Analysis and Implementation of Algorithms for Noncommutative Algebra

Craig A. Struble

Dissertation submitted to the faculty of the
Virginia Polytechnic Institute and State University
in partial fulfillment of the requirements for the degree of

Doctor of Philosophy
in
Computer Science and Applications

Lenwood S. Heath, Co-chairman
Edward L. Green, Co-chairman
Donald C. S. Allison
Daniel R. Farkas
Sanjay Gupta

April 24, 2000
Blacksburg, Virginia

Analysis and Implementation of Algorithms for Noncommutative Algebra

by

Craig A. Struble

Committee Co-chairmen:

Lenwood S. Heath

Computer Science

and

Edward L. Green

Mathematics

(ABSTRACT)

A fundamental task of algebraists is to classify algebraic structures. For example, the classification of finite groups has been widely studied and has benefited from the use of computational tools. Advances in computer power have allowed researchers to attack problems never possible before.

In this dissertation, algorithms for *noncommutative* algebra, when ab is not necessarily equal to ba, are examined with practical implementations in mind. Different encodings of *associative algebras* and *modules* are also considered. To effectively analyze these algorithms and encodings, the *encoding neutral* analysis framework is introduced. This framework builds on the ideas used in the arithmetic complexity framework defined by Winograd. Results in this dissertation fall into three categories: analysis of algorithms, experimental results, and novel algorithms.

Known algorithms for calculating the Jacobson radical and Wedderburn decomposition of associative algebras are reviewed and analyzed. The algorithms are compared experimentally and a recommendation for algorithms to use in computer algebra systems is given based on the results.

A new algorithm for constructing the Drinfel'd double of finite dimensional Hopf algebras is presented. The performance of the algorithm is analyzed and experiments are performed to demonstrate its practicality. The performance of the algorithm is elaborated upon for the special case of group algebras and shown to be very efficient.

The MeatAxe algorithm for determining whether a module contains a proper submodule is reviewed. Implementation issues for the MeatAxe in an encoding neutral environment are discussed.

A new algorithm for constructing endomorphism rings of modules defined over path algebras is presented. This algorithm is shown to have better performance than previously used algorithms.

Finally, a linear time algorithm, to determine whether a quotient of a path algebra, with a known Gröbner basis, is finite or infinite dimensional is described. This algorithm is based on the Aho-Corasick pattern matching automata. The resulting automata is used to efficiently determine the dimension of the algebra, enumerate a basis for the algebra, and reduce elements to normal forms.

To my grandparents, whose wisdom, faith, love, and charm have carried me through the toughest of times.

ACKNOWLEDGEMENTS

First, I thank my advisors, Dr. Lenny Heath and Dr. Ed Green, for their help formulating research problems, answering the dumbest of questions, and support throughout. Lenny's willingness to endure my writing goes beyond any thanks I can give. Next, I thank my committee members, Dr. Donald Allison, Dr. Daniel Farkas, and Dr. Sanjay Gupta, for weathering the challenging nature of my research. I especially thank Dr. Gupta for making the trip from northern Virginia to be at my defense. I also thank Dr. Cal Ribbens for serving on my proposal committee. For providing me with funding and research opportunities, I thank Dr. Jack Carroll, Dr. Ed Green, Dr. Rex Hartson, Dr. Lenny Heath, Dr. Verna Schuetz, and Dr. Cliff Shaffer. I gratefully acknowledge National Science Foundation grant CCR-9732068 for partially funding my research. I also thank Randall Smith at Sun Microsystems for providing me with a wonderful internship. The Computer Science Theory group, Ben Keller, Scott Guyer, and John Paul Vergara, participated in several thoughtful discussions. In particular, Ben provided help with Gröbner bases, path algebras, and modules. I am grateful to the GAP team for their helpful responses to several questions. Steve Linton and Günter Pilz provided funding for my attendance in the 1999 GAP workshop. Willem de Graaf and Andrew Solomon of the GAP team provided valuable correspondence and support. I also thank Wayne Eberly and Gábor Ivanyos for answering questions and sending preprints of recent work. Several bright students contributed to HOPF: Nick Allen, Gerard Brunick, Greg Taylor, Tim Terriberry, and George Yuhasz. Brian Amento, James "Bo" Begole, Pete DePasquale, Philip Isenhour, Christina Van Metre, and Patrick Van Metre helped me survive the rough times. The Isle of Jura will never be the same. I particularly thank Bo for his advice and continued friendship. My family has provided encouragement, faith, love, and support throughout. Finally, I thank my wife Cara. She has had to live with my mood swings, long nights, and excuses for too long. Her love, encouragement, and understanding have been key to my success. Cara, the end of this long journey has finally arrived.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

Chapter 1

Introduction

A fundamental task of algebraists is to classify algebraic structures. For example, the classification of finite groups has been widely studied and has benefited from the use of computational tools. For example, the construction of *The Atlas of Finite Groups* [18] made extensive use of the MeatAxe of Parker [80]. The availability of computer algebra systems such as Axiom [62], GAP [42], and Magma [12] is evidence of the importance of computer tools supporting algebra.

We separate the task of classification into three subtasks: decomposition, identification, construction. *Decomposition* is the process of finding substructures in an algebraic structure. An algebraic structure is often completely described by the substructures it contains. The substructures that an algebraic structure contains are generally simpler to understand and easier to describe. *Identification* is the determination of properties that an algebraic structure has. Properties of interest include whether or not a structure is finite, whether it is isomorphic to another structure, or if an operation on the structure is associative or commutative. Properties partition structures into classes. *Construction* is the concrete realization of algebraic structures. Construction may be accomplished by the selection of a random instance of a structure. Alternatively, an instance is used to realize another, related instance of an algebraic structure. Construction is used to build examples and may also be helpful in the decomposition and identification subtasks.

Our goal is to develop and analyze practical computer algorithms to assist algebraists in the classification of algebraic structures. We further hope that as the number of applications of algebra increases, the tools introduced here will be useful for these new applications.

1.1 Statement of Problems

In this section, we state the major problems covered in this dissertaiton. The underlying theme of this dissertation is the analysis of algebraic algorithms, specifically for noncommutative algebra. We begin this section by summarizing desirable features of analysis lacking in previous literature. The remainder of this section presents the three motivating algebraic problems we study in this dissertation.

1.1.1 Algorithm Analysis

The analysis of algebraic algorithms found in previous work has been unsatisfactory. In early work by Friedl and Rónyai [39, 86], it was sufficient to state that a problem is polynomial-time solvable. No explicit statement of the rate of growth or the parameters affecting the algorithm is provided. No explicit insight is stated for the dominating operations as well.

More recently, Eberly and Giesbrecht [35] and Ivanyos [58, 59] include rigorous analyses of the algorithms they developed; however, their analyses rely on a particular encoding of input that is not practical for even modest sized problem instances. Although the algorithms can be reformulated to accept different input encodings, it is unclear how changes in encoding affect the performance of their algorithms.

We wish to better understand the performance of existing algorithms through more rigorous analysis. The analysis needs to determine the effects of changing the input encodings as well as the dominating time and space costs of each algorithm analyzed. To meet these goals, we introduce *encoding neutral* algorithm analysis in Chapter 3. This analysis framework is used throughout to analyze the algorithms appearing in this dissertation. The next three subsections present an overview of the three algebraic problems we analyze.

In addition to the analysis, we have implemented many of the algorithms, experimentally verifying the analysis performed. When the analysis and experimental results do not match, we present potential causes for the differences.

1.1.2 Algebra Decomposition

An *algebra* A is a vector space with a compatible ring structure. A *decomposition* $\{A_i\}_{i=1}^n$ of A is a set of (two-sided) ideals in A such that

$$A \quad = \quad A_1 \oplus A_2 \oplus \cdots \oplus A_n.$$

If there exists a decomposition of A such that $n \geq 2$, then A is *decomposable*, otherwise A is *indecomposable*. If A is finite dimensional, the decomposition of A is complete if each A_i in the decomposition is indecomposable as an algebra. Our first algebraic problem is stated formally as follows:

ALGEBRA DECOMPOSITION
INSTANCE: A (finite dimensional) algebra A.
SOLUTION: Two-sided ideals $A_1, \ldots, A_n$ of A, such that $A = A_1 \oplus \cdots \oplus A_n$.

For a particular class of (finite dimensional) algebras, the decomposition can be refined to provide further information about each algebra. Let A be an algebra. The *radical* of A is the intersection of all maximal right ideals in A. We say A is *semisimple* if the radical of A is the trivial ideal 0; i.e., it contains only 0_A. An algebra is *simple* if it does not contain any proper two-sided ideals. By the Wedderburn structure theorem, if A is semisimple and

$$A \quad = \quad A_1 \oplus \cdots \oplus A_n$$

is a decomposition into indecomposable ideals A_i of A, then each A_i is a simple algebra. Furthermore, by the Wedderburn-Artin theorem, every simple algebra is isomorphic as an algebra to a full matrix algebra $M_n(D)$, where D is a division ring.

In the case where A is semisimple, we state the problem of algebra decomposition as follows:

SEMISIMPLE ALGEBRA DECOMPOSITION
INSTANCE: A (finite dimensional) semisimple algebra A.
SOLUTION: Two-sided ideals $A_1, \ldots, A_n$ of A, such that $A = A_1 \oplus \cdots \oplus A_n$. Integers n_i, division rings D_i, and maps $\rho_i : A_i \rightarrow M_{n_i}(D_i)$, such that each ρ_i is an algebra isomorphism from A_i to the full matrix algebra $M_{n_i}(D_i)$.

In some circumstances, it is computationally difficult to find a semisimple algebra decomposition. We elaborate on this in Chapter 5.

1.1.3 Drinfel'd Double

A *Hopf algebra* is a compatible algebra and coalgebra with a generalized inverse map called the *antipode*. Hopf algebras arise in many areas, including knot theory, quantum physics [63, 68, 69, 70, 71, 72, 79], and algebraic geometry [73]. The *Drinfel'd double* is a construction that takes a Hopf algebra and constructs a new, larger Hopf algebra with special properties. This construction involves a twisting of the multiplication in the original Hopf algebra. A detailed description of Hopf algebras and the Drinfel'd double construction is presented in Chapter 6. Formally, the problem is stated as follows:

DRINFEL'D DOUBLE
INSTANCE: A Hopf algebra H.
SOLUTION: The Drinfel'd double $D(H)$ of H.

1.1.4 Module Decomposition

A *module* M is a set of elements acted on by a ring R. The concept of a module generalizes the concept of a vector space, where scalars are taken from a ring instead of a field. A *decomposition* of M is a set $\{M_i\}_{i=1}^{n}$ of submodules of M such that

$$M \quad = \quad M_1 \oplus M_2 \oplus \cdots \oplus M_n.$$

We say that M is *decomposable* if there exists a decomposition of M such that $n \geq 2$ and *indecomposable* otherwise.

Let $M_1 \oplus \cdots \oplus M_n$ be a decomposition of M such that each M_i is indecomposable as a module. By the Krull-Schmidt theorem, all such decompositions are unique up to isomorphism; that is, if $M_1' \oplus \cdots \oplus M_n'$ is also a decomposition of M such that each M_i' is indecomposable, then the summands $\{M_i'\}_{i=1}^{n}$ can be reordered so that $M_i \cong M_i'$, for each $1 \leq i \leq n$. The problem of module decomposition is stated formally as follows:

MODULE DECOMPOSITION
INSTANCE: A module M.
SOLUTION: Submodules $M_1, \ldots, M_n$ of M such that $M = M_1 \oplus \cdots \oplus M_n$.

The problem **MODULE DECOMPOSITION** is to the problem **ALGEBRA DECOMPOSITION** in Chapter 8.

1.2 Overview of Results

The results in this dissertation fall into three categories: analysis of algorithms, experimental results, and new algorithmic approaches.

In Chapter 3, we introduce the *encoding neutral* algorithm analysis framework used throughout this dissertation. The framework is a generalization of the arithmetic complexity framework introduced by Winograd [98]. In our analysis, we elaborate on the time complexity of operations used during a computation.

In Chapter 4, we include a detailed worst case time and space analysis for three different encodings of algebras. Included is a comparative analysis of algorithms for calculating the center of an algebra, the set of elements which commute with every element in the algebra. We review specialized algorithms to calculate the center for multiplication table and group algebra encodings and experimentally show that these algorithms are more efficient than the general algorithm. We also obtain bounds on the space required by each algebra encoding.

In Chapter 5, we analyze several algorithms [24, 39, 86, 32, 33, 34, 35, 17, 58, 59] related to **ALGEBRA DECOMPOSITION** and **SEMISIMPLE ALGEBRA DECOMPOSITION**. We have analyzed these algorithms within a single framework for the first time. Pseudocode for several of the algorithms appears for the first time, and all have been modified to work in encoding neutral environments, such as the computer algebra system GAP [42]. We augment the analysis with experimental data to validate the analysis results obtained. One surprising result is the performance of Davis' algorithm for computing primitive central idempotents [24]. In our experiments, Davis' algorithm performs better than the other algorithms presented in most cases. With the exception of calculating the Jacobson radical of an algebra, we show that the multiplication table encoding for algebras performs best overall. When calculating the Jacobson radical, a matrix algebra encoding is the best used for performance, but is not space efficient. Based on our results, we summarize the chapter with recommended algorithms to use for **ALGEBRA DECOMPOSITION** and **SEMISIMPLE ALGEBRA DECOMPOSITION** when given an algebra defined over the finite fields and the rationals.

In Chapter 6, we present a new algorithm for **DRINFEL'D DOUBLE**. We employ simple caching techniques and demonstrate that these caching techniques greatly improve the performance of our basic algorithm, both theoretically and experimentally. We also analyze the cost to calculate the Drinfel'd double for group algebras, which we show to be much more efficient than the general

case of Hopf algebras.

In Chapter 8, we briefly review algorithms for **MODULE DECOMPOSITON**, and present a new algorithm for computing endomorphism rings of right modules defined over path algebras. We experimentally show that our algorithm has better rate of growth than the algorithm used in Magma [12], a commercial computer algebra system. Furthermore, our algorithm is deterministic, in contrast to algorithms used in practice.

As a standalone result, we present a new algorithm to determine whether a path algebra factored by an ideal, with a known (finite) Gröbner basis, is finite or infinite dimensional in Chapter 9. We show this algorithm to have linear time and space cost, which is an improvement over previously known algorithms.

Except where noted, the algorithms contained in this dissertation are implemented using GAP version 4.1 [42]. The implementations of recommended algorithms are included in GAP and Hopf, a GAP share package providing additional support for noncommutative algebra. Hopf is being developed at Virginia Tech.

1.3 Organization

The remainder of this dissertation is structured as follows. Chapter 2 reviews algebra terminology used throughout this dissertation. We introduce the encoding neutral analysis framework used to analyze the algorithms contained in this dissertation in Chapter 3. We describe and analyze different input encodings for algebras in Chapter 4. In Chapter 5, we analyze and present experimental data about algorithms for decomposing algebras. We present a new algorithm for constructing the Drinfel'd double of finite dimensional Hopf algebras in Chapter 6. We turn to describing and analyzing encodings for modules in Chapter 7. In Chapter 8, we analyze and present experimental data for algorithms related to the structure of modules. In Chapter 9, we present a new algorithm for determining if a quotient of a path algebra is finite or infinite dimensional. We summarize our results and present directions for future research in Chapter 10.

Chapter 2

Terminology

In this chapter, we present the basic terminology used throughout. We assume the reader is familiar with linear algebra. The linear algebra terminology used can be found in standard texts such as Curtis [20] and Anton [7]. The remaining terminology is standard and can be found in standard texts on algebra such as [21, 31, 36, 57, 94, 95].

2.1 Groups

Let G be a set. A *binary operation* $\cdot$, called multiplication, on G is a function $\cdot : G \times G \to G$. For $a, b \in G$, we write multiplication $\cdot(a, b)$ as $a \cdot b$. Multiplication is *associative* if $(a \cdot b) \cdot c = a \cdot (b \cdot c)$ for all $a, b, c \in G$. Multiplication is *commutative* if $a \cdot b = b \cdot a$ for all $a, b \in G$. If $H \subseteq G$, and if $a \cdot b \in H$ for all $a, b \in H$, then H is *closed* under multiplication.

A *group* is an ordered pair $(G, \cdot)$ satisfying the following three axioms:

1. Multiplication $\cdot$ an associative binary operation on G.
2. There exists an element $e \in G$ such that $e \cdot a = a \cdot e = a$ for all $a \in G$ (e is an *identity* of G).
3. For each $a \in G$, there exists an element $a^{-1} \in G$, such that $a \cdot a^{-1} = a^{-1} \cdot a = e$ (a^{-1} is an *inverse* of a).

If multiplication is a commutative operation on G, then $(G, \cdot)$ is an *abelian group*. We say G is a group under multiplication $\cdot$ if $(G, \cdot)$ is a group. Furthermore, if the operation is clear from context,

we simply say G is a group.

Example 2.1.1. The integers $\mathbb{Z}$, the rational numbers $\mathbb{Q}$, the real numbers $\mathbb{R}$, and the complex numbers $\mathbb{C}$ are groups under the binary operation $+$ with the identity $e = 0$ and inverses $a^{-1} = -a$ for all a. Furthermore, all are abelian groups.

Example 2.1.2. Let $\Omega = \{1, \ldots, n\}$, the non-empty set of positive integers from 1 to n. Recall that a *permutation* of Ω is a bijection from Ω to itself. Let S_n be the set of all permutations of Ω. Let $\circ$ denote function composition. If $\pi : \Omega \to \Omega$ and $\rho : \Omega \to \Omega$ are permutations, then so is $\pi \circ \rho$. The set S_n is a group under function composition $\circ$ with the identity $e = \mathrm{id}$, where $\mathrm{id}(x) = x$, and a^{-1} is just the inverse map of a (which exists because a is a bijection). The group S_n is the *symmetric group of degree n* and its cardinality is $|S_n| = n!$. The symmetric group is an abelian group for $n = 1$ or $n = 2$. For $n \geq 3$, the symmetric group is nonabelian.

Example 2.1.3. Let $\mathbb{K}$ be a field (fields are formally defined later) and let

$$GL_n(\mathbb{K}) \quad = \quad \{A \,|\, A \text{ is an } n \times n \text{ matrix with entries from } \mathbb{K} \text{ and } \det(A) \neq 0\},$$

the set of $n \times n$ matrices with entries from $\mathbb{K}$ that have nonzero determinant. Matrix multiplication is an associative operation. Also, since $\det(AB) = \det(A) \cdot \det(B)$, the set $GL_n(\mathbb{K})$ is closed under matrix multiplication. Every matrix in $GL_n(\mathbb{K})$ has an inverse because $\det(A) \neq 0$. Thus, $GL_n(\mathbb{K})$ is a group under matrix multiplication, with the identity matrix as the identity element and every A in $GL_n(\mathbb{K})$ having its normal matrix inverse as its inverse. The group $GL_n(\mathbb{K})$ is the *general linear group of degree n*.

If H is a subset of G such that H is closed under multiplication and the taking of inverses then H is a *subgroup* of G.

Example 2.1.4. Let $G = \left\{ \begin{pmatrix} a & b \\ 0 & c \end{pmatrix} \,|\, a, b, c \in \mathbb{R}, a \neq 0, c \neq 0 \right\}$. Clearly, G is a subset of $GL_2(\mathbb{R})$ and closed under multiplication. It is left to the reader to verify that G is closed under inverses. Then, G is a subgroup of $GL_2(\mathbb{R})$.

2.2 Rings

Let R be a set and let $+ : R \times R \to R$ and $\cdot : R \times R \to R$ (called addition and multiplication) be binary operations on R. The ordered triple $(R, +, \cdot)$ is a *ring* if it satisfies the following three

axioms:

1. $(R, +)$ is an abelian group.
2. Multiplication $\cdot$ is an associative operation.
3. These *distributive laws* hold in $(R, +, \cdot)$: $(a+b)\cdot c = (a\cdot c)+(b\cdot c)$ and $a\cdot(b+c) = (a\cdot b)+(a\cdot c)$ for all $a, b, c \in R$

The identity of $(R, +)$ is the *additive identity* or *zero*, denoted 0_R. The inverse of $a \in R$ in the group $(R, +)$ is the *additive inverse*, denoted $-a$. If multiplication is commutative, then $(R, +, \cdot)$ is a *commutative ring*. If there exists an element $1_R \in R$ such that $1_R \cdot a = a \cdot 1_R = a$ for all $a \in R$, then $(R, +, \cdot)$ is a *ring with one* and 1_R is the *multiplicative identity* or *one* of $(R, +, \cdot)$. If $(R, +, \cdot)$ is a ring with one, and if $a, b \in R$ satisfy $a \cdot b = b \cdot a = 1_R$, then b is a *multiplicative inverse* of a and vice versa.

Theorem 1.10 in Chapter 3 of Hungerford [57] states that every ring can be embedded in a ring with one. Unless otherwise stated, we assume that every ring is a ring with one as a result of this theorem.

If $(R, +, \cdot)$ is a ring, we say R is a ring under addition $+$ and multiplication $\cdot$, and we say R is a ring when the operations are clear from the context. We typically write the product $a \cdot b$ as ab for conciseness.

Example 2.2.1. The integers $\mathbb{Z}$, rational numbers $\mathbb{Q}$, real numbers $\mathbb{R}$, and complex numbers $\mathbb{C}$ are all rings under addition and multiplication. Furthermore, all are commutative rings with one.

Example 2.2.2. Let $\mathbb{K}$ be a field and let $M_n(\mathbb{K})$ be the set of all $n \times n$ matrices with entries from $\mathbb{K}$. The set $M_n(\mathbb{K})$ is a ring under matrix addition and matrix multiplication. The ring $M_n(\mathbb{K})$ has a multiplicative identity $1_{M_n(\mathbb{K})}$, the $n \times n$ identity matrix. Thus, $M_n(\mathbb{K})$ is a ring with one.

Example 2.2.3. Let R be a commutative ring with one and let x be in indeterminate in R. The set of polynomials in x with coefficients from R is denoted $R[x]$. The set $R[x]$ is a commutative ring with one under addition and multiplication of polynomials. The multiplicative identity for $R[x]$ is the constant polynomial 1_R.

A subset S of R is a *subring* if S is closed under addition and multiplication. Furthermore, if R is a ring with one, then S must contain 1_R.

Example 2.2.4. The *center* $Z(R)$ of a ring R is defined by

$$Z(R) \quad = \quad \{a \in R \,|\, ar = ra \text{ for all } r \in R\},$$

the set of elements in R that commute with every element of R. The center $Z(R)$ is a subring of R.

Example 2.2.5. The set of constant polynomials in a polynomial ring $R[x]$ is a subring of $R[x]$.

Let R be a ring with one such that $1_R \neq 0_R$. The *characteristic* of R is the smallest positive integer n such that $1_R + 1_R + \cdots + 1_R = 0_R$, where 1_R is added n times. If no such integer exists, R has characteristic 0. If there exists a multiplicative inverse for each nonzero element of R, then R is a *division ring*. If R is a division ring that is also a commutative ring, then R is a *field*.

Example 2.2.6. The rings $\mathbb{Q}$, $\mathbb{R}$, $\mathbb{C}$ are all fields, where $1/a$ is the multiplicative inverse of $a \neq 0$. Each has characteristic 0.

Example 2.2.7. The equivalence classes for $\mathbb{Z}$ modulo p, where p is prime forms a field under modular arithmetic. We denote this field $\mathbb{F}_p$. In general, a unique finite field of order $q = p^n$ exists, for each prime p and positive integer n. See Dummit and Foote [31, Sec. 14.3] for more details. The characteristic of $\mathbb{F}_q$ is p.

Example 2.2.8. The polynomial ring $R[x]$ is *not* a field because polynomials do not have an inverse in general. For example, the polynomial x does not have an inverse existing in $R[x]$.

2.3 Ideals

Let R be a ring. Let I be a subgroup of the additive group of R. We call I a *left ideal* if $ri \in I$ for all $r \in R$ and $i \in I$. We call I a *right ideal* if $ir \in I$ for all $r \in R$ and $i \in I$. We call I a *two-sided ideal*, or simply *ideal*, if I is both a left and right ideal.

Example 2.3.1. Let $I = \{0_R\}$. Clearly, I is an ideal. The ideal I is the *trivial ideal*, and is written 0.

Example 2.3.2. Let R be a ring with one. Let $X \subseteq R$ be a set of elements from R. The set $\langle X \rangle = \{r_1 x_1 r_1' + \cdots + r_n x_n r_n' \,|\, r_i, r_i' \in R, x_i \in X, 1 \leq n \in \mathbb{Z}\}$ of finite sums of elements of the form rxr', with $r, r' \in R$ and $x \in X$, is the *two-sided ideal generated by* X. The ideal $\langle X \rangle$ is also the smallest ideal of R containing X.

Example 2.3.3. Let $R[x]$ be a polynomial ring. Let $I = \{\text{polynomials with zero constant term}\}$. Let $f \in R[x]$ be a polynomial and let $h, h' \in I$ be polynomials with a zero constant term. Clearly, fh and hf are both polynomials with zero constant term, and $h + h'$ is also a polynomial with zero constant term. Hence, I is an ideal in $R[x]$.

An ideal I is a *maximal ideal* of R if $I \neq R$ and if J is an ideal satifying $I \subseteq J \subseteq R$, then $J = I$ or $J = R$. Similarly, an ideal I is a *minimal ideal* of R if $I \neq 0$ and if J is an ideal satisfying $0 \subseteq J \subseteq I$, then $J = 0$ or $J = I$. The above definitions are used for left and right ideals as well.

Let I be an ideal of R. The *quotient ring* $R/I = \{r + I \,|\, r \in R\}$ inherits a ring structure defined by $(r + I) + (s + I) = (r + s) + I$ and $(r + I)(s + I) = rs + I$ for $r, s \in R$.

An element $r \in R$ is *nilpotent* of $r^n = 0$ for some positive integer n. An ideal I is *nilpotent* if $I^n = 0$ for some positive integer n.

A ring R is called *simple* if the only ideals of R are 0 and R; that is, R does not contain any proper ideals.

2.4 Modules

Let R be a ring. A *right R-module* is an abelian group $(M, +)$, with an operation called multiplication $\cdot : M \times R \to M$, denoted by $m \cdot r \mapsto mr$ for $r \in R, m \in M$, such that

$$\begin{aligned} m(r + s) &= mr + ms \\ (m + n)r &= mr + nr \\ m(rs) &= (mr)s \end{aligned}$$

for all $r, s \in R$ and $m, n \in M$. If R is a ring with one, then we require the additional property $m1_R = m$ for all $m \in M$. The identity of the abelian group $(M, +)$ is the *additive identity* of M and is written 0_M.

A left R-module is similarly defined by changing multiplication to $\cdot : R \times M \to M$, denoted $r \cdot m \mapsto rm$, and the analogous properties that hold for a right module. Unless otherwise stated, we use the term R-module for right R-modules.

Example 2.4.1. Let $M = \{0_M\}$ be an R-module. Clearly, multiplication is defined as $0_M r = 0_M$ for each $r \in R$. The module M is called the *trivial module* and is written as 0.

Example 2.4.2. A right ideal of a ring R is a right R-module. In particular, R is a R-module.

Example 2.4.3. The cartesian product $R^n = R \times \cdots \times R$ is a right R-module via componentwise multiplication by elements in R on the right. The module R^n is called the *free module of rank n*.

A subset N of a module M is a *submodule* of M if N is a subgroup of M and if $nr \in N$ for all $r \in R$ and $n \in N$. An R-module M is *reducible* if M contains a proper submodule. Otherwise, M is *irreducible* (or *simple*).

For a submodule N, the *quotient module* $M/N = \{m + N \,|\, m \in M\}$ inherits an R-module structure given by $(m + N)r = (mr + N)$.

If R is a field, then M is an *R-vector space*. In general, we view vector spaces as left modules instead of right modules.

2.5 Products

Let R and S be rings. The cartesian product $R \times S = \{(r, s) \,|\, r \in R, s \in S\}$ is a ring via componentwise addition and multiplication. The ring $R \times S$ is the *product* of the rings R and S.

Similarly, the product $M \times N$ of R-modules M and N is $\{(m, n) \,|\, m \in M, n \in N\}$ with componentwise addition and the action of R is $(m, n) \cdot r = (mr, nr)$. The product of modules coincides with the definition of *external direct sum* for finite sets of R-modules (see Farb and Dennis [36, Sec. 0]). Hence, we use the more standard notation $M \oplus N$ when constructing the product of modules.

Let M_1 and M_2 be submodules of a module M. The *sum* $M_1 + M_2$ of M_1 and M_2 is

$$M_1 + M_2 \quad = \quad \{m_1 + m_2 \,|\, m_1 \in M_1, m_2 \in M_2\}.$$

Clearly, $M_1 + M_2$ is a submodule of M and is the smallest submodule of M containing M_1 and M_2. The *intersection* $M_1 \cap M_2$ is the largest submodule of M contained in both M_1 and M_2.

Let $M_1, M_2, \ldots, M_n$ be submodules of M. The module M is the *(internal) direct sum* of $M_1, M_2, \ldots, M_n$, denoted $M = M_1 \oplus \cdots \oplus M_n$, if

1. $M = M_1 + \cdots + M_n$, and
2. $M_i \cap (M_1 + \cdots + M_{i-1} + M_{i+1} + \cdots + M_n) = 0$ for each i.

Although we use the same notation for internal and external direct sums, the proper interpretation is generally clear from context.

The next product is formally defined as the quotient of groups (see [21, Sec. 12], [57, Ch. 4, Sec. 5]). We take a less formal approach and highlight the important properties. Let M be a right R-module and let N be a left R-module. The *tensor product* $M \otimes_R N$ consists of elements $m \otimes n$, where $m \in M$ and $n \in N$. The elements of $M \otimes_R N$ satisfy the properties

1. $(m_1 + m_2) \otimes n = m_1 \otimes n + m_2 \otimes n$,
2. $m \otimes (n_1 + n_2) = m \otimes n_1 + m \otimes n_2$,
3. $mr \otimes n = m \otimes rn$,

for all $m, m_i \in M$, $n, n_i \in N$ and $r \in R$. When R is a field, $M \otimes_R N$ forms a vector space with scalar multiplication $r(m \otimes n) = rm \otimes n = m \otimes rn$, $m \in M$, $n \in N$, and $r \in R$.

We can form the tensor products of homomorphisms as well as modules. Let $f : M \to M'$ and $g : N \to N'$ be R-module homomorphisms where M and M' are right R-modules and N and N' are left R-modules. Define $f \otimes g : M \otimes_R N \to M' \otimes_R N'$ by

$$(f \otimes g)(m \otimes n) \quad = \quad f(m) \otimes g(n).$$

The map $f \otimes g$ is a (group) homomorphism.

2.6 Algebras

Let $\mathbb{K}$ be a field. An *associative* $\mathbb{K}$*-algebra* A is a ring and a $\mathbb{K}$-vector space such that ring multiplication and scalar multiplication are compatible; that is, $\alpha(ab) = (\alpha a)b = a(\alpha b)$ for all $\alpha \in \mathbb{K}$ and $a, b \in A$. Throughout, we drop the term associative and use *algebra* to indicate an associative algebra.

Example 2.6.1. The set of $n \times n$ matrices $M_n(\mathbb{K})$ over a field $\mathbb{K}$ is a $\mathbb{K}$-algebra. Ring addition and multiplication is matrix addition and multiplication. Scalar multiplication is the obvious scalar multiplication on matrices. It is straightforward to check that ring multiplication and scalar multiplication are compatible. The algebra $M_n(\mathbb{K})$ is the *full matrix algebra of rank* n *over* $\mathbb{K}$.

Example 2.6.2. The polynomial ring $\mathbb{K}[x]$ of polynomials with coefficients from a field $\mathbb{K}$ is an algebra.

The definitions from rings and ideals extend naturally to algebras. In particular, a subring that is also a subspace of A is a *subalgebra* of A. An ideal of A is an ideal of A viewed as a ring as well as a subspace of A. Likewise, quotient rings are called *quotient algebras*.

2.7 Idempotents

In this section, we consider special elements of rings that help to identify the underlying structure of the ring. Let R be a ring. The element $e \in R$ is an *idempotent* if $e \neq 0_R$ and $e^2 = e$. Two idempotents $e, f \in R$ are called *orthogonal* if $ef = fe = 0_R$. An idempotent is *primitive* if it cannot be written as the sum of orthogonal idempotents. An idempotent e is *central* if $e \in Z(R)$.

If an idempotent $e = e_1 + \cdots + e_n$ where $e_1, \ldots, e_n$ are pairwise orthogonal idempotents, then $e_1 + \cdots + e_n$ is a *decomposition* of e.

The defined terms may be combined in the obvious way. *Orthogonal central idempotents* are central idempotents that are orthogonal. A *primitive central idempotent* is a central idempotent that cannot be written as the sum of orthogonal central idempotents. We see how idempotents lead to an understanding of ring structure in Chapter 5.

Chapter 3

Analysis of Algebraic Algorithms

In this chapter, we develop an analysis framework for the algorithms presented in this dissertation. We analyze algorithms to understand their time and space complexity. Traditional techniques for and models of algorithm analysis are presented in textbooks such as Aho, Hopcroft, and Ullman [3] and Cormen, Leiserson, and Rivest [19]. These techniques and models focus on the complexity of algorithms in terms of machine level operations and binary encoding.

Other models of analysis and complexity theory that focus on other aspects of algorithms have also been investigated. Winograd [98] discusses *arithmetic complexity*, where the number of arithmetic operations used by an algorithm is analyzed. Blum, Shub, and Smale [11] develop a framework for the theory of computation over commutative rings. Their framework leads to theories of undecidability and NP-completeness for computation over the real numbers.

In modern models of analysis, the study of algorithms is being abstracted away from the concrete implementation on modern computer hardware. Such abstraction is useful when studying algorithms for which the elements cannot be precisely encoded on a traditional computer (e.g., the real numbers) or when the operation of interest is not a logical operation (e.g., multiplication).

This kind of abstraction is also useful for studying implementation of algorithms. With the introduction of polymorphism in programming languages such as C++, Java, and ML, it is possible to describe and implement algorithms without knowledge of the underlying implementation of the objects that they manipulate. We call such algorithms *encoding neutral*. Encoding neutral algorithms also arise in computational algebra, where algebraic elements are manipulated only by operations

that are defined for the elements.

We still wish to analyze the time and space complexity of encoding neutral algorithms. However, such analysis cannot be done with models where the underlying encoding of elements and cost of operations is assumed to be known. Also, we wish to compare the performance of different encodings when used with a given encoding neutral algorithm without reanalyzing the entire algorithm for each encoding. To accomplish these goals, we introduce *encoding neutral complexity* analysis.

3.1 Analysis Framework

The model of computation we use for encoding neutral complexity analysis is an *encoding neutral random-access machine (ENRAM)*. Memory cells in the ENRAM model can store any abstract element e with an associated storage cost $s(e)$. The storage cost is left undetermined until an encoding for e is fixed.

In the ENRAM model, it is convenient to assume the underlying machine can perform any arbitrary operation f that we seek to analyze. Every operation has an associated time cost $t(f)$, which is left undetermined until encodings for its parameters have been fixed, or when an abstract analysis for an encoding neutral description of the operation is known.

Algorithms for the ENRAM model are presented using imperative style pseudocode similar to Cormen, Leiserson, and Rivest [19]. We wish to maintain consistent notation between the mathematical expression and pseudocode description of our algorithms. Variables written in italics store mathematical objects and hence are immutable. We use small caps font to denote algorithm names and function names. Teletype font is used for computer science style variables, which can be assigned multiple values during the execution lifetime of an algorithm. Table 3.1 summarizes the meaning of notation and font selection used in our algorithms.

Algorithms are assumed to maintain information locally; that is, global variables are not used unless explicitly stated.

Algorithms are executed in a sequence of steps. The sequence is determined by standard looping, decision, and subroutine conventions used by imperative languages. Each line in an algorithm corresponds to a single step. The running time of an algorithm is defined by Cormen, Leiserson, and Rivest [19] as the number of steps executed by the algorithm. Because we are interested in counting the number of times an algebraic operation is performed, this definition of running time

Table 3.1: Notation for Pseudocode

Notation	Meaning
A	mathematics variable (immutable value)
`variable`	computer science variable (assignable value)
ALGORITHM	algorithm name
FUNCTION(...)	function call
`algebra.dimension`	object property (e.g., dimension of an algebra)
`var`$[i]$	indices for indexed object (e.g., arrays or dictionaries)

is inadequate because an operation may be executed several times in a single line of pseudocode. However, we use the same measure of step when determining the encoding neutral space complexity of an algorithm. The running time and space complexity of an algorithm are elaborated below.

When analyzing algorithms and their growth in space and time, we analyze the growth with respect to the size of input. In most traditional analyses, the size of the input is described by a single parameter, e.g., n. For encoding neutral algorithms, we cannot easily describe the size of the input as a single parameter, because the encoding of the input is unknown. However, we often can (and will assume we can) parameterize an instance with l independent indeterminates $x_1, ..., x_l$. For example, a matrix algebra over a finite field is parameterized by matrix dimension m, vector space dimension d, and the size q of the finite field. For similar reasons, we describe the size of the output of an algorithm A as a function $r_{\mathrm{A}}(x_1, ..., x_l)$ in the same l indeterminates.

Let A be an encoding neutral algorithm. Let F be the set of operations executed by A. The *encoding neutral time complexity* $T_{\mathrm{A}}(x_1, ..., x_l)$ of A is

$$T_{\mathrm{A}}(x_1, ..., x_l) \quad = \quad \sum_{f \in F} c_f(x_1, ..., x_l) t(f)$$

where $c_f(x_1, ..., x_l)$ is a function denoting the number of times an operation f is executed by A. The function c_f can describe the worst case, best case, or average case number of times f is executed depending on the analysis being done.

We also wish to analyze the storage requirements of A. Let $E_i(x_1, \ldots, x_l)$ be the multiset of elements that A has stored at step i in its execution. The *encoding neutral space complexity* $S_{\mathrm{A}}(x_1, ..., x_l)$ of A is

$$S_{\mathrm{A}}(x_1, ..., x_l) \quad = \quad \max_i \sum_{e \in E_i(x_1, \ldots, x_l)} s(e).$$

The composition of E_i depends on the kind of analysis being done: worst case, average case, or best case.

3.2 Example: Finding the Center of an Algebra

We apply the analysis framework to a fundamental problem for algebra decomposition. Let A be an algebra over $\mathbb{K}$ of finite dimension d as a $\mathbb{K}$-vector space. Recall that the center $Z(A)$ of A is defined by

$$Z(A) \quad = \quad \{z \in A \,|\, y * z = z * y \text{ for all } y \in A\}.$$

Let $Y = \{y_1, y_2, \ldots, y_h\}$ be a set of generators for A, which are not necessarily a basis for A. Clearly, $Z(A)$ can be expressed as

$$Z(A) \quad = \quad \{z \in A \,|\, y_i * z = z * y_i, 1 \leq i \leq h\}.$$

The algebra A has a basis $b = \{b_1, \ldots, b_d\}$ as a $\mathbb{K}$-vector space. Generically, $z = \sum_{j=1}^{d} \alpha_j b_j$ for some $\alpha_j \in \mathbb{K}$. Calculating $Z(A)$ is then accomplished by simultaneously solving the system of linear equations, $\sum_{j=1}^{d} \alpha_i j(y_i b_j - b_j y_i) = 0$, for α_{ij}, where $1 \leq i \leq h$.

Figure 3.1 presents an encoding neutral algorithm for computing $Z(A)$ by solving the described system of linear equations. Lines 1–4 initialize the variables used by the algorithm. Their roles are similar to the variables used in the mathematical formulation for calculating $Z(A)$. The **for** statements on lines 5 and 6 loop over every basis element and every generator of A to set up the system of linear equations. Line 7 calculates the difference `a` of $y_i b_j - b_j y_i$. Line 8 stores the coefficients of the `a` in the triply indexed array X. Each $d \times d$ matrix $X[\mathtt{i}]$ represents the differences of $y_i b_j - b_j y_i$, for $1 \leq j \leq d$. For each `i`, where $1 \leq \mathtt{i} \leq h$, the nullspace of $X[\mathtt{i}]$ is the space of coefficients of b that describe elements commuting with y_i. Thus, on line 9, the intersection of the nullspaces of the $X[i]$ is the space of coefficients of b describing elements in $Z(A)$. Lines 12 and 13 compute a basis for $Z(A)$, storing the basis in z. Line 14 returns $Z(A)$ as a subalgebra of $Z(A)$ described by the basis z.

3.2.1 Encoding Neutral Time Complexity for Center

In general, when performing a time complexity analysis of an algorithm, we focus on operations that potentially dominate the time cost of the algorithm. For example, we frequently overlook the cost

CENTER(A) Compute $Z(A)$

INPUT: A finite dimensional algebra A.
OUTPUT: The center $Z(A)$ of A.

```
1   y ← A.generators
2   h ← |y|
3   b ← A.basis
4   d ← A.dimension
5   for i ← 1 to h
6       do for j ← 1 to d
7              do a ← y[i] * b[j] − b[j] * y[i]
8                  Define X[i, j, k] by a = Σ_{k=1}^{d} X[i, j, k] * b[k]
9   V ← ⋂_{i=1}^{d} NULLSPACE(X[i])
10  α ← V.basis
11  n ← V.dimension
12  for i ← 1 to n
13      do z[i] ← Σ_{j=1}^{d} α[i, j] * b[j]
14  return SUBALGEBRA(A, z)
```

Figure 3.1: Encoding neutral algorithm for computing $Z(A)$

Table 3.2: Operation counts for CENTER

Line	Times Executed	Operations			
		$*_A$	$+_A$	$*_{\mathbb{K},A}$	f_l
1	1				
2	1				
3	1				
4	1				
5	1				
6	h				
7	dh	2	1		
8	dh				yes
9	1				yes
10	1				
11	1				
12	1				
13	n		$d-1$	d	
14	1				
Total		$2dh$	$dh+dn-n$	dn	$dh*t(f_8)+t(f_9)$

of memory accesses, comparisons, and the overhead of function calls unless they will dominate the time cost of the algorithm. Let $*_A, +_A, *_{\mathbb{K},A}$ be the operations for algebra multiplication, algebra addition, and scalar multiplication respectively. Let f_l be an operation that performs the entire computation on line l in an algorithm. We have the following theorem about the encoding neutral time complexity for CENTER.

Theorem 3.2.1. *The worst case encoding neutral time complexity of algorithm* CENTER *is*

$$T_{\text{CENTER}}(d,h) \quad = \quad \Theta(dht(*_A)+(dh+d^2)t(+_A)+d^2t(*_{\mathbb{K},A})+dht(f_8)+t(f_9)).$$

Proof. Table 3.2 shows the operation counts for lines in CENTER that perform operations, with the exception of lines 8 and 9 which are included seperately in the complexity formula. In the worst case, where A is commutative, $n=d$, giving the complexity shown above. □

We refine our complexity analysis by elaborating on $t(f_9)$. The operation f_9 may be implemented by viewing X as a single $d \times hd$ matrix as in Figure 3.2 and column reducing X. Back substitution is used to find the solution space. A direct implementation of column reduction with pivoting and back substitution costs

$$t(f_9) \quad = \quad \Theta(d^3ht(*_{\mathbb{K}})+d^3ht(+_{\mathbb{K}})),$$

$$\underbrace{\begin{bmatrix} X[1,1,1] & X[1,1,2] & \cdots & X[1,1,d] \\ X[1,2,1] & X[1,2,2] & \cdots & X[1,2,d] \\ \vdots & \vdots & \cdots & \vdots \\ X[1,d,1] & X[1,d,2] & \cdots & X[1,d,d] \end{bmatrix}}_{X[1]} \cdots \underbrace{\begin{bmatrix} X[h,1,1] & X[h,1,2] & \cdots & X[h,1,d] \\ X[h,2,1] & X[h,2,2] & \cdots & X[h,2,d] \\ \vdots & \vdots & \cdots & \vdots \\ X[h,d,1] & X[h,d,2] & \cdots & X[h,d,d] \end{bmatrix}}_{X[h]}$$

Figure 3.2: Combining X into an $d \times hd$ matrix.

where $*_{\mathbb{K}}$ and $+_{\mathbb{K}}$ are multiplication and addition in the field.

Corollary 3.2.2. *The worst case encoding neutral time complexity of* CENTER *is*

$$T_{\text{CENTER}}(d,h) = \Theta(dht(*_A) + (dh+d^2)t(+_A) + d^2t(*_{\mathbb{K},A}) + d^3ht(*_{\mathbb{K}}) + d^3ht(+_{\mathbb{K}}) + dht(f_8)).$$

The cost for f_8 is left undetermined for now. We elaborate on the cost when fixing the encodings for input to CENTER and performing the associated analysis.

3.2.2 Encoding Neutral Space Complexity for Center

For space complexity, we are interested in the growth of the amount of memory that CENTER requires during its execution with respect to the size of its input. Let E denote the multiset of elements appearing during the execution of CENTER. Denote the maximum size cost of an element in E by $\hat{s}(E) = \max_{e \in E} s(e)$. Let the multiset $E_V = \{e \in E \mid e \in V\}$ be the multiset E restricted to elements contained in a (multi)set V.

Theorem 3.2.3. *The worst case encoding neutral space complexity of* CENTER *is*

$$S_{\text{CENTER}}(d,h) = \Theta(\max\left\{d^2h\hat{s}(E_{\mathbb{K}}), d^2\hat{s}(E_{\mathbb{K}}) + d\hat{s}(E_A)\right\})$$

Proof. The space requirements of CENTER are clearly dominated by the space allocated for X, the space required by the solution space, and the space required to construct the basis for the output algebra. By the description of the algorithm, X requires $\Theta(d^2h\hat{s}(E_{\mathbb{K}}))$ space. We can assume V is always encoded by a basis in a matrix encoding. Thus, V requires $\Theta(d^2\hat{s}(E_{\mathbb{K}}))$ space in the worst case. Finally, to encode the basis for the output algebra, z stores $\Theta(d)$ algebra elements requiring $\Theta(d\hat{s}(E_A))$ space. The algorithm can dispose of X following the computation of V. Hence, the space for X and V is required at the same time until the space for X is released. Likewise, the space for

V and z is required at the same time, until z is computed. Thus, the maximum space requirements for CENTER are either when X and V are in memory at the same time or when V and z are in memory. The encoding neutral space complexity follows directly from these observations. □

3.2.3 Fixing an Encoding

To refine the analysis of the time and space complexity of an algorithm A, we fix an encoding for the input (and output) of A. In this section, we focus on a commonly used and general encoding for finite dimensional algebras. A *matrix algebra* M is an algebra in which elements are encoded by square $m \times m$ matrices over a field $\mathbb{K}$. The standard matrix addition, matrix multiplication, and scalar multiplication operations form the operations of the algebra. Any subset of square matrices over a field that is closed under the algebra operations forms a matrix algebra.

Matrix algebras are used in several analyses of algebraic algorithms because the encoding is easy to analyze. However, as shown in remaining chapters, matrix algebras are often not the most space or time efficient encodings for algebras. Also, as stated previously, matrix algebras are a general way to encode any finite dimensional algebra. Suppose we have a d-dimensional $\mathbb{K}$-algebra A with a basis $b = \{b_1, b_2, \ldots, b_d\}$. Multiplication in A with respect to b is defined by

$$b_i b_j = \sum_{k=1}^{d} \alpha_{ijk} b_k,$$

for $1 \leq i, j \leq d$. The coefficients α_{ijk} are the *structure constants* for A. Define a matrix $M_i \in M_d(\mathbb{K})$ representing multiplication on the left by b_i as

$$M_i = \begin{bmatrix} \alpha_{i11} & \alpha_{i21} & \cdots & \alpha_{id1} \\ \alpha_{i12} & \alpha_{i22} & \cdots & \alpha_{id2} \\ \vdots & \vdots & \ddots & \vdots \\ \alpha_{i1d} & \alpha_{i2d} & \cdots & \alpha_{idd} \end{bmatrix}.$$

Define a map $\Phi : A \to M_d(\mathbb{K})$ by

$$\Phi(\beta_1 b_1 + \beta_2 b_2 + \cdots + \beta_d b_d) = \beta_1 M_1 + \beta_2 M_2 + \cdots + \beta_d M_d,$$

for $\beta_i \in \mathbb{K}$. Then $\Phi(A)$ is the *regular matrix representation* of A as $d \times d$ $\mathbb{K}$-matrices with respect to b. Curtis and Reiner [21, Sec 10.17, pg. 47] also contains a description of the regular matrix representation.

Proposition 3.2.4. *The map $\Phi : A \to \Phi(A)$, where $\Phi(A) \subseteq M_d(\mathbb{K})$ is an algebra isomorphism.*

Proof. Clearly, $\Phi(A)$ is a linear map by definition. Because multiplication in A is associative, $(b_i b_j)b_k = b_i(b_j b_k)$. We find

$$\begin{aligned}
(b_i b_j)b_k &= b_i(b_j b_k) \\
\left(\sum_{h=1}^{d} \alpha_{ijh} b_h\right) b_k &= b_i \left(\sum_{h=1}^{d} \alpha_{jkh} b_h\right) \\
\sum_{h=1}^{d} \alpha_{ijh}(b_h b_k) &= \sum_{h=1}^{d} \alpha_{jkh}(b_i b_h) \\
\sum_{h=1}^{d} \alpha_{ijh} \sum_{l=1}^{d} \alpha_{hkl} b_l &= \sum_{h=1}^{d} \alpha_{jkh} \sum_{l=1}^{d} \alpha_{ihl} b_l,
\end{aligned}$$

which implies

$$\sum_{h=1}^{d} \alpha_{ijh}\alpha_{hkl} = \sum_{h=1}^{d} \alpha_{jkh}\alpha_{ihl},$$

for $1 \leq i, j, k, l \leq d$. It follows directly that $\Phi(b_i b_j) = \Phi(b_i)\Phi(b_j)$ for $1 \leq i, j \leq d$. By linearity, $\Phi(aa') = \Phi(a)\Phi(a')$ for all $a, a' \in A$. By the definition of structure constants, $\Phi(0_A)$ and $\Phi(1_A)$ are the zero and identity matrices respectively.

Since b is a basis, $1_A = \sum_{k=1}^{d} \alpha_k b_k$. The fact that $b_i 1_A = b_i$, implies that $b_i b_j \neq 0$ for some b_j. By the definition of structure constants and Φ, $\Phi(b_i) \neq 0_{M_d(\mathbb{K})}$, for each i, $1 \leq i \leq d$.

Suppose $\Phi(a) = \Phi(a')$ and $a \neq a'$ for some $a, a' \in A$. Write $a - a' = \sum_{k=1}^{d} \beta_k b_k$. By the fact that $a - a' \neq 0_A$, some $\beta_k \neq 0_{\mathbb{K}}$. Furthermore, $0_{M_d(\mathbb{K})} = \Phi(a) - \Phi(a') = \Phi(a - a')$. By the linearity of Φ, $\Phi(a - a') = \sum_{k=1}^{d} \beta_k \Phi(b_k)$. Since some $\beta_k \neq 0_{\mathbb{K}}$, and $\Phi(b_k) \neq 0_{M_d(\mathbb{K})}$ for each k, $1 \leq k \leq d$, we obtain a contradiction. Hence, Φ is injective, completing the proof. □

Corollary 3.2.5. *If A is a d-dimensional $\mathbb{K}$-algebra, there exists an algebra isomorphism $\Phi : A \to \Phi(A)$ where $\Phi(A) \subseteq M_d(\mathbb{K})$.*

Proposition 3.2.6. *For every $d \geq 1$, there exists a d-dimensional $\mathbb{K}$-algebra A such that $A \ncong \Psi(A)$ for every map $\Psi : A \to M_n(\mathbb{K})$ and every integer n, where $1 \leq n < d$.*

Proof. Let $A = \bigoplus_{i=1}^{d} \mathbb{K}$, where A acts independently in each copy of $\mathbb{K}$. Let

$$b_i = (0_{\mathbb{K}}, 0_{\mathbb{K}}, \ldots, 0_{\mathbb{K}}, \underset{i}{1_{\mathbb{K}}}, 0_{\mathbb{K}}, \ldots, 0_{\mathbb{K}})$$

be an element of A, so that $B = \{b_i \mid 1 \leq i \leq d\}$ is a basis for A. Furthermore, the elements of B are orthogonal idempotents.

Suppose $\Psi : A \to M_n(\mathbb{K})$ is an algebra isomorphism. Then $\Psi(A)$ is a subalgebra of $M_n(\mathbb{K})$ and acts on $\bigoplus_{i=1}^{n}$ in the obvious way (by matrix multiplication). Since each b_i is an idempotent and A is commutative, $\Psi(b_i)$ is an $n \times n$ diagonal matrix with $0_{\mathbb{K}}$'s and $1_{\mathbb{K}}$'s on the diagonal, at least one of which is $1_{\mathbb{K}}$. Whenever $i \neq j$, we have $b_i b_j = 0_A$ and $\Psi(b_i)\Psi(b_j) = 0$. Hence each $\Psi(b_i)$ has a $1_{\mathbb{K}}$ in a position on the diagonal that has $0_{\mathbb{K}}$'s for all b_j, where $j \neq i$. We conclude that $n \geq d$. □

Corollary 3.2.5 and Proposition 3.2.6 imply a tight worst case bound, the dimension of an algebra A, on the matrix dimension needed to encode A as an isomorphic matrix algebra. In general, algebras can be encoded using matrices of smaller dimension. However, determining how small those matrices can be is directly related to the problem of algebra decomposition. In practice, the regular matrix representation is readily computable and is the encoding used by computer algebra systems.

Each encoding affects the costs of the algebra operations and element storage. We focus on practical implementations of operations in our analysis instead of theoretically efficient implementations. Practical implementations are ones most frequently used in current computer algebra systems and should provide more accurate conclusions for the best algorithms to use when building computational algebra systems.

For a matrix algebra M of dimension d, we use textbook implementations of matrix addition, matrix multiplication, and scalar multiplication. Also, matrices store every field element including zeroes; the encoding is not a sparse encoding. Given these assumptions, Table 3.3 shows the worst case costs for performing standard algebra operations when M is encoded as a matrix algebra.

Let $\tau(x)$ denote the encoding of element $x \in A$ and let $\tau(X) = \{\tau(x) | x \in X\}$ be the set of encodings of elements in the set $X \subseteq A$. Let $\tau_S(x)$ and $\tau_S(X)$ be the set of elements in a set S contained in the encoding $\tau(x)$ or set of encodings $\tau(X)$ respectively. Table 3.4 shows the cost of encoding a single element $a \in M$, encoding M using h generators, and encoding M using d basis elements.

Recall that line 8 from CENTER writes the product `a` in terms of a basis b for A. We can assume, without loss of generality, that the basis returned by A.`basis` is in semi-echelon form. Given this assumption, f_8 is implemented by using $\Theta(d)$ accesses in an element a to determine the coefficient for each basis element contributing to a. Since each access happens in constant time, the cost of f_8 is negligible in comparison to the algebraic operations in CENTER. Hence, since the cost of $t(f_8)$ is

Table 3.3: Time complexity of matrix algebra operations.

Operation	Complexity
$+_A$	$\Theta(m^2 t(+_{\mathbb{K}}))$
$*_A$	$\Theta(m^3 t(*_{\mathbb{K}}) + m^3 t(+_{\mathbb{K}}))$
$*_{\mathbb{K},A}$	$\Theta(m^2 t(*_{\mathbb{K}}))$

Table 3.4: Space cost of matrix algebras

Encoded	Complexity
$a \in M$	$\Theta(m^2 \hat{s}\,(\tau_{\mathbb{K}}(a)))$
M via generating set G	$\Theta(hm^2 \hat{s}\,(\tau_{\mathbb{K}}(G)) + s(\mathbb{K}))$
M via basis B	$\Theta(dm^2 \hat{s}\,(\tau_{\mathbb{K}}(B)) + s(\mathbb{K}))$

dominated by other costs in the algorithm, we remove the term from the complexity, obtaining the following theorem.

Theorem 3.2.7. *The worst case encoding neutral time complexity of* CENTER *when input algebra A is encoded as a matrix algebra is*

$$T_{\text{CENTER}}(d,h,m) \quad = \quad \Theta((dhm^3 + d^2m^2 + d^3h)t(*_{\mathbb{K}}) + (dhm^3 + d^2m^2 + d^3h)t(+_{\mathbb{K}}))$$

Proof. We substitute the matrix encoding operation costs for the operation costs in Corollary 3.2.2 to obtain

$$\begin{aligned} T_{\text{CENTER}}(d,h,m) &= \Theta(dh(m^3 t(*_{\mathbb{K}}) + m^3 t(+_{\mathbb{K}})) + (dh + d^2)m^2 t(+_{\mathbb{K}}) + d^2 m^2 t(*_{\mathbb{K}}) + d^3 h t(*_{\mathbb{K}}) \\ &\quad + d^3 h t(+_{\mathbb{K}})) \\ &= \Theta((dhm^3 + d^2m^2 + d^3h)t(*_{\mathbb{K}}) + (dhm^3 + dhm^2 + d^2m^2 + d^3h)t(+_{\mathbb{K}})) \\ &= \Theta((dhm^3 + d^2m^2 + d^3h)t(*_{\mathbb{K}}) + (dhm^3 + d^2m^2 + d^3h)t(+_{\mathbb{K}})). \end{aligned}$$

□

We see from Corollary 3.2.5 and Proposition 3.2.6 that $m = d$ is a tight bound on the matrix dimension. We replace the m parameter with d to obtain the following encoding neutral time complexity.

Corollary 3.2.8. *The worst case encoding neutral time complexity of* CENTER *when input algebra*

A is encoded as a matrix algebra is

$$T_{\text{CENTER}}(d,h) \quad = \quad \Theta(d^4 ht(*_{\mathbb{K}}) + d^4 ht(+_{\mathbb{K}})).$$

3.3 Summary

We have presented the encoding neutral analysis framework we use to analyze the algorithms appearing in the remainder of this dissertation. We demonstrated the framework by analyzing an algorithm CENTER for computing the center of an algebra. In addition to introducing the analysis framework, we presented matrix algebras, a general encoding for finite dimensional algebras, which is commonly used in computer algebra systems. Our analysis was refined to analyze the space and time complexity for computing the center of a matrix algebra.

In the next chapter, we continue to develop the framework we use to analyze the presented algorithms. Three additional encodings for algebras are described and their common operations are analyzed. For each encoding, we fix the encoding and analyze CENTER to determine the effects of the input encoding.

Chapter 4

Algebra Encodings

For the algorithms in this dissertation, algebras are inputs. We have seen matrix algebras, a general method for encoding algebras on the computer, in Section 3.2.3. Algebras may be encoded in several other ways, depending upon their description in the literature, their application for a particular problem domain, or their limitations as a result of encoding the input on a computer.

An encoding may be general, such as matrix algebras, or only applicable to a class of algebras. Encodings also affect the space and time complexity of algebraic algorithms. We want to understand in what ways and to what extent encodings affect algorithms. In this section, three more encodings for algebras are described in detail: multiplication tables, group algebras, and path algebras. The complexity of operations and the space required for encoding are analyzed using the ENRAM model. The analysis is applied to the problem of computing the center of an algebra (see Section 3.2) to compare the space requirements and time complexity for the different encodings, when possible.

4.1 Multiplication Tables

Multiplication tables are another general way to encode finite dimensional algebras. Both multiplication tables and the regular matrix representation (see Section 3.2.3) of algebras use structure constants to encode an algebra. The regular matrix representation is a dense encoding, storing $0_{\mathbb{K}}$ as part of every element. In contrast, multiplication tables are a sparse encoding, $0_{\mathbb{K}}$ is not stored in general. We describe the implementation of multiplication tables in GAP version 4 [42].

Let A be a $\mathbb{K}$-algebra of dimension d with a basis $b = \{b_1, \ldots, b_d\}$. Recall that any element

$a \in A$ is uniquely expressed as a linear combination of basis elements

$$a = \sum_{i=1}^{d} \alpha_i b_i,$$

for some $\alpha_i \in \mathbb{K}$.

An element a is encoded as a list $[\alpha_1, \ldots, \alpha_d]$ of coefficients. This encoding requires at most $d\hat{s}(\alpha)$ space to encode, where $\alpha = \{\alpha_i\}_{i=1}^{d}$. Addition of two elements is the component-wise addition of coefficients in the list. Clearly, to add two elements requires $\Theta(dt(+_{\mathbb{K}}))$ time. Scalar multiplication of an element is implemented by multiplying each component by the scalar, requiring $\Theta(dt(*_{\mathbb{K}}))$ time.

We now describe a table storing each product $b_i b_j$ for $1 \leq i, j \leq d$. This table is a *multiplication table* for A with respect to b. Given a multiplication table, we can compute any product aa' for a and $a' \in A$ using the linearity of multiplication.

A multiplication table is encoded sparsely. A table is a $d \times d$ array, with the ij^{th} entry containing the product $b_i b_j = \sum_{k=1}^{d} \alpha_{ijk} b_k$ encoded as a list containing two lists. Let $k_1, k_2, \ldots, k_n$ denote indices such that $\alpha_{ijk_x} \neq 0_{\mathbb{K}}$. The first list contains indices of basis elements with non-zero coefficients, and the second is a list of the corresponding coefficients:

$$[[k_1, \ldots, k_n], [\alpha_{ijk_1}, \ldots, \alpha_{ijk_n}]].$$

Multiplication via the multiplication table is implemented using the linearity properties of algebra elements and performing table lookups to find the results of multiplying basis elements. Figure 4.1 contains pseudocode for implementing multiplication using the multiplication table encoding.

Example 4.1.1. Let A be a three-dimensional $\mathbb{Q}$-algebra with basis $B = \{b_1, b_2, b_3\}$. Multiplication of the basis elements is defined by the equations:

$$\begin{array}{lll} b_1 b_1 = b_1 & b_2 b_1 = b_1 & b_3 b_1 = 0 \\ b_1 b_2 = b_2 & b_2 b_2 = b_2 & b_3 b_2 = 0 \\ b_1 b_3 = -2b_1 + 2b_2 & b_2 b_3 = 2b_1 - 2b_2 & b_3 b_3 = -4b_3 \end{array}$$

The multiplication table encoding for A is

$$\begin{bmatrix} [[1],[1]] & [[2],[1]] & [[1,2],[-2,2]] \\ [[1],[1]] & [[2],[1]] & [[1,2],[2,-2]] \\ [[],[]] & [[],[]] & [[3],[-4]] \end{bmatrix}.$$

MULTTABLEMULTIPLY($\texttt{table}, a, a', d$) Multiply a and a' using a multiplication table.

INPUT: Multiplication `table`, algebra elements a and a', d is the dimension of the algebra.
OUTPUT: The product aa'.

```
1   ▷ a and a′ are assumed to be densely encoded elements
2   product ← 1 × d array of 0_𝕂
3   for i ← 1 to d
4       do if a[i] ≠ 0_𝕂
5             then for j ← 1 to d
6                     do if a′[j] ≠ 0_𝕂
7                           then c ← a[i] ∗ a′[j]
8                                indices ← table[i][j][1]
9                                coeffs ← table[i][j][2]
10                               n ← |indices|
11                               for k ← 1 to n
12                                   do l ← indices[k]
13                                      product[l] ← product[l] + c ∗ coeffs[k]
14
15  return product
```

Figure 4.1: Multiplication using a multiplication table

There are 9 integers for the indices and 9 rational numbers for the coefficients stored in this encoding of A. The regular matrix encoding of A stores $d^3 = 27$ rational numbers, thus the multiplication table implementation is saving the cost of 18 rational numbers at the cost of storing 9 integers.

If a basis B is computable for an algebra A, and the structure constants for A with respect to B are obtainable, a multiplication table encoding for A is easily constructed by the definition of multiplication tables.

Theorem 4.1.2. *A finite dimensional algebra has an equivalent multiplication table encoding.*

Although in most cases the multiplication table encoding is sparse, dense multiplication tables do exist. One way to convert an algebra with a sparse multiplication table to one with a dense multiplication table is to perform a change of basis using a randomly selected invertible $d \times d$ matrix. Hence, in the worst case, we assume that each entry in the multiplication table has $\Theta(d)$ integers and field elements stored.

Theorem 4.1.3. *The worst case encoding neutral time complexity of* MultTableMultiply *is*

$$T_{\text{MultTableMultiply}}(d) \quad = \quad \Theta(d^3(t(*_{\mathbb{K}}) + t(+_{\mathbb{K}})))$$

Proof. Let a and a' be elements in an algebra A. Let $B = \{b_1, b_2, \ldots, b_d\}$ be a basis for A. Clearly, if $a = \sum_{i=1}^{d} \alpha_i b_i$ and $a' = \sum_{i=1}^{d} \alpha'_i b_i$ such that each $\alpha_i \neq 0_{\mathbb{K}}$ and $\alpha'_i \neq 0_{\mathbb{K}}$, then the bodies of **for** loops on line 3 and line 5 will be executed every time. If the product $b_i b_j = \sum_{k=1}^{d} \beta_{ijk} b_k$ such that $\beta_{ijk} \neq 0_{\mathbb{K}}$ for each i, j, and k, then n on line 10 is always equal to d. Thus, the body of the **for** loop on line 11 is executed d times each time the loop is entered.

On line 7 there is one execution of $*_{\mathbb{K}}$. The line is executed at most d^2 times, contributing $\Theta(d^2 t(*_{\mathbb{K}}))$ to the time complexity.

On line 13 there is one execution of $*_{\mathbb{K}}$ and one execution of $+_{\mathbb{K}}$. The line is executed at most d^3 times, contributing $\Theta(d^3(t(*_{\mathbb{K}}) + t(+_{\mathbb{K}})))$ to the time complexity. □

Table 4.1 and Table 4.2 summarize the worst case encoding neutral time and space complexity for algebras encoded using multiplication tables.

The average case space complexity for algebras encoded using multiplication tables is related to the average sparseness of a multiplication table. For matrix algebras given via a basis, the average case and worst case space complexities coincide because the matrix algebra encoding is dense. While

Table 4.1: Worst case time complexity of operations for multiplication table encoding.

Operation	Complexity
$+_A$	$\Theta(dt(+_{\mathbb{K}}))$
$*_A$	$\Theta(d^3(t(*_{\mathbb{K}}) + t(+_{\mathbb{K}})))$
$*_{\mathbb{K},A}$	$\Theta(dt(*_{\mathbb{K}}))$

Table 4.2: Worst case space complexity for multiplication table encoding.

Encoded	Complexity
$a \in A$	$\Theta(d\hat{s}\,(\tau_{\mathbb{K}}(a))$
A	$\Theta(d^3(\hat{s}\,(\tau_{\mathbb{K}}(A)) + s(d)) + s(\mathbb{K}))$

the worst case complexities of the two encodings are approximately the same (the multiplication table is slightly worse), in practice multiplication tables are frequently sparse, leading to a better average case space complexity. The operation costs for the multiplication table encoding are generally better than the matrix algebra encoding as well. Hence, using multiplication tables appear to be better in practice than the matrix algebra encoding. We see, however, that some algorithms related to algebra decomposition (see Chapter 5) rely on properties of matrices. We must convert elements into a matrix encoding to apply these algorithms.

4.1.1 Calculating the Center

We return to the analysis of CENTER. We fix the encoding of the input algebra A to be a multiplication table encoding.

Theorem 4.1.4. *The worst case encoding neutral time complexity of* CENTER *when input algebra A is encoded via a multiplication table is*

$$T_{\text{CENTER}}(d, h) \quad = \quad \Theta\left(d^4 h(t(*_{\mathbb{K}}) + t(+_{\mathbb{K}}))\right).$$

Proof. Let a be an element of A. It is straightforward to retrieve the coefficient of each basis element contributing to a from its encoding. As with matrix algebras, the cost $t(f_8)$ is dominated by the cost of the algebraic operations executed by CENTER.

Substituting the operation costs for the multiplication table encoding for the operation costs in

Corollary 3.2.2, we obtain

$$\begin{aligned} T_{\text{CENTER}}(d,h) &= \Theta\left(d^4h(t(*_{\mathbb{K}})+t(+_{\mathbb{K}}))+(d^2h+d^3)t(+_{\mathbb{K}})+d^3t(*_{\mathbb{K}})+d^3h(t(*_{\mathbb{K}})+t(+_{\mathbb{K}}))\right) \\ &= \Theta\left(d^4h(t(*_{\mathbb{K}})+t(+_{\mathbb{K}}))\right). \end{aligned}$$

□

Each product $b_ib_j = \sum_{k=1}^{d} \alpha_{ijk}b_k$ for two basis elements b_i and b_j is encoded directly in the multiplication table. In CENTER, two algebra element multiplications are used on line 7 to calculate $y_ib_j - b_jy_i$, where y_i is a generator of A. If A is encoded as a multiplication table, the only generating set we are guaranteed to know is b, a basis for A. The products in line 7 consist only of basis elements in this case. Since the products of basis elements are encoded in a multiplication table for A, each product can be calculated by reading directly from the table.

In GAP [42], this fact is used to implement a specialized algorithm for computing $Z(A)$ given A encoded via a multiplication table. Figure 4.2 contains the pseudocode for CENTERMULTTABLE, an algorithm to compute $Z(A)$ from a multiplication table. We see that the worst case time complexity for CENTERMULTTABLE is an improvement over CENTER.

Theorem 4.1.5. *The worst case encoding neutral time complexity of* CENTERMULTTABLE *is*

$$T_{\text{CENTERMULTTABLE}}(d) = \Theta\left(d^4(t(*_{\mathbb{K}})+t(+_{\mathbb{K}}))\right).$$

Proof. In the worst case, each product of basis elements has d non-zero coefficients. Hence, the bodies of the **for** loops on lines 7 and 15 are executed $\Theta(d^3)$ times. The only algebraic operation is a subtraction of field elements on line 18. The remaining operations are memory accesses, so the total complexity contribution of lines 1–18 is $\Theta(d^3t(+_{\mathbb{K}}))$.

The nullspace calculation on line 19 has complexity $\Theta(d^4(t(*_{\mathbb{K}})+t(+_{\mathbb{K}})))$ as previously stated in Section 3.2.1 (with the h parameter replaced by d). The remaining lines do not introduce a significant cost to the computation, so the nullspace calculation on line 19 is the dominant cost. □

4.2 Group Algebras

Recall that a finite group $G = \{g_1, g_2, \ldots, g_d\}$ is a set with a single associative operation, typically called multiplication. Every group contains an identity, denoted 1_G, and every element

CenterMultTable(T, d) Calculate $Z(A)$ from a multiplication table.

INPUT: A multiplication table T for A and the dimension d of A.
OUTPUT: The center $Z(A)$ of A.

```
1   for i ← 1 to d
2       do for j ← 1 to d
3           do indices ← T[i, j, 1]
4              m ← |indices|
5              coeffs ← T[i, j, 2]
6              X[i, j] ← 1 × d array of 0_𝕂
7              for l ← 1 to m
8                  do k ← indices[l]
9                     c ← coeffs[l]
10                    X[i, j, k] ← c
11             ▷ Subtract the opposite multiplication
12             indices ← T[j, i, 1]
13             m ← |indices|
14             coeffs ← T[j, i, 2]
15             for l ← 1 to m
16                 do k ← indices[l]
17                    c ← coeffs[l]
18                    X[i, j, k] ← X[i, j, k] − c
19  V ← ⋂_{i=1}^{d} Nullspace(X[i])
20  z ← V.basis
21  return Subalgebra(A, z)
```

Figure 4.2: Pseudocode to compute $Z(A)$ from a multiplication table.

$g \in G$ has a unique inverse, denoted $g^{-1} \in G$. Let $\mathbb{K}$ be a field. The *group algebra* $\mathbb{K}G$ is $\left\{\sum_{i=1}^{d} \alpha_i g_i \,|\, \alpha_i \in \mathbb{K}, g_i \in G\right\}$, the set of formal sums of elements in G with coefficients α_i from $\mathbb{K}$. Addition of elements in $\mathbb{K}G$ is defined componentwise: $\sum_{i=1}^{d} \alpha_i g_i + \sum_{i=1}^{d} \beta_i g_i = \sum_{i=1}^{d} (\alpha_i + \beta_i) g_i$. Scalar multiplication is also defined componentwise: $\beta \sum_{i=1}^{d} \alpha_i g_i = \sum_{i=1}^{d} \beta \alpha_i g_i$. Multiplication is defined by extending $(\alpha g_i)(\beta g_j) = (\alpha\beta) g_k$, where $g_i g_j = g_k$ in G, to all formal sums via the distributive laws. If $g_1 = 1_G$, we write $\alpha_1 g_1$ simply as α_1 and if $\alpha_i = 1_{\mathbb{K}}$, we write $\alpha_i g_i$ as g_i. It is straightforward to check that $\mathbb{K}G$ is a $\mathbb{K}$-algebra under addition, multiplication, and scalar multiplication.

Group algebras appear as examples of algebras in several texts, including Dummit and Foote [31, Sec 7.2], Farb and Dennis [36, pg. 11], and Hungerford [57, pg. 227]. Apart from being an easily constructed class of algebras group algebras play a vital role in the representation theory of finite groups (see Alperin [5], Curtis and Reiner [21, 22, 23], and Nagao and Tsushima [77]). Group algebras are also important examples of Hopf algebras (see Chari and Pressley [15], Majid [71], and Montgomery [75]). We return to group algebras as examples of Hopf algebras in Chapter 6.

Our primary interest in group algebras is to analyze their effect on the algebraic problems we are studying. While we elaborate on the encoding of group algebra elements, it is beyond our scope to review the encodings available for groups. GAP [42] includes several encodings (and their implementations) of groups, and the reader is referred there for further details on group encodings.

We also note that very advanced techniques are available for the study of groups. While we employ generally applicable algorithms on group algebras in this dissertation, we do not want to imply that the algorithms presented here are the best ones available for group algebras. To the best of our knowledge, no general survey exists for these advanced techniques. Also, implementations of algorithms exist in GAP and Magma [12] without formal publication. Again, we refer the reader to GAP and Magma for details on advanced group and group algebra algorithms.

Example 4.2.1. Let $G = S_3$ be the symmetric group of degree 3. The group elements are $g_1 = (), g_2 = (2\ 3), g_3 = (1\ 2), g_4 = (1\ 2\ 3), g_5 = (1\ 3\ 2)$, and $g_6 = (1\ 3)$, where permutations are in standard cycle notation. The multiplication table for S_3 is in Figure 4.3. Let $\mathbb{K} = \mathbb{F}_3$, the finite field of order 3. The group algebra $\mathbb{K}G = \mathbb{F}_3 S_3$ is a six-dimensional $\mathbb{F}_3$-vector space and hence has 3^6 elements. To demonstrate multiplication in this algebra, take the two elements $a = 2g_1 + g_2 + g_5$

$*$	g_1	g_2	g_3	g_4	g_5	g_6
g_1	g_1	g_2	g_3	g_4	g_5	g_6
g_2	g_2	g_1	g_4	g_3	g_6	g_5
g_3	g_3	g_5	g_1	g_6	g_2	g_4
g_4	g_4	g_6	g_2	g_5	g_1	g_3
g_5	g_5	g_3	g_6	g_1	g_4	g_2
g_6	g_6	g_4	g_5	g_2	g_3	g_1

Figure 4.3: Multiplication Table for S_3

and $b = g_4 + g_6$. Then

$$\begin{aligned} a * b &= (2g_1 + g_2 + g_5)(g_4 + g_6) \\ &= (2g_1)g_4 + g_2g_4 + g_5g_4 + (2g_1)g_6 + g_2g_6 + g_5g_6 \\ &= 2(g_1g_4) + g_3 + g_1 + 2(g_1g_6) + g_5 + g_2 \\ &= g_1 + g_2 + g_3 + 2g_4 + g_5 + 2g_6. \end{aligned}$$

Unlike the matrix algebra encoding and the multiplication table encoding, group algebras are not a general encoding for finite dimensional algebras. The next example describes a finite dimensional algebra that has no equivalent group algebra encoding.

Example 4.2.2. Let $A = M_2(\mathbb{K})$. The dimension of A is 4, so if A can be encoded as a group algebra, a group of order 4 forms a basis for A. The standard basis of A has basis elements that are not invertible, and under any change of basis, they remain noninvertible. However, every group element is invertible, so it is impossible have a group algebra of dimension 4 isomorphic to $M_2(\mathbb{K})$.

To encode elements in $\mathbb{K}G$, we assume there is a total order $<$ on elements in G. Reorder the elements of G so that $g_1, g_2, \ldots, g_d$ are the first, second, $\ldots$, d^{th} group element under $<$. The particular ordering used does not matter as long as it is efficiently computable. An element $a = \sum_{i=1}^{d} \alpha_i g_i$ is encoded sparsely as a list $[g_{j_1}, \alpha_{j_1}, \ldots, g_{j_n}, \alpha_{j_n}]$ of alternating group elements and non-zero coefficients, with $g_{j_k} < g_{j_{k+1}}$, for each k, where $1 \leq k \leq n-1$. Hence, the additive identity $0_{\mathbb{K}G}$ is encoded as [], the empty list.

Addition is implemented by merging lists, adding coefficients for common group elements. If the sum of coefficients for g_i is $0_{\mathbb{K}}$, then g_i and $0_{\mathbb{K}}$ coefficient are not included in the sum. The worst case encoding neutral time complexity for addition is $\Theta(d(t(<_G) + t(=_{\mathbb{K}}) + t(+_{\mathbb{K}})))$ because an element

MULTGROUPALGEBRA(a, a') Multiply a and a' in a group algebra.

INPUT: a and a', sparsely encoded group algebra elements.
OUTPUT: The product aa'.

```
1   n ← |a|
2   n' ← |a'|
3   i ← 1
4   product ← []   ▷ Empty list encodes 0_{KG}
5   while i ≤ n
6         do j ← 1
7            term ← []   ▷ Empty list
8            while j ≤ n'
9                  do k ← a[i] * a'[j]
10                    c ← a[i + 1] * a'[j + 1]
11                    INSERT(term, k, c)
12                    j ← j + 2
13           product ← product + term
14           i ← i + 2
15  return product
```

Figure 4.4: Pseudocode for multiplication of group algebra elements.

in $\mathbb{K}G$ has up to d non-zero coefficients. Merging two $\Theta(d)$ element lists performs $\Theta(d)$ operations.

