies g or h is a unit. Otherwise, f is called *decomposable*.

Definition 5.1. *A decomposition of f is a list of rational functions $(g_m, \ldots, g_1)$ such that $f = g_m \circ \cdots \circ g_1$. A complete decomposition is a decomposition $(g_m, \ldots, g_1)$ where each g_i is indecomposable. Moreover, two decompositions $(g_m, \ldots, g_1)$ and $(\tilde{g}_n, \ldots, \tilde{g}_1)$ of f are equivalent if $m = n$ and there are rational functions $u_1, \ldots, u_{m-1}$ of degree 1 such that $g_m = \tilde{g}_m \circ u_{m-1}^{-1}$, $g_1 = u_1 \circ \tilde{g}_1$ and*

$$g_i = u_i \circ \tilde{g}_i \circ u_{i-1}^{-1}, \quad 1 < i < m,$$

If f is decomposable with $f = g \circ h$, then h (resp. g) is called the *right component* (resp. *left component*) of the decomposition $g \circ h$. Furthermore, a decomposition $f = g \circ h$ is *minimal* if h is indecomposable.

Example 5.2. *Consider the rational function*

$$f = \frac{t^3 + t^2 + t}{(t^2+1)^2} \in \mathbb{Q}(t).$$

This rational function admits a decomposition $f = g \circ h$, where $g = t^2 + t$ and $h = t/(t^2+1)$. Since h is indecomposable, this is a minimal decomposition. Moreover, g is also indecomposable and hence, (g, h) is a complete decomposition of f.

It is well known by Lüroth's Theorem that if $K \subsetneq L \subseteq K(t)$, then there exists $h \in \mathbb{S}$ such that $L = K(h(t))$ (a proof can be found in van Der Waerden [48]). The rational function h is not unique however, $K(h(t)) = K(h'(t))$, if and only if, there exists a unit $u \in \mathbb{S}$ such that $h' = u \circ h$. As in Ayad & Fleischmann [6], we define the *normal form* of a rational function $f \in \mathbb{S}$.

Definition 5.3. *A rational function $f = p/q \in \mathbb{S}$ is in normal form or normalized if $p, q \in K[t]$ are monic, coprime, $p(0) = 0$ and either $deg(p) > \deg(q)$ or $m := \deg(p) < \deg(q) =: n$ and $q = t^n + q_{n-1}t^{n-1} + \cdots + q_0$, with $q_m = 0$.*

The normal form of a rational function f can be computed via a certain fractional transformation, i.e., there exists a rational function u of degree 1 such that $u \circ f$ is in normal form. Moreover, given $f \in \mathbb{S}$, there exists a unique normalized $\hat{f} \in \mathbb{S}$ such that $K(f(t)) = K(\hat{f}(t))$ (see [6, Proposition 2.1]). Hence, if $\mathcal{N}_K$ is the set of all normalized rational functions over K, then there exists a bijection between $\mathcal{N}_K$ and the set of fields L such that $K \subsetneq L \subseteq K(t)$.

Remark 5.4. *In particular, there is a bijection between normalized rational functions $h \in \mathbb{S}$ such that $f = g \circ h$, for some $g \in \mathbb{S}$, and the fields $L = K(h(t))$ such that $K(f(t)) \subseteq L \subseteq K(t)$. That is, by finding the subfield lattice of $K(t)/K(f(t))$, we obtain all decompositions of $f \in K(t)$.*

Definition 5.5. *For a rational function $g = g_n/g_d \in \mathbb{S}$, with $g_n, g_d \in K[t]$ coprime, define $\nabla_g(x,t) = \nabla_{g_n,g_d}(x,t) := g_n(x)g_d(t) - g_n(t)g_d(x) \in K[x,t]$ and*

$$\Phi_g(x) := g_n(x) - g(t)g_d(x) \in K(g(t))[x].$$

A bivariate polynomial $a(x,t) \in K[x,t]$ is called near-separate if $a(x,t) = \nabla_{g_n,g_d}(x,t)$, for $g_n, g_d \in K[t]$ coprime polynomials.

Remark 5.6. *By Gauss' Lemma, one can show that the polynomial $\Phi_f(x) \in K(f(t))[x]$ is irreducible. If $\Phi_f(x) \in K(f(t))[x]$ is monic, then $\Phi_f(x)$ is the minimal polynomial of t over $K(f(t))$. Otherwise, let $\tilde{f} = \tilde{f}_n/\tilde{f}_d$ be the normalization of f. By Definition 5.3, either $\deg(\Phi_{\tilde{f}}(x)) = \deg(\tilde{f}_n) > \deg(\tilde{f}_d)$ or $\deg(\Phi_{\tilde{f}}(x)) = \deg(\tilde{f}_d) > \deg(\tilde{f}_n)$. In the latter case, $\Phi_{\tilde{f}}(x)$ is not monic; however,*

$$\Phi_{1/\tilde{f}}(x) = \tilde{f}_d(x) - 1/\tilde{f}(t)\tilde{f}_n(x) \in K(1/\tilde{f})[x]$$

is monic, irreducible and vanishes at $x = t$. Since $1/\tilde{f} = u \circ f$, where u is a unit, it follows that $K(1/\tilde{f}) = K(f(t))$ and hence, $\Phi_{1/\tilde{f}}(x)$ is the minimal polynomial of t over $K(f(t))$. Conversely, if $\tilde{f} = u \circ f$, where u is a unit, and $(g_m, \ldots, g_1)$ is a complete decomposition of $\tilde{f}$, then $(u^{-1} \circ g_m, g_{m-1}, \ldots, g_1)$ is a complete decomposition of f. Therefore, we can assume, without loss of generality, that $f \in K(t)$ is such that $\Phi_f(x) \in K(f(t))[x]$ is the minimal polynomial of t over $K(f(t))$.

Remark 5.7. *Let $f \in K(t)$ of degree n and let $G_1, \ldots, G_r$ be the irreducible factors of $\nabla_f(x,t) \in K[x,t]$. Let $m_1, \ldots, m_r \in K[t]$ be the leading coefficients of $G_1, \ldots, G_r$ w.r.t. x. Then $m_1 \cdots m_r = f_d(t)$ and $F_i := G_i/m_i \in K(t)[x]$ are monic, irreducible and $\nabla_f/f_d(t) = \Phi_f(x) = F_1 \cdots F_r$.*

In particular, if the exponents of t in G_i are bounded by d_i, then $\sum d_i = n$. The following theorem is the key result behind all rational (and also polynomial) function decomposition algorithms based on near-separate polynomials, such as [4] and [6] (see also [7] for the polynomial case).

Theorem 5.8 ([4], Proposition 3.1)**.** *Let $f, h \in \mathbb{S}$ be rational functions. The following are equivalent:*

a) $K(f(t)) \subseteq K(h(t)) \subseteq K(t)$.

b) $f = g \circ h$, *for some* $g \in \mathbb{S}$.

c) $\nabla_h(x,t)$ *divides* $\nabla_f(x,t)$ *in* $K[x,t]$.

d) $\Phi_h(x)$ *divides* $\Phi_f(x)$ *in* $K(t)[x]$.

If $G_1, \ldots, G_r$ are the irreducible factors of ∇_f in $K[x,t]$, then the product of any subset of $\{G_1, \ldots, G_r\}$, which is a near-separate multiple of $x - t$, yields a right component h and hence, a decomposition $f = g \circ h$. Many authors use this approach to compute all decompositions of f: factor ∇_f and search for near-separate factors (see [4, 6, 7]). However, this approach leads to exponential time algorithms due to the number of factors we have to consider.

In the following sections we will use the algorithms developed so far, based on principal subfields and fast subfield-intersection techniques, to compute the subfield lattice of $K(t)/K(f(t))$. As previously mentioned, this yields all complete decompositions of f. The general algorithms given in Chapter 3 apply for $K(t)/K(f(t))$. In this Chapter we will explicitly re-state the algorithms in Chapter 3 to this particular case. We do this in order to find simplifications in our computations.

5.2 Principal Subfields of $K(t)/K(f(t))$

In this section we describe the principal subfields of the field extension defined by the rational function $f(t) \in K(t)$, given by $K(t)/K(f(t))$. We start by making some remarks on the separability of Φ_f.

Remark 5.9. *By Gauss' Lemma, one can show that Φ_f is irreducible in $K(f(t))[x]$. Thus, if K has characteristic 0, then Φ_f is also separable. On the other hand, if K has characteristic $p > 0$, then $\Phi_f(x)$ might not be separable. This happens only if $\Phi'_f(x) = 0$ and hence*

$$\Phi_f(x) = g(x^{p^k}) = g \circ x^{p^k}, \tag{5.1}$$

for some $k \geq 1$ and $g \in K(f(t))[x]$ separable. Since $\Phi_f(x) = f_n(x) - f(t)f_d(x)$, this means that $f_n(x) = \tilde{f}_n(x^{p^k})$ and $f_d(x) = \tilde{f}_d(x^{p^k})$, for some $\tilde{f}_n, \tilde{f}_d \in K[x]$, and hence,

$$f = \tilde{f} \circ t^{p^k} \in K(t),$$

where $\tilde{f} := \tilde{f}_n/\tilde{f}_d \in K(t)$ and $\Phi_{\tilde{f}}$ is separable. Thus, if one is only interested in the decompositions of f, it suffices to find the decompositions of $\tilde{f}$. For this reason, we may always assume that $f \in K(t)$ is such that $\Phi_f(x)$ is separable (see also [22], Section 4.6).

Definition 5.10. *Let $F_1, \ldots, F_r$ be the monic irreducible factors of $\Phi_f(x)$ over $K(t)$. For $j = 1, \ldots, r$, define the set*

$$L_j := \{g(t) \in K(t) \ : \ F_j \mid \Phi_g\}. \tag{5.2}$$

The irreducible factors of $\Phi_f(x) \in K(t)[x]$ can be computed by computing the irreducible factors of the bivariate polynomial $\nabla_f(x,t) \in K[x,t]$. Since $\Phi_f(x)$ is the minimal polynomial of t over $K(f(t))$, the irreducible factorization of $\Phi_f(x)$ is a subfield factorization of $K(t)/K(f(t))$ and the subsets L_j above are the principal subfields of $K(t)/K(f(t))$, as we shall prove below. If we assume that $F_1 = x - t$, then $L_1 = K(t)$. The next two results show that the sets defined in Equation (5.2) are the principal subfields of $K(t)/K(f(t))$.