Scalar multiplication is implemented as componentwise multiplication of the odd indexed components of the list. A single equality test to check whether the scalar is $0_{\mathbb{K}}$ is also needed. The implementation clearly results in a $\Theta(dt(*_{\mathbb{K}}) + t(=_{\mathbb{K}}))$ encoding neutral time complexity.

Multiplication of a and $a' \in \mathbb{K}G$ is implemented as follows. Loop through each term $\alpha_i g_i$ in a, and multiply it by each term $\alpha'_j g_j$ in a'. We note that each product $g_i g_j$ is unique for a fixed term in a by the properties of a group. Hence, we can build up $(\alpha_i g_i)a'$ by inserting the resulting $(\alpha_i \alpha_j)g_i g_j$ into a sorted list in proper order. We sum each $(\alpha_i g_i)a'$ to obtain the product aa'. Figure 4.4 contains the pseudocode for multiplying two elements in $\mathbb{K}G$.

Theorem 4.2.3. *The worst case encoding neutral time complexity of* MULTGROUPALGEBRA *is*

$$T_{\text{MULTGROUPALGEBRA}}(d) \quad = \quad \Theta\left(d^2(t(*_G) + t(*_{\mathbb{K}}) + t(+_{\mathbb{K}}) + t(=_{\mathbb{K}})) + d^2 \log d(t(<_G))\right).$$

Proof. Clearly, the worst case occurs when a and a' have d non-zero coefficients. In the worst case, the body of the **while** loop on line 5 is executed d times. The body of the **while** loop on line 8 is

Table 4.3: Worst case time complexity for group algebra encoding.

Operation	Complexity
$+_A$	$\Theta(d(t(<_G) + t(=_{\mathbb{K}}) + t(+_{\mathbb{K}})))$
$*_A$	$\Theta(d^2(t(*_G) + t(*_{\mathbb{K}}) + t(+_{\mathbb{K}}) + t(=_{\mathbb{K}})) + d^2 \log d(t(<_G)))$
$*_{\mathbb{K},A}$	$\Theta(dt(*_{\mathbb{K}}) + t(=_{\mathbb{K}}))$

Table 4.4: Worst case space complexity for group algebra encoding.

Encoded	Complexity
$a \in A$	$\Theta(d(\hat{s}\,(\tau_{\mathbb{K}}(a)) + \hat{s}\,(\tau_G(a)))$
A	$\Theta(s(G) + s(\mathbb{K}))$

executed d times.

On line 9 there is one multiplication of group elements. On line 10 there is one multiplication of field elements. Each of these operations is executed d^2 times. This contributes $\Theta(d^2(t(*_G) + t(*_{\mathbb{K}})))$ to the complexity.

On line 11, we are inserting into a sorted list. Using binary search (see Cormen, et al. [19] page 15) and assuming we can insert into the list in constant time, the cost of INSERT is $\Theta(\log d(t(<_G)))$. We obtain a total contribution from line 11 of $\Theta((d^2 \log d)t(<_G))$.

On line 13 we are adding two group algebra elements together. Addition has an encoding neutral cost of $\Theta(d(t(<_G)+t(=_{\mathbb{K}})+t(+_{\mathbb{K}})))$ This leads to a total contribution of $\Theta(d^2(t(<_G)+t(=_{\mathbb{K}})+t(+_{\mathbb{K}})))$.

The theorem follows. □

Tables 4.3 and 4.4 summarize the worst case encoding neutral time and space complexity for group algebra encodings. The time and space complexity of the group algebra encoding is comparable to the multiplication table encoding. The costs of multiplying and comparing group elements depend on the encoding of the underlying group. Since we have left the encoding unspecified we cannot definitively say that multiplication of group algebra elements is better than using multiplication tables or a matrix algebra encoding. However, the complexity grows in the parameter d more slowly for group algebra elements than for matrix algebra or multiplication table encodings, so we hypothesize that multiplying group algebra elements is fairly efficient.

4.2.1 Calculating the Center

We return to calculating $Z(A)$ for a group algebra A.

Theorem 4.2.4. *The worst case encoding neutral time complexity of* CENTER *is*

$$T_{\text{CENTER}} = \Theta\left(d^3h(t(*_{\mathbb{K}}) + t(+_{\mathbb{K}})) + d^3t(<_G) + d^2t(=_{\mathbb{K}})\right)$$

Proof. We begin by observing that the elements $1_{\mathbb{K}}g$ for $g \in G$ form a basis of A. Furthermore, generators of A can be obtained from $1_{\mathbb{K}}y$, where each y is a generator of G. The parameters for the algebra multiplications that occur in $y_ib_j - b_jy_i$ (see line 7 in CENTER) are elements of the form $1_{\mathbb{K}}g$, where $g \in G$. The cost $t(*_A)$ for these multiplications reduces to $\Theta(t(*_G) + t(*_{\mathbb{K}}))$ in this case.

Furthermore, the resulting products also have the form $1_{\mathbb{K}}g$. The cost $t(+_A)$ of the addition occuring in $y_ib_j - b_jy_i$ reduces to $\Theta(t(+_{\mathbb{K}}) + t(=_{\mathbb{K}}))$. This applies only to the dh algebra additions occuring as a result of line 7, for a total contribution of $\Theta(dh(t(+_{\mathbb{K}}) + t(=_{\mathbb{K}})))$.

To find the coefficients for the difference $a = y_ib_j - b_jy_i$ on line 8, we assume that we have an enumeration of G in sorted order. A linear search through the sorted list and the encoding of a is used to extract the coefficients from a. The cost $t(f_8) = \Theta(dt(<_G))$ for line 8).

On line 13, a linear combination of original basis elements is constructed to form a basis element for $Z(A)$. The same basis element never occurs twice in the linear combination. As a result, no field additions will take place for the group algebra encoding, however, the group comparison tests do still occur. Furthermore, the scalar multiplication is always with a group algebra element containing only one term, resulting in a single field multiplication and a single comparison to check for $0_{\mathbb{K}}$. The total contribution for this line is $\Theta(d^3t(<_G) + d^2(t(*_{\mathbb{K}}) + t(=_{\mathbb{K}})))$.

Substituting the operation costs for group algebra elements into corollary 3.2.2 to obtain

$$\begin{aligned} T_{\text{CENTER}}(d,h) &= \Theta\Big(dh(t(*_(G)) + t(*_{\mathbb{K}})) \\ &\quad + dh(t(+_{\mathbb{K}}) + t(=_{\mathbb{K}})) \\ &\quad + d^3t(<_G) + d^2(t(*_{\mathbb{K}}) + t(=_{\mathbb{K}})) \\ &\quad + d^3ht(*_{\mathbb{K}}) \\ &\quad + d^3ht(+_{\mathbb{K}}) \\ &\quad + d^2ht(<_G)\Big) \\ &= \Theta\left(d^3h(t(*_{\mathbb{K}}) + t(+_{\mathbb{K}})) + d^3t(<_G) + d^2t(=_{\mathbb{K}})\right). \end{aligned}$$

□

We note that some further improvements of the analysis are possible if we modify the CENTER to directly manipulate group algebra elements as we did for multiplication tables. Instead, we recall a well known fact about group algebras, that permits a more efficient algorithm to compute the center. Before stating the fact, we introduce more terminology.

Two elements g and h in a group G are *conjugate* if there exists an x in G such that $h = xgx^{-1}$. Let the relation $\sim$ be defined by $g \sim h$ if and only if g is conjugate with h. The relation $\sim$ is an equivalence relation on G, and the equivalence classes are the *conjugacy classes* of G. If $C = \{c_1, c_2, \ldots, c_k\}$ is a conjugacy class in G, the sum $c_1 + c_2 + \cdots + c_k$ is the *conjugacy class sum* of C.

The following theorem is a well known fact appearing as exercise 2 in Section 10 of Curtis and Reiner [21] and as exercise 13 in Section 7.2 of Dummit and Foote [31].

Theorem 4.2.5. *Let $A = \mathbb{K}G$ be a group algebra. Let $C = \{c_1, c_2, \ldots, c_k\}$ be a conjugacy class in G. The conjugacy class sum z of C is in $Z(A)$. Furthermore, if $C_1, C_2, \ldots, C_n$ are the conjugacy classes of G, then the conjugacy class sums $z_1, z_2, \ldots, z_n$ of $C_1, C_2, \ldots, C_n$ form a basis for $Z(A)$.*

Theorem 4.2.5 provides a group theoretic approach to computing $Z(A)$. Cannon and Souvignier [14] contains a brief survey and introduces algorithms for efficiently computing conjugacy classes. There appears to be little information regarding the complexity of these algorithms, but experimental data contained in the paper suggests that computing conjugacy classes is more efficient than using linear algebra techniques to calculate $Z(A)$.

In Chapter 5 we see another application of group theoretic techniques to compute a particular subalgebra of A.

4.3 Path Algebras

The encodings described in Section 3.2.3, Section 4.1, and Section 4.2 are inherently finite dimensional. In this section, we describe an encoding that is potentially infinite dimensional. Even given this drawback, the encoding is very expressive and compact. The nature of this encoding allows us to apply pattern matching, graph theoretic, and automata theoretic approaches for computation.

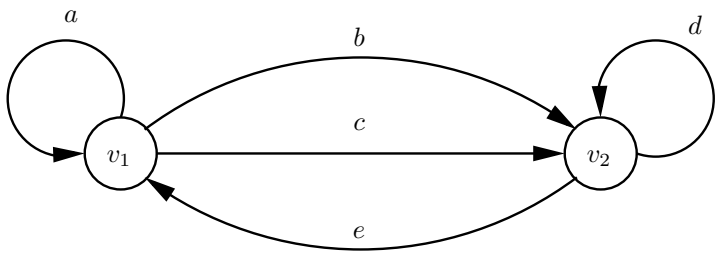


Figure 4.5: A sample quiver $\hat{G} = (\hat{V}, \hat{A})$.

Furthermore, the encoding provides structural information about the algebra that other, more general encodings do not. This additional information allows us to develop more efficient algorithms, for example, the algorithm to construct an endomorphism ring in Chapter 8.

Let V and Σ be disjoint alphabets, and let 0 be a symbol not in $V \cup \Sigma$. A *quiver* $G = (V, A)$ is a labeled, directed multigraph with loops; that is, V is a set of *vertices*, and $A \subseteq V \times V \times \Sigma$ is a set of triples such that every $\sigma \in \Sigma$ occurs exactly once in the third component. Each triple in A is called an *arrow*. The *label* of a vertex is the vertex itself, while the label of an arrow (u, v, σ) is its third component σ. As an example, suppose $\hat{V} = \{v_1, v_2\}$ and $\Sigma = \{a, b, c, d, e\}$. Then one possible quiver $\hat{G} = (\hat{V}, \hat{A})$ is given by the arrows

$$\hat{A} \quad = \quad \{(v_1, v_1, a), (v_1, v_2, b), (v_1, v_2, c), (v_2, v_2, d), (v_2, v_1, e)\}.$$

Small quivers are more conveniently presented in a drawing; see Figure 4.5 for a drawing of $\hat{G}$. Clearly the concept of a quiver generalizes the standard concept of a directed graph.

The set Γ of *walks* in a quiver $G = (V, A)$ consists of three kinds of elements:

1. 0, having *length* $\textsc{Length}(0) = -\infty$;
2. Every element $v \in V$, each v having *length* $\textsc{Length}(v) = 0$; and
3. A sequence (string) $\sigma_1\sigma_2 \cdots \sigma_k \in \Sigma^+$, of *length* $\textsc{Length}(\sigma_1\sigma_2 \cdots \sigma_k) = k$, such that, for each i, $1 \leq i \leq k-1$, where $(u_i, v_i, \sigma_i), (u_{i+1}, v_{i+1}, \sigma_{i+1}) \in A$, we have $v_i = u_{i+1}$. Such a sequence is called a *nontrivial walk*.

In particular, $\Gamma \subset (V \cup \Sigma \cup \{0\})^+$. Except for the 0 walk, this definition corresponds precisely to the standard definition of a walk in a graph or multigraph (see West [97]). The following strings

Table 4.5: Sources and targets for walks

w	$s(w)$	$t(w)$
0	0	0
$v \in V$	v	v
$\sigma_1\sigma_2\cdots\sigma_k \in \Sigma^+$	$s(\sigma_1)$	$t(\sigma_k)$

Table 4.6: Multiplication of non-zero walks

		$w_1 \cdot w_2$ if $t(w_1) = s(w_2)$	$w_1 \cdot w_2$ if $t(w_1) \neq s(w_2)$
$w_1 \in V$	$w_2 \in V$	w_1	0
$w_1 \in V$	$w_2 \notin V$	w_2	0
$w_1 \notin V$	$w_2 \in V$	w_1	0
$w_1 \notin V$	$w_2 \notin V$	w_1w_2	0

are walks in our sample quiver $\hat{G}$:

$$0 \qquad v_2 \qquad bdde \qquad eaabec.$$

The following strings are **not** walks in $\hat{G}$:

$$00 \qquad v_1v_1 \qquad bdc \qquad eea.$$

If $(u, v, \sigma) \in A$, then u is the *source* $s(\sigma)$ of σ, and v is the *target* $t(\sigma)$ of σ. For any walk w, we define its *source* $s(w)$ and *target* $t(w)$ as given in Table 4.5. We see that, except for the 0 walk, the source and target always correspond to the initial vertex and the terminal vertex of the walk, respectively.

We now define a multiplication between elements in Γ. Clearly, 0 should be a zero; for all $w \in \Gamma$, we have $0 \cdot w = w \cdot 0 = 0$. If $w_1, w_2 \in \Gamma - \{0\}$, then we have eight cases for $w_1 \cdot w_2$, based on whether the walks are vertices or nontrivial walks and on whether the target of w_1 equals the source of w_2. These cases are summarized in Table 4.6. In the third column of the last line of the table, w_1w_2 is simply the concatenation of two strings.

A *path algebra* $\mathbb{K}\Gamma = \left\{\sum_{w\in\Gamma-\{0\}} \alpha_w w \,|\, \text{a finite number of } \alpha_w \in \mathbb{K} \text{ are non-zero}\right\}$ is the set of finite formal sums of non-zero walks in Γ, where $\mathbb{K}$ is a field. Multiplication is defined by $(\alpha w)(\beta x) = (\alpha\beta)(w \cdot x)$ for all $\alpha, \beta \in \mathbb{K}$ and $w, x \in \Gamma - \{0\}$ extended to all elements in $\mathbb{K}\Gamma$ via the distributive laws. If the product $w \cdot x = 0$, the resulting term is $0_{\mathbb{K}\Gamma}$ and the term drops out. If $\alpha_w = 1_{\mathbb{K}}$, we write

$\alpha_w w$ as w. The reader may readily verify that $\mathbb{K}\Gamma$ is a $\mathbb{K}$-algebra with one $1_{\mathbb{K}\Gamma} = \sum_{v \in V} v$. Path algebras were introduced in Gabriel [41] and play an important role in the representation theory of associative algebras (see [28, 29, 78, 82]).

We describe the encoding and implementation of path algebras in HOPF [52]. Elements in path algebras are encoded similarly to elements in group algebras (see Section 4.2). Let $a = \sum_{w \in \Gamma - \{0\}} \alpha_w w \in \mathbb{K}\Gamma$. The *support* $\mathcal{S}(a)$ of a is $\{w \in \Gamma - \{0\} \mid \alpha_w \neq 0_{\mathbb{K}}\}$ the set of non-zero walks with non-zero coefficients in a. The a is encoded as the list $[w_1, \alpha_{w_1}, w_2, \alpha_{w_2}, \ldots, w_n, \alpha_{w_n}]$ of alternating walks and coefficients. Each $w_i \in \mathcal{S}(a)$ and because a is a finite sum, the list encoding a is also finite. The additive identity $0_{\mathbb{K}\Gamma}$ is encoded as [], the empty list.

For computation, we assume that there is a total order $<$ on the walks. We elaborate on orders for walks in Section 4.3.1. Given $<$, we implement addition of elements a and a' in $\mathbb{K}\Gamma$ by merging lists. If $n = |\mathcal{S}(a)|$ and $n' = |\mathcal{S}(a')|$, then the maximum length of the list encoding $a + a'$ is $\Theta(n + n')$. The precise worst case encoding neutral time complexity for adding two elements in $\mathbb{K}\Gamma$ is $\Theta((n + n')t(<_\Gamma) + \min\{n, n'\}(t(+_{\mathbb{K}}) + t(=_{\mathbb{K}})))$. If we fix n to be the larger of n and n', then the worst case occurs when $n' = n$. Hence, the worst case encoding neutral complexity is $\Theta(n(t(<_\Gamma) + t(+_{\mathbb{K}}) + t(=_{\mathbb{K}})))$

Scalar multiplication is implemented the same way as for group algebras. Each coefficient in the list encoding a is multiplied by the scalar α. We check for $\alpha = 0_{\mathbb{K}}$ initially to handle multiplication by zero. If a has n coefficients, the worst case encoding neutral time complexity is $\Theta(nt(*_{\mathbb{K}}) + t(=_{\mathbb{K}}))$.

Multiplication of path algebra elements is also similar to multiplication for group algebra elements. We require an additional check for the product of walks resulting in 0, the zero walk. The pseudocode for multiplication appears in Figure 4.6.

Theorem 4.3.1. *The encoding neutral time complexity of* MULTPATHALGEBRA *is*

$$T_{\text{MULTPATHALGEBRA}}(n) \quad = \quad \Theta\left(n^3 t(<_\Gamma) + n^2(t(*_\Gamma) + t(*_{\mathbb{K}}) + t(+_{\mathbb{K}}) + t(=_\Gamma) + t(=_{\mathbb{K}}))\right).$$

Proof. In the worst case, $n = |\mathcal{S}(a)| = |\mathcal{S}(a')|$. The product of walks is never 0, as is the case when the path algebra encoded is a free algebra. In this case, the bodies of the **while** loops on lines 5 and 8 are executed n times, when encountered.

One multiplication of paths appears on line 9. One comparison of a path to 0 appears on line 10. A field multiplication appears on line 11. The total complexity contribution of these lines is $\Theta(n^2(t(*_\Gamma) + t(*_{\mathbb{K}}) + t(=_\Gamma)))$.

MULTPATHALGEBRA(a, a') Multiply a and a' in a path algebra.

INPUT: a and a', sparsely encoded path algebra elements.
OUTPUT: The product aa'.

```
1   n ← |a|
2   n' ← |a'|
3   i ← 1
4   product ← []   ▷ Empty list encodes 0_{KG}
5   while i ≤ n
6         do j ← 1
7            term ← []   ▷ Empty list
8            while j ≤ n'
9                  do k ← a[i] * a'[j]
10                    if k ≠ 0
11                       then c ← a[i + 1] * a'[j + 1]
12                            INSERT(term, k, c)
13                            j ← j + 2
14           product ← product + term
15           i ← i + 2
16  return product
```

Figure 4.6: Pseudocode for multiplying elements in a path algebra encoding.

Table 4.7: Worst case time complexity for path algebra encoding.

Operation	Complexity
$+_A$	$\Theta(n(t(<_\Gamma) + t(=_\mathbb{K}) + t(+_\mathbb{K})))$
$*_A$	$\Theta\left(n^3 t(<_\Gamma) + n^2(t(*_\Gamma) + t(*_\mathbb{K}) + t(+_\mathbb{K}) + t(=_\Gamma) + t(=_\mathbb{K}))\right)$
$*_{\mathbb{K},A}$	$\Theta(nt(*_\mathbb{K}) + t(=_\mathbb{K}))$

Table 4.8: Worst case space complexity for path algebra encoding.

Encoded	Complexity
$a \in A$	$\Theta(n(\hat{s}\,(\tau_\mathbb{K}(a)) + \hat{s}\,(\tau_\Gamma(a))))$
A	$\Theta(s(G) + s(\mathbb{K}))$

The maximum cost of INSERT is $\Theta((\log n)t(<_\Gamma))$ if binary search is used for the implementation as with group algebras. The time complexity contribution is $\Theta((n^2 \log n)t(<_\Gamma))$.

One addition of path algebra elements appears on line 14. We note that $|\mathcal{S}(\texttt{term})| = n$. During the $\texttt{i}^{\text{th}}$ iteration of the outermost **while** loop, $|\mathcal{S}(\texttt{product})| = \Theta((\texttt{i}-1)n)$. In the worst case, there is a $\Theta(n(t(+_\mathbb{K}) + t(=_\mathbb{K})))$ contribution from field additions and comparisons in the path algebra element addition and a $\Theta(\texttt{i}nt(<_\Gamma))$ contribution from comparisons in the path algebra element addition during the $\texttt{i}^{\text{th}}$ of the outermost **while** loop. This gives a $\Theta(n^3 t(<_\Gamma) + n^2(t(+_\mathbb{K}) + t(=_\mathbb{K})))$ contribution to the time complexity.

Summing the contributions from each line gives the encoding neutral time complexity stated. □

We see from Theorem 4.3.1 that it is important to implement the comparison of walks efficiently for an efficient implementation of multiplication. The importance of orders is a primary difference between path algebra encodings and the other algebra encodings presented earlier. Table 4.7 and Table 4.8 summarize the encoding neutral time and space complexities for path algebra encodings.

Recall that path algebras are potentially infinite dimensional. In fact, any path algebra constructed from a quiver containing a cycle is infinite dimensional. This can be seen as a special case of the results presented in Chapter 9. Studying only path algebras constructed from acyclic quivers is overly restrictive. In addition, we can study finite dimensional quotients of path algebras using Gröbner basis theory, which we review in the next section.

4.3.1 Gröbner Bases

In this section, we review Gröbner basis theory for path algebras. Gröbner basis theory allows us to compute with quotients of path algebras. For a general overview of the Gröbner basis theory for noncommutative rings, see Mora [76]. Farkas, Feustel, and Green [37] extend the theory to path algebras.

The set of non-zero walks $B = \Gamma - \{0\}$ is a distinguished basis of $\mathbb{K}\Gamma$. Fix a well order $\prec$ on B. The order $\prec$ is an *admissible order* if the following two properties hold for $r, s, u, v \in B$, where urv and usv are both non-zero:

- $r \prec urv$;
- if $r \prec s$ then $urv \prec usv$.

One admissible order is *(left) length-lexicographic order* $\prec_{\text{lenllex}}$. Let $V = \{v_1, v_2, \ldots, v_n\}$ be the set of vertices and let $\Sigma = \{a_1, a_2, \ldots, a_m\}$ be the set of arrow labels. Order the vertices and arrow labels arbitrarily, such that the vertices are smaller than the arrows. One such ordering is

$$v_1 < \cdots < v_n < a_1 < \cdots < a_m.$$

If p and q are elements in B, then $p \prec_{\text{lenllex}} q$

- if $\textsc{Length}(p) < \textsc{Length}(q)$;
- if $\textsc{Length}(p) = \textsc{Length}(q)$, where $p = b_1 \cdots b_r$, $q = b'_1 \cdots b'_r$, and $b_i, b'_j \in V \cup A$, then for some $1 \leq j \leq r$, $b_i = b'_i$ for $i < j$ and $b_j < b'_j$.

Other examples of commonly used admissible orders are in Becker and Weispfenning [9], Green [48], and Keller [64].

For the remainder of this section, assume $\prec$ is an admissible order on B. Let x be an element of $\mathbb{K}\Gamma$. The *tip* $\text{Tip}(x) = w$ of x is the maximal walk w with respect to $\prec$ such that w has a non-zero coefficient in x. If $X \subseteq \mathbb{K}\Gamma$, then $\text{Tip}(X) = \{\text{Tip}(x) \,|\, x \in X\}$, the set of tips of some element in X. The set of *nontips* is $\text{Nontip}(X) = B - \text{Tip}(X)$, the set of non-zero walks that are not tips of some element in X.

Let x and y be elements of B. We say that x *divides* y, written $x|y$, if we can find $u, v \in \Gamma$ such that $x = uyv$.

Let $a \in \mathbb{K}\Gamma$ and $r \in \mathbb{K}\Gamma$ be such that there exists a $w \in \mathcal{S}(a)$, the support of a, where $\mathrm{Tip}(r)|w$. Since $\mathrm{Tip}(r)|w$, there exists u and v in Γ such that $u\mathrm{Tip}(r)v = w$. Let α and ρ be the leading coefficients of a and r respectively. The difference $a - \frac{\alpha}{\rho}urv$, is a *simple reduction* of a by r. If $S \subseteq \mathbb{K}\Gamma$ and there exists a sequence of $r_1, r_2, \ldots, r_n$ of elements in S such that the sequence $a_1, a_2, \ldots a_n$ defined by

$$\begin{aligned} a_0 &= a, \\ a_i &= a_{i-1} - \frac{\alpha_{i-1}}{\rho_i} u_i r_i v_i, \end{aligned}$$

is a sequence of simple reductions, we say that a *reduces* to a_n over S. An element $a' \in \mathbb{K}\Gamma$ is a *complete reduction* of a over S if a reduces to a' over S, and if $\mathrm{Tip}(r) \not| w$ for each $r \in S$ and $w \in \mathcal{S}(a')$. An element $\overline{a}$ is the *normal form* of a over S if $\overline{a}$ is the only complete reduction of a over S.

Let I be an ideal in $\mathbb{K}\Gamma$ and let $\mathcal{G} \subset I$. If $\langle \mathrm{Tip}(\mathcal{G}) \rangle = \langle \mathrm{Tip}(I) \rangle$ then $\mathcal{G}$ is a *Gröbner basis* for I with respect to $\prec$.

The definition of a Gröbner basis is rather abstract. However, Gröbner bases allow us to find normal forms of elements in $\mathbb{K}\Gamma$ over I as the next two theorems from Farkas, et al. [37] demonstrate.

Theorem 4.3.2. *If $\mathbb{K}\Gamma$ is a path algebra and I is an ideal in $\mathbb{K}\Gamma$, then*

$$\mathbb{K}\Gamma = I \oplus \mathrm{Span}(\mathrm{Nontip}(I))$$

as vector spaces.

Proof. See the proof of Theorem 4 in Farkas, et al. [37] □

Theorem 4.3.3. *If $\mathcal{G}$ is a Gröbner basis for I, then every element in $\mathbb{K}\Gamma$ has a normal form over $\mathcal{G}$, which is equal to its normal form over I.*

Proof. See the proof of Theorem 9 in Farkas, et al. [37] □

Example 4.3.4. Figure 4.7 shows a graph defining a quiver $G = (V, A)$. Let Γ be the set of paths in G. Let $\mathbb{K} = \mathbb{Q}$ the rational numbers. Let I be the ideal generated by the relations $\mathcal{G} = \{a^3, bc - a^2\}$. Using length left lexicographic ordering, $\mathcal{G}$ is a Gröbner basis for I. The set of nontips is

$$\mathrm{Nontip}(I) = \{v_1, v_2, a, b, c, a^2, ab, ca, cb, a^2b, ca^2, cab, ca^2b\}.$$

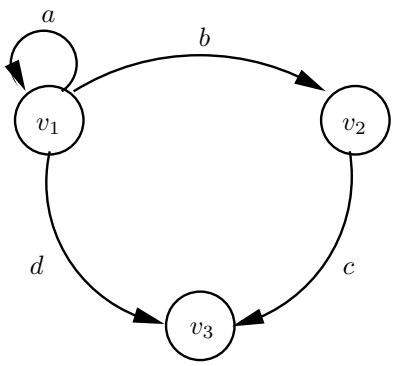


Figure 4.7: A sample quiver $G = (V, A)$.

Let $A = \mathbb{Q}\Gamma/I$. Then, $\mathrm{Dim}_{\mathbb{Q}} A = 13$. The normal form of the element $cbc - ca$ is $ca^2 - ca$, which is a linear combination of nontips.

The encoding for elements in $\mathbb{K}\Gamma/I$ is the same as the encoding for $\mathbb{K}\Gamma$. To compute with elements in $\mathbb{K}\Gamma/I$, we first perform algebra operations in $\mathbb{K}\Gamma$, then reduce the element to a normal form over $\mathcal{G}$. The normal form of an element is always in $\mathrm{Span}(\mathrm{Nontip}(I))$ as a result of Theorems 4.3.2 and 4.3.3.

An algorithm for finding a complete reduction of an element over a set is straightforward given the definition of reduction. For completeness, we include the pseudocode for the algorithm used in HOPF [52] in Figure 4.8.

There appears to be no complexity analysis for REDUCE or related algorithms in the literature. While within the scope of this dissertation, the analysis of REDUCE is left as an open problem. Finding non-trivial bounds for the time cost appears to be difficult and was not a primary goal for the author. An efficient algorithm for the search on line 6 is presented in Chapter 9 as part of a larger result.

If we assume that elements in $\mathbb{K}\Gamma/I$ are always reduced to their normal forms, then the implementations for addition and scalar multiplication require no change from their implementation in $\mathbb{K}\Gamma$. It is clear to see adding two elements in $\mathrm{Span}(\mathrm{Nontip}(I))$ remains in $\mathrm{Span}(\mathrm{Nontip}(I))$. It is also clear for scalar multiplication. The encoding neutral time complexity for these operations is the same for $\mathbb{K}\Gamma/I$ as for $\mathbb{K}\Gamma$.

For multiplication in $\mathbb{K}\Gamma/I$, we must compute the normal form for the product. We do this just

REDUCE(S, a) Find a complete reduction of a over S.

INPUT: S is a set of path algebra elements, a is a path algebra element.
OUTPUT: A complete reduction of a over S.

```
1   cr ← 0_{KΓ}
2   x ← a
3   while x ≠ 0_{KΓ}
4         do t ← Tip(x)
5            Let m be the leading coefficient of x.
6            Find an r ∈ S such that Tip(r)|t.
7            if r does not exist
8              then cr ← cr + m * t
9                   x ← x − m * t
10             else Find u and v such that t = uTip(r)v.
11                  Let n be the leading coefficient of r.
12                  x ← x − (m/n) * u * r * v
13  return cr
```

Figure 4.8: Pseudocode for an algorithm to completely reduce elements.

Table 4.9: Worst case time complexity for general path algebra encodings.

Operation	Complexity
$+_A$	$\Theta(n(t(<_\Gamma) + t(=_\mathbb{K}) + t(+_\mathbb{K})))$
$*_A$	$\Theta\left(n^3 t(<_\Gamma) + n^2(t(*_\Gamma) + t(*_\mathbb{K}) + t(+_\mathbb{K}) + t(=_\Gamma) + t(=_\mathbb{K})) + t(\text{REDUCE})\right)$
$*_{\mathbb{K},A}$	$\Theta(nt(*_\mathbb{K}) + t(=_\mathbb{K}))$

before returning the product, giving an encoding neutral time complexity of $\Theta(n^3 t(<_\Gamma) + n^2(t(*_\Gamma) + t(*_\mathbb{K}) + t(+_\mathbb{K}) + t(=_\Gamma) + t(=_\mathbb{K})) + t((Reduce)))$.

We note that the cost of REDUCE is effectively zero if I is the trivial ideal. Because of this fact, we view the computations for quotients of path algebras as a generalization of the computations for path algebras. The complexity of each operation is the same for the two situations. Henceforth, we will use the term path algebra for path algebras and their quotients unless stated otherwise. Table 4.9 summarizes the modified operation counts for general path algebra encodings.

We do not consider the cost of computing a Gröbner basis as part of the cost for the algebraic algorithms under consideration. We assume that the Gröbner basis is given as part of the input

for the construction of $\mathbb{K}\Gamma/I$. The problem of determining whether I has a finite Gröbner basis is undecidable in general, so this assumption will not always hold. A reduction from the well known word problem to the problem of determining if an element is a member of an ideal is briefly sketched in Mora [76]. Even though a Gröbner basis for $\mathbb{K}\Gamma/I$ is uncomputable in general, they do exist for the most important case of our work, when $\mathbb{K}\Gamma/I$ is finite dimensional, as Farkas, et al. [37] show. Computer tools exist to compute a Gröbner basis for an ideal of a path algebra, such as Opal by Keller [50, 64]. We use Opal as part of HOPF.

It is possible for a finite Gröbner basis to exist, but $\mathbb{K}\Gamma/I$ to have infinite dimension. In Chapter 9 we present an efficient algorithm to determine if $\mathbb{K}\Gamma/I$ is finite dimensional if we have a finite Gröbner basis for I. Thus, for those quotients $\mathbb{K}\Gamma/I$ that we consider computable, that is, I has a (known) finite Gröbner basis, we can determine if the algebraic algorithms we present are applicable.

4.3.2 Computing the Center

Unlike the multiplication table and group algebra encodings, the path algebra encoding does not have a special implementation for calculating $Z(A)$. In general, path algebras can be arbitrarily complex, with products of basis elements having dense supports (i.e., $|\mathcal{S}(b_i b_j)| = d$) after reduction. The following theorem summarizes the time complexity when the input algebra is encoded as a path algebra.

Theorem 4.3.5. *The worst case encoding neutral time complexity for a d-dimensional algebra A encoded as a path algebra is*

$$T_{\text{CENTER}}(d,h) \quad = \quad \Theta\left(d^4 ht(<_\Gamma) + d^3 h(t(*_\Gamma) + t(*_\mathbb{K}) + t(=_\Gamma) + t(=_\mathbb{K}) + t(\text{REDUCE})\right)$$

Proof. The only unspecified complexity is for the extraction of coefficients of basis elements for `a` on line 8 in CENTER. We implement this essentially the same way we did for group algebras. Since $\mathbb{K}\Gamma/I$ is finite dimensional, we may assume that we have an enumeration of Nontip(I) in sorted order. The techniques in Chapter 9 demonstrate how the enumeration can be done.

Given an enumeration of Nontip(I), we can perform a linear search through Nontip(I) and a to extract the coefficients. Thus the time complexity for line 8 is $t(f_8) = \Theta(dt(<_\Gamma))$.

The stated complexity is obtained by substituting the costs for path algebra elements into Corollary 3.2.2, replacing n for d, because d is an upper bound on the number of terms a path algebra element may contain. □

RegularMatrixRep(b, a) Compute the regular matrix representation of a wrt. b.

INPUT: Basis b and algebra element a.
OUTPUT: The regular matrix representation of a wrt. b.

```
1  d ← |b|   ▷ Dimension
2  r ← d × d array of 0_𝕂
3  for i ← 1 to d
4      do x ← a * b[i]
5          Assign entries in r by x = Σ_{j=1}^{d} r[j, i] * b[j]
6  return r
```

Figure 4.9: Pseudocode for converting an algebra element to its regular matrix representation.

4.4 Converting Between Encodings

Recall that matrix algebras and multiplication tables are general encodings for finite dimensional algebras. Several algorithms require properties of matrices (e.g., the characteristic polynomial) in order to function properly. Also, in some cases it may be more efficient to compute with a general encoding when computing with subalgebras of small dimension, for example, when the center of a large group algebra has small dimension relative to the dimension of the group algebra.

Let A be a finite dimensional algebra. Figure 4.9 contains the pseudocode for an encoding neutral algorithm to convert an element $a \in A$ to its corresponding regular matrix representation with respect to a given basis. To create the regular matrix representation for A, one simply executes the algorithm on each basis element.

Theorem 4.4.1. *The worst case encoding neutral time complexity for* RegularMatrixRep *is*

$$T_{\text{RegularMatrixRep}}(d) \quad = \quad \Theta\left(d(t(*_A) + t(f_5))\right)$$

Proof. The complexity is evident from the pseudocode. □

Corollary 4.4.2. *The worst case encoding neutral time complexity to compute the regular matrix representation for a d-dimensional algebra with respect to a given basis is* $\Theta(d^2(t(*_A) + t(f_5)))$

The basis used for computing the regular matrix representation can impact the conversion severely, by making the cost $t(f_5)$ high. The best choice of a basis is one that makes this cost very low. The cost of algebra multiplications generally changes very little between bases.

$$M_1 = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix} \qquad M_2 = \begin{bmatrix} 0 & 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 1 & 0 \end{bmatrix}$$

$$M_3 = \begin{bmatrix} 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \end{bmatrix} \qquad M_4 = \begin{bmatrix} 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \end{bmatrix}$$

$$M_5 = \begin{bmatrix} 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \end{bmatrix} \qquad M_6 = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

Figure 4.10: Regular matrix representation for $\mathbb{F}_3[S_3]$

Example 4.4.3. We use $\mathbb{F}_3 S_3$ and the elements of $S_3 = \{g_1, g_2, g_3, g_4, g_5, g_6\}$ as defined in Example 4.2.1. Recall that S_3 forms a basis for $\mathbb{F}_3 S_3$. In this example, the product of two basis elements is exactly one basis element. Furthermore, the coefficient of the resulting basis element is always $1_{\mathbb{K}}$. This produces matrices with a single $1_{\mathbb{K}}$ per column and row (also known as permutation matrices). The regular matrix representation for $\mathbb{F}_3 S_3$ is given in Figure 4.10.

We can also readily convert an algebra encoding for a finite dimensional algebra to a multiplication table encoding. Figure 4.11 shows an algorithm for constructing a multiplication table encoding for A with respect to a given basis. We note that this is similar to computing the regular representation of each basis element.

Theorem 4.4.4. *The worst case encoding neutral time complexity for* ConvertToMultTable *is*

$$T_{\text{ConvertToMultTable}}(d) \quad = \quad \Theta\left(d^2(t(*_A) + t(f_6)) + d^3(t(=_{\mathbb{K}}) + t(\text{Append}))\right)$$

Proof. The complexity is clear from the pseudocode. □

In practice, the cost $t(\text{Append})$ can be removed from the complexity, as Append can be imple-

ConvertToMultTable(b) Compute the regular matrix representation of a wrt. b.

INPUT: Basis b and algebra element a.
OUTPUT: The regular matrix representation of a wrt. b.

```
1   d ← |b|   ▷ Dimension
2   mt ← d × d array of [[], []]
3   for i ← 1 to d
4       do for j ← 1 to d
5              do x ← b[i] * b[j]
6                  Let c be the 1 × d array defined by x = Σ_{k=1}^{d} c[k] * b[k]
7                  for k ← 1 to d
8                      do if c[k] ≠ 0_𝕂
9                             then Append k to mt[i, j, 1]
10                                Append c[k] to mt[i, j, 2]
11
```

Figure 4.11: Pseudocode for computing a multiplication table encoding for an algebra with respect to a basis B.

mented with insignificant cost (e.g., using linked lists).

The time complexity of converting to a matrix algebra encoding or a multiplication table encoding is approximately the same. The matrix algebra encoding will generally result in much higher space usage than the multiplication table encoding. However, we can convert single elements to a matrix encoding, allowing us to use the matrix encoding as we need it.

4.5 Experimental Data

We have provided theoretical analyses for computing $Z(A)$, the center of an algebra A, and for converting between encodings. In this section, we provide experimental data and relate the data to the analyses. For our experiments, we used GAP version 4.1 fix 7 on a 333 MHz Pentium III machine with 112MB of RAM running FreeBSD version 3.4. GAP was given a workspace size of 64MB to reduce the impact of garbage collection.

We used only group algebras in our experiments because they are easily constructed and we can control their dimension. While we have implementations for path algebras available, constructing examples with the necessary properties we needed for running experiments is difficult, as such

examples do not occur naturally.

Group algebras were selected at random using GAP's "small group library" via the `SmallGroup` command. The group used during testing is shown in the data tables in Appendix A. Each experiment was executed five times, and the graphs below show the average over the five executions. The field used for the experiments is $\mathbb{F}_{53}$.

4.5.1 Converting Algebras

In this experiment, we converted each group algebra to a multiplication table encoding and to its regular matrix representation. The `StructureConstantsTable` command in GAP is used to convert to the multiplication table encoding. Each group algebra is converted to its regular matrix representation with two commands: `AdjointMatrix` and `IsomorphismMatrixAlgebra`. The `AdjointMatrix` command is the RegularMatrixRep algorithm in Figure 4.9. The `IsomorphismMatrixAlgebra` constructs the regular matrix representation isomorphism as a map in GAP. Each basis element in A is then converted using the `Image` command. A graph summarizing the results of the experiment are in Figure 4.12.

We see from the graph that the time for each method grows at approximately the same rate. The spikes displayed for the multiplication table and matrix algebra via `IsomorphismMatrixAlgebra` occur when the dimension is a prime number and is the result of the group encoding in GAP. In contrast, the times for matrix algebra via `AdjointMatrix` grows very smoothly.

To estimate the time complexity for each conversion, we hypothesize that the time complexity has the form $T(d) = cd^k$, where c and k are constants. Taking logarithms, we obtain $\ln T(d) = \ln c + k \ln d$. Using the linear modeling command `lm` in the R statistical package [1], we fit a line to a function of the form $\ln c + k \ln d$. For converting to multiplication tables, we obtain the estimate $\ln T(d) \approx 0.72111 + 1.59004 \ln d$. For converting to matrix algebras using `IsmorphismMatrixAlgebra`, we obtain the estimate $\ln T(d) \approx 0.41674 + 1.61115 \ln d$. Finally, the estimate for converting to matrix algebras using `AdjointMatrix` is $\ln T(d) \approx -2.11097 + 2.33895 \ln d$.

For the multiplication table and matrix algebra via `IsomorphismMatrixAlgebra`, these estimates are slightly better than the theoretical bounds. This is likely caused by the group algebra property that the product of two basis elements always results in a single basis element. For matrix algebras via `AdjointMatrix`, the estimate is slightly worse, but that may be due to the cost of algebra operations not being constant. In any case, the behaviors of `StructureConstantsTable` and

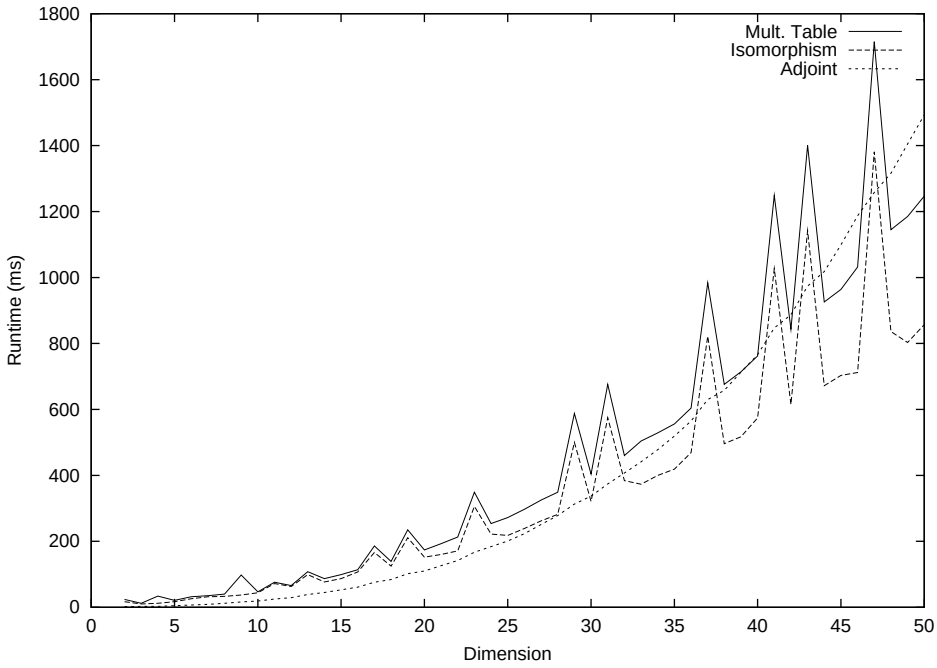


Figure 4.12: The time required to convert a group algebra of dimension d to a multiplication table, a matrix algebra using `AdjointMatrix`, and a matrix algebra using `IsmorphismMatrixAlgebra`. Runtimes are in CPU milliseconds.

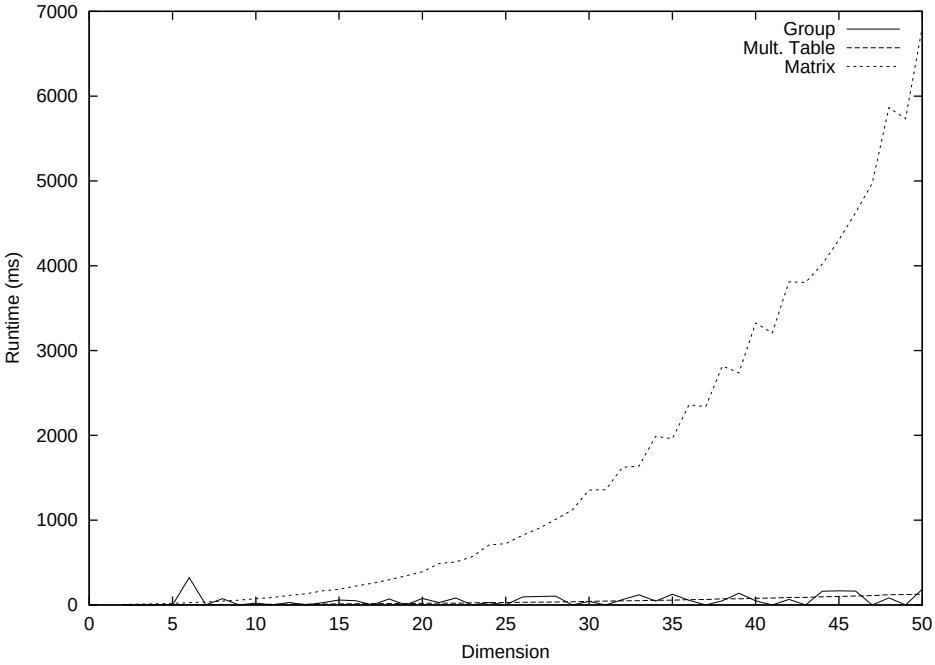


Figure 4.13: The results of calculating $Z(A)$, where A is a group algebra. The "Group" algorithm computes conjugacy classes of the group. The multiplication table algorithm is applied to the multiplication table encoding for A. The standard algorithm is applied to the regular matrix representation encoding for A. Runtimes are in CPU milliseconds.

`IsomorphismMatrixAlgebra` are interesting and warrant further investigation. We recommend using `IsomorphismMatrixAlgebra` for converting to matrix algebras in general because the complexity appears to be better.

4.5.2 Computing the Center

For computing the center, we have seen three different algorithms. In this experiment, we calculate the center of a group algebra using the native group algebra algorithm (i.e., via conjugacy classes of the group), using a multiplication table encoding, and with the standard algorithm using the regular matrix representation. The costs of converting to the multiplication table encoding and regular matrix representation are not included in the data. A graph summarizing the data is shown in Figure 4.13.

In the graph, we see immediately that using the standard algorithm on matrices takes the most time. The algorithm computing conjugacy classes of the group is very irregular, but generally performs better than the standard algorithm. The multiplication table algorithm performs very well in this experiment. The results for the multiplication table are taking advantage of the group algebra structure (making the multiplication table very sparse). These experiments do not exemplify worst case behavior, however, we do see that multiplication tables can take advantage of the sparseness of the encoding, in contrast to the matrix algebra encoding.

We estimate the complexity as we did when converting algebras. We hypothesize that the complexity has the form $T(d) = cd^k$, where c and k are constants and take logarithms of both sides obtaining $\ln T(d) = \ln c + k \ln d$. Using the linear modelling command `lm` in the R statistical package [1], we fit a line to a function of the form $\ln c + k \ln d$. For the group algebra specific algorithm, this failed due to zeros in the data. From the nature of the graph, we hypothesize that $1 \leq k \leq 2$. For the multiplication table specific algorithm, we obtain an estimate of $-0.90624 + 1.37908 \ln d$. We obtain an estimate of $-0.88947 + 2.37629 \ln d$ for the standard algorithm applied to the regular matrix representation.

For the multiplication table and standard algorithms, we perform better than the theoretical estimates. In the case of the multiplication table algorithm, this is most likely due to the nice structure of group algebras. For the standard algorithm, the reason for better performance is less clear. GAP employs standard algorithms for matrix multiplication and solving systems of equations, both of which have a higher complexity cost than the estimate. It is possible that sparse matrices in the computations account for the better performance, or that more data is needed to obtain a better estimate.

4.6 Summary

In this chapter, we examined three possible encodings for algebras: multiplication tables, group algebras, and path algebras. We analyzed the time complexity of the algebra operations for each encoding. The worst case time complexity for multiplication tables is approximately the same as for matrix algebras, although the average case is probably better for multiplication tables when multiplying sparse elements. The time complexity for group algebras and path algebras include the cost of multiplying elements other than field elements, so comparing complexities is difficult.

The analysis for CENTER from Section 3.2 was performed for each encoding. Specialized algorithms for the multiplication table encoding and group algebras were obtained. For multiplication tables, the theoretical complexity of the algorithm removes the cost of multiplication in the algebra, suggesting an improvement over the standard algorithm.

We presented experimental data for converting a group algebra to its multiplication table encoding and its regular matrix representation. The cost of conversion is approximately the same in both cases. We also presented data comparing the different algorithms for computing the center $Z(A)$ of a group algebra A in different encodings. As expected, the group algebra specific and multiplication table specific algorithms perform much better than the standard algorithm. In all three cases, however, the experimental data demonstrates performance better than what was theoretically proposed.

In subsequent chapters, we analyze and discuss the effects of the encodings presented in this chapter on the algebraic algorithms we study.

Chapter 5

Decomposition of Algebras

In this chapter, we present the first of three algebraic problems appearing in this dissertation: algebra decomposition. Throughout, we assume that A is a d-dimensional $\mathbb{K}$-algebra. We restrict the field $\mathbb{K}$ to $\mathbb{F}_q$, finite fields, and $\mathbb{Q}$, the rationals because they are the fields most frequently used in practice.

In Section 5.1, we review the structure theorems that form a basis for our problem. In Section 5.2, we analyze and compare algorithms for computing primitive central idempotents for semisimple algebras. Section 5.3 discusses the identification of simple algebras with full matrix algebras over division rings. In Section 5.4, we analyze and compare algorithms for computing the Jacobson radical of an algebra. Section 5.5 discusses lifting idempotents from a quotient algebra to the original algebra. We present experimental data comparing the presented algorithms in Section 5.6. Finally, we summarize the chapter and recommend algorithms to use for algebra decomposition in Section 5.7

5.1 Structure Theorems

As stated in the introduction, the classification of algebraic structures is a primary goal of algebraists. Wedderburn [96] laid the foundation for the classification of semisimple algebras. Wedderburn's results are standard content in algebra textbooks such as Dummit and Foote [31, Sec 15.3], Farb and Dennis [36, Ch. 1] and Hungerford [57, Ch. 9]. We review the main structure theorems and several results that form the groundwork for the presented computer algorithms.

If $A = A_1 \oplus A_2 \oplus \cdots \oplus A_n$, where each A_i is an ideal of A, the set of ideals $\{A_i\}_{i=1}^n$ is a

decomposition of A. If a decomposition of A exists such that $n \geq 2$, then A is *decomposable*, otherwise A is *indecomposable*. In general, each ideal A_i of A is not a subalgebra of A, because A_i does not contain 1_A.

We are particularly interested in the situation where each A_i is indecomposable as an algebra. The next theorem establishes the uniqueness of decompositions of A into indecomposable ideals.

Theorem 5.1.1. *The (finite dimensional) algebra A has a unique decomposition*

$$A = A_1 \oplus A_2 \oplus \cdots \oplus A_n$$

into a direct sum of ideals A_i such that each A_i is indecomposable as an algebra.

Proof. See Curtis and Reiner [21, pg. 378] or Alperin [5, pp. 92–93]. □

Each ideal A_i in Theorem 5.1.1 is a *block* of A. The following is the first algebraic problem we consider.

ALGEBRA DECOMPOSITION
INSTANCE: A (finite dimensional) algebra A.
SOLUTION: Blocks $A_1, \ldots, A_n$ of A.

It is easily seen that finding central idempotents of A leads to a decomposition of A. Let e be a central idempotent in A such that $e \neq 1_A$. Clearly, the sets Ae and $A(1_A - e)$ are ideals of A. The intersection $Ae \cap A(1_A - e) = 0$, because if $a \in Ae \cap A(1_A - e)$, then $a = ex(1_A - e) = xe(1_A - e) = x(e - e^2) = 0$. If $a \in A$, then $a1_A = a(e + 1_A - e) = ae + a(1_A - e) \in Ae \oplus A(1_A - e)$ and $Ae \oplus A(1_A - e) \subseteq A$. Hence, $A = Ae \oplus A(1_A - e)$ is a decomposition of A. The following theorem states that we can use primitive central idempotents for finding the blocks of A.

Theorem 5.1.2. *Let $1_A = e_1 + e_2 + \cdots + e_n$ be a decomposition of 1_A into primitive central orthogonal idempotents. The following two statements hold:*

1. *The decomposition $\{e_i\}_{i=1}^n$ of 1_A is unique.*
2. *The direct sum*

$$A = Ae_1 \oplus Ae_2 \oplus \cdots Ae_n$$

is the decomposition of A into indecomposable ideals Ae_i.

Proof. See Theorems 4.6 and 4.7 in Nagao and Tsushima [77, pp. 19–20]. □

We have reduced the problem **ALGEBRA DECOMPOSITION** to finding a set of primitive central orthogonal idempotents $\{e_i\}_{i=1}^n$ such that $\sum_{i=1}^n = 1_A$. We cover algorithms for finding primitive central orthogonal idempotents in Section 5.2.

Although we have a method for decomposing algebras into indecomposable ideals, we still know very little about the structure of the indecomposable ideals comprising such a decomposition. In general, the structure of indecomposable ideals is still being classified. There does exist a class of algebras that we have explicit knowledge of the underlying structure.

The *Jacobson radical* $J(A)$ of A is the intersection of all maximimal right ideals of A. The Jacobson radical $J(A)$ is also an ideal of A. If $J(A) = 0$, then A is a *semisimple* algebra. Determining whether A is semisimple requires computing $J(A)$ in general. The computation of $J(A)$ is discussed in Section 5.4. The following theorem is due to Wedderburn.

Theorem 5.1.3 (Wedderburn Structure Theorem). *If A is a (finite dimensional) semisimple algebra such that*

$$A \quad = \quad A_1 \oplus A_2 \oplus \cdots \oplus A_n$$

is the decomposition of A into indecomposable ideals A_i, then each ideal A_i is a simple algebra.

Proof. Several proofs exist; see Theorem 25.15 and Corollary 25.22 in Curtis and Reiner [21]. □

The ideals A_i in Theorem 5.1.3 are the *simple components* of A. The structure of simple algebras is well known by the Wedderburn-Artin theorem.

Theorem 5.1.4 (Wedderburn-Artin Theorem). *If A is a (finite dimensional) simple $\mathbb{K}$-algebra, then $A \cong M_n(D)$, a full matrix algebra over a division ring D, which is a finite (possibly noncommutative) extension of $\mathbb{K}$.*

Proof. See Theorem 26.4 in Curtis and Reiner [21]. □

When algebras are semisimple, we wish to determine as much information as possible. The problem **SEMISIMPLE ALGEBRA DECOMPOSITION** summarizes our goal for semisimple algebras.

SEMISIMPLE ALGEBRA DECOMPOSITION

INSTANCE: A semisimple algebra A.

SOLUTION: Simple components $A_1, \ldots, A_n$ of A, integers n_i, division rings D_i, and maps $\rho_i : A_i \rightarrow M_{n_i}(D_i)$, such that each ρ_i is an algebra isomorphism from A_i to the full matrix algebra $M_{n_i}(D_i)$, for $1 \leq i \leq n$.

As a result of Theorem 5.1.2 and Theorem 5.1.3, finding primitive central idempotents is sufficient to find the simple components of a semisimple algebra. In Section 5.3, we focus on explicitly constructing isomorphims between simple components and full matrix algebras over division rings. We will see situations where finding isomorphisms is computationally difficult.

5.2 Primitive Central Idempotents

In the introduction of this chapter, we saw that primitive central idempotents play an important role in finding solutions for **ALGEBRA DECOMPOSITION** and **SEMISIMPLE ALGEBRA DECOMPOSITION**. In particular, a decomposition of $1_A = e_1 + e_2 + \cdots + e_n$ such that $\{e_1, e_2, \cdots e_n\}$ are primitive central orthogonal idempotents in A leads to a (actually the) solution of **ALGEBRA DECOMPOSITION**. We show how to compute the center $Z(A)$ of A in Section 3.2. The central idempotents of A are found in $Z(A)$ by definition. Hence, for the remainder of this section, we assume A is commutative.

In this section, several algorithms for finding a decomposition 1_A into primitive central orthogonal idempotents are analyzed and compared. We highlight the important features of each algorithm and give experimental data on the performance of each algorithm for each encoding discussed in Section 3.2.3 and Chapter 4.

The following theorem allows us to restrict the problem domain to semisimple algebras.

Theorem 5.2.1. *Let $\overline{A} = A/J(A)$. If $1_{\overline{A}} = \overline{e_1} + \overline{e_2} + \cdots + \overline{e_n}$ is a decomposition of $1_{\overline{A}}$ into primitive central orthogonal idempotents $\overline{e_i}$ of $\overline{A}$, then $1_A = e_1 + e_2 + \cdots + e_n$ is a decomposition of 1_A into primitive central orthogonal idempotents e_i of A such that $e_i \equiv \overline{e_i} \mod J(A)$.*

Proof. The Jacobson radical $J(A)$ is a nilpotent ideal (see Theorem 2.4 in Farb and Dennis [36] for example). The theorem follows from Theorems 4.9, 4.10, 4.11, and 4.12 in Nagao and Tsushima [77]. Proofs also exist in Curtis and Reiner [22, Sec. 6] and in Gianni, et al. [44, Sec. 4] □

We give an explicit construction of primitive central idempotents in a non-semisimple algebra A from primitive central idempotents found in $A/J(A)$ in Section 5.5. The commutative algebra A is assumed to be semisimple for the remainder of this section.

5.2.1 Friedl and Rónyai

The algorithm of Friedl and Rónyai [39, 86] appears to be the first published algorithm for **ALGEBRA DECOMPOSITION**. Their algorithm does not explicitly create primitive central idempotents but does find simple ideals $A_1, A_2, \ldots, A_n$ such that $A = A_1 \oplus A_2 \oplus \cdots \oplus A_n$.

Let $a \in A$. The *minimal polynomial* $m_a(x)$ of a is the minimal degree monic polynomial $f \in \mathbb{K}[x]$ such that $f(a) = 0_{\mathbb{K}}$. The key observation for the algorithm of Friedl and Rónyai is stated in the following theorem.

Theorem 5.2.2. *Let $a \in A$. If $m_a(x) = f(x)g(x)$ such that $f, g \in \mathbb{K}[x]$, $\deg f > 0$, and $\deg g > 0$, then $A = I \oplus J$ is a decomposition of A into ideals, such that $I = f(a)A$ and $J = g(a)A$.*

Proof. See Proposition 6.5 in Friedl and Rónyai [39]. □

Theorem 5.2.2 leads to the divide and conquer approach used by Friedl and Rónyai. Pseudocode for the main divide and conquer procedure appears in Figure 5.1.

Friedl and Rónyai include two versions of the SPLIT: one if A is a $\mathbb{Q}$-algebra and one of A is an $\mathbb{F}_q$-algebra. The primary difference between the two versions is the additional calculations (i.e., the functions REDUCTION and PRIMELEM in Friedl and Rónyai [39]) needed to polynomially bound the growth of coefficient sizes for $\mathbb{Q}$-algebras.

The technical details of bounding the growth of coefficients for $\mathbb{Q}$-algebras are quite complex. For our work, it is sufficient to focus on the analysis of the algorithm for $\mathbb{F}_q$-algebras and leave a detailed analysis of the algorithm for $\mathbb{Q}$-algebras as future work.

Pseudocode for the $\mathbb{F}_q$-algebra version of SPLIT appears in Figure 5.2. We see that SPLIT uses the result from Theorem 5.2.2 to find ideals of A contained in I.

Constructing field extensions on line 16 of SPLIT makes this algorithm difficult to analyze in detail. However, there are several situations when constructing field extensions is unnecessary, including when $a = a^q$ for each $a \in A$. This situation occurs in Section 5.2.2 and in our experiments in Section 5.6.

FRMAIN(A) Find a decomposition of A into simple ideals.

INPUT: A commutative semisimple algebra A.
OUTPUT: A list of simple ideals comprising a decomposition of A.

```
1   ideals ← an empty queue
2   ideals.enqueue(A)
3   minIdeals ← an empty queue
4   while not ideals.empty
5         do ideal ← ideals.dequeue
6            subIdeals ← SPLIT(ideal)
7            if not subIdeals.empty
8              then ideals.enqueue(subIdeals.dequeue)
9                   ideals.enqueue(subIdeals.dequeue)
10             else minIdeals.enqueue(ideal)
11  return minIdeals   ▷ convert to list if necessary
```

Figure 5.1: Main divide and conquer procedure for Friedl and Rónyai algorithm.

Lemma 5.2.3. *Let A be a d-dimensional $\mathbb{F}_q$-algebra such that* SPLIT *never constructs a field extension. The worst case encoding neutral time complexity of* SPLIT *executed on A is*

$$T_{\text{SPLIT}}(d,q) \quad = \quad \Theta\left(d(T_{\text{MINPOLY}}(d) + T_{\text{FACTOR}}(d,q) + t(*_A) + t(+_A) + t(*_{\mathbb{K},A}))\right).$$

Proof. The worst case occurs when the last basis element is encountered during the execution, and the minimal polynomial factors nontrivially. In this case, the **for** loop on line 5 is executed d times. This gives d executions of MINPOLY and FACTOR within the body of the loop. The contribution is $\Theta(d(T_{\text{MINPOLY}}(d) + T_{\text{FACTOR}}(d,q)))$.

Once the minimal polynomial factors nontrivially, we evaluate two factors `g` and `h` on the basis element b_d. The total cost of this evaluation is no worse than evaluating the minimal polynomial `f` on b_d, which has degree at most d. The evaluation costs at most $\Theta(d(t(*_A) + t(+_A) + t(*_{\mathbb{K},A})))$.

Totalling the costs gives the stated complexity. □

Theorem 5.2.4. *Let A be a d-dimensional $\mathbb{F}_q$-algebra such that* SPLIT *never creates a field extension. The worst case encoding neutral time complexity of* FRMAIN *executed on A is*

$$T_{\text{FRMAIN}}(d,q) \quad = \quad O\left(d^2(T_{\text{MINPOLY}}(d) + T_{\text{FACTOR}}(d,q) + t(*_A) + t(+_A) + t(*_{\mathbb{K},A}))\right),$$

SPLIT(I) Finds ideals contained in I.

INPUT: An ideal I of a $\mathbb{F}_q$-algebra.
OUTPUT: A queue containing zero or two non-trivial ideals in I.

```
1   b ← I.basis
2   d ← |b|
3   F ← I.field
4   ideals ← empty queue
5   for i ← 1 to d
6       do f ← MinPoly(F, b[i])
7          factors ← Factor(f)   ▷ over F
8          if |factors| > 1   ▷ non-trivial factorization
9            then g ← factors[1]
10                h ← f/g
11                J ← Ideal(I, [g(b[i])])
12                ideals.enqueue(J)
13                J ← Ideal(I, [h(b[i])])
14                ideals.enqueue(J)
15                return ideals
16           else F ← FieldExt(F, b[i])
17  return ideals
```

Figure 5.2: Pseudocode for splitting ideals.

Proof. To simplify our analysis, we view FRMAIN as recursive calls to SPLIT. In the worst case, we always obtain a splitting of ideals into one of dimension one and its complement. Thus, the complexity can be stated by the recurrence relation

$$\begin{aligned} T_{\text{SPLIT}}(d,q) &= T_{\text{SPLIT}}(d-1,q) \\ &\quad + \Theta\left(d(T_{\text{MINPOLY}}(d) + T_{\text{FACTOR}}(d,q) + t(*_A) + t(+_A) + t(*_{\mathbb{K},A}))\right). \end{aligned}$$

We assume the complexity $T_{\text{MINPOLY}}(d) = \Omega(d^2)$ operations. The naive algorithm for MINPOLY using Gaussian elimination clearly satisfies this complexity. More efficient randomized algorithms such as the algorithm of Giesbrecht [45] requires $\Omega(MM(d)) = \Omega(d^2)$ operations in $\mathbb{F}_q$, where $MM(d)$ is the complexity of matrix multiplication.

For FACTOR, we assume the randomized version of Berlekamp's algorithm [10] is used. Theorem 7.4.6 in Bach and Shallit [8], states that $T_{\text{FACTOR}}(d,q) = O((d\log^2 d) + (\log q \log^2 d))$ bit operations.

Expanding the recurrence relation, we obtain

$$T_{\text{SPLIT}}(d,q) = \sum_{i=1}^{d} \Theta\left(i(T_{\text{MINPOLY}}(i) + T_{\text{FACTOR}}(i,q) + t(*_A) + t(+_A) + t(*_{\mathbb{K},A}))\right).$$

This is bounded above by

$$T_{\text{SPLIT}}(d,q) = O\left(d^2(T_{\text{MINPOLY}}(d) + T_{\text{FACTOR}}(d,q) + t(*_A) + t(+_A) + t(*_{\mathbb{K},A}))\right),$$

giving the stated complexity by replacing SPLIT with FRMAIN. □

5.2.2 Gianni, Miller, and Trager

In this subsection, we present the algorithm of Gianni, Miller, and Trager [44]. This algorithm explicitly constructs primitive central idempotents of $\mathbb{F}_q$-algebras. Additionally, Gianni, et al. describe a Hensel lifting technique to apply their algorithm for $\mathbb{F}_q$-algebras to find idempotents in $\mathbb{Q}$-algebras. We do not include the details for finding idempotents in $\mathbb{Q}$-algebras, but we summarize the principal ideas to obtain a rough analysis.

Unless otherwise noted, A is an $\mathbb{F}_q$-algebra and is not necessarily semisimple. A semisimple subalgebra of A that contains all of the idempotents is first constructed.

The *Frobenius homomorphism* $\psi_q : A \to A$ maps $a \mapsto a^q$. Let $A^{\psi_q} = \{a \in A \mid a^q = a\}$, the set of elements in A fixed under the Frobenius homomorphism. The set A^{ψ_q} is a subalgebra of A as

PowerSubalgebra(A, q) Compute A^{ψ_q}.

INPUT: An $\mathbb{F}_q$-algebra A and q the order of $\mathbb{F}_q$.
OUTPUT: The subalgebra A^{ψ_q}

```
1   b ← A.basis
2   d ← |b|
3   for i ← 1 to d
4       do x ← b[i]^q − b[i]
5           Define X[i, j] by x = Σ_{j=1}^{d} X[i, j] ∗ b[j]
6   V ← Nullspace(X)
7   α ← V.basis
8   n ← |α|
9   for i ← 1 to n
10      do z[i] ← Σ_{j=1}^{d} α[i, j] ∗ b[j]
11  return Subalgebra(A, z)
```

Figure 5.3: Pseudocode for calculating A^{ψ_q}.

follows. Let $a, b \in A^{\psi_q}$. We obtain $(a+b)^q = a^q + b^q = a + b$ because A^{ψ_q} in an $\mathbb{F}_q$-algebra. The product $(ab)^q = a^q b^q = ab$ because A is commutative. Clearly, 0_A and 1_A are contained in A^{ψ_q}. Hence, A^{ψ_q} is a subalgebra.

We also see that A^{ψ_q} contains every idempotent in A because $e^2 = e$, so $e^q = e$, if e is idempotent. The next theorem shows that A^{ψ_q} is semisimple.

Theorem 5.2.5. *The algebra A^{ψ_q} is semisimple.*

Proof. Every element in $J(A^{\psi_q})$ is nilpotent because $J(A^{\psi_q})$ is nilpotent (see Farb and Dennis [36, Thm. 2.4]). However, the only nilpotent element in A^{ψ_q} is clearly 0_A. Hence, $J(A^{\psi_q}) = \{0_A\} = 0$. □

We can find A^{ψ_q} by solving a system of linear equations much like we did for computing $Z(A)$ because ψ_q is an algebra homomorphism. We leave the details to the reader and present only the pseudocode for PowerSubalgebra in Figure 5.3.

Theorem 5.2.6. *The worst case encoding neutral time complexity for* PowerSubalgebra *execut-*

ing on a d-dimensional $\mathbb{F}_q$-algebra is

$$T_{\textsc{PowerSubalgebra}}(d,q) = \Theta\left((d\log q)t(*_A) + d^2(t(+_A) + t(*_{\mathbb{K},A})) + d^3(t(*_{\mathbb{K}}) + t(+_{\mathbb{K}})) + dt(f_5)\right).$$

Proof. There are $\log q$ algebra multiplications on line 4 by implementing exponentiation through repeated squaring (see Cormen, et al. [19, pg. 829]). This line is executed d times for a contribution of $\Theta((d\log q)t(*_A))$. Line 4 also contains one subtraction, for a contribution of $\Theta(dt(+_A))$ to the complexity.

Line 6 is implemented with standard Gaussian elimination techniques on a $d \times d$ $\mathbb{F}_q$-matrix. This contributes $\Theta(d^3(t(*_{\mathbb{K}}) + t(+_{\mathbb{K}})))$ to the complexity.

In the worst case, $n = d$. Line 10 contains d scalar multiplications and $d-1$ algebra additions. These are executed up to d times. The total contribution is $\Theta(d^2(t(*_{\mathbb{K},A}) + t(+_A)))$.

We have left the cost of line 5 undetermined, because this cost varies depending on the algebra encoding used. The stated complexity follows directly. □

An alternative algorithm to calculate A^{ψ_q} for a group algebra $A = \mathbb{K}G$ when the characteristic of $\mathbb{F}_q$ does not divide the order of G is suggested by Davis [24]. It is not necessary for $\mathbb{K}G$ to be commutative to apply Davis' algorithm. We experimentally compare Davis' suggested algorithm with PowerSubalgebra in Section 5.6.

The algebra A^{ψ_q} has a well defined structure as seen in the following theorem.

Theorem 5.2.7. *The $\mathbb{F}_q$-algebra A^{ψ_q} is isomorphic to the direct sum of m copies of $\mathbb{F}_q$, where m is the number of primitive idempotents in A^{ψ_q}; that is,*

$$A^{\psi_q} \cong \oplus_{i=1}^{m} \mathbb{F}_q.$$

Proof. This is a special case of Davis [24, Thm. 2.1]. □

Gianni, et al. outline a brute force algorithm for finding primitive idempotents by finding roots of characteristic polynomials of each basis element in A^{ψ_q} (viewing each basis element by its regular matrix representation). This algorithm is essentially equivalent to the Friedl and Rónyai algorithm in the previous subsection. In addition to the brute force algorithm, Gianni, et al. give a randomized algorithm for constructing idempotents using the following theorem.

Theorem 5.2.8. *Let $v \in A^{\psi_q}$ such that $v \neq 0_A$. We have two cases,*

1. *If* $q = 2^n$*, then* $\sum_{i=0}^{m-1} v^{2^i}$ *is idempotent.*

2. *If* $q = p^n$ *where* $p \neq 2$*, then let* $t = v^{(q-1)/2}$*. Both* $(t^2 + t)/2$ *and* $(t^2 - t)/2$ *are idempotent or* 0_A*.*

Proof. First, assume $q = 2^n$. Then $\left(\sum_{i=0}^{n-1} v^{2^i}\right)^2 = \sum_{i=1}^{n} v^{2^i}$ because $\mathbb{F}_q$ has characteristic 2. We have $v^{2^n} = v = v^{2^0}$, so $\sum_{i=0}^{n-1} v^{2^i}$ is idempotent.

Now, assume $q = p^n$ such that $p \neq 2$. Let $t = v^{\frac{q-1}{2}}$. We note that

$$(v^{q-1})^2 = v^{2q-2} = v^q v^{q-2} = v v^{q-2} = v^{q-1}$$

and

$$v^{q-1} v^{\frac{q-1}{2}} = v^q v^{\frac{q-3}{2}} = v v^{\frac{q-3}{2}} = v^{\frac{q-1}{2}}.$$

It clearly follows that $(t^2 + t)/2$ and $(t^2 - t)/2$ are idempotent or 0_A. □

The next theorem demonstrates that a non-trivial idempotent can be constructed at random with a non-zero probability.

Theorem 5.2.9. *The idempotents constructed in Theorem 5.2.8 occur with probability* $\geq 1/2$ *if* v *is uniformly selected at random from* A^{ψ_q}*.*

Proof. See Lemma 2 in Gianni, et al. [44]. □

Elements can be selected in A^{ψ_q} uniformly at random given a basis for A^{ψ_q}, which we construct using PowerSubalgebra. We now have a strategy for finding primitive central idempotents in A^{ψ_q}. Let I be an ideal in A^{ψ_q}. If $\dim I = 1$, then I is a copy of $\mathbb{F}_q$, and 1_I is a primitive central idempotent. Otherwise, select a non-zero element $v \in I$ uniformly at random. Using Theorems 5.2.8 and 5.2.9, we construct a non-trivial idempotent e with probability $\geq 1/2$. If e is a non-trivial idempotent, construct two new ideals Ie and $I(1_I - e)$ and repeat the process. We note that $1_I \neq 1_A$ unless $I = A^{\psi_q}$. Pseudocode for the Gianni, et al. randomized algorithm GianniIdempotents appears in Figure 5.4.

Theorem 5.2.10. *The expected encoding neutral time complexity for* GianniIdempotents *given*

GianniIdempotents(A) Find primitive central idempotents in A.

INPUT: A commutative $\mathbb{F}_q$-algebra A with $A = A^{\psi_q}$.
OUTPUT: Primitive central idempotents in A.

```
1   b ← A.basis
2   d ← |b|
3   if d = 1
4     then return [1_A]
5   q ← |A.field|
6   p ← A.field.characteristic
7   n ← log_p q
8   two ← 1_{F_q} + 1_{F_q}
9   id1 ← 0_A
10  do
11      v ← an element in A selected at random
12      if p = 2
13        then e ← Σ_{i=0}^{n-1} v^{2^i}
14             f ← 1_A
15        else t ← v^{(q-1)/2}
16             e ← (t^2 + t)/two
17             f ← (t^2 - t)/two
18      if e ≠ 1_A and e ≠ 0_A
19        then id1 ← e
20             id2 ← 1_A - e
21        else if f ≠ 1_A and f ≠ 0_A
22             then id1 ← f
23                  id2 ← 1_A - f
24  while id1 = 0_A
25  I ← Ideal(A, [id1])
26  J ← Ideal(A, [id2])
27  return GianniIdempotents(I) ∪ GianniIdempotents(J)
```

Figure 5.4: Pseudocode for the Gianni, et al. randomized idempotent algorithm.

a d-dimensional $\mathbb{F}_q$-algebra A such that $A = A^{\psi_q}$ is

$$\begin{aligned} T_{\textsc{GianniIdempotents}}(d,q) &= O\Big(d \log q (t(*_A) + t(+_A)) \\ &\quad + d^2 (t(\textsc{Random}_{\mathbb{K}}) + t(*_{\mathbb{K},A}) + t(+_A)) \\ &\quad + t(=_A)\Big). \end{aligned}$$

Proof. From Theorem 5.2.9, an idempotent is constructed from an element chosen from A^{ψ_q} uniformly at random with $\geq 1/2$ probability. We have a Bernoulli trial with a probability of $1/2$ for success (and failure). The expected number of times the **do** loop on line 10 executes is 2 (see Cormen, et al. [19, Sec. 6.4]).

On line 11, we choose d random elements from $\mathbb{K}$, perform d scalar multiplications (one for each basis element), and $\Theta(d)$ algebra additions.

If the characteristic of $\mathbb{F}_q$ is 2, then on line 13, we have $\Theta(n) = \Theta(\log q)$ algebra multiplications from repeated squarings and $\Theta(\log q)$ algebra additions. Otherwise, we have $\Theta(\log q)$ algebra multiplications on line 15, by using repeated squaring to implement exponentiation.

The costs of algebraic operations in the remainder of the algorithm are dominated by the above costs because the expected number of times they are executed is constant. The only operation not accounted for is $=_A$, which is executed an expected $\Theta(1)$ number of times.

There are two recursive calls to GianniIdempotents on line 27. In the worst case, one ideal is one dimensional and the other is its complement. We obtain the recurrence relation

$$\begin{aligned} T_{\textsc{GianniIdempotents}}(d,q) &= T_{\textsc{GianniIdempotents}}(d-1,q) \\ &\quad + O\Big(\log q (t(*_A) + t(+_A)) \\ &\quad + d(t(\textsc{Random}_{\mathbb{K}}) + t(*_{\mathbb{K},A}) + t(+_A)) \\ &\quad + t(=_A)\Big), \end{aligned}$$

omitting the term $T_{\textsc{GianniIdempotents}}(1,q)$, which has constant cost. We obtain the complexity

$$\begin{aligned} T_{\textsc{GianniIdempotents}}(d,q) &= \sum_{i=1}^{d} O\Big(\log q (t(*_A) + t(+_A)) \\ &\quad + i(t(\textsc{Random}_{\mathbb{K}}) + t(*_{\mathbb{K},A}) + t(+_A)) \\ &\quad + t(=_A)\Big), \end{aligned}$$

which is bounded above by

$$\begin{aligned} T_{\text{GianniIdempotents}}(d,q) &= O\Big(d\log q(t(*_A)+t(+_A)) \\ &\quad + d^2(t(\text{Random}_{\mathbb{K}})+t(*_{\mathbb{K},A})+t(+_A)) \\ &\quad + t(=_A)\Big). \end{aligned}$$

□

To find idempotents in a $\mathbb{Q}$-algebra A, Gianni, et al. use a Hensel lifting technique that is covered in detail in Section 5.5. An $\mathbb{F}_q$-algebra B containing a unique image of each idempotent in A (see Proposition 9 in Gianni [44]). The idempotents in B are found using PowerSubalgebra and GianniIdempotents. Let $e = \{e_1, e_2, \ldots, e_m\}$ be the set of idempotents found. Then every sum $\hat{e}$ of a subset of e is lifted and tested to see if the coefficients of $\hat{e}$ fall within a specific bound. If the coefficients of $\hat{e}$ satisfy the bound, then $\hat{e}$ is an idempotent in A.

Although we left out some detail in the previous description, it is obvious that the Gianni, et al. algorithm is exponential in d in the worst case. This is because B can contain d idempotents (when $B = B^{\psi_q}$), and there are 2^d subsets of those d idempotents. Thus, the time complexity for finding idempotents in $\mathbb{Q}$-algebras using the Gianni, et al. algorithm is clearly not satisfactory. An efficient algorithm for finding primitive central idempotents in $\mathbb{Q}$-algebras is presented in Section 5.2.4.

5.2.3 Davis' Algorithm

In this subsection, we present the algorithm of Davis [24]. This algorithm finds primitive central idempotents in (commutative) $\mathbb{F}_q$-algebras. Davis works in a generalization of the power subalgebra A^{ψ_q}. Again, we lift the restriction of A being semisimple because the generalization of A^{ψ_q} is semisimple. In this subsection, let $q = p^n$. Recall $A^{\psi_q} = \{a \in A \mid a^q = a\}$ is a subalgebra of A. Davis considers the set $A^{\psi_{p^k}} = \left\{a \in A \mid a^{p^k} = a\right\}$. It is straightforward to show that for each positive divisor k of n, $A^{\psi_{p^k}}$ is an $\mathbb{F}_{p^k}$-algebra with one 1_A. The following theorem demonstrates that the primitive idempotents of A are contained in $A^{\psi_{p^k}}$.

Theorem 5.2.11. *For each positive divisor k of n, $A^{\psi_{p^k}} = \oplus_{i=1}^{m} \mathbb{F}_{p^k} e_i$, where $e_1, \ldots, e_m$ are the primitive (central) idempotents of A.*

Proof. See Theorem 2.1 in Davis [24]. □

The algebra $A^{\psi_q p^k}$ can be constructed using PowerSubalgebra from Section 5.2.2 with p^k passed as a parameter in place of q.

The key idea behind the algorithm of Davis is finding a common eigenvector for a set of basis elements for $A^{\psi_{p^k}}$. Let $\{b_1, \ldots, b_m\}$ be a $\mathbb{F}_{p^k}$ basis of $A^{\psi_{p^k}}$. If a non-zero element $v \in A^{\psi_q}$ is a common eigenvector of $\{b_1, \ldots b_m\}$, then v is an eigenvector for each element in $A^{\psi_{p^k}}$. The algebra $A^{\psi_{p^k}} v$ is equal to $\mathbb{F}_{p^k} v$, and some primitive idempotent e_i exists in $\mathbb{F}_{p^k} v$. The idempotent e_i is a scalar multiple of v. If $v = \alpha e_i$ for $\alpha \in \mathbb{F}_{p^k}$, then $v^2 = \alpha v$ and we can compute $e_i = \alpha^{-1} v$. The process is then repeated for the ideal $A^{\psi_{p^k}}(1_A - e_i)$.

To compute a common eigenvector, suppose u_0 is a nonzero common eigenvector of $\{b_1, \ldots, b_{j-1}\}$. but not of b_j. Let α_i be defined by $\mathbb{F}_{p^k} = \left\{\alpha_1 = 0_{\mathbb{F}_{p^k}}, \alpha_2, \ldots, \alpha_{p^k}\right\}$. The last non-zero term v of the sequence $u_s = (b_j - \alpha_s) u_{s-1}$, for $1 \leq s \leq p^k$, is an eigenvector of b_j (see Davis [24, pg. 240]). The element v is also an eigenvector of $\{b_1, \ldots, b_{j-1}\}$ because $A^{\psi_{p^k}}$ is commutative and v is in the ideal generated by u_0.

Davis describes several variations for finding a common eigenvector of $\{b_1, \ldots, b_m\}$ more efficiently. We have implemented the "polynomial factorization" approach appearing on pages 246–248. Furthermore, Davis includes a modified version of the basic algorithm that uses only a subset (in most cases) of the basis to compute the primitive idempotents. We include pseudocode for the modified algorithm of Davis in Figure 5.5. The following theorem is included in Davis. We have slightly modified the theorem so that it follows our framework.

Theorem 5.2.12. *Given an $\mathbb{F}_{p^k}$-basis $\{b_1, b_2, \ldots, b_d\}$ for $A^{\psi_{p^k}}$, the worst case encoding neutral time complexity of* DavisIdempotents *is*

$$T_{\text{DavisIdempotents}}(d, p^k) \quad = \quad \Theta\left((d^2 + dp^k)(t(*_A) + t(+_A) + t(*_{\mathbb{K},A})) + dp^k t(+_{\mathbb{K}}) + d^2 t(*_{\mathbb{K}})\right).$$

Proof. See Theorem 5.1 in Davis [24]. We note that D in Theorem 5.1 is easily bounded by d^2, which we have used above. In addition, we have assumed subtractions cost the same as additions, scalar subtractions cost the same as algebra additions, and field divisions cost the same as field multiplications. □

Theorem 5.2.12 implies that the size of the field $\mathbb{F}_{p^k}$ plays a potentially dominating role in the performance of the algorithm. At first glance, always choosing $k = 1$ appears to be the best choice. However, the variations for finding common eigenvectors makes the choice of $k = 1$ not necessarily

DavisIdempotents(A, q) Construct primitive central idempotents in A.

INPUT: An $\mathbb{F}_q$-algebra $A = A^{\psi_q}$, and field size q
OUTPUT: A list of primitive central idempotents of A.

```
1   b ← A.basis
2   d ← |b|
3   E ← an empty queue
4   E.enqueue(1_A)
5   j ← 1
6   k ← F_q.elements   ▷ k[1] = 0_{F_q}
7   while E.size < d
8         do F ← an empty queue
9            while not E.empty
10                 do e ← E.dequeue
11                    while e ≠ 0_A
12                          do x ← e   ▷ u_0
13                             y ← b[j] * x   ▷ u_1
14                             if y is a scalar multiple of x
15                               then f ← e
16                               else i ← 2
17                                    do
18                                       ▷ Find last nonzero element in {u_n} sequence
19                                       ▷ Defined by u_n = (b_j − k_n)u_{n−1}, 0 < n < q
20                                       x ← y   ▷ u_{n−1}
21                                       y ← (b[j] − k[i] * 1_A) * x   ▷ u_n
22                                       i ← i − 1
23                                    while y ≠ 0_A
24                                    v ← x
25                                    Find α ∈ F_q such that v^2 = α * v
26                                    f ← α^{−1} * v
27                             F.enqueue(f)
28                             e ← e − f
29           j ← j + 1
30           E ← F
31  return E
```

Figure 5.5: Pseudocode for the idempotent algorithm of Davis.

the best. Although experimental data comparing different choices for k would be valuable, we leave this for future work due in part to current limitations on finite field sizes in GAP.

5.2.4 Eberly and Giesbrecht

The final algorithm for computing primitive central idempotents we analyze is the algorithm of Eberly and Giesbrecht [35]. In fact, two algorithms are described: one for fields of sufficiently large cardinality (relative to the dimension of A) and one for fields of small cardinality. The descriptions and results contained in Eberly and Giesbrecht [35] assume algebras using the matrix algebra encoding. We give encoding neutral descriptions and analyses here and discuss difficulties in obtaining their theoretical complexities in encoding neutral environments. We begin with the algorithm for large fields.

Large Fields

From the algorithm of Friedl and Rónyai (see Section 5.2.1), recall the key idea of their algorithm was to factor the minimal polynomial of each basis element obtaining a refinement of ideals or determining that an ideal is a field. Eberly [33, Sec. 3] notes that a complete decomposition can be found from a single element if $\mathbb{K}$ is sufficiently large. Let $b = \{b_1, b_2, \ldots, b_d\}$ be a basis for A. An element $a \in A$ is a *splitting element* of A if $m_a(x)$ (the minimal polynomial of a) over $\mathbb{K}$ is squarefree and has degree d. The following theorem states that splitting elements may not exist for small fields, but do exist fairly frequently for large enough fields.

Theorem 5.2.13. *Let $c > 0$ and let H be a finite subset of $\mathbb{K}$ of cardinality $|H| = \lceil cd(d-1) \rceil$. Then one of the following occurs:*

1. *A does not contain a splitting element.*
2. *If $(\alpha_1, \alpha_2, \ldots, \alpha_d) \in H^d$ is uniformly chosen from H^d, then $a = \alpha_1 b_1 + \alpha_2 b_2 + \cdots + \alpha_d b_d$ is a splitting element of A with probability at least $1 - (1/c)$.*

Proof. See Lemma 2.1 in Eberly [33]. □

In most cases, we require the field size to be at least $2d^2$, giving at least a $1/2$ probability of finding a splitting element randomly. We now assume that the field is sufficiently large and a is a splitting element of A. Recall that A is a commutative semisimple algebra. By the Wedderburn and

Wedderburn-Artin structure theorems, $A \cong E_1 \oplus E_2 \oplus \cdots \oplus E_n$ for simple algebras E_i. Furthermore, each E_i is commutative because A is commutative. Thus, each E_i is a field; in particular, each E_i is a field extension of $\mathbb{K}$.

Let $\phi : A \to E_1 \oplus E_2 \oplus \cdots \oplus E_n$ be an algebra isomorphism and let $\phi(a) = (a_1, a_2, \ldots, a_n)$, where $a_i \in E_i$. The minimal polynomial a and the minimal polynomial of each a_i are related by

$$m_a(x) \quad = \quad \operatorname{lcm}\left(m_{a_1}(x), m_{a_2}(x), \ldots, m_{a_n}(x)\right) \tag{5.1}$$

(see Friedl and Rónyai [39] Proposition 6.4 for example). Let d_i be the $\mathbb{K}$-dimension of E_i. Then, $d = d_1 + d_2 + \cdots + d_n$ by the isomorphism. Furthermore, we know $m_{a_i}(x)$ has degree at most d_i. Because a has a squarefree minimal polynomial, we obtain $m_a(x) = \prod_{i=1}^{n} m_{a_i}(x)$ by Equation 5.1. The minimal polynomials $m_{a_i}(x)$ are pairwise relatively prime, so, by the Chinese Remainder Theorem, there exist polynomials $g_1, g_2, \ldots, g_n \in R[x]$, each with degree $< d$, such that

$$\begin{aligned} g_i &\equiv 1 \mod m_{a_i}(x) \text{ and} \\ g_i &\equiv 0 \mod m_{a_j}(x) \text{ if } j \neq i, \end{aligned}$$

where $1 \leq i, j \leq n$. Each element $g_i(a)$ is a primitive central idempotent and maps to the identity element in E_i.

A randomized algorithm is readily obtained from the preceding description. Pseudocode for EberlyGiesbrechtLarge, an encoding neutral version of the algorithm of Eberly and Giesbrecht, appears in Figure 5.6. This version of the algorithm has the following time complexity.

Theorem 5.2.14. *The expected encoding neutral time complexity of* EberlyGiesbrechtLarge *given a d-dimensional commutative semisimple algebra A is*

$$\begin{aligned} T_{\text{EberlyGiesbrechtLarge}}(d) \quad = \quad & O\Big(T_{\text{MinPoly}}(d) + T_{\text{Factor}}(d) \\ & + d^3(t(+_{\mathbb{K}}) + t(*_{\mathbb{K}})) \\ & + d^2(t(*_A) + t(+_A) + t(*_{\mathbb{K},A}))\Big). \end{aligned}$$

Proof. We begin by analyzing the **do** loop on line 5. Each random choice on line 6 results in $\Theta(d(t(+_A) + t(*_{\mathbb{K},A})))$ operations. There is one call to MinPoly on line 7. By Theorem 5.2.13, a splitting element is found with probability at least 1/2. The random choices are independent, resulting a geometric probability distribution (see Cormen, et al. [19, Sec. 6.4]). The expected

EberlyGiesbrechtLarge(A) Find primitive central idempotents in A (over large fields).

INPUT: A d-dimensional commutative semisimple $\mathbb{K}$-algebra Asuch that $|\mathbb{K}| \geq 2d^2$.
OUTPUT: A list of then primitive central idempotents in A.

```
1   H ← a finite subset of 𝕂 such that |H| ≥ 2d^2
2   b ← A.basis
3   d ← |b|
4   id ← []
5   do
6       a ← Σ_{i=1}^{d} α_i * b[i] where α_i ∈ H chosen uniformly.
7       f ← MinPoly(𝕂, a)
8   while deg f < d
9   factors ← Factor(f)
10  l ← |factors|
11  for i ← 1 to l
12      do y ← 1 × l array of 0_𝕂
13         y[i] ← 1_𝕂
14         ▷ ChineseRemainder applies the Chinese remainder theorem
15         ▷ to find a function g such that g ≡ y[i]  mod facts[i]
16         g ← ChineseRemainder(facts, y)
17         id[i] ← g(a)
18  return id
```

Figure 5.6: Pseudocode for the Eberly-Giesbrecht algorithm (large fields).

number of loop executions is 2, and the expected contribution of the loop is $O(d(t(+_A) + t(*_{\mathbb{K},A})) + T_{\text{MinPoly}}(d))$.

On line 9, the minimal polynomial is factored at a cost of $T_{\text{Factor}}(d)$. In the worst case, there are d factors, hence the **for** loop on line 11 is executed at most d times. Each call to ChineseRemainder on line 16 has a cost of $\Theta(d^2(t(+_{\mathbb{K}}) + t(*_{\mathbb{K}})))$ (see Bach and Shallit [8, Sec. 6.6]). Each polynomial evaluation on line 17 costs $\Theta(d(t(*_A) + t(+_A) + t(*_{\mathbb{K},A})))$. The total contribution is $\Theta(d^3(t(+_{\mathbb{K}}) + t(*_{\mathbb{K}})) + d^2(t(*_A) + t(+_A) + t(*_{\mathbb{K},A})))$.

The stated time complexity follows immediately. □

Eberly and Giesbrecht [35] present a version of EberlyGiesbrechtLarge that works explicitly with matrix algebras. They achieve an expected time complexity of $O(d^3)$ operations in $\mathbb{K}$ using several results of Giesbrecht [45] for fast randomized algorithms to compute normal forms of matrices. In particular, Giesbrecht gives algorithms for finding the Frobenius and rational Jordan forms of a matrix as well as an efficient algorithm for evaluating polynomials on a matrix. Finding the rational Jordan form allows one to compute a factorization of a minimal polynomial directly.

An encoding neutral algorithm is less likely to benefit from the algorithms of Giesbrecht unless each algebra is converted to its regular matrix representation. In practice, the matrix encoding is too costly to store as the dimension of the algebra grows even moderately (i.e., when d is approximately 200). The additional computational overhead of temporarily converting an element to a matrix also defeats the utility of potential performance increases from Giesbrecht's work. Finding efficient ways to take advantage of Giesbrecht's results in an encoding neutral fashion is left for future work.

The algorithm EberlyGiesbrechtLarge can be applied to an algebra over a small field as well. If E is an extension of $\mathbb{K}$ such that E has $\geq 2d^2$ elements, the idempotents in A can be found by finding idempotents in $A \otimes_{\mathbb{K}} E$. Details for this construction appear in Curtis and Reiner [21, Sec. 69] and Eberly [33, Sec. 3]. Working with $A \otimes_{\mathbb{K}} E$ may prove costly in practice, although no known implementation using this approach exists. We leave experimentation with $A \otimes_{\mathbb{K}} E$ as future work and turn to the algorithm of Eberly and Giesbrecht that works over small fields.

Small Fields

In this section, we review the algorithm of Eberly and Giesbrecht for finding primitive central idempotents in a $\mathbb{K}$-algebra A where $\mathbb{K}$ is small relative to the dimension of A. Initially, the restriction that A be commutative and semisimple is lifted. Much like the algorithm for large fields, we look for

elements that allow use to find several idempotents at once. A *decomposable element* $a \in A$ has a minimal polynomial $m_a(x)$ with two or more monic, pairwise relatively prime factors $f_1, f_2, \ldots, f_l$. As in the large field case, we compute (using the Chinese remainder theorem) $g_i \in \mathbb{K}[x]$, for each $1 \leq i \leq l$, where $g_i \equiv 1 \mod f_i$ and $g_i \equiv 0 \mod f_j$ when $i \neq j$. We see again that each $e_i = g_i(a)$ is an idempotent with $\sum_{i=1}^{l} e_i = 1_A$, and the e_i are pairwise orthogonal. These idempotents are not necessarily central or primitive however. Eberly and Giesbrecht [35, Thm. 3.2] show that a, an element chosen uniformly at random from A, is a decomposable element with probability at least $1/22$

In order to find primitive idempotents, the process of finding decomposable idempotents is iterated. Suppose $e_1, e_2, \ldots, e_l$ are the pairwise orthogonal idempotents found in the preceding paragraph. The *Pierce decomposition* of A with respect to $e_1, e_2, \ldots, e_l$ is $A = \oplus_{i=1}^{l} \oplus_{j=1}^{l} e_i A e_j$ (see Curtis and Reiner [22, pg. 140]). The search for idempotents is restricted to the subalgebra $\oplus_{i=1}^{l} e_i A e_i$. If e_i is primitive, then e_i is the only idempotent in $e_i A e_i$ (see Lemma 54.8 and Theorem 54.9 in Curtis and Reiner [21]). If e_i is not primitive, then $e_i = e_{i1} + e_{i2}$, the sum of two orthogonal idempotents. The idempotents e_{i1} and e_{i2} are in $e_i A e_i$, because $e_i e_{i1} e_i = e_{i1}$ and likewise for e_{i2}. Eberly and Giesbrecht [35, Sec. 3.2] show that selecting a random element a in A leads to random elements in $\oplus_{i=1}^{l} e_i A e_i$, by mapping a into $\oplus_{i=1}^{l} e_i A e_i$. If a_i is the image of a in $e_i A e_i$, then a_i is used to construct idempotents as before.

The process of constructing primitive idempotents is left as a Monte Carlo randomized algorithm by Eberly and Giesbrecht. They elaborate on how to construct central primitive idempotents using the techniques above when A is semisimple, as well as providing a test to verify that central primitive idempotents were in fact found. Their resulting algorithm is a Las Vegas randomized algorithm. We do not include the details here because the remainder of their algorithm requires A to be given as a matrix algebra. We give an encoding neutral method for using their techniques instead.

We note that if A is required to be commutative, then any idempotents found are central idempotents. Thus, if A is noncommutative, we can work in $Z(A)$ as with the other algorithms. We also note that if A is commutative, then $e_i A e_i = e_i A = A e_i A$, so $e_i A e_i$ is in fact an ideal of A. There are two methods we can use to check the primitivity of e_i. The first is to work in A^{ψ_q} as we did in earlier sections. In this case, if e_i is primitive then $e_i A$ is a copy of $\mathbb{K}$ (and has dimension one). Alternatively, $(e_i A)^{\psi_q}$ is a copy of $\mathbb{K}$ if e_i is primitive. The second method is used in GAP. In Figure 5.7, pseudocode for EberlyGiesbrechtSmall appears, which uses the first method to test

the primitivity of idempotents.

Theorem 5.2.15. *The expected encoding neutral time complexity for* EberlyGiesbrechtSmall *when given a d-dimensional commutative algebra* $A = A^{\psi_q}$ *is*

$$\begin{aligned} T_{\text{EberlyGiesbrechtSmall}}(d) &= O\Big(d(T_{\text{MinPoly}}(d) + T_{\text{Factor}}(d,q)) \\ &\quad + d^2(t(*_A) + t(+_A) + t(*_{\mathbb{K},A})) \\ &\quad + d^3(t(*_{\mathbb{K}}) + t(+_{\mathbb{K}}))\Big). \end{aligned}$$

Proof. In the worst case, an element is always selected such that the minimal polynomial has two irreducible factors. We know there are d primitive idempotents to find, implying that the condition of the **if** statement on line 14 is true $d-1$ times given our worst case assumption.

The probability that a randomly chosen element is a decomposable element is at least $1/22$. Thus, we expect $22 = O(1)$ iterations of the **while** loop on line 5 before a reduction occurs. There are an expected number of $O(d)$ random elements generated, each at a cost of $\Theta(d(t(\text{Random}_{\mathbb{K}}) + t(*_{\mathbb{K},A}) + t(+_A)))$. The total contribution is $O(d^2(t(\text{Random}_{\mathbb{K}}) + t(*_{\mathbb{K},A}) + t(+_A)))$.

We assume that $T_{\text{MinPoly}}(d)$ is the dominant cost in each iteration of the **while** loop on line 5. This is reasonable to assume by the previous argument given in the proof of Theorem 5.2.4. As a result of this assumption, we further assume that the idempotents constructed on lines 15–26 generate a one dimensional ideal and its complement.

By our assumptions, we see that calls to MinPoly are made on elements in an algebras of dimension $d, d-1, d-2, \ldots, 2$. throughout the execution of the algorithm. The calls at each dimension are executed an expected number of $O(1)$ times. We roughly bound the total complexity of the calls to MinPoly as $O(dT_{\text{MinPoly}}(d))$.

The same argument may be used to bound the total complexity of the calls to Factor at $O(dT_{\text{Factor}}(d,q))$.

By our assumption that the minimal polynomial always has two factors, there are precisely two calls to ChineseRemainder for each reducible element found. The degree of the minimal polynomial in this case is no larger than the degree of the algebra, so $T_{\text{ChineseRemainder}}(d)$ has parameters ranging from d to 2. Again, we bound this complexity crudely at $O(d^3(t(*_{\mathbb{K}} + +_{\mathbb{K}})))$.

We get an expected contribution of $O(dt(*_A))$ from the multiplication on line 9. We also get a contribution of $O(d^2(t(*_A) + t(*_{\mathbb{K},A}) + t(+_A)))$ from the function evaluation on line 22.

Totalling the contributions, we obtain the stated complexity. □

EberlyGiesbrechtSmall(A) Return primitive central idempotents of A.

INPUT: A commutative $\mathbb{F}_q$-algebra $A = A^{\psi_q}$.
OUTPUT: A list of primitive central idempotents.

```
1   b ← A.basis
2   d ← |b|
3   id ← []
4   E ← [1_A]
5   while |id| < d
6         do a ← randomly chosen element from A
7            F ← []
8            for each e ∈ E
9                 do x ← e * a
10                   f ← MinPoly(F_q, x)
11                   facts ← Factor(f)
12                   Combine common factors in facts
13                   l ← |facts|
14                   if l > 1
15                     then for i ← 1 to l
16                                do y ← 1 × l array of 0_K
17                                   y[i] ← 1_K
18                                   ▷ ChineseRemainder applies the Chinese
19                                   ▷ remainder theorem to find a function g such that
20                                   ▷ g ≡ y[i]  mod facts[i]
21                                   g ← ChineseRemainder(facts, y)
22                                   z ← g(x)
23                                   I ← Ideal(A, [z])
24                                   if I.dimension = 1
25                                     then Add z to list id
26                                     else Add z to list F
27                     else Add e to list F
28           E ← F
29  return id
```

Figure 5.7: Pseudocode for the algorithm of Eberly and Giesbrecht (over small fields).

We note that in practice, the worst case conditions rarely appear. Also, we believe the worst case analysis can be improved given the structure of A^{ψ_q}, but this is left for future work.

5.2.5 Other Algorithms

In this section, we briefly review algorithms related to algebra decomposition. Eberly [34] discusses algorithms for decomposing algebras over $\mathbb{R}$ and $\mathbb{C}$. The algebras are required to have a $\mathbb{K}$-basis, where $\mathbb{K}$ is a number field. This restriction allows for polynomial time computations (see Loos [66] and Landau [65]). The Eberly and Giesbrecht algorithm for large fields can then be applied to find primitive central idempotents.

Ivanyos, Rónyai, and Szántó [61] over finite algebraic extensions of the rational function field $\mathbb{F}_q(X_1, \ldots, X_m)$. The techniques in the paper apply the lifting ideas from Gianni, et al. [44] to achieve polynomial time bounds.

Also of interest is a randomized algorithm of Eberly and Giesbrecht [35, Sec. 2.2] for computing $Z(A)$ over sufficiently large fields. We have not implemented this algorithm to compare it to the well known determinstic algorithms and leave its study for future work.

Rónyai [87] gives a determinstic method for computing splitting elements in simple $\mathbb{Q}$-algebras. The method is applied to several previously known Las Vegas algorithms to obtain determinstic algorithms. Recent work by de Graaf and Ivanyos [25] has led to deterministic polynomial time algorithms to compute splitting elements in $\mathbb{F}_q$-algebras, when $\mathbb{F}_q$ is sufficiently large (as defined earlier).

5.3 Explicit Isomorphisms for Simple Algebras

By the Wedderburn-Artin structure theorem, a simple $\mathbb{K}$-algebra A is isomorphic to a full matrix algebra $M_n(D)$ over a division ring D. Constructing an isomorphism between A and $M_n(D)$ allows one to use the well known structure of $M_n(D)$ to compute efficiently in A. For example, finding a (module) decomposition of A into minimal right ideals $L_1, L_2, \ldots, L_n$,

$$A \quad = \quad L_1 \oplus L_2 \oplus \cdots \oplus L_n$$

is equivalent to finding the preimages of e_{ii}, the matrix having a 1_D in entry (i,i) and 0_D elsewhere. This is by the well known decomposition of $M_n(D)$ into minimal right ideals,

$$M_n(D) \quad = \quad e_{11}M_n(D) \oplus e_{22}M_n(D) \oplus \cdots \oplus e_{nn}M_n(D).$$

Given an explicit isomorphism, the problem is readily solved.

In this section, we review algorithms of Rónyai [84, 86] and Eberly [33] for finding explicit isomorphisms between A, and the corresponding full matrix algebra $M_n(D)$. In addition, we apply Eberly's algorithm for finding idempotents in algebras over small fields (see Section 5.2.4) to finding explicit isomorphisms for simple $\mathbb{F}_q$-algebras. In the case of a simple $\mathbb{Q}$-algebra A, we review a result of Rónyai [84, 85] stating that finding an isomorphism between A and $M_n(D)$ is at least as hard as factoring squarefree integers, assuming the Generalized Riemann Hypothesis (see Bach and Shallit [8] Section 6.4).

5.3.1 Rónyai

In this subsection, we assume A is a d-dimensional simple $\mathbb{F}_q$-algebra. The division ring D and degree n of the matrix algebra $M_n(D)$ isomorphic to A is readily found as a consequence of the following theorem of Wedderburn.

Theorem 5.3.1. *Every finite division ring is commutative.*

Proof. See Theorem 3.18 in Farb and Dennis [36]. □

Since A is a finite algebra, D must be finite, and hence a finite field. If k is the dimension of $Z(A)$, then $D = \mathbb{F}_{q^k}$, and $n^2 = d/k$.

Rónyai [84, 86], describes the following method to find an explicit isomorphism. First, find a primitive idempotent $e \in A$. Generally, e will not be central unless A is commutative. It follows that eAe is commutative (and a copy of $\mathbb{F}_{q^k}$). The right ideal eA is minimal with $\dim_{\mathbb{F}_{q^k}} eA = n$. The A acts on eA nontrivially, and the action is readily mapped to matrices (using the $\mathbb{F}_{q^k}$-representation of A, see Section 7.1).

To find a primitive idempotent $e \in A$, Rónyai shows that it is sufficient to find a zero divisor $x \in A$, which must exist if A is not commutative. An idempotent f is found as the left identity in the ideal xA. The algebra fAf has dimension smaller than A, and if fAf is not commutative, we find a zero divisor in fAf and repeat the process until fAf is commutative. Rónyai gives a complicated,

RandomizedZeroDiv(A) Find a zero divisor in simple algebra A.

INPUT: A simple $\mathbb{F}_q$-algebra A
OUTPUT: A list of two zero divisors or empty list of A is a field.

```
1   if IsCommutative(A)
2     then return []
3   do
4       a ← randomly chosen element from A
5       f ← MinPoly(F_q, a)
6       facts ← Factor(f)
7       l ← |facts|
8   while l < 2
9   g ← facts[1]
10  h ← ∏_{i=2}^{l} facts[i]
11  return [g(a), h(a)]
```

Figure 5.8: Pseudocode randomized algorithm for finding zero divisors in a simple $\mathbb{F}_q$-algebra.

deterministic algorithm for finding zero divisors in A. In light of Eberly and Giesbrecht's algorithm in Section 5.2.4, randomly choosing an element $a \in A$ until $m_a(x)$ factors nontrivially is a much easier approach to finding zero divisors. If $m_a(x) = gh$ is a nontrivial factorization where $g, h \in \mathbb{F}_q[x]$, then clearly $g(a)$ and $h(a)$ are zero divisors. This essentially iterates Step 4 in Rónyai's ZERODIV algorithm until successful. Figure 5.8 contains pseudocode for RandomizedZeroDiv, our randomized algorithm for finding zero divisors.

Theorem 5.3.2. *The expected encoding neutral time complexity for* RandomizedZeroDiv *given a d-dimensional algebra A as input is*

$$\begin{aligned} T_{\text{RandomizedZeroDiv}}(d) &= O\Big(T_{\text{IsCommutative}}(d) + T_{\text{MinPoly}}(d) + T_{\text{Factor}}(d) \\ &\quad + d(t(*_{\mathbb{F}_q[x]}) + t(*_A) + t(+_A))\Big) \end{aligned}$$

Proof. There is one call to IsCommutative to determine if A is a field. By Theorem 3.2 in Eberly and Giesbrecht [35], the probability of finding an element $a \in A$ such that $m_a(x)$ the minimal polynomial of a has nontrivial factorization is at least $1/22$. Thus, we expect to find such an a after 22 iterations of the **do** loop on line 3. We obtain the contribution of $O(T_{\text{MinPoly}}(d) + T_{\text{Factor}}(d))$ from the **do** loop.

After a satisfactory a is found, the product of no more than $l-1 < d$ polynomials is performed, for a contribution of $O(dt(*_{\mathbb{F}_q[x]}))$ Evaluating the polynomials g and h on a costs at most $O(d(t(*_A)+ t(+_A)))$.

Totalling the contributions results in the stated complexity. □

Although Rónyai's procedure works with matrix algebras, it is difficult to create an encoding neutral implementation. In the next subsection, we review the algorithm of Eberly [33], which is more explicit in its description and leads easily to an encoding neutral algorithm.

5.3.2 Eberly

We review the basic procedure used by Eberly [33] to find an isomorphism between A and $M_n(D)$ without proof. The reader is referred to Eberly's paper for more details. Eberly's algorithm finds elements e_{ij} for $1 \leq i \leq n$ which are mapped to matrices in D with the (i,j) entry being 1_D and all other entries 0_D. In addition, a basis for D is found. Eberly's algorithm is applicable to any finite dimensional algebra A, assuming we can find a primitive idempotent $e \in A$. For now, assume e is a primitive idempotent in A.

We first find primitive idempotents e_{ii} such that $1_A = e_{11} + e_{22} + \cdots + e_{nn}$. Let r be the $\mathbb{F}_q$ dimension of eA. Let $f \in A$ be an idempotent that is not primitive and let $B = \{b_1, b_2, \ldots, b_d\}$ be a basis for A. The right module fA is generated by elements of the form $l_i = fb_ie$ for $1 \leq i \leq d$ (because $A = AeA$). Either $l_i = 0_A$ or $\dim l_iA = r$. If $\dim l_iA = r$, then find a left identity g of l_iA. The elements gf and $f - gf$ are idempotents. In particular, gf is a primitive idempotent, and we repeat the process with $f - gf$ for further refinement. Starting with $f = 1_A$ and iterating $n-1$ times is sufficient to find $1_A = e_{11} + e_{22} + \cdots + e_{nn}$.

Given $e_{11} + e_{22} + \cdots + e_{nn}$, we find the remaining e_{ij} as follows. Set e_{1i} to be any non-zero element in $e_{11}Ae_{ii}$. Then, there exists a unique element $e_{i1} \in e_{ii}Ae_{11}$ such that $e_{1i}e_{i1} = e_{11}$ and $e_{i1}e_{1i} = e_{ii}$. The element e_{i1} is found by solving the system of linear equations $e_{1i}y = e_{11}$ for y where $y \in e_{ii}Ae_{11}$. The remaining elements are set to $e_{ij} = e_{i1}e_{1j}$.

A basis for D is found by solving the set of linear equations $ye_{ij} = e_{ij}y$ for y, where $y \in A$ and $1 \leq i, j \leq d$.

Figure 5.9 contains psuedocode for ExplicitIsomorphism, Eberly's algorithm for computing an isomorphism between a simple algebra A and $M_n(D)$. The analysis of ExplicitIsomorphism

is left for future work.

We have shown how to find e when A is an $\mathbb{F}_q$-algebra in the Section 5.3.1, so Eberly's algorithm applies in this situation. In the next subsection, we review a result of Rónyai that shows finding idempotents in a $\mathbb{Q}$-algebra is more difficult.

5.3.3 Rationals

The algorithm of Eberly [33] (in the previous subsection) for computing an explicit isomorphism between a d-dimensional simple $\mathbb{K}$-algebra A and $M_n(D)$ only assumes that a primitive idempotent in A has been found. When $\mathbb{K} = \mathbb{F}_q$, a finite field, such a primitive idempotent is efficiently computable (see Section 5.3.1). In this section, we review results of Rónyai [84, 85] that relate the problem of finding a primitive idempotent in a $\mathbb{Q}$-algebra to the problem of factoring squarefree integers.

Let A be a four-dimensional simple $\mathbb{Q}$-algebra such that $Z(A) = \mathbb{Q}$. Rónyai considers the quesion, "Does A have zero divisors?" If A does have zero divisors, then $A \cong M_2(\mathbb{Q})$. Otherwise, A is a division ring. If a primitive idempotent $e \in A$ is found such that $e \neq 1_A$, then e and $1_A - e$ are zero divisors and the question is answered. So, the problem of finding zero divisors in A is reduced to finding a primitive idempotent in A.

Rónyai shows that "Does A have zero divisors?" is polynomial time equivalent to finding integers x, y, and z not all zero such that

$$ax^2 + by^2 + cz^2 \quad = \quad 0,$$

where a, b, and c are non-zero integers. This problem is shown to be in the complexity class $NP \cap co - NP$ (see Section 3 of Rónyai [84]).

Finally, in Section 4 of Rónyai [84], finding zero divisors in A is shown to be (Las Vegas) polynomial time equivalent to the problem of factoring squarefree integers, assuming the Generalized Riemann Hypothesis (see Bach and Shallit [8] Section 6.4).

Factoring squarefree integers is currently considered to be difficult problem, and Rónyai has shown that finding zero divisors is of equivalent difficulty. In this case, we have no efficient algorithm available for solving **SEMISIMPLE ALGEBRA DECOMPOSITION** for $\mathbb{Q}$-algebras, because there is no known efficient algorithm for finding an explicit isomorphism for simple $\mathbb{Q}$-algebras.

ExplicitIsomorphism(A, e) Compute an isomorphism between A and $M_n(D)$.

INPUT: A simple d-dimensional $\mathbb{K}$-algebra A and a primitive idempotent $e \in A$
OUTPUT: A $\mathbb{K}$-basis $D_1, D_2, \ldots, D_h$ for D and elements $E_{ij} \in A$ to map to a standard basis for $M_n(D)$.

```
1  b ← A.basis
2  d ← |b|
3  r ← RightIdeal(A, [e]).dimension
4  n ← d/r
5  E[1,1] ← e
6  t ← 1_A
7  for i ← 2 to n
8      do t ← t − E[i − 1, i − 1]
9         j ← 0
10        do
11            j ← j + 1
12            x ← t ∗ b[j] ∗ e
13        while x = 0_A
14        I ← RightIdeal(A, [x])
15        f ← LeftIdentity(I)
16        E[i, i] ← f ∗ t
17 for i ← 2 to n
18     do E[1, i] ← nonzero element of E[1,1]AE[i, i]
19        E[i, 1] ← nonzero element y ∈ E[i, i]AE[1, 1] such that E[1, i]y = E[1, 1]
20 for i ← 2 to n − 1
21     do for j ← i + 1 to n
22            do E[i, j] ← E[i, 1] ∗ E[1, j]
23               E[j, i] ← E[j, 1] ∗ E[1, i]
24 D ← basis for the solution space of E[i, j]z = zE[i, j], solving for z, where 1 ≤ i, j ≤ n
25 return [D, E]
```

Figure 5.9: Pseudocode for Eberly's explicit isomorphism algorithm.

5.4 Jacobson Radical

Recall that the Jacobson radical $J(A)$ of a $\mathbb{K}$-algebra A is the intersection of the maximal right ideals of A. In general, in order to determine if an algebra A is semisimple, the Jacobson radical $J(A)$ of A must be calculated. Once calculated, we readily test for $J(A) = 0$. In the case of group algebras, Maschke's theorem leads to an easy test for semisimplicity.

Theorem 5.4.1 (Maschke's Theorem). *Let $\mathbb{K}G$ be a group algebra. If the characteristic of the field $\mathbb{K}$ does not divide the order of the group G, then $\mathbb{K}G$ is semisimple.*

Proof. Many versions of Maschke's theorem appear in standard texts. The proof of this version appears as Proposition 5.8 in Hungerford [57] Chapter 9, Section 5. □

If A is determined not to be semisimple, the quotient algebra $A/J(A)$ is semisimple and we may attempt to find a solution of **SEMISIMPLE ALGEBRA DECOMPOSITION** for $A/J(A)$.

When $\mathbb{K}$ has characteristic 0, $J(Z(A))$ must be factored out of $Z(A)$ to find primitive central idempotents of A and find a solution for **ALGEBRA DECOMPOSITION**. The same may also be done for fields $\mathbb{K}$ with characteristic $p > 0$; however, we see from our experimental data in Section 5.6 and from our analysis that calculating A^{ψ_q} (see Section 5.2.2) is more efficient in practice.

In the remainder of this section, we review algorithms for calculating $J(A)$. For $\mathbb{K}$ with characteristic zero, we review a straightforward linear algebra solution obtained from Dickson's criterion, which was first noted Friedl and Rónyai [39] in an algorithmic context, in Section 5.4.1. For $\mathbb{K}$ with characteristic $p > 0$, we review the algorithms of Friedl and Rónyai [39] and Cohen, Ivanyos, and Wales [17] in Section 5.4.2.

5.4.1 Characteristic Zero

In this subsection, A is a finite dimensional $\mathbb{K}$-algebra where $\mathbb{K}$ has characteristic zero. Calculating the radical in this situation is straightforward. The *trace* $\mathrm{Tr}\,(a)$ of an element $a \in A$ is the trace of $\Phi(a)$, the regular matrix representation (see Section 3.2.3) of a. Dickson [27] states the following theorem.

Theorem 5.4.2 (Dickson's Criteria). *Let A be a finite dimensional $\mathbb{K}$-algebra such that $\mathbb{K}$ has characteristic zero. Then $x \in J(A)$ if and only if $\mathrm{Tr}\,(xy) = 0_{\mathbb{K}}$ for every $y \in A$.*

RadicalZero(A) Compute $J(A)$

INPUT: An algebra A over a field of characteristic 0
OUTPUT: $J(A)$

```
1   b ← A.basis
2   d ← |b|
3   ▷ Create the set of linear equations
4   for i ← 1 to d
5       do for j ← 1 to d
6              do x ← RegularMatrixRep(b, b[i] * b[j])
7                 X[i, j] ← Trace(x)
8   V ← Nullspace(X)
9   α ← V.basis
10  n ← |z|
11  ▷ Construct a basis for J(A)
12  for i ← 1 to n
13      do z[i] ← Σ_{j=1}^{d} α[i, j] * b[j]
14  return Ideal(A, z)
```

Figure 5.10: Pseudocode for calculating $J(A)$. (characteristic zero)

Friedl and Rónyai [39] appear to be the first to apply this theorem in an algorithmic context. Because the trace function is a linear function, it is sufficient to show $\mathrm{Tr}\,(xb_i) = 0_{\mathbb{K}}$ for each $1 \leq i \leq d$ where $\{b_1, b_2, \ldots, b_d\}$ forms a basis of A. A basis for $J(A)$ is computed by solving a system of linear equations for $\mathrm{Tr}\,(xb_i) = 0_{\mathbb{K}}$. Pseudocode for RadicalZero appears in Figure 5.10.

Example 5.4.3. Let A be the $\mathbb{Q}$-algebra of 2×2 upper diagonal matrices. We use the basis $B = \{b_1, b_2, b_3\}$, where

$$b_1 = \begin{bmatrix} 1 & 1 \\ 0 & 0 \end{bmatrix}, \qquad b_2 = \begin{bmatrix} 0 & 1 \\ 0 & 0 \end{bmatrix}, \qquad b_3 = \begin{bmatrix} 0 & 0 \\ 0 & 1 \end{bmatrix}.$$

Assume A is given by its regular matrix representation $\Phi(A)$ with respect to B. The representation of each basis element is

$$\Phi(b_1) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 1 \\ 0 & 0 & 0 \end{bmatrix}, \qquad \Phi(b_2) = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & 0 & 0 \end{bmatrix}, \qquad \Phi(b_3) = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix}.$$

To find x such that $\mathrm{Tr}\,(xb_i) = 0_{\mathbb{Q}}$, we write $x = \sum_{i=1}^{d} \alpha_i b_i$ and solve a system of equations for α_i. The system of equations is

$$\begin{bmatrix} 2 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \alpha_1 \\ \alpha_2 \\ \alpha_3 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix}.$$

A basis for the solution space is $\left\{ \begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix} \right\}$, which implies $\{b_2\}$ is a basis for $J(A)$.

Theorem 5.4.4. *The worst case encoding neutral time complexity for* RadicalZero *when executed on a d-dimensional algebra A is*

$$\begin{aligned} T_{\text{RadicalZero}}(d) &= \Theta\Big(d^2(t(*_A) + T_{\text{RegularMatrixRep}}(d) + t(*_{\mathbb{K},A}) + t(+_A)) \\ &\quad + d^3(t(*_{\mathbb{K}}) + t(+_{\mathbb{K}}))\Big). \end{aligned}$$

Proof. On line 6, there is one algebra multiplication and one call to RegularMatrixRep. These are executed d^2 times for a contribution of $\Theta(d^2(t(*_A) + T_{\text{RegularMatrixRep}}(d)))$.

The function Trace on line 7 sums the diagonal entries of a $d \times d$ matrix, for a cost of $\Theta(dt(+_{\mathbb{K}}))$. The line is executed d^2 times for a total cost of $\Theta(d^3 t(+_{\mathbb{K}}))$.

Finding the nullspace on line 8 is accomplished using Gaussian elimination of a $d \times d$ matrix at a cost of $\Theta(d^3(t(*_{\mathbb{K}}) + t(+_{\mathbb{K}})))$.

The dimension of the nullspace can be at most $d - 1 = \Theta(d)$. Creating basis elements in A on line 13 costs $\Theta(d^2(t(*_{\mathbb{K},A}) + t(+_A)))$.

The total cost follows immediately. □

From Theorem 4.4.1, $T_{\text{RegularMatrixRep}}(d) = \Theta(d(t(*_A)+$ the cost to express an element in terms of a basis $))$. Thus, $T_{\text{RegularMatrixRep}}$ varies depending on the encoding used for A. If A is encoded as a matrix, the cost associated with $T_{\text{RegularMatrixRep}}$ can be dropped. Also, it may be more efficient to compute $\mathrm{Tr}\,(b_i)$ for each basis element b_i and then use linearity of the trace map to compute the trace for arbitrary elements.

5.4.2 Positive Characteristic

In this subsection, we assume A is a $\mathbb{K}$-algebra, such that $\mathbb{K}$ has characteristic $p > 0$. We begin by showing that Dickson's criterion does not correctly determine $J(A)$.

Example 5.4.5. Let A be the 2×2 upper triangular matrices over $\mathbb{F}_2$. We use the basis B and notation from Example 5.4.3, and obtain the same set of structure constants. We obtain the system of linear equations,

$$\begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \alpha_1 \\ \alpha_2 \\ \alpha_3 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix},$$

and solve for α_i. In this case, we see that $\left\{ \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix}, \begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix} \right\}$ forms a basis for the solution space. In particular b_1 is determined to be in $J(A)$. However, $b_1^2 = b_1$ is an idempotent, so $b_1 \notin J(A)$, which shows that Dickson's criterion does not correctly determine $J(A)$.

In the remainder of this subsection, we review the algorithms of Friedl and Rónyai [39] and Cohen, Ivanyos and Wales [17] for finding $J(A)$ when $\mathbb{K}$ has characteristic $p > 0$.

Friedl and Rónyai

The algorithm of Friedl and Rónyai [39] is a generalization of Dickson's criterion for $\mathbb{K} = \mathbb{F}_p$, where p is prime; that is, $\mathbb{K}$ is a finite prime field. Their algorithm constructs a sequence of ideals $A = I_{-1}, I_0, \ldots, I_l = J(A)$, where $p^l \leq d < p^{l+1}$. Let $\hat{\Phi} : A \to M_d(\mathbb{Z})$ map $a \in A$ to an integer matrix X such that $X \mod p = \Phi(a)$. Friedl and Rónyai define $\hat{g}_i : M_d(\mathbb{Z}) \to \mathbb{Q}$ by $\hat{g}_i(x) = \mathrm{Tr}\left(\hat{\Phi}(x)^{p^i}\right)/p^i$. Then, each ideal $I_i = \{x \in I_{i-1} \mid g_i(xy) = 0_{\mathbb{K}} \text{ for all } y \in A\}$, where $0 \leq i \leq l$, and $g_i = \hat{g}_i(x) \mod p$. Each function g_i is a linear function on I_{i-1}, so each I_i is found by solving a system of linear equations. See Friedl and Rónyai [39] for details and proofs regarding the correctness of this algorithm. We give pseudocode for RadicalPrimeFR in Figure 5.11.

Example 5.4.6. Let A be the $\mathbb{F}_2$-algebra of 2×2 upper triangular matrices given its regular matrix representation as in Examples 5.4.3 and 5.4.5. The dimension of A is 3, so $l = 1$. To find I_0, find

RadicalPrimeFR(A) Compute $J(A)$.

INPUT: A d-dimensional $\mathbb{F}_p$-algebra A.
OUTPUT: $J(A)$

```
1   b ← A.basis
2   d ← |b|
3   p ← characteristic of F_p
4   r ← 1   ▷ stores powers of p
5   z ← b
6   m ← d
7   while m ≠ 0 and r ≤ d
8         do for j ← 1 to d
9               do for k ← 1 to m
10                    do X ← RegularMatrixRep(b, b[j] * z[k])
11                       X̂ ← lift X to ℤ
12                       S[j,k] ← (Tr(X̂^r)/r)  mod p
13            V ← Nullspace(S)
14            cf ← V.basis
15            m' ← |α|
16            z' ← []
17            for j ← 1 to m'
18
19               do z'[j] = Σ_{k=1}^{m} cf[k] * z[k]
20            m ← m'
21            z ← z'
22            r ← r * p
23
24  return Ideal(A, z)
```

Figure 5.11: Pseudocode for Friedl and Rónyai algorithm to calculate $J(A)$.

$x \in A$ such that $g_0(xy) = \text{Tr}\left(\hat{\Phi}(xy)\right) \mod 2 = 0$ for all $y \in A$. The system of linear equations to solve is

$$\begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \alpha_1 \\ \alpha_2 \\ \alpha_3 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix}.$$

A basis for I_0 is then $B_0 = \{b_1, b_2\}$.

To find I_1, find $x \in I_0$, such that $g_1(xy) = \text{Tr}\left(\hat{\Phi}(xy)^2\right)/2 \mod 2 = 0$ for all $y \in A$. In this step, $x = \sum_{i=1}^{2} \alpha_i b_i$. We now see that $g_1(b_1 b_1) = \text{Tr}\left(\hat{\Phi}(b_1 b_1)^2\right)/2 \mod 2 = 1$, which correctly identifies that b_1 is not in $J(A)$. The entire system of linear equations to solve is

$$\begin{bmatrix} 1 & 0 \\ 0 & 0 \\ 0 & 0 \end{bmatrix} \begin{bmatrix} \alpha_1 \\ \alpha_2 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix}.$$

A basis for the solution space is $\left\{ \begin{bmatrix} 0 \\ 1 \end{bmatrix} \right\}$, which implies a basis for $J(A)$ is $\{b_2\}$.

Theorem 5.4.7. *The worst case encoding neutral time complexity for* RadicalPrimeFR *executing on a d-dimensional $\mathbb{F}_p$ algebra A is*

$$\begin{aligned} T_{\text{RadicalPrimeFR}}(d,p) = \Theta\Big(& (d^5 \log_p^2 d \log p)(t(*_{\mathbb{Z}}) + t(+_{\mathbb{Z}})) \\ & + (d^4 \log_p d) t(\mathit{lift}_{\mathbb{K},\mathbb{Z}}) \\ & + (d^3 \log_p d)(t(*_{\mathbb{K}}) + t(+_{\mathbb{K}})) \\ & + (d^2 \log_p d)(t(*_A) + t(+_A) + t(*_{\mathbb{K},A}) + T_{\text{RegularMatrixRep}}(d)) \Big). \end{aligned}$$

Proof. In the worst case, $A = I_0 = \cdots = I_{l-1}$, which occurs when $d = p^l$ and A is the set of upper triangular $d \times d$ matrices with identical diagonal entries.

On line 10, there is one algebra multiplication and one call to RegularMatrixRep. These are executed at most $d^2 \log_p d$ times for a contribution of $\Theta((d^2 \log_p d)(t(*_A) + T_{\text{RegularMatrixRep}}(d)))$.

Line 11 lifts d^2 elements at most $d^2 \log_p d$ times for a contribution of $\Theta((d^4 \log_p d) t(\text{lift}_{\mathbb{K},\mathbb{Z}}))$.

To determine the cost of line 12, we note that the matrix multiplications required by exponentiation is $\Theta(\log r)$ using binary exponentations, where $r = 1, p, \ldots, p^l$. In the worst case, we have the same number of matrices to multiply per value of r. Thus, over all values of r, we have a total of

$\log 1 + \log p + \cdots + \log p^l = \log p^{\frac{l^2-l}{2}} = (l^2 - l)/2 \log p$ matrix multiplications for a given $(\mathtt{j}, \mathtt{k})$ pair. Each matrix multiplication costs $\Theta(d^3(t(*_{\mathbb{Z}}) + t(+_{\mathbb{Z}})))$ and there are d^2 iterations of line 12. The total cost contribution due to matrix multiplications on line 12 is $\Theta((d^5 \log_p^2 d \log p)(t(*_{\mathbb{Z}}) + t(+_{\mathbb{Z}})))$. The trace costs d integer additions, and is executed a total of d^2 times for an addition contribution of $\Theta(d^3 t(+_{\mathbb{Z}}))$. The line contributes an additional $\Theta(d^2 t(*_{\mathbb{Z}}))$, if we assume the integer division and modulus operations cost the same as integer multiplication. The total cost contribution of line 12 is $\Theta((d^5 \log_p^2 d \log p)(t(*_{\mathbb{Z}}) + t(+_{\mathbb{Z}})))$.

Calculating the nullspace on line 13 uses Gaussian elimination, costing $\Theta(d^3(t(*_{\mathbb{K}}) + t(+_{\mathbb{K}})))$. This is executed $\log_p d$ times for a total contribution of $\Theta((d^3 \log_p d)(t(*_{\mathbb{K}}) + t(+_{\mathbb{K}})))$.

Computing the d linear combinations on line 18 costs $\Theta(d^2(t(+_A) + t(*_{\mathbb{K},A})))$. This line is executed $\log_p d$ times for a total contribution of $\Theta((d^2 \log_p d)(t(+_A) + t(*_{\mathbb{K},A})))$.

The remainder of the operations are dominated by the operations we have already discussed. The stated complexity follows by totalling the contributions given. □

The cost of lifting elements from $\mathbb{F}_p$ to $\mathbb{Z}$ can generally be removed from consideration because elements in $\mathbb{F}_p$ are often encoded as integers. If space is available, computing the regular matrix representation of A beforehand improves the complexity by removing redundant calls to RegularMatrixRep. There is no benefit to precomputing the trace of basis elements as could be done in the characteristic zero situation. This is because we must exponentiate integral matrices, not the algebra elements over $\mathbb{F}_p$. In practice, computing the radical with RadicalPrimeFR dominates the calculation for **SEMISIMPLE ALGEBRA DECOMPOSITION**.

The algorithm RadicalPrimeFR can be applied to a d-dimensional $\mathbb{F}_q$-algebra A by viewing A as a dn-dimensional $\mathbb{F}_p$-algebra, where $q = p^n$. This approach tends to greatly increase the space requirements for encoding A. Eberly [32] generalizes the algorithm of Friedl and Rónyai by lifting elements in $\mathbb{F}_q = \mathbb{F}_p[x]/\langle I \rangle f$, to $\mathbb{Z}[x]/\langle I \rangle f$, where f is an irreducible $\mathbb{F}_p$-polynomial (and hence an irreducible $\mathbb{Z}$-polynomial) in the hopes that this would lead to a performance improvement. Informal experiments in GAP show that Eberly's approach does not significantly improve performance, and we have chosen not to pursue Eberly's generalization further.

Cohen, Ivanyos, Wales

The method of Friedl and Rónyai for computing $J(A)$ has the undesirable property of dimensional blowup when working over arbitrary finite fields. Cohen, Ivanyos, and Wales [17] introduced a method for computing $J(A)$ which does not involve lifting to $\mathbb{Z}$ and works in $\mathbb{F}_q$ directly. The basic structure of the algorithm is the same as Friedl and Rónyai's: constructe a sequence of ideals $(I_i)_{0\leq i\leq l}$ $A = I_0 \supseteq I_1 \supseteq \cdots \supseteq I_l = J(A)$, where $l = \log_p d$. Instead taking the trace of powered matrices lifted to $\mathbb{Z}$, the characterization,

$$c_a(x) \quad = \quad \sum_{i=0}^{d} (-1_{\mathbb{K}})^i \mathrm{Tr}\,(i,a)\, x^{d-i},$$

for the trace map $\mathrm{Tr}\,(i,a)$ is used, where $a \in A$ and $c_a(x)$ is the characteristic polynomial of the regular matrix representation of a. The ideal $I_0 = A$, and the ideals I_i for $1 \leq i \leq l$ are defined by $I_i = \left\{a \in I_{i-1} \mid \mathrm{Tr}\left(p^{i-1}, a\right) = 0_{\mathbb{K}} \text{ and } \mathrm{Tr}\left(p^{i-1}, ab\right) = 0_{\mathbb{K}} \text{ for all } b \in Ii - 1\right\}$. The characterization of the trace map is not linear, so constructing the ideals I_i requires solving equations of the form $\alpha_1 x_1^p + \alpha_2 x_2^p + \cdots + \alpha_r x_r^p = 0_{\mathbb{K}}$. In the case where $\mathbb{K} = \mathbb{F}_q$, Cohen, et al. state that this is readily accomplished, (see Cohen, et al. [17] page 191). Figure 5.12 contains encoding neutral pseudocode for RadicalPrimeCIW, the Cohen, Ivanyos, Wales algorithm specialized for $\mathbb{K} = \mathbb{F}_q$.

A precise analysis of RadicalPrimeCIW is left for future work. The main difficulty with analyzing RadicalPrimeCIW is recognizing a worst case situation. Unlike RadicalPrimeFR, RadicalPrimeCIW can benefit when $I_i = I_{i-1}$. The characteristic polynomials can be used from the previous iteration, thus the trace is readily available. We implement this feature in the GAP implementation. Also, RadicalPrimeCIW does not suffer from the dimension explosion when $\mathbb{F}_q$ is an extension of a prime field. The author's implementation of RadicalPrimeCIW is included as a standard part of GAP versions 4.1 and 4.2.

Recent Algorithms

Recently, Ivanyos [58, 59] has introduced randomized algorithms for computing a generating set of $J(A)$. His work builds on the work of Eberly and Giesbrecht [35] and de Graaf and Ivanyos [25] using properties of splitting elements and primitive idempotents. These randomized algorithms may prove very useful in practice because calculating $J(A)$, particularly when A is an $\mathbb{F}_q$-algebra, is very time consuming. Implementations of these algorithms are not available at this time, and are left for future work.

RadicalPrimeCIW(A) Compute $J(A)$ for a $\mathbb{F}_q$-algebra.

INPUT: An $\mathbb{F}_q$-algebra A.
OUTPUT: $J(A)$

```
1   p ← characteristic of F_q
2   b ← A.basis
3   d ← |b|
4   l ← 1
5   I ← b
6   while l ≤ d
7         do H ← I ∪ {1_A}
8            r ← |I|
9            s ← |J|
10           for i ← 1 to r
11               do for j ← 1 to r
12                      do a ← I[i] * H[j]
13                         M ← RegularMatrixRep(b, a)
14                         f ← CharPoly(M)
15                         t ← coefficient of x^(d−l) term in f
16                         X[i, j] ← (−1_{F_q})^l t
17           V ← Nullspace(X)
18           v ← V.basis
19           k ← |v|
20           ▷ Recall q is order of F_q
21           if 1 < l < q
22             then for i ← 1 to k
23                      do for j ← 1 to r
24                             do v[i, j] ← v[i, j]^(1/l)
25           for i ← 1 to k
26               do T[i] ← Σ_{j=1}^{r} v[i, j] * I[j]
27           I ← T
28           l ← l * p
29  return Ideal(A, I)
```

Figure 5.12: Pseudocode for Cohen, Ivanyos, and Wales algorithm to calculate $J(A)$.

5.5 Lifting Idempotents

When calculating a solution for **ALGEBRA DECOMPOSITION**, it may be necessary to factor $J(A)$ from A in order to find primitive central idempotents. In this section, we discuss relationships between idempotents in A and $A/J(A)$, and the algorithm of Gianni, et al. [44] The process of computing an idempotent in A from one in $A/J(A)$ is *lifting*.

We recall Theorem 6.7 from Curtis and Reiner [21] which summarizes the relationship between idempotents in A and idempotents in $A/J(A)$.

Theorem 5.5.1 (Lifting Idempotents). *Let $\overline{A} = A/J(A)$. Then, the following three statements hold*

1. *For every idempotent $\epsilon \in \overline{A}$ there exists an idempotent $e \in A$ such that $\overline{e} = \epsilon$.*
2. *$Ae_1 \cong Ae_2$ as left A-modules if and only if $\overline{A}\overline{e}_1 \cong \overline{A}\overline{e}_2$ for arbitrary idempotents $e_1, e_1 \in A$.*
3. *An idempotent $e \in A$ is primitive if and only if $\overline{e}$ is primitive in $\overline{A}$.*

Theorem 5.5.1 demonstrates a strong relationship between idempotents found in $A/J(A)$ and those in A. In particular, if we find a set of primitive central idempotents in $Z(A)/J(Z(A))$, then there exist primitive central idempotents in $Z(A)$ and hence in A.

The algorithm of Gianni, et al. assumes that A is commutative, which we assume for the remainder of this section. Let I be an ideal in A. For $k \geq 1$, define $A_k = A/I^k$. We summarize the results of Gianni, et al. in the following propositions. Proofs are available in their paper.

Proposition 5.5.2. *If e and f are idempotents $\mod I^2$, and $e \equiv f \mod I$, then $e \equiv f \mod I^2$.*

Proposition 5.5.3. *If e and f are idempotents $\mod I^2$ and are orthogonal $\mod I$, then they are orthogonal $\mod I^2$.*

Proposition 5.5.4. *If $e_k \in A_k$ is an idempotent and $e_k \neq 0_{A_k}$ or 1_{A_k}, then it is possible to construct a unique idempotent $e_{2k} \in A_{2k}$ such that $e_{2k} \equiv e_k \mod I^k$.*

Proposition 5.5.5. *Let $e_1, e_2, \ldots, e_n \in A_k$ be orthogonal idempotents such that $\sum i = 1^n e_i = 1_{A_k}$. If f_i is the lifting of e_i in A_{2k}, where $1 \leq i < n$, and $f_n = 1_{A_{2k}} - \sum_{i=1}^{n-1} f_i$, then $f_1, f_2, \ldots, f_n$ is a set of orthogonal idempotents in A_{2k} such that $\sum_{i=1}^{n} = 1_{A_{2k}}$.*

LIFTIDEMPOTENTS$(A, J(A), E)$ Lift the set of idempotents E from $A/J(A)$ to A.

INPUT: A commutative algebra A, the radical $J(A)$, a set E of orthogonal idempotents in $A/J(A)$.
OUTPUT: A set of orthogonal idempotents in A.

```
1   m ← J(A).dimension
2   q ← ⌈log m⌉
3   n ← |E|
4   for i ← 1 to n
5       do f[i] ← preimage representative of E[i] in A
6   for i ← 1 to q
7       do sum ← 0_A
8          for j ← 1 to n − 1
9              do e ← f[j]
10                f[j] ← e^2 + e^2 + e^2 − (e^3 + e^3)
11                sum ← sum + f[j]
12         f[n] ← 1_A − sum
13  return f
```

Figure 5.13: Pseudocode for the Gianni, et al. algorithm to lift idempotents.

Suppose that we have found a set $\{\overline{e}_1, \overline{e}_2, \ldots, \overline{e}_n\}$ primitive central idempotents in $A/J(A)$ such that $1_{A/J(A)} = \sum_{i=1}^{n} e_i$. Let $I = J(A)$. The construction of an idempotent modulo A_{2k} is a Newton approximation of e, which is an idempotent modulo A_k:

$$\begin{aligned} f &= e + (1_A - (e+e))(e^2 - e) \\ &= (e^2 + e^2 + e^2 - (e^3 + e^3)). \end{aligned}$$

Since $J(A)$ is nilpotent, $I^m = 0$ for some integer $m \leq \dim J(A)$. Thus, $A_{\lceil \log m \rceil} = A$. After $\lceil \log m \rceil$ iterations, lifting the entire set of idempotents during each iteration, we obtain idempotents $\{e_1, e_2, \ldots, e_n\}$ in A such that $1_A = \sum_{i=1}^{n} e_i$. Pseudocode for LIFTIDEMPOTENTS appears in Figure 5.13.

Theorem 5.5.6. *The worst case encoding neutral time complexity for* LIFTIDEMPOTENTS*, if* $m = \dim J(A)$ *and* n *is the number of idempotents to lift, is*

$$T_{\text{LIFTIDEMPOTENTS}}(m, n) = \Theta\left((n \log m)(t(*_A) + t(+_A)) + nt(\text{PREIMAGE}_A)\right).$$

Proof. Initially, we compute the preimage of n idempotents in the **for** loop on line 4. The body of the **for** loop on line 6 is executed $\lceil \log m \rceil$ times. The body of the **for** loop is executed $n-1$ times, and contains a constant number of algebra multiplications and additions. The time complexity is evident from the previous statements and the pseudocode for LiftIdempotents. □

5.6 Experimental Results

We have provided theoretical analyses for several algorithms related to algebra decomposition. In this section, we provide experimental data and relate the data to the analyses. For our experiments, we used GAP version 4.1 fix 7 on a 333 MHz Pentium II machine with 112MB of RAM running FreeBSD version 3.4. GAP was given a workspace size of 64MB to reduce the impact of garbage collection.

We used only group algebras in our experiments because they are easily constructed and we can control their dimension. While we have implementations for path algebras available, constructing examples with the necessary properties we needed for running experiments is difficult, as such examples do not occur naturally.

Group algebras were selected at random using GAP's "small group library" via the `SmallGroup` command. The group used during testing is shown in the data tables in Appendix A. Each experiment was executed five times, and the graphs below show the average over the five executions.

5.6.1 Power Subalgebra

We collected data for our implementation of PowerSubalgebra as well as a group algebra specific algorithm to compute A^{ψ_q}. We performed two different experiments. In the first experiment, we fixed the field $\mathbb{F}_{53}$ and varied the dimension of the group algebras. In the second experiment, we fixed a 20-dimensional group algebra and varied the field size. The group algebras and fields used are displayed in Appendix A.

In each experiment, we used the group algebra specific algorithm on the group algebra encoding, and PowerSubalgebra on the multiplication table and regular matrix representation. The costs of conversion and computing the center are not included in the data. However, the group algebra specific algorithm computes both $Z(A)$ and $A\psi_q$ together.

Figure 5.14 contains a graph summarizing the results of the first experiment. The graph displays

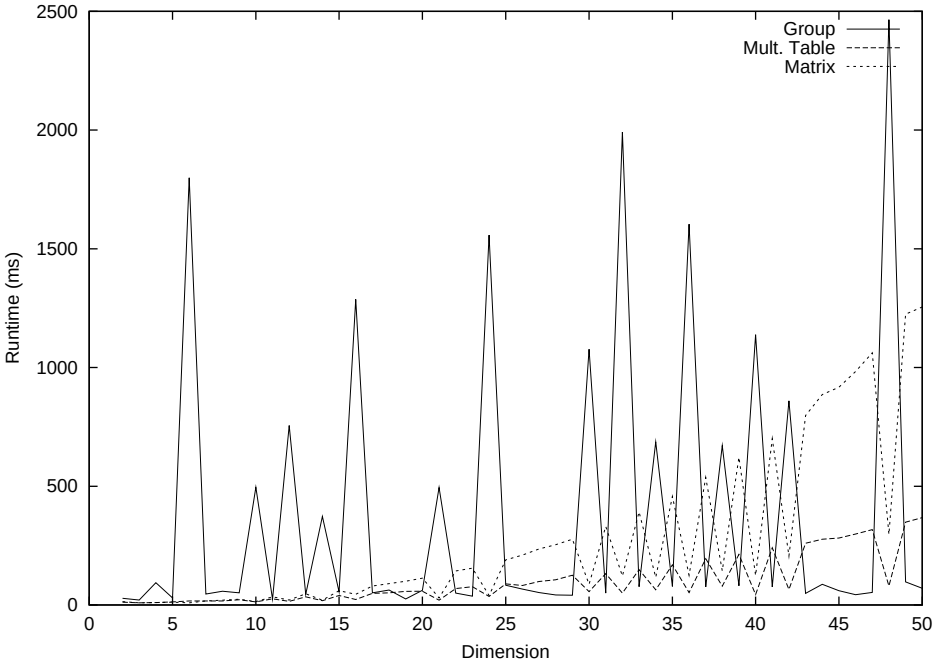


Figure 5.14: Experimental results for computing A^{ψ_q} with the field $\mathbb{F}_{53}$. We compare the group algebra specific algorithm with the standard PowerSubalgebra algorithm applied to multiplication tables and the regular matrix representation. Runtimes are in CPU milliseconds.

very interesting behavior. When the multiplication table and matrix algebra encodings perform well, the group algebra specific algorithm performs very poorly. When the group algebra specific algorithm performs well, the multiplication table and matrix algebra encodings perform poorly. Visually scanning the data, we notice that the differences generally occur when the dimension of $Z(A)$ differs from the dimension of A. This behavior makes sense when considering that the group algebra specific algorithm is computing both $Z(A)$ and A^{ψ_q} at the same time, while the standard algorithm is taking advantage of working within the smaller dimensional $Z(A)$.

The multiplication table and matrix algebra encodings show approximately the same behavior, but the multiplication table encoding generally performs better than the matrix algebra encoding. We estimate the complexity as we did in previous experiments: we hypothesize that $T(d) = cd^k$, take logarithms for both sides, and fit a line to $\ln T(d) = \ln c + k \ln d$ using the R statistical package [1]. We performed this estimation only for multiplication table and matrix algebra encodings,

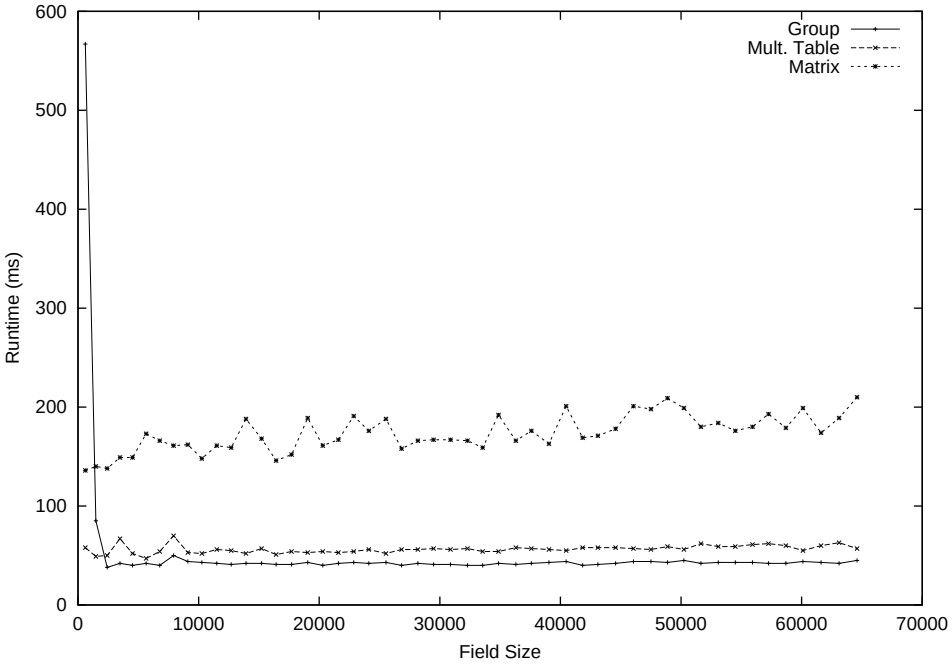


Figure 5.15: Experimental results for computing A^{ψ_q} with a fixed 20-dimensional algebra. We compare the group algebra specific algorithm with the standard PowerSubalgebra algorithm applied to multiplication tables and the regular matrix representation. Runtimes are in CPU milliseconds.

as the group algebra specific algorithm does not demonstrate consistent growth characteristics. We obtain an estimate of $0.5598 + 1.1685 \ln d$ for the multiplication table encoding and an estimate of $-0.3009 + 1.6655 \ln d$ for the regular matrix representation. The estimates are better than the proposed theoretical complexity, suggesting that we are most likely not in a worst case scenario.

Figure 5.15 contains a graph summarizing the second experiment, where the algebra is held fixed and the field size varies. The theoretical complexity contains only a $\log q$ factor, where q is the size of the field, so we expect very little difference due to changing the field size. From the graph, we see this is in fact true: the runtime grows very slowly as the field size is increased.

The anomalous behavior for the group algebra algorithm at small field sizes can be explained for $\mathbb{F}_2$, but not the other fields. For $\mathbb{F}_2$, the group algebra algorithm is not applicable, because 2 divides the order of the group (20). Hence, the cost includes computing the center as well as using the standard algorithm for computing A^{ψ_q} in this case.

In general, we see that the multiplication table encoding performs much better than the regular representation encoding. This is most likely due to a lower cost of algebra multiplications.

5.6.2 Primitive Central Idempotents

In this subsection, we analyze data collected for our implementations of FRMain, the algorithm of Friedl and Rónyai, GianniIdempotents, the algorithm of Gianni, et al., DavisIdempotents, the algorithm of Davis, EberlyGiesbrechtLarge, the algorithm of Eberly and Giesbrecht for large fields, and EberlyGiesbrechtSmall, the algorithm of Eberly and Giesbrecht for small fields. For each experiment, $Z(A)$ and A^{ψ_q} are precomputed. The algebra A is converted to the multiplication table encoding and its regular matrix representation prior to calculating $Z(A)$ and A^{ψ_q}. The runtimes shown include only costs incurred while executing each idempotent algorithm.

We performed two different experiments. In the first experiment, we defined each group algebra using the field $\mathbb{F}_{53}$. Implementations of FRMain, GianniIdempotents, DavisIdempotents, and EberlyGiesbrechtSmall were executed on each group algebra and the runtime was recorded. In the second experiment, implementations of DavisIdempotents and EberlyGiesbrechtLarge were executed using the field $\mathbb{F}_{65521}$ to define each group algebra. We begin by reviewing the data collected for the first experiment.

Figures 5.16–5.19 contain graphs summarizing the data collected for each algorithm. The runtimes for each encoding are displayed in the graphs. The multiplication table encoding is generally the best encoding to use with FRMain, GianniIdempotents, and EberlyGiesbrechtSmall. The multiplication table encoding for group algebras is sparse, and the data show that the sparseness is being used efficiently. The matrix encoding suffers from unnecessarily multiplying zeroes, but still performs better in general than the native group algebra encoding. The peaks in each graph correspond to test cases where a large number of idempotents exist. Because the dimension of the input algebra to each algorithm is the number of idempotents, this behavior is expected.

For DavisIdempotents, the behavior is more unpredictable. Although generally the multiplication table encoding performs the best, several datapoints exist where the matrix encoding performs best (e.g., at dimensions 41–43). We also see situations where the matrix encoding performs the worst (e.g., at dimensions 27 and 48). This is in contrast to the performance of the other three algorithms, where each encoding behaves similarly. Furthermore, the number of idempotents is not a clear indicator for the location of peaks, although peaks appear to be more likely to occur where the number of idempotents is large.

Next, we compare algorithms for each encoding used. Figures 5.20–5.22 contain graphs summarizing the performance of each algorithm on the group algebra encoding, multiplication table encoding,

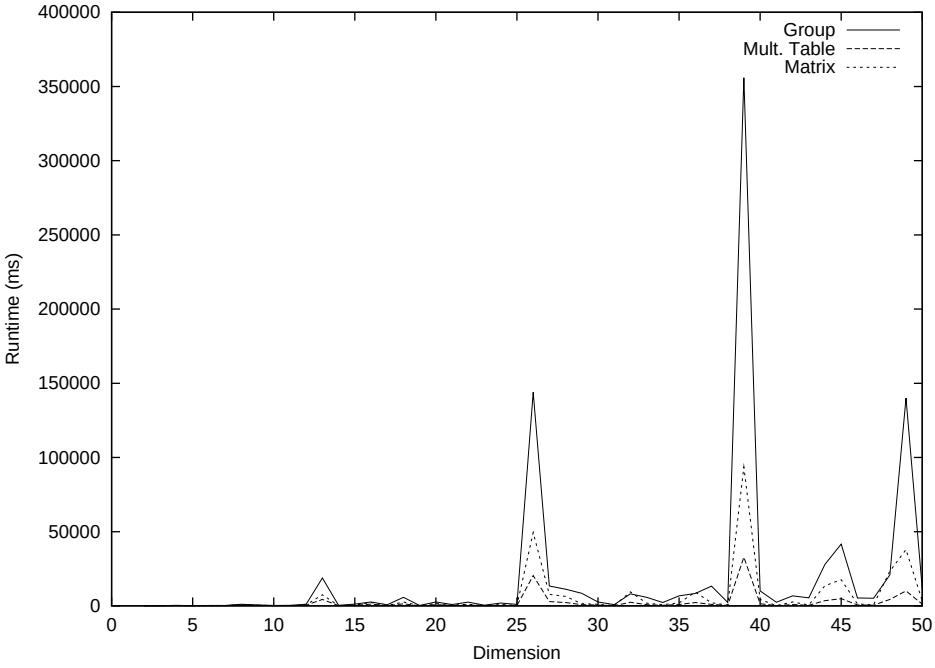


Figure 5.16: Summary of runtimes (in CPU milliseconds) for FRMAIN.

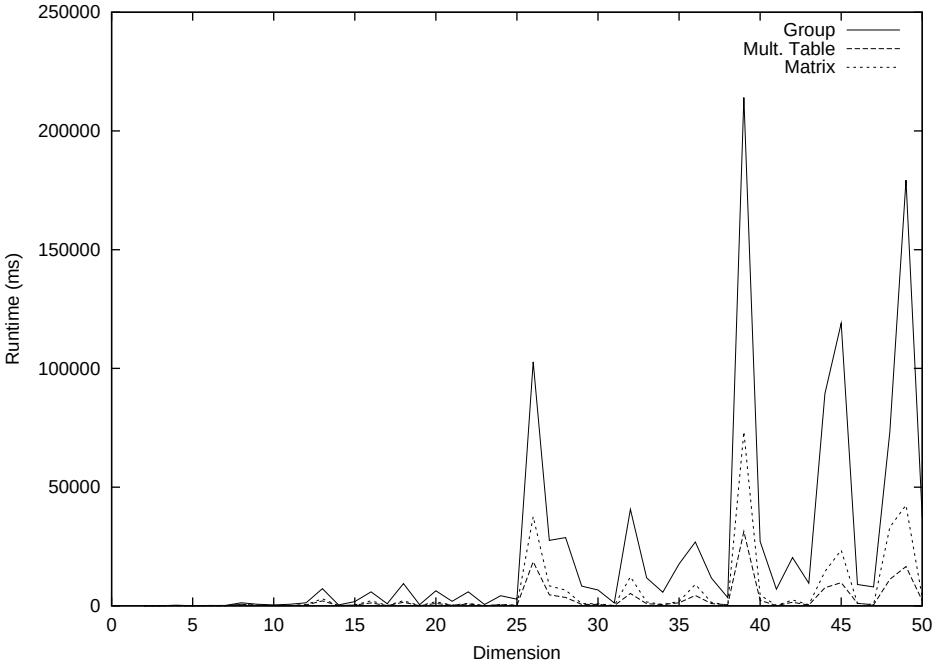


Figure 5.17: Summary of runtimes (in CPU milliseconds) for GianniIdempotents.

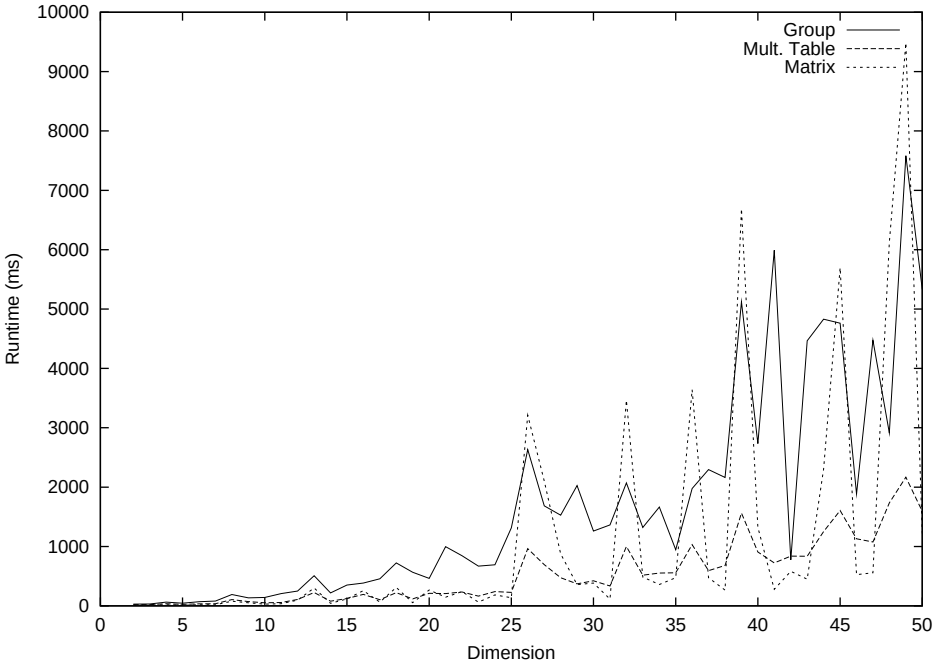


Figure 5.18: Summary of runtimes (in CPU milliseconds) for DavisIdempotents.