Theorem 5.11. *Let $F_1, \ldots, F_r$ be the monic irreducible factors of $\Phi_f(x)$ over $K(t)$. Then the sets $L_1, \ldots, L_r$ are subfields of $K(t)/K(f(t))$.*

Proof. We show that L_j is closed under multiplications and taking inverses. The remaining properties can be shown in the same fashion. Let $g(t) = g_n(t)/g_d(t)$ and $h(t) = h_n(t)/h_d(t)$ be elements of L_j. By definition,

$$F_j \mid \Phi_g \text{ and } F_j \mid \Phi_h. \tag{5.3}$$

Now $g(t)h(t) \in L_j$ if and only if, $F_j \mid \Phi_{gh}$. By a simple manipulation, one can show that

$$\Phi_{gh} = g_n(x)\Phi_h + h(t)h_d(x)\Phi_g. \tag{5.4}$$

Therefore, by Equation (5.3), it follows that $F_j \mid \Phi_{gh}$ and hence, $g(t)h(t) \in L_j$. To show that the inverse of $g(t)$ is in L_j, notice that

$$F_j \mid \Phi_g \text{ if and only if } F_j \mid \Phi_{1/g}, \tag{5.5}$$

since $\Phi_g = -g(t)\Phi_{1/g}$ as polynomials in $K(t)[x]$. Therefore, $1/g(t) \in L_j$. □

Finally, we show that the subfields $L_1, \ldots, L_r$ defined above are the principal subfields of $K(t)/K(f(t))$.

Theorem 5.12. *The set of subfields $\{L_1, \ldots, L_r\}$ of $K(t)/K(f(t))$, where L_j is defined as in* (5.2)*, for $1 \leq j \leq r$, is the set of principal subfields of $K(t)/K(f(t))$.*

Proof. Given a subfield L of $K(t)/K(f(t))$, by Lüroth's Theorem, there exists a rational function $h(t) \in K(t)$ such that $L = K(h(t))$ and therefore, $f = g \circ h$, for some $g \in K(t)$. By Theorem 5.8 it follows that $\Phi_h \mid \Phi_f$. Therefore, there exists a set $I_L \subseteq \{1, \ldots, r\}$ such that $\Phi_h = \prod_{i \in I_L} F_i$. We shall prove that

$$L = \{g(t) \in K(t) \ : \ \Phi_h \mid \Phi_g\} = \bigcap_{i \in I_L} L_i. \tag{5.6}$$

Let $g(t) \in K(t)$. Then $g(t) \in L = K(h(t))$ if and only if $g(t) = \tilde{g} \circ h(t)$, for some $\tilde{g}(t) \in K(t)$, if and only if $\Phi_h \mid \Phi_g$, by Theorem 5.8. For the second equality, suppose that $g(t) \in \cap_{i \in I_L} L_i$. Then $F_i \mid \Phi_g$, for every $i \in I_L$. Since we are assuming

Φ_f to be separable (see Remark 5.9), it follows that $\Phi_h = \prod_{i \in I_L} F_i \mid \Phi_g$. Conversely, if $\Phi_h \mid \Phi_g$, then $F_i \mid \Phi_g$, for every $i \in I_L$, that is, $g(t) \in L_i$, for every $i \in I_L$ and hence, $g(t) \in \cap_{i \in I_L} L_i$. □

5.3 The Partition of a Principal Subfield of $K(t)/K(f(t))$

In this section we present two algorithms for computing the partitions $P_1, \ldots, P_r$ of the principal subfields $L_1, \ldots, L_r$ of $K(t)/K(f(t))$, one deterministic and one probabilistic. These algorithms are based on the general algorithms given in Chapter 3, applied to the special case of $K(t)/K(f(t))$.

Recall that to find the partition of L_i, it is enough to find a basis of the vectors $(e_1, \ldots, e_r) \in \{0,1\}^r$ such that $\prod_{j=1}^r F_j^{e_j} \in L_i[x]$. Moreover, let $c_1, \ldots, c_{2n} \in K(f(t))$ be distinct elements and let $h_{j,k}(t) := F_j'(c_k)/F_j(c_k) \in K(t)$. If $(e_1, \ldots, e_r) \in \{0,1\}^r$ is such that $\sum_{j=1}^r e_j h_{j,k}(t) \in L_i$, for $k = 1, \ldots, 2n$, then, by Lemmas 3.34 and 3.35, it follows that

$$\prod_{j=1}^{r} F_j^{e_j} \in L_i[x].$$

In the number field case, $f_j'(c_k)/f_j(c_k) \in \mathbb{Q}(\alpha)$ can be rewriten as $g(\alpha)$, for some $g(x) \in \mathbb{Q}[x]_{<n}$. In the function field case, however, this is a bit different.

5.3.1 Deterministic Algorithm

Let us consider $e_1, \ldots, e_r$ above as variables. To show that $\sum e_j h_{j,k}(t) \in L_i$ we need an expression of the form $a(t)/b(t)$, where $a, b \in K[t]$ are coprime. For a rational function $g(t) = \frac{g_n(t)}{g_d(t)} \in K(t)$, with $g_n(t), g_d(t) \in K[t]$ coprime, define functions $\mathrm{Num}, \mathrm{Den} : K(t) \to K[t]$, with $\mathrm{Num}(g) = g_n(t)$ and $\mathrm{Den}(g) = g_d(t)$.

Hence

$$\sum_{j=1}^{r} e_j \frac{F_j'(c_k)}{F_j(c_k)} = \sum_{j=1}^{r} e_j h_{j,k}(t) = \sum_{j=1}^{r} e_j \frac{n_{j,k}(t)}{d_{j,k}(t)},$$

where $n_{j,k}(t) := \mathrm{Num}\left(h_{j,k}(t)\right) \in K[t]$ and $d_{j,k}(t) := \mathrm{Den}\left(h_{j,k}(t)\right) \in K[t]$, for $1 \leq j \leq r$. Furthermore, let $l_k(t) \in K[t]$ be the least common multiple of $d_{1,k}(t), \ldots, d_{r,k}(t) \in K[t]$. Hence

$$\sum_{j=1}^{r} e_j h_{j,k}(t) = \sum_{j=1}^{r} e_j \frac{n_{j,k}(t)}{d_{j,k}(t)} = \frac{\sum_{j=1}^{r} e_j p_{j,k}(t)}{l_k(t)}, \tag{5.7}$$

where $p_{j,k}(t) := l_k(t)\frac{n_{j,k}(t)}{d_{j,k}(t)} \in K[t]$. Hence, $\sum_{j=1}^{r} e_j h_{j,k}(t) \in L_i$ if, and only if (see Definition 5.10)

$$\left[\sum_{j=1}^{r} e_j p_{j,k}(x) - \frac{\sum_{j=1}^{r} e_j p_{j,k}(t)}{l_k(t)} l_k(x)\right] \bmod F_i = 0, \tag{5.8}$$

where $a \bmod b$ is the remainder of division of a by b. By manipulating Equation (5.8) we have

$$\sum_{j=1}^{r} e_j \left[\left(p_{j,k}(x) - h_{j,k}(t) l_k(x)\right) \bmod F_i\right] = 0. \tag{5.9}$$

Hence, if $(e_1, \ldots, e_r) \in \{0,1\}^r$ is a solution of Equation (5.9), for $k = 1, \ldots, 2n$, then it follows that $\prod_{j=1}^{r} F_j^{e_j} \in L_i[x]$. We will now explicitly present the system given by Equation (5.9). This will help us analyse the complexity of the algorithm. Let

$$q_{j,k}(x) := p_{j,k}(x) - h_{j,k}(t) l_k(x) \in K(t)[x]. \tag{5.10}$$

Notice that $\deg_x(q_{j,k}) \leq dn$, where $d = \deg_t(c_k)$. Furthermore, let

$$r_{i,j,k}(x) := q_{j,k}(x) \bmod F_i \in K(t)[x]. \tag{5.11}$$

Let $m_j(t) \in K[t]$ be the monic lowest degree polynomial such that $m_j(t) r_{i,j,k} \in K[t][x]$ and let $l \in K[t]$ be the least common multiple of $m_1(t), \ldots, m_r(t)$. Hence

$$l \sum_{j=1}^{r} e_j r_{i,j,k} = \sum_{j=1}^{r} e_j \hat{r}_{i,j,k} \in K[t][x],$$

where $\hat{r}_{i,j,k} = lr_{i,j,k} \in K[t][x]$. Notice that Equation (5.9) holds if and only if, $\sum_{j=1}^{r} e_j \hat{r}_{i,j,k} = 0$. Next, let us write

$$\hat{r}_{i,j,k} = \sum_{d=0}^{d_i-1} \sum_{s=0}^{S} c_j(s,d,k) t^s x^d, \quad \text{where } c_j(s,d,k) \in K,$$

where d_i is the degree of F_i and $S \geq 0$ is a bound for the t-exponents. Therefore,

$$\sum_{j=1}^{r} e_j \hat{r}_{i,j,k} = \sum_{d=0}^{d_i-1} \sum_{s=0}^{S} \left(\sum_{j=1}^{r} e_j c_j(s,d,k) \right) t^s x^d$$

and hence, the system in $e_1, \ldots, e_r$ from Equation (5.9) is given by

$$\mathcal{S}_i := \left\{ \begin{array}{ll} & d = 0, \ldots, d_i - 1, \\ \displaystyle\sum_{j=1}^{r} e_j c_j(s,d,k) = 0, & s = 0, \ldots, S, \\ & k = 1, \ldots, 2n. \end{array} \right. \tag{5.12}$$

By computing the $\{0,1\}$-echelon basis of the system $\mathcal{S}_i$ given in (5.12) (notice that $\mathcal{S}_i$ admits such basis), the partition defined by this basis is the partition of L_i. This is summarized in the next algorithm.

Algorithm 5.1 `Partition` (slow, rational function version)

Input: Irreducible factors $F_1, \ldots, F_r$ of $\Phi_f(x)$ over $K(t)$ and an index $1 \leq i \leq r$.

Output: The partition P_i of L_i.

1. Compute the system $\mathcal{S}_i$ as in (5.12).
2. Compute the $\{0,1\}$-echelon basis of $\mathcal{S}_i$.
3. Let P_i be the partition defined by this basis.
4. **return** P_i.

However, algorithm `Partition` is not efficient in practice due to the (costly) $2nr$ polynomial divisions in $K(t)[x]$ required to compute the system $\mathcal{S}_i$. We shall present a probabilistic version of this algorithm in Subsection 5.3.3, which allows us to compute P_i much faster.

5.3.2 Valuation rings of $K(t)/K$

In this section we briefly recall the definition and some properties of valuation rings of a rational function field. We will use valuation rings to simplify and speed up the computation of the partition P_i of L_i.

Definition 5.13. *A valuation ring of $K(t)/K$ is a ring $\mathcal{O} \subseteq K(t)$ such that $K \subsetneq \mathcal{O} \subsetneq K(t)$ and for every $g \in K(t)$ we have $g \in \mathcal{O}$ or $1/g \in \mathcal{O}$.*

Valuation rings are *local rings*, that is, if $\mathcal{O}$ is a valuation ring, then there exists a unique maximal (and principal) ideal $\mathcal{P} \subseteq \mathcal{O}$.

Lemma 5.14 ([43])**.** *Let $p(x) \in K[x]$ be an irreducible polynomial. Define*

$$\mathcal{O}_p := \{g(t) \in K(t) \ : \ p(x) \nmid Den(g(x))\} \ \text{ and}$$

$$\mathcal{P}_p := \{g(t) \in K(t) \ : \ p(x) \nmid Den(g(x)) \ \text{and} \ p(x) \mid Num(g(x))\}.$$

Then $\mathcal{O}_p$ is a valuation ring of $K(t)/K$ with maximal ideal $\mathcal{P}_p$.