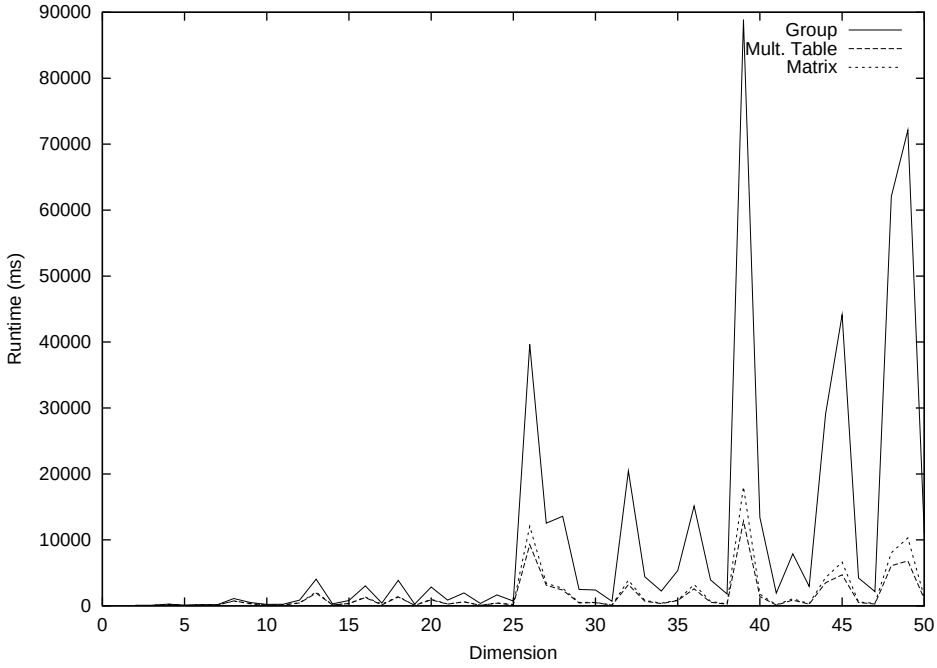


Figure 5.19: Summary of runtimes (in CPU milliseconds) for EberlyGiesbrechtSmall.

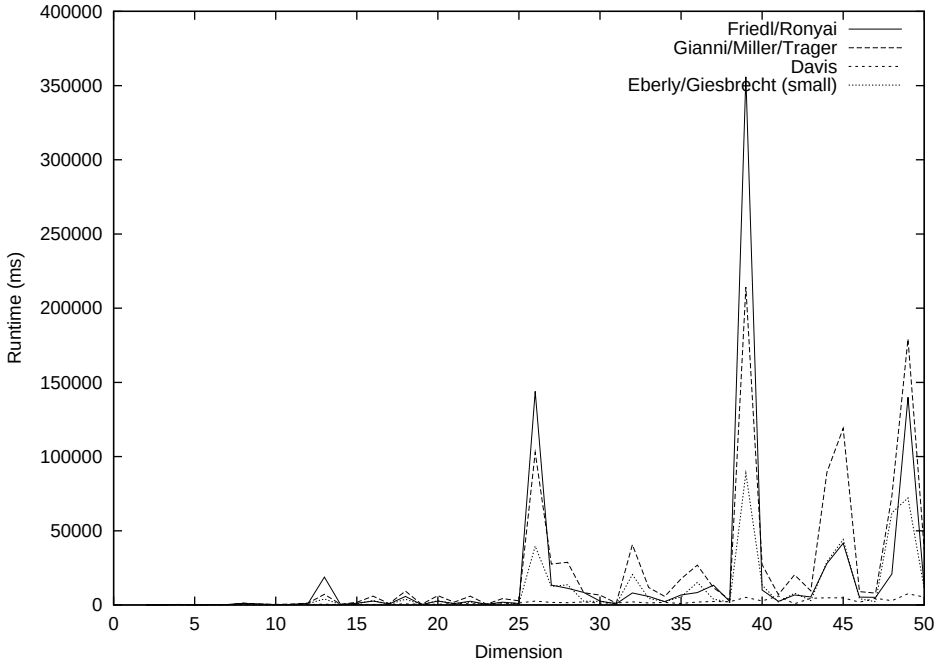


Figure 5.20: Summary of runtimes (in CPU milliseconds) for each idempotent algorithm executed on the group algebra encoding (small field).

and matrix algebra encoding respectively. For the group algebra encoding, DavisIdempotents generally performs the best, especially when the number of idempotents is large, causing the other algorithms to have large peaks. For the multiplication table encoding, DavisIdempotents performs well overall, but the other three algorithms tend to perform better when the dimension of the original group algebra is large, but the number of idempotents is small (implying $\dim A^{\psi_q}$ is small). Still, when the number of idempotents is large, DavisIdempotents is generally the best. For the matrix algebra encoding, both DavisIdempotents and EberlyGiesbrechtSmall perform very well, even when the number of idempotents is large.

The performance of DavisIdempotents is suprising, given the analysis in Theorem 5.2.12. In part, this is due to the fact that the implementation uses an improvement of Davis over the basic algorithm we presented. We also conjecture that the cost of computing bases for intermediate ideals used in the other algorithms is a primary factor for their poorer performance, especially when the number of idempotents is large. Another factor for FRMain and EberlyGiesbrechtSmall is the

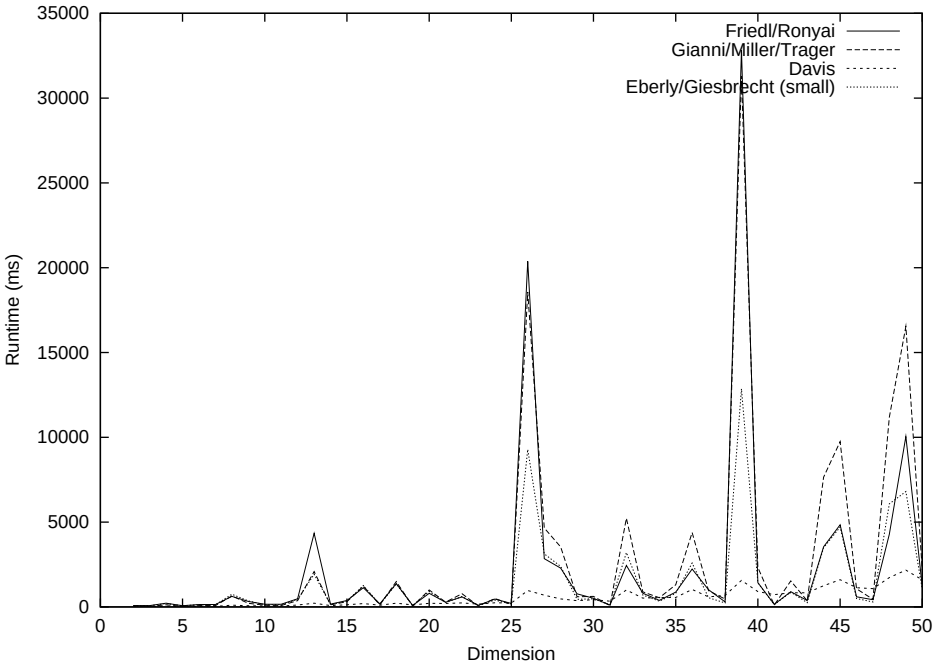


Figure 5.21: Summary of runtimes (in CPU milliseconds) for each idempotent algorithm executed on the multiplication table encoding (small field).

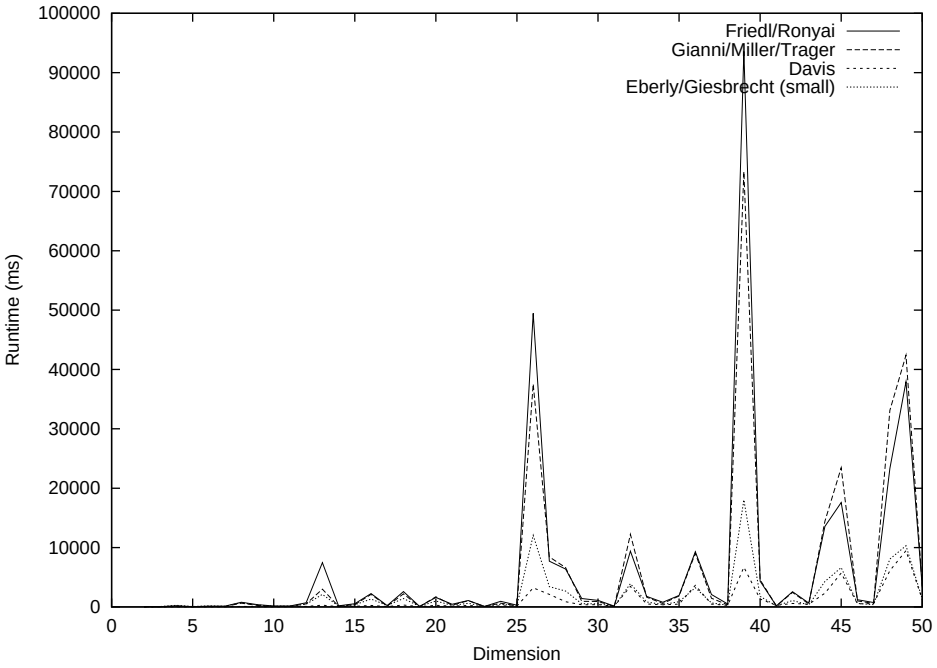


Figure 5.22: Summary of runtimes (in CPU milliseconds) for each idempotent algorithm executed on the matrix algebra encoding (small field).

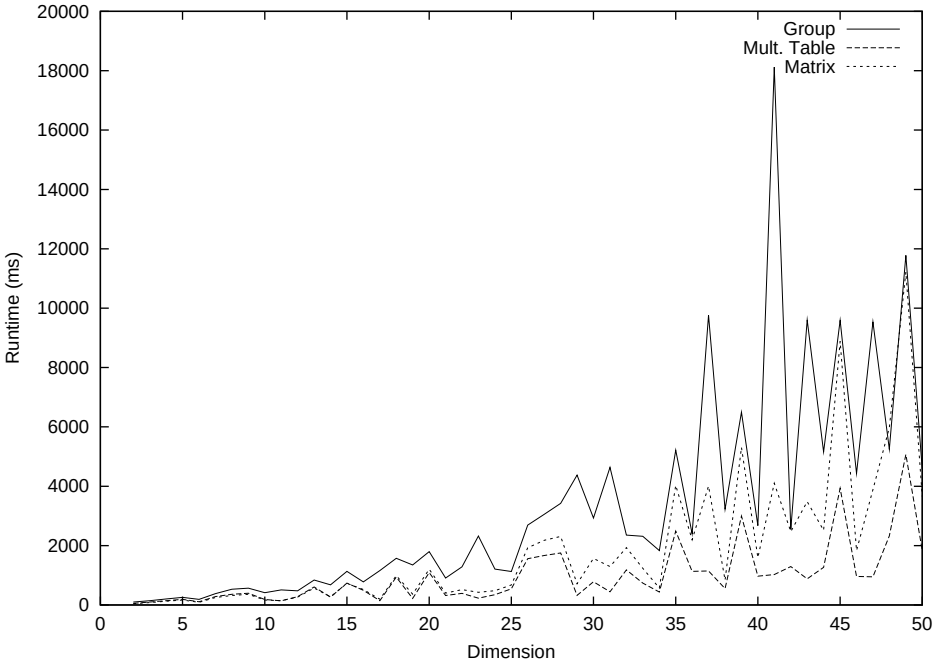


Figure 5.23: Summary of runtimes (in CPU milliseconds) for DavisIdempotents using $\mathbb{F}_{65521}$.

computation of minimal polynomials, which is expensive for an encoding neutral implementation.

We further investigate the performance of DavisIdempotents as well as the performance of EberlyGiesbrechtLarge in the second experiment, using $\mathbb{F}_{65521}$. Figures 5.23 and 5.24 contain graphs summarizing the results for each algorithm executed on each algebra encoding. For both algebras, the multiplication table encoding is generally the best, and the group algebra encoding is the worst. In constrast to DavisIdempotents executed when the field is $\mathbb{F}_{53}$, the behavior is more predictable.

The peaks appearing for EberlyGiesbrechtLarge occur when the ratio of idempotents to dimension of the original algebras approaches one. We see similar peaks for DavisIdempotents, although they are not as pronounced, and do not always correspond directly to the number of idempotents (e.g., the 41-dimensional group algebra in the group algebra encoding is the most expensive to compute even though it only has 6 idempotents).

The graphs in Figures 5.25–5.27 compare DavisIdempotents and EberlyGiesbrechtLarge

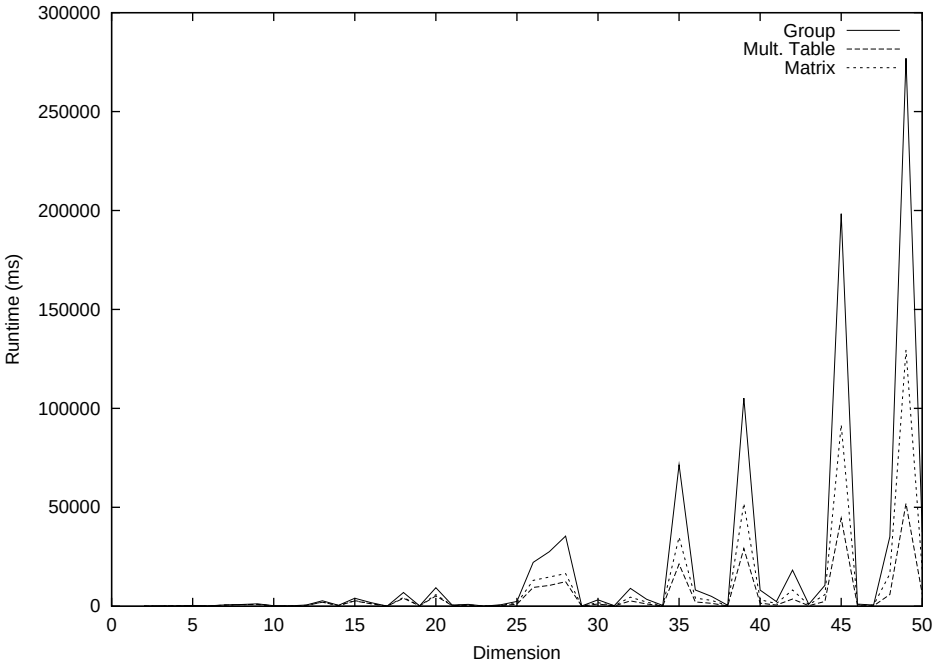


Figure 5.24: Summary of runtimes (in CPU milliseconds) for EberlyGiesbrechtLarge using $\mathbb{F}_{65521}$.

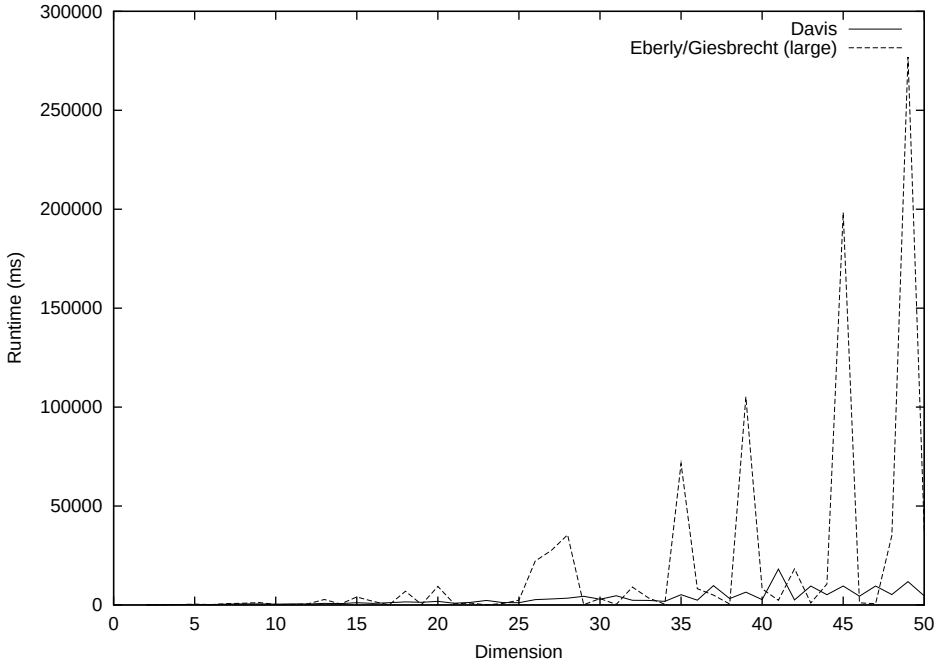


Figure 5.25: Summary of runtimes (in CPU milliseconds) for each idempotent algorithm executed on the group algebra encoding (large field).

executed on the group algebra encoding, multiplication table encoding, and matrix algebra encoding respectively. We see the same behavior in all three cases. When the number of idempotents is small, regardless of the dimension size of the original algebra, EberlyGiesbrechtLarge performs better. However, DavisIdempotents performs better when the number of idempotents is large. This is most likely due to the computation of the minimal polynomial in EberlyGiesbrechtLarge, which is directly affected by the number of idempotents (since it is the dimension of the input algebra).

We also note that the impact of increasing the field size did not affect the performance of DavisIdempotents as dramatically as would be expected from Theorem 5.2.12. Again, one reason is due to the implementation of Davis' improvement to the basic algorithm. We cannot directly compare the runtimes of the two experiments because the number of idempotents is frequently different for group algebras with the same dimension. However, by scanning the data tables directly, one sees that the runtimes are of approximately the same magnitude.

We do not have experimental data when the field $\mathbb{K}$ has characteristic zero because only one

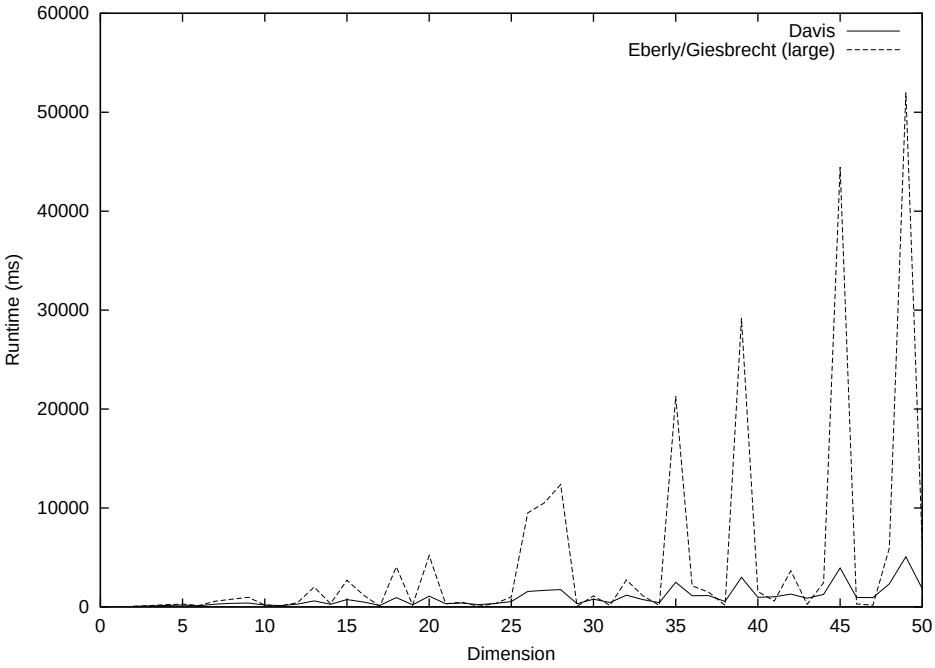


Figure 5.26: Summary of runtimes (in CPU milliseconds) for each idempotent algorithm executed on the multiplication table encoding (large field).

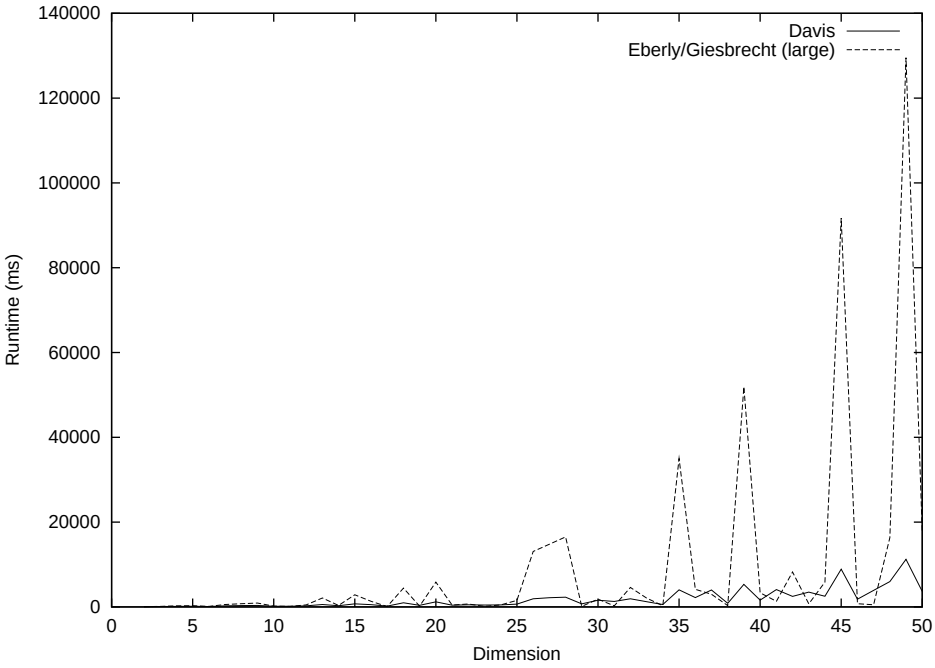


Figure 5.27: Summary of runtimes (in CPU milliseconds) for each idempotent algorithm executed on the matrix algebra encoding (large field).

of the implemented algorithms is applicable to such $\mathbb{K}$-algebras: EberlyGiesbrechtLarge. We expect the behavior and performance to be similar to our large field experiment, because the field size does not affect the encoding neutral complexity of operations. The cost of the field operations themselves is the only difference.

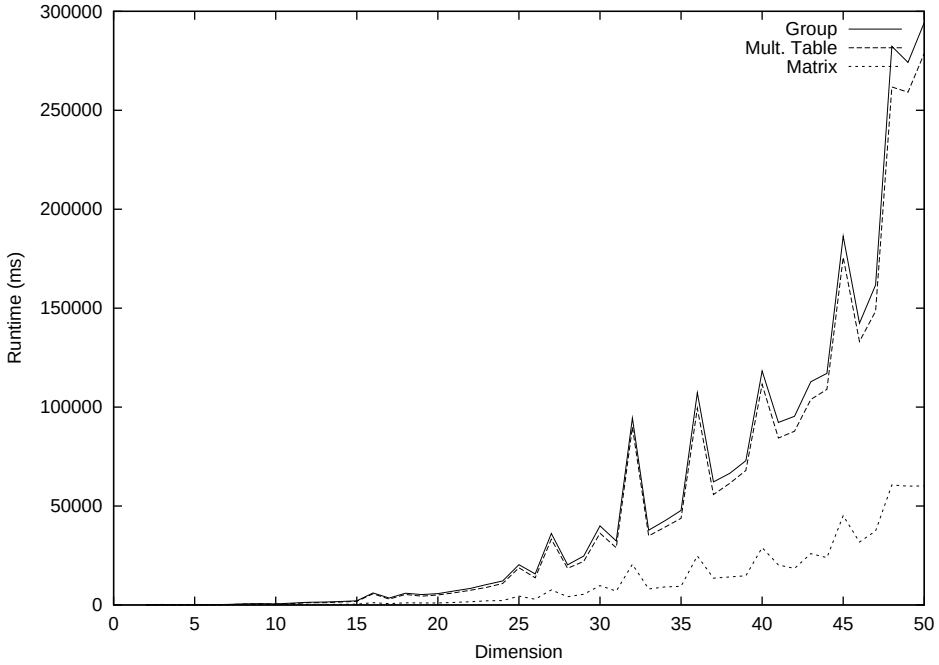


Figure 5.28: Summary of runtimes (in CPU milliseconds) for RadicalPrimeFR.

5.6.3 Jacobson Radical

In this subsection, we analyze data collected for our implementations of RadicalPrimeFR, the algorithm of Friedl and Rónyai calculating $J(A)$, and RadicalPrimeCIW, the algorithm of Cohen, Ivanyos, and Wales. For our experiment, we constructed group algebra $\mathbb{K}G$ with the field $\mathbb{K} = \mathbb{F}_p$, where p is the largest prime factor of the order of G. Selecting the field in this manner guarantees that $\mathbb{K}G$ has a non-trivial radical.

Figure 5.28 contains a graph summarizing the results for RadicalPrimeFR. We see that the matrix algebra encoding performs much better than either the group algebra or multiplication table encodings. This observation suggests that a primary cost in the encoding neutral version of RadicalPrimeFR is the conversion of elements to regular matrix representation. The peaks in the graph are difficult to explain without further analysis of the particular group algebra. Since the peaks occur for each encoding, we hypothesize that the peaks are exhibiting close to worst case behavior.

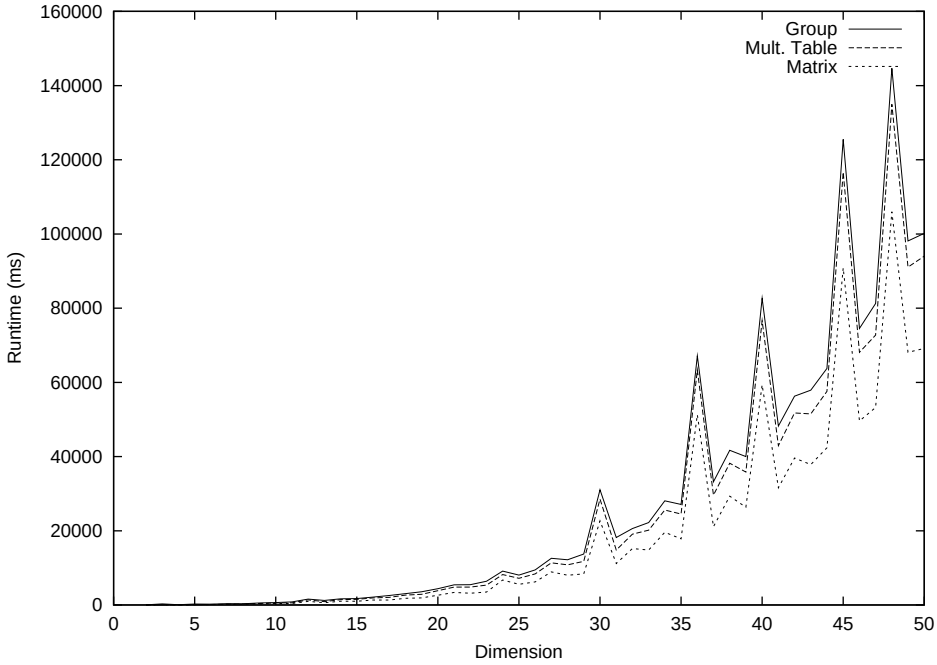


Figure 5.29: Summary of runtimes (in CPU milliseconds) for RadicalPrimeCIW.

Figure 5.29 contains a graph summarizing the results for RadicalPrimeCIW. Again, the matrix algebra encoding performs the best. The impact of using the group algebra or path algebra encoding is not as significant as for RadicalPrimeFR, implying that the conversion of elements to regular matrix representation is not as significant of a factor for RadicalPrimeCIW. The peaks in the graph for RadicalPrimeCIW appear at the same location as for RadicalPrimeFR, supporting the hypothesis that these particular algebras are exhibiting worst case characteristics.

Figures 5.30–5.32 compare the algorithms on each of the group algebra, multiplication table, and matrix algebra encodings. We see that RadicalPrimeFR performs better on algebras in their matrix algebra encoding, but worse in either of the group algebra or multiplication table encodings. This suggests that RadicalPrimeCIW is better to use in an encoding neutral environment, but RadicalPrimeFR is better if algebras are always encoded as matrix algebras.

We must be careful interpreting these results though. In our experiment, we worked only with prime fields (i.e., $\mathbb{F}_p$ where p is a prime, but not $\mathbb{F}_q$ where q is a power of a prime). Recall that

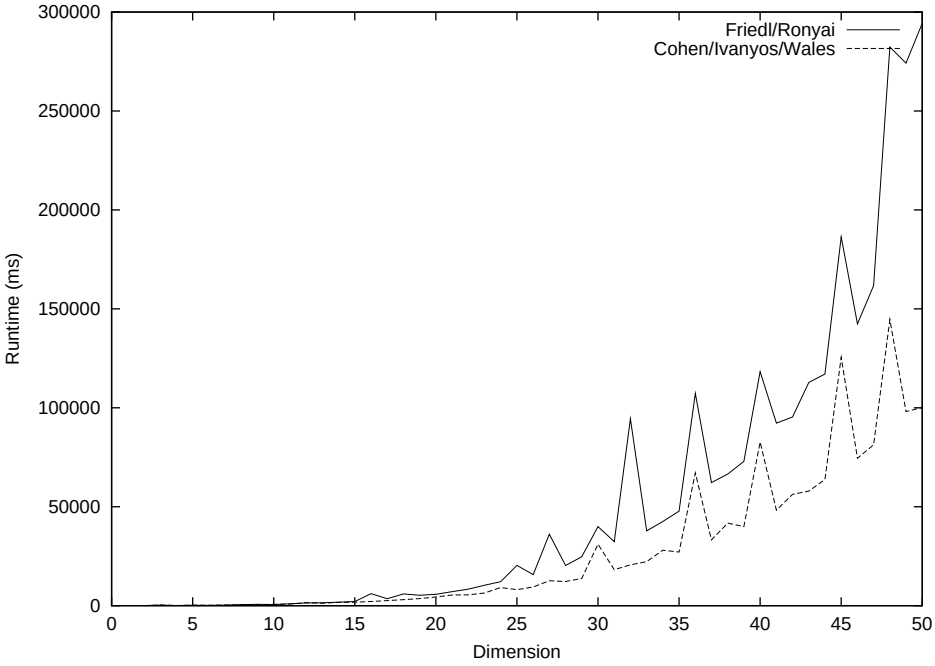


Figure 5.30: Summary of runtimes (in CPU milliseconds) for each radical algorithm executed on the group algebra encoding.

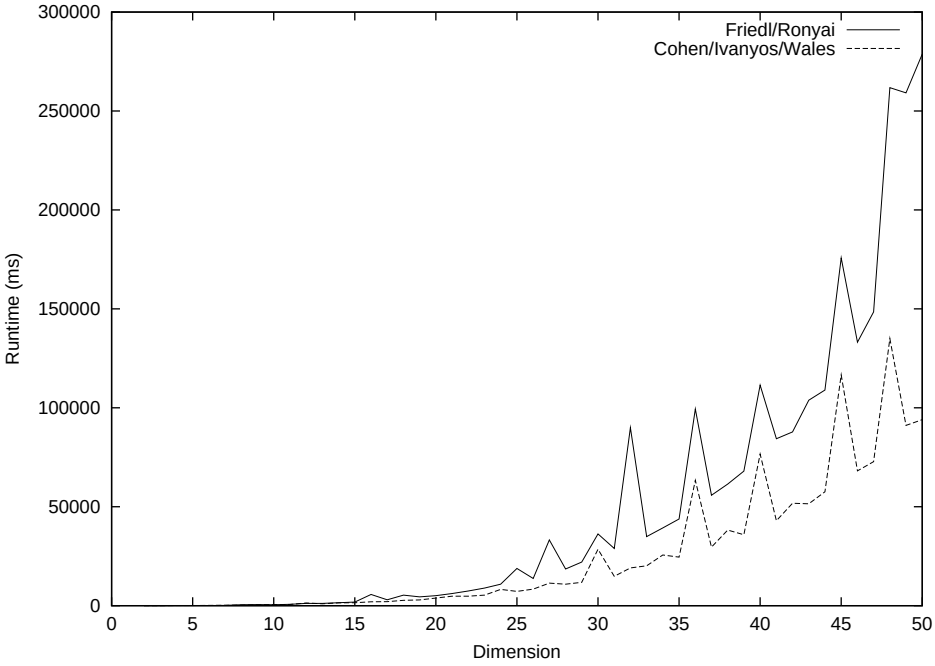


Figure 5.31: Summary of runtimes (in CPU milliseconds) for each radical algorithm executed on the multiplication table encoding.

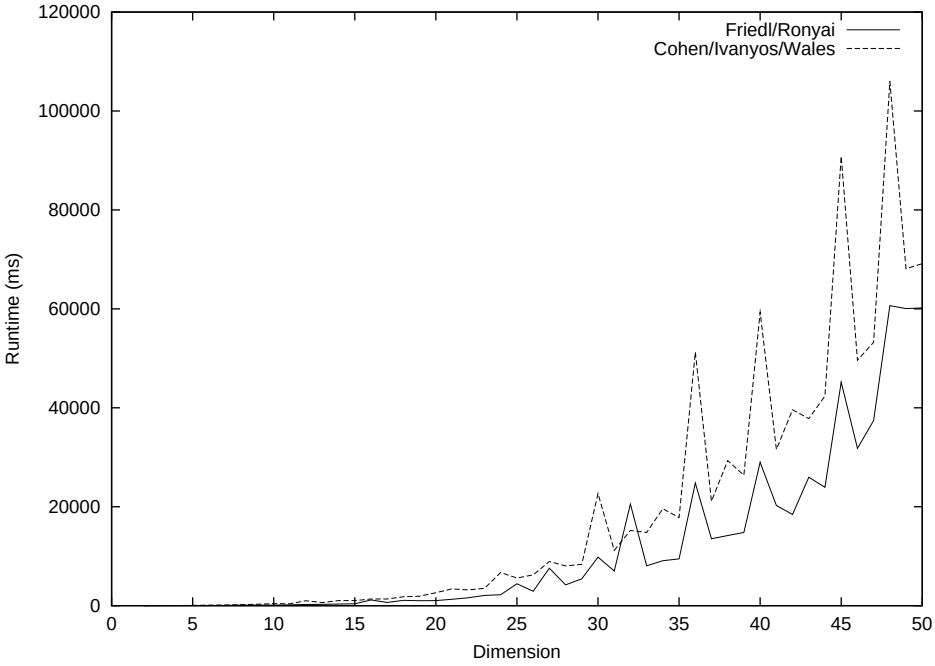


Figure 5.32: Summary of runtimes (in CPU milliseconds) for each radical algorithm executed on the matrix algebra encoding.

RadicalPrimeFR requires an increase in the dimension of the input algebra A when A is an $\mathbb{F}_q$-algebra, but RadicalPrimeCIW does not. We did not run any experiments to test the impact of increased dimension because an implementation to properly handle this situation was not available (suggesting another reason to use RadicalPrimeCIW in practice).

5.7 Summary

In this chapter, we presented and analyzed several algorithms related to the problems **ALGEBRA DECOMPOSITION** and **SEMISIMPLE ALGEBRA DECOMPOSITION**. We experimentally compared several of the presented algorithms to validate the analysis we performed. We also compared the performance of each algorithm on each of the group algebra, multiplication table, and matrix algebra encodings.

In general, the performance of each algorithm was best for the multiplication table encoding. The primary exception was the algorithms to calculate $J(A)$, the Jacobson radical of a finite dimensional $\mathbb{F}_q$-algebra.

A surprising result was the performance of Davis' algorithm DavisIdempotents for computing primitive central idempotents in $\mathbb{F}_q$-algebras. The theoretical analysis suggested Davis' algorithm would perform poorly, especially as the size of $\mathbb{F}_q$ increased. In fact, Davis' algorithm performed the best overall in our experiments, even with large increases in field size. We hypothesize that the cost of the computation of bases for intermediate ideals, which was not accounted for in our analysis, is a primary factor for the poor performance of the other idempotent algorithms. Also, the calculation of minimal polynomials in Split, EberlyGiesbrechtSmall, and EberlyGiesbrechtLarge may be particularly costly for encoding neutral algorithms.

For calculating the Jacobson radical of $\mathbb{F}_q$-algebras, RadicalPrimeFR, the algorithm of Friedl and Rónyai, performs best when q is a prime number and A is encoded as a matrix algebra. In an encoding neutral environment or when $q = p^k$, where $k > 1$, RadicalPrimeCIW, the algorithm of Cohen, et al., is more appropriate.

When finding idempotents in a $\mathbb{F}_q$-algebra, it is generally more efficient to calculate A^{ψ_q} than to to calculate $J(A)$. Since A^{ψ_q} is semisimple, and all primitive central idempotents are contained in A^{ψ_q}, we recommend working in A^{ψ_q} to find the primitive central idempotents. However, for some applications it is necessary to compute $J(A)$ of an $\mathbb{F}_q$-algebra and then find the primitive central idempotents of $A/J(A)$. For those applications, we recommend using RadicalPrimeCIW unless A is always encoded as a matrix algebra, and q is a prime number.

We briefly reviewed the results of Rónyai, showing that finding an explicit isomorphism of a simple $\mathbb{Q}$-algebra is at least as difficult as factoring squarefree integers. As a result, **SEMISIMPLE ALGEBRA DECOMPOSITION** is not currently tractable for $\mathbb{Q}$-algebras.

We conclude this chapter with pseudocode for recommended solutions to **ALGEBRA DE-**

AlgebraDecomposition(A) Decompose of A into the direct sum of indecomposable ideals.

INPUT: A finite dimensional $\mathbb{Q}$ or $\mathbb{F}_q$-algebra A.
OUTPUT: A list of indecomposable ideals $I_1, \ldots, I_n$ such that $A = I_1 \oplus \cdots \oplus I_n$.

```
1   K ← A.field
2   Z ← Z(A)
3   if K = ℚ
4     then J ← RadicalZero(A)
5          R ← Z/J
6          idem ← EberlyGiesbrechtLarge(R)
7          idem ← LiftIdempotents(Z, J, idem)
8     else R ← PowerSubalgebra(A, 1)
9          p ← K.characteristic
10         idem ← DavisIdempotents(R, p)
11  n ← |idem|
12  return [Ideal(A, idem[i])]_{i=1}^n
```

Figure 5.33: Recommended encoding neutral solution to **ALGEBRA DECOMPOSITION**.

COMPOSITION and **SEMISIMPLE ALGEBRA DECOMPOSITION** based on the results we obtained. Figure 5.33 contains pseudocode for **ALGEBRA DECOMPOSITON** when A is an $\mathbb{F}_q$ or $\mathbb{Q}$ algebra. Figure 5.34 contains pseudocode for **SEMISIMPLE ALGEBRA DECOMPOSITION** when A is an $\mathbb{F}_q$ algebra.

SSAlgebraDecomposition(A) Decompose A into the direct sum of simple ideals.

INPUT: A finite dimensional semisimple $\mathbb{F}_q$-algebra A.
OUTPUT: A list of simple ideals $I_1, \ldots, I_n$ such that $A = I_1 \oplus \cdots \oplus I_n$, and isomorphisms $\rho_i : I_i \to M_{n_i}(D_i)$.

```
1   ideals ← AlgebraDecomposition(A)
2   for each I ∈ ideals
3       do ▷ Find a primitive idempotent in I
4          V ← I
5          while not IsCommutative(V)
6                do x ← RandomizedZeroDiv(V)[1]
7                   R ← RightIdeal(V, [x])
8                   li ← LeftIdentity(R)
9                   V ← li * V * li
10         e ← 1_V   ▷ This is not 1_A
11         Append result of ExplicitIsomorphism(I, e) to rho.
12  return rho
```

Figure 5.34: Recommended encoding neutral solution to **SEMISIMPLE ALGEBRA DECOMPOSITION** for $\mathbb{F}_q$-algebras.

Chapter 6

The Drinfel'd Double

A Hopf algebra H is a (not necessarily commutative) algebra over a field $\mathbb{K}$ with additional structure that also makes H a coalgebra together with an additional operation called an antipode. Hopf algebras are an important algebraic structure [75]. Quantum groups, an important area of study both in mathematics and physics [68, 69, 70, 71, 72, 79], are Hopf algebras with special properties. Quantum groups are important in the study of solutions to the Yang-Baxter equations of quantum physics [63, 71].

The kind of Hopf algebra that arises in the study of the Yang-Baxter equations are the quasi-triangular Hopf algebras. Drinfel'd [30] shows that any Hopf algebra H can be extended to a larger Hopf algebra $D(H)$ that is quasi-triangular. The Hopf algebra $D(H)$ is often called the Drinfel'd double of H. If H is finite-dimensional, then $D(H)$ has dimension the square of the dimension of H. One difficulty in the construction of $D(H)$ is the tremendous increase in the space required to encode $D(H)$ as a matrix algebra. We elaborate on this difficulty later.

This chapter reports on our implementation in GAP 3.4.4 of the Drinfel'd double construction for a finite-dimensional Hopf algebra. In the process, we implement the dual of a Hopf algebra and the tensor product of algebras. The output of our construction includes a matrix algebra encoding of the Drinfel'd double so that GAP functions that expect a parameter that is a matrix algebra can be applied to $D(H)$. Green [47] introduces the concept of covering algebras, a parameterized family of finite-dimensional Hopf algebras based on path algebras. We implement covering algebras and utilize these algebras as test input for the Drinfel'd double construction. Our GAP code constitutes

the first implementations of the Drinfel'd double construction and of general covering algebras.

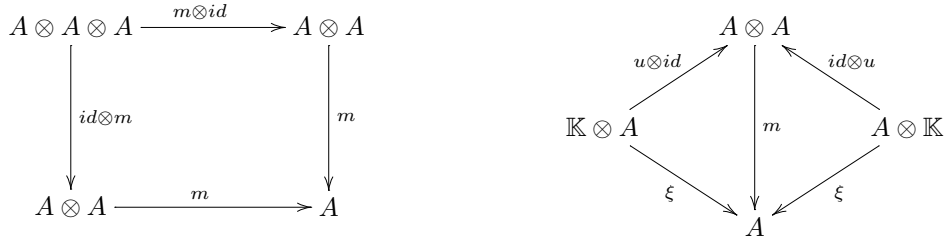
In testing the Drinfel'd double construction, we found that the straightforward GAP implementation is very slow, even for H of modest dimension. Subsequently, we developed several alternate implementations based on caching results of operations that are recomputed frequently within the straightforward implementation. These alternate implementations all improve the time performance of the construction. The remaining bottleneck to computing $D(H)$ for higher dimensional H is the rapidly expanding space required to encode $D(H)$ as a matrix algebra. In the HOPF project, we are providing the two alternatives of encoding $D(H)$ as a matrix algebra or of encoding $D(H)$ in a much more space efficient format.

The remainder of the chapter is structured as follows. In Section 6.1, we review the necessary mathematical background for Hopf algebras and the Drinfel'd double construction. Section 6.2 gives our explicit algorithm for the Drinfel'd double and details the important aspects of its GAP implementation. Section 6.3 describes the caching ideas behind our alternate implementations. In Section 6.4, we define covering algebras and discuss two implementations in GAP. Section 6.5 develops the Drinfel'd double construction for Hopf algebras derived from group algebras. In Section 6.6, we report on the performance of the different implementations and arrive at some recommendations. Finally, Section 6.7 briefly summarizes the chapter and suggests some additional research directions. An earlier version of the work in this chapter appears in Brunick, et al. [13].

6.1 The Drinfel'd Double Construction

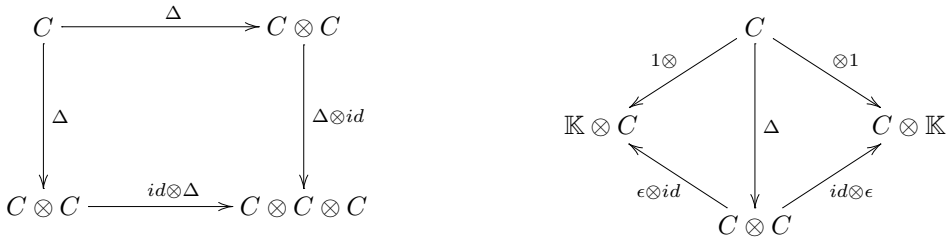
This section presents terminology and the Drinfel'd double construction. We assume the reader is familiar with vector spaces and algebras. Definitions not found here can be found in a standard algebra text such as Hungerford [57]. The presentation here follows that of Montgomery [75] and is included for completeness. Throughout, $\mathbb{K}$ denotes a field and $\mathbb{F}_q$ denotes the finite field of cardinality q. If V and W are vector spaces over $\mathbb{K}$, then $Hom_{\mathbb{K}}(V, W)$ denotes the $\mathbb{K}$-linear maps from V to W. All tensors are over the common base field of the vector spaces.

Suppose A is a $\mathbb{K}$-vector space with the identity map id. Let $\xi : \mathbb{K} \otimes A \cup A \otimes \mathbb{K} \to A$ denote scalar multiplication in A. Suppose $m : A \otimes A \to A$ is a $\mathbb{K}$-linear map (called *multiplication*) such that the diagram for associativity in Figure 6.1(a) commutes. Suppose $u : \mathbb{K} \to A$ is a $\mathbb{K}$-linear map (called *unit*) such that the diagram in Figure 6.1(b) commutes. The vector space A together with



(a) (b)

Figure 6.1: Commutative diagrams for (a) associativity of m and (b) unit map u



(a) (b)

Figure 6.2: Commutative diagrams for (a) coassociativity of Δ and (b) counit map ϵ

the maps m and u form a $\mathbb{K}$-*algebra*. For $x, y \in A$, we often write $x \cdot y$ or just xy for $m(x \otimes y)$.

To construct an example, suppose H_4 is a 4 dimensional $\mathbb{F}_3$-vector space with basis $\{1, g, x, y\}$. Let multiplication in H_4 be defined by the relations $\{1 \cdot g = g \cdot 1 = g, 1 \cdot x = x \cdot 1 = x, 1 \cdot y = y \cdot 1 = y, 1^2 = g^2 = 1, x^2 = 0, gx = y, xg = -y\}$, and let $u(k) = k \cdot 1$, for all $k \in \mathbb{K}$. Then, H_4 together with the defined multiplication and unit form an algebra. (H_4 is the smallest noncommutative and noncocommutative Hopf algebra and was first described by Sweedler [92]. We define cocommutativity and continue to build the Hopf algebra structure for H_4 as we continue the definitions of this section.)

As we now define, a coalgebra has a structure dual to that of an algebra. Let C be a $\mathbb{K}$-vector space. Suppose $\Delta : C \to C \otimes C$ is a $\mathbb{K}$-linear map (called *comultiplication*) such that the diagram for coassociativity in Figure 6.2(a) commutes. Suppose $\epsilon : C \to \mathbb{K}$ is a $\mathbb{K}$-linear map (called *counit*) such that the diagram in Figure 6.2(b) commutes. Then, the vector space C together with the two maps Δ and ϵ is a $\mathbb{K}$-*coalgebra*.

One may verify that H_4 is also a coalgebra with comultiplication Δ and counit ϵ defined on the

basis elements as follows:

$$\begin{aligned}
\Delta(1) &= 1 \otimes 1 \\
\Delta(g) &= g \otimes g \\
\Delta(x) &= x \otimes 1 + g \otimes x \\
\Delta(y) &= y \otimes g + 1 \otimes y \\
\epsilon(1) &= 1 \\
\epsilon(g) &= 1 \\
\epsilon(x) &= 0 \\
\epsilon(y) &= 0.
\end{aligned}$$

Generally, the result of a comultiplication can be written as $\Delta(c) = \sum_{i=1}^{k} a_i \otimes b_i$, where each $a_i, b_i \in C$. To simplify writing expressions for Hopf algebras, we use *sigma notation* for Δ, written

$$\Delta(c) = \sum c_{(1)} \otimes c_{(2)}.$$

For example, comultiplying $x \in H_4$ gives

$$\Delta(x) = x \otimes 1 + g \otimes x$$

by definition. Thus, in the first summand $x_{(1)} = x$ and $x_{(2)} = 1$ and in the second summand $x_{(1)} = g$ and $x_{(2)} = x$. In a computer implementation, sigma notation translates into a loop over all summands, processing each summand individually. Sigma notation is powerful when applying Δ multiple times, because of coassociativity:

$$\begin{aligned}
\sum c_{(1)} \otimes \Delta(c_{(2)}) &= \sum c_{(1)} \otimes c_{(2)_{(1)}} \otimes c_{(2)_{(2)}} \\
&= \sum c_{(1)_{(1)}} \otimes c_{(1)_{(2)}} \otimes c_{(2)} \\
&= \sum c_{(1)} \otimes c_{(2)} \otimes c_{(3)},
\end{aligned}$$

where in this last summation the $c_{(2)}$ notation means the second C value in a tensor of 3 elements of C.

If C is a coalgebra, then C^{cop} denotes the coalgebra with opposite comultiplication. In sigma notation, we have

$$\begin{aligned}
\Delta_C(c) &= \sum c_{(1)} \otimes c_{(2)} \\
\Delta_{C^{cop}}(c) &= \sum c_{(2)} \otimes c_{(1)}.
\end{aligned}$$

A coalgebra C is *cocommutative* if $\Delta_C = \Delta_{C^{cop}}$.

We note that H_4 has both algebra and coalgebra structures. It is possible to have algebra and coalgebra structures coexist, as follows. A $\mathbb{K}$-vector space B is a *bialgebra* if B is both an algebra and a coalgebra and if, in addition,

$$\begin{aligned} \Delta(ab) &= \Delta(a)\Delta(b) \\ \epsilon(ab) &= \epsilon(a)\epsilon(b) \end{aligned}$$

for all $a, b \in B$. In the H_4 example, we verify that $\Delta(gx) = \Delta(g)\Delta(x)$:

$$\begin{aligned} \Delta(g)\Delta(x) &= (g \otimes g)(x \otimes 1 + g \otimes x) \\ &= gx \otimes g + g^2 \otimes gx \\ &= y \otimes g + 1 \otimes y \\ &= \Delta(y) \\ &= \Delta(gx). \end{aligned}$$

The reader may readily complete the verification that H_4 is a bialgebra.

A bialgebra H is a *Hopf algebra* if there exists a map $S \in Hom_{\mathbb{K}}(H, H)$ satisfying (in sigma notation)

$$\sum S(h_{(1)})h_{(2)} = \epsilon(h)1_H = \sum h_{(1)}S(h_{(2)})$$

for all $h \in H$. S is called an *antipode* for H. For example, the antipode S for H_4 is given by

$$\begin{aligned} S(1) &= 1 \\ S(g) &= g \\ S(x) &= -y \\ S(y) &= x. \end{aligned}$$

We note that the antipode is an anti-algebra map, i.e., $S(hk) = S(k)S(h)$, for all $h, k \in H_4$ (see Montgomery [75], Proposition 1.5.10). This is important to keep in mind when implementing an antipode map defined only on the generators of the Hopf algebra.

Constructing examples of Hopf algebras for study is often difficult. The Drinfel'd double constructs a larger Hopf algebra $D(H)$ from an existing Hopf algebra H, thus expanding the base of

examples available for study. Additionally $D(H)$ is quasi-triangular, implying that it generates a solution to the Yang-Baxter equation in quantum mechanics. For the remainder of this chapter, we assume the vector spaces, algebras, and coalgebras under consideration are finite dimensional.

Recall that the *dual* for a $\mathbb{K}$-vector space V is $V^* = Hom_{\mathbb{K}}(V, \mathbb{K})$. Let A be a $\mathbb{K}$-algebra. The vector spaces V and V^* determine a non-degenerate bilinear form $\langle , \rangle : V^* \otimes V \to \mathbb{K}$ via $\langle f, v \rangle = f(v)$. Let V and W be $\mathbb{K}$-vector spaces. If $\phi : V \to W$ is $\mathbb{K}$-linear, then the *transpose* of ϕ is $\phi^* : W^* \to V^*$, given by

$$\phi^*(f)(v) = f(\phi(v)),$$

for all $f \in W^*$ and $v \in V$.

Suppose that C is a coalgebra. Montgomery [75] shows that C^* is an algebra with multiplication $m = \Delta^*$ and unit $u = \epsilon^*$. Explicitly, m is given by $m(f \otimes g)(c) = \Delta^*(f \otimes g)(c) = (f \otimes g)\Delta c$, for all $f, g \in C^*$ and $c \in C$.

Suppose that A is an algebra. Montgomery also shows that A^* is a coalgebra with comultiplication $\Delta = m^*$ and counit $\epsilon = u^*$. Explicitly, Δ is given by $\Delta f(a \otimes b) = m^* f(a \otimes b) = f(ab)$, for all $f \in A^*$ and $a, b \in A$.

Montgomery [75, page 151] shows that given a Hopf algebra H, the dual H^* is also a Hopf algebra with m, u, Δ, ϵ defined by the previous two facts, and with the transposed antipode S^*.

We now define two actions for any finite dimensional algebra A and its dual A^*. The *left hatchet* $\rightharpoonup : A \otimes A^* \to A^*$ is defined as

$$h \rightharpoonup f \quad = \quad \sum \langle f_{(2)}, h \rangle f_{(1)}$$

for $h \in A$, $f \in A^*$, and $\Delta(f) = \sum f_{(1)} \otimes f_{(2)}$. The corresponding *right hatchet* $\leftharpoonup : A^* \otimes A \to A^*$ is defined as

$$f \leftharpoonup h \quad = \quad \sum \langle f_{(1)}, h \rangle f_{(2)}$$

for $h \in A$ and $f \in A^*$.

We can now define the Drinfeld double construction. Let H be a finite dimensional Hopf algebra. The *Drinfel'd double* $D(H) = H^{*^{cop}} \bowtie H$ has $H^{*^{cop}} \otimes H$ as its underlying vector space. The

operations for $D(H)$ are given by

$$(f \bowtie h)(f' \bowtie h') = \sum f((h_{(1)} \rightharpoonup f') \leftharpoonup S^{-1}h_{(3)}) \bowtie h_{(2)}h' \tag{6.1}$$

$$u_{D(H)}(k) = k(1_{H^*} \bowtie 1_H) \tag{6.2}$$

$$\Delta_{D(H)}(f \bowtie h) = \sum (f_{(2)} \bowtie h_{(1)}) \otimes (f_{(1)} \bowtie h_{(2)}) \tag{6.3}$$

$$\epsilon_{D(H)} = \epsilon_{H^{*cop}} \otimes \epsilon_H \tag{6.4}$$

$$S_{D(H)}(f \bowtie h) = \sum (S_H(h_{(2)}) \rightharpoonup S_{H^{*cop}}(f_{(1)})) \bowtie (f_{(2)} \rightharpoonup S_H(h_{(1)})), \tag{6.5}$$

for $f \in H^*$, $h \in H$, and $k \in \mathbb{K}$. $D(H)$ has identity element $1_{H^*} \bowtie 1_H$.

6.2 Algorithm and Implementation

We now present an algorithm for constructing the Drinfel'd double $D(H)$ of a finite dimensional Hopf algebra H. The output of the algorithm is a matrix algebra isomorphic to $D(H)$. Users of the HOPF system may provide the resulting matrix algebra as input to the Jacobson radical and algebra decomposition algorithms to study the algebra structure of $D(H)$. The remainder of this section discusses the algorithm and implementation details for constructing the Drinfel'd double.

6.2.1 Tensor Products

GAP has computational structures, called *domains*, which are used to encode structured sets and their elements for high-level computation. Computation of the Drinfel'd double requires tensor products of algebras. We have implemented a GAP domain that supports computations in tensor products of algebras.

Suppose that A and B are finite-dimensional algebras with bases $\{a_i \in A \,|\, 1 \leq i \leq m\}$ and $\{b_j \in B \,|\, 1 \leq j \leq n\}$, respectively. Then a basis for $A \otimes B$ is $\{a_i \otimes b_j \,|\, 1 \leq i \leq m, 1 \leq j \leq n\}$. Our Drinfel'd double algorithm relies on elements being encoded in terms of basis elements for the underlying vector space for $D(H)$. To do this, we implemented the function NORMALFORM that takes an element in a finite dimensional tensor product and writes the element in terms of basis elements (Figure 6.3). Suppose $a = \sum_{i=1}^{m} \alpha_i a_i \in A$ and $b = \sum_{j=1}^{n} \beta_j b_j \in B$ are arbitrary elements of A and B, respectively. The function call NORMALFORM$(a \otimes b)$ returns $\sum_{i=1}^{m} \sum_{j=1}^{n} \alpha_i \beta_j \; a_i \otimes b_j$. In the case where $x \in A \otimes B$ is the sum of tensor products of elements, NORMALFORM processes

NormalForm(V, e) Compute the normal form of e.

INPUT: A tensor product V of finite dimensional vector spaces, and an element e.
OUTPUT: The element e in terms of tensor products of basis elements.

```
1   A ← V.algebras
2   l ← |A|
3   for i ← 1 to l
4       do b[i] ← A[i].basis
5          d[i] ← A[i].dimension
6   summands ← e.summands
7   n ← |summands|
8   result ← 0_V
9   for i ← 1 to n
10      do mainCoeff ← summands[i].coefficient
11         for j ← 1 to l
12             do elem ← summands[i].elements[j]
13                Define elemCoeffs[j,k] by elem ← Σ_{k=1}^{d[j]} elemCoeffs[j,k] * b[j,k]
14         for each (c[1], ..., c[l]) ∈ [1..d[1]] × ··· × [1..d[l]]
15             do coeff ← mainCoeff * Π_{j=1}^{l} elemCoeffs[j, c[j]]
16                result ← result + coeff * (b[1, c[1]] ⊗ ··· ⊗ b[l, c[l]])
17  return result
```

Figure 6.3: Algorithm for computing the normal form of a tensor product.

each summand in turn and uses the addition operation implemented in the tensor product domain to collect common terms.

Theorem 6.2.1. *Let $V = A_1 \otimes \cdots \otimes A_l$ where $A_1, \ldots, A_l$ are finite dimensional $\mathbb{K}$-algebras of dimension $d_1, \ldots, d_l$. Let $d = \prod_{i=1}^{l} d_i$, the dimension of V. Furthermore, let $e \in V$ consist of n summands. The encoding neutral time complexity of* NormalForm *when given e and V as parameters is*

$$T_{\text{NormalForm}}(n, l, d) = \Theta\left(nld\,t(*_{\mathbb{K}}) + nd\,t(+_V) + n\sum_{j=1}^{l} t(\text{NormalForm}_{A_j})\right),$$

where NormalForm_{A_j} *is an algorithm that returns the coefficients of an element of A_j expressed in normal form.*

Proof. Line 13 presents an element from `summand` in terms of its underlying basis. Each of the l pos-

sible forms of NORMALFORM$_{A_j}$ is called n times, contributing the term $n \sum_{j=1}^{l} t(\text{NORMALFORM}_{A_j})$.

Line 15 contains l executions of $*_{\mathbb{K}}$ (multiplications in $\mathbb{K}$). There are $d = \prod_{i=1}^{l} d_i$ tuples in $[1..d_1] \times \cdots \times [1..d_l]$, and the set of tuples is iterated over n times, once for each summand in e. So line 15 contributes $nld\, t(*_{\mathbb{K}})$ to the time complexity.

Line 16 contains one execution of $+_V$ and one execution of $*_{\mathbb{K},V}$. We can let the cost $t(*_{\mathbb{K},V}) = 0$ by recognizing that `coeff` can be stored directly as the coefficient of $(b[1, c[1]] \otimes \cdots \otimes b[l, c[l]])$, requiring no arithmetic computation. From the previous paragraph, it is easily seen that line 16 is executed nd times. Hence the contribution for line 16 is $nd\, t(+_V)$.

The remainder of the lines can be implemented with simple statements, assignments, and array accesses. Hence, their total cost is negligible. The theorem follows directly. □

In some situations the cost of NORMALFORM$_{A_j}$ for the underlying vectors spaces in a tensor product is negligible. For example, matrix algebras, where the basis can be assumed in semi-echelon form, provide for very efficient computation of normal forms, while in group algebras the elements are always in normal form and the coefficients are readily obtained. The cost may be expensive though, such as when the underlying vector space is itself a tensor product of vector spaces. Hence, the cost of this computation cannot be removed from the complexity in all situations.

6.2.2 Constructing the Dual

The Drinfel'd double $D(H)$ of the Hopf algebra H is defined to be $H^{*^{cop}} \bowtie H$ where $H^{*^{cop}} \otimes H$ is the underlying vector space. We need to construct $H^{*^{cop}}$ for the Drinfel'd double. Instead of constructing $H^{*^{cop}}$ directly, we construct a matrix representation for H^*, as $H^{*^{cop}}$ is easily derivable from that representation.

As H is finite dimensional, we can directly construct a matrix algebra isomorphic to H^*. Let $\{h_1, \ldots h_n\}$ be a basis for H, and let $\{h_1^*, \ldots, h_n^*\}$ represent the dual basis for H^* with respect to the chosen basis for H. Define γ_{ijk} by $\Delta(h_i) = \sum_{j,k=1}^{n} \gamma_{ijk}\ h_j \otimes h_k$. Multiplication in H^* is given by

$$h_r^* h_s^* \quad = \quad \sum_t \gamma_{trs}\ h_t^*. \tag{6.6}$$

Define α_{ijk} by $h_i h_j = \sum_k \alpha_{ijk} h_k$. Comultiplication in H^* is given by

$$\Delta(h_r^*) \quad = \quad \sum_{s,t} \alpha_{str}\ h_s^* \otimes h_t^*. \tag{6.7}$$

Suppose that $\epsilon_H(h_i) = k_i$ for $k_i \in \mathbb{K}$. Then we define the unit u in H^* by

$$u(1_{\mathbb{K}}) \quad = \quad \sum_{i=1}^{n} k_i h_i^*. \tag{6.8}$$

The unit extends to all of $\mathbb{K}$ by linearity. Suppose that $u_H(1_K) = \sum_{i=1}^{n} k_i h_i$. Then the counit ϵ in H^* is defined by

$$\epsilon(h_i^*) \quad = \quad k_i. \tag{6.9}$$

Finally, if ρ_{ij} satisfies $S_H(h_i) = \sum_{j=1}^{n} \rho_{ij} h_j$, then the antipode S in H^* is given by

$$S(h_i^*) \quad = \quad \sum_{j=1}^{n} \rho_{ji} h_j^*. \tag{6.10}$$

The reader can verify that these definitions satisfy the definitions of Section 6.1 for a Hopf algebra. Figure 6.4 contains the algorithm HopfDual for constructing the dual of a finite dimensional Hopf algebra. The algorithm follows the equations directly to create a matrix algebra isomorphic to the dual of a finite dimensional Hopf algebra H. The function HopfAlgebra constructs a Hopf algebra object with appropriate implementations of maps for comultiplication, unit, counit, and antipode (multiplication is matrix multiplication) from the matrices passed as parameters. The implementation of HopfAlgebra is straightforward and is not included in this chapter.

Theorem 6.2.2. *The encoding neutral time complexity of* HopfDual *is*

$$\begin{aligned} T_{\text{HopfDual}} \quad = \quad \Theta\Big(& d\,t(\Delta_H) + d\,t(\text{NormalForm}_{H\otimes H}) + d^2\,t(*_H) + \\ & d^2\,t(\text{NormalForm}_H) + d\,t(\epsilon_H) + t(u_H) + \\ & t(S_H) + t(\text{HopfAlgebra})\Big), \end{aligned}$$

and the encoding neutral space complexity is

$$S_{\text{HopfDual}}(d) \quad = \quad \Theta\Big(d^3\,s(k)\Big),$$

where $s(k)$ is the maximum size of an element in $\mathbb{K}$ encountered during the execution of HopfDual.

Proof. Line 6 contains one execution of comultiplication on an element of H. The line is executed d times giving the $d\,t(\Delta_H)$ term. Line 7 requires `coproduct` to be in normal form, making a

HopfDual(H) Calculate the Hopf dual of H.

INPUT: A finite dimensional $\mathbb{K}$-Hopf algebra H.
OUTPUT: The Hopf dual H^*.

```
1  𝕂 ← H.field
2  h ← H.basis
3  d ← H.dimension
4  ▷ Compute the multiplication map
5  for t ← 1 to d
6      do coproduct ← Comultiply(h[t])
7          Define dualMult[r,t,s] by coproduct = Σ_{r,s=1}^{d} dualMult[r,t,s] * (h[r] ⊗ h[s])
8  ▷ Compute the comultiplication map
9  for s,t ← 1 to d
10     do product ← h[s] * h[t]
11         Define dualComult[r,s,t] by product = Σ_{r=1}^{d} dualComult[r,s,t] * h[r]
12 ▷ Compute the unit
13 for r ← 1 to d
14
15     do dualUnit[r] ← Counit(h[r])
16 ▷ Compute the counit
17 unit ← Unit(1_𝕂)
18 Define dualCounit[r] by unit = Σ_{r=1}^{d} dualCounit[r] * h[r]
19 ▷ Compute the antipode map
20 for r ← 1 to d
21     do antipode ← Antipode(h[r])
22         Define dualAntipode[s,r] by antipode = Σ_{s=1}^{d} dualAntipode[s,r] * h[s]
23 return HopfAlgebra(𝕂, dualMult, dualComult, dualUnit, dualCounit, dualAntipode)
```

Figure 6.4: Encoding neutral algorithm for computing the dual of a finite Hopf algebra

call to NormalForm$_{H\otimes H}$) necessary. The remainder of the cost for line 7 is d^2 element accesses and assignment, all of which have a cost of zero. Line 7 is executed d times, producing the term $d\,t(\text{NormalForm}_{H\otimes H})$. Line 10 contains a single execution of $*_H$. The line is executed d^2 times, giving the terms $d^2\,t(*_H)$ and $d^2\,t(\text{NormalForm}_H)$. Line 14 contains one application of ϵ_H and is executed d times. Hence, the line contributes a term of $d\,t(\epsilon_H)$. Line 18 contains one application of u_H and is executed one time, contributing a term of $t(u_H)$. Line 21 contains one application of S_H and is executed d times. The term $d\,t(S)$ is the contribution. Line 23 contains one call to HopfAlgebra to construct a Hopf algebra from the given matrices. The construction of the appropriate maps has non-trivial cost, so the term $t(\text{HopfAlgebra})$ is included.

The remainder of the lines are simple statements or assignments having negligible cost. The time complexity follows.

The space complexity is obtained by noting that `dualMult` and `dualComult` are triply indexed arrays with each index in the range $[1..d]$. Hence, the arrays store d^3 elements in $\mathbb{K}$. The remaining arrays each store d elements in $\mathbb{K}$. The overhead required by HopfAlgebra is constant, leading to the encoding neutral space complexity stated. □

6.2.3 Hatchets

The left and right hatchet actions are a central part of the definition of Drinfel'd double multiplication. Recall that the left hatchet $\rightharpoonup: A \otimes A^* \to A^*$ is defined as

$$h \rightharpoonup f \quad = \quad \sum \langle f_{(2)}, h\rangle f_{(1)}$$

for $h \in A$, $f \in A^*$, and $\Delta(f) = \sum f_{(1)} \otimes f_{(2)}$. As shown previously, the result of a comultiplication is a sum of tensor products. The reader may verify that the hatchet actions are linear.

Figure 6.5 contains the algorithm LeftHatchet for computing the left hatchet operation on $h \in H$ and $f \in H^*$. In our implementation of the left hatchet action, we apply comultiplication to f, using Equation (6.7) extended by linearity (line 4). The coefficient matrix α in line 6 is readily obtained in most encodings of finite dimensional algebras, such as matrix algebras, group algebras, or finite dimensional path algebras. The coefficient matrix β in line 7 is obtained by using the NormalForm function described for tensor products of algebras and then returning the coefficients for each basis element $b_i^* \otimes b_j^*$, where $1 \le i, j \le d$. Lines 8–10 compute the result using the linearity properties of the hatchet. We note that since h and `coproduct` are written in terms of basis

LeftHatchet(H, H^*, h, f) Compute $h \rightharpoonup f$.

INPUT: A finite dimensional Hopf algebra H, its Hopf dual H^*, $h \in H$, $f \in H^*$.
OUTPUT: $h \rightharpoonup f$.

```
1   b ← H.basis
2   b* ← H*.basis
3   d ← H.dimension
4   coproduct ← Δ(f)
5   result ← 0_{H*}
6   Define α by h = Σ_{i=1}^{d} α[i] * b[i]
7   Define β by coproduct = Σ_{i,j=1}^{d} β[i,j] * (b*[i] ⊗ b*[j])
8   for j ← 1 to d
9       do for i ← 1 to d
10              do result ← result + (α[j] * β[i,j]) * b*[i]
11  return result
```

Figure 6.5: An encoding neutral algorithm for computing $h \rightharpoonup f$.

elements in lines 6 and 7, the only possible non-zero contributions are when the second component of a basis element for `coproduct` is the dual of a basis element for h. This implies that only the coefficients $\alpha[\mathtt{j}] * \beta[\mathtt{i},\mathtt{j}]$ need to be computed, for $1 \leq i, j \leq d$, since those are the coefficients of those terms where the basis elements for `coproduct` and h are dual. The right hatchet action is implemented similarly and requires only switching the part of the comultiplied f applied to h and the order in which the parameters appear.

Theorem 6.2.3. *The encoding neutral time complexity of* LeftHatchet *is*

$$T_{\text{LeftHatchet}}(d) = \Theta\Big(d^2(t(+_{H^*}) + t(*_{\mathbb{K}}) + t(*_{\mathbb{K},H^*})) + t(\Delta_{H^*}) + t(\text{NormalForm}_H) + t(\text{NormalForm}_{H^*\otimes H^*})\Big).$$

Proof. There is one comultiplication of an element in H^* on line 4. Line 6 requires a call to NormalForm_H. There is one computation of $\text{NormalForm}_{H^*\otimes H^*}$ on line 7. Line 10 contains an addition in H^*, a multiplication in $\mathbb{K}$, and a multiplication of an element in $\mathbb{K}$ times an element in H^*. Clearly, line 10 is executed d^2 times, leading to the time complexity stated. □

6.2.4 Computing an Isomorphic Matrix Algebra

We now focus on the central algorithm, computing a matrix algebra isomorphic to the Drinfel'd double $D(H)$ of a finite dimensional Hopf algebra H. We compute a matrix algebra because algorithms currently implemented assume input in matrix algebra form for performing such computations as the Jacobson radical and for finding the Wedderburn decomposition of semisimple algebras. We are more interested in the algebra structure of $D(H)$ at this point, so the maps for comultiplication, counit, and antipode in $D(H)$ are not constructed explicitly as matrices. Instead, these operations are constructed in a straightforward manner from the operations for H and H^* as described in Equations (6.3–6.5) and returned as functions in GAP.

Recall Equation (6.1) for multiplication in $D(H)$:

$$(f \bowtie h)(f' \bowtie h') = \sum f((h_{(1)} \rightharpoonup f') \leftharpoonup S^{-1}(h_{(3)})) \bowtie h_{(2)}h',$$

where $f \in H^*$, $h \in H$. In this equation, we have $h_{(1)}$, $h_{(2)}$, and $h_{(3)}$ given by

$$\Delta^2(h) \quad = \quad \sum h_{(1)} \otimes h_{(2)} \otimes h_{(3)}.$$

Each summand is the tensor product of three elements in H. As we did with the hatchets, we apply the NormalForm$_{H\otimes H\otimes H}$ function to transform the resulting coproduct into terms of basis elements in $H \otimes H \otimes H$.

To compute the product in $D(H)$, we loop over each triple $(h_{(1)}, h_{(2)}, h_{(3)})$, substituting the pieces in their proper location in the definition. In our algorithm shown in Figure 6.6, lines 9 and 10 perform the comultiplications and calculation of the triples in terms of basis elements. The variable `total` on line 12 collects the result of the multiplication as each triple is looped over and used in the definition.

Lines 15–22 compute the contribution for the Hopf element, denoted by variable `hopfPart` in the algorithm, the left hatchet, denoted `leftHatchet`, and right hatchet, denoted `rightHatchet`, that each triple contributes to the product. We short circuit the evaluation by noting that if any of `hopfPart`, `leftHatchet`, or `rightHatchet` is zero, then the whole contribution from that triple is zero, and no further computation is needed. We chose to compute the Hopf element first, because multiplying two elements is often more efficient than the comultiplication overhead required to compute the hatchets.

Since we are computing a basis for a matrix algebra isomorphic to $D(H)$, we loop over each basis element in H and $H^{*^{cop}}$, to multiply each basis element in $D(H)$ and compute the structure

DRINFEL'D-DOUBLE(H) Compute $D(H)$.

INPUT: A finite dimensional $\mathbb{K}$-Hopf algebra H.
OUTPUT: The Drinfel'd double $D(H)$.

1 ▷ `D` will be the regular matrix representation of $D(H)$
2 $b \leftarrow H$.`basis`
3 $d \leftarrow |b|$
4 $H^* \leftarrow$ HOPFDUAL(H)
5 ▷ b^* is the dual basis for b
6 $b^* \leftarrow H^*$.`basis`
7 ▷ Compute $({b_j}^* \bowtie b_i)({b_l}^* \bowtie b_k)$
8 **for** `i` $\leftarrow 1$ **to** d
9 **do** `c` $\leftarrow \Delta^2(b[\mathtt{i}])$
10 Let α be defined by `c` $= \sum_{x,y,z=1}^{d} \alpha[x,y,z](b_x \otimes b_y \otimes b_z)$.
11 **for** `j,k,l` $\leftarrow 1$ **to** d
12 **do** `total` $\leftarrow 0_{H^*} \bowtie 0_H$
13 ▷ Go over each $b_m \otimes b_n \otimes b_o$
14 **for** `m,n,o` $\leftarrow 1$ **to** d
15 **do if** $\alpha[\mathtt{m},\mathtt{n},\mathtt{o}] \neq 0_{\mathbb{K}}$
16 **then** `hopfPart` $\leftarrow b[\mathtt{n}] * b[\mathtt{k}]$
17 **if** `hopfPart` $\neq 0_H$
18 **then** `leftHatchet` $\leftarrow$ LEFTHATCHET($b[\mathtt{m}], b^*[\mathtt{l}]$)
19 **if** `leftHatchet` $\neq 0_{H^*}$
20 **then** `rightHatchet` $\leftarrow$ RIGHTHATCHET(`leftHatchet`, $S^{-1}(b[\mathtt{o}])$)
21 `dualPart` $\leftarrow b^*[\mathtt{k}] *$ `rightHatchet`
22 `total` $\leftarrow$ `total` $+ \alpha[\mathtt{m},\mathtt{n},\mathtt{o}]$(`dualPart` $\bowtie$ `hopfPart`)
23 ▷ Product is computed. Now compute appropriate matrix entries.
24 Assign entries in D based on i, j, k, and l and finding the normal form of `total` to obtain the regular representation. The array D encodes d^2 $d^2 \times d^2$ matrices in $\mathbb{K}$.
25 **return** ALGEBRA(`D`)

Figure 6.6: The basic Drinfel'd double algorithm.

constants from the resulting product. Care must be taken in the order in which the basis elements in H and $H^{*^{cop}}$ are processed. Let d be the dimension of H (and H^*). Fix a basis element $b_i \in H$. Then, b_i is a part of every basis element in $D(H)$ of the form $b_j^* \bowtie b_i$ for $1 \leq j \leq d$. We can perform the required comultiplication for b_i once, and use it in the definition of multiplication for each basis element in $D(H)$ that contains b_i. If we loop over the dual basis elements, the comultiplication of basis elements in H may be done many times, and since comultiplication is generally expensive, this should be avoided. The outermost loop on line 8 is responsible for processing the basis elements in H properly.

The loop on line 11 is used to compute each product of basis elements $(b_j^* \bowtie b_i)(b_l^* \bowtie b_k)$ in $D(H)$. On line 24, the final product in `total` is then used to compute and store in table D the structure constants resulting from `total`.

The analysis of DRINFEL'D-DOUBLE provides initial insight into the complexity of constructing a matrix algebra isomorphic to $D(H)$. Currently, due to relatively little information known regarding Hopf algebras, we make the assumption that in the worst case, an input Hopf algebra H may cause DRINFEL'D-DOUBLE to execute every possible statement; that is, every arithmetic computation in DRINFEL'D-DOUBLE gives non-zero results, so no short circuiting of computation takes place. In practice, this situation has not been observed, and we have been unable to produce such a worst case input. We elaborate on our observations in Section 6.6.

Theorem 6.2.4. *Assume that every possible statement of* DRINFEL'D-DOUBLE *is executed for a given finite dimensional Hopf algebra H of dimension d. Then its encoding neutral time complexity when given H as input is*

$$\begin{aligned} T_{\text{DRINFEL'D-DOUBLE}}(d) \quad = \quad & O\Big(d^7\, t(*_H) + d^7\, t(*_{H^*}) + d^7\, t(\rightharpoonup_{H,H^*}) + d^7\, t(\leftharpoonup_{H^*,H}) + \\ & d^7\, t(S_H^{-1}) + d^7\, t(+_{H^* \bowtie H}) + d^7\, t(*_{\mathbb{K},H^* \bowtie H}) + \\ & d^4\, t(\text{NORMALFORM}_{H^* \bowtie H}) + d\, t(\text{NORMALFORM}_{H \otimes H \otimes H}) + \\ & d\, t(\Delta_H^2) + t(\text{HOPFDUAL}_H)\Big), \end{aligned}$$

and its encoding neutral space complexity is

$$\begin{aligned} S_{\text{DRINFEL'D-DOUBLE}}(d) \quad = \quad & \Theta\Big(d^6\, s(k) + s(H^*) + \max(s(h \otimes h \otimes h), s(h^* \bowtie h) + \\ & s(h) + s(h^*))\Big), \end{aligned}$$

where, during the execution of DRINFEL'D-DOUBLE, $k \in \mathbb{K}$, $h \otimes h \otimes h \in H \otimes H \otimes H$, $h^* \bowtie h \in H^* \bowtie H$, $h^* \in H$, *and* $h \in H$ *are the largest elements encountered in their respective domains.*

Proof. Line 4 constructs the dual of our input Hopf algebra H using HOPFDUAL. This construction occurs once and contributes a cost of $t(\text{HOPFDUAL})$.

Line 9 computes $\Delta^2(b)$ for some basis element $b \in H$. We have chosen to make Δ_H^2 an operation different from Δ_H because the computation involved, while similar, is generally more expensive than two calls to Δ_H. This is easily seen by the fact that the size of the output of Δ_H is bounded by d^2 terms when the element is put into normal form, but the output of Δ_H^2 is bounded by d^3 terms when the element is put into normal form. Line 9 is executed d times for any input, contributing a cost of $dt(\Delta_H^2)$.

Line 10 requires a call to $\text{NORMALFORM}_{H \otimes H \otimes H}$. This occurs d times for any input, contributing a cost of $dt(\text{NORMALFORM}_{H \otimes H \otimes H})$. Line 16 executes one multiplication $*_H$ and is executed d^7 times given our worst case assumption. Thus, we have a contribution of $d^7 t(*_H)$ to the complexity. Line 18 executes one left hatchet operation $\rightharpoonup_{H,H^*}$ and is executed d^7 times given our worst case assumption. This contributes $d^7 t(\rightharpoonup_{H,H^*})$ to the complexity. Line 20 executes one right hatchet operation $\leftharpoonup_{H^*,H}$ and one inverse antipode operation S_H^{-1}. The line is executed d^7 times with our worst case assumption. Hence, we obtain a $d^7 t(\leftharpoonup_{H^*,H}) + d^7 t(S_H^{-1})$ contribution to the complexity. Line 21 executes one multiplication $*_{H^*}$ and is executed d^7 given our worst case assumption. Hence, $d^7 t(*_{H^*})$ is contributed to the complexity. Line 22 performs one addition operation $+_{H^* \bowtie H}$ and a scalar multiplication operation $*_{\mathbb{K}, H^* \bowtie H}$. This line is executed d^7 times given our worst case assumption for a contribution of $d^7 t(+_{H^* \bowtie H}) + d^7 t(*_{\mathbb{K}, H^* \bowtie H})$ to the complexity. Line 24 requires a call to $\text{NORMALFORM}_{H^* \bowtie H}$. This is performed using the NORMALFORM defined for tensor products of algebras using $H^* \otimes H$ as the underlying vector space. The line is executed d^4 times for any input, contributing $d^4 t(\text{NORMALFORM}_{H^* \bowtie H})$ to the complexity.

The remainder of the lines perform simple statements, assignments, or accesses whose costs are negligible. The encoding neutral time complexity follows directly by totaling our observations.

The space complexity comes from the fact that D is an array storing d^6 elements in $\mathbb{K}$. The variable c briefly stores an element in $H \otimes H \otimes H$, which is quickly converted to an array containing d^3 elements in $\mathbb{K}$. The variable `total` stores an element in $H \bowtie H$, but it never has to at the same time c stores an element. During the time that `total` holds values, the variable `hopfPart` holds an element in H, and the variables `leftHatchet`, `rightHatchet`, and `dualPart` holds an element in

H^*. Since H^* is constructed during the execution of DRINFEL'D-DOUBLE, its space cost is included. The result follows immediately. □

Of great importance is the space complexity of the Drinfel'd double algorithm, due to the encoding of the resulting algebra as a matrix algebra, contributing the term $\Theta(d^6\, s(k))$. Suppose $d = 15$; then $\approx 1.14 \times 10^7$ elements in $\mathbb{K}$ are needed to encode $D(H)$. If each field element requires just 4 bytes of storage, a common assumption, then approximately 43.5 megabytes of RAM is needed to store $D(H)$. In practice, the storage requirements are much higher because of additional space overhead of computational algebra systems such as GAP.

6.3 Caching

The basic algorithm presented in Figure 6.6 included the cost of executing unnecessary comultiplications, as well as short circuiting the evaluation of the contribution a triple makes in the computation of Drinfel'd double multiplication. These initial considerations were enough to construct $D(H)$ for H of relatively small dimension (≤ 8) in a reasonable amount of time (≤ 4 hours). However, H of larger dimension (e.g., 12) proved to take an unreasonable amount of time (> 2 weeks on one modern workstation).

Notice that the multiplication done on line 16 is performed only between basis elements in H. Since there are only d basis elements, there are only d^2 unique multiplications during the execution of DRINFEL'D-DOUBLE. However, our worst case complexity implies that $O(d^7)$ multiplications may be performed, resulting in $O(d^7 - d^2) = O(d^7)$ repeated computations! A similar observation can be made about the left hatchet operation on line 18.

For the right hatchet operation on line 20, we observe that there are d^2 possible results from the previous left hatchet operation, and d possible results from applying S^{-1} to a basis element in H. Thus, there are d^3 possible results from the right hatchet operation during the execution of DRINFEL'D-DOUBLE (in the worst case), resulting in $O(d^7)$ repeated computations.

To avoid the repeated computations, we implemented a table for each of the multiplication, left hatchet, and right hatchet operations. Tables are indexed by the elements involved in the computation. Each table is initialized with unbound values, allowing us to check if a computation has or has not already been performed. Figure 6.7 presents pseudocode for caching the left hatchet operation.

CACHED-LEFTHATCHET(h, f) Compute $h \rightharpoonup f$ with cached results.

INPUT: A Hopf algebra (basis) element h and a dual Hopf algebra (basis) element f.
OUTPUT: $h \rightharpoonup f$.

```
1   ▷ leftHatchetTable is a global variable storing the caching array.
2   if leftHatchetTable[h, f] contains a value
3     then return leftHatchetTable[h, f]
4     else leftHatchetTable[h, f] ← LEFTHATCHET(h, f)
5          return leftHatchetTable[h, f]
```

Figure 6.7: Pseudocode for caching left hatchet operations.

When an operation is about to be performed, the table is checked to see if the selected operation has already been performed on a pair of elements. If it has not, the operation is computed, and the result is stored in the table. If it has, the result is retrieved from the table and used in the remaining computations.

Theorem 6.3.1. *Assume that every possible statement of* DRINFEL'D-DOUBLE *is executed for a given finite dimensional Hopf algebra H of dimension d. Then its encoding neutral time complexity, with cached multiplication (in H) and hatchet operations, when given H as input is*

$$\begin{aligned} T_{\text{DRINFEL'D-DOUBLE}}(d) &= O\Big(d^7\, t(*_{H^*}) + d^7\, t(+_{H^* \bowtie H}) + d^7\, t(*_{\mathbb{K}, H^* \bowtie H}) + \\ &\quad d^4\, t(\text{NORMALFORM}_{H^* \bowtie H}) + d^3\, t(\leftharpoonup_{H^*, H}) + d^3\, t(S_H^{-1}) + \\ &\quad d^2\, t(*_H) + d^2\, t(\rightharpoonup_{H, H^*}) + d\, t(\text{NORMALFORM}_{H \otimes H \otimes H}) + \\ &\quad d\, t(\Delta_H^2) + t(\text{HOPFDUAL}_H)\Big) \end{aligned}$$

and the encoding neutral space complexity is

$$\begin{aligned} S_{\text{DRINFEL'D-DOUBLE}}(d) &= \Theta\Big(d^6\, s(k) + s(H^*) + \max(s(h \otimes h \otimes h), s(h^* \bowtie h)) + \\ &\quad d^2\, s(h) + d^3\, s(h^*)\Big), \end{aligned}$$

using the notation from Theorem 6.2.4.

Proof. The upper bound on the number of times $*_H$, $\rightharpoonup_{H,H^*}$, and $\leftharpoonup_{H^*,H}$ is given by the number of unique results there can be for each operation as described above. The complexity of the computation

of S_H^{-1} can also be reduced by using the input basis element as an index into caching the computation on line 20 instead of the result of S_H^{-1}, which we use in our implementation.

The remainder of the time complexity follows directly from Theorem 6.2.4.

The change in the space complexity is primarily due to caching, which requires that an element be stored for each unique computation of multiplication, left hatchet, and right hatchet operations. These occur d^2, d^2, and d^3 times respectively. The results of multiplication is an element in H, and the results of the hatchet operations are in H^*. The caching tables are stored throughout the execution of DRINFEL'D-DOUBLE, so the additional space required by caching is additive. The remainder of the space complexity follows from Theorem 6.2.4. □

These modifications to the basic algorithm improved performance dramatically as the experimental results in Section 6.6 demonstrate. The space requirements are increased for caching, with the additional non-constant terms $\Theta(d^2\, s(h) + d^3\, s(h^*))$. In practice, the dominating space cost is that of encoding $D(H)$ as a matrix algebra. Hence, caching operations is always worthwhile for this construction.

6.4 Covering Algebras

To test our algorithm, we constructed a class of Hopf algebras called *covering algebras* defined by Green [47]. Covering algebras are instances of path algebras (see Section 4.3). We review how to construct a covering algebra A and finite quotients of A that are Hopf algebras.

Let G be a finite group, and $\mathbb{K}$ a field. Let n be the number of *levels* in the covering algebra. For each level i, define a *weight* $w_i \in G$. Define the directed graph Γ with the vertex set $\Gamma_0 = \{v_g \mid g \in G\}$ and edge set $\Gamma_1 = \{a_{i,g} \mid 1 \leq i \leq n \text{ and } g \in G\}$ where $a_{i,g}$ is the edge in level i from v_g to v_{gw_i}.

Example 6.4.1. Let G be the cyclic group of order 3 C_3, written additively (i.e., multiplication in G is addition in C_3) and let $n = 2$. Let $w_1 = 1$ and $w_2 = 2$. The directed graph Γ is given in Figure 6.8.

To define the coalgebra structure of $\mathbb{K}\Gamma$, we need additional group homomorphisms $f_i : G \to \mathbb{K}^\times$, for $1 \leq i \leq n$ and where $\mathbb{K}^\times$ denotes the nonzero elements of $\mathbb{K}$ viewed as a multiplicative group.

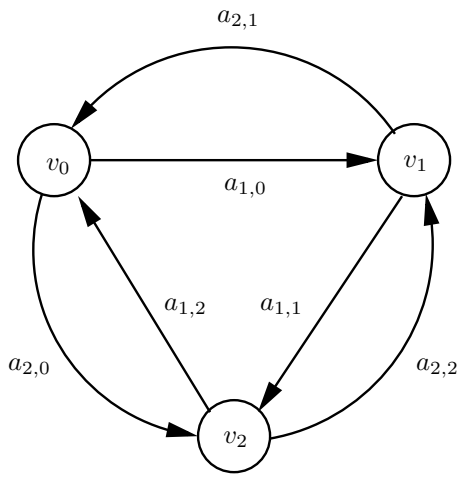


Figure 6.8: An example graph defining a covering algebra.

The coalgebra structure for covering algebras is given by the following definitions on the generators:

$$\begin{aligned}
\Delta v_g &= \sum_{h\in G} v_{h^{-1}g} \otimes v_h \\
\Delta a_{i,g} &= \sum_{h\in G} \left(v_{h^{-1}g} \otimes a_h + f_i(h) \cdot a_{i,h^{-1}g} \otimes v_h \right) \\
\epsilon(v_g) &= \begin{cases} 1 & \text{if } g = 1_G \\ 0 & \text{otherwise} \end{cases} \\
\epsilon(a_{i,g}) &= 0 \\
S(v_g) &= v_{g^{-1}} \\
S(a_{i,g}) &= -f_i(gw_i) \cdot a_{i,(gw_i)^{-1}}
\end{aligned}$$

The remaining requirement is to make finite dimensional quotients of covering algebras. We use two methods to construct a finite-dimensional quotient. The first method requires the homomorphisms f_i to be non-trivial and to satisfy

$$f_i(w_j) = f_j(w_i),$$

for $1 \leq i, j \leq n$. Let l_i denote the order of $f_i(w_i)$. Let I be the ideal in $\mathbb{K}\Gamma$ generated by all relations

of the form

$$\left(a_{j,g}a_{i,gw_j}\right) - f_i(w_j)\left(a_{i,g}a_{j,gw_i}\right)$$

and

$$a_{i,g}a_{i,gw_i}\cdots a_{i,gw_i^{l_i-2}}a_{i,gw_i^{l_i-1}}$$

where $1 \leq i, j \leq n$, $i \neq j$, and $g \in G$. The quotient $\mathbb{K}\Gamma/I$ is a Hopf algebra with the operations induced by those for $\mathbb{K}\Gamma$. The order of the antipode S for algebras constructed with the first method is $\text{lcm}(l_1, \ldots, l_n)$, implying that $S^{-1} = S^{\text{lcm}(l_1,\ldots,l_n)-1}$.

The second method requires that the characteristic of $\mathbb{K}$ be $p > 0$ and each of the homomorphisms f_i to be the trivial homomorphism; that is,

$$f_i(g) \quad = \quad 1_{\mathbb{K}}, \text{ for all } g \in G,$$

where $1 \leq i \leq n$. Suppose l is a positive integer such that the binomial coefficients $\binom{l}{k}$ are divisible by p for all $1 \leq k \leq l-1$. For example, $l = p^2$ has this property. Let I be the ideal in $\mathbb{K}\Gamma$ generated by all relations of the form

$$a_{j,g}a_{i,gw_j} - a_{i,g}a_{j,gw_i},$$

and

$$a_{i,g}a_{i,gw_i}\cdots a_{i,gw_i^{l-2}}a_{i,gw_i^{l-1}},$$

where $g \in G$ and $1 \leq i, j \leq n$. Again, $\mathbb{K}\Gamma/I$ is a Hopf algebra with operations induced by those for $\mathbb{K}\Gamma$. The order of the antipode S for algebras constructed with the second method is 2, so $S^{-1} = S$.

6.4.1 Implementation of Covering Algebras

To create covering algebras, we first implemented generic path algebras in GAP. The implementation includes all standard arithmetic operations for elements in path algebras. We support quotients of path algebras by interfacing GAP to Opal [50], a software package for computing noncommutative Gröbner bases for algebras encoded as quotients of path algebras. The interface is used to compute a Gröbner basis of an ideal and a basis for the quotient of the path algebra by the ideal, if the quotient is finite dimensional. We use the resulting Gröbner basis to perform total reduction of elements in

```
gap> cov := Covering(GF(3), 1, CyclicGroup(2),
>                    [ (1,2) ], [ 3 ] );
PathAlgebra( GF(3),
    Quiver( ["v{0}", "v{1}"],
            [ [ v{0}, v{1}, "a{1,0}" ],
              [ v{1}, v{0}, "a{1,1}" ] ] ) )
    / [ a{1,0}*a{1,1}*a{1,0},
        a{1,1}*a{1,0}*a{1,1} ]
gap> Comultiply(cov.3);
(v{0})@(a{1,0})+(v{1})@(a{1,1})+(a{1,0})@(v{0})\
+(a{1,1})@(v{1})
```

Figure 6.9: GAP commands to construct a covering algebra and comultiply the third generator ($a_{1,0}$).

the quotient. Total reduction uses our implementation of the Knuth-Morris-Pratt pattern matching algorithm [19] in GAP.

Next, we implemented functions to construct covering algebras as quotients of path algebras. These are straightforward implementations of the definitions presented in Section 6.4.

The comultiplication, counit, and antipode maps are created by functions that use the parameters passed to create the covering algebras. The maps are returned as GAP functions that replace the generators with the appropriate output as well as extend the definition of the maps on the generators to all elements in the covering algebra. The comultiplication map uses our implementation of tensor products to encode its output. Sample GAP commands to construct a covering algebra and comultiply an element with resulting output is shown in Figure 6.9.

6.5 Group Algebras

Another important class of Hopf algebras is group algebras. Recall that a group algebra $\mathbb{K}G$ is the set of linear combinations of elements of G with coefficients from $\mathbb{K}$ where multiplication is group multiplication distributed across addition (see Section 4.2). This becomes a Hopf algebra with the

following definitions for comultiplication, counit, and antipode:

$$\begin{aligned} \Delta(g) &= g \otimes g \\ \epsilon(g) &= 1 \\ S(g) &= g^{-1}, \end{aligned}$$

for all $g \in G$ $\mathbb{K}G$ and extended to all of $\mathbb{K}G$ by linearity. In the remainder of this section, we use the term group algebra to refer to a group algebra with the given Hopf algebra structure.

The Drinfel'd double can be constructed for group algebras much more efficiently than for Hopf algebras in general. Additionally, Gould [46] and Witherspoon [99] investigate several properties of Drinfel'd doubles of group algebras using group theoretic methods. These methods should provide efficient computational means for studying Drinfel'd doubles of group algebras.

The dual of a group algebra can be easily encoded in terms of group element operations. Let the elements of G be indexed from $[1..d]$, and use them as the natural basis for $\mathbb{K}G$. Use the dual basis for $\mathbb{K}G^*$, where $g_i^*(g_j) = 1$ if $i = j$ and 0 if $i \neq j$. From Equation (6.6), multiplication in $\mathbb{K}G^*$ is

$$g_r^* g_s^* = \sum_{t=1}^{d} \gamma_{trs} g_t^*. \tag{6.11}$$

By the definition of comultiplication for group algebras,

$$\gamma_{ijk} = \begin{cases} 1 & \text{if } i = j = k; \\ 0 & \text{otherwise.} \end{cases}$$

Hence, $g_r^* g_r^* = g_r^*$ and $g_r^* g_s^* = 0$ for $r \neq s$.

From Equation (6.7), comultiplication in $\mathbb{K}G^*$ is

$$\Delta(g_r^*) = \sum_{s,t=1}^{d} \alpha_{str} g_s^* \otimes g_t^*.$$

The coefficients α_{str} come from $g_i g_j = \sum_{k=1}^{d} \alpha_{ijk} g_k$. In a group algebra, $g_i g_j = g_k$ for some basis element g_k. So, this implies for comultiplication in $\mathbb{K}G^*$ that

$$\begin{aligned} \Delta(g_r^*) &= \sum_{g_s g_t = g_r} g_s^* \otimes g_t^* \\ &= \sum_{t=1}^{d} (g_r g_t^{-1})^* \otimes g_t^* \\ &= \sum_{s=1}^{d} g_s^* \otimes (g_s^{-1} g_r)^*. \end{aligned}$$

The unit for $\mathbb{K}G^*$ follows directly from Equation (6.8) and the definition of the counit for $\mathbb{K}G$:

$$u(1_{\mathbb{K}}) = \sum_{r=1}^{d} g_r^*.$$

It is easily seen that $\sum_{r=1}^{d} g_r^* = 1_{\mathbb{K}G^*}$.

The counit for $\mathbb{K}G^*$ follows immediately from Equation (6.9) and the definition of the unit for $\mathbb{K}G$:

$$\epsilon(g_r^*) = \begin{cases} 1 & \text{if } g_r = 1_G; \\ 0 & \text{otherwise.} \end{cases}$$

Finally, the antipode for $\mathbb{K}G^*$ follows from Equation (6.10) and the definition of the antipode for group algebras:

$$\begin{aligned} S(g_r^*) &= \sum_{s=1}^{d} \rho_{sr} g_s^* \\ &= (g_r^{-1})^*. \end{aligned}$$

To construct the Drinfel'd double, one only needs to substitute the appropriate expressions for $\mathbb{K}G$ and $\mathbb{K}G^*$ into Equations (6.1–6.5). As we did with DRINFEL'D-DOUBLE, we focus on the multiplication of basis elements and discuss the cost of constructing an isomorphic matrix algebra to study the algebra structure of the Drinfel'd double.

We derive an expression for $(g_r^* \bowtie g_s)(g_t^* \bowtie g_u)$. The multiplication defined in Equation (6.1) is simplified for group algebras due to the definition of comultiplication and the antipode,

$$\begin{aligned} (g_r^* \bowtie g_s)(g_t^* \bowtie g_u) &= \sum g_r^*((g_{s(1)} \rightharpoonup g_t^*) \leftharpoonup S^{-1}(g_{s(3)})) \bowtie g_{s(2)} g_u \\ &= \sum g_r^*((g_s \rightharpoonup g_t^*) \leftharpoonup g_s^{-1}) \bowtie g_s g_u. \end{aligned}$$

The calculation of $g_s \rightharpoonup g_t^*$ is

$$\begin{aligned} g_s \rightharpoonup g_t^* &= \sum \langle g_{t\,(2)}^*, g_s \rangle g_{t\,(1)}^* \\ &= \sum_{v=1}^{d} \langle g_v^*, g_s \rangle (g_t g_v^{-1})^* \\ &= (g_t g_s^{-1})^*. \end{aligned}$$

Using the result of $g_s \rightharpoonup g_t^*$ in the computation of $\leftharpoonup$, we calculate $(g_t g_s^{-1})^* \leftharpoonup g_s^{-1}$ obtaining

$$\begin{aligned}(g_t g_s^{-1})^* \leftharpoonup g_s^{-1} &= \sum \langle (g_t g_s^{-1})^*{}_{(1)}, g_s^{-1} \rangle (g_t g_s^{-1})^*{}_{(2)} \\ &= \sum_{v=1}^{d} \langle g_v^*, g_s^{-1} \rangle (g_v^{-1} g_t g_s^{-1})^* \\ &= (g_s^{-1^{-1}} g_t g_s^{-1})^* \\ &= (g_s g_t g_s^{-1})^*.\end{aligned}$$

Returning to the expression for multiplication,

$$\sum g_r^*((g_s \rightharpoonup g_t^*) \leftharpoonup g_s^{-1}) \bowtie g_s g_u \quad = \quad g_r^*(g_s g_t g_s^{-1})^* \bowtie g_s g_u.$$

So, $(g_r^* \bowtie g_s)(g_t^* \bowtie g_u) = g_r^*(g_s g_t g_s^{-1})^* \bowtie g_s g_u$, which is non-zero only when $g_r = g_s g_t g_s^{-1}$.

Theorem 6.5.1. *The encoding neutral time complexity for computing $D(\mathbb{K}G)$ is*

$$T_{\text{DRINFEL'D-DOUBLE}}(d) \quad = \quad O\Big(d^3\, t(*_G) + d\, t(\bar{}_G^1)\Big).$$

Proof. An isomorphic matrix algebra is readily obtained using a triply nested loop iterating over elements in G to compute only the non-zero matrix entries. If one makes the outermost loop iterate over g_s, only d inversions in G are needed. If the innermost loop iterates over g_u, then d^3 multiplications in G are performed by $g_s g_u$, and $2d^2$ multiplications are performed by $g_s g_t g_s^{-1}$, yielding the complexity above. □

The complexity can be improved to $O(d^2\, t(*_G) + d\, t(\bar{}_G^1))$ if caching is used on the $*_G$ operation.

6.6 Performance

We tested the performance of our implementation on a Sparc Ultra 30 running Solaris version 2.6 and containing one 296 MHz UltraSparc-II processor and 1 gigabyte of RAM. We used GAP version 3.4.4 compiled with EGCS version 1.0.2 (a modified version of the GCC compiler) using -O6 optimization. Finally, to compute Gröbner bases we used Opal prerelease version 0.02 compiled with G++ version 2.7.2 using no optimization.

Table 6.1 shows the parameters and resulting dimension for each of the algebras used for experimentation. We chose to use only quotients of covering algebras constructed by the second method

Table 6.1: Input algebras used in testing.

Case #	d	Field	Group	Weights	l
1	4	$\mathbb{F}_2$	C_2	$w_1 = 1$	2
2	6	$\mathbb{F}_3$	C_2	$w_1 = 1$	3
3	6	$\mathbb{F}_2$	C_3	$w_1 = 1$	2
4	8	$\mathbb{F}_2$	C_2	$w_1 = w_2 = 1$	2
5	9	$\mathbb{F}_3$	C_3	$w_1 = 1$	3
6	10	$\mathbb{F}_5$	C_2	$w_1 = 1$	5
7	12	$\mathbb{F}_3$	C_4	$w_1 = 1$	3

Table 6.2: Total user and system time used (in CPU seconds).

		Caching		
Case #	d	None	$\rightharpoonup, \leftharpoonup$	$*, \rightharpoonup, \leftharpoonup$
1	4	47.9	7.4	5.3
2	6	568.6	30.1	18.3
3	6	463.1	29.7	15.4
4	8	13230.1	395.9	118.3
5	9	23994.7	581.1	246.1
6	10	89440.7[1]	945.0	502.2
7	12	434292.8[1]	1840.5	697.6

to simplify the performance comparisons. Similar speedups are expected for quotients of covering algebras constructed using the first method as well.

Table 6.2 shows the total user and system time in CPU seconds used to compute the Drinfel'd double for a given algebra averaged over 5 runs. The *None* column gives the time for our original implementation without any caching of hatchet or multiplication operations. The column labeled "$\rightharpoonup, \leftharpoonup$" gives the time used by our implementation that caches hatchet operations. The column labeled "$*, \rightharpoonup, \leftharpoonup$" gives the time used by our implementation that caches multiplications and hatchet operations.

We see that caching improves the performance in every case and that caching multiplications and hatchet operations together performs the best. We estimate the rate of growth of the time complexity as a function of d from the experimental times as follows. First we hypothesize that the time complexity in each case has the form $T(d) = cd^k$, where c and k are constants to be estimated.

[1] Only one run used for timing.

Taking logarithms, we obtain $\ln T(d) = \ln c + k \ln d$. Using the `LinearRegression` package in *Mathematica*, we fit a function of the form $\ln c + k \ln d$ to each column of data. For the case of no caching, we obtain the estimate $\ln T(d) \approx -8.7213 + 8.64744 \ln d$. For the case of hatchet caching, we obtain the estimate $\ln T(d) \approx -6.04985 + 5.54805 \ln d$. For the case of hatchet and multiplication caching, we obtain the estimate $\ln T(d) \approx -5.68983 + 5.00096 \ln d$. Significant in these estimates is that the exponent k is steadily decreasing — 8.64744 to 5.54805 to 5.00096. Also significant is that the exponent of 8.64744 is inconsistent with the exponent 7 obtained in Theory 6.2.4. Several factors might be contributing to this inconsistency. The operations performed during the construction of the Drinfel'd double are not constant time operations; hence, their boolean complexity is affecting the exponent obtained by the experimental results. We also believe that the tremendous space requirements are contributing to a memory management overhead that causes the exponent of the observed time complexity to exceed the exponent of the theoretical time complexity. On the other hand, the exponents between 5 and 6 for the cached cases indicates the much greater practicality of these approaches.

6.7 Conclusions and Future Work

In this chapter, we have described the Drinfel'd double, a central construction included in HOPF. The algorithm used in HOPF to compute Drinfel'd doubles of finite-dimensional Hopf algebras was reported and analyzed. In addition, the GAP domains and operations to support our implementation of the Drinfel'd double were presented, including path algebras, tensor products, connection to the noncommutative Gröbner basis package Opal, construction of duals of Hopf algebras, and covering algebras. We are updating HOPF to GAP4 [42], enhancing our existing system in the process.

Empirical results demonstrating improvements in running time and in growth of running time for the Drinfel'd double algorithm were presented. The growth in space used by the output of the algorithm is shown to prevent computing the Drinfel'd double of moderate to large dimensional Hopf algebras. We are researching ways to avoid computing entire matrix encodings of algebras, such as using different encodings and Gröbner basis techniques. This research will allow the HOPF system to be used in the study of larger dimensional algebras.

Chapter 7

Module Encodings

In Chapter 4, we analyzed several methods for encoding algebras on a computer. In this chapter, we consider encodings schemes for right A-modules where A is a $\mathbb{K}$-algebra. The time and space complexity for each encoding is analyzed. We begin by describing a general encoding method in Section 7.1. In Section 7.2, we view an alternative method for encoding a right A-module where A is a path algebra.

7.1 $\mathbb{K}$-Representations

The representation theory of finite groups and associative algebras plays an important role in determining the structure of groups, algebras, and modules over them. Representation theory also leads to a straightforward method of encoding finite $\mathbb{K}$-dimensional A-modules. We refer the reader to several texts [21, 22, 23, 40, 77] for a detailed background on representation theory and its applications. The terminology used in this section is based on Nagao and Tsushima [77] Chapter 2.

Let A be a $\mathbb{K}$-algebra, not necessarily finite dimensional. A $\mathbb{K}$-*representation* of A is an algebra homomorphism $\rho : A \to \rho(A)$ such that $\rho(A) \subseteq M_n(\mathbb{K})$. Let M be a (right) A-module that is also a $\mathbb{K}$-vector space with $\mathbb{K}$-basis $\{b_1, b_2, \ldots, b_d\}$. Define the map $\rho : A \to M_d(\mathbb{K})$ by $\rho(a) = (\alpha_{ij})_{i,j=1}^{d}$ where $a \in A$ and $b_i a = \sum_j \alpha_{ij} b_j$. It is easily checked that ρ is a representation, which is called a *representation of A defined by M*. From the definition of ρ, an element m in M is viewed as a row $\mathbb{K}$-vector. The action of $a \in A$ on m is the product $m\rho(a)$.

Example 7.1.1. The regular matrix representation described in Section 3.2.3 is a representation defined by viewing A as an A-module.

Because ρ is an algebra homomorphism, defining ρ on a generating set of A is sufficient to define ρ for all elements in A.

Example 7.1.2. Let $G = C_3 = \{g_0, g_1, g_2\}$, the cyclic group of order 3. Multiplication in G is defined as $g_i g_j = g_{i+j}$, where $i + j$ is addition modulo 3. Let $\mathbb{K} = \mathbb{F}_3$. The group algebra $\mathbb{F}_3 C_3$ has a representation $\rho : \mathbb{F}_3 C_3 \to M_2(\mathbb{F}_3)$ defined by

$$\rho(g_1) \mapsto \begin{pmatrix} 1 & 1 \\ 0 & 1 \end{pmatrix}.$$

The encoding for a d dimensional (over $\mathbb{K}$) A-module M is evident. Module elements are encoded as row $\mathbb{K}$-vectors and the action of an algebra element is encoded as a $d \times d$ $\mathbb{K}$-matrix. If A is generated by n elements, n $d \times d$ matrices is sufficient to describe the action of A on M. The maximum number of $d \times d$ matrices needed to encode the action of A on M is d^2, when the representation maps A onto $M_d(\mathbb{K})$.

Addition of two module elements is just addition of two length d vectors, with an encoding neutral time complexity of $\Theta(dt(+_{\mathbb{K}}))$. The action of an algebra element on a module element is the multiplication of a $d \times d$ matrix with a length d vector. The encoding neutral time complexity is $\Theta(d^2(t(*_{\mathbb{K}}) + t(+_{\mathbb{K}})))$ using a standard implementation of multiplication. Scalar multiplication has a time complexity of $\Theta(dt(*_{\mathbb{K}}))$. The time complexity is summarized in Table 7.1 The space complexity is straightforward and summarized in Table 7.2.

7.2 Projective Presentations

In this section, we describe an encoding for A-modules which may be more efficient in some circumstances. The idea is to use the natural algebra multiplication to define the action and to encode module elements as A-vectors instead of $\mathbb{K}$-vectors. We use this encoding in Section 8.2 to construct endomorphism rings of modules over path algebras.

Throughout this section, let $\mathbb{K}$ be a field, let $\mathbb{K}\Gamma$ be a path algebra, let I be an ideal in $\mathbb{K}\Gamma$, and let $A = \mathbb{K}\Gamma/I$. We recall some results of Green [49] about the right Gröbner basis theory of right modules. We extend these results to use right Gröbner bases to compute with right A-modules.

Table 7.1: Worst case time complexity for $\mathbb{K}$-representation encoding.

Operation	Complexity
$+_M$	$\Theta(dt(+_{\mathbb{K}}))$
$*_{M,A}$	$\Theta(d^2(t(*_{\mathbb{K}}) + t(+_{\mathbb{K}})))$
$*_{\mathbb{K},M}$	$\Theta(dt(*_{\mathbb{K}}))$

Table 7.2: Worst case space complexity for $\mathbb{K}$-representation encoding.

Encoded	Complexity
$m \in M$	$\Theta(d\hat{s}\,(\tau_{\mathbb{K}}(m)))$
M with n A generators G	$\Theta(d^2 n\hat{s}\,(\tau_{\mathbb{K}}(G)) + s(\mathbb{K}))$

Projective right $\mathbb{K}\Gamma$-modules have an easily described structure. Define the set $v\mathbb{K}\Gamma$ by

$$v\mathbb{K}\Gamma = \left\{ \sum \alpha_p p \in \mathbb{K}\Gamma \mid \text{ all but a finite number of } \alpha_p = 0 \text{ and } s(p) = v \right\}.$$

Observe that $\mathbb{K}\Gamma = \oplus_{v \in V} v\mathbb{K}\Gamma$, hence every $v\mathbb{K}\Gamma$ is a projective right-$\mathbb{K}\Gamma$ module. The following theorem describes the structure of every projective right $\mathbb{K}\Gamma$-module.

Theorem 7.2.1. *Let $\mathbb{K}$ be a field, let $G = (V, A)$ be a quiver, let Γ be the set of paths in G, and let $\mathbb{K}\Gamma$ be a path algebra. If P is a projective right $\mathbb{K}\Gamma$-module, then $P = \oplus_{i \in \mathcal{I}} v(i)\mathbb{K}\Gamma$ where $\mathcal{I}$ is an index set, and $v : \mathcal{I} \to V$ is a set function.*

Proof. The proof appears in Green [49, Corollary 5.5]. □

We view elements in right projective $\mathbb{K}\Gamma$-modules as row vectors with components in $v(i)\mathbb{K}\Gamma$. A vector m is *right uniform* (or just *uniform*) if there exists a vertex v such that $t(m_i) = v$ or $m_i = 0$, for each component m_i in m. The target $t(m) = v$ for the right uniform vector m and is undefined if m is not right uniform.

Let B be the set of non-zero walks in Γ. We have already shown that B is a distinguished basis for $\mathbb{K}\Gamma$ in Section 4.3. Let M be a right $\mathbb{K}\Gamma$-module and let $\mathcal{M}$ be a $\mathbb{K}$-basis for M. The basis $\mathcal{M}$ is *coherent* if for all $m \in \mathcal{M}$ and all $b \in B$, $mb = 0$ or $mb \in \mathcal{M}$. Let $<$ be an admissible order on B (see Section 4.3.1) and let $\prec$ be a well-order on $\mathcal{M}$. Let m, m_1, m_2 be elements in $\mathcal{M}$ and b, b_1, b_2 be elements in B. The order $\prec$ is a *right admissible order* if $\prec$ satisfies the properties:

1. if $m_1 \prec m_2$, then $m_1 b \prec m_2 b$ whenever $m_1 b$ and $m_2 b$ are both nonzero;

2. if $b_1 < b_2$, then $mb_1 \prec mb_2$ whenever mb_1 and mb_2 are both nonzero.

If $\mathcal{M}$ is a coherent basis and $\prec$ is a right admissible order, then $(\mathcal{M}, \prec)$ is an *ordered basis* for M. Suppose $x \in M$ and $x \neq 0_M$. We may write x uniquely as $x = \sum_{i=1}^{r} \alpha_i m_i$ where each $a_i \in \mathbb{K} - \{0_{\mathbb{K}}\}$ and each $m_i \in \mathcal{M}$. The *tip* of x is $\mathrm{Tip}(x) = \max_{\prec}\{m_i\}$. If $X \subseteq M$, then its *tip set* is $\mathrm{Tip}(X) = \{\mathrm{Tip}(x) \mid x \in X \text{ and } x \neq 0_M\}$. The set of *nontips* of X is $\mathrm{Nontip}(X) = \mathcal{M} - \mathrm{Tip}(X)$. If N is a right submodule of M, then $\mathcal{G}$ is a *right Gröbner basis* of N with respect to $\prec$ if $\mathcal{G} \subset N$ and the right submodule of M generated by $\mathrm{Tip}(\mathcal{G})$ equals the right submodule of M generated by $\mathrm{Tip}(N)$.

Let $\mathcal{I}$ be an index set and let $v : \mathcal{I} \to V$ be a set function. Let $M = \oplus_{i\in\mathcal{I}} v(i)\mathbb{K}\Gamma$ be a projective right $\mathbb{K}\Gamma$-module and let N be a right submodule of M. The module M has an ordered basis [49]. We use right Gröbner bases to compute with quotient modules much as we use ideal-theoretic Gröbner bases to compute with quotient algebras. Elements in $U = M/N$ are written as elements in the span of $\mathrm{Nontip}(N)$.

Because M and N are both projective right $\mathbb{K}\Gamma$-modules, we obtain the following exact sequence:

$$\oplus_{j\in\mathcal{J}} w(j)\mathbb{K}\Gamma \xrightarrow{(f_{ji})} \oplus_{i\in\mathcal{I}} v(i)\mathbb{K}\Gamma \to U \to 0$$

where $\mathcal{I}, \mathcal{J}$ are index sets, $v : \mathcal{I} \to V$ and $w : \mathcal{J} \to V$ are set functions, and $f_{ji} \in \mathbb{K}\Gamma$. Each row f_j in the matrix (f_{ji}) is an element in $\oplus_{i\in\mathcal{I}} v(i)\mathbb{K}\Gamma$, each target $t(f_j)$ is $w(j)$, and the rows generate N. The exact sequence is a *projective presentation* for U.

Let $\mathcal{G}$ be a Gröbner basis for I. Recall that we write elements in A as elements in the span of $\mathrm{Nontip}(I)$. Naturally extend the definition of $v\mathbb{K}\Gamma$ to vA. A *vertex projective* right A-module is a right A-module of the form $\oplus_{i\in\mathcal{I}} v(i)A$, where $v : \mathcal{I} \to V$ is a set function. A *vertex projective presentation* for a right A-module U is an exact sequence

$$\oplus_{j\in\mathcal{J}} w(j)A \xrightarrow{(\lambda_{ji})} \oplus_{i\in\mathcal{I}} v(i)A \to U \to 0,$$

where $\mathcal{I}, \mathcal{J}$ are index sets, $v : \mathcal{I} \to V$ and $w : \mathcal{J} \to V$ are set functions, and $\lambda_{ji} \in A$. Each row λ_j in the matrix (λ_{ji}) is an element in $\oplus_{i\in\mathcal{I}} v(i)\mathbb{K}\Gamma$, and $t(\lambda_j) = w(j)$. The following theorem demonstrates that right Gröbner basis techniques are applicable to modules described by a vertex projective presentation. All right A-modules have a vertex projective presentation, because all right A-modules have a free presentations and $A = \oplus_{v\in V} vA$.

Theorem 7.2.2. *Let U be a right A-module given by a vertex projective presentation. Then, U is also a right $\mathbb{K}\Gamma$-module given by a projective presentation.*

Proof. Let $\mathcal{I}$ and $\mathcal{J}$ be index sets, and let $v : \mathcal{I} \to V$ and $w : \mathcal{J} \to V$ be set functions mapping the index sets to vertices. We assume that we are given the following commutative diagram:

$$\begin{array}{ccccccc}
 & & 0 & & 0 & & \\
 & & \uparrow & & \uparrow & & \\
\oplus_{j\in\mathcal{J}} w(j)A & \xrightarrow{(\lambda_{ji})} & \oplus_{i\in\mathcal{I}} v(i)A & \to & U & \to & 0 \\
 & & \uparrow & & \uparrow= & & \\
 & & \oplus_{i\in\mathcal{I}} v(i)\mathbb{K}\Gamma & \to & U & \to & 0 \\
 & & \uparrow & & \uparrow & & \\
 & & \oplus_{i\in\mathcal{I}} v(i)I & \to & 0 & & \\
 & & \uparrow & & & & \\
 & & 0 & & & &
\end{array}$$

We wish to find an index set $\mathcal{J}'$, a set map $w' : \mathcal{J}' \to V$, and a matrix $(f_{i,j})$ to complete the diagram as follows:

$$\begin{array}{ccccccccc}
 & & & & 0 & & 0 & & \\
 & & & & \uparrow & & \uparrow & & \\
 & & & & \oplus_{j\in\mathcal{J}} w(j)A & \xrightarrow{(\lambda_{ji})} & \oplus_{i\in\mathcal{I}} v(i)A & \to & U & \to & 0 \\
 & & & & \uparrow & & \uparrow & & \uparrow= & & \\
0 & \to & \oplus_{j\in\mathcal{J}'} w'(j)\mathbb{K}\Gamma & \xrightarrow{(f_{ji})} & \oplus_{i\in\mathcal{I}} v(i)\mathbb{K}\Gamma & \to & U & \to & 0 \\
 & & \uparrow & & \uparrow & & \uparrow & & \\
0 & \to & \oplus_{i\in\mathcal{I}} v(i)I & \xrightarrow{=} & \oplus_{i\in\mathcal{I}} v(i)I & \to & 0 & & \\
 & & \uparrow & & \uparrow & & & & \\
 & & 0 & & 0 & & & &
\end{array}$$

To find (f_{ji}), first we write $\oplus_{i\in\mathcal{I}} v(i)I$ as $\oplus_{l\in\mathcal{L}} h_l K\Gamma$ for some $h_l \in \oplus_{i\in\mathcal{I}} v(i)I$. The h_l can be computed by computing a right Gröbner basis $\mathcal{H}$ for I such that $I = \oplus_{h\in\mathcal{H}} h\mathbb{K}\Gamma$, and partitioning $\mathcal{H}$ by starting vertices. Each component $v(i)I$ is written by running through the elements of $v(i)\mathcal{H}$ and putting them in the i-th component with all other components set to zero. View the rows of (λ_{ji}) as elements in $\oplus_{i\in\mathcal{I}} v(i)\mathbb{K}\Gamma$ and add in the h_l to obtain a set F. Compute a (tip-reduced uniform) right Gröbner basis $\mathcal{F}$ from F to obtain (f_{ji}) by making the elements in $\mathcal{F}$ the rows in (f_{ji}). The index set $\mathcal{J}'$ and the function $w' : \mathcal{J}' \to V$, are obtained from the terminating vertices of the rows in (f_{ji}) □

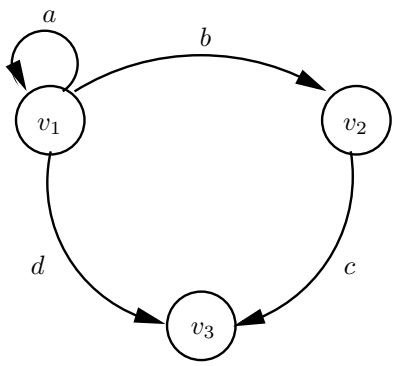


Figure 7.1: A sample quiver $G = (V, A)$.

Example 7.2.3. Figure 7.1 shows a graph defining a quiver $G = (V, A)$. Let Γ be the set of paths in G. Let $\mathbb{K} = \mathbb{Q}$ the rational numbers. Let I be the ideal generated by the relations $\mathcal{G} = \{a^3, bc - a^2d\}$. Using length left lexicographic ordering, $\mathcal{G}$ is a Gröbner basis for I. The set of nontips is

$$\text{Nontip}(I) \quad = \quad \left\{v_1, v_2, v_3, a, b, c, d, a^2, ab, ad, bc, a^2b\right\}.$$

Let $\Lambda = \mathbb{Q}\Gamma/I$. Then, $\text{Dim}_{\mathbb{Q}}\Lambda = 13$.

Define U with the vertex projective presentation

$$v_1A \oplus v_2A \oplus v_3A \xrightarrow{\begin{pmatrix} a^2+a & a \\ b+ab & a^2b \\ bc & ad \end{pmatrix}} v_1A \oplus v_1A \to U \to 0.$$

A tip-reduced uniform right Gröbner basis $\mathcal{F}$ for I is $\left\{a^3, a^2d - bc, abc, a^2bc\right\}$. Using $\mathcal{F}$ to lift (λ_{ji})

to $\mathbb{K}\Gamma$, we obtain the matrix

$$F = \begin{pmatrix} a^2+a & a \\ b+ab & a^2b \\ bc & ad \\ a^3 & 0 \\ a^2d-bc & 0 \\ abc & 0 \\ a^2bc & 0 \\ 0 & a^3 \\ 0 & a^2d-bc \\ 0 & abc \\ 0 & a^2bc \end{pmatrix}.$$

Calculating the right Gröbner basis from F, (f_{ji}) is

$$\begin{pmatrix} a & -a^2+a \\ 0 & a^3 \\ b & 2a^2-ab \\ 0 & ad \\ 0 & bc \\ 0 & a^3 \\ 0 & a^2d \\ 0 & abc \\ 0 & a^2bc \end{pmatrix}.$$

The right Gröbner basis $\mathcal{H}$ for U are the rows of (f_{ji}). Viewing elements in U as nontips, a basis $\mathcal{B}$ for U is

$$\mathcal{B} = \left\{(0,v_1),(0,a),(0,b),(0,d),(0,a^2),(0,ab),(0,a^2b),(v_1,0),(d,0)\right\}.$$

Thus, $\dim_{\mathbb{Q}} U = 9$.

7.2.1 Encoding via Projective Presentations

The ideas behind project presentations and right Gröbner bases lead to a straightforward computer encoding. To define a A-module $U = M/N$, by a vertex projective presentation we first construct

Table 7.3: Worst case time complexity for projective presentation encoding.

Operation	Complexity
$+_M$	$\Theta(nt(+_A))$
$*_{M,A}$	$\Theta(nt(*_A) + t(\text{REDUCE}))$
$*_{\mathbb{K},M}$	$\Theta(nt(*_{\mathbb{K},A}))$

Table 7.4: Worst case space complexity for projective presentation encoding.

Encoded	Complexity
$m \in M$	$\Theta(n\hat{s}\,(\tau_A(m)))$
M with right Gröbner basis $\mathcal{G}$	$\Theta(ns(v) + s(\mathcal{G}) + s(A))$

a vertex projective module M. A list of vertices is used to define the number and source vertex for each component. Elements in M are viewed as row vectors containing path algebra elements, restricted by the starting vertex for the component. Addition of module elements is performed componentwise. The action of $a \in A$ on a module element is just componentwise multiplication by a on the right.

The submodule N is constructed by a generating set $G \subset M$. It is straightforward to find a (right) uniform generating set from G. Replace each $g \in G$ with the set of non-zero elements $\{gv\}$, where v ranges over the vertices (embedded) in A. We assume that the elements in G are right uniform. The elements in G are the rows of the matrix λ in the projective presentation.

To construct U, we construct a right Gröbner basis $\mathcal{G}$ from G as previously described. Elements in U are encoded in the same way as elements in M, row vectors with components in A. Addition remains componentwise addition. Elements of A act on elements in U via componentwise multiplication on the right, however the multiplication takes place in $\mathbb{K}\Gamma$ (i.e., do **not** reduce by a Gröbner basis for I). Following the componentwise multiplication in $\mathbb{K}\Gamma$, the result is reduced via $\mathcal{G}$. The algorithm for reduction is essentially the same as REDUCE in Section 4.3.1. The differences are that $\text{Tip}(r)|\text{Tip}(x)$ if $\text{Tip}(r)$ is a prefix of $\text{Tip}(x)$, and, as a result, `u` in the algorithm is unnecessary.

Assuming n is the number of components in an element of M, Tables 7.3 and 7.4 summarize the encoding neutral time and space complexity of modules encoded via projective presentations. For the space complexity, $s(v)$ is the space needed to encode a vertex in A.

Chapter 8

Structure of Modules

A primary application of module theory is the representation theory and classification of finite groups. Several applications of module theory are discussed by Curtis and Reiner [21, 22, 23]. We assume the reader is familiar with the terminology contained in Section 2.4.

Within the theme of this dissertation, we are particularly interested in the structure of A-modules, which aids algebraists in the study of algebra A as well. In this chapter, we present two tools used by algebraists to study the structure of modules: the MeatAxe algorithm of Parker [80] and endomorphism rings.

The remainder of this chapter is organized as follows. Section 8.1, we review the MeatAxe of Parker and provide an encoding neutral analysis. In Section 8.2, we discuss the relationship between endomorphism rings and the decomposition of modules. We briefly review existing algorithms and present a new algorithm to compute endomorphism rings of A-modules, where A is a path algebra. We experimentally show that this algorithm has a slower rate of growth than the algorithm used by Magma [12], a commercial computer algebra system. We summarize the chapter in Section 8.3.

8.1 The MeatAxe

An early computational tool in the study of modular representation theory is the MeatAxe of Parker [80]. The MeatAxe of Parker is used to determine if a A-module is reducible or irreducible, where A is a $\mathbb{F}_q$-algebra. Implementations of the MeatAxe algorithm are included in GAP [42], Magma [12], and as a standalone application [83]. Applications of MeatAxe include the computa-

MeatAxe(A, M) Determine if M is irreducible.

INPUT: An algebra A and an A-module M.
OUTPUT: "Irreducible" if M is irreducible, "Reducible" otherwise.

```
1   do
2       a ← randomly chosen element from A
3   while Ker_M(a) = {0_M}
4   b ← Ker_M(a).basis
5   for x ∈ b
6       do if x · A ≠ M
7             then return "Reducible"
8   b* ← Ker_{M*}(a).basis
9   for x ∈ b*
10      do if A · x = M*
11            then return "Irreducible"
12  return "Reducible"
```

Figure 8.1: The basic MeatAxe algorithm.

tion of modular character tables [18] and to calculate the submodule lattice of a module [67].

To determine if a module is irreducible, MeatAxe employs Norton's irreducibility criterion. Let M^* be the vector space dual of M. Let $a \in A$ be an element of the algebra. The *kernel of a in M* $\mathrm{Ker}_M(a) = \{m \in M \mid ma = 0_M\}$ is the set of elements annihilated by a.

Theorem 8.1.1 (Norton's irreducibility criterion). *Let M be an A-module of finite $\mathbb{F}_q$-dimension and let $a \in A$ be an element such that $\mathrm{Ker}_M(a) \neq \{0_M\}$. Then M is irreducible if and only if*

1. *$v \cdot A = M$ for all $v \in \mathrm{Ker}_M(a), v \neq 0$, and*
2. *$A \cdot w = M^*$ for some $w \in \mathrm{Ker}_{M^*}(a), w \neq 0$.*

Proof. See Theorem 2.1.2 in Szőke. □

Traditionally in the literature, Norton's criterion is presented in terms of transposes of matrices. We show in Theorem 8.2.7 that the two descriptions coincide. The description we use in Theorem 8.1.1 is more appropriate for developing an encoding neutral MeatAxe algorithm.

MeatAxeHR(A, M) Determine if M is irreducible.

INPUT: An algebra A and an A-module M.
OUTPUT: "Irreducible" if M is irreducible, "Reducible" otherwise.

```
1   while true
2         do y ← randomly chosen element from A
3            c ← CharPoly(y)
4            facts ← Factor(c)
5            for p ∈ facts   ▷ (in order of increasing degree)
6                do good ← false
7                   a ← p(y)
8                   Calculate Ker_M (a)
9                   if Ker_M (a).dimension = p.degree
10                    then good ← true
11                  v ← a non-zero vector ∈ Ker_M (a)
12                  if the module generated by v ≠ M
13                    then return "Reducible"
14                  Calculate Ker_{M*} (a)
15                  w ← a non-zero vector ∈ Ker_{M*} (a)
16                  if the module generated by w ≠ M*
17                    then return "Reducible"
18                  if good = true
19                    then return "Irreducible"
```

Figure 8.2: The Holt-Rees MeatAxe algorithm.

The basic MeatAxe algorithm works well over small finite fields, but its performance over large finite fields can be poor. Holt and Rees [54] recognize that the probability of $\mathrm{Ker}_M(a) \neq \{0\}$ for a randomly selected a is approximately $1/q$, for the field $\mathbb{F}_q$. Hence, for large q, the basic MeatAxe algorithm will spend significant time finding an appropriate a satisfying $\mathrm{Ker}_M(a) \neq \{0\}$ needed by the Norton test.

Instead, Holt and Rees use a different strategy for choosing a. Let y be a randomly chosen element from A. Compute the characteristic polynomial $c_y(x)$ of y and then compute the factors of $c_y(x)$. Let $p(x)$ be an irreducible factor of $c_y(x)$. Let $a = p(y)$, evaluating $p(x)$ on y. The element a always has a non-trivial kernel. Figure 8.2 contains psuedocode for the Holt-Rees version of MeatAxe.

Holt and Rees show that their procedure has a probability of 0.144 of reaching a definitive answer

except for one class of modules. Ivanyos and Lux [60] treat this case as follows. Select a factor $p(x)$ of minimum degree and minimum multiplicity among the factors of $c_y(x)$. By the Chinese Remainder Theorem,

$$\mathbb{F}_x/\langle c_y(x)\rangle \quad \cong \quad \mathbb{F}_x/\langle p(x)^l\rangle \oplus \mathbb{F}_x/\langle q(x)\rangle,$$

where l is the multiplicity of $p(x)$ in $c_y(x)$ and $q(x) = c_y(x)/(p(x)^l)$. Find the identity element $i(x)$ of the component isomorphic to $\mathbb{F}_x/\langle p(x)^l\rangle$. This is computed using the extended Euclidean algorithm.

Once $i(x)$ is computed, randomly choose another element $z \in A$ and a random vector $v \in M$. Let $t = i(y)$ and calculate the submodule N generated by $v(ytzt - tzty)$. If N is a proper non-zero submodule, return "Reducible", otherwise start the process over. The probability of successfully finding a submodule when M is reducible is shown to be 0.08. Figure 8.3 contains pseudocode for the Ivanyos-Lux version of MeatAxe.

We leave a detailed analysis of MeatAxeIL for future work, but remark on some aspects of the algorithm. The algebra A need not be finite dimensional, but it must be finitely generated. Furthermore, the module M is finite dimensional over $\mathbb{F}_q$. If A is not finite dimensional, it is currently not known how to choose elements in A uniformly at random. However, $\rho(A)$, the representation of A defined by M is finite dimensional, and a basis could be found to generate elements in $\rho(A)$ if necessary. In practice, guaranteed uniformity is not needed for MeatAxe to perform well, so computing a basis for $\rho(A)$ is avoided in practical implementations, because it is generally considered too expensive.

In the encoding neutral version of MeatAxeIL, it is necessary to compute the representation of `y` and `a` defined by M in order to calculate $c_y(x)$, the characteristic polynomial, and the kernels $\mathrm{Ker}_M(a)$ and $\mathrm{Ker}_{M^*}(a)$. If M is encoded via a projective presentation, it is unclear whether converting M to a module encoded by a representation or generating representations for algebra elements as needed is more efficient. Experimentation in this area is left for future work.

8.2 Endomorphism Rings

In this section, we introduce the endomorphism ring of a module M. The endomorphism ring provides additional information about the structure of M. In particular, decompositions of M are

MeatAxeIL(A, M) Determine if M is irreducible.

INPUT: An algebra A and an A-module M.
OUTPUT: "Irreducible" if M is irreducible, "Reducible" otherwise.

```
1   while true
2         do y ← randomly chosen element from A
3            c ← CharPoly(y)
4            facts ← Factor(c)
5            for p ∈ facts (in order of increasing degree)
6                 do good ← false
7                    a ← p(y)
8                    Calculate Ker_M (a)
9                    if Ker_M (a).dimension = p.degree
10                     then good ← true
11                   v ← a non-zero vector ∈ Ker_M (a)
12                   if the module generated by v ≠ M
13                     then return "Reducible"
14                   Calculate Ker_{M*} (a)
15                   w ← a non-zero vector ∈ Ker_{M*} (a)
16                   if the module generated by w ≠ M*
17                     then return "Reducible"
18                   if good = true
19                     then return "Irreducible"
20           p ← {f ∈ facts | f has minimum degree and minimum multiplicity }
21           l ← multiplicity of p
22           q ← c/p^l
23           Find a(x) and b(x) such that 1 = a(x)p^l(x) + b(x)q(x)
24           i ← b(x)q(x) (mod c(x))
25           z ← randomly chosen element from A
26           v ← randomly chosen vector from M
27           t ← i(y)
28           Calculate module N generated by v · (ytzt − tzty)
29           if N ≠ 0 and N ≠ M
30             then return "Reducible"
```

Figure 8.3: The Ivanyos-Lux extended MeatAxe.

strongly related to decompositions of its endomorphism ring. An earlier version of the research in this section appears in Green, Heath, and Struble [51]. We begin by defining some terminology used throughout this section. The terminology used is found in several textbooks about algebra [36, 31, 57].

Let A be an algebra and let M and N be A modules. The *homomorphism group* of M and N is the set

$$\mathrm{Hom}_A(M,N) \quad = \quad \{f : M \to N \,|\, f \text{ is an } A\text{-module homomorphism}\},$$

which is an abelian group under addition of maps. The set $\mathrm{End}_A(M) = \mathrm{Hom}_A(M,M)$ is the *endomorphism ring* of M, which is a ring under addition of maps and composition of maps (as multiplication). Since A is an algebra, $\mathrm{End}_A(M)$ is also an algebra.

As with algebras and their ideals, finite dimensional (over $\mathbb{K}$) modules are often classified by the submodules they contain. A *decomposition* of M is a set $\{M_i\}_{i=1}^n$ of submodules of M such that

$$M \quad = \quad M_1 \oplus M_2 \oplus \cdots \oplus M_n.$$

We say that M is *decomposable* if there exists a decomposition of M such that $n \geq 2$ and *indecomposable* otherwise. Decompositions of M such that every submodule in the decomposition is indecomposable are unique up to isomorphism.

Theorem 8.2.1 (Krull-Schmidt). *Let M be a finite dimensional (over $\mathbb{K}$) A-module. If M has two decompositions,*

$$M \quad = \quad M_1 \oplus \cdots \oplus M_s$$

and

$$M \quad = \quad N_1 \oplus \cdots \oplus N_t$$

such that each M_i and N_j are indecomposable, then $s = t$ and there exists a permutation π of $1 \leq i \leq s$ such that $M_{\pi(i)} \cong N_i$ for $1 \leq i \leq s$.

Proof. See Chapter 2, Theorem 3.8 in Hungerford [57]. □

We obtain the following problem regarding the structure of modules.

MODULE DECOMPOSITION

INSTANCE: A module M.

SOLUTION: Submodules $M_1, \ldots, M_n$ of M such that $M = M_1 \oplus \cdots \oplus M_n$.

We relate decompositions of M to decompositions of the identity map Id_M in $\mathrm{End}_A(M)$ as direct sums of pairwise orthogonal idempotents (not necessarily central) as follows. Assume that $M_1 \oplus \cdots \oplus M_n$ is a decomposition of M. For $1 \leq i \leq n$, let π_i denote the projection onto M_i with kernel $\sum_{j \neq i} M_j$. The maps π_i are pairwise orthogonal idempotents in $\mathrm{End}_A(M)$ with $\sum_{i=1}^{m} \pi_i = \mathrm{Id}_M$. If $\mathrm{Id}_M = \pi_1 + \cdots + \pi_n$ as the sum of pairwise orthogonal idempotents, then $M = M_1 \oplus \cdots \oplus M_n$ where M_i is the image of π_i for $1 \leq i \leq m$.

Recall that $\mathrm{End}_A(M)$ is an algebra itself. We briefly review the results of Chistov, et al. [16], applying the algorithms in Chapter 5 to find a complete set of primitive idempotents in $\mathrm{End}_A(M)$. By the Wedderburn-Malcev principle theorem (see Pierce [81]), there exists a subalgebra S of an algebra A' such that $S \cong A'/J(A')$. An algorithm for finding a suitable subalgebra S appears in de Graaf, et al. [26]. We apply the algorithm to $\mathrm{End}_A(M)$ and find a complete set of primitive idempotents in $\mathrm{End}_A(M)$ using the results in Chapter 5, assuming $\mathrm{End}_A(M)$ (and thus A) is an $\mathbb{F}_q$-algebra. We note that the results in Section 5.3.3 imply that $\mathbb{Q}$-algebra modules cannot be decomposed efficiently with this approach. To the best of our knowledge, decomposing $\mathbb{Q}$ modules efficiently is an open problem.

Consequently, the efficient construction of endomorphism rings is important to studying the structure of modules. Observe that module M is also a $\mathbb{K}$-vector space of dimension d. The actions of the generators of A can be written as $d \times d$ matrices with entries in $\mathbb{K}$. If g is a generator of A, let f_g be the matrix describing the action of g on M. The ring of $d \times d$ matrices X that satisfy the system of equations

$$X \cdot f_g - f_g \cdot X = 0,$$

for all generators g of M, is isomorphic to $\mathrm{End}_A(M)$.

Schneider [90] appears to have written the first implementations of algorithms for constructing endomorphism rings, including the linear algebra approach just described. These implementations are included in the computer algebra system Cayley (the predecessor of Magma). In the same paper, Schneider improves on the linear algebra approach for group algebras by reducing the number of unknowns and the number of equations to solve by randomly choosing endomorphisms and refining

approximations of the solution space. The process iterates, using information gained from the random endomorphism, until the entire solution space is determined.

Leedham-Green implements another algorithm, included in Magma [12], but it is unpublished. Smith implements another randomized algorithm in the AutAG share package included in GAP version 3.4.4 [89]. The algorithm is tuned for modules over group algebras. The comments in the source code state that the algorithm is based on discussions with Leedham-Green, Lux, and Niemeyer, but that its performance is not as good as the implementation in Magma.

Szőke [93] presents a randomized algorithm using peakwords [67] to construct the regular representation of $\mathrm{End}_A(M)$. That algorithm is included in the C implementation of the MeatAxe package [83]. According to Szőke's dissertation, the performance of her algorithm appears to be better than Magma in most situations.

In the remainder of this section, we present a new deterministic algorithm for computing endomorphism rings of modules defined over path algebras, encoded via projective presentations. We assume the reader is familiar with the content of Section 7.2. Section 8.2.1 presents the mathematical foundations behind our algorithm. Section 8.2.2 elaborates on the actions of path algebras on modules and their duals. Section 8.2.3 ties together the two previous sections into a concrete algorithm for constructing endomorphism rings of modules defined over path algebras. In Section 8.2.4, we present experimental results comparing our algorithm to the algorithm contained in Magma.

8.2.1 Adjoint Associativity

We begin by recalling a general result found in Hungerford [57, pg. 214].

Theorem 8.2.2 (Hungerford 5.10, Adjoint Associativity). *Let R and S be rings, and let A_R, ${}_RB_S$, C_S be (bi)-modules. Then there is an isomorphism of abelian groups*

$$\Upsilon : \mathrm{Hom}_S(A \otimes_R B, C) \cong \mathrm{Hom}_R(A, \mathrm{Hom}_S(B, C)),$$

defined for each $f : A \otimes_R B \to C$ by

$$[(\Upsilon f)(a)](b) = f(a \otimes b).$$

If A, B, and C are as given in Theorem 8.2.2, then $A \otimes_R B$ is a right S-module via $(a \otimes b)s = a \otimes (bs)$. We also have that $\mathrm{Hom}_S(B, C)$ is a right R-module given by the left R-module structure

of B: $(gr)(b) = g(rb)$ for $r \in R, b \in B$, and $g \in \mathrm{Hom}_S(B, C)$. Checking that the map (gr) is a right R-module homomorphism is left as an exercise.

Our particular interest in endomorphism rings is a specific application of this general theorem, as we now demonstrate. Let $\mathbb{K}$ be a field, let Λ be a $\mathbb{K}$-algebra, and let M and N be finite dimensional right Λ-modules. Since M and N are $\mathbb{K}$-vector spaces, M and N are both left and right $\mathbb{K}$-modules.

The *dual* of N, denoted N^*, is $\mathrm{Hom}_{\mathbb{K}}(N, \mathbb{K})$, the set of $\mathbb{K}$-linear maps from N to $\mathbb{K}$. The dual N^* is a left Λ-module via

$$(\lambda h)(n) \quad = \quad h(n\lambda), \tag{8.1}$$

where $\lambda \in \Lambda, h \in \mathrm{Hom}_{\mathbb{K}}(N, \mathbb{K})$, and $n \in N$. Hence, N^* is a left Λ-module and a right $\mathbb{K}$-module with $\lambda(h\alpha) = (\lambda h)\alpha$, where $\lambda \in \Lambda$, $h \in N^*$, and $\alpha \in \mathbb{K}$.

Since N is finite dimensional, there exists a natural isomorphism between $\mathrm{Hom}_{\mathbb{K}}(N^*, \mathbb{K})$ and N. Suppose $\{n_1, ..., n_d\}$ is a $\mathbb{K}$-basis of N. Define $n_i^* : N \to \mathbb{K}$ by

$$n_i^*(n_j) \quad = \quad \begin{cases} 0 & \text{if } i \neq j, \\ 1 & \text{if } i = j. \end{cases}$$

The set $\{n_1^*, ..., n_d^*\}$ forms the *dual basis* of N^*. Define $\Psi : \mathrm{Hom}_{\mathbb{K}}(N^*, \mathbb{K}) \to N$ by $\Psi(h) = \sum_{i=1}^d h(n_i^*)n_i$. One can check that the inverse of Ψ is the map $\rho : N \to \mathrm{Hom}_{\mathbb{K}}(N^*, \mathbb{K})$ given by $\rho(n) : N^* \to \mathbb{K}$ where $\rho(n)(h) = h(n)$ for $h \in N^*$. Thus, Ψ is an isomorphism. Note that ρ is independent of the choice of basis for N, so it follows that Ψ is also independent of the choice of basis, even though it is defined with a choice of basis.

Theorem 8.2.3. *Let $\mathbb{K}$ be a field, let Λ be a $\mathbb{K}$-algebra, and let M and N be finite dimensional right Λ modules. Then there exists an isomorphism of abelian groups*

$$\Phi : \mathrm{Hom}_{\mathbb{K}}(M \otimes_\Lambda N^*, \mathbb{K}) \quad \cong \quad \mathrm{Hom}_\Lambda(M, N).$$

given by $[(\Phi f)(m)] = \sum_{i=1}^d f(m \otimes n_i^*)n_i$.

Proof. Applying Theorem 8.2.2, we obtain the existence of the isomorphism

$$\Upsilon : \mathrm{Hom}_{\mathbb{K}}(M \otimes_\Lambda N^*, \mathbb{K}) \cong \mathrm{Hom}_\Lambda(M, \mathrm{Hom}_{\mathbb{K}}(N^*, \mathbb{K})),$$

and the map $[(\Upsilon f)(m)] : N^* \to \mathbb{K}$ is a $\mathbb{K}$-homomorphism. Applying Ψ, we get $\Psi[(\Upsilon f)(m)] = \sum_{i=1}^d [(\Upsilon f)(m)](n_i^*)(n_i) = \sum_{i=1}^d f(m \otimes n_i^*)n_i$. Hence Φ defined by $[(\Phi f)(m)] = \sum_{i=1}^d f(m \otimes n_i^*)n_i$ is an appropriate isomorphism. □

Corollary 8.2.4. *Let $\mathbb{K}$ be a field, Λ be a $\mathbb{K}$-algebra, and M a finite dimensional right Λ-module, then*

$$\Phi : \mathrm{Hom}_{\mathbb{K}}(M \otimes_{\Lambda} M^*, \mathbb{K}) \quad \cong \quad \mathrm{End}_{\Lambda}(M).$$

given by $[(\Phi f)(m)] = \sum_{i=1}^{d} f(m \otimes m_i^*)m_i$.

8.2.2 Bases, Actions of Λ, and Dual Bases

Let $G = (V, A)$ be a quiver, with $V = \{v_1, v_2, \ldots, v_l\}$. Let $\mathbb{K}\Gamma$ be a path algebra, and let I be an ideal in $\mathbb{K}\Gamma$ such that $\Lambda = \mathbb{K}\Gamma/I$ is finite dimensional. Let M be a finite dimensional right Λ-module. For each i, where $1 \leq i \leq l$, define $M_i = Mv_i$ and $M_i^* = v_i M^*$.

Lemma 8.2.5. *The $\mathbb{K}$-vector spaces M and M^* can be written as $M = \oplus_{i=1}^{l} M_i$ and $M^* = \oplus_{i=1}^{l} M_i^*$.*

Proof. The vertices v_i are orthogonal idempotents such that $\sum_{i=1}^{l} v_i = 1_{\Lambda}$. Clearly, the result holds. □

For i, where $1 \leq i \leq l$, let $d_i = \dim_{\mathbb{K}} M_i$ and choose a $\mathbb{K}$-basis $\{m_j^i\}_{j=1}^{d_i}$ of M_i. Let $(v_i, v_j, \sigma) \in A$ be an arrow in G. Define the $\mathbb{K}$-linear map $L_M(\sigma) : M \to M$ by $L_M(\sigma)(m) = m\sigma$. Since $v_i\sigma = \sigma v_j = \sigma$, $L_M(e)(M_i)$ is contained in M_j. Because vertices are orthogonal idempotents in Λ, $v_k\sigma = 0$ for all $v_k \in V$, $k \neq i$. As a result, $L_M(\sigma)(M_k) = \{0_M\}$ for $k \neq i$. It follows that $L_M(\sigma) : M_i \to M_j$.

Suppose we write vectors in M, and hence M_i as row vectors. A $d_i \times d_j$ $\mathbb{K}$-matrix (γ_{st}) encoding $L_M(\sigma)$ is defined by

$$L_M(\sigma)(m_s^i) = \sum_{t=1}^{d_j} \gamma_{st} m_t^j.$$

We view $L_M(\sigma)$ as its corresponding matrix (γ_{st}) for the remainder of this section. If $x = \sigma_1\sigma_2\cdots\sigma_q$ is a path, then right actions by x on elements of M are given by the product of matrices,

$$L_M(\sigma_1)L_M(\sigma_2)\cdots L_M(\sigma_q).$$

Thus, the action of each element a in Λ has an associated $\mathbb{K}$-matrix $L_M(a)$ by viewing a as an element in the span of $\mathrm{Nontip}(I)$.

Example 8.2.6. Using Example 7.2.3, we compute $L_U(\sigma)$ for each arrow $\sigma \in \Sigma$. First note that U_1 has basis $\{(0, v_1), (0, a), (0, a^2), (v_1, 0)\}$, U_2 has basis $\{(0, b), (0, ab), (0, a^2b)\}$, and U_3 has basis

$\{(0,d),(d,0)\}$. To calculate $L_U(a)$ for $a : v_1 \to v_2$, act on each basis element in U_1 via right multiplication by a in each component. The result is an element in U_1. Thus, we obtain

$$\begin{aligned}
(0,v_1)\cdot a &= (0,a) \text{ when reduced by } \mathcal{H},\\
(0,a)\cdot a &= (0,a^2) \text{ when reduced by } \mathcal{H},\\
(0,a^2)\cdot a &= (0,a^3) = (0,0) \text{ when reduced by } \mathcal{H},\\
(v_1,0)\cdot a &= (a,0) = (0,a^2-a) \text{ when reduced by } \mathcal{H}.
\end{aligned}$$

Hence,

$$L_U(a) = \begin{pmatrix} 0 & 1 & 0 & 0\\ 0 & 0 & 1 & 0\\ 0 & 0 & 0 & 0\\ 0 & -1 & 1 & 0 \end{pmatrix}.$$

Using the same approach, we compute the remaining maps, obtaining

$$L_U(b) = \begin{pmatrix} 1 & 0 & 0\\ 0 & 1 & 0\\ 0 & 0 & 1\\ 0 & 1 & -2 \end{pmatrix}, \qquad L_U(c) = \begin{pmatrix} 0 & 0\\ 0 & 0\\ 0 & 0 \end{pmatrix}, \qquad L_U(d) = \begin{pmatrix} 1 & 0\\ 0 & 0\\ 0 & 0\\ 0 & 1 \end{pmatrix}.$$

Next, we look at M^*. Recall that $\{m_j^i\}_{j=1}^{d_i}$ is a $\mathbb{K}$-basis of M_i. Dualizing, we obtain the $\mathbb{K}$-basis $\{(m_j^i)^*\}_{j=1}^{d_i}$ for M_i^*, where $\dim_{\mathbb{K}} M_i^* = d_i$. Let $(v_i, v_j, \sigma) \in A$ as before. As we did for the right action of σ on M, we want to compute the matrix $L_{M^*}(\sigma) : M_j^* \to M_i^*$ obtained from the left action of σ on M^*. Recall that $\sigma \cdot (m_t^j)^* : M^* \to \mathbb{K}$ is given by

$$(\sigma \cdot (m_t^j)^*)(m_s^i) = (m_t^j)^*((m_s^i)\sigma).$$

Let (γ_{xy}) be the $\mathbb{K}$-matrix encoding for $L_M(\sigma)$. From our earlier discussion,

$$(m_s^i)\sigma = \sum_{u=1}^{d_j} \gamma_{su} m_u^j.$$

Hence,

$$\sigma \cdot (m_t^j)^* = \sum_{s=1}^{d_i} \gamma_{ts} (m_s^i)^*;$$

that is, $L_{M^*}(\sigma) = (L_M(\sigma))^{\mathrm{T}}$. The left Λ structure of M^* is readily obtained from the right Λ structure of M.

Theorem 8.2.7. *If $x = \sigma_1\sigma_2\cdots\sigma_q$ is a path in Γ, then the left action of x on M^* is computed via the transpose of the right action of x on M:*

$$(L_M(\sigma_q))^{\mathrm{T}} \cdot (L_M(\sigma_{q-1}))^{\mathrm{T}} \cdots (L_M(\sigma_1))^{\mathrm{T}}.$$

8.2.3 Constructing the Endomorphism Ring

We now use the results from the previous sections to construct the endomorphism ring of a finite dimensional right Λ-module M, where Λ is a finite dimensional quotient of a path algebra as defined in Section 8.2.2. We assume we have constructed a basis $\mathcal{B} = \{m_1, ..., m_d\}$ of M as described in Section 7.2. For each $(v_i, v_j, \sigma) \in A$, we compute $L_M(\sigma) : M_i \to M_j$ as discussed in Section 8.2.2.

It remains to compute $M \otimes_\Lambda M^*$ to be able to apply Corollary 8.2.4 for the construction of $\mathrm{End}_\Lambda(M)$. Recall that we have a vertex projective presentation for M,

$$\oplus_{j=1}^{r} w(j)\Lambda \xrightarrow{(\lambda_{ji})} \oplus_{i=1}^{g} v(i)\Lambda \to M \to 0.$$

Tensoring on the right with M^*, we obtain the exact sequence

$$\oplus_{j=1}^{r} w(j)\Lambda \otimes_\Lambda M^* \xrightarrow{(\lambda_{ji})\otimes 1} \oplus_{i=1}^{g} v(i)\Lambda \otimes_\Lambda M^* \to M \otimes_\Lambda M^* \to 0.$$

We see that $\oplus_{j=1}^{r} w(j)\Lambda \otimes_\Lambda M^* = \oplus_{j=1}^{r} w(j)M^*$ and $\oplus_{i=1}^{g} v(i)\Lambda \otimes_\Lambda M^* = \oplus_{i=1}^{g} v(i)M^*$. Thus $M \otimes_\Lambda M^*$ is the cokernel of

$$\oplus_{j=1}^{r} w(j)M^* \xrightarrow{(\lambda_{ji})} \oplus_{i=1}^{g} v(i)M^*. \tag{8.2}$$

The previous calculations also hold when viewing M as a right $\mathbb{K}\Gamma$-module by replacing Λ with $\mathbb{K}\Gamma$, (λ_{ji}) with (f_{ji}) obtained in Section 7.2, and the fact that $M \otimes_{\mathbb{K}\Gamma} M^* \cong M \otimes_\Lambda M^*$. Thus, if (f_{ji}) has fewer rows than (λ_{ji}), it may be advantageous to use (f_{ji}) in the following description. We assume that (λ_{ji}) has fewer rows and use it to continue with the construction of the endomorphism ring.

To compute the cokernel, we construct a $\mathbb{K}$-matrix D for the action of (λ_{ji}) in Equation 8.2 and use linear algebra to find the cokernel. Choose the dual basis $\mathcal{B}^* = \{m_1^*, m_2^*, \cdots m_d^*\}$ of M^*. If

$v \in V$, then choose the $\mathbb{K}$-basis $\{m_i^* \in \mathcal{B}^* \mid t(m_i) = v\}$ for vM^*. The previous statement is sensible from definitions and description of the basis for M in Section 7.2. If the entries

$$\lambda_{ji} \quad = \quad \sum_k \alpha_{jik}\sigma_{jik1}\sigma_{jik2}\cdots\sigma_{jikq_k},$$

where $\alpha_{jik} \in \mathbb{K}$ and $\sigma_{jikq} \in \Sigma$, then

$$(\lambda_{ji})(m_s^*) \quad = \quad (\lambda_{j1}m_s^*, \lambda_{j2}m_s^*, \ldots, \lambda_{jg}m_s^*).$$

Using the results from Section 8.2.2, each entry of D can be computed since $(\lambda_{ji})(m_s^*)$ is the row for m_s of

$$\sum_k \alpha_{jik} L_{M^*}(\sigma_{jikq_k}) \cdots L_{M^*}(\sigma_{jik1}) = \sum_k \alpha_{jik} (L_M(\sigma_{jikq_k}))^{\mathrm{T}} \cdots (L_M(\sigma_{jik1}))^{\mathrm{T}}.$$

We obtain a $\mathbb{K}$-matrix D for (λ_{ji}) where the rows are easily mapped to elements of $\sum_{i=1}^g v(i)M^*$. The solution space $\mathcal{D}$ of $x \cdot (D)^{\mathrm{T}} = 0$ is isomorphic to the cokernel, which is isomorphic to $M \otimes_\Lambda M^*$.

The map from $\sum_{i=1}^g v(i)M^*$ to $M \otimes_\Lambda M^*$ is $v(i)m_s^*$ is sent to $b_i \otimes m_s^*$, where b_i is the vector with the i^{th} component containing $v(i)$ and all other components containing 0. Thus, a basis for $\mathcal{D}$ can be interpreted directly in the form $\sum_{s,t} \beta_{s,t} b_s \otimes m_t^*$ to obtain a basis $\mathcal{C}$ for $M \otimes_\Lambda M^*$.

Using the results in Section 8.2.1, the endomorphisms are computed in the following way. Dualize $\mathcal{C}$ to obtain a basis $\mathcal{C}^*$ for $(M \otimes_\Lambda M^*)^*$. Applying Lemma 8.2.3, every element $c^* \in \mathcal{C}^*$ maps to a basis element for $\mathrm{End}_\Lambda(M)$ by $\Phi(c^*)(m) = \sum_{i=1}^d c^*(m \otimes m_i^*)m_i$. Writing the elements in $\mathcal{C}$ as $\sum_{s,t} \beta_{st} b_s \otimes m_t^*$, we obtain

$$\Phi\left((\sum_{s,t} \beta_{st} b_s \otimes m_t^*)^*\right)(m) \quad = \quad \sum_{i=1}^d \left((\sum_{s,t} \beta_{st} b_s \otimes m_t^*)^*\right)(m \otimes m_i^*)m_i.$$

The elements b_j, for $1 \leq j \leq g$, generate M as a module. Thus, computing $\Phi((\sum_{s,t} \beta_{st} b_s \otimes m_t^*)^*)(b_j)$ is sufficient to determine the endomorphisms forming a basis for $\mathrm{End}_\Lambda(M)$. We also have

$$\begin{aligned}\Phi\left((\sum_{s,t} \beta_{st} b_s \otimes m_t^*)^*\right)(b_j) \quad &= \quad \sum_{i=1}^d \left((\sum_{s,t} \beta_{st} b_s \otimes m_t^*)^*\right)(b_j \otimes m_i^*)m_i \\ &= \quad \sum_t \beta_{jt} m_t.\end{aligned}$$

Essentially, the basis elements c^* of $(M \otimes_\Lambda M^*)^*$ are describing where $\Phi(c^*)$ sends the generators of M. The computation of the endomorphism ring is complete.