Furthermore, every valuation ring $\mathcal{O}$ of $K(t)/K$ is of the form $\mathcal{O}_p$, for some irreducible polynomial $p(x) \in K[x]$, or is the place at infinity of $K(t)/K$, that is, $\mathcal{O} = \{g(t) \in K(t) \ : \ \deg(\mathrm{Num}(g(x))) \leq \deg(\mathrm{Den}(g(x)))\}$.

Lemma 5.15 ([43])**.** *Let $\mathcal{O}_p$ be a valuation ring of $K(t)/K$, where $p \in K[x]$ is an irreducible polynomial and let $\mathcal{P}_p$ be its maximal ideal. Let $\boldsymbol{F}_p$ be the residue class field $\mathcal{O}_p/\mathcal{P}_p$. Then $\boldsymbol{F}_p \cong K[x]/\langle p(x)\rangle$.*

5.3.3 Probabilistic Algorithm

In this section we present a probabilistic version of Algorithm 5.1. As in the number field case, we start noticing that fewer points are enough to find the partition P_i (usually much less than $2n$). Furthermore, the equations of the system

$\mathcal{S}_i$ come from the computation of $r_{i,j,k} \in K(t)[x]$ in (5.11), which involves a (costly) polynomial division over $K(t)$. Let us define a *good ideal* $\mathcal{P}_p$:

Definition 5.16. *Let $f \in K(t)$ and let $F_1, \ldots, F_r$ be the monic irreducible factors of $\Phi_f(x)$ over $K(t)$. Let $\mathcal{O}_p \subset K(t)$ be a valuation ring with maximal ideal $\mathcal{P}_p$, where $p = p(x) \in K[x]$ is irreducible. Let $\boldsymbol{F}_p$ be its residue field. We say that $\mathcal{P}_p$ is a good $K(t)$-ideal (with respect to f) if*

1) $F_i \in \mathcal{O}_p[x]$, $i = 1, \ldots, r$.

2) The image of f in $\boldsymbol{F}_p$ is not zero.

3) The image of $\Phi_f(x)$ in $\boldsymbol{F}_p[x]$ is separable.

To avoid the expensive computations of $r_{i,j,k} \in K(t)[x]$, we shall only compute its image modulo a good $K(t)$-ideal $\mathcal{P}_p$. These reductions will simplify our computations and we will still be able to construct a system of equations $\tilde{S}_i$ which is likely to give us the partition P_i.

Let $\mathcal{P}_p$ be a good $K(t)$-ideal, where $p = p(x) \in K[x]$ is irreducible. Let $\mathcal{O}_p$ be its valuation ring and $\mathbf{F}_p$ be the residue class field. Let $c \in K(f(t))$ be such that

$$h_{j,c}(t) := F_j'(c)/F_j(c) \in \mathcal{O}_p \subseteq K(t),$$

for $j = 1, \ldots, r$, and let $p_{j,c}(t), l_c(t) \in K[t]$ be as in Equation (5.7), that is, $\sum e_j h_{j,c}(c) = \frac{\sum e_j p_{j,c}(t)}{l_c(t)}$. Let $\tilde{F}_i$ be the image of F_i in $\mathbf{F}_p[x]$ and let $\tilde{h}_{j,c}$ be the image of $h_{j,c}$ in $\mathbf{F}_p$. Consider

$$\tilde{q}_{j,c} := p_{j,c}(x) - \tilde{h}_{j,c} l_c(x) \in \mathbf{F}_p[x] \tag{5.13}$$

and let $\tilde{r}_{i,j,c} := \tilde{q}_{j,c} \bmod \tilde{F}_i \in \mathbf{F}_p[x]$. Let d_p be the degree of $p(x) \in K[x]$ and let α be one of its roots. By Lemma 5.15 we have $\mathbf{F}_p \cong K[\alpha]$ and hence, we can write

$$\tilde{r}_{i,j,c} = \sum_{d=0}^{d_i - 1} \sum_{s=0}^{d_p - 1} C_j(s,d) \alpha^s x^d, \text{ where } C_j(s,d) \in K.$$

Consider the system $\tilde{\mathcal{S}}_{i,c}$ given by

$$\tilde{\mathcal{S}}_{i,c} := \begin{cases} \displaystyle\sum_{j=1}^{r} e_j C_j(s,d) = 0, & d = 0, \ldots, d_i - 1, \\ & s = 0, \ldots, d_p - 1. \end{cases} \tag{5.14}$$

If $(e_1, \ldots, e_r) \in \{0,1\}^r$ is a solution of $\mathcal{S}_i$ in (5.12), then $(e_1, \ldots, e_r)$ must also satisfy the system $\tilde{\mathcal{S}}_{i,c}$ in (5.14). The converse, however, need not be true. A basis of solutions of $\tilde{\mathcal{S}}_{i,c}$ is not necessarily a basis of solutions of $\mathcal{S}_i$. In fact, a basis of solutions of $\tilde{\mathcal{S}}_{i,c}$ might not even be a $\{0,1\}$-echelon basis. If this happens we need to consider more equations by taking $c' \in K(f(t))$ such that $h_{j,c'}(t) \in \mathcal{O}_p$, for $j = 1, \ldots, r$, and solving $\tilde{S}_i := \tilde{\mathcal{S}}_{i,c} \cup \tilde{\mathcal{S}}_{i,c'}$, and so on.

Remark 5.17. *Considering $\tilde{\mathcal{S}}_i$ over $\mathcal{S}_i$ has several advantages.*

1. *Smaller number of polynomial divisions to construct $\tilde{\mathcal{S}}_i$.*
2. *The polynomial divisions are now over $K[x]/p(x)$, where $p(x) \in K[x]$ is the polynomial defining the good ideal $\mathcal{P}$.*
3. *Smaller system: $\tilde{\mathcal{S}}_i$ has at most dd_id_p equations, where d is the number of c's used to construct $\tilde{S}_i$, while $\mathcal{S}_i$ has at most $2nd_iS$ equations in $r \leq n$ variables. Usually, $d \ll 2n$ and in several cases (e.g. when $char(K) = 0$) we can take $p(x)$ linear and hence $d_p = 1$ and $\tilde{\mathcal{S}}_i$ has at most dd_i equations (see Table 1 and Remark 5.24).*

Although in practice very few elements $c \in K(f(t))$ are needed to find P_i, we were not able to show that $2n$ elements are sufficient to correctly compute P_i. Let $\tilde{\mathcal{S}}_{i,c}$ be as in (5.14). Then $\tilde{S}_{i,c}$ might not have enough equations to correctly compute P_i (the correct partition of L_i). Thus, as mentioned above, we try to find P_i by solving $\tilde{S}_{i,c} \cup \tilde{S}_{i,c'}$, for $c, c' \in K(f(t))$, and so on. As in the general case, we give a halting condition that tells us when to stop adding more equations, that is, when the partition $\tilde{P}_i$ we computed by solving the system $\tilde{\mathcal{S}}_i$ equals P_i.

Let $\tilde{S}_i = \cup \tilde{S}_{i,c}$ be a system constructed from several $c \in K(f(t))$. If $\tilde{S}_i$ does not have a $\{0,1\}$-echelon basis then we clearly need more equations. Now let us suppose that $\tilde{S}_i$ has a $\{0,1\}$-echelon basis. Then the partition $\tilde{P}_i$ corresponding to this basis might still be a proper refinement of P_i. To show that $\tilde{P}_i = P_i$ it suffices to show that the $\tilde{P}_i$-products are polynomials in $L_i[x]$ (recall Theorem 3.43). In order to apply this theorem, consider the following map

$$\begin{aligned} \Psi_i : K(t) &\rightarrow K(t,x) \\ g(t) &\mapsto \tfrac{g_n(x) \bmod F_i}{g_d(x) \bmod F_i}. \end{aligned}$$

Hence, $g(t) \in L_i$ if, and only if, $\Psi_i(g) = g$ (see Definition 5.10) and therefore, we can rewrite L_i as $L_i = \{g(t) \in K(t) \ : \ \Psi_i(g(t)) = g(t)\}$.

Theorem 5.18. *Let P_i be the partition of L_i and let $\tilde{P}_i$ be the partition found by solving a system $\tilde{\mathcal{S}}_i$ defined as above. Let $\mathcal{P}_p$ be a good $K(t)$-ideal. If $\tilde{g}_1, \ldots, \tilde{g}_s \in K(t)[x]$ are the $\tilde{P}_i$-products and if*

$$\Psi_i(\tilde{g}_j) \equiv \tilde{g}_j \bmod \mathcal{P}_p, \ j = 1, \ldots, s, \tag{5.15}$$

where Ψ_i acts on $\tilde{g}_j$ coefficient-wise, then $\tilde{P}_i = P_i$.

Proof. Since $\tilde{P}_i$ is a refinement of P_i, it suffices to show that the $\tilde{P}_i$-products $\tilde{g}_1 \ldots, \tilde{g}_s$ are polynomials in $L_i[x]$. That is, we have to show that $\Psi_i(\tilde{g}_j) = \tilde{g}_j$, for $j = 1, \ldots, s$. Since

$$\tilde{g}_1 \cdots \tilde{g}_s = \Phi_f(x) = \Psi_i(\Phi_f(x)) = \Psi_i(\tilde{g}_1) \cdots \Psi_i(\tilde{g}_s)$$

and $\Psi_i(\tilde{g}_j) = \tilde{g}_j \bmod \mathcal{P}_p$, for $1 \leq j \leq s$, then Theorem 3.43 implies that $\Psi_i(\tilde{g}_j) = \tilde{g}_j$. Thus $\tilde{g}_j \in L_i[x]$, for $j = 1, \ldots, s$, and $\tilde{P}_i = P_i$. □

Here, we could just use Algorithm 3.5, where $\tilde{\mathcal{S}}_i$ is computed as above and the correctness check is given by Equation (5.15) in Theorem 5.18. However, we notice that the polynomials $\tilde{q}_{j,c} \in \mathbf{F}_p[x]$, $j = 1, \ldots, r$, in Equation (5.13) can be used to construct the system $\tilde{\mathcal{S}}_{i,c}$, for every $1 \leq i \leq r$. Since computing $\tilde{q}_{j,c}$ is

not free of charge, we want to use this fact. In the number field case, $\tilde{q}_{j,c} \in \mathbf{F}_p[x]$ corresponds to the element $f_j'(c)/f_j(c) \bmod \mathfrak{p} \in \mathbf{F}[\alpha]$ in step 3 of Algorithm 3.4, which is easily computed. First, we will present an algorithm that checks when we have the correct partition of L_i from a given system $\tilde{\mathcal{S}}_i$. Then, we present the algorithm that computes the partitions $P_1, \ldots, P_r$.