Table 8.1: Correspondence between matrix column and element of $U \otimes_\Lambda U^*$.

column	element
1	$(v_1, 0) \otimes (0, v_1)^*$
2	$(v_1, 0) \otimes (0, a)^*$
3	$(v_1, 0) \otimes (0, a^2)^*$
4	$(v_1, 0) \otimes (v_1, 0)^*$
5	$(0, v_1) \otimes (0, v_1)^*$
6	$(0, v_1) \otimes (0, a)^*$
7	$(0, v_1) \otimes (0, a^2)^*$
8	$(0, v_1) \otimes (v_1, 0)^*$

Example 8.2.8. We continue with Examples 7.2.3 and 8.2.6. Recall that the matrix (λ_{ji}) is

$$\begin{pmatrix} a^2 + a & a \\ b + ab & a^2 b \\ bc & ad \end{pmatrix}.$$

The $\mathbb{K}$-matrix D describing the action of (λ_{ji}) is

$$\begin{pmatrix} (L_U(a))^{\mathrm{T}}(L_U(a))^{\mathrm{T}} + (L_U(a))^{\mathrm{T}} & (L_U(a))^{\mathrm{T}} \\ (L_U(b))^{\mathrm{T}} + (L_U(b))^{\mathrm{T}}(L_U(a))^{\mathrm{T}} & (L_U(b))^{\mathrm{T}}(L_U(a))^{\mathrm{T}}(L_U(a))^{\mathrm{T}} \\ (L_U(c))^{\mathrm{T}}(L_U(b))^{\mathrm{T}} & (L_U(d))^{\mathrm{T}}(L_U(a))^{\mathrm{T}} \end{pmatrix}$$

$$= \left(\begin{array}{cccc|cccc} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & -1 & 1 & 0 & 0 & -1 \\ 1 & 1 & 0 & 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ \hline 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & -1 & 1 & 0 & 0 & -1 \\ \hline 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{array}\right).$$

The columns in the matrix correspond to elements in $U \otimes_\Lambda U^*$ as shown in Table 8.1. The third row corresponds to the element $-(v_1, 0) \otimes (0, v_1)^* + (v_1, 0) \otimes (0, a)^* + (0, v_1) \otimes (0, a)^* + (0, v_1) \otimes (0, v_1)^*$.

The set

$$\mathcal{C} \quad = \quad \{(0,0,0,1,1,0,0,0),(0,0,0,0,0,0,1,0),(0,0,0,-1,0,-1,0,1)\}$$

is a basis for the cokernel of D. The correspondence between elements of $\mathcal{C}$ and elements of $U \otimes_\Lambda U^*$ is

$$\begin{aligned}(0,0,0,1,1,0,0,0) &\mapsto (v_1,0)\otimes(v_1,0)^* + (0,v_1)\otimes(0,v_1)^*,\\ (0,0,0,0,0,0,1,0) &\mapsto (0,v_1)\otimes(0,a^2)^*,\\ (0,0,0,-1,0,-1,0,1) &\mapsto -(v_1,0)\otimes(v_1,0)^* - (0,v_1)\otimes(0,a)^* + (0,v_1)\otimes(v_1,0)^*.\end{aligned}$$

The basis elements are interpreted directly as endomorphisms. For example, $(0,v_1)\otimes(0,a)^*$ corresponds to the endomorphism mapping $(0,v_1)\mapsto(0,a)$ and $(v_1,0)\mapsto(0,0)$. The basis element $(v_1,0)\otimes(v_1,0)^* + (0,v_1)\otimes(0,v_1)^*$ corresponds to the endomorphism mapping $(v_1,0)\mapsto(v_1,0)$ and $(0,v_1)\mapsto(0,v_1)$; i.e., the identity map. Since $(v_1,0)$ and $(0,v_1)$ generate U, the images for the remaining basis elements, thus all elements, of U are found by naturally extending the map.

8.2.4 Performance

The algorithm described in this paper has been implemented using GAP version 4.1 bugfix 7 [42] and is included in HOPF [52]. The implementations of quivers, right modules over path algebras, and factor modules over path algebras were written by the author. The implementations of path algebras and quotients of path algebras build on implementations of *magma rings* from the GAP library.

We compared our algorithm experimentally with Magma version 2.5-1. Algebras are described using Magma's implementation of basic algebras, which are equivalent to the path algebras described above. Modules are constructed as projective modules in Magma, and the endomorphism ring is constructed with the ENDOMORPHISMALGEBRA function. Because the source for Magma is unavailable, we are unable to expand on the implementations of operations for basic algebras or for computing endomorphism rings.

All tests were run on a Sparc Ultra 30 running Solaris version 2.7 containing a 296 MHz UltraSparc-II processor and 1 gigabyte of RAM. For our first test, we used the quiver in Figure 8.4, the field $\mathbb{K}=\mathbb{Q}$, the path algebra $\mathbb{K}\Gamma$. The module U is given with the presentation

$$(\oplus_{k=1}^n v_3\mathbb{K}\Gamma)\oplus(\oplus_{k=1}^n v_4\mathbb{K}\Gamma) \xrightarrow{(f_{ji})} (\oplus_{k=1}^n v_1\mathbb{K}\Gamma)\oplus(\oplus_{k=1}^n v_2\mathbb{K}\Gamma) \to U \to 0$$

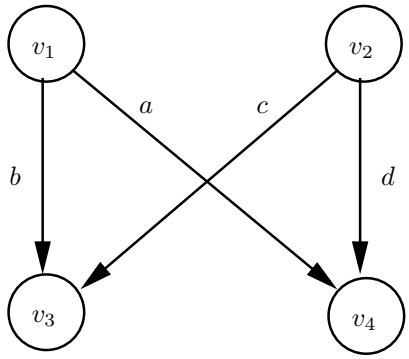


Figure 8.4: Quiver for test 1.

where (f_{ji}) has the form

$$\left(\begin{array}{cccc|cccc} -a & 0 & 0 & 0 & d & 0 & 0 & 0 \\ 0 & -a & 0 & 0 & 0 & d & 0 & 0 \\ 0 & 0 & \ddots & 0 & 0 & 0 & \ddots & 0 \\ 0 & 0 & 0 & -a & 0 & 0 & 0 & d \\ \hline -b & 0 & 0 & 0 & 0 & c & 0 & 0 \\ 0 & -b & 0 & 0 & 0 & 0 & c & 0 \\ 0 & 0 & \ddots & 0 & 0 & 0 & 0 & \ddots \\ 0 & 0 & 0 & -b & 0 & 0 & 0 & 0 \end{array}\right)$$

and each block is $n \times n$. The resulting module U has dimension $4n$ and the dimension of the endomorphism ring is n.

Our second test used $\mathbb{K}\langle x, y\rangle$, the free algebra in two variables, where $\mathbb{K}$ is $GF(7)$, the Galois field with seven elements. Let I be $\langle$words of length 3$\rangle$ and let Λ be $\mathbb{K}\langle x, y\rangle / I$. The module U is given with the presentation

$$\oplus_{k=1}^{n-1}\Lambda \xrightarrow{(\lambda_{ji})} \oplus_{k=1}^{n}\Lambda \to U \to 0$$

where (λ_{ji}) is an $(n-1) \times n$ matrix of the form

$$\begin{pmatrix} x^2 & y^2 & 0 & \cdots & 0 & 0 \\ 0 & x^2 & y^2 & \cdots & 0 & 0 \\ & & & \vdots & & \\ 0 & 0 & 0 & \cdots & x^2 & y^2 \end{pmatrix}.$$

The resulting module U has dimension $6n+1$ and the endomorphism ring has dimension $5n^2+n+1$.

Figure 8.5 displays the running times in CPU milliseconds of our implementation in GAP and the Magma implementation for constructing the endomorphism of U for a range of n. Each test was run five times. The minimum, average, and maximum times for constructing the endomorphism are displayed.

We see that the rate of growth of our implementation is lower than the rate of growth for the Magma implementation. This was expected behavior for quivers with multiple vertices. But as the results demonstrate, our algorithm also performs well for quotients of free algebras, which are modeled by quotients of path algebras defined by a quiver containing only one vertex.

We are not sure why there is a wide range in the minimum and maximum times in some cases in test 1 for the Magma implementation. One reason might be that Magma is using a randomized algorithm, and in some executions the randomized algorithm is able to achieve a solution quickly. However, the average performance is still worse than our algorithm in those cases.

8.3 Summary

In this chapter, we review two approaches to determine the structure of modules: the MeatAxe and endomorphism rings. For the MeatAxe, we review recent results and comment on encoding neutral specific issues of the algorithm. A detailed analysis has been left as future work.

We present background material on endomorphism rings and review existing algorithms for calculating them. The connection between endomorphism rings and decompositions of modules is also reviewed. In addition, we present a new deterministic algorithm for computing endomorphism rings of modules defined over path algebras. This algorithm is experimentally compared with the algorithm used in Magma. We see from our experiments that the new algorithm has better growth characteristics than the one included in Magma.

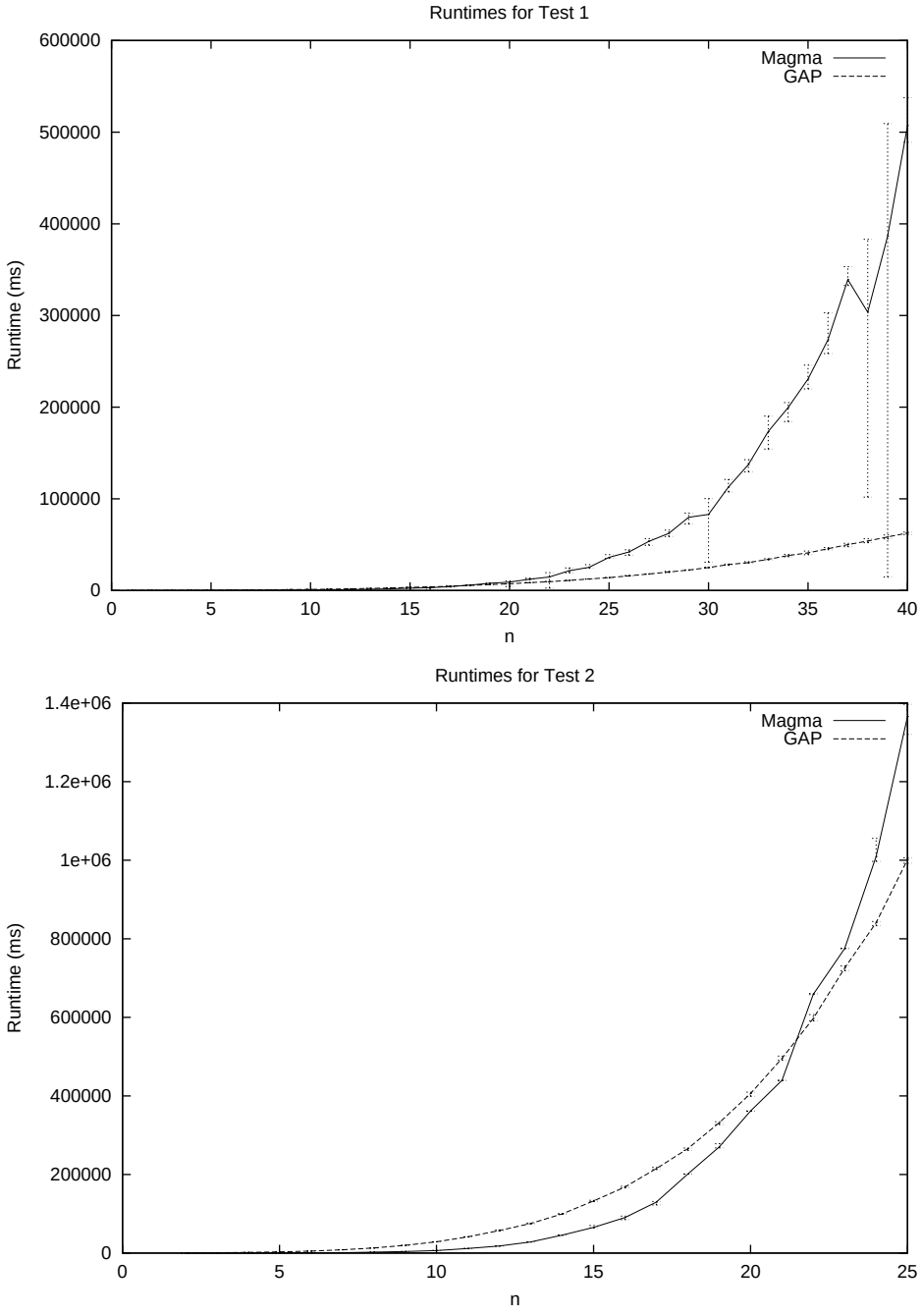


Figure 8.5: Runtimes for computing endomorphism rings.

Chapter 9

Determining Properties of Quotients of Path Algebras

In general, path algebras (and their quotients) may be infinite dimensional. Many of the theorems and algorithms presented in this dissertation assume that an algebra A is finite dimensional. It is necessary for use to determine whether a path algebra $A = \mathbb{K}\Gamma/I$ is finite or infinite dimensional, where I is an ideal of $\mathbb{K}\Gamma$. We assume that a finite Gröbner basis for I is known, as we require this for computation. Given this information, we can efficiently compute several properties of A: whether A is finite or infinite dimensional, the dimensional of A if it is finite dimensional, and an enumeration of Nontip(I) again if A is finite dimensional.

The presentation of our algorithm uses semigroups, which simplifies our arguments. We present definitions needed for our algorithm in Section 9.1. The main algorithm is described in Section 9.2. We discuss enumeration of elements in Section 9.3. The application of our algorithm to path algebras and Gröbner basis is presented in Section 9.4. We briefly compare our algorithm to previously known algorithms for determining similar properties of free associative algebras in Section 9.5.

9.1 Definitions

Let V and Σ be disjoint alphabets, and let 0 be a symbol not in $V \cup \Sigma$. A *quiver* $G = (V, A)$ is a labeled, directed multigraph with loops; that is, V is a set of *vertices*, and $A \subseteq V \times V \times \Sigma$ is a set

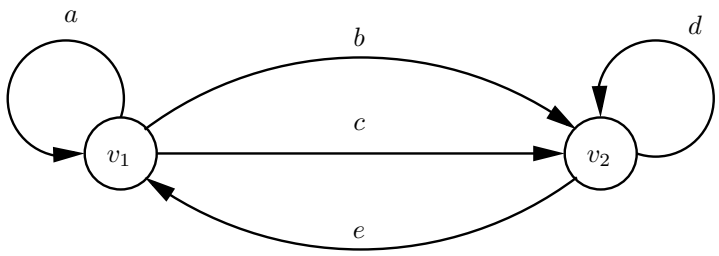


Figure 9.1: A sample quiver $\hat{G} = (\hat{V}, \hat{A})$.

of triples such that every $\sigma \in \Sigma$ occurs exactly once in the third component. Each triple in A is called an *arrow*. The *label* of a vertex is the vertex itself, while the label of an arrow (u, v, σ) is its third component σ. As an example, suppose $\hat{V} = \{v_1, v_2\}$ and $\Sigma = \{a, b, c, d, e\}$. Then one possible quiver $\hat{G} = (\hat{V}, \hat{A})$ is given by the arrows

$$\hat{A} \quad = \quad \{(v_1, v_1, a), (v_1, v_2, b), (v_1, v_2, c), (v_2, v_2, d), (v_2, v_1, e)\}.$$

Small quivers are more conveniently presented in a drawing; see Figure 9.1 for a drawing of $\hat{G}$. Clearly the concept of a quiver generalizes the standard concept of a directed graph.

The set Γ of *walks* in a quiver $G = (V, A)$ consists of three kinds of elements:

1. 0, having *length* $\textsc{Length}(0) = -\infty$;
2. Every element $v \in V$, each v having *length* $\textsc{Length}(v) = 0$; and
3. A sequence (string) $\sigma_1\sigma_2 \cdots \sigma_k \in \Sigma^+$, of *length* $\textsc{Length}(\sigma_1\sigma_2 \cdots \sigma_k) = k$, such that, for each i, $1 \leq i \leq k-1$, where $(u_i, v_i, \sigma_i), (u_{i+1}, v_{i+1}, \sigma_{i+1}) \in A$, we have $v_i = u_{i+1}$. Such a sequence is called a *nontrivial walk*.

In particular, $\Gamma \subset (V \cup \Sigma \cup \{0\})^+$. Except for the 0 walk, this definition corresponds precisely to the standard definition of a walk in a graph or multigraph (see West [97]). The following strings are walks in our sample quiver $\hat{G}$:

$$0 \qquad v_2 \qquad bdde \qquad eaabec.$$

The following strings are **not** walks in $\hat{G}$:

$$00 \qquad v_1v_1 \qquad bdc \qquad eea.$$

Table 9.1: Sources and targets for walks

w	$s(w)$	$t(w)$
0	0	0
$v \in V$	v	v
$\sigma_1\sigma_2\cdots\sigma_k \in \Sigma^+$	$s(\sigma_1)$	$t(\sigma_k)$

Table 9.2: Multiplication of non-zero walks

		$w_1 \cdot w_2$ if $t(w_1) = s(w_2)$	$w_1 \cdot w_2$ if $t(w_1) \neq s(w_2)$
$w_1 \in V$	$w_2 \in V$	w_1	0
$w_1 \in V$	$w_2 \notin V$	w_2	0
$w_1 \notin V$	$w_2 \in V$	w_1	0
$w_1 \notin V$	$w_2 \notin V$	w_1w_2	0

If $(u, v, \sigma) \in A$, then u is the *source* $s(\sigma)$ of σ, and v is the *target* $t(\sigma)$ of σ. For any walk w, we define its *source* $s(w)$ and *target* $t(w)$ as given in Table 9.1. We see that, except for the 0 walk, the source and target always correspond to the initial vertex and the terminal vertex of the walk, respectively.

We have arrived at the juncture of constructing from Γ a semigroup with 0. Clearly, 0 should be a zero; for all $w \in \Gamma$, we have $0 \cdot w = w \cdot 0 = 0$. If $w_1, w_2 \in \Gamma - \{0\}$, then we have eight cases for $w_1 \cdot w_2$, based on whether the walks are vertices or nontrivial walks and on whether the target of w_1 equals the source of w_2. These cases are summarized in Table 9.2. In the third column of the last line of the table, w_1w_2 is simply the concatenation of two strings. The reader can readily verify that Γ is closed under multiplication and that multiplication is associative. We have the following proposition.

Proposition 9.1.1. *The set of walks Γ is a semigroup with 0 under the multiplication defined.*

We also see that this definition of multiplication allows us to express any string in $(V \cup \Sigma \cup \{0\})^+$ as a unique walk by inserting a multiplication between every pair of adjacent symbols. In $\hat{G}$ for example, we compute $v_1av_2v_2be = v_1 \cdot a \cdot v_2 \cdot v_2 \cdot b \cdot e = abe \in \Gamma$.

The following standard definitions from semigroup theory can be found in Higgins [53] or Howie [56]. Let S be a semigroup with 0. A *(two-sided) ideal* in S is a nonempty subset I satisfying $S \cdot I = I \cdot S = I$. Clearly $0 \in I$. The factor semigroup S/I of S by an ideal I consists of the set

$\{I\} \cup \{\{y\} \mid y \in S - I\}$, where in the factor semigroup all elements of I have been identified as a single element. The obvious surjective morphism from S to S/I is the *Rees morphism*. If $X \subset S$, then the *ideal* $\langle X \rangle$ *generated by* X is the intersection of all ideals of S containing X. Intuitively, $\langle X \rangle - \{0\}$ consists of every element $\tau \in S - \{0\}$ that can be expressed as a product $\tau = \prod_{i=1}^{k} \alpha_i$, where $1 \leq k \leq 3$ and where at least one $\alpha_i \in X$. See [74, 88] for research related to such factor semigroups.

In the context of Γ, we are especially interested in a finite generating set $X \subset \Gamma$. A walk x is a *subwalk* of a walk w if w can be expressed as a product $w = \prod_{i=1}^{k} \alpha_i$ where $1 \leq k \leq 3$ and where some $\alpha_i = x$. Then $\langle X \rangle - \{0\}$ consists of every element of $\Gamma - \{0\}$ that has some element of X as a subwalk. Clearly, if $x, y \in X - \{0\}$, $x \neq y$, and x is a subwalk of y, then $\langle X \rangle = \langle X - \{y\} \rangle$, as any walk having y as a subwalk also has x as a subwalk. For any $X \subset \Gamma$, $\textsc{Reduce}(X)$, the *reduced set of generators for* $\langle X \rangle$, is the unique subset of $X - \{0\}$ such that, for any $x, y \in \textsc{Reduce}(X)$, neither x nor y is a subwalk of the other; the reader may readily verify that $\textsc{Reduce}(X)$ is well-defined. If $X = \textsc{Reduce}(X)$, then we say that X is *reduced*.

The *length* of a finite set $X \subset \Gamma - \{0\}$ is

$$\textsc{Length}(X) \quad = \quad \sum_{x \in X} \textsc{Length}(x).$$

Using dynamic dictionary matching techniques (see Amir et al. [6]), we can show the following.

Theorem 9.1.2. *Let G be a quiver and Γ the corresponding set of walks. Let $X \subset \Gamma - \{0\}$ be a finite set, and let $X_0 = X \cap V$ be the set of vertices in X. Finally, let*

$$Z \quad = \quad \{z \in \Sigma \mid (x, y, z) \text{ or } (y, x, z) \in A \text{ for some } x \in X_0\},$$

the set of arcs incident to some vertex in X, and let $Y = (X - X_0) \cup Z$. Then $\textsc{Reduce}(Y)$ can be constructed in $O(\textsc{Length}(X))$ time. Moreover, $\textsc{Reduce}(X) = \textsc{Reduce}(Y) \cup X_0 - Z$ and can be constructed in $O(\textsc{Length}(X))$ time.

Proof. First note that $|X_0|$ and $|Z|$ are constants. Hence we can construct X_0 and Z from G and X in $O(\textsc{Length}(X))$ time via a linear scan of X for vertices followed by a scan of G for arcs incident to those vertices. The set Y contains no vertices by definition. Clearly, $\textsc{Length}(Y)$ is within an additive constant of $\textsc{Length}(X)$. Moreover, it is clear that $\langle X \rangle = \langle Y \rangle \cup X_0$. Since every element of $\langle Y \rangle$ that is a superwalk of an element of X_0 is also a superwalk of an element of Z, we have

$\textsc{Reduce}(X) = \textsc{Reduce}(Y) - Z \cup X_0$. It remains to show that $\textsc{Reduce}(Y)$ can be constructed in $O(\textsc{Length}(X)) = O(\textsc{Length}(Y))$ time.

Let $Y = \{y_1, \ldots, y_n\}$ be the walks in the finite set Y. Let $\hat{Y}_0 = \emptyset$. For $1 \leq i \leq n$, construct $\hat{Y}_i$ from $\hat{Y}_{i-1}$ using one of the following two cases:

1. If y_i contains an element of $\hat{Y}_{i-1}$ as a subwalk, then $\hat{Y}_i = \hat{Y}_{i-1}$;
2. Otherwise, if W_i is the set of walks from $\hat{Y}_{i-1}$ containing y_i as a subwalk, then $\hat{Y}_i = (\hat{Y}_{i-1} - W_i) \cup \{y_i\}$.

We claim that every set $\hat{Y}_i$ is reduced. Clearly, $\hat{Y}_0$ is reduced. For purposes of induction, assume that $\hat{Y}_{i-1}$ is reduced. In Case 1, the element y_i is a superwalk of some element in $\hat{Y}_{i-1}$, so $\langle \hat{Y}_{i-1} \rangle = \langle \hat{Y}_{i-1} \cup \{y_i\} \rangle$. By our assumption, $\hat{Y}_i = \hat{Y}_{i-1}$ is reduced. In Case 2, the elements in W_i are superwalks of y_i, and y_i is not a subwalk of any element in $\hat{Y}_{i-1} - W_i$. Since we are not in Case 1, no element in $\hat{Y}_{i-1}$ is a subwalk of y_i. Because $\hat{Y}_{i-1}$ is reduced, it follows that $\hat{Y}_i = (\hat{Y}_{i-1} - W_i) \cup \{y_i\}$ is also reduced.

The dynamic dictionary matching of Amir et al. [6] is used to obtain the $O(\textsc{Length}(Y))$ processing time. Because $\hat{Y}_i$ is always reduced, no walk is a subwalk of another walk. This fact implies that the dynamic tree data structure augmenting the suffix tree is unnecessary. Removing the dynamic tree data structure removes the $\log |D_i|$ term from their analysis.

For Case 1, only the dynamic dictionary $\textsc{Search}$ procedure, requiring $O(\textsc{Length}(y_i))$ time, is performed. Case 2 requires in addition $O(\textsc{Length}(W_i))$ time for finding superwalks in the dynamic dictionary; $O(\textsc{Length}(W_i))$ time for removing superwalks from the dynamic dictionary; and $O(\textsc{Length}(y_i))$ time for inserting y_i in the dynamic dictionary. The sum of all $O(\textsc{Length}(y_i))$ is $O(\textsc{Length}(Y))$. The sum of all $O(\textsc{Length}(W_i))$ is $O(\textsc{Length}(Y))$, since, once elements in W_i are removed, they never reappear. The total time to compute $\textsc{Reduce}(Y)$ is $O(\textsc{Length}(Y))$.

□

Theorem 9.1.3. *Using the notation from Theorem 9.1.2, there exists a surjective homomorphism* $\rho : \Gamma/\langle Y \rangle \to \Gamma/\langle X \rangle$. *Moreover,* $\Gamma/\langle X \rangle$ *is finite if and only if* $\Gamma/\langle Y \rangle$ *is finite. Finally,* $|\Gamma/\langle X \rangle| = |\Gamma/\langle Y \rangle| - |X_0|$.

Proof. Define $\rho : \Gamma/\langle Y \rangle \to \Gamma/\langle X \rangle$ by $\rho(\{\langle Y \rangle\}) = \{\langle X \rangle\}$, $\rho(\{v\}) = \{\langle X \rangle\}$, if v is in X_0 and $\rho(\{w\}) = \{w\}$ otherwise. Clearly, ρ is a surjective homomorphism. From ρ, we see that $|\Gamma/\langle X \rangle| =$

$|\Gamma/\langle Y\rangle| - |X_0|$ because we are mapping $\{v\}$ in $|\Gamma/\langle Y\rangle|$ for all $v \in |X_0|$ to $\{\langle X\rangle\}$ and mapping the remaining elements directly. Finally, because X_0 is a finite set, $\Gamma/\langle X\rangle$ is finite if and only if $\Gamma/\langle Y\rangle$ is finite. □

Henceforth, we assume, without loss of generality, that any generating set is reduced and contains no vertices.

9.2 Finiteness of a Factor Semigroup

In this section, we address the problem of determining whether a factor semigroup is finite or infinite. We fix a quiver $G = (V, A)$ together with the notation of Section 9.1, including the set Γ of walks in G. In addition, we fix a finite reduced set of generators $X \subset \Gamma$ containing no vertices. Let

$$\begin{aligned} H &= \Gamma/\langle X\rangle \\ &= \{\langle X\rangle\} \cup \{\{y\} \mid y \in \Gamma - \langle X\rangle\} \end{aligned}$$

be the factor semigroup. To determine whether H is finite or infinite, it suffices to determine whether $\tilde{H} = \Gamma - \langle X\rangle - V$ is finite or infinite. Note that $\tilde{H}$ is the set of nontrivial walks that are **not** superwalks of any element of X.

Since walks are strings, we develop our solutions in the context of automata theory (see, for example, [55] or [91]). Our goal is to construct a deterministic finite automaton (DFA) accepting the strings that are representatives for the walks in $\tilde{H}$. We do this by first constructing a DFA M accepting the strings in $\langle X\rangle - \{0\}$ and then "complementing" the DFA. The approach for constructing M is inspired by the Aho-Corasick pattern matching algorithm [2].

Given the set of walks X, the Aho-Corasick algorithm constructs a pattern-matching automaton

$$C = (Q, \Sigma, \delta, h, q_0, F)$$

that recognizes exactly the nontrivial walks in $\langle X\rangle$. Here Q is a set of states, $q_0 \in Q$ is the initial state, Σ is the alphabet of arrow labels, $\delta : Q \times \Sigma \to Q \cup \{\texttt{fail}\}$ is a forward transition function, $h : Q \to Q$ is a failure transition function, and $F \subseteq Q$ is a set of accepting states. The algorithm in Figure 9.2, using the automaton C, recognizes nontrivial walks in $\langle X\rangle$. We summarize the construction of C here, highlighting the steps important for our results.

ACCEPT(M, w) Determine whether a non-trivial walk is accepted by C.

INPUT: A non-trivial walk $w = \sigma_1\sigma_2\cdots\sigma_n$ and a machine $C = (Q, \Sigma, \delta, h, q_0, F)$
OUTPUT: "Yes" if w is accepted by C, "No" if not.

```
1  state ← q_0
2  n → LENGTH(w)
3  for i ← 1 to n
4  do while δ(state, σ_i) = fail
5           do state ← h(state)
6     state → δ(state, σ_i)
7     if state ∈ F
8     then return "Yes"
9  return "No"
```

Figure 9.2: Algorithm for recognizing nontrivial walks in $\langle X\rangle$.

The first step in the Aho-Corasick algorithm is to construct a data structure called a trie [4] encoding all the walks in X. A trie for X is a rooted tree in which each arc is labeled with an element of Σ and each path from the root to a leaf is labeled with a string in X gotten by concatenating the labels of the arcs on the path. Because our set is reduced, we do not require and do not use the extra terminating symbol \$. Each node v of the trie is uniquely identified by the walk $w(v) \in \Gamma$ that is gotten by concatenating the labels of the arcs on the path from the root to v.

The nodes of the trie are the states Q of C, the root is q_0, and the leaves of the trie are the accepting states F. The forward transition function δ is given by the arcs in the trie. If there is an arc in the trie from state q to state q' labeled σ, then $\delta(q, \sigma) = q'$. When there is no arc out of state q labeled σ, then we define

$$\delta(q,\sigma) \quad = \quad \begin{cases} q_0 & \text{if } q = q_0; \\ \texttt{fail} & \text{otherwise.} \end{cases}$$

The failure function h for a node $q \in Q$ is defined as follows. Define $h(q_0) = q_0$. Now suppose that $q \in Q - \{q_0\}$. Recall that $w(q)$ is the concatenation of the labels of the arcs on the path from q_0 to q. Let x be the longest proper suffix of $w(q)$ that is a prefix of some walk in X. Then $x = w(q')$, for some $q' \in Q$. Define $h(q) = q'$. Aho and Corasick [2] provide the details for constructing h in $O(\text{LENGTH}(X))$ time. See Figure 9.3 for a trie encoded from the set of walks $\{abd, bdeb, bdec\}$

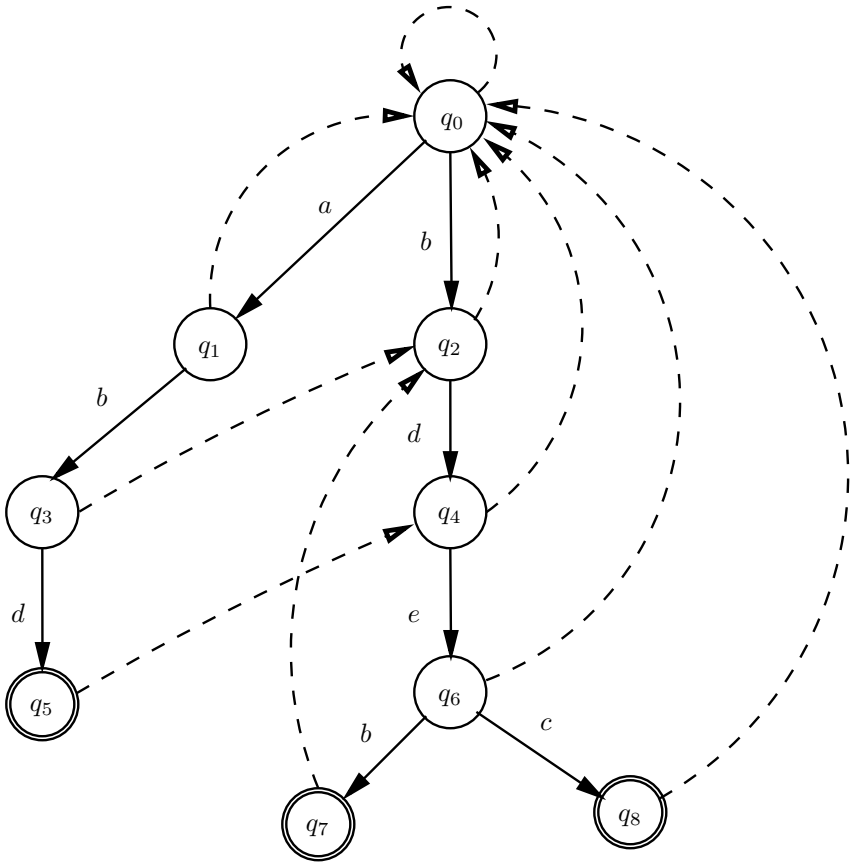


Figure 9.3: The Aho-Corasick automaton for the walks $abd, bdeb, bdec$.

found in the quiver of Figure 9.1. The dashed arcs in Figure 9.3 are failure transitions. The entire construction of C takes $O(\textsc{Length}(X))$ time.

We now construct from C a DFA $M = (Q', \Sigma, \delta', q_0, F')$ accepting the nontrivial walks in $\langle X \rangle$, where Q' is the set of states and $\delta' : Q' \times \Sigma \to Q'$ is the transition function. In Hopcroft and Ullman [55], $\delta' : Q' \times \Sigma \to Q'$ is required to be a total function. However, we take advantage of the quiver $G = (V, A)$ to eliminate outgoing transitions that cannot happen if M has only nontrivial walks in Γ as input. Every state $q \in Q - \{q_0\}$ has an associated target $t(q) \in V$ that indicates the ending vertex of all walks terminating at q; we require that all outgoing transitions σ from q have source $s(\sigma) = t(q)$. As a consequence, δ' is a partial function, but there is a path in M labeled by every nontrivial walk in Γ.

First we need to derive Q' from Q. Define $Q_0 = \{q_0\}$ and $Q_1 = Q - F - \{q_0\}$; note that Q is the disjoint union of Q_0, Q_1, and F. Observe that once C has reached F, it accepts the walk and so may be thought of as staying in an accepting state while it processes the remainder of the walk. Let $P = \{\sigma \in \Sigma \mid \delta(q_0, \sigma) = q_0\}$, the set consisting of every symbol σ for which there is no element of X having σ as a prefix. Define $Q' = Q_0 \cup Q_1 \cup P \cup V$, and $F' = V$. Intuitively, we have replaced the accepting states of C by the vertices of G, and added P so that we can eliminate all transitions that go to q_0.

We now utilize δ to obtain δ'. First, suppose $\delta(q, \sigma) \neq \texttt{fail}$, where $q \neq q_0$. If $\delta(q, \sigma) \in Q_1$, then define $\delta'(q, \sigma) = \delta(q, \sigma)$. If $\delta(q, \sigma) \in F$, then define $\delta'(q, \sigma) = t(\sigma)$. Second, define $\delta'(q_0, \sigma) = \sigma$ if $\sigma \in P$, and $\delta'(q_0, \sigma) = \delta(q_0, \sigma)$ otherwise. Third, for every $q \in V$ and $\sigma \in \Sigma$, where $s(\sigma) = q$, define $\delta'(q, \sigma) = t(\sigma)$; once a final state is reached, the automaton must remain in a final state. Finally, suppose $q \in Q - Q_1$ and $\sigma \in \Sigma$ are such that $\delta(q, \sigma) = \texttt{fail}$. Let x be the longest proper suffix of $w(q) \cdot \sigma$ that is a prefix of some walk in X. Then $x = w(q')$, for some $q' \in Q$. Define $\delta'(q, \sigma) = q'$. This completes the definition of δ'.

To obtain M from C in $O(\textsc{Length}(X))$ time, we employ a breadth-first search on the trie and use the following observation.

Proposition 9.2.1. *Suppose $q \in Q - \{q_0\}$ and $\sigma \in \Sigma$ are such that $\delta(q, \sigma) = \texttt{fail}$ and $h(q) = q'$. Then $|w(q)| > |w(q')|$, and $\delta'(q, \sigma) = \delta'(q', \sigma)$.*

The resulting algorithm Extend can be found in Figure 9.4. In Extend, we assume the existence of a queue object `queue` with methods `enqueue`, `dequeue`, and `empty` to add to the queue, remove from the queue, and test the queue for emptiness, respectively. It is clear that Extend executes in $O(\textsc{Length}(X))$ time, and its correctness follows by Proposition 9.2.1. We obtain the following theorem.

Theorem 9.2.2. *The equivalent DFA M can be constructed from C in $O(\textsc{Length}(X))$ time.*

Summarizing the discussion so far, we have the following theorem.

Theorem 9.2.3. *A DFA accepting the nontrivial walks in $\langle X \rangle$ can be constructed in $O(\textsc{Length}(X))$ time.*

Figure 9.5 shows the state diagram of the DFA obtained by converting the automaton in Figure 9.3 using the above process.

EXTEND(C) Extend Aho-Corasick machine C to a DFA.

INPUT: Aho-Corasick automaton $C = (Q, \Sigma, \delta, h, q_0, F)$
OUTPUT: Transition function δ' for the equivalent DFA DFA.

```
1   for symbol ∈ Σ
2       do destination ← δ(q_0, symbol)
3          if destination = fail
4            then δ'(q_0, symbol) ← symbol
5            else if destination ∈ Q_1
6            then δ'(q_0, symbol) ← destination
7                 queue.enqueue(destination)
8            else δ'(q_0, symbol) ← t(destination)
9   while not queue.empty
10        do state ← queue.dequeue
11           for symbol ∈ Σ
12               do destination ← δ(state, symbol)
13                  if destination = fail
14                    then δ'(state, symbol) ← δ(h(state), symbol)
15                    else if destination ∈ Q_1
16                    then δ'(state, symbol) ← destination
17                         queue.enqueue(destination)
18                    else δ'(q, symbol) ← t(destination)
```

Figure 9.4: Algorithm for extending C to a DFA

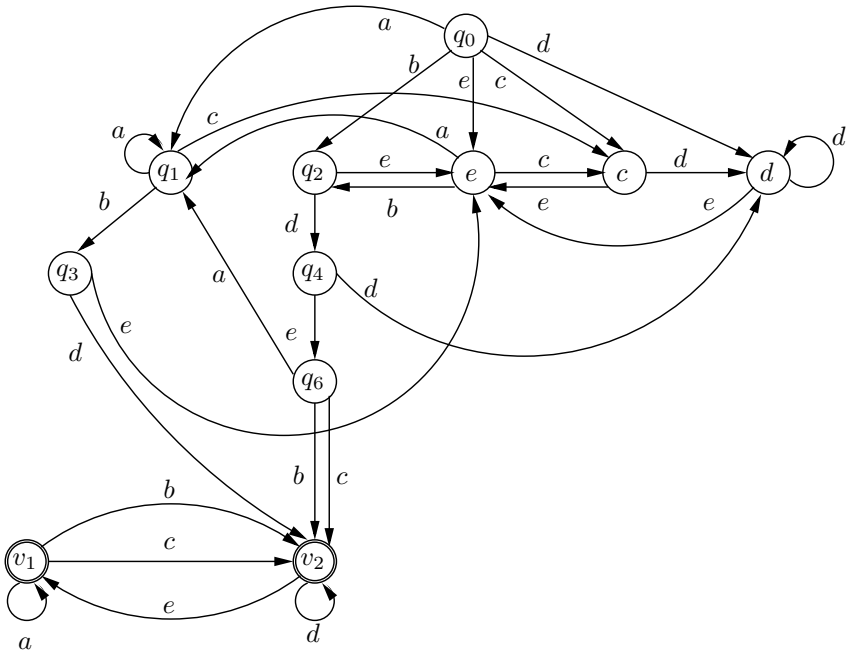


Figure 9.5: A DFA accepting the nontrivial walks in $\langle abd, bdeb, bdec \rangle$.

The DFA $\overline{M} = (Q', \Sigma, \delta', q_0, Q' - F) = (Q', \Sigma, \delta', q_0, Q_0 \cup Q_1 \cup P)$ is the complement of M. The complement of a DFA accepts the complement of the language accepted by the DFA (see [91, Theorem 6.3.3]). Hence, $\overline{M}$ accepts walks that are not nontrivial walks in $\langle X \rangle$. The next result allows us to determine if there are a finite number of nontrivial walks not in $\langle X \rangle$.

Theorem 9.2.4. *Let W be the set of nontrivial walks not in $\langle X \rangle$. Let $\overline{M} = (Q', \Sigma, \delta', q_0, Q' - F')$ be the DFA accepting W constructed using the process described above. Let $k = |Q'|$. Let S be the state diagram of $\overline{M}$. Then the set W is finite if and only if the subgraph S' of S induced by $Q' - F' = Q_0 \cup Q_1 \cup P$ is a directed acyclic graph.*

Proof. By the construction of $\overline{M}$, there are no transitions from its non-accepting states F' to its accepting states $Q' - F'$. Hence W is exactly the set of labels of paths of length ≥ 1 in S'. Also, every state in S' is reachable by a path from the initial state q_0 that is wholly contained in S'.

First, assume that W is finite. To obtain a contradiction, suppose that S' contains a cycle. Let q be some state in that cycle, let x be the label of any path from q_0 to q, and let y be the label on the cycle starting at q. Then xy^i is the label of a path in S' for every $i \geq 1$. Each such xy^i is in W, implying that W is infinite. This contradiction to the finiteness of W shows that S does not contain a cycle. Hence S' is a directed acyclic graph.

Now, assume that S' is a directed acyclic graph. As S' contains no cycles, the number of paths in S' starting at q_0 must be bounded. The cardinality of W is bounded above by that number of paths. Hence, W is finite. □

If we examine the DFA in Figure 9.5, we see that the corresponding S' is induced by $\{q_0, q_1, q_2, q_3, q_4, q_6, c, d, e\}$. As that graph is clearly not acyclic, we know that $\tilde{H}$ is an infinite set.

Theorem 9.2.5. *The finiteness of $\tilde{H}$ can be determined in $O(\text{LENGTH}(X))$ time.*

Proof. By Theorem 9.1.2, we can find $\text{REDUCE}(X)$ in $O(\text{LENGTH}(X))$ time. We have demonstrated a method for constructing a DFA M for accepting the nontrivial walks in $\langle X \rangle$ in $O(\text{LENGTH}(X))$ time. The complement $\overline{M}$ of M can be computed in $O(\text{LENGTH}(X))$ by visiting each node and toggling its accepting state. By Theorem 9.2.4, we can determine finiteness by determining if the state diagram S' induced by the accepting states of $\overline{M}$ is acyclic. Using topological sorting algorithms [19], determining if S' is acyclic can be accomplished in $O(\text{LENGTH}(X))$ time. The theorem follows immediately. □

We can determine the number of elements in $\tilde{H}$ in $O(\text{LENGTH}(X))$ time as well, given that $\tilde{H}$ is finite. Let $M = (Q', \Sigma, \delta', q_0, F')$ be the DFA constructed above accepting the nontrivial walks in $\langle X \rangle$. Let $\overline{M}$ be the complement of M and let S' be the state diagram induced by the accepting states of $\overline{M}$. Let n be the number of nontrivial walks accepted by $\overline{M}$. Clearly, the number of elements in $\tilde{H}$ is n.

Let n_q be the number of distinct paths ending at state q in S'. Then, $n = \sum_{q \in S' - \{q_0\}} n_q$. Let I_q be the set of incoming edges to q in S' and let $s(e)$ be the source state of edge e in S'. The number of distinct paths ending at state q is $n_q = \sum_{e \in I_q} n_{s(e)}$.

Thus, n_q is computed in the following manner. Order the states of S' using topological sort. Initialize n_{q_0} to 1 and n_q to 0 for the remaining states q. Visit each state in topologically sorted order. When visiting state q, for each outgoing edge e from q, add n_q to $n_{t(e)}$, where $t(e)$ is the target of edge e. This works because when q is visited, the contribution from every incoming edge to q has been incorporated into n_q. Thus, n_q has been computed by the time q is reached. The value n can be computed by summing each n_q as q is visited. Because topological sort is an $O(\text{LENGTH}(X))$ algorithm and we visit each node in S' exactly once, n is computed in $O(\text{LENGTH}(X))$ time. We obtain the following theorem.

Theorem 9.2.6. *The number of elements in $\tilde{H}$ can be computed in $O(\text{LENGTH}(X))$ time.*

9.3 Enumeration of a Factor Semigroup

We turn to enumerating elements in factor semigroups. As in the previous sections, let $G = (V, A)$ be a quiver, let Γ be the set of walks in G, and let $X \subset \Gamma$ be a finite subset of walks. In addition, let M be the DFA accepting nontrivial walks in $\langle X \rangle$ described in Section 9.2, and let $\overline{M}$ be the complemented DFA.

Let S' be the state diagram induced by the accepting states of $\overline{M}$. Recall that every path in S' corresponds to a nontrivial walk in $\tilde{H}$. Thus, if we enumerate every path in S', we enumerate every nontrivial walk in $\tilde{H}$.

Let w_q be the set of walks accepted by state q. Clearly, $w_q = \bigcup_{e \in I_q} w_{s(e)}\sigma_e$ where I_q is the set of incoming edges to q, σ_e is the label of e, $s(e)$ is the source of e, and $w_x\sigma = \{w \cdot \sigma \mid w \in w_x\}$ for all states x and all $\sigma \in \Sigma$.

If $\tilde{H}$ is finite, then we can use the following algorithm to enumerate nontrivial walks. Initialize

w_q to $\emptyset$ for each q in S'. Topologically sort the states in S'. At state q_0, for each outgoing edge e, set $w_{t(e)} = \{\sigma_e\}$ where $t(e)$ is the target state of e and σ_e is the label of e. Visit the remaining states in sorted order. When visiting state q, first output w_q. Next, for each outgoing edge e from q, union the set $w_q\sigma_e$ with $w_{t(e)}$. Continue processing with the next state in sorted order. Clearly, the algorithm enumerates every nontrivial walk in $\tilde{H}$.

Theorem 9.3.1. *There exists an algorithm to enumerate the elements of $\tilde{H}$ in $O(\textsc{Length}(\tilde{H}))$ time.*

9.4 Applications

The results in the previous sections have applications in other areas of computer algebra, in particular, in the Gröbner basis theory of path algebras [38]. Let $G = (V, A)$ be a quiver, let Γ be the non-zero walks of G, and let $\mathbb{K}$ be a field. A *path algebra* $\mathbb{K}\Gamma = \{\sum_{w\in\Gamma} \alpha_w w \mid \text{a finite number of } \alpha_w \text{ are non-zero } \}$ is an associative algebra with one; the one is equal to $\sum_{v\in V} 1_{\mathbb{K}} v$. Addition in $\mathbb{K}\Gamma$ is formal addition, and multiplication is defined by $(\alpha w)(\beta x) = (\alpha\beta)(w \cdot x)$ for all $\alpha, \beta \in \mathbb{K}$ and $w, x \in \Gamma$ extended to all elements via distributing multiplication across addition. Also, $\alpha 0$ is zero in $\mathbb{K}\Gamma$ for all $\alpha \in \mathbb{K}$.

Fix a well order $\prec$ on the walks in Γ. The order $\prec$ is an *admissible order* if the following two properties hold for $a, b, u, v \in \Gamma$ where uav and ubv are both non-zero: $a \prec uav$ or $a = uav$; if $a \prec b$ then $uav \prec ubv$. For the remainder of this section, assume $\prec$ is an admissible order.

The *tip* $\text{Tip}(x) = w$ of an element $x \in \mathbb{K}\Gamma$ is the maximal walk w with respect to $\prec$ such that w has a non-zero coefficient in x. If $X \subseteq \mathbb{K}\Gamma$, then $\text{Tip}(X) = \{\text{Tip}(x) \mid x \in X\}$ is the set of tips of elements in X and $\text{Nontip}(X) = \Gamma - \text{Tip}(X) - \{0\}$ is the set of non-zero walks that are not tips of some element in X.

Let I be an ideal in $\mathbb{K}\Gamma$ and let $\mathcal{G} \subset I$. If $\langle \text{Tip}(\mathcal{G})\rangle = \langle \text{Tip}(I)\rangle$ then $\mathcal{G}$ is a *Gröbner basis* for I with respect to $\prec$. Computing arbitrary Gröbner bases is an undecidable problem; however, finite Gröbner bases do exist for some i for which $\mathbb{K}\Gamma/I$ is infinite dimensional and are guaranteed to exist when $\mathbb{K}\Gamma/I$ is finite dimensional.

Using Gröbner basis theory, elements in $\mathbb{K}\Gamma/I$ are viewed as elements in the span of $\text{Nontip}(I)$. From the definition, $\text{Nontip}(I) \cup \{0\} \equiv \Gamma/\langle \text{Tip}(I)\rangle$ in a natural way. If $\mathcal{G}$ is a Gröbner basis for I, then $\text{Nontip}(I) \cup \{0\} \equiv \Gamma/\langle \text{Tip}(\mathcal{G})\rangle$. Thus, if a finite Gröbner basis for I is provided, we obtain the

following theorem by applying the results presented earlier.

Theorem 9.4.1. *Let $\mathbb{K}\Gamma$ be a path algebra, let I be a $\mathbb{K}\Gamma$-ideal, and let $\mathcal{G}$ be a finite Gröbner basis for I under some admissible ordering $\prec$. Then there exist algorithms to determine whether $\mathbb{K}\Gamma/I$ is finite dimensional in $O(\textsc{Length}(\text{Tip}(\mathcal{G})))$ time, to compute $\text{Dim}_{\mathbb{K}}\mathbb{K}\Gamma/I$ in $O(\textsc{Length}(\text{Tip}(\mathcal{G})))$ time, and to enumerate $\text{Nontip}(I)$ in $O(\textsc{Length}(\text{Nontip}(I)))$ time.*

In addition to determining these properties, the DFA constructed in Section 9.2 is also used for pattern matching in the reduction step for computing normal forms with respect to $\mathcal{G}$.

9.5 Related Algorithms

Gateva-Ivanova and Latyshev [43] present alternative algorithms for determining the same properties of associative algebras. Their algorithms depend on the construction of an Ufnarovskij graph for $\langle \text{Tip}(I) \rangle$. Let $\mathcal{G}$ be a Gröbner basis for I, and let l be the maximal length of elements in $\textsc{Reduce}(\text{Tip}(\mathcal{G}))$. The number of nodes in the Ufnarovskij graph is exponential in l. Furthermore, determining edges for the Ufnarovskij graph requires several reductions to be performed. In contrast, our algorithms are linear in the size of the input and no reductions are required to construct the DFA. Hence, the construction of Ufnarovskij graphs is significantly more expensive in general than the approach we have presented in this chapter.

Chapter 10

Conclusions and Future Directions

The goal of this research is to provide practical computer tools for the classification of algebraic structures. Recommended choices of previously known algorithms to employ in computer algebra systems are provided as a result. Several new algorithms in different areas of noncommutative algebra are discovered and demonstrated to be efficient in practice. This chapter lists the contributions made in this dissertation in Section 10.1 and presents directions for future work in Section 10.2.

10.1 Contributions

The contributions of the research are considered separately in the areas of algebra decomposition, Drinfel'd double, structure of modules, and properties of path algebras.

10.1.1 Algebra Decomposition

The primary contributions of the research in algebra decomposition are a survey and comparative analysis of previously known algorithms, and recommendations for algorithms to employ as a result of experimental testing. The analysis and experimentation of algorithms, particularly for computing primitive central idempotents in $\mathbb{F}_q$-algebras, highlights two dominating costs: computing minimal polynomials of algebra elements and calculating bases for intermediate polynomials. The algorithm of Davis' which does not perform either operation, generally performed better in our experiments. A recommended choice of algorithms for **ALGEBRA DECOMPOSITION** and **SEMISIMPLE**

ALGEBRA DECOMPOSITION summarize the chapter on algebra decomposition.

A secondary contribution is the formulation of each algorithm in an encoding neutral setting. This provides easy implementation of the algorithms in computer algebra systems such as GAP [42] and Magma [12]. The presentation also demonstrates when conversions to a specific encoding are necessary.

10.1.2 Drinfel'd Double

The primary contribution is the first known algorithm for calculating the Drinfel'd double of a finite dimensional Hopf algebra is described. The algorithm is made efficient by employing simple caching techniques. The complexity of the algorithm is shown to be $O(d^7)$ Hopf algebra operations. In experiments, the algorithm with caching employed is estimated to be closer to $O(d^5)$. In the case of group algebras, a specialized version of the algorithm is shown use $O(d^3)$ group operations, which is very efficient.

10.1.3 Structure of Modules

A new algorithm for constructing endomorphism rings of modules over path algebras is the primary contribution in this area. The algorithm is shown experimentally to have better performance characteristics than the algorithm used in [12]. A secondary contribution is a review of Parker's MeatAxe [80]. Pseudocode for the basic algorithm and improvements by Holt and Rees [54], and by Ivanyos and Lux [60] are presented. Several observations about the implementation of MeatAxe in an encoding neutral environment are made.

10.1.4 Properties of Path Algebras

A new, linear time algorithm for determining whether a quotient of a path algebra is finite or infinite dimensional is presented. With minor extensions, the algorithm also determines the dimension and enumerates a basis of the algebra efficiently. This algorithm improves on previous results, which have exponential complexity.

10.2 Future Directions

10.2.1 Analysis

Analyses for some of the algorithms presented are currently not available. These analyses are necessary for a complete understanding of the available algorithms in noncommutative algebra.

In Chapter 5, the analysis for ExplicitIsomorphism, Eberly's algorithm for computing an isomorphism between a simple algebra A and a full matrix algebra $M_n(D)$ is left for future work. The analysis for RadicalPrimeCIW, the algorithm of Cohen, Ivanyos, and Wales for calculating $J(A)$ is also left undone.

10.2.2 Jacobson Radical

The recent algorithms of Ivanyos [58, 59] for calculating $J(A)$ look very promising. Currently, no pseudocode descriptions or implementations exist for these algorithms. Analyzing the behavior of Ivanyos' algorithms both theoretically and experimentally will determine the practicality of these new methods.

10.2.3 Path Algebras

The sparse encoding used by path algebras and group algebras alleviates the space issues in computer algebra. The quiver defining a path algebra also provides additional information regarding the multiplicative compatibility of path algebra elements during computation. We saw in Section 8.2 that the structure of path algebras contributes to the efficiency of computing endomorphism rings. The structure of path algebras might play an important role in other algebraic algorithms as well.

The description and implementation of the path algebra encoding in this dissertation is straightforward. We took no steps to avoid unnecessary multiplications of path algebra elements (i.e., when the elements are known to have incompatible source and target vertices). We believe that specialized data structures for storing path algebra elements partitioned by source and target vertices might improve the cost computations. The primary challenge is maintaining that other operations are efficient. This is particularly important when writing an element as a linear combination of basis elements, where maintaining basis elements in sorted order is important.

Also of interest is an analysis of Reduce in Section 4.3.1. This algorithm plays an important role

in the cost complexity for the multiplication of path algebra elements. To date, no analysis appears to be available. Users of Gröbner basis techniques assume this cost is not great, and that the space benefits obtained by applying Gröbner bases outweigh the performance costs. Not only would a rigorous analysis help determine what impact reduction via a Gröbner basis has on computations, but it may provide insight for improvements to the basic algorithm.

10.2.4 Hopf Algebras

The area of computational Hopf algebras is very new and the Drinfel'd double algorithm and implementation in Chapter 6 is a very early result in this area.

The classification of Hopf algebras is a very active area of research. Related to the classification problem is searching for *Hopf ideals* in Hopf algebras. Hopf ideals are two-sided ideals with additional properties regarding the comultiplication, counit, and antipode maps. Finding Hopf ideals is an important task for a better understanding of the structure of Hopf algebras.

Another important problem is the efficient computation of R-matrices [15, pg. 123] for Hopf algebras. When a Hopf algebra is quasitriangular, as is the case with Hopf algebras constructed via the Drinfel'd double, the R-matrix contains solutions to the Yang-Baxter equations as well as knot invariants, which are important in knot theory.

REFERENCES

[1] The R statistical package. `http://www.r-project.org`, February 2000. version 1.0.0.

[2] Aho, A. V., and Corasick, M. J. Efficient string matching: an aid to bibliographic search. *Communications of the ACM 18* (1975), 333–340.

[3] Aho, A. V., Hopcroft, J. E., and Ullman, J. D. *The Design and Analysis of Computer Algorithms*. Addison Wesley, 1974.

[4] Aho, A. V., Hopcroft, J. E., and Ullman, J. D. *Data Structures and Algorithms*. Addison-Wesley Series in Computer Science and Information Processing. Addison-Wesley, 1983.

[5] Alperin, J. L. *Local Representation Theory*. Cambridge University Press, Cambridge, 1986.

[6] Amir, A., Farach, M., Galil, Z., Giancarlo, R., and Park, K. Dynamic dictionary matching. *Journal of Computer and System Sciences 49* (1994), 208–222.

[7] Anton, H. *Elementary Linear Algebra*, fifth ed. John Wiley and Sons, 1987.

[8] Bach, E., and Shallit, J. *Algorithmic Number Theory: Efficient Algorithms*, vol. 1. The MIT Press, Cambridge, Massachusetts, 1996.

[9] Becker, T., and Weispfenning, V. *Gröbner bases: A Computational Approach to Commutative Algebra*. No. 141 in Graduate Texts in Mathematics. Springer-Verlag, 1991.

[10] Berlekamp, E. R. Factoring polynomials over large finite fields. *Mathematics of Computation 24* (1970), 713–715.

[11] Blum, L., Schub, M., and Smale, S. On a theory of computation and complexity over the real numbers: NP-completeness, recursive functions and universal machines. *Bulletin (New Series) of the American Mathematical Society 21*, 1 (July 1989), 1–46.

[12] Bosma, W., Cannon, J. J., and Matthews, G. Programming with algebraic structures: design of the magma language. In *Proceedings of the 1994 International Symposium on Symbolic and Algebraic Computation* (1994), M. Giesbrecht, Ed., ACM Press, pp. 52–57.

[13] Brunick, G., Heath, L. S., Struble, C. A., and Green, E. L. Efficient construction of Drinfel'd doubles. In *International Symposium on Symbolic and Algebraic Computation (ISSAC'99)* (1999), pp. 45–52.

[14] Cannon, J., and Souvignier, B. On the computation of conjugacy classes in permutation groups. In *Proceedings of the 1997 International Symposium on Symbolic and Algebraic Computation* (1997), ACM Press, pp. 392–399.

[15] Chari, V., and Pressley, A. *A Guide to Quantum Groups*, first paperback edition ed. Cambridge University Press, 1994.

[16] Chistov, A., Ivanyos, G., and Karpinski, M. Polynomial time algorithms for modules over finite dimensional algebras. In *Proceedings of 1997 International Symposium on Symbolic and Algebraic Computation* (1997), ACM Press, pp. 68–74.

[17] Cohen, A. M., Ivanyos, G., and Wales, D. B. Finding the radical of an algebra of linear transformations. *Journal of Pure and Applied Algebra*, 117 & 118 (1997), 177–193.

[18] Conway, J. H., Curtis, R. T., Norton, S. P., Parker, R. A., and Wilson, R. A. *Atlas of Finite Groups*. Oxford University Press, 1985.

[19] Cormen, T. H., Leiserson, C. E., and Rivest, R. L. *Introduction to Algorithms*. MIT Press, 1990.

[20] Curtis, C. W. *Linear Algebra: An Introductory Approach*. Undergraduate Texts in Mathematics. Springer-Verlag, 1984.

[21] Curtis, C. W., and Reiner, I. *Representation Theory of Finite Groups and Associative Algebras*. Pure and Applied Mathematics. Interscience Publishers, 1962.

[22] Curtis, C. W., and Reiner, I. *Methods of Representation Theory with Applications to Finite Groups and Orders*, vol. 1. John Wiley & Sons, 1981.

[23] Curtis, C. W., and Reiner, I. *Methods of Representation Theory with Applications to Finite Groups and Orders*, vol. 2. John Wiley & Sons, 1981.

[24] Davis, R. A. Idempotent computation over finite fields. *Journal of Symbolic Computation 17* (1994), 237–258.

[25] de Graaf, W. A., and Ivanyos, G. Finding maximal tori and splitting elements in matrix algebras. In *Interactions Between Ring Theory and Representations of Algebras (Proc. Euroconference in Murcia, 1998)* (2000?), F. van Oystaeyen and M. Saorin, Eds., Lecture Notes in Pure and Applied Mathematics, Marcel Dekker. to appear.

[26] de Graaf, W. A., Ivanyos, G., Küronya, A., and Rónyai, L. Computing levi decompositions in lie algebras. *Applicable Algebra in Engineering, Communication, and Computing 8* (1997), 291–303.

[27] Dickson, L. E. *Algebras and Their Arithmetics*. Dover Publications, Inc., New York, NY, 1923.

[28] Dlab, V., and Ringel, C. M. On algebras of finite representation type. *Journal of Algebra 33* (1975), 306–394.

[29] Dlab, V., and Ringel, C. M. *Indecomposable representations of graphs and algebras*, vol. 173 of *Memoirs of the American Mathematical Society*. American Mathematical Society, 1976.

[30] Drinfel'd, V. G. Quantum groups. In *Proceedings, International Congress of Mathematicians* (Berkeley, CA, 1987), pp. 798–820.

[31] Dummit, D. S., and Foote, R. M. *Abstract Algebra*. Prentice Hall, Englewood Cliffs, NJ, 1991.

[32] Eberly, W. *Computations for Algebras and Group Presentations*. PhD thesis, University of Toronto, 1989.

[33] Eberly, W. Decomposition of algebras over finite fields and number fields. *Computational Complexity 1* (1991), 183–210.

[34] Eberly, W. Decompositions of algebras over $\mathbb{R}$ and $\mathbb{C}$. *Computational Complexity 1* (1991), 211–234.

[35] Eberly, W., and Giesbrecht, M. Efficient decomposition of associative algebras. Extended abstract to appear in "Proceedings, International Symposium on Symbolic and Algebraic Computation", Zurich, 1996., May 1996.

[36] Farb, B., and Dennis, R. K. *Noncommutative Algebra*. Graduate Texts in Mathematics. Springer-Verlag, 1993.

[37] Farkas, D. R., Feustel, C., and Green, E. L. Synergy in the theories of Gröbner bases and path algebras. *Canadian Journal of Mathematics 45*, 4 (1993), 727–739.

[38] Feustel, C. D., Green, E. L., Kirkman, E., and Kuzmanovich, J. Constructing projective resolutions. *Communications in Algebra 21* (1993), 1869–1887.

[39] Friedl, K., and Rónyai, L. Polynomial time solutions of some problems in computational algebra. In *ACM Symposium on Theory of Computing* (1985), vol. 17, pp. 153–162.

[40] Fulton, W., and Harris, J. *Representation Theory: A First Course*. Graduate Texts in Mathematics. Springer, 1991.

[41] Gabriel, P. Unzerlegbare darstellungen i. *Manuscripta Mathematica 6* (1972), 71–103.

[42] The GAP Group. *GAP — Groups, Algorithms, and Programming, Version 4.1*. Aachen, St. Andrews, 1999. `http://www-gap.dcs.st-and.ac.uk/~gap`.

[43] Gateva-Ivanova, T., and Latyshev, V. On recognizable properties of associative algebras. *Journal of Symbolic Computation 6* (1988), 371–388.

[44] Gianni, P., Miller, V., and Trager, B. Decomposition of algebras. In *Proceedings of ISSAC* (1988), Springer, pp. 300–308.

[45] Giesbrecht, M. Nearly optimal algorithms for canonical matrix forms. *SIAM Journal of Computing 24*, 5 (October 1995), 948–969.

[46] Gould, M. D. Quantum double finite group algebras and their representations. *Bulletin of the Australian Math Society 48* (1993), 275–301.

[47] Green, E. L. Constructing quantum groups and Hopf algebras from coverings. *Journal of Algebra 176* (1995), 12–33.

[48] Green, E. L. Noncommutative Gröbner bases, and projective resolutions. In *Computational methods for representations of groups and algebras. Papers from the First Euroconference held at the University of Essen.* (Basel, 1999), P. Dräxler, G. O. Michler, and C. M. Ringel, Eds., no. 173 in Progress in Mathematics, Birkhäuser Verlag, pp. 29–60.

[49] Green, E. L. Multiplicative bases, Gröbner bases, and right Gröbner bases. *Journal of Symbolic Computation* (2000). to appear.

[50] Green, E. L., Heath, L. S., and Keller, B. J. Opal: A system for computing noncommutative Gröbner bases (system description). In *Eighth International Conference on Rewriting Techniques and Applications (RTA-97)* (1997), pp. 331–334.

[51] Green, E. L., Heath, L. S., and Struble, C. A. Constructing endomorphism rings via duals. In *Proceedings of the 2000 International Symposium on Symbolic and Algebraic Computation* (2000). to appear.

[52] Green, E. L., Heath, L. S., and the Hopf team. Hopf project site. `http://hal.cs.vt.edu/hopf`, 1999.

[53] Higgins, P. M. *Techniques of Semigroup Theory.* Oxford University Press, Oxford, 1992.

[54] Holt, D. F., and Rees, S. Testing modules for irreducibility. *Journal of the Australian Mathematical Society (Series A) 57* (1994), 1–16.

[55] Hopcroft, J. E., and Ullman, J. D. *Introduction to Automata Theory, Languages, and Computation.* Addison-Wesley Series in Computer Science. Addison-Wesley, 1979.

[56] Howie, J. M. *Fundamentals of Semigroup Theory.* No. 12 in London Mathematical Society Monographs. Oxford University Press, Oxford, 1995.

[57] Hungerford, T. W. *Algebra.* Graduate Texts in Mathematics. Springer-Verlag, 1974.

[58] Ivanyos, G. Finding the radical of matrix algebras using Fitting decompositions. *Journal of Pure and Applied Algebra*, 139 (1999), 159–182.

[59] Ivanyos, G. Fast randomized algorithms for the structure of matrix algebras over finite fields. under review, 2000.

[60] Ivanyos, G., and Lux, K. Treating the exceptional cases of the meataxe. to appear in Experimental Mathematics: `http://www.expmath.org`.

[61] Ivanyos, G., Rónyai, L., and Ágnes Szántó. Decomposition of algebras over $f_q(x_1, ..., x_m)$. *Applicable Algebra in Engineering, Communication, and Computing*, 5 (1994), 71–90.

[62] Jenks, R. D., and Sutor, R. S. *AXIOM. The scientific computation system.* Numerical Algorithms Group, Ltd., Oxford; Springer-Verlag, New York, 1992.

[63] Jimbo, M. Introduction to the Yang-Baxter equation. *International Journal of Modern Physics A 4*, 15 (1989), 3759–3777.

[64] Keller, B. J. *Algorithms and Orders for Finding Noncommutative Gröbner Bases.* PhD thesis, Virginia Tech, 1997.

[65] Landau, S. Factoring polynomials over algebraic number fields. *SIAM Journal of Computing 14*, 1 (February 1985), 184–195.

[66] Loos, R. Computing in algebraic extensions. In *Computer Algebra: Symbolic and Algebraic Computation*, B. Buchberger, G. E. Collins, and R. Loos, Eds., 2 ed. Springer, New York, 1983, pp. 173–187.

[67] Lux, K., Müller, J., and Ringe, M. Peakword condensation and submodule lattices: An application of the meat-axe. *Journal of Symbolic Computation 17* (1994), 529–544.

[68] Majid, S. Hopf algebras for physics at the Planck scale. *Classical and Quantum Gravity 5*, 12 (1988), 1587–606.

[69] Majid, S. Physics for algebraists: Noncommutative and noncocommutative Hopf algebras by a bicrossproduct construction. *Journal of Algebra 130*, 1 (1990), 17–64.

[70] Majid, S. Quasitriangular Hopf algebras and Yang-Baxter equations. *International Journal of Modern Physics A 5*, 1 (1990), 1–91.

[71] Majid, S. *Foundations of Quantum Group Theory*. Cambridge University Press, Cambridge, Great Britain, 1995.

[72] Majid, S. Quantum double for quasi-Hopf algebras. *Letters in Mathematical Physics 45*, 1 (1998), 1–9.

[73] Manin, Y. I. *Topic in noncommutative geometry*. M. B. Porter Lectures. Princeton University Press, Princeton, NJ, 1991.

[74] Márki, L., and Wiegandt, R. On semisimple classes of semigroups with zero. In *Semigroups* (Berlin, 1981), no. 855 in Lecture Notes in Mathematics, Springer-Verlag, pp. 55–72.

[75] Montgomery, S. *Hopf Algebras and Their Actions on Rings*. No. 82 in Regional Conference Series in Mathematics. Conference Board of the Mathematical Sciences, 1993.

[76] Mora, T. An introduction to commutative and non-commutative Gröbner bases. *Theoretical Computer Science*, 134 (1994), 131–173.

[77] Nagao, H., and Tsushima, Y. *Representations of Finite Groups*. Academic Press, Inc., 1989.

[78] Nazarova, L. A. Representations of quivers of infinite type. *Izv. Akad. Nauk. SSSR Ser. Mat 37* (1973), 752–791.

[79] Nill, F., and Szlachanyi, K. Quantum chains of Hopf algebras with quantum double cosymmetry. *Communications in Mathematical Physics 187*, 1 (1997), 159–200.

[80] Parker, R. A. The computer calculation of modular characters (the meat-axe). In *Computational Group Theory*. Academic Press, 1984, pp. 267–274.

[81] Pierce, R. S. *Associative Algebras*. Springer, New York, Heidelberg, Berlin, 1982.

[82] Riedtmann, C. Algebren, darstellungskocher, uberlagerungen and zuruck. *Comm. Math. Helv. 55* (1980), 199–224.

[83] Ringe, M. Meataxe. `http://www.math.rwth-aachen.de/LDFM/homes/MTX/`.

[84] Rónyai, L. Simple algebras are difficult. In 19^{th} *ACM Symposium on Theory of Computing* (1987), pp. 398–408.

[85] Rónyai, L. Zero divisors in quaternion algebras. *Journal of Algorithms 9* (1988), 494–506.

[86] Rónyai, L. Computing the structure of finite algebras. *Journal of Symbolic Computation 9* (1990), 355–373.

[87] Rónyai, L. A deterministic method for computing splitting elements in simple algebras over $\mathbb{Q}$. *Journal of Algorithms 16* (1994), 24–32.

[88] Ruškuc, N. On large subsemigroups and finiteness conditions of semigroups. *Proceedings of the London Mathematical Society 76* (1998), 383–405.

[89] Schönert, M., et al. *GAP – Groups, Algorithms, and Programming*, fifth ed. Lehrstuhl D für Mathematik, Rheinisch Westfälische Technische Hochschule, Aachen, Germany, 1995.

[90] Schneider, G. J. A. Computer with endomorphism rings of modular representations. *Journal of Symbolic Computation 9* (1990), 607–636.

[91] Sudkamp, T. A. *Languages and Machines*. Addison-Wesley series in Computer Science. Addison-Wesley, 1988.

[92] Sweedler, M. E. *Hopf Algebras*. W. A. Benjamin, New York, 1969.

[93] Szőke, M. *Examining Green Correspondents of Weight Modules*. PhD thesis, RWTH, RWTH-A, 1998.

[94] van der Waerden, B. L. *Algebra*, vol. 1. Ungar, New York, 1970.

[95] van der Waerden, B. L. *Algebra*, vol. 2. Ungar, New York, 1970.

[96] Wedderburn, J. H. M. On hypercomplex numbers. In *Proceedings of the London Mathematical Society* (November 1907), no. 6 in 2, pp. 77–118.

[97] West, D. B. *Introduction to Graph Theory*. Prentice Hall, Upper Saddle River, NJ, 1996.

[98] Winograd, S. *Arithmetic Complexity of Computations*. No. 33 in Regional Conference Series in Applied Mathematics. Society for Industrial and Applied Mathematics, Philadelphia, Pennsylvania, 1980.

[99] Witherspoon, S. J. The representation ring of the quantum double of a finite group. *Journal of Algebra 179* (1996), 305–329.

List of Symbols

$-a$	the additive inverse of ring element a	pg. 9
A	an (associative) algebra	pg. 13
$\mathbb{C}$	the complex numbers	pg. 8
$c_a(x)$	characteristic polynomial of a	pg. 94
$\mathbb{F}_q$	the finite field of order $q = p^n$.	pg. 10
ψ_q	Frobenius homomorphism mapping $x \mapsto x^q$	pg. 65
G	a group	pg. 8
I	a left, right, or two-sided ideal	pg. 10
a^{-1}	inverse of group element a	pg. 7
$\mathbb{K}$	a field	pg. 13
$\prec_{\text{lenllex}}$	(left) length lexicographic order	pg. 45
$M_n(\mathbb{K})$	full matrix algebra of rank n over $\mathbb{K}$	pg. 13
$\hat{s}\,(S)$	maximum size cost of an element in a set S	pg. 21
$m_a(x)$	minimal polynomial of a	pg. 62
$M \otimes_R N$	tensor product of R-modules	pg. 13
M	a right module	pg. 11

$M_1 + M_2$	the sum of modules M_1 and M_2	pg. 12
$*_{M,R}$	Right multiplication for a right R-module *module*	pg. 20
$*_{R,M}$	Left multiplication for a left R-module M	pg. 20
$*_R$	multiplication for a ring R	pg. 20
$\overline{a}$	normal form of a	pg. 46
1_R	multiplicative identity for ring R	pg. 9
$\mathbb{K}\Gamma$	a path algebra	pg. 41
$+_R$	addition for a ring R	pg. 20
$R[x]$	The polynomial ring with coefficients from R	pg. 9
$\mathbb{Q}$	the rational numbers	pg. 8
$\mathbb{R}$	the real numbers	pg. 8
$R \times S$	product of rings R and S	pg. 12
R	a ring	pg. 9
$\mathcal{S}(a)$	the basis elements with non-zero contributions to a	pg. 42
$\tau(x)$	the encoding of element x	pg. 24
$\mathrm{Tr}\,(a)$	the trace of $a \in A$.	pg. 87
Γ	the set of walks in a quiver	pg. 40
$\mathbb{Z}$	the integers	pg. 8
$Z(R)$	the center of ring R	pg. 10
0_R	the additive identity of ring R	pg. 9
e	identity element in a group	pg. 7
f_l	operation representing the computing on line l in an algorithm	pg. 20

$M \oplus N$	direct sum of modules M and N	pg. 12
$s(e)$	storage cost of element e	pg. 16
$S_{\mathrm{A}}(x_1, \ldots, x_l)$	encoding neutral space complexity of A	pg. 17
$t(f)$	time cost of an operation f	pg. 16
$T_{\mathrm{A}}(x_1, \ldots, x_l)$	encoding neutral time complexity of A	pg. 17
$\det(A)$	determinant of matrix A	pg. 8
n!	n factorial	pg. 8
$GL_n(\mathbb{K})$	general linear group of degree n	pg. 8
$\|S\|$	the order or cardinality of the set S	pg. 8
S_n	the symmetric group of degree n	pg. 8

Index

Appendix A

Experimental Data

A.1 Conversion

This section contains the experimental data collected for converting algebras from a group algebra encoding to the multiplication table and regular matrix representation encodings. All algebras are group algebras defined over the field $\mathbb{F}_{53}$. The group used in the experiment is labeled with the `SmallGroup` command used in GAP version 4.1 fix 7. All recorded times are given in CPU milliseconds.

Table A.1: Conversion experiment, run 1.

Group	Dimension	Mult. Table	Isomorphism	Adjoint
SmallGroup(2, 1)	2	25	18	2
SmallGroup(3, 1)	3	12	10	3
SmallGroup(4, 1)	4	35	13	4
SmallGroup(5, 1)	5	21	12	6
SmallGroup(6, 1)	6	33	27	8
SmallGroup(7, 1)	7	35	32	11
SmallGroup(8, 5)	8	40	33	12
SmallGroup(9, 2)	9	97	38	16
SmallGroup(10, 1)	10	49	44	20
SmallGroup(11, 1)	11	70	72	27
SmallGroup(12, 1)	12	66	63	22
SmallGroup(13, 1)	13	109	99	39
SmallGroup(14, 1)	14	87	78	45
SmallGroup(15, 1)	15	98	87	53
SmallGroup(16, 13)	16	114	107	61
SmallGroup(17, 1)	17	185	167	77
SmallGroup(18, 2)	18	141	126	84
SmallGroup(19, 1)	19	237	211	102
SmallGroup(20, 5)	20	174	151	105
SmallGroup(21, 1)	21	193	160	126
SmallGroup(22, 2)	22	215	166	142
SmallGroup(23, 1)	23	349	308	168
SmallGroup(24, 12)	24	254	221	183
SmallGroup(25, 1)	25	275	214	195
SmallGroup(26, 2)	26	291	239	225
SmallGroup(27, 5)	27	325	257	251
SmallGroup(28, 2)	28	351	282	278
SmallGroup(29, 1)	29	588	502	316
SmallGroup(30, 3)	30	392	315	338
SmallGroup(31, 1)	31	678	577	378
SmallGroup(32, 30)	32	466	384	407
SmallGroup(33, 1)	33	505	373	444
SmallGroup(34, 1)	34	524	401	478
SmallGroup(35, 1)	35	560	422	522
SmallGroup(36, 13)	36	602	470	566
SmallGroup(37, 1)	37	987	820	631
SmallGroup(38, 1)	38	682	497	662
SmallGroup(39, 2)	39	719	518	714
SmallGroup(40, 3)	40	765	571	769
SmallGroup(41, 1)	41	1253	1029	849
SmallGroup(42, 4)	42	846	616	886
SmallGroup(43, 1)	43	1403	1150	978
SmallGroup(44, 2)	44	934	674	1014
SmallGroup(45, 2)	45	955	704	1103
SmallGroup(46, 2)	46	1037	719	1188
SmallGroup(47, 1)	47	1718	1386	1263
SmallGroup(48, 34)	48	1150	839	1316
SmallGroup(49, 2)	49	1182	812	1408
SmallGroup(50, 2)	50	1254	864	1496

Table A.2: Conversion experiment, run 2.