Algorithm 5.2 `Check`

Input: A linear system $\mathcal{S}$ in $e_1, \ldots, e_r$ and an index i.

Output: The partition P_i of L_i or *false*.

1. Compute a basis of solutions of $\mathcal{S}$.
2. **if** this basis is not a $\{0,1\}$-echelon basis **then**
3. **return** false **Need more equations.*
4. Let $\tilde{P}_i$ be the partition defined by this basis.
5. Let $\tilde{F}_i$ be the image of F_i in $\mathbf{F}_p$.
6. Let $\tilde{g}_1, \ldots, \tilde{g}_d$ be the $\tilde{P}_i$-products.
7. **for** every coefficient $c = \frac{c_n(t)}{c_d(t)} \in K(t)$ of $\tilde{g}_1, \ldots, \tilde{g}_d$ **do**
8. Let $\tilde{c}$ be the image of c in $\mathbf{F}_p$.
9. **if** $c_n(x) \bmod \tilde{F}_i \neq \tilde{c} \cdot (c_d(x) \bmod \tilde{F}_i)$ **then**
10. **return** false **Need more equations.*
11. **return** $\tilde{P}_i$

We shall now compute the complexity for one call of the algorithm `Check`. To simplify the proof, we first prove the following lemma.

Lemma 5.19. *Let $f_1, \ldots, f_r \in K[x,y]$ with $\sum \deg_x f_i = d$ and $\sum \deg_y f_i \leq n$. We can compute $F = \prod f_i$ with $\mathcal{O}(M(dn)\log_2 r)$ field operations.*

Proof. First of all, recall that the product fg, with f, g bivariate polynomials can be computed with $\mathcal{O}(M(d_x d_y))$, where d_x and d_y are bounds for the degrees of f and g in x and y, respectively. Let $d_i = \deg_x(f_i)$. We may suppose that $d_1 \leq d_2 \leq \cdots \leq d_r$.

Moreover, let us first consider the case where r is a power of 2. We shall compute F in $\log_2 r$ steps. The first step is to compute the $r/2$ products $f_1 \cdot f_2$, $f_3 \cdot f_4$, ..., $f_{r-1} \cdot f_r$. The cost of each product $f_{i-1} \cdot f_i$ is bounded by $\mathcal{O}(M(d_i n))$. Recall that M is super-additive, so that we have

$$M(d_2 n) + M(d_4 n) + \cdots + M(d_r n) \leq M((d_2 + d_4 + \cdots + d_r)n) \leq M(dn),$$

and the cost of this step is bounded by $\mathcal{O}(M(dn))$. The next step is to compute the $r/4$ products $(f_1 f_2) \cdot (f_3 f_4)$, $(f_5 f_6) \cdot (f_7 f_8)$, and so on. The cost of each product $(f_{i-2} f_{i-1}) \cdot (f_i f_{i+1})$ is bounded by $\mathcal{O}(M((d_i + d_{i+1})n))$. Again, by the super-additive property of M, this step can be bounded by $\mathcal{O}(M(dn))$. It is not difficult to see that every step can be bounded by $\mathcal{O}(M(dn))$. Since $\log_2 r$ steps are sufficient, the total cost is bounded by $\mathcal{O}(M(dn) \log_2 r)$. If r is not a power of 2, we can "complete" the factorization $f_1, \ldots, f_r$ to a factorization $f_1, \ldots, f_r, f_{r+1}, \ldots, f_{\tilde{r}}$, where $\tilde{r}$ is the smallest power of 2 greater than r and $f_{r+1} = \ldots = f_{\tilde{r}} = 1$. The sum of the degrees in each step is still bounded by d, so that each step is still bounded by $\mathcal{O}(M(dn))$. The number of steps is now $\log_2 \tilde{r} = \lceil \log_2 r \rceil \leq \log_2 r + 1$, and the result follows. □

Theorem 5.20. *Let $F_1, \ldots, F_r$ be the irreducible factors of $\Phi_f(x)$. Let $\tilde{\mathcal{S}}_i$ be a system computed as above. If algorithm* `Check` *returns a partition P, then P is the partition of L_i. Moreover, one call of Algorithm* `Check` *can be performed with*

$$\mathcal{O}(n_e r^{\omega-1} + M(n^2) \log_2 r + nM(n)M(d_p)) \text{ field operations},$$

where d_p is the degree of the polynomial defining $\mathcal{O}_p$, n_e is the number of equations in $\mathcal{S}$ and ω is a feasible matrix multiplication exponent.

Proof. The correctness of the algorithm follows from Theorem 5.18. Suppose that we are given the factors $F_1, \ldots, F_r$ of $\Phi_f(x)$. A basis of solutions of $\mathcal{S}$ can be computed with $\mathcal{O}(n_e r^{\omega-1})$ field operations. If this basis is not a $\{0,1\}$-echelon basis, then the algorithm returns *false*. If $\tilde{d}_i$ is the degree in x of $\tilde{g}_i$, then $\tilde{g}_i$ can be computed with $\mathcal{O}(M(\tilde{d}_i n) \log_2 r)$ field operations, by Lemma 5.19. By the super-additive property

of M, the computation of the polynomials $\tilde{g}_1, \ldots, \tilde{g}_d$ in Step 5 can be done with $\mathcal{O}(M(n^2)\log_2 r)$ field operations. For each coefficient of $\tilde{g}_1, \ldots, \tilde{g}_d$, we have to verify the condition in Step 9, which can be performed with a reduction modulo $\mathcal{P}_p$ (to compute $\tilde{c}$) and two polynomial divisions over $\mathbf{F}_p$. Therefore, for each c, we can perform Steps 8 and 9 with $\mathcal{O}(M(n)M(d_p))$ field operations. Since $\sum \deg \tilde{g}_i = n$, we have a total cost of $\mathcal{O}(nM(n)M(d_p))$ field operations for Steps 6-10. □

Finally, the following probabilistic algorithm computes the partitions $P_1, \ldots, P_r$ of the principal subfields $L_1, \ldots, L_r$ of $K(t)/K(f(t))$. The correctness of the algorithm follows from the correctness of the algorithm `Check`.

Algorithm 5.3 `Partitions` (fast, rational function version)

Input: The irreducible factors $F_1, \ldots, F_r$ of Φ_f and a good $K(t)$-ideal $\mathcal{P}_p$

Output: The partitions $P_1, \ldots, P_r$ of $L_1, \ldots, L_r$.

1. Let $\tilde{\mathcal{S}}_i = \{\ \}$, $i = 1, \ldots, r$.
2. $I := \{1, \ldots, r\}$.
3. **while** $I \neq \emptyset$ **do**
4. Let $c \in K(f(t))$ such that $h_{j,c}(t) \in \mathcal{O}_p$, $j = 1, \ldots, r$.
5. Compute $\tilde{q}_{j,c} \in \mathbf{F}_p[x]$, $j = 1, \ldots, r$.
6. **for** $i \in I$ **do**
7. Compute the system $\tilde{\mathcal{S}}_{i,c}$ (see Equation (5.14)).
8. Let $\tilde{\mathcal{S}}_i := \tilde{\mathcal{S}}_i \cup \tilde{\mathcal{S}}_{i,c}$.
9. **if** `Check`$(\tilde{S}_i, i) \neq$ *false* **then**
10. `Remove`(I, i).
11. Let P_i be the output of `Check`$(\tilde{S}_i, i)$.
12. **return** $P_1, \ldots, P_r$.

Remark 5.21. *In general, the elements in Step 4 can be taken inside K. This will work except, possibly, when K has very few elements, which might not be enough to find P_i. If this happens we have two choices:*

1) *Choose $c \in K(f(t))\backslash K$ or*

2) *Extend the base field K and compute/solve the system $\tilde{\mathcal{S}}_i$ over this extension.*

We choose the latter. Recall that the solutions we are looking for are composed of 0's and 1's and hence can be computed over any extension of the base field K. Furthermore, extending the base field does not create new solutions since the partitions are determined by the factorization of $\Phi_f(x)$ computed over $K(t)$, where K is the original field.

In what follows, we determine the complexity of computing $P_1, \ldots, P_r$. We assume, based on our experiments (see Table 5.1), that the algorithm finishes using $\mathcal{O}(1)$ elements $c \in K$ (or over a finite extension of K) to generate a system $\tilde{\mathcal{S}}_i$ which gives us the partition P_i.

Theorem 5.22. *Assuming that* Algorithm `Partitions` *finishes using $\mathcal{O}(1)$ elements inside K in Step 4, the partitions $P_1, \ldots, P_r$, corresponding to the principal subfields $L_1, \ldots, L_r$ of the extension $K(t)/K(f(t))$, can be computed with an expected number of*

$$\mathcal{O}(r(rM(n)M(d_p) + M(n^2)\log_2 r)) \text{ field operations,}$$

where d_p is the degree of the polynomial defining the ideal $\mathcal{P}_p$.

Proof. Given an element $g = \frac{g_n(t)}{g_d(t)} \in \mathcal{O}_p$, we can compute its image in $\mathbf{F}_p$ with $\mathcal{O}(M(d_g) + M(d_p))$ field operations, where d_g is the degree of $g(t) \in K(t)$. Hence,

given the irreducible factors $F_1, \ldots, F_r$ of Φ_f, we can compute their images over $\mathbf{F}_p$ with $\mathcal{O}(n(M(n) + M(d_p)))$ field operations.

Let $c \in K$. The first step is to compute $h_{j,c} := F_j'(c)/F_j(c) = G_j'(c)/G_j(c) \in \mathcal{O}_p$, $j = 1, \ldots, r$ (see Remark 5.7). Evaluating $G_j \in K[x,t]$ at $x = c$ costs $\mathcal{O}(nd_j^x)$, where $d_j^x = \deg_x(G_j)$. If $d_j^t = \deg_t(G_j)$, then simplifying the rational function $G_j'(c)/G_j(c)$ to its minimal form costs $\mathcal{O}(M(d_j^t)\log d_j^t)$. Keeping in mind the super-additive property of $M(\cdot)$ and that $\sum d_j^t = \sum d_j^x = n$, one can compute $h_{j,c}, j = 1, \ldots, r$, with $\mathcal{O}(n^2 + M(n)\log n)$ field operations. Furthermore, since $c \in K$, then $\deg_t(h_{j,c}) \leq d_j^t$ and we can compute the image of $h_{j,c}, j = 1, \ldots, r$, in $\mathbf{F}_p$ with $\mathcal{O}(M(n) + rM(d_p))$ field operations.

Let us write $h_{j,c} = n_{j,c}/d_{j,c}$, where $n_{j,c}, d_{j,c} \in K[t]$ are coprime. The next step is the computation of $l_c(t) = \operatorname{lcm}(d_{1,c}(t), \ldots, d_{r,c}(t))$, which can be done with r lcm computations with a total cost of $\mathcal{O}(rM(n)\log n)$ field operations. Next, we compute $p_{j,c}(x) := l_c(x)\frac{n_{j,c}(x)}{d_{j,c}(x)}$, which involves one polynomial division and one polynomial multiplication, for each $j = 1, \ldots, r$. Hence, the total cost to compute $\tilde{q}_{j,c} = p_{j,c}(x) - h_{j,c}(t)l_c(x)$, $j = 1, \ldots, r$, is $\mathcal{O}(rM(n))$ field operations.

For each $i = 1, \ldots, r$, to compute the partition P_i we have to compute the system $\tilde{S}_{i,c}$, which involves the division of $\tilde{q}_{j,c}$ by $\tilde{F}_i$, for $j = 1, \ldots, r$. Since $\deg(\tilde{q}_{j,c}(x)) \leq n$, each of these divisions cost $\mathcal{O}(M(n)M(d_p))$ field operations and hence, we can compute the system $\tilde{S}_{i,c}$ with $\mathcal{O}(rM(n)M(d_p))$ field operations. This system has at most $d_i d_p$ equations and hence, one call of algorithm `Check` costs $\mathcal{O}(d_i d_p r^{\omega-1} + M(n^2)\log_2 r + M(n)M(d_p))$. The result follows by adding terms. □

Remark 5.23. *Let #c be the number of elements $c \in K$ needed to correctly compute all partitions $P_1, \ldots, P_r$. Then the total cost of Algorithm* `Partitions` *is bounded by #c times the cost given in Theorem 5.22. In practice, however, $\#c \ll 2n$ (see Table 5.1 for a few examples).*

Algorithm `Partitions` gives us the partitions of the principal subfields of the extension $K(t)/K(f(t))$. One can now compute the partitions of every subfield of $K(t)/K(f(t)$ by joining partition-vectors, as explained in Chapter 3. As the complexity depends on the degree of $p(x)$, we wish to find a bound for this number. As it turns out, choosing $p(x)$ of degree $\mathcal{O}(\log n)$ suffices.

Remark 5.24 (A bound for the degree of $p(x)$)**.** *Condition* 1) *in Definition 5.16 is equivalent to* $p(x) \nmid f_d(x)$ *(see Remark 5.7) and condition 2) is equivalent to* $p(x) \nmid f_n(x)$*. Let* $\bar{\Phi}_f$ *be the image of* Φ_f *in* $\boldsymbol{F}_p[x]$*. By Gauss' Lemma,* $\bar{\Phi}_f$ *is separable if, and only if,* $\bar{\nabla}_f$ *is separable, where* $\bar{\nabla}_f$ *is the image of* $\nabla_f = f_n(x)f_d(t) - f_n(t)f_d(x) \in K[x,t]$ *in* $\boldsymbol{F}_p[x]$*. Notice that* $\bar{\nabla}_f = \nabla_f(x,\alpha) \in \boldsymbol{F}_p[x]$*, where* α *is a root of* $p(x)$*. Consider* α *as a variable. Hence,* $\nabla_f(x,t)$ *separable means that* $\nabla_f(x,\alpha)$ *is separable in* $K[x,\alpha]$ *and by Gauss' Lemma,* $\nabla_f(x,\alpha)$ *is separable in* $K(\alpha)[x]$*. The latter is equivalent to* $R := Res_x(\nabla_f(x,\alpha), \nabla'_f(x,\alpha)) \in K[\alpha]$ *being a non-zero polynomial in* α *of degree at most* $(2n-1)n$*, where* Res_x *is the resultant w.r.t. the variable* x*. If we let* α *vary, for instance, be a root of* $p(x)$*, then* $\bar{\Phi}_f \in \boldsymbol{F}_p[x]$ *is separable if, and only if,* α *is not a root of the polynomial* R*. If* α *is a root of* R*, then the ideal is not a good ideal and we choose a different* $p(x)$*. However, instead of choosing a different* $p(x)$*, we can pick a different element* $c \in K[x]/\langle p(x)\rangle$*. We still need to check whether* c *is a "good evaluation point" (i.e.,* $f_n(c) \neq 0$*,* $f_d(c) \neq 0$ *and* $\Phi(x,c) \in \boldsymbol{F}_p[x]$ *is separable), but this gives us* $size(K)^{\deg(p(x))}$ *possible evaluation points. Hence, if* $size(K)^{\deg(p(x))} > \deg(R) = (2n-1)n$*, then there exists at least one good evaluation point* c *such that* $\Phi_f(x,c)$ *is separable. Choosing a random* $c \in K[x]/\langle p(x)\rangle$ *instead of a root of* $p(x)$ *might slow the computations down, but this proves that the degree of* $p(x)$ *does not have to be so high. Hence, it suffices to choose* $p(x)$ *with* $\deg(p(x)) \in \mathcal{O}(\log n)$*. In practice, however, we start by choosing* $p(x)$ *of lowest degree (e.g., 1,2,...). If we do not find a good ideal (i.e., a good* $p(x)$*) for a certain degree* d *after a certain number of tries, then we increase the number* d *and try to find a good* $p(x)$ *of degree* d*. This is motivated by the fact*

that R appears to have only "few" irreducible factors over $K[x]/\langle p(x)\rangle$, that is, R appears to be the product of a few irreducible polynomials with high multiplicity (this was observed in a few examples where R was computable in a reasonable amount of time). Moreover, choosing a small $p(x)$ (degree-wise) helps to speed up the algorithm, since computations in $\boldsymbol{F}_p[x]$ depend on $\deg(p(x))$. Notice that this is only necessary in positive characteristic. In characteristic 0, we can always choose $p(x)$ linear.

Theorem 5.25. *Let $f \in K(t)$ of degree n and let $F_1, \ldots, F_r$ be the irreducible factors of $\Phi_f(x) \in K(t)[x]$. Let m be the number of subfields of $K(t)/K(f(t))$. One can compute, using fast arithmetic, the subfield lattice of $K(t)/K(f(t))$ (in terms of partitions) with $\tilde{\mathcal{O}}(rn^2)$ field operations plus $\tilde{\mathcal{O}}(mr^2)$ CPU operations.*

Proof. Using fast arithmetic, by Theorem 5.22, we can compute the partitions of the principal subfields with $\tilde{\mathcal{O}}(rn^2 d_p)$ field operations. By Remark 5.24, $d_p \in \mathcal{O}(\log n)$ in the worst case. The complete subfield lattice can be computed with $\tilde{\mathcal{O}}(mr^2)$ CPU operations, as in the general case. □

5.4 General Algorithm, the Polynomial Case and some Timings

In this section we briefly mention all steps for computing all complete decompositions of a rational function and give an example. Some timings, comparing our algorithm with [6], are also given. We also analyze the case $f \in K[t]$, and conclude that our algorithm has better complexity than that of [10].

5.4.1 General Algorithm

Let $f \in K(t)$ be a rational function and let $F_1, \ldots, F_r$ be the monic irreducible factors of Φ_f. By Theorem 5.8, each complete decomposition corresponds

to a maximal sequence of subfields of $K(t)/K(f(t))$ and vice-versa. Using Algorithm 5.3 to compute the partitions of every principal subfield and Algorithm `Join` from Chapter 3, we can (quickly) compute the subfield lattice of $K(t)/K(f(t))$, where each subfield is represented by a partition.

To actually compute the decompositions of f we need a *Lüroth generator* for each subfield. That is, given a partition P_L of $\{1,\ldots,r\}$ representing a subfield L of $K(t)/K(f(t))$, we want to find a rational function $h \in K(t)$ such that $L = K(h(t))$. An algorithm for computing $h \in K(t)$ is given in Ayad & Fleischmann [6]. In what follows we show a simpler way for computing this generator.

Theorem 5.26 (Lüroth Generator)**.** *Let $f \in K(t)$ be a rational function and let $F_1,\ldots,F_r$ be the monic irreducible factors of $\Phi_f(x) \in K(t)[x]$. Let L be a subfield of $K(t)/K(f(t))$ and let $P = \{P^{(1)},\ldots,P^{(s)}\}$ be the partition of L. Let $g := \prod_{i\in P^{(1)}} F_i \in L[x]$. If $c \in K(t)$ is any coefficient of g not in K, then $L = K(c)$.*

Proof. By Luröth's Theorem, there exists a rational function $h(t) \in K(t)$ such that $L = K(h(t))$. Let $\Phi_h \in L[x]$. By Remark 5.6 we may suppose that $\Phi_h \in L[x]$ is the minimal polynomial of t over L. Let $g = \prod_{i\in P^{(1)}} F_i \in L[x]$. Since $1 \in P^{(1)}$ (recall that $F_1 = x - t$), it follows that $g(t) = 0$ and hence, $\Phi_h \mid g$. However, Φ_h and g are monic and irreducible polynomials (over L) and hence, $g = \Phi_h$. Therefore, $g = h_n(x) - h(t)h_d(x)$. Let c_i be the coefficient of x^i in g, then

$$c_i = h_{ni} - h(t)h_{d,i} = (-h_{d,i}t + h_{n,i}) \circ h(t),$$

where $h_{n,i}$ and $h_{d,i}$ are the coefficients of x^i in $h_n(x)$ and $h_d(x)$, respectivly. If $h_{d,i} \neq 0$, then $-h_{d,i}t + h_{n,i}$ is a unit and hence, $L = K(h(t)) = K(c_i)$. □

The last ingredient we need for computing all complete decompositions of f is the computation of the left component: given $f, h \in K(t)$, find $g \in K(t)$ such that $f = g \circ h$. It is known that g is unique (see [4]) and several methods exist for

finding g. The most straightforward method is to solve a linear system of equations in the coefficients of g, a detailed approach is given by Dickerson [16]. Another approach is given by Giesbrecht [22] and uses $\mathcal{O}(nM(n)\log n)$ field operations. We shall not dwell on this step and use one of these known algorithms.