Group	Dimension	Mult. Table	Isomorphism	Adjoint
SmallGroup(2, 1)	2	25	18	2
SmallGroup(3, 1)	3	12	10	4
SmallGroup(4, 1)	4	33	12	3
SmallGroup(5, 1)	5	22	20	6
SmallGroup(6, 1)	6	33	27	7
SmallGroup(7, 1)	7	35	33	3
SmallGroup(8, 5)	8	40	33	13
SmallGroup(9, 2)	9	97	37	16
SmallGroup(10, 1)	10	48	44	19
SmallGroup(11, 1)	11	79	74	27
SmallGroup(12, 1)	12	66	64	31
SmallGroup(13, 1)	13	109	100	39
SmallGroup(14, 1)	14	88	77	45
SmallGroup(15, 1)	15	99	87	52
SmallGroup(16, 13)	16	114	107	61
SmallGroup(17, 1)	17	185	167	76
SmallGroup(18, 2)	18	133	125	84
SmallGroup(19, 1)	19	237	211	102
SmallGroup(20, 5)	20	173	151	112
SmallGroup(21, 1)	21	193	160	126
SmallGroup(22, 2)	22	213	172	142
SmallGroup(23, 1)	23	348	307	167
SmallGroup(24, 12)	24	253	222	183
SmallGroup(25, 1)	25	275	222	203
SmallGroup(26, 2)	26	298	239	216
SmallGroup(27, 5)	27	324	265	251
SmallGroup(28, 2)	28	350	282	278
SmallGroup(29, 1)	29	588	493	314
SmallGroup(30, 3)	30	406	324	338
SmallGroup(31, 1)	31	671	576	378
SmallGroup(32, 30)	32	464	384	406
SmallGroup(33, 1)	33	505	373	445
SmallGroup(34, 1)	34	531	401	479
SmallGroup(35, 1)	35	544	420	523
SmallGroup(36, 13)	36	605	470	565
SmallGroup(37, 1)	37	978	828	631
SmallGroup(38, 1)	38	666	496	662
SmallGroup(39, 2)	39	717	517	714
SmallGroup(40, 3)	40	761	578	768
SmallGroup(41, 1)	41	1238	1029	848
SmallGroup(42, 4)	42	844	615	892
SmallGroup(43, 1)	43	1404	1132	977
SmallGroup(44, 2)	44	930	666	1014
SmallGroup(45, 2)	45	976	703	1089
SmallGroup(46, 2)	46	1025	717	1187
SmallGroup(47, 1)	47	1715	1384	1250
SmallGroup(48, 34)	48	1145	837	1318
SmallGroup(49, 2)	49	1186	811	1395
SmallGroup(50, 2)	50	1247	861	1491

Table A.3: Conversion experiment, run 3.

Group	Dimension	Mult. Table	Isomorphism	Adjoint
SmallGroup(2, 1)	2	24	18	3
SmallGroup(3, 1)	3	13	11	4
SmallGroup(4, 1)	4	35	13	3
SmallGroup(5, 1)	5	21	19	5
SmallGroup(6, 1)	6	32	26	7
SmallGroup(7, 1)	7	36	32	11
SmallGroup(8, 5)	8	41	33	13
SmallGroup(9, 2)	9	97	37	16
SmallGroup(10, 1)	10	47	44	20
SmallGroup(11, 1)	11	79	72	27
SmallGroup(12, 1)	12	68	64	32
SmallGroup(13, 1)	13	110	101	41
SmallGroup(14, 1)	14	89	79	45
SmallGroup(15, 1)	15	102	88	53
SmallGroup(16, 13)	16	116	110	63
SmallGroup(17, 1)	17	189	168	78
SmallGroup(18, 2)	18	143	127	86
SmallGroup(19, 1)	19	239	216	104
SmallGroup(20, 5)	20	177	155	113
SmallGroup(21, 1)	21	197	163	129
SmallGroup(22, 2)	22	218	176	146
SmallGroup(23, 1)	23	353	303	169
SmallGroup(24, 12)	24	258	225	188
SmallGroup(25, 1)	25	278	220	204
SmallGroup(26, 2)	26	301	241	228
SmallGroup(27, 5)	27	329	267	254
SmallGroup(28, 2)	28	346	284	283
SmallGroup(29, 1)	29	591	502	314
SmallGroup(30, 3)	30	408	323	338
SmallGroup(31, 1)	31	678	578	377
SmallGroup(32, 30)	32	450	384	407
SmallGroup(33, 1)	33	503	374	443
SmallGroup(34, 1)	34	532	400	478
SmallGroup(35, 1)	35	559	421	523
SmallGroup(36, 13)	36	605	471	565
SmallGroup(37, 1)	37	986	828	631
SmallGroup(38, 1)	38	682	496	661
SmallGroup(39, 2)	39	703	518	713
SmallGroup(40, 3)	40	764	578	768
SmallGroup(41, 1)	41	1253	1029	841
SmallGroup(42, 4)	42	843	616	883
SmallGroup(43, 1)	43	1404	1143	969
SmallGroup(44, 2)	44	908	674	1021
SmallGroup(45, 2)	45	968	704	1105
SmallGroup(46, 2)	46	1035	703	1185
SmallGroup(47, 1)	47	1716	1379	1261
SmallGroup(48, 34)	48	1148	830	1313
SmallGroup(49, 2)	49	1186	812	1402
SmallGroup(50, 2)	50	1248	839	1487

Table A.4: Conversion experiment, run 4.

Group	Dimension	Mult. Table	Isomorphism	Adjoint
SmallGroup(2, 1)	2	24	17	2
SmallGroup(3, 1)	3	11	10	4
SmallGroup(4, 1)	4	34	12	3
SmallGroup(5, 1)	5	22	19	7
SmallGroup(6, 1)	6	32	26	7
SmallGroup(7, 1)	7	36	33	10
SmallGroup(8, 5)	8	40	34	13
SmallGroup(9, 2)	9	98	38	16
SmallGroup(10, 1)	10	48	44	20
SmallGroup(11, 1)	11	77	73	27
SmallGroup(12, 1)	12	66	63	30
SmallGroup(13, 1)	13	108	100	40
SmallGroup(14, 1)	14	86	77	45
SmallGroup(15, 1)	15	98	86	53
SmallGroup(16, 13)	16	113	107	61
SmallGroup(17, 1)	17	185	166	76
SmallGroup(18, 2)	18	140	125	85
SmallGroup(19, 1)	19	229	210	102
SmallGroup(20, 5)	20	174	152	112
SmallGroup(21, 1)	21	192	160	126
SmallGroup(22, 2)	22	213	173	142
SmallGroup(23, 1)	23	348	308	168
SmallGroup(24, 12)	24	254	221	182
SmallGroup(25, 1)	25	259	214	202
SmallGroup(26, 2)	26	298	238	224
SmallGroup(27, 5)	27	325	265	251
SmallGroup(28, 2)	28	349	282	278
SmallGroup(29, 1)	29	588	502	307
SmallGroup(30, 3)	30	406	323	337
SmallGroup(31, 1)	31	677	575	370
SmallGroup(32, 30)	32	464	384	407
SmallGroup(33, 1)	33	504	373	429
SmallGroup(34, 1)	34	530	400	478
SmallGroup(35, 1)	35	559	413	522
SmallGroup(36, 13)	36	605	463	566
SmallGroup(37, 1)	37	985	818	630
SmallGroup(38, 1)	38	679	496	662
SmallGroup(39, 2)	39	717	517	712
SmallGroup(40, 3)	40	761	563	761
SmallGroup(41, 1)	41	1252	1029	849
SmallGroup(42, 4)	42	843	616	885
SmallGroup(43, 1)	43	1394	1148	977
SmallGroup(44, 2)	44	929	673	1021
SmallGroup(45, 2)	45	966	703	1102
SmallGroup(46, 2)	46	1032	718	1187
SmallGroup(47, 1)	47	1714	1375	1255
SmallGroup(48, 34)	48	1138	837	1317
SmallGroup(49, 2)	49	1186	787	1409
SmallGroup(50, 2)	50	1238	861	1495

Table A.5: Conversion experiment, run 5.

Group	Dimension	Mult. Table	Isomorphism	Adjoint
SmallGroup(2, 1)	2	25	18	2
SmallGroup(3, 1)	3	13	11	4
SmallGroup(4, 1)	4	35	14	4
SmallGroup(5, 1)	5	22	19	5
SmallGroup(6, 1)	6	33	25	7
SmallGroup(7, 1)	7	35	32	11
SmallGroup(8, 5)	8	41	33	12
SmallGroup(9, 2)	9	103	37	16
SmallGroup(10, 1)	10	47	44	19
SmallGroup(11, 1)	11	77	72	19
SmallGroup(12, 1)	12	67	63	30
SmallGroup(13, 1)	13	108	99	39
SmallGroup(14, 1)	14	86	77	45
SmallGroup(15, 1)	15	99	87	54
SmallGroup(16, 13)	16	113	108	62
SmallGroup(17, 1)	17	186	166	76
SmallGroup(18, 2)	18	141	125	85
SmallGroup(19, 1)	19	237	211	102
SmallGroup(20, 5)	20	174	151	112
SmallGroup(21, 1)	21	192	160	126
SmallGroup(22, 2)	22	206	172	142
SmallGroup(23, 1)	23	349	308	167
SmallGroup(24, 12)	24	254	222	184
SmallGroup(25, 1)	25	275	222	202
SmallGroup(26, 2)	26	297	239	224
SmallGroup(27, 5)	27	324	257	251
SmallGroup(28, 2)	28	349	282	277
SmallGroup(29, 1)	29	588	502	314
SmallGroup(30, 3)	30	408	323	338
SmallGroup(31, 1)	31	677	569	369
SmallGroup(32, 30)	32	457	384	408
SmallGroup(33, 1)	33	505	373	444
SmallGroup(34, 1)	34	530	400	479
SmallGroup(35, 1)	35	558	420	506
SmallGroup(36, 13)	36	606	471	566
SmallGroup(37, 1)	37	984	820	631
SmallGroup(38, 1)	38	673	497	647
SmallGroup(39, 2)	39	717	517	715
SmallGroup(40, 3)	40	761	580	768
SmallGroup(41, 1)	41	1253	1021	850
SmallGroup(42, 4)	42	842	616	892
SmallGroup(43, 1)	43	1404	1141	972
SmallGroup(44, 2)	44	930	674	1021
SmallGroup(45, 2)	45	959	705	1099
SmallGroup(46, 2)	46	1034	703	1189
SmallGroup(47, 1)	47	1715	1385	1265
SmallGroup(48, 34)	48	1146	838	1316
SmallGroup(49, 2)	49	1185	795	1413
SmallGroup(50, 2)	50	1248	862	1496

A.2 Center

This section contains the experimental data collected for calculating $Z(A)$ for a group algebra, encoded as a group algebra, as a multiplication table, and as its regular matrix representation. All algebras are group algebras defined over the field $\mathbb{F}_{53}$. The group used in the experiment is labeled with the `SmallGroup` command used in GAP version 4.1 fix 7. All recorded times are given in CPU milliseconds.

Table A.6: Center experiment, run 1.

Group	dim A	dim $Z(A)$	Group	Mult. Table	Matrix
SmallGroup(2, 1)	2	2	3	4	7
SmallGroup(3, 1)	3	3	0	3	10
SmallGroup(4, 1)	4	4	1	4	14
SmallGroup(5, 1)	5	5	1	5	19
SmallGroup(6, 1)	6	3	331	6	26
SmallGroup(7, 1)	7	7	3	6	36
SmallGroup(8, 5)	8	8	74	6	45
SmallGroup(9, 2)	9	9	0	7	57
SmallGroup(10, 1)	10	4	23	9	75
SmallGroup(11, 1)	11	11	0	8	88
SmallGroup(12, 1)	12	6	31	10	113
SmallGroup(13, 1)	13	13	0	10	124
SmallGroup(14, 1)	14	5	27	12	167
SmallGroup(15, 1)	15	15	59	12	179
SmallGroup(16, 13)	16	10	52	14	226
SmallGroup(17, 1)	17	17	0	15	256
SmallGroup(18, 2)	18	18	70	17	296
SmallGroup(19, 1)	19	19	0	18	342
SmallGroup(20, 5)	20	20	78	20	392
SmallGroup(21, 1)	21	5	28	23	488
SmallGroup(22, 2)	22	22	81	24	505
SmallGroup(23, 1)	23	23	1	25	574
SmallGroup(24, 12)	24	5	29	28	711
SmallGroup(25, 1)	25	25	1	28	729
SmallGroup(26, 2)	26	26	95	31	822
SmallGroup(27, 5)	27	27	101	33	897
SmallGroup(28, 2)	28	28	105	37	1009
SmallGroup(29, 1)	29	29	0	38	1117
SmallGroup(30, 3)	30	9	42	43	1360
SmallGroup(31, 1)	31	31	0	45	1355
SmallGroup(32, 30)	32	14	64	49	1625
SmallGroup(33, 1)	33	33	121	51	1647
SmallGroup(34, 1)	34	10	45	56	1996
SmallGroup(35, 1)	35	35	126	57	1950
SmallGroup(36, 13)	36	12	55	63	2354
SmallGroup(37, 1)	37	37	1	65	2340
SmallGroup(38, 1)	38	11	50	72	2810
SmallGroup(39, 2)	39	39	140	73	2729
SmallGroup(40, 3)	40	10	50	79	3320
SmallGroup(41, 1)	41	41	0	82	3211
SmallGroup(42, 4)	42	15	67	88	3799
SmallGroup(43, 1)	43	43	0	90	3798
SmallGroup(44, 2)	44	44	162	98	4027
SmallGroup(45, 2)	45	45	168	101	4295
SmallGroup(46, 2)	46	46	167	106	4613
SmallGroup(47, 1)	47	47	0	112	5031
SmallGroup(48, 34)	48	18	84	121	5904
SmallGroup(49, 2)	49	49	1	123	5732
SmallGroup(50, 2)	50	50	188	128	6810

Table A.7: Center experiment, run 2.

Group	dim A	dim $Z(A)$	Group	Mult. Table	Matrix
SmallGroup(2, 1)	2	2	3	3	8
SmallGroup(3, 1)	3	3	0	3	11
SmallGroup(4, 1)	4	4	1	4	14
SmallGroup(5, 1)	5	5	0	5	19
SmallGroup(6, 1)	6	3	315	6	27
SmallGroup(7, 1)	7	7	2	6	35
SmallGroup(8, 5)	8	8	73	6	46
SmallGroup(9, 2)	9	9	0	7	56
SmallGroup(10, 1)	10	4	23	8	75
SmallGroup(11, 1)	11	11	0	8	88
SmallGroup(12, 1)	12	6	23	9	111
SmallGroup(13, 1)	13	13	1	11	132
SmallGroup(14, 1)	14	5	26	12	166
SmallGroup(15, 1)	15	15	58	12	187
SmallGroup(16, 13)	16	10	43	15	217
SmallGroup(17, 1)	17	17	0	15	256
SmallGroup(18, 2)	18	18	69	17	296
SmallGroup(19, 1)	19	19	0	18	342
SmallGroup(20, 5)	20	20	77	19	391
SmallGroup(21, 1)	21	5	28	22	487
SmallGroup(22, 2)	22	22	81	16	506
SmallGroup(23, 1)	23	23	1	25	567
SmallGroup(24, 12)	24	5	29	29	703
SmallGroup(25, 1)	25	25	0	29	713
SmallGroup(26, 2)	26	26	96	32	828
SmallGroup(27, 5)	27	27	101	34	913
SmallGroup(28, 2)	28	28	105	30	1016
SmallGroup(29, 1)	29	29	1	39	1132
SmallGroup(30, 3)	30	9	42	44	1380
SmallGroup(31, 1)	31	31	0	45	1370
SmallGroup(32, 30)	32	14	64	49	1626
SmallGroup(33, 1)	33	33	120	51	1662
SmallGroup(34, 1)	34	10	47	56	1986
SmallGroup(35, 1)	35	35	126	58	1964
SmallGroup(36, 13)	36	12	55	63	2361
SmallGroup(37, 1)	37	37	1	64	2341
SmallGroup(38, 1)	38	11	49	72	2803
SmallGroup(39, 2)	39	39	139	72	2752
SmallGroup(40, 3)	40	10	49	79	3330
SmallGroup(41, 1)	41	41	1	82	3197
SmallGroup(42, 4)	42	15	67	89	3809
SmallGroup(43, 1)	43	43	0	91	3809
SmallGroup(44, 2)	44	44	163	97	4026
SmallGroup(45, 2)	45	45	167	102	4304
SmallGroup(46, 2)	46	46	165	106	4635
SmallGroup(47, 1)	47	47	0	105	4970
SmallGroup(48, 34)	48	18	83	122	5907
SmallGroup(49, 2)	49	49	0	122	5735
SmallGroup(50, 2)	50	50	187	129	6853

Table A.8: Center experiment, run 3.

Group	dim A	dim $Z(A)$	Group	Mult. Table	Matrix
SmallGroup(2, 1)	2	2	2	3	7
SmallGroup(3, 1)	3	3	0	3	10
SmallGroup(4, 1)	4	4	0	5	14
SmallGroup(5, 1)	5	5	1	4	19
SmallGroup(6, 1)	6	3	310	5	27
SmallGroup(7, 1)	7	7	3	5	35
SmallGroup(8, 5)	8	8	76	7	45
SmallGroup(9, 2)	9	9	1	6	56
SmallGroup(10, 1)	10	4	23	8	75
SmallGroup(11, 1)	11	11	0	9	89
SmallGroup(12, 1)	12	6	31	10	113
SmallGroup(13, 1)	13	13	0	10	133
SmallGroup(14, 1)	14	5	27	12	167
SmallGroup(15, 1)	15	15	59	13	180
SmallGroup(16, 13)	16	10	53	15	228
SmallGroup(17, 1)	17	17	0	15	258
SmallGroup(18, 2)	18	18	71	17	298
SmallGroup(19, 1)	19	19	0	18	345
SmallGroup(20, 5)	20	20	78	20	395
SmallGroup(21, 1)	21	5	29	23	493
SmallGroup(22, 2)	22	22	81	23	509
SmallGroup(23, 1)	23	23	0	25	571
SmallGroup(24, 12)	24	5	29	28	709
SmallGroup(25, 1)	25	25	0	29	727
SmallGroup(26, 2)	26	26	97	32	826
SmallGroup(27, 5)	27	27	101	33	913
SmallGroup(28, 2)	28	28	105	37	1009
SmallGroup(29, 1)	29	29	0	39	1126
SmallGroup(30, 3)	30	9	43	44	1353
SmallGroup(31, 1)	31	31	1	45	1357
SmallGroup(32, 30)	32	14	63	48	1625
SmallGroup(33, 1)	33	33	120	51	1600
SmallGroup(34, 1)	34	10	45	56	1979
SmallGroup(35, 1)	35	35	125	57	1975
SmallGroup(36, 13)	36	12	56	63	2382
SmallGroup(37, 1)	37	37	0	65	2335
SmallGroup(38, 1)	38	11	50	74	2846
SmallGroup(39, 2)	39	39	140	72	2735
SmallGroup(40, 3)	40	10	49	79	3327
SmallGroup(41, 1)	41	41	0	82	3215
SmallGroup(42, 4)	42	15	67	89	3807
SmallGroup(43, 1)	43	43	0	89	3832
SmallGroup(44, 2)	44	44	163	95	4014
SmallGroup(45, 2)	45	45	167	101	4301
SmallGroup(46, 2)	46	46	165	105	4622
SmallGroup(47, 1)	47	47	0	111	4969
SmallGroup(48, 34)	48	18	84	121	5847
SmallGroup(49, 2)	49	49	0	122	5733
SmallGroup(50, 2)	50	50	186	129	6801

Table A.9: Center experiment, run 4.

Group	dim A	dim $Z(A)$	Group	Mult. Table	Matrix
SmallGroup(2, 1)	2	2	2	3	8
SmallGroup(3, 1)	3	3	0	4	9
SmallGroup(4, 1)	4	4	0	3	14
SmallGroup(5, 1)	5	5	0	5	20
SmallGroup(6, 1)	6	3	341	6	27
SmallGroup(7, 1)	7	7	2	6	35
SmallGroup(8, 5)	8	8	75	6	45
SmallGroup(9, 2)	9	9	0	7	58
SmallGroup(10, 1)	10	4	23	9	75
SmallGroup(11, 1)	11	11	1	8	90
SmallGroup(12, 1)	12	6	33	10	117
SmallGroup(13, 1)	13	13	1	10	135
SmallGroup(14, 1)	14	5	26	11	168
SmallGroup(15, 1)	15	15	59	13	189
SmallGroup(16, 13)	16	10	52	15	227
SmallGroup(17, 1)	17	17	0	16	261
SmallGroup(18, 2)	18	18	72	17	299
SmallGroup(19, 1)	19	19	0	17	342
SmallGroup(20, 5)	20	20	79	21	393
SmallGroup(21, 1)	21	5	28	23	491
SmallGroup(22, 2)	22	22	82	23	505
SmallGroup(23, 1)	23	23	0	25	574
SmallGroup(24, 12)	24	5	29	29	703
SmallGroup(25, 1)	25	25	1	29	721
SmallGroup(26, 2)	26	26	95	31	811
SmallGroup(27, 5)	27	27	101	33	895
SmallGroup(28, 2)	28	28	104	37	1006
SmallGroup(29, 1)	29	29	0	39	1115
SmallGroup(30, 3)	30	9	42	43	1353
SmallGroup(31, 1)	31	31	0	45	1357
SmallGroup(32, 30)	32	14	64	49	1617
SmallGroup(33, 1)	33	33	120	51	1645
SmallGroup(34, 1)	34	10	46	55	1994
SmallGroup(35, 1)	35	35	126	57	1949
SmallGroup(36, 13)	36	12	54	63	2353
SmallGroup(37, 1)	37	37	1	65	2342
SmallGroup(38, 1)	38	11	50	71	2817
SmallGroup(39, 2)	39	39	140	72	2751
SmallGroup(40, 3)	40	10	50	80	3319
SmallGroup(41, 1)	41	41	1	81	3210
SmallGroup(42, 4)	42	15	66	88	3822
SmallGroup(43, 1)	43	43	0	90	3783
SmallGroup(44, 2)	44	44	162	95	4007
SmallGroup(45, 2)	45	45	167	101	4317
SmallGroup(46, 2)	46	46	166	105	4637
SmallGroup(47, 1)	47	47	1	113	4939
SmallGroup(48, 34)	48	18	84	122	5838
SmallGroup(49, 2)	49	49	0	122	5744
SmallGroup(50, 2)	50	50	188	128	6806

Table A.10: Center experiment, run 5.

Group	dim A	dim $Z(A)$	Group	Mult. Table	Matrix
SmallGroup(2, 1)	2	2	2	4	8
SmallGroup(3, 1)	3	3	0	3	10
SmallGroup(4, 1)	4	4	0	4	14
SmallGroup(5, 1)	5	5	0	4	19
SmallGroup(6, 1)	6	3	322	5	26
SmallGroup(7, 1)	7	7	2	6	35
SmallGroup(8, 5)	8	8	75	6	45
SmallGroup(9, 2)	9	9	0	7	57
SmallGroup(10, 1)	10	4	22	8	74
SmallGroup(11, 1)	11	11	1	8	88
SmallGroup(12, 1)	12	6	32	10	113
SmallGroup(13, 1)	13	13	0	10	132
SmallGroup(14, 1)	14	5	26	12	166
SmallGroup(15, 1)	15	15	58	12	186
SmallGroup(16, 13)	16	10	52	14	226
SmallGroup(17, 1)	17	17	0	15	255
SmallGroup(18, 2)	18	18	69	16	296
SmallGroup(19, 1)	19	19	0	18	342
SmallGroup(20, 5)	20	20	77	19	390
SmallGroup(21, 1)	21	5	28	22	488
SmallGroup(22, 2)	22	22	81	23	505
SmallGroup(23, 1)	23	23	1	25	573
SmallGroup(24, 12)	24	5	29	28	711
SmallGroup(25, 1)	25	25	1	29	728
SmallGroup(26, 2)	26	26	96	32	820
SmallGroup(27, 5)	27	27	101	34	904
SmallGroup(28, 2)	28	28	104	37	1007
SmallGroup(29, 1)	29	29	0	39	1118
SmallGroup(30, 3)	30	9	42	43	1337
SmallGroup(31, 1)	31	31	1	45	1356
SmallGroup(32, 30)	32	14	63	48	1623
SmallGroup(33, 1)	33	33	120	51	1645
SmallGroup(34, 1)	34	10	45	55	1994
SmallGroup(35, 1)	35	35	126	58	1962
SmallGroup(36, 13)	36	12	55	63	2361
SmallGroup(37, 1)	37	37	0	64	2332
SmallGroup(38, 1)	38	11	50	71	2810
SmallGroup(39, 2)	39	39	140	72	2727
SmallGroup(40, 3)	40	10	49	79	3328
SmallGroup(41, 1)	41	41	0	81	3204
SmallGroup(42, 4)	42	15	67	89	3815
SmallGroup(43, 1)	43	43	0	90	3799
SmallGroup(44, 2)	44	44	162	95	4021
SmallGroup(45, 2)	45	45	167	101	4310
SmallGroup(46, 2)	46	46	158	106	4637
SmallGroup(47, 1)	47	47	1	112	4958
SmallGroup(48, 34)	48	18	84	121	5827
SmallGroup(49, 2)	49	49	0	122	5731
SmallGroup(50, 2)	50	50	179	128	6795

A.3 Power Subalgebra

This section contains the experimental data collected for calculating A^{ψ_q} for a group algebra, encoded as a group algebra, as a multiplication table, and as its regular matrix representation. The computation of $Z(A)$ occurs before executing PowerSubalgebra. The dimension of $Z(A)$ is included in the table. All recorded times are given in CPU milliseconds.

A.3.1 Fixed Field

All algebras are group algebras defined over the field $\mathbb{F}_{53}$. The group used in the experiment is labeled with the `SmallGroup` command used in GAP version 4.1 fix 7.

Table A.11: Power subalgebra, field fixed to $\mathbb{F}_{53}$, run 1.

Group	$\dim A$	$\dim Z(A)$	$\dim A^{\psi_q}$	Group	Mult. Table	Matrix
SmallGroup(2, 1)	2	2	2	28	12	15
SmallGroup(3, 1)	3	3	2	21	10	10
SmallGroup(4, 1)	4	4	4	94	11	10
SmallGroup(5, 1)	5	5	2	30	14	13
SmallGroup(6, 1)	6	3	3	1829	17	8
SmallGroup(7, 1)	7	7	3	46	17	17
SmallGroup(8, 5)	8	8	8	58	17	21
SmallGroup(9, 2)	9	9	5	52	22	24
SmallGroup(10, 1)	10	4	3	499	13	14
SmallGroup(11, 1)	11	11	3	24	27	35
SmallGroup(12, 1)	12	6	6	740	16	21
SmallGroup(13, 1)	13	13	13	46	34	47
SmallGroup(14, 1)	14	5	3	374	18	20
SmallGroup(15, 1)	15	15	5	57	39	60
SmallGroup(16, 13)	16	10	10	1296	23	45
SmallGroup(17, 1)	17	17	3	51	49	80
SmallGroup(18, 2)	18	18	10	63	51	91
SmallGroup(19, 1)	19	19	2	26	57	100
SmallGroup(20, 5)	20	20	8	63	59	112
SmallGroup(21, 1)	21	5	4	494	20	32
SmallGroup(22, 2)	22	22	6	53	69	143
SmallGroup(23, 1)	23	23	2	37	77	156
SmallGroup(24, 12)	24	5	5	1554	36	39
SmallGroup(25, 1)	25	25	3	83	89	189
SmallGroup(26, 2)	26	26	26	68	84	211
SmallGroup(27, 5)	27	27	14	53	99	235
SmallGroup(28, 2)	28	28	12	43	106	254
SmallGroup(29, 1)	29	29	5	43	125	276
SmallGroup(30, 3)	30	9	5	1077	56	87
SmallGroup(31, 1)	31	31	2	51	126	332
SmallGroup(32, 30)	32	14	14	1973	50	123
SmallGroup(33, 1)	33	33	6	78	147	392
SmallGroup(34, 1)	34	10	4	697	64	118
SmallGroup(35, 1)	35	35	6	77	168	450
SmallGroup(36, 13)	36	12	12	1599	52	125
SmallGroup(37, 1)	37	37	5	75	194	534
SmallGroup(38, 1)	38	11	3	671	79	144
SmallGroup(39, 2)	39	39	26	81	209	620
SmallGroup(40, 3)	40	10	8	1141	44	126
SmallGroup(41, 1)	41	41	2	76	240	701
SmallGroup(42, 4)	42	15	6	860	67	206
SmallGroup(43, 1)	43	43	3	49	260	793
SmallGroup(44, 2)	44	44	12	89	277	888
SmallGroup(45, 2)	45	45	14	60	284	916
SmallGroup(46, 2)	46	46	4	44	298	970
SmallGroup(47, 1)	47	47	3	53	317	1058
SmallGroup(48, 34)	48	18	16	2472	79	297
SmallGroup(49, 2)	49	49	17	96	350	1224
SmallGroup(50, 2)	50	50	6	70	369	1257

Table A.12: Power subalgebra, field fixed to $\mathbb{F}_{53}$, run 2.

Group	$\dim A$	$\dim Z(A)$	$\dim A^{\psi_q}$	Group	Mult. Table	Matrix
SmallGroup(2, 1)	2	2	2	28	13	15
SmallGroup(3, 1)	3	3	2	21	9	8
SmallGroup(4, 1)	4	4	4	94	10	11
SmallGroup(5, 1)	5	5	2	31	13	13
SmallGroup(6, 1)	6	3	3	1796	17	10
SmallGroup(7, 1)	7	7	3	45	19	18
SmallGroup(8, 5)	8	8	8	58	18	21
SmallGroup(9, 2)	9	9	5	51	23	25
SmallGroup(10, 1)	10	4	3	494	13	14
SmallGroup(11, 1)	11	11	3	25	28	34
SmallGroup(12, 1)	12	6	6	733	16	22
SmallGroup(13, 1)	13	13	13	46	34	47
SmallGroup(14, 1)	14	5	3	376	18	20
SmallGroup(15, 1)	15	15	5	57	40	60
SmallGroup(16, 13)	16	10	10	1298	23	45
SmallGroup(17, 1)	17	17	3	50	49	79
SmallGroup(18, 2)	18	18	10	64	51	89
SmallGroup(19, 1)	19	19	2	25	57	100
SmallGroup(20, 5)	20	20	8	62	59	112
SmallGroup(21, 1)	21	5	4	494	20	24
SmallGroup(22, 2)	22	22	6	45	69	144
SmallGroup(23, 1)	23	23	2	37	76	155
SmallGroup(24, 12)	24	5	5	1563	36	40
SmallGroup(25, 1)	25	25	3	82	89	189
SmallGroup(26, 2)	26	26	26	67	83	210
SmallGroup(27, 5)	27	27	14	53	99	238
SmallGroup(28, 2)	28	28	12	43	107	254
SmallGroup(29, 1)	29	29	5	44	127	276
SmallGroup(30, 3)	30	9	5	1077	56	87
SmallGroup(31, 1)	31	31	2	51	134	332
SmallGroup(32, 30)	32	14	14	1976	50	117
SmallGroup(33, 1)	33	33	6	71	153	394
SmallGroup(34, 1)	34	10	4	699	65	119
SmallGroup(35, 1)	35	35	6	80	170	454
SmallGroup(36, 13)	36	12	12	1631	53	125
SmallGroup(37, 1)	37	37	5	76	194	535
SmallGroup(38, 1)	38	11	3	673	79	144
SmallGroup(39, 2)	39	39	26	81	219	626
SmallGroup(40, 3)	40	10	8	1155	45	127
SmallGroup(41, 1)	41	41	2	77	242	701
SmallGroup(42, 4)	42	15	6	867	68	213
SmallGroup(43, 1)	43	43	3	50	262	795
SmallGroup(44, 2)	44	44	12	82	281	896
SmallGroup(45, 2)	45	45	14	61	289	925
SmallGroup(46, 2)	46	46	4	43	302	991
SmallGroup(47, 1)	47	47	3	54	320	1077
SmallGroup(48, 34)	48	18	16	2468	81	300
SmallGroup(49, 2)	49	49	17	99	355	1239
SmallGroup(50, 2)	50	50	6	70	381	1271

Table A.13: Power subalgebra, field fixed to $\mathbb{F}_{53}$, run 3.

Group	$\dim A$	$\dim Z(A)$	$\dim A^{\psi_q}$	Group	Mult. Table	Matrix
SmallGroup(2, 1)	2	2	2	27	12	14
SmallGroup(3, 1)	3	3	2	22	11	9
SmallGroup(4, 1)	4	4	4	95	10	10
SmallGroup(5, 1)	5	5	2	31	14	13
SmallGroup(6, 1)	6	3	3	1793	18	10
SmallGroup(7, 1)	7	7	3	46	18	18
SmallGroup(8, 5)	8	8	8	59	17	21
SmallGroup(9, 2)	9	9	5	52	23	25
SmallGroup(10, 1)	10	4	3	500	13	15
SmallGroup(11, 1)	11	11	3	25	28	35
SmallGroup(12, 1)	12	6	6	779	15	21
SmallGroup(13, 1)	13	13	13	46	34	47
SmallGroup(14, 1)	14	5	3	373	18	21
SmallGroup(15, 1)	15	15	5	49	41	62
SmallGroup(16, 13)	16	10	10	1268	24	46
SmallGroup(17, 1)	17	17	3	52	51	81
SmallGroup(18, 2)	18	18	10	63	52	91
SmallGroup(19, 1)	19	19	2	26	57	101
SmallGroup(20, 5)	20	20	8	64	55	117
SmallGroup(21, 1)	21	5	4	503	21	32
SmallGroup(22, 2)	22	22	6	48	70	150
SmallGroup(23, 1)	23	23	2	38	78	157
SmallGroup(24, 12)	24	5	5	1593	37	41
SmallGroup(25, 1)	25	25	3	85	93	197
SmallGroup(26, 2)	26	26	26	70	84	212
SmallGroup(27, 5)	27	27	14	53	100	237
SmallGroup(28, 2)	28	28	12	43	106	257
SmallGroup(29, 1)	29	29	5	43	126	281
SmallGroup(30, 3)	30	9	5	1083	57	88
SmallGroup(31, 1)	31	31	2	51	135	328
SmallGroup(32, 30)	32	14	14	2069	52	127
SmallGroup(33, 1)	33	33	6	81	154	393
SmallGroup(34, 1)	34	10	4	682	65	110
SmallGroup(35, 1)	35	35	6	77	168	458
SmallGroup(36, 13)	36	12	12	1576	52	125
SmallGroup(37, 1)	37	37	5	76	194	535
SmallGroup(38, 1)	38	11	3	672	79	144
SmallGroup(39, 2)	39	39	26	81	209	619
SmallGroup(40, 3)	40	10	8	1137	44	118
SmallGroup(41, 1)	41	41	2	75	241	700
SmallGroup(42, 4)	42	15	6	858	66	207
SmallGroup(43, 1)	43	43	3	50	260	801
SmallGroup(44, 2)	44	44	12	89	277	873
SmallGroup(45, 2)	45	45	14	61	285	915
SmallGroup(46, 2)	46	46	4	44	299	986
SmallGroup(47, 1)	47	47	3	53	317	1059
SmallGroup(48, 34)	48	18	16	2444	80	297
SmallGroup(49, 2)	49	49	17	96	342	1216
SmallGroup(50, 2)	50	50	6	70	354	1257

Table A.14: Power subalgebra, field fixed to $\mathbb{F}_{53}$, run 4.

Group	$\dim A$	$\dim Z(A)$	$\dim A^{\psi_q}$	Group	Mult. Table	Matrix
SmallGroup(2, 1)	2	2	2	29	12	15
SmallGroup(3, 1)	3	3	2	22	10	9
SmallGroup(4, 1)	4	4	4	94	10	10
SmallGroup(5, 1)	5	5	2	31	12	12
SmallGroup(6, 1)	6	3	3	1790	17	9
SmallGroup(7, 1)	7	7	3	46	17	18
SmallGroup(8, 5)	8	8	8	57	18	20
SmallGroup(9, 2)	9	9	5	52	21	24
SmallGroup(10, 1)	10	4	3	494	14	14
SmallGroup(11, 1)	11	11	3	25	20	35
SmallGroup(12, 1)	12	6	6	772	15	22
SmallGroup(13, 1)	13	13	13	46	35	47
SmallGroup(14, 1)	14	5	3	373	18	21
SmallGroup(15, 1)	15	15	5	57	40	60
SmallGroup(16, 13)	16	10	10	1291	24	45
SmallGroup(17, 1)	17	17	3	51	49	80
SmallGroup(18, 2)	18	18	10	63	51	90
SmallGroup(19, 1)	19	19	2	25	57	100
SmallGroup(20, 5)	20	20	8	62	59	112
SmallGroup(21, 1)	21	5	4	495	20	32
SmallGroup(22, 2)	22	22	6	53	69	136
SmallGroup(23, 1)	23	23	2	37	77	155
SmallGroup(24, 12)	24	5	5	1535	36	40
SmallGroup(25, 1)	25	25	3	83	89	189
SmallGroup(26, 2)	26	26	26	67	83	210
SmallGroup(27, 5)	27	27	14	52	100	235
SmallGroup(28, 2)	28	28	12	42	106	254
SmallGroup(29, 1)	29	29	5	35	125	276
SmallGroup(30, 3)	30	9	5	1082	55	87
SmallGroup(31, 1)	31	31	2	50	134	334
SmallGroup(32, 30)	32	14	14	1970	51	122
SmallGroup(33, 1)	33	33	6	79	147	392
SmallGroup(34, 1)	34	10	4	687	64	119
SmallGroup(35, 1)	35	35	6	79	171	466
SmallGroup(36, 13)	36	12	12	1628	54	128
SmallGroup(37, 1)	37	37	5	77	199	548
SmallGroup(38, 1)	38	11	3	680	80	146
SmallGroup(39, 2)	39	39	26	83	220	626
SmallGroup(40, 3)	40	10	8	1129	45	127
SmallGroup(41, 1)	41	41	2	76	241	700
SmallGroup(42, 4)	42	15	6	860	58	207
SmallGroup(43, 1)	43	43	3	50	260	801
SmallGroup(44, 2)	44	44	12	89	277	887
SmallGroup(45, 2)	45	45	14	61	269	915
SmallGroup(46, 2)	46	46	4	43	298	986
SmallGroup(47, 1)	47	47	3	53	317	1067
SmallGroup(48, 34)	48	18	16	2470	80	298
SmallGroup(49, 2)	49	49	17	97	349	1223
SmallGroup(50, 2)	50	50	6	70	369	1241

Table A.15: Power subalgebra, field fixed to $\mathbb{F}_{53}$, run 5.

Group	$\dim A$	$\dim Z(A)$	$\dim A^{\psi_q}$	Group	Mult. Table	Matrix
SmallGroup(2, 1)	2	2	2	28	12	15
SmallGroup(3, 1)	3	3	2	22	9	9
SmallGroup(4, 1)	4	4	4	93	11	10
SmallGroup(5, 1)	5	5	2	31	13	13
SmallGroup(6, 1)	6	3	3	1788	17	9
SmallGroup(7, 1)	7	7	3	47	18	18
SmallGroup(8, 5)	8	8	8	58	17	20
SmallGroup(9, 2)	9	9	5	51	22	24
SmallGroup(10, 1)	10	4	3	494	14	15
SmallGroup(11, 1)	11	11	3	24	28	35
SmallGroup(12, 1)	12	6	6	760	15	21
SmallGroup(13, 1)	13	13	13	45	35	48
SmallGroup(14, 1)	14	5	3	372	19	21
SmallGroup(15, 1)	15	15	5	57	40	61
SmallGroup(16, 13)	16	10	10	1282	23	45
SmallGroup(17, 1)	17	17	3	51	49	80
SmallGroup(18, 2)	18	18	10	63	51	90
SmallGroup(19, 1)	19	19	2	25	57	99
SmallGroup(20, 5)	20	20	8	62	60	113
SmallGroup(21, 1)	21	5	4	491	21	32
SmallGroup(22, 2)	22	22	6	53	70	144
SmallGroup(23, 1)	23	23	2	37	77	155
SmallGroup(24, 12)	24	5	5	1545	36	40
SmallGroup(25, 1)	25	25	3	82	89	189
SmallGroup(26, 2)	26	26	26	67	76	210
SmallGroup(27, 5)	27	27	14	52	98	235
SmallGroup(28, 2)	28	28	12	42	105	254
SmallGroup(29, 1)	29	29	5	44	125	277
SmallGroup(30, 3)	30	9	5	1068	56	86
SmallGroup(31, 1)	31	31	2	50	134	332
SmallGroup(32, 30)	32	14	14	1975	50	123
SmallGroup(33, 1)	33	33	6	77	147	392
SmallGroup(34, 1)	34	10	4	674	65	117
SmallGroup(35, 1)	35	35	6	78	168	458
SmallGroup(36, 13)	36	12	12	1587	53	126
SmallGroup(37, 1)	37	37	5	76	195	534
SmallGroup(38, 1)	38	11	3	663	79	144
SmallGroup(39, 2)	39	39	26	82	209	619
SmallGroup(40, 3)	40	10	8	1131	44	126
SmallGroup(41, 1)	41	41	2	76	241	700
SmallGroup(42, 4)	42	15	6	859	66	199
SmallGroup(43, 1)	43	43	3	50	260	801
SmallGroup(44, 2)	44	44	12	89	276	887
SmallGroup(45, 2)	45	45	14	60	284	915
SmallGroup(46, 2)	46	46	4	43	298	986
SmallGroup(47, 1)	47	47	3	53	317	1051
SmallGroup(48, 34)	48	18	16	2474	80	297
SmallGroup(49, 2)	49	49	17	97	350	1223
SmallGroup(50, 2)	50	50	6	71	369	1249

A.3.2 Fixed Algebra

For this experiment, the group algebra was defined using only the group `SmallGroup(20, 5)` and the field was varied. The field used is labeled with the GAP 4.1 fix 7 command used to construct the field. The number in parentheses is the size of the field.

Table A.16: Power subalgebra, group fixed to SmallGroup(20, 5), run 1.

Field	dim A	dim $Z(A)$	dim A^{ψ_q}	Group	Mult. Table	Matrix
GF(2)	20	20	2	1758	44	166
GF(607)	20	20	8	574	58	137
GF(1483)	20	20	8	85	49	141
GF(2417)	20	20	8	41	50	140
GF(3481)	20	20	20	42	67	149
GF(4523)	20	20	8	41	52	149
GF(5651)	20	20	20	43	49	175
GF(6779)	20	20	12	42	55	167
GF(7921)	20	20	20	53	72	164
GF(9109)	20	20	12	45	54	163
GF(10267)	20	20	8	47	53	150
GF(11503)	20	20	8	44	56	163
GF(12703)	20	20	8	42	56	162
GF(13931)	20	20	20	43	53	189
GF(15227)	20	20	8	43	58	167
GF(16433)	20	20	8	42	52	147
GF(17713)	20	20	8	42	55	153
GF(19051)	20	20	20	44	53	191
GF(20297)	20	20	8	42	57	163
GF(21589)	20	20	12	43	56	169
GF(22861)	20	20	20	44	55	192
GF(24109)	20	20	12	43	57	177
GF(25541)	20	20	20	44	52	189
GF(26833)	20	20	8	42	58	158
GF(28183)	20	20	8	43	59	168
GF(29483)	20	20	8	43	59	168
GF(30893)	20	20	8	42	58	169
GF(32297)	20	20	8	41	59	167
GF(33547)	20	20	8	36	55	160
GF(34871)	20	20	20	44	54	195
GF(36307)	20	20	8	43	59	171
GF(37589)	20	20	12	43	58	179
GF(39047)	20	20	8	44	58	165
GF(40471)	20	20	20	46	56	202
GF(41843)	20	20	8	42	59	171
GF(43103)	20	20	8	42	59	173
GF(44579)	20	20	12	43	59	179
GF(46051)	20	20	20	45	58	199
GF(47501)	20	20	20	46	57	201
GF(48871)	20	20	20	44	60	211
GF(50231)	20	20	20	46	57	200
GF(51647)	20	20	8	43	62	181
GF(53089)	20	20	12	45	59	186
GF(54503)	20	20	8	44	53	177
GF(55927)	20	20	8	44	64	181
GF(57259)	20	20	12	43	63	197
GF(58687)	20	20	8	43	61	182
GF(60101)	20	20	20	45	57	201
GF(61613)	20	20	8	44	60	175
GF(63103)	20	20	8	43	64	189
GF(64591)	20	20	20	46	51	212

Table A.17: Power subalgebra, group fixed to SmallGroup(20, 5), run 2.

Field	dim A	dim $Z(A)$	dim A^{ψ_q}	Group	Mult. Table	Matrix
GF(2)	20	20	2	1743	45	166
GF(607)	20	20	8	547	58	136
GF(1483)	20	20	8	86	50	141
GF(2417)	20	20	8	40	50	133
GF(3481)	20	20	20	42	68	150
GF(4523)	20	20	8	41	53	149
GF(5651)	20	20	20	42	48	173
GF(6779)	20	20	12	40	54	166
GF(7921)	20	20	20	52	70	155
GF(9109)	20	20	12	44	53	162
GF(10267)	20	20	8	45	52	149
GF(11503)	20	20	8	42	56	163
GF(12703)	20	20	8	41	55	160
GF(13931)	20	20	20	42	52	187
GF(15227)	20	20	8	42	57	167
GF(16433)	20	20	8	41	51	146
GF(17713)	20	20	8	41	55	152
GF(19051)	20	20	20	43	53	189
GF(20297)	20	20	8	40	56	161
GF(21589)	20	20	12	42	47	169
GF(22861)	20	20	20	43	54	192
GF(24109)	20	20	12	42	55	175
GF(25541)	20	20	20	43	52	188
GF(26833)	20	20	8	40	56	158
GF(28183)	20	20	8	41	58	166
GF(29483)	20	20	8	41	57	167
GF(30893)	20	20	8	40	57	167
GF(32297)	20	20	8	41	57	166
GF(33547)	20	20	8	42	55	159
GF(34871)	20	20	20	42	54	193
GF(36307)	20	20	8	41	58	161
GF(37589)	20	20	12	42	57	176
GF(39047)	20	20	8	43	57	163
GF(40471)	20	20	20	43	56	201
GF(41843)	20	20	8	40	59	170
GF(43103)	20	20	8	41	58	172
GF(44579)	20	20	12	43	58	178
GF(46051)	20	20	20	44	57	206
GF(47501)	20	20	20	43	56	201
GF(48871)	20	20	20	42	59	210
GF(50231)	20	20	20	45	56	192
GF(51647)	20	20	8	42	61	180
GF(53089)	20	20	12	43	60	183
GF(54503)	20	20	8	43	61	175
GF(55927)	20	20	8	42	60	180
GF(57259)	20	20	12	42	62	195
GF(58687)	20	20	8	42	61	180
GF(60101)	20	20	20	44	55	192
GF(61613)	20	20	8	44	60	173
GF(63103)	20	20	8	42	63	188
GF(64591)	20	20	20	44	59	211

Table A.18: Power subalgebra, group fixed to SmallGroup(20, 5), run 3.

Field	dim A	dim $Z(A)$	dim A^{ψ_q}	Group	Mult. Table	Matrix
GF(2)	20	20	2	1750	44	165
GF(607)	20	20	8	572	58	136
GF(1483)	20	20	8	85	50	140
GF(2417)	20	20	8	40	50	140
GF(3481)	20	20	20	42	67	150
GF(4523)	20	20	8	41	52	149
GF(5651)	20	20	20	42	47	174
GF(6779)	20	20	12	40	54	166
GF(7921)	20	20	20	44	71	162
GF(9109)	20	20	12	44	53	162
GF(10267)	20	20	8	45	52	149
GF(11503)	20	20	8	41	56	155
GF(12703)	20	20	8	41	56	160
GF(13931)	20	20	20	42	52	187
GF(15227)	20	20	8	42	57	168
GF(16433)	20	20	8	41	51	146
GF(17713)	20	20	8	41	54	152
GF(19051)	20	20	20	43	53	189
GF(20297)	20	20	8	40	56	161
GF(21589)	20	20	12	41	55	169
GF(22861)	20	20	20	43	54	191
GF(24109)	20	20	12	42	56	175
GF(25541)	20	20	20	43	52	188
GF(26833)	20	20	8	41	56	157
GF(28183)	20	20	8	42	58	165
GF(29483)	20	20	8	41	57	166
GF(30893)	20	20	8	41	56	168
GF(32297)	20	20	8	40	57	165
GF(33547)	20	20	8	42	54	159
GF(34871)	20	20	20	42	54	185
GF(36307)	20	20	8	41	59	161
GF(37589)	20	20	12	42	57	176
GF(39047)	20	20	8	43	56	163
GF(40471)	20	20	20	44	55	201
GF(41843)	20	20	8	41	58	169
GF(43103)	20	20	8	41	58	171
GF(44579)	20	20	12	43	58	178
GF(46051)	20	20	20	45	57	198
GF(47501)	20	20	20	44	56	200
GF(48871)	20	20	20	43	59	202
GF(50231)	20	20	20	44	57	200
GF(51647)	20	20	8	42	62	180
GF(53089)	20	20	12	43	59	188
GF(54503)	20	20	8	44	62	181
GF(55927)	20	20	8	44	64	180
GF(57259)	20	20	12	43	62	195
GF(58687)	20	20	8	42	61	172
GF(60101)	20	20	20	44	56	200
GF(61613)	20	20	8	44	60	174
GF(63103)	20	20	8	43	63	193
GF(64591)	20	20	20	45	59	216

Table A.19: Power subalgebra, group fixed to SmallGroup(20, 5), run 4.

Field	dim A	dim $Z(A)$	dim A^{ψ_q}	Group	Mult. Table	Matrix
GF(2)	20	20	2	1758	44	165
GF(607)	20	20	8	569	59	136
GF(1483)	20	20	8	85	50	140
GF(2417)	20	20	8	40	50	140
GF(3481)	20	20	20	42	67	150
GF(4523)	20	20	8	40	52	149
GF(5651)	20	20	20	42	47	173
GF(6779)	20	20	12	40	54	166
GF(7921)	20	20	20	52	71	163
GF(9109)	20	20	12	44	52	162
GF(10267)	20	20	8	37	54	146
GF(11503)	20	20	8	42	56	165
GF(12703)	20	20	8	42	56	163
GF(13931)	20	20	20	43	54	192
GF(15227)	20	20	8	43	59	172
GF(16433)	20	20	8	42	52	150
GF(17713)	20	20	8	42	54	153
GF(19051)	20	20	20	43	53	189
GF(20297)	20	20	8	40	48	161
GF(21589)	20	20	12	42	55	160
GF(22861)	20	20	20	43	54	191
GF(24109)	20	20	12	41	57	180
GF(25541)	20	20	20	44	52	191
GF(26833)	20	20	8	41	56	160
GF(28183)	20	20	8	42	51	166
GF(29483)	20	20	8	41	57	167
GF(30893)	20	20	8	41	56	167
GF(32297)	20	20	8	41	57	166
GF(33547)	20	20	8	42	54	159
GF(34871)	20	20	20	42	54	193
GF(36307)	20	20	8	41	58	170
GF(37589)	20	20	12	42	57	177
GF(39047)	20	20	8	43	57	163
GF(40471)	20	20	20	43	55	201
GF(41843)	20	20	8	40	59	170
GF(43103)	20	20	8	41	59	171
GF(44579)	20	20	12	43	58	178
GF(46051)	20	20	20	44	57	206
GF(47501)	20	20	20	44	56	200
GF(48871)	20	20	20	43	59	211
GF(50231)	20	20	20	45	58	207
GF(51647)	20	20	8	43	64	183
GF(53089)	20	20	12	43	59	183
GF(54503)	20	20	8	43	60	175
GF(55927)	20	20	8	42	60	180
GF(57259)	20	20	12	42	62	195
GF(58687)	20	20	8	42	60	181
GF(60101)	20	20	20	44	56	204
GF(61613)	20	20	8	44	61	177
GF(63103)	20	20	8	43	66	190
GF(64591)	20	20	20	45	60	203

Table A.20: Power subalgebra, group fixed to SmallGroup(20, 5), run 5.

Field	dim A	dim $Z(A)$	dim A^{ψ_q}	Group	Mult. Table	Matrix
GF(2)	20	20	2	1759	44	165
GF(607)	20	20	8	573	59	137
GF(1483)	20	20	8	85	50	140
GF(2417)	20	20	8	33	50	140
GF(3481)	20	20	20	43	67	150
GF(4523)	20	20	8	41	52	149
GF(5651)	20	20	20	43	47	173
GF(6779)	20	20	12	41	54	166
GF(7921)	20	20	20	52	70	163
GF(9109)	20	20	12	44	53	163
GF(10267)	20	20	8	45	52	149
GF(11503)	20	20	8	42	56	163
GF(12703)	20	20	8	42	55	152
GF(13931)	20	20	20	43	52	188
GF(15227)	20	20	8	42	57	167
GF(16433)	20	20	8	41	51	145
GF(17713)	20	20	8	41	54	152
GF(19051)	20	20	20	43	53	190
GF(20297)	20	20	8	41	56	162
GF(21589)	20	20	12	42	55	168
GF(22861)	20	20	20	43	54	192
GF(24109)	20	20	12	42	56	175
GF(25541)	20	20	20	42	52	188
GF(26833)	20	20	8	40	56	158
GF(28183)	20	20	8	42	58	166
GF(29483)	20	20	8	41	57	167
GF(30893)	20	20	8	41	56	167
GF(32297)	20	20	8	40	57	167
GF(33547)	20	20	8	42	56	161
GF(34871)	20	20	20	44	55	195
GF(36307)	20	20	8	43	59	170
GF(37589)	20	20	12	42	57	176
GF(39047)	20	20	8	43	56	163
GF(40471)	20	20	20	44	56	201
GF(41843)	20	20	8	41	59	169
GF(43103)	20	20	8	40	58	171
GF(44579)	20	20	12	42	59	178
GF(46051)	20	20	20	44	56	198
GF(47501)	20	20	20	44	57	192
GF(48871)	20	20	20	43	61	215
GF(50231)	20	20	20	45	56	200
GF(51647)	20	20	8	41	61	180
GF(53089)	20	20	12	43	60	184
GF(54503)	20	20	8	43	60	174
GF(55927)	20	20	8	43	60	180
GF(57259)	20	20	12	42	63	187
GF(58687)	20	20	8	41	61	180
GF(60101)	20	20	20	44	55	200
GF(61613)	20	20	8	43	60	173
GF(63103)	20	20	8	42	63	188
GF(64591)	20	20	20	45	59	212

A.4 Primitive Central Idempotents

This section contains experimental data for the primitive central idempotent algorithms discussed in this dissertation. In each case, the algorithm was executed on a group algebra encoded as a group algebra, multiplication table, and regular matrix representation. The center $Z(A)$ and power subalgebra A^{ψ_q} was constructed before executing the algorithm tested, and their runtime is not included in the data below. The group used to define the group algebra is labeled with the `SmallGroup` command used in GAP version 4.1 fix 7. All times are in CPU milliseconds.

A.4.1 Small Field

The field used in each of these experiments is $\mathbb{F}_{53}$.

Friedl and Rónyai's Algorithm

This is the data for FRMain.

Table A.21: Idempotents, Friedl and Rónyai algorithm (small field), run 1.

Group	dim A	dim $Z(A)$	dim A^{ψ_q}	Group	Mult. Table	Matrix
SmallGroup(2, 1)	2	2	2	124	78	52
SmallGroup(3, 1)	3	3	2	94	74	51
SmallGroup(4, 1)	4	4	4	362	223	210
SmallGroup(5, 1)	5	5	2	103	74	58
SmallGroup(6, 1)	6	3	3	224	143	132
SmallGroup(7, 1)	7	7	3	254	144	137
SmallGroup(8, 5)	8	8	8	1092	619	794
SmallGroup(9, 2)	9	9	5	634	323	376
SmallGroup(10, 1)	10	4	3	282	154	162
SmallGroup(11, 1)	11	11	3	369	159	167
SmallGroup(12, 1)	12	6	6	1123	478	686
SmallGroup(13, 1)	13	13	13	18723	4335	7437
SmallGroup(14, 1)	14	5	3	383	164	191
SmallGroup(15, 1)	15	15	5	1183	377	537
SmallGroup(16, 13)	16	10	10	2562	1132	2246
SmallGroup(17, 1)	17	17	3	684	180	218
SmallGroup(18, 2)	18	18	10	5719	1378	2595
SmallGroup(19, 1)	19	19	2	305	89	85
SmallGroup(20, 5)	20	20	8	2582	802	1593
SmallGroup(21, 1)	21	5	4	931	277	448
SmallGroup(22, 2)	22	22	6	2456	586	1062
SmallGroup(23, 1)	23	23	2	437	96	98
SmallGroup(24, 12)	24	5	5	1862	460	934
SmallGroup(25, 1)	25	25	3	1016	195	300
SmallGroup(26, 2)	26	26	26	143970	20350	49443
SmallGroup(27, 5)	27	27	14	13365	2858	7732
SmallGroup(28, 2)	28	28	12	11210	2305	6392
SmallGroup(29, 1)	29	29	5	8436	777	1426
SmallGroup(30, 3)	30	9	5	2577	504	1115
SmallGroup(31, 1)	31	31	2	833	106	130
SmallGroup(32, 30)	32	14	14	8138	2439	9399
SmallGroup(33, 1)	33	33	6	5741	812	1779
SmallGroup(34, 1)	34	10	4	2249	388	785
SmallGroup(35, 1)	35	35	6	6698	874	1952
SmallGroup(36, 13)	36	12	12	8441	2231	9338
SmallGroup(37, 1)	37	37	5	13307	1006	2073
SmallGroup(38, 1)	38	11	3	2306	298	509
SmallGroup(39, 2)	39	39	26	355526	32913	93432
SmallGroup(40, 3)	40	10	8	10061	1427	4345
SmallGroup(41, 1)	41	41	2	2427	147	177
SmallGroup(42, 4)	42	15	6	6696	904	2570
SmallGroup(43, 1)	43	43	3	5389	378	644
SmallGroup(44, 2)	44	44	12	27802	3554	13552
SmallGroup(45, 2)	45	45	14	41549	4824	17619
SmallGroup(46, 2)	46	46	4	5344	599	1212
SmallGroup(47, 1)	47	47	3	5150	425	751
SmallGroup(48, 34)	48	18	16	21100	4281	23531
SmallGroup(49, 2)	49	49	17	140190	10080	37982
SmallGroup(50, 2)	50	50	6	11444	1201	3271

Table A.22: Idempotents, Friedl and Rónyai algorithm (small field), run 2.

Group	dim A	dim $Z(A)$	dim A^{ψ_q}	Group	Mult. Table	Matrix
SmallGroup(2, 1)	2	2	2	127	77	53
SmallGroup(3, 1)	3	3	2	101	77	52
SmallGroup(4, 1)	4	4	4	366	225	213
SmallGroup(5, 1)	5	5	2	104	74	51
SmallGroup(6, 1)	6	3	3	226	144	133
SmallGroup(7, 1)	7	7	3	259	144	137
SmallGroup(8, 5)	8	8	8	1104	628	801
SmallGroup(9, 2)	9	9	5	633	329	384
SmallGroup(10, 1)	10	4	3	288	156	165
SmallGroup(11, 1)	11	11	3	374	162	167
SmallGroup(12, 1)	12	6	6	1118	484	692
SmallGroup(13, 1)	13	13	13	18794	4335	7477
SmallGroup(14, 1)	14	5	3	379	156	191
SmallGroup(15, 1)	15	15	5	1183	379	530
SmallGroup(16, 13)	16	10	10	2565	1135	2247
SmallGroup(17, 1)	17	17	3	685	179	218
SmallGroup(18, 2)	18	18	10	5717	1380	2595
SmallGroup(19, 1)	19	19	2	297	89	86
SmallGroup(20, 5)	20	20	8	2590	803	1596
SmallGroup(21, 1)	21	5	4	930	282	456
SmallGroup(22, 2)	22	22	6	2497	593	1065
SmallGroup(23, 1)	23	23	2	441	96	98
SmallGroup(24, 12)	24	5	5	1881	458	937
SmallGroup(25, 1)	25	25	3	1026	207	301
SmallGroup(26, 2)	26	26	26	144229	20427	49585
SmallGroup(27, 5)	27	27	14	13361	2845	7689
SmallGroup(28, 2)	28	28	12	11218	2296	6371
SmallGroup(29, 1)	29	29	5	8443	786	1446
SmallGroup(30, 3)	30	9	5	2582	518	1117
SmallGroup(31, 1)	31	31	2	848	114	130
SmallGroup(32, 30)	32	14	14	8146	2467	9400
SmallGroup(33, 1)	33	33	6	5739	812	1790
SmallGroup(34, 1)	34	10	4	2269	388	792
SmallGroup(35, 1)	35	35	6	6761	859	1933
SmallGroup(36, 13)	36	12	12	8377	2247	9310
SmallGroup(37, 1)	37	37	5	13304	1012	2084
SmallGroup(38, 1)	38	11	3	2296	299	508
SmallGroup(39, 2)	39	39	26	355456	32736	93422
SmallGroup(40, 3)	40	10	8	10040	1429	4340
SmallGroup(41, 1)	41	41	2	2428	146	169
SmallGroup(42, 4)	42	15	6	6699	905	2568
SmallGroup(43, 1)	43	43	3	5396	377	646
SmallGroup(44, 2)	44	44	12	27865	3554	13538
SmallGroup(45, 2)	45	45	14	41512	4822	17572
SmallGroup(46, 2)	46	46	4	5342	599	1214
SmallGroup(47, 1)	47	47	3	5120	410	744
SmallGroup(48, 34)	48	18	16	20747	4268	23261
SmallGroup(49, 2)	49	49	17	139780	10039	37903
SmallGroup(50, 2)	50	50	6	11418	1191	3317

Table A.23: Idempotents, Friedl and Rónyai algorithm (small field), run 3.

Group	dim A	dim $Z(A)$	dim A^{ψ_q}	Group	Mult. Table	Matrix
SmallGroup(2, 1)	2	2	2	123	75	52
SmallGroup(3, 1)	3	3	2	95	74	52
SmallGroup(4, 1)	4	4	4	358	221	210
SmallGroup(5, 1)	5	5	2	102	65	57
SmallGroup(6, 1)	6	3	3	228	146	132
SmallGroup(7, 1)	7	7	3	258	144	137
SmallGroup(8, 5)	8	8	8	1102	628	799
SmallGroup(9, 2)	9	9	5	635	325	372
SmallGroup(10, 1)	10	4	3	284	154	161
SmallGroup(11, 1)	11	11	3	360	159	167
SmallGroup(12, 1)	12	6	6	1105	477	686
SmallGroup(13, 1)	13	13	13	18682	4340	7432
SmallGroup(14, 1)	14	5	3	382	164	191
SmallGroup(15, 1)	15	15	5	1182	377	535
SmallGroup(16, 13)	16	10	10	2559	1131	2245
SmallGroup(17, 1)	17	17	3	685	178	217
SmallGroup(18, 2)	18	18	10	5685	1368	2614
SmallGroup(19, 1)	19	19	2	306	89	85
SmallGroup(20, 5)	20	20	8	2581	794	1593
SmallGroup(21, 1)	21	5	4	929	276	448
SmallGroup(22, 2)	22	22	6	2478	598	1083
SmallGroup(23, 1)	23	23	2	445	97	97
SmallGroup(24, 12)	24	5	5	1862	460	934
SmallGroup(25, 1)	25	25	3	1016	203	301
SmallGroup(26, 2)	26	26	26	143767	20328	49498
SmallGroup(27, 5)	27	27	14	13452	2875	7732
SmallGroup(28, 2)	28	28	12	11215	2267	6360
SmallGroup(29, 1)	29	29	5	8429	751	1424
SmallGroup(30, 3)	30	9	5	2549	511	1109
SmallGroup(31, 1)	31	31	2	848	113	130
SmallGroup(32, 30)	32	14	14	8110	2475	9388
SmallGroup(33, 1)	33	33	6	5732	818	1779
SmallGroup(34, 1)	34	10	4	2257	387	784
SmallGroup(35, 1)	35	35	6	6685	881	1948
SmallGroup(36, 13)	36	12	12	8367	2234	9255
SmallGroup(37, 1)	37	37	5	13272	1015	2111
SmallGroup(38, 1)	38	11	3	2317	303	500
SmallGroup(39, 2)	39	39	26	357671	33183	93798
SmallGroup(40, 3)	40	10	8	10105	1425	4322
SmallGroup(41, 1)	41	41	2	2427	147	178
SmallGroup(42, 4)	42	15	6	6711	881	2568
SmallGroup(43, 1)	43	43	3	5392	380	645
SmallGroup(44, 2)	44	44	12	27857	3521	13617
SmallGroup(45, 2)	45	45	14	41566	4871	17691
SmallGroup(46, 2)	46	46	4	5342	599	1213
SmallGroup(47, 1)	47	47	3	5093	430	737
SmallGroup(48, 34)	48	18	16	20801	4299	23232
SmallGroup(49, 2)	49	49	17	140041	10073	37995
SmallGroup(50, 2)	50	50	6	11516	1189	3275

Table A.24: Idempotents, Friedl and Rónyai algorithm (small field), run 4.

Group	dim A	dim $Z(A)$	dim A^{ψ_q}	Group	Mult. Table	Matrix
SmallGroup(2, 1)	2	2	2	125	76	52
SmallGroup(3, 1)	3	3	2	96	76	51
SmallGroup(4, 1)	4	4	4	360	222	210
SmallGroup(5, 1)	5	5	2	103	73	58
SmallGroup(6, 1)	6	3	3	224	144	131
SmallGroup(7, 1)	7	7	3	256	144	136
SmallGroup(8, 5)	8	8	8	1090	619	791
SmallGroup(9, 2)	9	9	5	634	322	377
SmallGroup(10, 1)	10	4	3	282	154	161
SmallGroup(11, 1)	11	11	3	369	159	167
SmallGroup(12, 1)	12	6	6	1122	482	690
SmallGroup(13, 1)	13	13	13	18904	4347	7515
SmallGroup(14, 1)	14	5	3	387	167	193
SmallGroup(15, 1)	15	15	5	1187	381	541
SmallGroup(16, 13)	16	10	10	2588	1136	2277
SmallGroup(17, 1)	17	17	3	692	172	220
SmallGroup(18, 2)	18	18	10	5777	1390	2564
SmallGroup(19, 1)	19	19	2	304	88	86
SmallGroup(20, 5)	20	20	8	2590	822	1611
SmallGroup(21, 1)	21	5	4	941	280	453
SmallGroup(22, 2)	22	22	6	2511	592	1064
SmallGroup(23, 1)	23	23	2	438	96	97
SmallGroup(24, 12)	24	5	5	1863	470	934
SmallGroup(25, 1)	25	25	3	1017	203	293
SmallGroup(26, 2)	26	26	26	144148	20458	49397
SmallGroup(27, 5)	27	27	14	13353	2838	7683
SmallGroup(28, 2)	28	28	12	11326	2288	6393
SmallGroup(29, 1)	29	29	5	8468	774	1430
SmallGroup(30, 3)	30	9	5	2599	502	1109
SmallGroup(31, 1)	31	31	2	849	114	131
SmallGroup(32, 30)	32	14	14	8118	2424	9400
SmallGroup(33, 1)	33	33	6	5731	812	1771
SmallGroup(34, 1)	34	10	4	2241	388	785
SmallGroup(35, 1)	35	35	6	6709	883	1939
SmallGroup(36, 13)	36	12	12	8368	2228	9277
SmallGroup(37, 1)	37	37	5	13307	1012	2099
SmallGroup(38, 1)	38	11	3	2283	299	500
SmallGroup(39, 2)	39	39	26	355409	32662	93313
SmallGroup(40, 3)	40	10	8	10047	1428	4321
SmallGroup(41, 1)	41	41	2	2427	146	178
SmallGroup(42, 4)	42	15	6	6674	889	2568
SmallGroup(43, 1)	43	43	3	5409	378	645
SmallGroup(44, 2)	44	44	12	27858	3532	13543
SmallGroup(45, 2)	45	45	14	41525	4811	17516
SmallGroup(46, 2)	46	46	4	5320	600	1206
SmallGroup(47, 1)	47	47	3	5102	424	744
SmallGroup(48, 34)	48	18	16	20737	4279	23241
SmallGroup(49, 2)	49	49	17	139563	10037	37880
SmallGroup(50, 2)	50	50	6	11451	1181	3252

Table A.25: Idempotents, Friedl and Rónyai algorithm (small field), run 5.

Group	dim A	dim $Z(A)$	dim A^{ψ_q}	Group	Mult. Table	Matrix
SmallGroup(2, 1)	2	2	2	122	76	52
SmallGroup(3, 1)	3	3	2	94	74	52
SmallGroup(4, 1)	4	4	4	360	222	210
SmallGroup(5, 1)	5	5	2	102	73	58
SmallGroup(6, 1)	6	3	3	223	144	131
SmallGroup(7, 1)	7	7	3	254	142	135
SmallGroup(8, 5)	8	8	8	1089	620	789
SmallGroup(9, 2)	9	9	5	630	323	375
SmallGroup(10, 1)	10	4	3	280	153	161
SmallGroup(11, 1)	11	11	3	368	159	167
SmallGroup(12, 1)	12	6	6	1121	478	685
SmallGroup(13, 1)	13	13	13	18708	4342	7425
SmallGroup(14, 1)	14	5	3	381	163	183
SmallGroup(15, 1)	15	15	5	1182	376	536
SmallGroup(16, 13)	16	10	10	2545	1131	2261
SmallGroup(17, 1)	17	17	3	684	177	217
SmallGroup(18, 2)	18	18	10	5725	1376	2587
SmallGroup(19, 1)	19	19	2	305	89	85
SmallGroup(20, 5)	20	20	8	2587	795	1593
SmallGroup(21, 1)	21	5	4	929	277	440
SmallGroup(22, 2)	22	22	6	2486	586	1062
SmallGroup(23, 1)	23	23	2	437	96	97
SmallGroup(24, 12)	24	5	5	1854	469	934
SmallGroup(25, 1)	25	25	3	1017	203	293
SmallGroup(26, 2)	26	26	26	143782	20320	49333
SmallGroup(27, 5)	27	27	14	13352	2819	7680
SmallGroup(28, 2)	28	28	12	11223	2294	6349
SmallGroup(29, 1)	29	29	5	8426	766	1400
SmallGroup(30, 3)	30	9	5	2571	511	1109
SmallGroup(31, 1)	31	31	2	848	113	130
SmallGroup(32, 30)	32	14	14	8135	2428	9401
SmallGroup(33, 1)	33	33	6	5724	787	1756
SmallGroup(34, 1)	34	10	4	2249	380	776
SmallGroup(35, 1)	35	35	6	6711	880	1939
SmallGroup(36, 13)	36	12	12	8351	2234	9277
SmallGroup(37, 1)	37	37	5	13281	1010	2100
SmallGroup(38, 1)	38	11	3	2305	290	507
SmallGroup(39, 2)	39	39	26	355367	32655	94077
SmallGroup(40, 3)	40	10	8	10145	1442	4380
SmallGroup(41, 1)	41	41	2	2447	147	179
SmallGroup(42, 4)	42	15	6	6745	912	2578
SmallGroup(43, 1)	43	43	3	5474	382	651
SmallGroup(44, 2)	44	44	12	28112	3565	13542
SmallGroup(45, 2)	45	45	14	41749	4881	17542
SmallGroup(46, 2)	46	46	4	5381	603	1224
SmallGroup(47, 1)	47	47	3	5162	423	742
SmallGroup(48, 34)	48	18	16	20872	4292	23358
SmallGroup(49, 2)	49	49	17	140346	10000	37947
SmallGroup(50, 2)	50	50	6	11539	1201	3325

Gianni, Miller, Trager Algorithm

This is the data for GianniIdempotents.

Table A.26: Idempotents, Gianni, et al. algorithm (small field), run 1.

Group	dim A	dim $Z(A)$	dim A^{ψ_q}	Group	Mult. Table	Matrix
SmallGroup(2, 1)	2	2	2	49	61	23
SmallGroup(3, 1)	3	3	2	48	32	24
SmallGroup(4, 1)	4	4	4	273	128	120
SmallGroup(5, 1)	5	5	2	73	31	26
SmallGroup(6, 1)	6	3	3	154	83	67
SmallGroup(7, 1)	7	7	3	196	78	74
SmallGroup(8, 5)	8	8	8	1271	645	617
SmallGroup(9, 2)	9	9	5	652	214	246
SmallGroup(10, 1)	10	4	3	404	102	91
SmallGroup(11, 1)	11	11	3	660	90	94
SmallGroup(12, 1)	12	6	6	1273	372	526
SmallGroup(13, 1)	13	13	13	7258	2092	2983
SmallGroup(14, 1)	14	5	3	426	130	112
SmallGroup(15, 1)	15	15	5	1818	317	375
SmallGroup(16, 13)	16	10	10	5929	1226	2113
SmallGroup(17, 1)	17	17	3	912	116	143
SmallGroup(18, 2)	18	18	10	9363	1499	2214
SmallGroup(19, 1)	19	19	2	494	49	57
SmallGroup(20, 5)	20	20	8	6361	1002	1721
SmallGroup(21, 1)	21	5	4	1897	277	306
SmallGroup(22, 2)	22	22	6	5919	781	1054
SmallGroup(23, 1)	23	23	2	576	56	76
SmallGroup(24, 12)	24	5	5	4227	494	630
SmallGroup(25, 1)	25	25	3	2804	147	192
SmallGroup(26, 2)	26	26	26	102857	18588	37558
SmallGroup(27, 5)	27	27	14	27739	4679	8441
SmallGroup(28, 2)	28	28	12	28913	3554	6669
SmallGroup(29, 1)	29	29	5	8346	600	937
SmallGroup(30, 3)	30	9	5	6688	626	907
SmallGroup(31, 1)	31	31	2	1230	79	88
SmallGroup(32, 30)	32	14	14	40788	5243	12283
SmallGroup(33, 1)	33	33	6	11781	896	1660
SmallGroup(34, 1)	34	10	4	5737	537	582
SmallGroup(35, 1)	35	35	6	17748	1308	1836
SmallGroup(36, 13)	36	12	12	26954	4426	9085
SmallGroup(37, 1)	37	37	5	11745	967	1458
SmallGroup(38, 1)	38	11	3	3488	465	367
SmallGroup(39, 2)	39	39	26	214540	31444	73797
SmallGroup(40, 3)	40	10	8	27290	2352	4594
SmallGroup(41, 1)	41	41	2	7032	112	163
SmallGroup(42, 4)	42	15	6	20329	1551	2466
SmallGroup(43, 1)	43	43	3	9591	363	494
SmallGroup(44, 2)	44	44	12	89631	7721	14456
SmallGroup(45, 2)	45	45	14	119595	9723	23676
SmallGroup(46, 2)	46	46	4	9063	1062	1010
SmallGroup(47, 1)	47	47	3	8044	449	593
SmallGroup(48, 34)	48	18	16	73811	11251	33372
SmallGroup(49, 2)	49	49	17	179757	16674	42745
SmallGroup(50, 2)	50	50	6	37345	2196	3516

Table A.27: Idempotents, Gianni, et al. algorithm (small field), run 2.

Group	dim A	dim $Z(A)$	dim A^{ψ_q}	Group	Mult. Table	Matrix
SmallGroup(2, 1)	2	2	2	48	60	22
SmallGroup(3, 1)	3	3	2	48	31	23
SmallGroup(4, 1)	4	4	4	269	128	122
SmallGroup(5, 1)	5	5	2	74	32	28
SmallGroup(6, 1)	6	3	3	156	83	68
SmallGroup(7, 1)	7	7	3	197	77	75
SmallGroup(8, 5)	8	8	8	1287	654	624
SmallGroup(9, 2)	9	9	5	653	215	246
SmallGroup(10, 1)	10	4	3	396	102	91
SmallGroup(11, 1)	11	11	3	663	83	93
SmallGroup(12, 1)	12	6	6	1273	373	527
SmallGroup(13, 1)	13	13	13	7248	2082	2982
SmallGroup(14, 1)	14	5	3	418	129	109
SmallGroup(15, 1)	15	15	5	1796	304	375
SmallGroup(16, 13)	16	10	10	5954	1237	2120
SmallGroup(17, 1)	17	17	3	903	117	144
SmallGroup(18, 2)	18	18	10	9462	1511	2206
SmallGroup(19, 1)	19	19	2	489	50	57
SmallGroup(20, 5)	20	20	8	6366	1002	1735
SmallGroup(21, 1)	21	5	4	1911	287	304
SmallGroup(22, 2)	22	22	6	5904	776	1063
SmallGroup(23, 1)	23	23	2	582	55	76
SmallGroup(24, 12)	24	5	5	4312	496	629
SmallGroup(25, 1)	25	25	3	2802	147	203
SmallGroup(26, 2)	26	26	26	103235	18705	37757
SmallGroup(27, 5)	27	27	14	27678	4686	8477
SmallGroup(28, 2)	28	28	12	28821	3523	6626
SmallGroup(29, 1)	29	29	5	8332	601	945
SmallGroup(30, 3)	30	9	5	6670	622	912
SmallGroup(31, 1)	31	31	2	1222	78	88
SmallGroup(32, 30)	32	14	14	40773	5259	12309
SmallGroup(33, 1)	33	33	6	11849	888	1622
SmallGroup(34, 1)	34	10	4	5710	553	597
SmallGroup(35, 1)	35	35	6	17723	1294	1844
SmallGroup(36, 13)	36	12	12	27077	4398	9084
SmallGroup(37, 1)	37	37	5	11696	956	1468
SmallGroup(38, 1)	38	11	3	3439	459	367
SmallGroup(39, 2)	39	39	26	215010	31497	73304
SmallGroup(40, 3)	40	10	8	27093	2317	4531
SmallGroup(41, 1)	41	41	2	6991	111	165
SmallGroup(42, 4)	42	15	6	20421	1550	2458
SmallGroup(43, 1)	43	43	3	9577	366	494
SmallGroup(44, 2)	44	44	12	89843	7702	14514
SmallGroup(45, 2)	45	45	14	119301	9839	23423
SmallGroup(46, 2)	46	46	4	8992	1073	1019
SmallGroup(47, 1)	47	47	3	7949	458	584
SmallGroup(48, 34)	48	18	16	73600	11208	33167
SmallGroup(49, 2)	49	49	17	179441	16530	42338
SmallGroup(50, 2)	50	50	6	37009	2164	3530

Table A.28: Idempotents, Gianni, et al. algorithm (small field), run 3.