Example 5.27. *Let* $f := (t^{24}-2t^{12}+1)/(t^{16}+2t^{12}+t^8)$ *and consider the extension* $\mathbb{Q}(t)/\mathbb{Q}(f)$. *The minimal polynomial of* t *over* $\mathbb{Q}(f(t))$ *is*

$$\Phi_f(x) = x^{24} - 12x^{12} + 1 - \frac{t^{24}-2t^{12}+1}{t^{16}+2t^{12}+t^8}(x^{16}+2x^{12}+x^8).$$

The irreducible factors of $\Phi_f(x)$ *over* $\mathbb{Q}(t)$ *are*

$$F_1 = x - t,\ F_2 = x + t,\ F_3 = x + \frac{1}{t},\ F_4 = x - \frac{1}{t},\ F_5 = x^2 + t^2,\ F_6 = x^2 + \frac{1}{t^2},$$

$$F_7 = x^8 + \frac{t^8+1}{t^8+t^4}x^4 + \frac{1}{t^4} \text{ and } F_8 = x^8 + \frac{t^8+1}{t^4+1}x^4 + t^4.$$

Using Algorithm 5.3*, we get the following partitions of the principal subfields* $L_1, \ldots, L_8$*:*

$$\begin{aligned}
P_1 &= \{\{1\},\{2\},\{3\},\{4\},\{5\},\{6\},\{7\},\{8\}\}\\
P_2 &= \{\{1,2\},\{3,4\},\{5\},\{6\},\{7\},\{8\}\}\\
P_3 &= \{\{1,3\},\{2,4\},\{5,6\},\{7,8\}\}\\
P_4 &= \{\{1,4\},\{2,3\},\{5,6\},\{7,8\}\}\\
P_5 &= \{\{1,2,5\},\{3,4,6\},\{7\},\{8\}\}\\
P_6 &= \{\{1,2,6\},\{3,4,5\},\{7,8\}\}\\
P_7 &= \{\{1,2,5,7\},\{3,4,6,8\}\}\\
P_8 &= \{\{1,2,3,4,5,6,7,8\}\}.
\end{aligned}$$

By joining the partitions of all subsets of $\{P_1, \ldots, P_8\}$*, we get the following new partitions:*

$$\begin{aligned}
P_9 &= P_2 \vee P_4 = \{\{1,2,3,4\},\{5,6\},\{7,8\}\}\\
P_{10} &= P_3 \vee P_6 = \{\{1,2,3,4,5,6\},\{7,8\}\}.
\end{aligned}$$

Hence, $P_1, \ldots, P_{10}$ are the partitions corresponding to every subfield of $\mathbb{Q}(t)/\mathbb{Q}(f(t))$. Next we compute all maximal sequences of subfields. Recall that the subfield relation is translated as refinement of partitions, for instance, $L_5 \subseteq L_2$, since P_2 refines P_5. Therefore, by looking at the partitions $P_1, \ldots, P_{10}$, we see that one maximal sequence of subfields is

$$\mathbb{Q}(f) = L_8 \subseteq L_7 \subseteq L_5 \subseteq L_2 \subseteq L_1 = \mathbb{Q}(t).$$

Now, let us compute generators for these fields. As an example, let us compute a generator for L_7. Following Theorem 5.26, let

$$g = \prod_{i \in P_7^{(1)}} F_i = F_1F_2F_5F_7 = x^{12} - \left(\frac{t^{12}-1}{t^8+t^4}\right)x^8 - \left(\frac{t^{12}-1}{t^8+t^4}\right)x^4 - 1.$$

Hence, a generator of L_7 is $\frac{t^{12}-1}{t^8+t^4} \in \mathbb{Q}(t)$. That is, $L_7 = \mathbb{Q}\left(\frac{t^{12}-1}{t^8+t^4}\right)$. By computing a Lüroth generator for every subfield in this sequence of subfields we get

$$\mathbb{Q}(f) \subseteq \mathbb{Q}\left(\frac{t^{12}-1}{t^8+t^4}\right) \subseteq \mathbb{Q}(t^4) \subseteq \mathbb{Q}(t^2) \subseteq \mathbb{Q}(t).$$

Finally, we compute the corresponding complete decomposition of f by computing left components. For instance, $\mathbb{Q}(f) \subseteq \mathbb{Q}\left(\frac{t^{12}-1}{t^8+t^4}\right)$ implies that there exists $g \in K(t)$ such that $f = g \circ \frac{t^{12}-1}{t^8+t^4}$. In this case we have $g = t^2$ and hence

$$f = t^2 \circ \frac{t^{12}-1}{t^8+t^4}.$$

Now $\mathbb{Q}(\frac{t^{12}-1}{t^8+t^4}) \subseteq \mathbb{Q}(t^4)$ and we can write $\frac{t^{12}-1}{t^8+t^4} = \frac{t^3-1}{t^2+t} \circ t^4$, and so on. This yields the following complete decomposition of f:

$$f = t^2 \circ \frac{t^3-1}{t^2+t} \circ t^2 \circ t^2.$$

Doing this for every maximal sequence of subfields yields all non-equivalent complete decompositions of f. In Magma, all 6 non-equivalent complete decompositions of f were computed in 0.02 seconds.

5.4.2 The Polynomial Case

Finally, we consider the case where $f \in K[t]$ is a polynomial. There are already several algorithms for computing decompositions of polynomials, including the polynomial time algorithm of Kozen & Landau [29] and the improvements from von zur Gathen [55], which computes a decomposition $f = g \circ h$ with $\tilde{\mathcal{O}}(n)$ field operations. However, these algorithms only work if $\deg(g)$ is invertible in K. Algorithms with no restrictions on the degree of g are presented by Alonso *et al.* [4], Ayad & Fleischmann [6] (based on Theorem 5.8) and more recently, by Blankertz [10] (based on blocks of imprimitivity and Zippel's work [60]). Our algorithm also works when $f \in K[t]$ as long as we normalize the subfield generators. In other words, the output is the set of all (polynomial) decompositions of f. This follows from the following fact.

Lemma 5.28 ([6], Corollary 2.3)**.** *Let $L = K(\tilde{h}) \subseteq K(t)$, with $\tilde{h} \in K(t)$ normalized. Then L contains a non-constant polynomial if and only if $\tilde{h} \in K[t]$.*

Blankertz [10], following the ideas of Zippel [60] and Landau & Miller [30], proposes an algorithm to compute all minimal decompositions of a polynomial $f \in K[t]$. If $f = g \circ h$ is a minimal decomposition then $K(h(t))$ is a *maximal subfield*, that is, there exists no subfield L of $K(t)/K(f(t))$ such that

$$K(h(t)) \subsetneq L \subsetneq K(t). \tag{5.16}$$

That is, $K(h(t))$ must be a principal subfield. Thus, once the partitions $P_1, \ldots, P_r$ are computed, it is very easy to verify which of these partitions represent a maximal subfield. For a principal subfield, a Lüroth generator can be obtained as a byproduct of Algorithm `Check`. Hence, to compute all minimal decompositions of f, we need to compute at most r left components.

Theorem 5.29. *Given $f \in K[t]$ and the factorization of $f(x) - f(t)$ in $K[x,t]$, one can compute, using fast arithmetic, all minimal decompositions of f with*

$$\tilde{\mathcal{O}}(rn^2) \text{ field operations.}$$

When $\mathrm{char}(K) > 0$, the factorization of $f(x) - f(t)$ can be computed with $\tilde{\mathcal{O}}(n^{\omega+1})$ field operations, where $2 < \omega \leq 3$ is a feasible matrix multiplication exponent (see [12] and [34]) and hence, dominates the complexity. Moreover, our algorithm has better complexity than the algorithm presented in [10], whose expected complexity (for K finite) is $\tilde{\mathcal{O}}(n^6)$ field operations.

5.4.3 Timings

In this last section we give some timings, comparing our algorithm (column `Decompose`), which returns all non-equivalent complete decompositions of f, with the algorithms `full_decomp`(f) and `all_decomps`(f), from Ayad & Fleischmann [6]. The Algorithm `full_decomp`(f) returns a single complete decomposition of f, while `all_decomps`(f) returns all non-equivalent complete decompositions of f. Both of these algorithms were implemented by the authors of [6], and are available at `http://www.kent.ac.uk/ims/personal/pf10/calais/decomp`.

Some of the rational functions used in the table below were constructed by composing rational functions of smaller degree, however, these examples are not very interesting. Thus, for interesting examples, we need to pick rational functions rather carefully. In the table below, n is the degree of $f \in K(t)$, r is the number of irreducible factors of Φ_f and $\#dec$ is the number of non-equivalent complete decompositions of f. We also list d_p, the degree of the polynomial defining the good $K(t)$-ideal and $\#c$, the number of elements in K (or an extension of K, see Remark 5.21), that were used to determine the partitions $P_1, \ldots, P_r$.

K	n	r	$\#dec$	$d_p, \#c$	`Decompose`	Ayad & Fleischmann [6]	
						`full_decomp`	`all_decomps`
$\mathbb{F}_{11}$	12	7	3	3,1	0.01	0.02	0.03
$\mathbb{Q}$	24	8	6	1,4	0.02	0.00	0.09
$\mathbb{Q}$	144	10	6	1,4	1.82	1.88	101.08
$\mathbb{F}_{11}$	24	10	8	3,1	0.02	0.01	0.20
$\mathbb{F}_3$	18	12	12	4,1	0.05	0.06	0.81
$\mathbb{F}_{11}$	24	14	12	4,1	0.07	0.51	10.57
$\mathbb{F}_3$	60	17	5	5,1	0.18	91.68	981.43
$\mathbb{Q}$	60	17	5	1,8	0.77	485.19	4,338.47
$\mathbb{F}_{17}$	96	26	44	2,4	0.42	211.30	$> 12h$
$\mathbb{F}_{11}$	60	60	111	3,5	1.91	$> 12h$	n.a.
$\mathbb{F}_{11}$	120	61	111	3,5	2.36	n.a.	n.a.
$\mathbb{F}_{13}$	169	91	14	3,7	3.41	n.a.	n.a.
$\mathbb{F}_5$	120	120	587	5,4	18.59	n.a.	n.a.
$\mathbb{F}_7$	168	168	680	4,9	50.53	n.a.	n.a.

n.a. - not attempted.

Tabela 5.1: Comparison Table - increasing values of r

In theory, our algorithm better compares to `all_decomps`, since both algorithms return all non-equivalent complete decompositions of f. According to our experiments, for *small* values of r (see Table 5.1), the time spent by algorithm `Decompose` to compute all complete decompositions is similar to the time spent by `full_decomp` to compute a single decomposition. However, as r increases, we see a noticeable improvement compared to `full_decomp` and more so to `all_decomps`.

We remark that, even if the number of factors is small (say $r \leq 10$) and one only needs one complete decomposition (not all of them), then computing

all decompositions using our algorithm is at least as fast as computing a single decomposition using `full_decomp` (we found only one example, with $r = 4$, where our algorithm was slower than `full_decomp`). See the following examples.

K	n	r	$\#dec$	$d_p, \#c$	`Decompose`	Ayad & Fleischmann [6]	
						`full_decomp`	`all_decomps`
$\mathbb{F}_3$	130	3	1	4,1	58.25	59.35	61.51
$\mathbb{Q}$	60	4	1	1,1	3.93	5.08	6.93
$\mathbb{F}_2$	360	4	1	4,1	0.44	0.31	4.55
$\mathbb{Q}$	48	5	1	1,2	1.88	2.48	4.35
$\mathbb{F}_3$	64	7	3	4,1	2.62	2.68	5.01
$\mathbb{Q}$	480	8	1	1,2	3.94	4.50	1,946.02
$\mathbb{Q}$	192	11	7	1,5	1,802.23	4,063.16	35,386.89

Tabela 5.2: Comparison Table - small values of r

More examples and details about timings can be found at `www.math.fsu.edu/~jszutkos/timings` and the implementation of our algorithm at `www.math.fsu.edu/~jszutkos/Decompose`.

CONCLUSION

Computer Algebra is an interesting and important area of Mathematics (and Computer Science!). Taking a handful of theorems and devising an algorithm has always fascinated me. During my period as a Ph.D. student, I was able to do just that. I have learned many results and ideas on various topics and being able to use them and write an algorithm has been a satisfying experience. However, I cannot not mention the countless hours trying to find bugs in the code and the nights spent trying to understand why the result is wrong when it shouldn't (and sometimes, vice-versa). Nevertheless, this has been an interesting experience.

The approach given by van Hoeij *et al.* [51], to compute the subfield lattice by intersecting principal subfields, has proven to be better than the algorithms presented in [26], especially when the number of factors is large. This was already noticed in [51]. Let $L_1, \ldots, L_r$ be (any) subfields of K/k. If every subfield L of K/k is the intersection of some of these subfields, then we say that $\{L_1, \ldots, L_r\}$ is an intersection-generating set for K/k. In another words, the set of principal subfields is an intersection-generating set for K/k. If g is any factor of f, we define the set

$$L_g = \{h(\alpha) \in K : h(x) \equiv h(\alpha) \bmod g(x)\}.$$

The set L_g is also a subfield of K/k. Let $g_1, \ldots, g_r$ be any factorization of f over K (that is, g_i not necessarily irreducible over K). If $g_1 = x - \alpha$ and if $L_{g_1}, \ldots, L_{g_r}$ is an intersection-generating set for K/k, then $g_1, \ldots, g_r$ is said to be a *subfield factorization* for K/k. Thus, we can find all subfields of K/k with three steps:

1. Find a *subfield factorization* $g_1, \ldots, g_r$ for K/k.
2. Compute the subfields $L_{g_1}, \ldots, L_{g_r}$ of K/k.
3. Compute all intersections of $L_{g_1}, \ldots, L_{g_r}$.

Steps 1 and 2 can be executed in polynomial time (provided factorization in $K[x]$ can be done in polynomial time as well). However, step 3 can not be executed in polynomial time. The number of intersections executed in step 3 can actually be bounded by rm, where m is the total number of subfields of K/k. However, the number m is not polynomially bounded in the degree of the extension K/k. Thus, the cost of step 3 is given by rm times the cost of each intersection, which is done using Linear Algebra over k (as in [51]).

Let $f_1, \ldots, f_r$ be (a subfied) factorization of f over K into irreducible factors. The main contribution of this work is the improvement of step 3 above. To that end, we associated, to every subfield L of K/k, a partition P_L of the set $\{1, \ldots, r\}$, where each i corresponds to the irreducible factor f_i. The partition P_L is defined as follows: let $g_1, \ldots, g_s$ be the irreducible factors of f over L. Each g_j is the product of some of the f_i's. Define $P_L := \{\{i : f_i \mid g_j\}, j = 1, \ldots, s\}$. In the example below, f has 5 irreducible factors over $k(\alpha)$ and 2 irreducible factors over L, giving us the partition P_L. Note that $L \subseteq L'$ if and only if, $P_{L'}$ refines P_L.

$$
\begin{array}{ccc}
k(\alpha) & f = f_1 \cdot f_2 \cdot f_3 \cdot f_4 \cdot f_5 & \longrightarrow P_K = \{\{1\},\{2\},\{3\},\{4\},\{5\}\} \\
| & \uparrow & \\
L & f = (f_1 f_2 f_3) \cdot (f_4 f_5) & \longrightarrow P_L = \{\{1,2,3\},\{4,5\}\} \\
| & \uparrow & \\
k & f = (f_1 f_2 f_3 f_4 f_5) & \longrightarrow P_k = \{\{1,2,3,4,5\}\}
\end{array}
$$

One of the main results shown in this work is stated below.

Theorem 3.26. *Let L, L' be subfields of K/k and let P_L and $P_{L'}$ be the corresponding partitions. Then $P_{L \cap L'} = P_L \vee P_{L'}$.*

The partition $P_L \vee P_{L'}$ is called the *join* of P_L and $P_{L'}$ and is defined as the finest partition that is refined by both P_L and $P_{L'}$, and can be computed using an algorithm proposed by Freese [19, 20]. We also presented an algorithm (which is

similar to Freese's algorithm) for computing this join. Its complexity is worse than Freese's algorithm however, it appears to be slightly faster in practice. Thus, given two subfields L_i, L_j, instead of directly computing their intersection $L_i \cap L_j$ using Linear Algebra over k (step 1 below), we first compute the partitions P_{L_i}, P_{L_j} (step 2), compute the join $P_{L_i} \vee P_{L_j}$ (step 3) and finally, compute a generator for $L_i \cap L_j$ from the partition $P_{L_i} \vee P_{L_j}$ and the subfield factorization (step 4).

$$\begin{array}{ccc} L_i, L_j & \xrightarrow{(2)} & P_{L_i}, P_{L_j} \\ \downarrow (1) & & \downarrow (3) \\ L_i \cap L_j & \xleftarrow[(4)]{} & P_{L_i} \vee P_{L_j} \end{array}$$

Steps (2), (3) and (4) are explained in Chapter 3. When working with number fields, that is, $k = \mathbb{Q}$, this method allowed us to show the following result.

Theorem 3.47. *Let m be the total number of subfields of K/k. When $k = \mathbb{Q}$, we can compute all subfields of K/k (in terms of partitions) with an expected number of $\tilde{\mathcal{O}}(rn^7 + rn^5 \log^2 \|f\|_2 + mr^2)$ CPU operations, where r is the number of factors in the subfield factorization and n is the degree of the extension.*

Using partitions to compute all subfields also improves CPU timings, especially when number of subfields is large (see Table 4.2). When $k = \mathbb{Q}$, van Hoeij *et al.* [51] also presented a method to compute the principal subfields using the LLL algorithm [35], thus avoiding the need to factor f over $\mathbb{Q}(\alpha)$. By following these ideas, we showed how to compute a *subfield factorization* for $\mathbb{Q}(\alpha)/\mathbb{Q}$. We also showed how to improve this step (Remark 4.6), thus reducing the number of calls to the LLL algorithm. For more details, see Chapter 4.

Finally, we looked at rational function fields. That is, if K is a field and $f(t) \in K(t)$ is a rational function, then $K(f(t))$ is a subfield of $K(t)$ and thus, $K(t)/K(f(t))$ is a finite degree extension. Moreover, we have a bijection between

subfields of this extension and the decompositions of f, that is, L is a subfield of $K(t)/K(f(t))$ if and only if, there exists $h(t) \in K(t)$ such that $L = K(h(t))$ and $f = g \circ h$, for some $g(t) \in K(t)$ (for more details, see [4]).

Hence, to find all decompositions of $f(t) \in K(t)$, it sufficed to find all subfields of $K(t)/K(f(t))$. The element $t \in K(t)$ is a primitive element and its minimal polynomial over $K(f(t))$ is given by[1] $\Phi_f := f_n(x) - f(t)f_d(x) \in K(f(t))[x]$, where $f_n(x), f_d(x) \in K[x]$ are coprime and $f(t) = f_n(t)/f_d(t)$. Let $F_1, \ldots, F_r$ be the irreducible factors of Φ_f over $K(t)$ and define

$$L_i := \{g(t) \in K(t) : F_i \mid \Phi_g\},\ i = 1, \ldots, r. \tag{5.17}$$

Theorems 5.11 e 5.12. *Let $f(t) \in K(t)$ and let $F_1, \ldots, F_r$ be the irreducible factors of Φ_f over $K(t)$. Then the set $\{L_1, \ldots, L_r\}$, with L_i as in* (5.17)*, is the set of principal subfields of $K(t)/K(f(t))$.*

We then used partitions to compute all intersections of the principal subfields, which significantly simplifies the computation of these intersections. A generator for each subfield L of $K(t)/K(f(t))$ is found using the following result.

Theorem 5.26. *Let $f(t) \in K(t)$ and let $F_1, \ldots, F_r$ be the irreducible factors of Φ_f over $K(t)$, with $F_1 = x - t$. Let $P_L = \{P^{(1)}, \ldots, P^{(s)}\}$ be the partition corresponding to the subfield L. Let $g := \prod_{i \in P^{(1)}} F_i$, with $1 \in P^{(1)}$ and let $c(t) \in K(t)$ be a non-constant coefficient of g. Then $L = K(c(t))$.*

These results allowed us to compute all decompositions of $f(t) \in K(t)$ in a more efficient way, both in theory and practice (see Corollary 5.25 and Tables 5.1 e 5.2). Moreover, when $f(t) \in K[t]$, our algorithm has better complexity than that present by Blankertz [10]. The implementation of this algorithm was included in the Computer Algebra System *Magma*. For more details, see Chapter 5.

[1]Here we can always assume that $\deg(f_n(x)) > \deg(f_d(x))$, which guarantees that Φ_f is monic.

FUTURE WORK

In what follows we mention a few ideas that might be worth investigating. The first idea is to improve the `Subfields` algorithm for the number field case. As we have seen in Table 4.2, for most cases, the bottleneck of the algorithm is the LLL computations (this was true even before our improvements). Even in cases where the number of subfields is large and the number of LLL calls is small (see ex. 1, 10 and 21 in Table 4.2), LLL dominates the CPU timings. Hence, it is important to "skip" as many factors as possible. Recently, Elsenhans and Klüners [18] presented an algorithm, based on Klüners [26] and van Hoeij *et al.* [51], that significantly reduces the number of LLL calls (and hence, might outrun our algorithm).

However, the algorithm presented in [18] does not generate an intersection-generating set of subfields, but rather a Galois-generating set of subfields (a much smaller set of subfields). If one wishes all subfields then more work has to be done (such as computing the intersection of wreath products). We believe that our algorithm, more specifically, the join of partitions, could be used together with this new algorithm to quickly find all subfields. This is motivated by the fact that the time for finding all subfields of ex.10 of Table 4.2 using the algorithm from [18] is about 339.92s, while our algorithm took 43.62s. However, this is one of the few examples where the number of LLL calls in both algorithms is the same.

One might also ask for a *minimal subfield factorization*. We have seen that we do not necessarily need the irreducible factorization of f, but a subfield factorization $f_1, \ldots, f_r$ of f, where the f_i are not necessarily irreducible. Hence, we might ask for a subfield factorization where the number of factors is minimal. By knowing some information about a minimal subfield factorization, one could also minimize the number of LLL calls.

For the rational function decomposition algorithm, one could ask for a generalization of this decomposition algorithm, presented in Chapter 5, for the multivariate case. Such generalization is not trivial, as in the multivariate case one loses the algebraic property of the extension. That is, if $K \subseteq L \subseteq K(x_1, \ldots, x_m)$, then $[K(x_1, \ldots, x_n) : L]$ might not be finite. This means that we could not directly apply the theory developed in Chapter 5, where we were working with extensions $K(t)/K(f(t))$, which always have finite degree. Moreover, the very own definition of decomposition has to be restated, as there are different ways into which we can decompose a multivariate polynomial (see [23]). However, we do believe that some generalization of the theory presented in Chapter 5 exists, because Theorem 5.8 can be generalized to the multivariate case (at least for some *types of decomposition*).

The rational function decomposition algorithm depends on the factorization of the bivariate polynomial $\nabla_f = f_n(x)f_d(y) - f_n(y)f_d(x) \in K[x, y]$. So far, we have used general algorithms for bivariate factorization. However, since ∇_f is a rather special bivariate polynomial, one might ask if there is a more efficient algorithm for factoring ∇_f. This would hopefully improve the algorithm complexity, specially in the polynomial decomposition algorithm, as factoring ∇_f dominates the complexity in this case. This could also improve timings, especially in characteristic zero, where most of the time is spent factoring ∇_f.

Referências

[1] M. Ajtai. The shortest vector problem in l2 is np-hard for randomized reductions (extended abstract). In *Proceedings of the Thirtieth Annual ACM Symposium on Theory of Computing*, STOC '98, pages 10–19, New York, NY, USA, 1998. ACM.

[2] V. S. Alagar and M. Thanh. *Fast polynomial decomposition algorithms*, pages 150–153. Springer Berlin Heidelberg, Berlin, Heidelberg, 1985.

[3] L. E. Allem, J. G. Capaverde, M. van Hoeij, and J. Szutkoski. Functional decomposition using principal subfields. In *Proceedings of the 2017 ACM on International Symposium on Symbolic and Algebraic Computation*, ISSAC '17, pages 421–428, New York, NY, USA, 2017. ACM.

[4] C. Alonso, J. Gutierrez, and T. Recio. A rational function decomposition algorithm by near-separated polynomials. *Journal of Symbolic Computation*, 19(6):527 – 544, 1995.

[5] M.-E. Alonso, E. Becker, M.-F. Roy, and T. Wörmann. Zeros, multiplicities, and idempotents for zero-dimensional systems. In *Algorithms in algebraic geometry and applications (Santander, 1994)*, volume 143 of *Progr. Math.*, pages 1–15. Birkhäuser, Basel, 1996.

[6] M. Ayad and P. Fleischmann. On the decomposition of rational functions. *Journal of Symbolic Computation*, 43(4):259 – 274, 2008.

[7] D. R. Barton and R. Zippel. Polynomial decomposition algorithms. *Journal of Symbolic Computation*, 1(2):159 – 168, 1985.

[8] K. Belabas. A relative van Hoeij algorithm over number fields. *J. Symbolic Comput.*, 37(5):641–668, 2004.

[9] D. Bini and V. Y. Pan. *Polynomial and Matrix Computations (Vol. 1): Fundamental Algorithms*. Birkhauser Verlag, Basel, Switzerland, Switzerland, 1994.

[10] R. Blankertz. A polynomial time algorithm for computing all minimal decompositions of a polynomial. *ACM Commun. Comput. Algebra*, 48(1/2):13–23, July 2014.

[11] W. Bosma, J. Cannon, and C. Playoust. The Magma algebra system. I. The user language. *Journal of Symbolic Computation*, 24(3-4):235–265, 1997. Computational algebra and number theory (London, 1993).

[12] A. Bostan, G. Lecerf, B. Salvy, E. Schost, and B. Wiebelt. Complexity issues in bivariate polynomial factorization. In *Proceedings of the 2004 International Symposium on Symbolic and Algebraic Computation*, ISSAC '04, pages 42–49, New York, NY, USA, 2004.

[13] D. Casperson and J. McKay. Symmetric functions, m-sets, and Galois groups. *Math. Comp.*, 63(208):749–757, 1994.

[14] H. Cohen and F. Diaz y Diaz. A polynomial reduction algorithm. *Sém. Théor. Nombres Bordeaux (2)*, 3(2):351–360, 1991.

[15] X. Dahan and É. Schost. Sharp estimates for triangular sets. In *Proceedings of the 2004 International Symposium on Symbolic and Algebraic Computation*, ISSAC '04, pages 103–110. ACM, New York, 2004.

[16] M. T. Dickerson. *The Functional Decomposition of Polynomials*. PhD thesis, Cornell University, Ithaca, NY, USA, 1989.

[17] J. D. Dixon. Computing subfields in algebraic number fields. *J. Austral. Math. Soc. Ser. A*, 49(3):434–448, 1990.

[18] A.-S. Elsenhans and J. Klüners. Computing subfields of number fields and applications to galois group computations. 2016. Preprint, available at `https://arxiv.org/abs/1610.06837`.

[19] R. Freese. Partition algorithms. unpublished notes, available at `http://www.math.hawaii.edu/~ralph/Notes/Partitions/partitions.pdf`, 1997.

[20] R. Freese. Computing congruences efficiently. *Algebra universalis*, 59(3):337–343, 2008.

[21] S. Gao. Factoring multivariate polynomials via partial differential equations. *Math. Comput.*, 72(242):801–822, Apr. 2003.

[22] M. W. Giesbrecht. Some results on the functional decomposition of polynomials. Master's thesis, University of Toronto, Toronto, Ontario, Canada, 1988.

[23] J. Gutierrez, R. Rubio, and D. Sevilla. On multivariate rational function decomposition. *Journal of Symbolic Computation*, 33(5):545 – 562, 2002.

[24] M. T. Heideman, D. H. Johnson, C. S. Burrus, and C. C. Truesdell. Gauss and the history of the fast fourier transform, 1985.

[25] A. Hulpke. Block systems of a Galois group. *Experiment. Math.*, 4(1):1–9, 1995.

[26] J. Klüners. On computing subfields. A detailed description of the algorithm. *J. Théor. Nombres Bordeaux*, 10(2):243–271, 1998.

[27] J. Klüners. *The van Hoeij Algorithm for Factoring Polynomials*, pages 283–291. Springer Berlin Heidelberg, Berlin, Heidelberg, 2010.

[28] J. Klüners and M. Pohst. On computing subfields. *Journal of Symbolic Computation*, 24(3):385 – 397, 1997.

[29] D. Kozen and S. Landau. Polynomial decomposition algorithms. *Journal of Symbolic Computation*, 7(5):445 – 456, 1989.

[30] S. Landau and G. L. Miller. Solvability by radicals is in polynomial time. In *Proceedings of the Fifteenth Annual ACM Symposium on Theory of Computing*, STOC '83, pages 140–151, New York, NY, USA, 1983. ACM.

[31] S. Lang. *Algebra*. Springer-Verlag New York, 3rd edition, 2002.

[32] D. Lazard and A. Valibouze. Computing subfields: reverse of the primitive element problem. In *Computational algebraic geometry (Nice, 1992)*, volume 109 of *Progr. Math.*, pages 163–176. Birkhäuser Boston, Boston, MA, 1993.

[33] F. Le Gall. Powers of tensors and fast matrix multiplication. In *Proceedings of the 39th International Symposium on Symbolic and Algebraic Computation*, ISSAC '14, pages 296–303, New York, NY, USA, 2014. ACM.

[34] G. Lecerf. Improved dense multivariate polynomial factorization algorithms. *Journal of Symbolic Computation*, 42(4):477 – 494, 2007.

[35] A. K. Lenstra, H. W. Lenstra, Jr., and L. Lovász. Factoring polynomials with rational coefficients. *Math. Ann.*, 261(4):515–534, 1982.

[36] M. Mignotte. Some useful bounds. In *Computer algebra*, pages 259–263. Springer, Vienna, 1983.

[37] M. Mignotte. An inequality about irreducible factors of integer polynomials. *Journal of Number Theory*, 30(2):156 – 166, 1988.

[38] H. Niederreiter and R. Göttfert. On a new factorization algorithm for polynomials over finite fields. *Mathematics of Computation*, 64(209):347–353, 1995.

[39] J. F. Ritt. Prime and composite polynomials. *Transactions of the American Mathematical Society*, 23(1):51–66, 1922.

[40] X.-F. Roblot. Polynomial factorization algorithms over number fields. *Journal of Symbolic Computation*, 38(5):1429 – 1443, 2004.

[41] A. Schinzel. Arbitrary polynomials over an arbitrary field. In *Polynomials with Special Regard to Reducibility:*, pages 12–91. Cambridge University Press, Cambridge, 004 2000.

[42] I. Stewart. *Galois Theory*. Chapman & Hall, 3rd edition, 2003.

[43] H. Stichtenoth. *Algebraic Function Fields and Codes*. Springer Publishing Company, Incorporated, 2nd edition, 2008.

[44] J. Szutkoski and M. van Hoeij. The complexity of computing all subfields of an algebraic number field. *Journal of Symbolic Computation*, 2016. To appear.

[45] The PARI Group. *PARI/GP version* `2.9.2`. Univ. Bordeaux, 2016. Available from `http://pari.math.u-bordeaux.fr/`.

[46] The Sage Developers. *SageMath, the Sage Mathematics Software System (Version 7.6)*, 2017. Available from `http://www.sagemath.org`.

[47] B. M. Trager. Algebraic factoring and rational function integration. In *Proceedings of the Third ACM Symposium on Symbolic and Algebraic Computation*, SYMSAC '76, pages 219–226, New York, NY, USA, 1976. ACM.

[48] B. L. van Der Waerden. *Modern Algebra*. Fredrick Ungar Publishing Co., New York, NY, 1964.

[49] M. van Hoeij. Factoring polynomials and the knapsack problem. *Journal of Number Theory*, 95(2):167 – 189, 2002.

[50] M. van Hoeij. The complexity of factoring univariate polynomials over the rationals. ISSAC'13 Tutorial, 2013. Available at `http://www.math.fsu.edu/~hoeij/papers/issac13/slides.pdf`.

[51] M. van Hoeij, J. Klüners, and A. Novocin. Generating subfields. *Journal of Symbolic Computation*, 52:17–34, 2013.

[52] M. van Hoeij and M. Monagan. A modular GCD algorithm over number fields presented with multiple extensions. In *Proceedings of the 2002 International Symposium on Symbolic and Algebraic Computation*, pages 109–116. ACM, New York, 2002.

[53] M. van Hoeij and A. Novocin. Gradual sub-lattice reduction and a new complexity for factoring polynomials. *Algorithmica*, 63(3):616–633, 2012.

[54] M. van Hoeij and V. Pal. Isomorphisms of algebraic number fields. *J. Théor. Nombres Bordeaux*, 24(2):293–305, 2012.

[55] J. von zur Gathen. Functional decomposition of polynomials: the tame case. *Journal of Symbolic Computation*, 9(3):281 – 299, 1990.

[56] J. von zur Gathen. Functional decomposition of polynomials: the wild case. *Journal of Symbolic Computation*, 10(5):437 – 452, 1990.

[57] J. von zur Gathen and J. Gerhard. *Modern Computer Algebra*. Cambridge University Press, New York, NY, USA, 2 edition, 2003.

[58] J. von zur Gathen and D. Panario. Factoring polynomials over finite fields: A survey. *Journal of Symbolic Computation*, 31(1-2):3 – 17, 2001.

[59] S. Weinzierl. Computer algebra in particle physics. 2002. Available at `arXiv:hep-ph/0209234`.

[60] R. Zippel. Rational function decomposition. In *Proceedings of the 1991 International Symposium on Symbolic and Algebraic Computation*, ISSAC '91, pages 1–6, New York, NY, USA, 1991.