Group	dim A	dim $Z(A)$	dim A^{ψ_q}	Group	Mult. Table	Matrix
SmallGroup(2, 1)	2	2	2	48	61	22
SmallGroup(3, 1)	3	3	2	47	32	23
SmallGroup(4, 1)	4	4	4	269	128	120
SmallGroup(5, 1)	5	5	2	73	31	27
SmallGroup(6, 1)	6	3	3	154	82	67
SmallGroup(7, 1)	7	7	3	195	77	76
SmallGroup(8, 5)	8	8	8	1296	645	618
SmallGroup(9, 2)	9	9	5	631	213	244
SmallGroup(10, 1)	10	4	3	399	101	90
SmallGroup(11, 1)	11	11	3	655	88	91
SmallGroup(12, 1)	12	6	6	1252	368	521
SmallGroup(13, 1)	13	13	13	7193	2090	2983
SmallGroup(14, 1)	14	5	3	417	121	109
SmallGroup(15, 1)	15	15	5	1812	312	369
SmallGroup(16, 13)	16	10	10	5875	1207	2098
SmallGroup(17, 1)	17	17	3	895	115	142
SmallGroup(18, 2)	18	18	10	9361	1495	2183
SmallGroup(19, 1)	19	19	2	493	49	56
SmallGroup(20, 5)	20	20	8	6291	998	1696
SmallGroup(21, 1)	21	5	4	1884	284	297
SmallGroup(22, 2)	22	22	6	5909	778	1052
SmallGroup(23, 1)	23	23	2	575	56	76
SmallGroup(24, 12)	24	5	5	4271	492	636
SmallGroup(25, 1)	25	25	3	2793	146	202
SmallGroup(26, 2)	26	26	26	102638	18506	37482
SmallGroup(27, 5)	27	27	14	27477	4631	8435
SmallGroup(28, 2)	28	28	12	28616	3532	6595
SmallGroup(29, 1)	29	29	5	8240	601	930
SmallGroup(30, 3)	30	9	5	6724	621	912
SmallGroup(31, 1)	31	31	2	1213	77	87
SmallGroup(32, 30)	32	14	14	40525	5187	12217
SmallGroup(33, 1)	33	33	6	11734	886	1644
SmallGroup(34, 1)	34	10	4	5708	545	593
SmallGroup(35, 1)	35	35	6	17589	1308	1829
SmallGroup(36, 13)	36	12	12	26790	4372	9014
SmallGroup(37, 1)	37	37	5	11626	948	1467
SmallGroup(38, 1)	38	11	3	3452	459	363
SmallGroup(39, 2)	39	39	26	213789	31174	73146
SmallGroup(40, 3)	40	10	8	27048	2321	4538
SmallGroup(41, 1)	41	41	2	6989	111	162
SmallGroup(42, 4)	42	15	6	20243	1533	2450
SmallGroup(43, 1)	43	43	3	9493	363	489
SmallGroup(44, 2)	44	44	12	89179	7606	14398
SmallGroup(45, 2)	45	45	14	118806	9735	23398
SmallGroup(46, 2)	46	46	4	8954	1071	1016
SmallGroup(47, 1)	47	47	3	7928	434	579
SmallGroup(48, 34)	48	18	16	73128	11145	32986
SmallGroup(49, 2)	49	49	17	179017	16479	42269
SmallGroup(50, 2)	50	50	6	37019	2172	3505

Table A.29: Idempotents, Gianni, et al. algorithm (small field), run 4.

Group	dim A	dim $Z(A)$	dim A^{ψ_q}	Group	Mult. Table	Matrix
SmallGroup(2, 1)	2	2	2	49	59	22
SmallGroup(3, 1)	3	3	2	48	32	23
SmallGroup(4, 1)	4	4	4	270	127	122
SmallGroup(5, 1)	5	5	2	74	31	27
SmallGroup(6, 1)	6	3	3	155	82	67
SmallGroup(7, 1)	7	7	3	196	77	74
SmallGroup(8, 5)	8	8	8	1302	646	618
SmallGroup(9, 2)	9	9	5	648	214	243
SmallGroup(10, 1)	10	4	3	400	101	82
SmallGroup(11, 1)	11	11	3	657	89	92
SmallGroup(12, 1)	12	6	6	1270	369	521
SmallGroup(13, 1)	13	13	13	7197	2084	2979
SmallGroup(14, 1)	14	5	3	418	129	110
SmallGroup(15, 1)	15	15	5	1813	312	370
SmallGroup(16, 13)	16	10	10	5877	1232	2116
SmallGroup(17, 1)	17	17	3	903	114	142
SmallGroup(18, 2)	18	18	10	9334	1492	2178
SmallGroup(19, 1)	19	19	2	493	50	56
SmallGroup(20, 5)	20	20	8	6304	1000	1721
SmallGroup(21, 1)	21	5	4	1896	284	305
SmallGroup(22, 2)	22	22	6	5906	787	1053
SmallGroup(23, 1)	23	23	2	576	55	76
SmallGroup(24, 12)	24	5	5	4265	493	637
SmallGroup(25, 1)	25	25	3	2795	147	200
SmallGroup(26, 2)	26	26	26	102534	18525	37502
SmallGroup(27, 5)	27	27	14	27503	4632	8424
SmallGroup(28, 2)	28	28	12	28643	3543	6600
SmallGroup(29, 1)	29	29	5	8286	603	938
SmallGroup(30, 3)	30	9	5	6613	616	912
SmallGroup(31, 1)	31	31	2	1212	77	87
SmallGroup(32, 30)	32	14	14	40567	5228	12212
SmallGroup(33, 1)	33	33	6	11746	886	1647
SmallGroup(34, 1)	34	10	4	5717	546	589
SmallGroup(35, 1)	35	35	6	17592	1300	1830
SmallGroup(36, 13)	36	12	12	26828	4373	9008
SmallGroup(37, 1)	37	37	5	11640	957	1469
SmallGroup(38, 1)	38	11	3	3441	460	350
SmallGroup(39, 2)	39	39	26	213699	31219	73235
SmallGroup(40, 3)	40	10	8	27083	2326	4551
SmallGroup(41, 1)	41	41	2	7000	111	162
SmallGroup(42, 4)	42	15	6	20242	1518	2421
SmallGroup(43, 1)	43	43	3	9500	363	490
SmallGroup(44, 2)	44	44	12	89144	7609	14414
SmallGroup(45, 2)	45	45	14	118752	9728	23425
SmallGroup(46, 2)	46	46	4	8955	1073	1017
SmallGroup(47, 1)	47	47	3	7959	443	587
SmallGroup(48, 34)	48	18	16	73206	11162	33024
SmallGroup(49, 2)	49	49	17	179108	16521	42334
SmallGroup(50, 2)	50	50	6	37030	2165	3506

Table A.30: Idempotents, Gianni, et al. algorithm (small field), run 5.

Group	dim A	dim $Z(A)$	dim A^{ψ_q}	Group	Mult. Table	Matrix
SmallGroup(2, 1)	2	2	2	48	60	22
SmallGroup(3, 1)	3	3	2	47	32	23
SmallGroup(4, 1)	4	4	4	267	128	120
SmallGroup(5, 1)	5	5	2	72	30	25
SmallGroup(6, 1)	6	3	3	154	82	65
SmallGroup(7, 1)	7	7	3	194	76	74
SmallGroup(8, 5)	8	8	8	1289	645	618
SmallGroup(9, 2)	9	9	5	647	213	243
SmallGroup(10, 1)	10	4	3	399	100	91
SmallGroup(11, 1)	11	11	3	649	89	92
SmallGroup(12, 1)	12	6	6	1260	368	520
SmallGroup(13, 1)	13	13	13	7186	2068	2968
SmallGroup(14, 1)	14	5	3	417	129	109
SmallGroup(15, 1)	15	15	5	1810	304	369
SmallGroup(16, 13)	16	10	10	5872	1234	2107
SmallGroup(17, 1)	17	17	3	903	115	143
SmallGroup(18, 2)	18	18	10	9363	1489	2185
SmallGroup(19, 1)	19	19	2	493	49	56
SmallGroup(20, 5)	20	20	8	6325	992	1719
SmallGroup(21, 1)	21	5	4	1895	285	304
SmallGroup(22, 2)	22	22	6	5911	786	1053
SmallGroup(23, 1)	23	23	2	575	55	75
SmallGroup(24, 12)	24	5	5	4272	493	637
SmallGroup(25, 1)	25	25	3	2802	147	200
SmallGroup(26, 2)	26	26	26	102503	18533	37481
SmallGroup(27, 5)	27	27	14	27485	4652	8395
SmallGroup(28, 2)	28	28	12	28662	3543	6598
SmallGroup(29, 1)	29	29	5	8273	602	937
SmallGroup(30, 3)	30	9	5	6621	616	911
SmallGroup(31, 1)	31	31	2	1220	78	88
SmallGroup(32, 30)	32	14	14	40528	5219	12196
SmallGroup(33, 1)	33	33	6	11752	887	1645
SmallGroup(34, 1)	34	10	4	5694	547	588
SmallGroup(35, 1)	35	35	6	17581	1307	1829
SmallGroup(36, 13)	36	12	12	26822	4364	9003
SmallGroup(37, 1)	37	37	5	11644	968	1468
SmallGroup(38, 1)	38	11	3	3453	460	364
SmallGroup(39, 2)	39	39	26	213590	31203	73231
SmallGroup(40, 3)	40	10	8	27078	2324	4531
SmallGroup(41, 1)	41	41	2	6989	111	162
SmallGroup(42, 4)	42	15	6	20280	1526	2451
SmallGroup(43, 1)	43	43	3	9500	363	489
SmallGroup(44, 2)	44	44	12	89107	7607	14404
SmallGroup(45, 2)	45	45	14	118714	9727	23379
SmallGroup(46, 2)	46	46	4	8989	1074	1016
SmallGroup(47, 1)	47	47	3	7942	442	588
SmallGroup(48, 34)	48	18	16	73171	11165	33024
SmallGroup(49, 2)	49	49	17	179056	16497	42308
SmallGroup(50, 2)	50	50	6	37016	2170	3490

Davis' Algorithm

This is the data for DavisIdempotents.

Table A.31: Idempotents, Davis algorithm (small field), run 1.

Group	dim A	dim $Z(A)$	dim A^{ψ_q}	Group	Mult. Table	Matrix
SmallGroup(2, 1)	2	2	2	31	21	9
SmallGroup(3, 1)	3	3	2	32	21	10
SmallGroup(4, 1)	4	4	4	64	38	18
SmallGroup(5, 1)	5	5	2	47	24	4
SmallGroup(6, 1)	6	3	3	70	34	18
SmallGroup(7, 1)	7	7	3	82	35	20
SmallGroup(8, 5)	8	8	8	193	106	80
SmallGroup(9, 2)	9	9	5	135	73	53
SmallGroup(10, 1)	10	4	3	142	52	29
SmallGroup(11, 1)	11	11	3	208	57	36
SmallGroup(12, 1)	12	6	6	253	107	97
SmallGroup(13, 1)	13	13	13	510	224	301
SmallGroup(14, 1)	14	5	3	218	79	42
SmallGroup(15, 1)	15	15	5	351	125	117
SmallGroup(16, 13)	16	10	10	384	194	261
SmallGroup(17, 1)	17	17	3	457	104	63
SmallGroup(18, 2)	18	18	10	718	224	305
SmallGroup(19, 1)	19	19	2	566	118	54
SmallGroup(20, 5)	20	20	8	464	213	270
SmallGroup(21, 1)	21	5	4	997	206	146
SmallGroup(22, 2)	22	22	6	844	235	241
SmallGroup(23, 1)	23	23	2	665	165	64
SmallGroup(24, 12)	24	5	5	692	242	184
SmallGroup(25, 1)	25	25	3	1311	223	137
SmallGroup(26, 2)	26	26	26	2642	969	3217
SmallGroup(27, 5)	27	27	14	1690	694	2108
SmallGroup(28, 2)	28	28	12	1528	474	882
SmallGroup(29, 1)	29	29	5	2021	370	358
SmallGroup(30, 3)	30	9	5	1262	423	383
SmallGroup(31, 1)	31	31	2	1367	335	115
SmallGroup(32, 30)	32	14	14	2075	1001	3461
SmallGroup(33, 1)	33	33	6	1324	516	481
SmallGroup(34, 1)	34	10	4	1666	554	360
SmallGroup(35, 1)	35	35	6	948	545	476
SmallGroup(36, 13)	36	12	12	1980	1035	3618
SmallGroup(37, 1)	37	37	5	2298	594	475
SmallGroup(38, 1)	38	11	3	2161	681	260
SmallGroup(39, 2)	39	39	26	5121	1568	6675
SmallGroup(40, 3)	40	10	8	2736	902	1325
SmallGroup(41, 1)	41	41	2	5981	723	279
SmallGroup(42, 4)	42	15	6	770	837	582
SmallGroup(43, 1)	43	43	3	4467	840	456
SmallGroup(44, 2)	44	44	12	4824	1259	2310
SmallGroup(45, 2)	45	45	14	4754	1596	5704
SmallGroup(46, 2)	46	46	4	1884	1126	524
SmallGroup(47, 1)	47	47	3	4457	1072	557
SmallGroup(48, 34)	48	18	16	2907	1734	6108
SmallGroup(49, 2)	49	49	17	7557	2166	9494
SmallGroup(50, 2)	50	50	6	5336	1595	1205

Table A.32: Idempotents, Davis algorithm (small field), run 2.

Group	dim A	dim $Z(A)$	dim A^{ψ_q}	Group	Mult. Table	Matrix
SmallGroup(2, 1)	2	2	2	31	20	8
SmallGroup(3, 1)	3	3	2	32	19	10
SmallGroup(4, 1)	4	4	4	63	37	18
SmallGroup(5, 1)	5	5	2	46	24	11
SmallGroup(6, 1)	6	3	3	70	36	18
SmallGroup(7, 1)	7	7	3	82	36	18
SmallGroup(8, 5)	8	8	8	185	106	79
SmallGroup(9, 2)	9	9	5	134	73	54
SmallGroup(10, 1)	10	4	3	143	52	30
SmallGroup(11, 1)	11	11	3	208	57	36
SmallGroup(12, 1)	12	6	6	253	108	96
SmallGroup(13, 1)	13	13	13	502	225	301
SmallGroup(14, 1)	14	5	3	218	79	42
SmallGroup(15, 1)	15	15	5	352	125	117
SmallGroup(16, 13)	16	10	10	383	194	252
SmallGroup(17, 1)	17	17	3	456	104	62
SmallGroup(18, 2)	18	18	10	727	223	312
SmallGroup(19, 1)	19	19	2	567	118	54
SmallGroup(20, 5)	20	20	8	464	213	270
SmallGroup(21, 1)	21	5	4	998	207	147
SmallGroup(22, 2)	22	22	6	845	227	241
SmallGroup(23, 1)	23	23	2	673	165	64
SmallGroup(24, 12)	24	5	5	693	242	185
SmallGroup(25, 1)	25	25	3	1313	231	138
SmallGroup(26, 2)	26	26	26	2636	970	3219
SmallGroup(27, 5)	27	27	14	1676	687	2094
SmallGroup(28, 2)	28	28	12	1529	474	883
SmallGroup(29, 1)	29	29	5	2036	371	358
SmallGroup(30, 3)	30	9	5	1264	423	384
SmallGroup(31, 1)	31	31	2	1369	335	114
SmallGroup(32, 30)	32	14	14	2063	993	3463
SmallGroup(33, 1)	33	33	6	1310	516	489
SmallGroup(34, 1)	34	10	4	1666	554	360
SmallGroup(35, 1)	35	35	6	940	561	477
SmallGroup(36, 13)	36	12	12	1982	1028	3641
SmallGroup(37, 1)	37	37	5	2291	586	468
SmallGroup(38, 1)	38	11	3	2161	682	268
SmallGroup(39, 2)	39	39	26	5115	1569	6693
SmallGroup(40, 3)	40	10	8	2736	911	1326
SmallGroup(41, 1)	41	41	2	5981	723	279
SmallGroup(42, 4)	42	15	6	779	838	582
SmallGroup(43, 1)	43	43	3	4472	824	455
SmallGroup(44, 2)	44	44	12	4836	1253	2304
SmallGroup(45, 2)	45	45	14	4764	1612	5690
SmallGroup(46, 2)	46	46	4	1878	1134	525
SmallGroup(47, 1)	47	47	3	4469	1074	558
SmallGroup(48, 34)	48	18	16	2917	1735	6117
SmallGroup(49, 2)	49	49	17	7581	2159	9498
SmallGroup(50, 2)	50	50	6	5339	1594	1230

Table A.33: Idempotents, Davis algorithm (small field), run 3.

Group	dim A	dim $Z(A)$	dim A^{ψ_q}	Group	Mult. Table	Matrix
SmallGroup(2, 1)	2	2	2	30	21	8
SmallGroup(3, 1)	3	3	2	31	20	10
SmallGroup(4, 1)	4	4	4	63	37	19
SmallGroup(5, 1)	5	5	2	47	25	11
SmallGroup(6, 1)	6	3	3	69	34	18
SmallGroup(7, 1)	7	7	3	82	35	20
SmallGroup(8, 5)	8	8	8	191	106	78
SmallGroup(9, 2)	9	9	5	134	73	53
SmallGroup(10, 1)	10	4	3	143	52	29
SmallGroup(11, 1)	11	11	3	208	56	36
SmallGroup(12, 1)	12	6	6	244	108	96
SmallGroup(13, 1)	13	13	13	509	224	302
SmallGroup(14, 1)	14	5	3	217	78	42
SmallGroup(15, 1)	15	15	5	352	125	117
SmallGroup(16, 13)	16	10	10	383	193	252
SmallGroup(17, 1)	17	17	3	455	104	62
SmallGroup(18, 2)	18	18	10	726	224	313
SmallGroup(19, 1)	19	19	2	568	117	55
SmallGroup(20, 5)	20	20	8	464	213	271
SmallGroup(21, 1)	21	5	4	998	206	147
SmallGroup(22, 2)	22	22	6	845	235	241
SmallGroup(23, 1)	23	23	2	674	164	64
SmallGroup(24, 12)	24	5	5	693	241	185
SmallGroup(25, 1)	25	25	3	1321	231	138
SmallGroup(26, 2)	26	26	26	2626	945	3201
SmallGroup(27, 5)	27	27	14	1682	695	2101
SmallGroup(28, 2)	28	28	12	1527	473	874
SmallGroup(29, 1)	29	29	5	2020	370	357
SmallGroup(30, 3)	30	9	5	1262	423	384
SmallGroup(31, 1)	31	31	2	1368	335	114
SmallGroup(32, 30)	32	14	14	2077	1000	3469
SmallGroup(33, 1)	33	33	6	1324	516	489
SmallGroup(34, 1)	34	10	4	1665	553	360
SmallGroup(35, 1)	35	35	6	947	561	476
SmallGroup(36, 13)	36	12	12	1980	1026	3641
SmallGroup(37, 1)	37	37	5	2290	594	474
SmallGroup(38, 1)	38	11	3	2168	681	267
SmallGroup(39, 2)	39	39	26	5114	1552	6689
SmallGroup(40, 3)	40	10	8	2735	909	1316
SmallGroup(41, 1)	41	41	2	5998	723	280
SmallGroup(42, 4)	42	15	6	777	838	583
SmallGroup(43, 1)	43	43	3	4457	840	440
SmallGroup(44, 2)	44	44	12	4837	1254	2311
SmallGroup(45, 2)	45	45	14	4746	1612	5698
SmallGroup(46, 2)	46	46	4	1893	1135	524
SmallGroup(47, 1)	47	47	3	4466	1073	558
SmallGroup(48, 34)	48	18	16	2898	1734	6093
SmallGroup(49, 2)	49	49	17	7568	2158	9461
SmallGroup(50, 2)	50	50	6	5320	1593	1221

Table A.34: Idempotents, Davis algorithm (small field), run 4.

Group	$\dim A$	$\dim Z(A)$	$\dim A^{\psi_q}$	Group	Mult. Table	Matrix
SmallGroup(2, 1)	2	2	2	31	21	8
SmallGroup(3, 1)	3	3	2	33	20	9
SmallGroup(4, 1)	4	4	4	64	38	18
SmallGroup(5, 1)	5	5	2	47	25	11
SmallGroup(6, 1)	6	3	3	70	35	18
SmallGroup(7, 1)	7	7	3	82	36	19
SmallGroup(8, 5)	8	8	8	193	104	78
SmallGroup(9, 2)	9	9	5	135	73	53
SmallGroup(10, 1)	10	4	3	143	52	29
SmallGroup(11, 1)	11	11	3	200	57	36
SmallGroup(12, 1)	12	6	6	246	108	96
SmallGroup(13, 1)	13	13	13	510	225	301
SmallGroup(14, 1)	14	5	3	218	80	42
SmallGroup(15, 1)	15	15	5	351	125	118
SmallGroup(16, 13)	16	10	10	384	194	259
SmallGroup(17, 1)	17	17	3	456	105	62
SmallGroup(18, 2)	18	18	10	727	224	313
SmallGroup(19, 1)	19	19	2	567	117	55
SmallGroup(20, 5)	20	20	8	464	212	262
SmallGroup(21, 1)	21	5	4	998	207	146
SmallGroup(22, 2)	22	22	6	846	235	241
SmallGroup(23, 1)	23	23	2	666	164	64
SmallGroup(24, 12)	24	5	5	692	242	184
SmallGroup(25, 1)	25	25	3	1305	231	138
SmallGroup(26, 2)	26	26	26	2627	969	3209
SmallGroup(27, 5)	27	27	14	1683	695	2100
SmallGroup(28, 2)	28	28	12	1528	474	866
SmallGroup(29, 1)	29	29	5	2029	370	357
SmallGroup(30, 3)	30	9	5	1262	423	384
SmallGroup(31, 1)	31	31	2	1360	335	115
SmallGroup(32, 30)	32	14	14	2079	1002	3462
SmallGroup(33, 1)	33	33	6	1325	516	488
SmallGroup(34, 1)	34	10	4	1664	553	360
SmallGroup(35, 1)	35	35	6	947	560	468
SmallGroup(36, 13)	36	12	12	1974	1035	3651
SmallGroup(37, 1)	37	37	5	2298	594	475
SmallGroup(38, 1)	38	11	3	2163	681	268
SmallGroup(39, 2)	39	39	26	5126	1560	6697
SmallGroup(40, 3)	40	10	8	2737	910	1324
SmallGroup(41, 1)	41	41	2	6000	723	279
SmallGroup(42, 4)	42	15	6	778	838	583
SmallGroup(43, 1)	43	43	3	4458	840	448
SmallGroup(44, 2)	44	44	12	4811	1237	2288
SmallGroup(45, 2)	45	45	14	4765	1613	5705
SmallGroup(46, 2)	46	46	4	1885	1134	524
SmallGroup(47, 1)	47	47	3	4473	1072	558
SmallGroup(48, 34)	48	18	16	2924	1733	6116
SmallGroup(49, 2)	49	49	17	7573	2168	9477
SmallGroup(50, 2)	50	50	6	5329	1593	1212

Table A.35: Idempotents, Davis algorithm (small field), run 5.

Group	dim A	dim $Z(A)$	dim A^{ψ_q}	Group	Mult. Table	Matrix
SmallGroup(2, 1)	2	2	2	31	20	8
SmallGroup(3, 1)	3	3	2	32	20	9
SmallGroup(4, 1)	4	4	4	63	38	18
SmallGroup(5, 1)	5	5	2	47	25	12
SmallGroup(6, 1)	6	3	3	69	34	18
SmallGroup(7, 1)	7	7	3	81	37	20
SmallGroup(8, 5)	8	8	8	193	105	80
SmallGroup(9, 2)	9	9	5	135	73	47
SmallGroup(10, 1)	10	4	3	142	53	29
SmallGroup(11, 1)	11	11	3	208	57	36
SmallGroup(12, 1)	12	6	6	253	109	96
SmallGroup(13, 1)	13	13	13	510	225	301
SmallGroup(14, 1)	14	5	3	218	80	42
SmallGroup(15, 1)	15	15	5	352	124	118
SmallGroup(16, 13)	16	10	10	384	194	260
SmallGroup(17, 1)	17	17	3	457	104	62
SmallGroup(18, 2)	18	18	10	726	223	313
SmallGroup(19, 1)	19	19	2	567	118	55
SmallGroup(20, 5)	20	20	8	457	213	270
SmallGroup(21, 1)	21	5	4	998	207	147
SmallGroup(22, 2)	22	22	6	846	235	242
SmallGroup(23, 1)	23	23	2	674	157	64
SmallGroup(24, 12)	24	5	5	693	234	185
SmallGroup(25, 1)	25	25	3	1321	223	137
SmallGroup(26, 2)	26	26	26	2635	969	3219
SmallGroup(27, 5)	27	27	14	1691	695	2110
SmallGroup(28, 2)	28	28	12	1528	474	883
SmallGroup(29, 1)	29	29	5	2031	370	357
SmallGroup(30, 3)	30	9	5	1255	422	384
SmallGroup(31, 1)	31	31	2	1360	335	114
SmallGroup(32, 30)	32	14	14	2070	1000	3460
SmallGroup(33, 1)	33	33	6	1324	516	481
SmallGroup(34, 1)	34	10	4	1664	546	360
SmallGroup(35, 1)	35	35	6	947	560	476
SmallGroup(36, 13)	36	12	12	1973	1035	3649
SmallGroup(37, 1)	37	37	5	2298	594	475
SmallGroup(38, 1)	38	11	3	2170	681	268
SmallGroup(39, 2)	39	39	26	5133	1569	6696
SmallGroup(40, 3)	40	10	8	2730	910	1325
SmallGroup(41, 1)	41	41	2	5994	723	279
SmallGroup(42, 4)	42	15	6	778	837	583
SmallGroup(43, 1)	43	43	3	4472	839	455
SmallGroup(44, 2)	44	44	12	4845	1256	2305
SmallGroup(45, 2)	45	45	14	4784	1603	5692
SmallGroup(46, 2)	46	46	4	1883	1125	528
SmallGroup(47, 1)	47	47	3	4511	1085	562
SmallGroup(48, 34)	48	18	16	2925	1732	6166
SmallGroup(49, 2)	49	49	17	7639	2193	9487
SmallGroup(50, 2)	50	50	6	5336	1591	1229

Eberly and Giesbrecht Algorithm

This is the data for EberlyGiesbrechtSmall.

Table A.36: Idempotents, Eberly and Giesbrecht algorithm (small field), run 1.

Group	$\dim A$	$\dim Z(A)$	$\dim A^{\psi_q}$	Group	Mult. Table	Matrix
SmallGroup(2, 1)	2	2	2	82	68	58
SmallGroup(3, 1)	3	3	2	87	69	60
SmallGroup(4, 1)	4	4	4	261	199	262
SmallGroup(5, 1)	5	5	2	95	72	67
SmallGroup(6, 1)	6	3	3	177	126	180
SmallGroup(7, 1)	7	7	3	189	131	124
SmallGroup(8, 5)	8	8	8	1120	755	722
SmallGroup(9, 2)	9	9	5	510	354	308
SmallGroup(10, 1)	10	4	3	224	130	130
SmallGroup(11, 1)	11	11	3	246	137	134
SmallGroup(12, 1)	12	6	6	868	454	460
SmallGroup(13, 1)	13	13	13	4052	1936	2111
SmallGroup(14, 1)	14	5	3	310	143	147
SmallGroup(15, 1)	15	15	5	821	346	356
SmallGroup(16, 13)	16	10	10	3021	1257	1327
SmallGroup(17, 1)	17	17	3	426	150	248
SmallGroup(18, 2)	18	18	10	3861	1339	1414
SmallGroup(19, 1)	19	19	2	256	84	88
SmallGroup(20, 5)	20	20	8	2864	920	1002
SmallGroup(21, 1)	21	5	4	868	260	278
SmallGroup(22, 2)	22	22	6	1950	606	600
SmallGroup(23, 1)	23	23	2	363	89	100
SmallGroup(24, 12)	24	5	5	1666	403	445
SmallGroup(25, 1)	25	25	3	725	166	197
SmallGroup(26, 2)	26	26	26	39599	9286	12042
SmallGroup(27, 5)	27	27	14	12537	3126	3423
SmallGroup(28, 2)	28	28	12	13603	2393	2678
SmallGroup(29, 1)	29	29	5	2479	449	529
SmallGroup(30, 3)	30	9	5	2406	476	523
SmallGroup(31, 1)	31	31	2	696	109	130
SmallGroup(32, 30)	32	14	14	20454	3216	3916
SmallGroup(33, 1)	33	33	6	4405	697	834
SmallGroup(34, 1)	34	10	4	2237	326	405
SmallGroup(35, 1)	35	35	6	5346	801	882
SmallGroup(36, 13)	36	12	12	15131	2590	3225
SmallGroup(37, 1)	37	37	5	3931	547	676
SmallGroup(38, 1)	38	11	3	1796	238	288
SmallGroup(39, 2)	39	39	26	88854	12859	18046
SmallGroup(40, 3)	40	10	8	13434	1427	1777
SmallGroup(41, 1)	41	41	2	1935	130	169
SmallGroup(42, 4)	42	15	6	7883	866	1067
SmallGroup(43, 1)	43	43	3	2957	250	342
SmallGroup(44, 2)	44	44	12	29091	3498	4280
SmallGroup(45, 2)	45	45	14	44118	4699	6618
SmallGroup(46, 2)	46	46	4	4229	499	595
SmallGroup(47, 1)	47	47	3	2177	292	386
SmallGroup(48, 34)	48	18	16	62050	6092	8061
SmallGroup(49, 2)	49	49	17	72135	6807	10310
SmallGroup(50, 2)	50	50	6	10953	1046	1339

Table A.37: Idempotents, Eberly and Giesbrecht algorithm (small field), run 2.

Group	dim A	dim $Z(A)$	dim A^{ψ_q}	Group	Mult. Table	Matrix
SmallGroup(2, 1)	2	2	2	89	69	58
SmallGroup(3, 1)	3	3	2	86	68	61
SmallGroup(4, 1)	4	4	4	260	199	262
SmallGroup(5, 1)	5	5	2	95	73	67
SmallGroup(6, 1)	6	3	3	169	127	188
SmallGroup(7, 1)	7	7	3	189	130	124
SmallGroup(8, 5)	8	8	8	1119	753	720
SmallGroup(9, 2)	9	9	5	495	353	308
SmallGroup(10, 1)	10	4	3	223	130	130
SmallGroup(11, 1)	11	11	3	247	137	135
SmallGroup(12, 1)	12	6	6	868	445	459
SmallGroup(13, 1)	13	13	13	4058	1893	2098
SmallGroup(14, 1)	14	5	3	312	142	146
SmallGroup(15, 1)	15	15	5	806	345	355
SmallGroup(16, 13)	16	10	10	3029	1255	1326
SmallGroup(17, 1)	17	17	3	426	150	255
SmallGroup(18, 2)	18	18	10	3860	1354	1414
SmallGroup(19, 1)	19	19	2	255	84	88
SmallGroup(20, 5)	20	20	8	2856	918	987
SmallGroup(21, 1)	21	5	4	874	259	278
SmallGroup(22, 2)	22	22	6	1958	615	599
SmallGroup(23, 1)	23	23	2	362	89	100
SmallGroup(24, 12)	24	5	5	1641	402	436
SmallGroup(25, 1)	25	25	3	725	166	196
SmallGroup(26, 2)	26	26	26	39594	9274	12088
SmallGroup(27, 5)	27	27	14	12518	3123	3422
SmallGroup(28, 2)	28	28	12	13587	2388	2677
SmallGroup(29, 1)	29	29	5	2471	448	530
SmallGroup(30, 3)	30	9	5	2406	476	530
SmallGroup(31, 1)	31	31	2	696	109	130
SmallGroup(32, 30)	32	14	14	20451	3214	3899
SmallGroup(33, 1)	33	33	6	4411	696	834
SmallGroup(34, 1)	34	10	4	2236	341	405
SmallGroup(35, 1)	35	35	6	5345	801	866
SmallGroup(36, 13)	36	12	12	15144	2589	3229
SmallGroup(37, 1)	37	37	5	3921	546	684
SmallGroup(38, 1)	38	11	3	1796	238	289
SmallGroup(39, 2)	39	39	26	88871	12844	18031
SmallGroup(40, 3)	40	10	8	13428	1410	1776
SmallGroup(41, 1)	41	41	2	1934	130	170
SmallGroup(42, 4)	42	15	6	7862	866	1066
SmallGroup(43, 1)	43	43	3	2957	258	343
SmallGroup(44, 2)	44	44	12	29091	3496	4278
SmallGroup(45, 2)	45	45	14	44100	4702	6601
SmallGroup(46, 2)	46	46	4	4222	500	595
SmallGroup(47, 1)	47	47	3	2170	284	386
SmallGroup(48, 34)	48	18	16	62058	6084	8041
SmallGroup(49, 2)	49	49	17	72142	6788	10301
SmallGroup(50, 2)	50	50	6	10952	1038	1339

Table A.38: Idempotents, Eberly and Giesbrecht algorithm (small field), run 3.

Group	$\dim A$	$\dim Z(A)$	$\dim A^{\psi_q}$	Group	Mult. Table	Matrix
SmallGroup(2, 1)	2	2	2	91	69	58
SmallGroup(3, 1)	3	3	2	87	68	61
SmallGroup(4, 1)	4	4	4	259	199	260
SmallGroup(5, 1)	5	5	2	96	72	67
SmallGroup(6, 1)	6	3	3	177	127	187
SmallGroup(7, 1)	7	7	3	188	121	117
SmallGroup(8, 5)	8	8	8	1120	753	721
SmallGroup(9, 2)	9	9	5	510	354	308
SmallGroup(10, 1)	10	4	3	216	130	121
SmallGroup(11, 1)	11	11	3	247	135	135
SmallGroup(12, 1)	12	6	6	868	453	459
SmallGroup(13, 1)	13	13	13	4040	1935	2103
SmallGroup(14, 1)	14	5	3	311	142	146
SmallGroup(15, 1)	15	15	5	822	338	355
SmallGroup(16, 13)	16	10	10	3030	1257	1328
SmallGroup(17, 1)	17	17	3	427	150	255
SmallGroup(18, 2)	18	18	10	3861	1348	1414
SmallGroup(19, 1)	19	19	2	256	84	88
SmallGroup(20, 5)	20	20	8	2856	919	1001
SmallGroup(21, 1)	21	5	4	874	259	277
SmallGroup(22, 2)	22	22	6	1941	614	599
SmallGroup(23, 1)	23	23	2	362	89	101
SmallGroup(24, 12)	24	5	5	1665	402	445
SmallGroup(25, 1)	25	25	3	725	166	197
SmallGroup(26, 2)	26	26	26	39624	9276	12062
SmallGroup(27, 5)	27	27	14	12519	3133	3423
SmallGroup(28, 2)	28	28	12	13596	2384	2678
SmallGroup(29, 1)	29	29	5	2486	449	529
SmallGroup(30, 3)	30	9	5	2405	477	531
SmallGroup(31, 1)	31	31	2	695	109	131
SmallGroup(32, 30)	32	14	14	20444	3217	3901
SmallGroup(33, 1)	33	33	6	4412	697	834
SmallGroup(34, 1)	34	10	4	2244	342	405
SmallGroup(35, 1)	35	35	6	5351	801	874
SmallGroup(36, 13)	36	12	12	15158	2583	3242
SmallGroup(37, 1)	37	37	5	3939	546	684
SmallGroup(38, 1)	38	11	3	1782	237	280
SmallGroup(39, 2)	39	39	26	88906	12876	18059
SmallGroup(40, 3)	40	10	8	13414	1427	1761
SmallGroup(41, 1)	41	41	2	1934	130	170
SmallGroup(42, 4)	42	15	6	7879	866	1067
SmallGroup(43, 1)	43	43	3	2943	258	343
SmallGroup(44, 2)	44	44	12	29081	3489	4294
SmallGroup(45, 2)	45	45	14	44137	4707	6634
SmallGroup(46, 2)	46	46	4	4221	499	587
SmallGroup(47, 1)	47	47	3	2177	291	386
SmallGroup(48, 34)	48	18	16	62026	6064	8068
SmallGroup(49, 2)	49	49	17	72137	6799	10296
SmallGroup(50, 2)	50	50	6	10950	1046	1324

Table A.39: Idempotents, Eberly and Giesbrecht algorithm (small field), run 4.

Group	$\dim A$	$\dim Z(A)$	$\dim A^{\psi_q}$	Group	Mult. Table	Matrix
SmallGroup(2, 1)	2	2	2	90	70	60
SmallGroup(3, 1)	3	3	2	87	69	61
SmallGroup(4, 1)	4	4	4	260	198	262
SmallGroup(5, 1)	5	5	2	95	73	67
SmallGroup(6, 1)	6	3	3	178	128	188
SmallGroup(7, 1)	7	7	3	189	129	123
SmallGroup(8, 5)	8	8	8	1121	754	721
SmallGroup(9, 2)	9	9	5	510	353	309
SmallGroup(10, 1)	10	4	3	223	129	130
SmallGroup(11, 1)	11	11	3	247	136	135
SmallGroup(12, 1)	12	6	6	870	453	460
SmallGroup(13, 1)	13	13	13	4062	1930	2112
SmallGroup(14, 1)	14	5	3	311	143	139
SmallGroup(15, 1)	15	15	5	815	346	355
SmallGroup(16, 13)	16	10	10	3030	1257	1328
SmallGroup(17, 1)	17	17	3	426	150	256
SmallGroup(18, 2)	18	18	10	3867	1358	1409
SmallGroup(19, 1)	19	19	2	257	84	89
SmallGroup(20, 5)	20	20	8	2848	921	995
SmallGroup(21, 1)	21	5	4	875	259	278
SmallGroup(22, 2)	22	22	6	1958	615	600
SmallGroup(23, 1)	23	23	2	362	90	101
SmallGroup(24, 12)	24	5	5	1650	403	437
SmallGroup(25, 1)	25	25	3	725	167	196
SmallGroup(26, 2)	26	26	26	39667	9311	12085
SmallGroup(27, 5)	27	27	14	12539	3136	3409
SmallGroup(28, 2)	28	28	12	13589	2394	2680
SmallGroup(29, 1)	29	29	5	2486	449	529
SmallGroup(30, 3)	30	9	5	2398	478	523
SmallGroup(31, 1)	31	31	2	697	109	130
SmallGroup(32, 30)	32	14	14	20454	3219	3920
SmallGroup(33, 1)	33	33	6	4413	705	835
SmallGroup(34, 1)	34	10	4	2245	342	397
SmallGroup(35, 1)	35	35	6	5349	801	882
SmallGroup(36, 13)	36	12	12	15141	2593	3233
SmallGroup(37, 1)	37	37	5	3945	546	684
SmallGroup(38, 1)	38	11	3	1796	238	289
SmallGroup(39, 2)	39	39	26	88848	12881	18046
SmallGroup(40, 3)	40	10	8	13419	1420	1762
SmallGroup(41, 1)	41	41	2	1925	130	169
SmallGroup(42, 4)	42	15	6	7864	869	1067
SmallGroup(43, 1)	43	43	3	2958	258	343
SmallGroup(44, 2)	44	44	12	29107	3472	4289
SmallGroup(45, 2)	45	45	14	44117	4707	6628
SmallGroup(46, 2)	46	46	4	4236	500	596
SmallGroup(47, 1)	47	47	3	2171	292	386
SmallGroup(48, 34)	48	18	16	62087	6080	8065
SmallGroup(49, 2)	49	49	17	72138	6814	10315
SmallGroup(50, 2)	50	50	6	10955	1054	1340

Table A.40: Idempotents, Eberly and Giesbrecht algorithm (small field), run 5.

Group	$\dim A$	$\dim Z(A)$	$\dim A^{\psi_q}$	Group	Mult. Table	Matrix
SmallGroup(2, 1)	2	2	2	90	70	59
SmallGroup(3, 1)	3	3	2	86	67	61
SmallGroup(4, 1)	4	4	4	260	199	254
SmallGroup(5, 1)	5	5	2	96	72	66
SmallGroup(6, 1)	6	3	3	176	127	186
SmallGroup(7, 1)	7	7	3	189	129	124
SmallGroup(8, 5)	8	8	8	1114	738	720
SmallGroup(9, 2)	9	9	5	509	353	308
SmallGroup(10, 1)	10	4	3	223	130	130
SmallGroup(11, 1)	11	11	3	248	137	135
SmallGroup(12, 1)	12	6	6	869	454	459
SmallGroup(13, 1)	13	13	13	4047	1939	2115
SmallGroup(14, 1)	14	5	3	310	142	146
SmallGroup(15, 1)	15	15	5	822	345	359
SmallGroup(16, 13)	16	10	10	3005	1271	1317
SmallGroup(17, 1)	17	17	3	425	149	254
SmallGroup(18, 2)	18	18	10	3859	1353	1411
SmallGroup(19, 1)	19	19	2	257	84	88
SmallGroup(20, 5)	20	20	8	2860	901	1000
SmallGroup(21, 1)	21	5	4	874	259	278
SmallGroup(22, 2)	22	22	6	1950	614	598
SmallGroup(23, 1)	23	23	2	362	89	100
SmallGroup(24, 12)	24	5	5	1665	401	444
SmallGroup(25, 1)	25	25	3	725	166	197
SmallGroup(26, 2)	26	26	26	39701	9274	12063
SmallGroup(27, 5)	27	27	14	12519	3130	3419
SmallGroup(28, 2)	28	28	12	13592	2379	2666
SmallGroup(29, 1)	29	29	5	2485	448	528
SmallGroup(30, 3)	30	9	5	2405	475	530
SmallGroup(31, 1)	31	31	2	696	108	131
SmallGroup(32, 30)	32	14	14	20462	3187	3903
SmallGroup(33, 1)	33	33	6	4418	703	834
SmallGroup(34, 1)	34	10	4	2244	341	404
SmallGroup(35, 1)	35	35	6	5339	801	880
SmallGroup(36, 13)	36	12	12	15165	2571	3228
SmallGroup(37, 1)	37	37	5	3945	538	683
SmallGroup(38, 1)	38	11	3	1787	238	288
SmallGroup(39, 2)	39	39	26	88852	12859	18038
SmallGroup(40, 3)	40	10	8	13420	1425	1775
SmallGroup(41, 1)	41	41	2	1935	130	170
SmallGroup(42, 4)	42	15	6	7915	891	1075
SmallGroup(43, 1)	43	43	3	2952	247	343
SmallGroup(44, 2)	44	44	12	29115	3492	4286
SmallGroup(45, 2)	45	45	14	44232	4750	6643
SmallGroup(46, 2)	46	46	4	4257	496	597
SmallGroup(47, 1)	47	47	3	2172	293	380
SmallGroup(48, 34)	48	18	16	62063	6075	8030
SmallGroup(49, 2)	49	49	17	72128	6809	10327
SmallGroup(50, 2)	50	50	6	10942	1038	1338

A.4.2 Large Field

The field used in each of these experiments is $\mathbb{F}_{65521}$.

Davis' Algorithm

This is the data for DavisIdempotents.

Table A.41: Idempotents, Davis algorithm (large field), run 1.

Group	dim A	dim $Z(A)$	dim A^{ψ_q}	Group	Mult. Table	Matrix
SmallGroup(2, 1)	2	2	2	100	56	44
SmallGroup(3, 1)	3	3	3	145	100	82
SmallGroup(4, 1)	4	4	4	206	146	124
SmallGroup(5, 1)	5	5	5	257	190	159
SmallGroup(6, 1)	6	3	3	189	106	89
SmallGroup(7, 1)	7	7	7	383	285	254
SmallGroup(8, 5)	8	8	8	530	358	317
SmallGroup(9, 2)	9	9	9	561	398	358
SmallGroup(10, 1)	10	4	4	415	184	169
SmallGroup(11, 1)	11	11	3	508	140	144
SmallGroup(12, 1)	12	6	6	477	277	263
SmallGroup(13, 1)	13	13	13	841	608	570
SmallGroup(14, 1)	14	5	5	672	264	274
SmallGroup(15, 1)	15	15	15	1131	738	731
SmallGroup(16, 13)	16	10	10	779	495	518
SmallGroup(17, 1)	17	17	2	1156	143	191
SmallGroup(18, 2)	18	18	18	1579	944	990
SmallGroup(19, 1)	19	19	3	1346	220	362
SmallGroup(20, 5)	20	20	20	1802	1092	1198
SmallGroup(21, 1)	21	5	5	907	318	399
SmallGroup(22, 2)	22	22	6	1286	391	517
SmallGroup(23, 1)	23	23	2	2318	224	430
SmallGroup(24, 12)	24	5	5	1196	349	482
SmallGroup(25, 1)	25	25	9	1130	537	673
SmallGroup(26, 2)	26	26	26	2695	1558	1929
SmallGroup(27, 5)	27	27	27	3065	1662	2185
SmallGroup(28, 2)	28	28	28	3427	1746	2311
SmallGroup(29, 1)	29	29	2	4374	325	715
SmallGroup(30, 3)	30	9	9	2913	772	1569
SmallGroup(31, 1)	31	31	3	4641	444	1290
SmallGroup(32, 30)	32	14	14	2355	1182	1937
SmallGroup(33, 1)	33	33	9	2312	745	1243
SmallGroup(34, 1)	34	10	3	1847	443	572
SmallGroup(35, 1)	35	35	35	5220	2489	4033
SmallGroup(36, 13)	36	12	12	2390	1134	2196
SmallGroup(37, 1)	37	37	10	9737	1147	4004
SmallGroup(38, 1)	38	11	3	3227	551	838
SmallGroup(39, 2)	39	39	39	6472	2997	5327
SmallGroup(40, 3)	40	10	10	2663	981	1600
SmallGroup(41, 1)	41	41	6	18132	1028	4101
SmallGroup(42, 4)	42	15	15	2549	1292	2469
SmallGroup(43, 1)	43	43	4	9661	869	3480
SmallGroup(44, 2)	44	44	12	5151	1281	2523
SmallGroup(45, 2)	45	45	45	9638	3959	8866
SmallGroup(46, 2)	46	46	4	4465	958	1836
SmallGroup(47, 1)	47	47	3	9556	956	3882
SmallGroup(48, 34)	48	18	18	5282	2342	5961
SmallGroup(49, 2)	49	49	49	11769	5125	11217
SmallGroup(50, 2)	50	50	18	4570	1835	3656

Table A.42: Idempotents, Davis algorithm (large field), run 2.

Group	dim A	dim $Z(A)$	dim A^{ψ_q}	Group	Mult. Table	Matrix
SmallGroup(2, 1)	2	2	2	104	57	43
SmallGroup(3, 1)	3	3	3	146	101	86
SmallGroup(4, 1)	4	4	4	197	144	124
SmallGroup(5, 1)	5	5	5	256	191	166
SmallGroup(6, 1)	6	3	3	189	105	89
SmallGroup(7, 1)	7	7	7	384	285	254
SmallGroup(8, 5)	8	8	8	533	357	317
SmallGroup(9, 2)	9	9	9	568	400	365
SmallGroup(10, 1)	10	4	4	424	186	169
SmallGroup(11, 1)	11	11	3	509	139	144
SmallGroup(12, 1)	12	6	6	478	278	263
SmallGroup(13, 1)	13	13	13	840	609	569
SmallGroup(14, 1)	14	5	5	680	265	277
SmallGroup(15, 1)	15	15	15	1149	739	731
SmallGroup(16, 13)	16	10	10	779	496	523
SmallGroup(17, 1)	17	17	2	1167	144	192
SmallGroup(18, 2)	18	18	18	1571	943	990
SmallGroup(19, 1)	19	19	3	1351	219	362
SmallGroup(20, 5)	20	20	20	1794	1093	1222
SmallGroup(21, 1)	21	5	5	915	324	399
SmallGroup(22, 2)	22	22	6	1286	394	523
SmallGroup(23, 1)	23	23	2	2345	235	423
SmallGroup(24, 12)	24	5	5	1204	346	474
SmallGroup(25, 1)	25	25	9	1128	544	680
SmallGroup(26, 2)	26	26	26	2699	1583	1933
SmallGroup(27, 5)	27	27	27	3055	1669	2176
SmallGroup(28, 2)	28	28	28	3425	1777	2309
SmallGroup(29, 1)	29	29	2	4374	325	715
SmallGroup(30, 3)	30	9	9	2926	788	1568
SmallGroup(31, 1)	31	31	3	4648	443	1299
SmallGroup(32, 30)	32	14	14	2355	1184	1937
SmallGroup(33, 1)	33	33	9	2318	730	1239
SmallGroup(34, 1)	34	10	3	1828	438	572
SmallGroup(35, 1)	35	35	35	5211	2483	4019
SmallGroup(36, 13)	36	12	12	2390	1134	2200
SmallGroup(37, 1)	37	37	10	9781	1147	4012
SmallGroup(38, 1)	38	11	3	3227	543	838
SmallGroup(39, 2)	39	39	39	6487	2997	5297
SmallGroup(40, 3)	40	10	10	2670	969	1593
SmallGroup(41, 1)	41	41	6	18090	1020	4109
SmallGroup(42, 4)	42	15	15	2547	1292	2471
SmallGroup(43, 1)	43	43	4	9593	885	3480
SmallGroup(44, 2)	44	44	12	5167	1263	2515
SmallGroup(45, 2)	45	45	45	9583	3917	8897
SmallGroup(46, 2)	46	46	4	4422	949	1837
SmallGroup(47, 1)	47	47	3	9534	944	3891
SmallGroup(48, 34)	48	18	18	5243	2327	5970
SmallGroup(49, 2)	49	49	49	11745	5063	11192
SmallGroup(50, 2)	50	50	18	4559	1833	3637

Table A.43: Idempotents, Davis algorithm (large field), run 3.

Group	dim A	dim $Z(A)$	dim A^{ψ_q}	Group	Mult. Table	Matrix
SmallGroup(2, 1)	2	2	2	101	56	44
SmallGroup(3, 1)	3	3	3	144	101	82
SmallGroup(4, 1)	4	4	4	205	146	124
SmallGroup(5, 1)	5	5	5	258	190	166
SmallGroup(6, 1)	6	3	3	189	106	90
SmallGroup(7, 1)	7	7	7	384	286	253
SmallGroup(8, 5)	8	8	8	531	357	317
SmallGroup(9, 2)	9	9	9	570	399	351
SmallGroup(10, 1)	10	4	4	406	184	168
SmallGroup(11, 1)	11	11	3	508	140	144
SmallGroup(12, 1)	12	6	6	478	277	263
SmallGroup(13, 1)	13	13	13	841	608	569
SmallGroup(14, 1)	14	5	5	680	264	274
SmallGroup(15, 1)	15	15	15	1129	739	731
SmallGroup(16, 13)	16	10	10	779	495	520
SmallGroup(17, 1)	17	17	2	1157	144	192
SmallGroup(18, 2)	18	18	18	1581	943	991
SmallGroup(19, 1)	19	19	3	1346	219	362
SmallGroup(20, 5)	20	20	20	1795	1093	1208
SmallGroup(21, 1)	21	5	5	907	318	399
SmallGroup(22, 2)	22	22	6	1286	390	510
SmallGroup(23, 1)	23	23	2	2325	232	423
SmallGroup(24, 12)	24	5	5	1212	347	482
SmallGroup(25, 1)	25	25	9	1119	544	672
SmallGroup(26, 2)	26	26	26	2687	1559	1921
SmallGroup(27, 5)	27	27	27	3058	1669	2184
SmallGroup(28, 2)	28	28	28	3421	1749	2312
SmallGroup(29, 1)	29	29	2	4369	325	715
SmallGroup(30, 3)	30	9	9	2935	788	1568
SmallGroup(31, 1)	31	31	3	4642	443	1298
SmallGroup(32, 30)	32	14	14	2358	1183	1915
SmallGroup(33, 1)	33	33	9	2321	738	1239
SmallGroup(34, 1)	34	10	3	1827	436	572
SmallGroup(35, 1)	35	35	35	5205	2499	4036
SmallGroup(36, 13)	36	12	12	2409	1134	2205
SmallGroup(37, 1)	37	37	10	9766	1147	4003
SmallGroup(38, 1)	38	11	3	3229	560	838
SmallGroup(39, 2)	39	39	39	6508	3005	5310
SmallGroup(40, 3)	40	10	10	2678	970	1600
SmallGroup(41, 1)	41	41	6	18128	1028	4100
SmallGroup(42, 4)	42	15	15	2550	1292	2462
SmallGroup(43, 1)	43	43	4	9586	886	3488
SmallGroup(44, 2)	44	44	12	5144	1265	2508
SmallGroup(45, 2)	45	45	45	9579	3930	8900
SmallGroup(46, 2)	46	46	4	4425	974	1829
SmallGroup(47, 1)	47	47	3	9532	945	3898
SmallGroup(48, 34)	48	18	18	5232	2319	5967
SmallGroup(49, 2)	49	49	49	11760	5065	11215
SmallGroup(50, 2)	50	50	18	4569	1825	3660

Table A.44: Idempotents, Davis algorithm (large field), run 4.

Group	dim A	dim $Z(A)$	dim A^{ψ_q}	Group	Mult. Table	Matrix
SmallGroup(2, 1)	2	2	2	101	57	44
SmallGroup(3, 1)	3	3	3	144	100	82
SmallGroup(4, 1)	4	4	4	204	145	124
SmallGroup(5, 1)	5	5	5	259	190	166
SmallGroup(6, 1)	6	3	3	189	106	89
SmallGroup(7, 1)	7	7	7	384	285	254
SmallGroup(8, 5)	8	8	8	531	359	317
SmallGroup(9, 2)	9	9	9	571	399	359
SmallGroup(10, 1)	10	4	4	416	184	168
SmallGroup(11, 1)	11	11	3	508	140	144
SmallGroup(12, 1)	12	6	6	478	277	263
SmallGroup(13, 1)	13	13	13	833	594	569
SmallGroup(14, 1)	14	5	5	672	264	274
SmallGroup(15, 1)	15	15	15	1113	739	731
SmallGroup(16, 13)	16	10	10	779	488	519
SmallGroup(17, 1)	17	17	2	1156	144	191
SmallGroup(18, 2)	18	18	18	1580	943	989
SmallGroup(19, 1)	19	19	3	1353	220	362
SmallGroup(20, 5)	20	20	20	1795	1093	1198
SmallGroup(21, 1)	21	5	5	898	319	399
SmallGroup(22, 2)	22	22	6	1287	390	518
SmallGroup(23, 1)	23	23	2	2316	232	423
SmallGroup(24, 12)	24	5	5	1212	347	482
SmallGroup(25, 1)	25	25	9	1127	544	672
SmallGroup(26, 2)	26	26	26	2687	1557	1922
SmallGroup(27, 5)	27	27	27	3042	1669	2177
SmallGroup(28, 2)	28	28	28	3421	1746	2310
SmallGroup(29, 1)	29	29	2	4373	325	715
SmallGroup(30, 3)	30	9	9	2935	772	1570
SmallGroup(31, 1)	31	31	3	4650	443	1299
SmallGroup(32, 30)	32	14	14	2356	1184	1937
SmallGroup(33, 1)	33	33	9	2321	730	1238
SmallGroup(34, 1)	34	10	3	1827	421	572
SmallGroup(35, 1)	35	35	35	5222	2480	4037
SmallGroup(36, 13)	36	12	12	2401	1134	2199
SmallGroup(37, 1)	37	37	10	9778	1146	4012
SmallGroup(38, 1)	38	11	3	3228	559	830
SmallGroup(39, 2)	39	39	39	6499	2981	5314
SmallGroup(40, 3)	40	10	10	2665	962	1594
SmallGroup(41, 1)	41	41	6	18139	1019	4102
SmallGroup(42, 4)	42	15	15	2550	1293	2477
SmallGroup(43, 1)	43	43	4	9579	885	3487
SmallGroup(44, 2)	44	44	12	5169	1265	2514
SmallGroup(45, 2)	45	45	45	9576	3922	8880
SmallGroup(46, 2)	46	46	4	4423	965	1837
SmallGroup(47, 1)	47	47	3	9527	945	3882
SmallGroup(48, 34)	48	18	18	5232	2326	5998
SmallGroup(49, 2)	49	49	49	11845	5069	11252
SmallGroup(50, 2)	50	50	18	4584	1831	3649

Table A.45: Idempotents, Davis algorithm (large field), run 5.

Group	dim A	dim $Z(A)$	dim A^{ψ_q}	Group	Mult. Table	Matrix
SmallGroup(2, 1)	2	2	2	101	57	44
SmallGroup(3, 1)	3	3	3	144	101	82
SmallGroup(4, 1)	4	4	4	205	146	125
SmallGroup(5, 1)	5	5	5	257	191	166
SmallGroup(6, 1)	6	3	3	189	107	89
SmallGroup(7, 1)	7	7	7	383	285	254
SmallGroup(8, 5)	8	8	8	531	359	317
SmallGroup(9, 2)	9	9	9	569	391	358
SmallGroup(10, 1)	10	4	4	415	184	167
SmallGroup(11, 1)	11	11	3	509	139	143
SmallGroup(12, 1)	12	6	6	478	277	263
SmallGroup(13, 1)	13	13	13	841	607	570
SmallGroup(14, 1)	14	5	5	681	264	274
SmallGroup(15, 1)	15	15	15	1130	739	730
SmallGroup(16, 13)	16	10	10	778	495	519
SmallGroup(17, 1)	17	17	2	1149	144	192
SmallGroup(18, 2)	18	18	18	1557	936	982
SmallGroup(19, 1)	19	19	3	1352	220	363
SmallGroup(20, 5)	20	20	20	1803	1084	1207
SmallGroup(21, 1)	21	5	5	907	319	399
SmallGroup(22, 2)	22	22	6	1279	390	518
SmallGroup(23, 1)	23	23	2	2318	232	430
SmallGroup(24, 12)	24	5	5	1212	346	481
SmallGroup(25, 1)	25	25	9	1127	545	679
SmallGroup(26, 2)	26	26	26	2687	1549	1920
SmallGroup(27, 5)	27	27	27	3065	1669	2185
SmallGroup(28, 2)	28	28	28	3429	1747	2302
SmallGroup(29, 1)	29	29	2	4386	325	715
SmallGroup(30, 3)	30	9	9	2927	787	1561
SmallGroup(31, 1)	31	31	3	4611	443	1291
SmallGroup(32, 30)	32	14	14	2357	1183	1930
SmallGroup(33, 1)	33	33	9	2305	737	1224
SmallGroup(34, 1)	34	10	3	1828	438	572
SmallGroup(35, 1)	35	35	35	5215	2491	4029
SmallGroup(36, 13)	36	12	12	2408	1134	2199
SmallGroup(37, 1)	37	37	10	9751	1148	4012
SmallGroup(38, 1)	38	11	3	3222	560	838
SmallGroup(39, 2)	39	39	39	6508	3006	5296
SmallGroup(40, 3)	40	10	10	2680	970	1602
SmallGroup(41, 1)	41	41	6	18124	1030	4100
SmallGroup(42, 4)	42	15	15	2541	1293	2477
SmallGroup(43, 1)	43	43	4	9587	878	3479
SmallGroup(44, 2)	44	44	12	5161	1264	2523
SmallGroup(45, 2)	45	45	45	9571	3917	8896
SmallGroup(46, 2)	46	46	4	4414	973	1837
SmallGroup(47, 1)	47	47	3	9542	947	3890
SmallGroup(48, 34)	48	18	18	5246	2333	5970
SmallGroup(49, 2)	49	49	49	11770	5037	11206
SmallGroup(50, 2)	50	50	18	4561	1833	3660

Eberly and Giesbrecht Algorithm

This is the data for EberlyGiesbrechtLarge.

Table A.46: Idempotents, Eberly and Giesbrecht algorithm (large field), run 1.

Group	dim A	dim $Z(A)$	dim A^{ψ_q}	Group	Mult. Table	Matrix
SmallGroup(2, 1)	2	2	2	95	66	67
SmallGroup(3, 1)	3	3	3	143	129	117
SmallGroup(4, 1)	4	4	4	232	238	254
SmallGroup(5, 1)	5	5	5	377	283	280
SmallGroup(6, 1)	6	3	3	235	150	122
SmallGroup(7, 1)	7	7	7	689	581	563
SmallGroup(8, 5)	8	8	8	876	777	756
SmallGroup(9, 2)	9	9	9	1186	958	915
SmallGroup(10, 1)	10	4	4	256	191	205
SmallGroup(11, 1)	11	11	3	175	119	138
SmallGroup(12, 1)	12	6	6	584	434	485
SmallGroup(13, 1)	13	13	13	2750	2006	2126
SmallGroup(14, 1)	14	5	5	462	318	340
SmallGroup(15, 1)	15	15	15	4033	2706	2851
SmallGroup(16, 13)	16	10	10	1844	1218	1304
SmallGroup(17, 1)	17	17	2	73	58	85
SmallGroup(18, 2)	18	18	18	6959	4060	4453
SmallGroup(19, 1)	19	19	3	196	188	187
SmallGroup(20, 5)	20	20	20	9354	5182	5853
SmallGroup(21, 1)	21	5	5	627	327	447
SmallGroup(22, 2)	22	22	6	936	470	702
SmallGroup(23, 1)	23	23	2	83	73	138
SmallGroup(24, 12)	24	5	5	722	330	469
SmallGroup(25, 1)	25	25	9	2431	1008	1529
SmallGroup(26, 2)	26	26	26	22160	9509	13098
SmallGroup(27, 5)	27	27	27	27565	10518	14768
SmallGroup(28, 2)	28	28	28	35333	12355	16507
SmallGroup(29, 1)	29	29	2	151	69	114
SmallGroup(30, 3)	30	9	9	3251	1116	1822
SmallGroup(31, 1)	31	31	3	302	153	249
SmallGroup(32, 30)	32	14	14	9039	2715	4585
SmallGroup(33, 1)	33	33	9	3514	1127	2072
SmallGroup(34, 1)	34	10	3	380	151	273
SmallGroup(35, 1)	35	35	35	71482	21280	34977
SmallGroup(36, 13)	36	12	12	8188	2131	4134
SmallGroup(37, 1)	37	37	10	4998	1496	2970
SmallGroup(38, 1)	38	11	3	471	168	312
SmallGroup(39, 2)	39	39	39	105120	29167	51917
SmallGroup(40, 3)	40	10	10	8161	1551	3344
SmallGroup(41, 1)	41	41	6	2319	595	1337
SmallGroup(42, 4)	42	15	15	18250	3644	8239
SmallGroup(43, 1)	43	43	4	1045	268	710
SmallGroup(44, 2)	44	44	12	10542	2449	5858
SmallGroup(45, 2)	45	45	45	198373	44416	91665
SmallGroup(46, 2)	46	46	4	1047	302	749
SmallGroup(47, 1)	47	47	3	589	173	512
SmallGroup(48, 34)	48	18	18	35177	5949	16373
SmallGroup(49, 2)	49	49	49	276909	51926	129506
SmallGroup(50, 2)	50	50	18	37419	6100	17464

Table A.47: Idempotents, Eberly and Giesbrecht algorithm (large field), run 2.

Group	dim A	dim $Z(A)$	dim A^{ψ_q}	Group	Mult. Table	Matrix
SmallGroup(2, 1)	2	2	2	95	66	66
SmallGroup(3, 1)	3	3	3	142	131	117
SmallGroup(4, 1)	4	4	4	231	236	253
SmallGroup(5, 1)	5	5	5	377	277	279
SmallGroup(6, 1)	6	3	3	233	149	121
SmallGroup(7, 1)	7	7	7	687	579	562
SmallGroup(8, 5)	8	8	8	874	774	754
SmallGroup(9, 2)	9	9	9	1184	963	913
SmallGroup(10, 1)	10	4	4	255	182	204
SmallGroup(11, 1)	11	11	3	175	119	138
SmallGroup(12, 1)	12	6	6	582	432	485
SmallGroup(13, 1)	13	13	13	2751	1999	2120
SmallGroup(14, 1)	14	5	5	470	316	340
SmallGroup(15, 1)	15	15	15	3999	2688	2835
SmallGroup(16, 13)	16	10	10	1850	1218	1325
SmallGroup(17, 1)	17	17	2	73	59	86
SmallGroup(18, 2)	18	18	18	6958	4060	4446
SmallGroup(19, 1)	19	19	3	195	189	187
SmallGroup(20, 5)	20	20	20	9354	5190	5859
SmallGroup(21, 1)	21	5	5	627	327	447
SmallGroup(22, 2)	22	22	6	935	455	691
SmallGroup(23, 1)	23	23	2	83	73	139
SmallGroup(24, 12)	24	5	5	723	329	470
SmallGroup(25, 1)	25	25	9	2416	1008	1537
SmallGroup(26, 2)	26	26	26	22190	9496	13078
SmallGroup(27, 5)	27	27	27	27562	10514	14793
SmallGroup(28, 2)	28	28	28	35410	12367	16499
SmallGroup(29, 1)	29	29	2	151	69	114
SmallGroup(30, 3)	30	9	9	3249	1116	1821
SmallGroup(31, 1)	31	31	3	301	154	249
SmallGroup(32, 30)	32	14	14	9054	2715	4593
SmallGroup(33, 1)	33	33	9	3520	1134	2071
SmallGroup(34, 1)	34	10	3	372	151	274
SmallGroup(35, 1)	35	35	35	71453	21296	34991
SmallGroup(36, 13)	36	12	12	8202	2125	4127
SmallGroup(37, 1)	37	37	10	4992	1481	2995
SmallGroup(38, 1)	38	11	3	472	168	312
SmallGroup(39, 2)	39	39	39	105113	29147	51900
SmallGroup(40, 3)	40	10	10	8137	1550	3328
SmallGroup(41, 1)	41	41	6	2334	595	1337
SmallGroup(42, 4)	42	15	15	18235	3657	8225
SmallGroup(43, 1)	43	43	4	1043	268	718
SmallGroup(44, 2)	44	44	12	10555	2462	5852
SmallGroup(45, 2)	45	45	45	198360	44417	91667
SmallGroup(46, 2)	46	46	4	1055	302	749
SmallGroup(47, 1)	47	47	3	582	175	512
SmallGroup(48, 34)	48	18	18	35128	5937	16367
SmallGroup(49, 2)	49	49	49	276979	51982	129473
SmallGroup(50, 2)	50	50	18	37456	6068	17466

Table A.48: Idempotents, Eberly and Giesbrecht algorithm (large field), run 3.

Group	dim A	dim $Z(A)$	dim A^{ψ_q}	Group	Mult. Table	Matrix
SmallGroup(2, 1)	2	2	2	94	65	67
SmallGroup(3, 1)	3	3	3	143	131	117
SmallGroup(4, 1)	4	4	4	231	238	254
SmallGroup(5, 1)	5	5	5	377	284	280
SmallGroup(6, 1)	6	3	3	226	149	122
SmallGroup(7, 1)	7	7	7	690	579	564
SmallGroup(8, 5)	8	8	8	878	778	750
SmallGroup(9, 2)	9	9	9	1188	967	909
SmallGroup(10, 1)	10	4	4	247	191	206
SmallGroup(11, 1)	11	11	3	175	119	138
SmallGroup(12, 1)	12	6	6	584	433	486
SmallGroup(13, 1)	13	13	13	2771	2005	2124
SmallGroup(14, 1)	14	5	5	470	318	340
SmallGroup(15, 1)	15	15	15	4040	2706	2859
SmallGroup(16, 13)	16	10	10	1836	1219	1328
SmallGroup(17, 1)	17	17	2	73	59	85
SmallGroup(18, 2)	18	18	18	6935	4040	4434
SmallGroup(19, 1)	19	19	3	196	181	187
SmallGroup(20, 5)	20	20	20	9349	5169	5858
SmallGroup(21, 1)	21	5	5	628	328	448
SmallGroup(22, 2)	22	22	6	935	472	699
SmallGroup(23, 1)	23	23	2	84	73	139
SmallGroup(24, 12)	24	5	5	722	329	470
SmallGroup(25, 1)	25	25	9	2439	1009	1530
SmallGroup(26, 2)	26	26	26	22186	9489	13093
SmallGroup(27, 5)	27	27	27	27586	10501	14797
SmallGroup(28, 2)	28	28	28	35442	12381	16496
SmallGroup(29, 1)	29	29	2	150	69	115
SmallGroup(30, 3)	30	9	9	3236	1119	1814
SmallGroup(31, 1)	31	31	3	301	153	249
SmallGroup(32, 30)	32	14	14	9052	2719	4597
SmallGroup(33, 1)	33	33	9	3515	1128	2072
SmallGroup(34, 1)	34	10	3	380	151	266
SmallGroup(35, 1)	35	35	35	71483	21303	35010
SmallGroup(36, 13)	36	12	12	8214	2137	4138
SmallGroup(37, 1)	37	37	10	4986	1482	2981
SmallGroup(38, 1)	38	11	3	471	169	312
SmallGroup(39, 2)	39	39	39	105171	29108	51950
SmallGroup(40, 3)	40	10	10	8152	1554	3323
SmallGroup(41, 1)	41	41	6	2329	595	1338
SmallGroup(42, 4)	42	15	15	18266	3661	8229
SmallGroup(43, 1)	43	43	4	1044	268	718
SmallGroup(44, 2)	44	44	12	10565	2457	5862
SmallGroup(45, 2)	45	45	45	198380	44459	91696
SmallGroup(46, 2)	46	46	4	1056	302	749
SmallGroup(47, 1)	47	47	3	582	174	512
SmallGroup(48, 34)	48	18	18	35150	5938	16356
SmallGroup(49, 2)	49	49	49	276991	51983	129581
SmallGroup(50, 2)	50	50	18	37437	6058	17475

Table A.49: Idempotents, Eberly and Giesbrecht algorithm (large field), run 4.

Group	dim A	dim $Z(A)$	dim A^{ψ_q}	Group	Mult. Table	Matrix
SmallGroup(2, 1)	2	2	2	95	66	66
SmallGroup(3, 1)	3	3	3	142	130	118
SmallGroup(4, 1)	4	4	4	232	238	254
SmallGroup(5, 1)	5	5	5	377	285	279
SmallGroup(6, 1)	6	3	3	233	149	122
SmallGroup(7, 1)	7	7	7	691	572	563
SmallGroup(8, 5)	8	8	8	876	779	758
SmallGroup(9, 2)	9	9	9	1189	967	900
SmallGroup(10, 1)	10	4	4	254	192	205
SmallGroup(11, 1)	11	11	3	175	120	139
SmallGroup(12, 1)	12	6	6	583	434	471
SmallGroup(13, 1)	13	13	13	2773	2007	2109
SmallGroup(14, 1)	14	5	5	469	318	341
SmallGroup(15, 1)	15	15	15	4024	2699	2858
SmallGroup(16, 13)	16	10	10	1852	1220	1328
SmallGroup(17, 1)	17	17	2	74	59	85
SmallGroup(18, 2)	18	18	18	6951	4068	4461
SmallGroup(19, 1)	19	19	3	196	189	187
SmallGroup(20, 5)	20	20	20	9355	5201	5845
SmallGroup(21, 1)	21	5	5	627	328	448
SmallGroup(22, 2)	22	22	6	936	472	700
SmallGroup(23, 1)	23	23	2	83	74	140
SmallGroup(24, 12)	24	5	5	723	329	462
SmallGroup(25, 1)	25	25	9	2440	1010	1531
SmallGroup(26, 2)	26	26	26	22209	9513	13121
SmallGroup(27, 5)	27	27	27	27628	10540	14763
SmallGroup(28, 2)	28	28	28	35483	12397	16515
SmallGroup(29, 1)	29	29	2	151	62	114
SmallGroup(30, 3)	30	9	9	3247	1120	1816
SmallGroup(31, 1)	31	31	3	303	153	249
SmallGroup(32, 30)	32	14	14	9050	2724	4591
SmallGroup(33, 1)	33	33	9	3507	1138	2074
SmallGroup(34, 1)	34	10	3	379	151	274
SmallGroup(35, 1)	35	35	35	71524	21313	34988
SmallGroup(36, 13)	36	12	12	8196	2132	4147
SmallGroup(37, 1)	37	37	10	4988	1499	2991
SmallGroup(38, 1)	38	11	3	471	169	305
SmallGroup(39, 2)	39	39	39	105244	29286	51964
SmallGroup(40, 3)	40	10	10	8147	1549	3340
SmallGroup(41, 1)	41	41	6	2337	596	1331
SmallGroup(42, 4)	42	15	15	18255	3657	8229
SmallGroup(43, 1)	43	43	4	1045	268	718
SmallGroup(44, 2)	44	44	12	10556	2465	5861
SmallGroup(45, 2)	45	45	45	198432	44491	91748
SmallGroup(46, 2)	46	46	4	1050	302	734
SmallGroup(47, 1)	47	47	3	589	174	513
SmallGroup(48, 34)	48	18	18	35161	5952	16343
SmallGroup(49, 2)	49	49	49	277059	52083	129532
SmallGroup(50, 2)	50	50	18	37466	6102	17468

Table A.50: Idempotents, Eberly and Giesbrecht algorithm (large field), run 5.

Group	dim A	dim $Z(A)$	dim A^{ψ_q}	Group	Mult. Table	Matrix
SmallGroup(2, 1)	2	2	2	94	66	67
SmallGroup(3, 1)	3	3	3	143	131	118
SmallGroup(4, 1)	4	4	4	233	229	254
SmallGroup(5, 1)	5	5	5	377	285	279
SmallGroup(6, 1)	6	3	3	234	149	123
SmallGroup(7, 1)	7	7	7	689	580	564
SmallGroup(8, 5)	8	8	8	876	776	755
SmallGroup(9, 2)	9	9	9	1186	965	905
SmallGroup(10, 1)	10	4	4	254	190	204
SmallGroup(11, 1)	11	11	3	175	119	138
SmallGroup(12, 1)	12	6	6	583	433	486
SmallGroup(13, 1)	13	13	13	2768	2002	2098
SmallGroup(14, 1)	14	5	5	471	317	340
SmallGroup(15, 1)	15	15	15	4025	2700	2853
SmallGroup(16, 13)	16	10	10	1851	1218	1325
SmallGroup(17, 1)	17	17	2	73	59	77
SmallGroup(18, 2)	18	18	18	6936	4061	4448
SmallGroup(19, 1)	19	19	3	196	181	187
SmallGroup(20, 5)	20	20	20	9345	5191	5838
SmallGroup(21, 1)	21	5	5	626	327	447
SmallGroup(22, 2)	22	22	6	934	471	699
SmallGroup(23, 1)	23	23	2	84	74	139
SmallGroup(24, 12)	24	5	5	723	322	470
SmallGroup(25, 1)	25	25	9	2437	1007	1536
SmallGroup(26, 2)	26	26	26	22170	9493	13073
SmallGroup(27, 5)	27	27	27	27564	10502	14757
SmallGroup(28, 2)	28	28	28	35397	12379	16514
SmallGroup(29, 1)	29	29	2	150	69	115
SmallGroup(30, 3)	30	9	9	3234	1117	1806
SmallGroup(31, 1)	31	31	3	301	153	249
SmallGroup(32, 30)	32	14	14	9052	2716	4584
SmallGroup(33, 1)	33	33	9	3504	1134	2066
SmallGroup(34, 1)	34	10	3	379	151	273
SmallGroup(35, 1)	35	35	35	71433	21260	35002
SmallGroup(36, 13)	36	12	12	8202	2118	4134
SmallGroup(37, 1)	37	37	10	4991	1495	2980
SmallGroup(38, 1)	38	11	3	471	168	312
SmallGroup(39, 2)	39	39	39	105135	29103	51919
SmallGroup(40, 3)	40	10	10	8150	1544	3330
SmallGroup(41, 1)	41	41	6	2335	595	1338
SmallGroup(42, 4)	42	15	15	18259	3640	8210
SmallGroup(43, 1)	43	43	4	1045	268	703
SmallGroup(44, 2)	44	44	12	10549	2434	5868
SmallGroup(45, 2)	45	45	45	198346	44426	91642
SmallGroup(46, 2)	46	46	4	1049	302	741
SmallGroup(47, 1)	47	47	3	589	173	512
SmallGroup(48, 34)	48	18	18	35144	5935	16319
SmallGroup(49, 2)	49	49	49	276971	51934	129503
SmallGroup(50, 2)	50	50	18	37515	6067	17484

A.5 Jacobson Radical

In this section, the experimental data for calculating $J(A)$ is presented. The experiments are executed on group algebras encoded as group algebras, multiplication tables, and the regular matrix representation. The field for each group algebra is $\mathbb{F}_p$ where p is the largest prime factor of the order of the group. The group is labeled with the `SmallGroup` command in GAP version 4.1 fix 7 used to construct the group. All times are in CPU milliseconds.

A.5.1 Friedl and Rónyai

This is the data for RadicalPrimeFR.

Table A.51: Jacobson radical experiment, Friedl and Rónyai algorithm, run 1.

Group	dim A	dim $J(A)$	Group	Mult. Table	Matrix
SmallGroup(2, 1)	2	1	39	9	5
SmallGroup(3, 1)	3	2	272	26	9
SmallGroup(4, 1)	4	3	88	48	15
SmallGroup(5, 1)	5	4	200	64	18
SmallGroup(6, 1)	6	4	155	101	26
SmallGroup(7, 1)	7	6	300	161	41
SmallGroup(8, 5)	8	7	522	440	97
SmallGroup(9, 2)	9	8	662	514	115
SmallGroup(10, 1)	10	8	569	472	105
SmallGroup(11, 1)	11	10	926	661	156
SmallGroup(12, 1)	12	8	1342	1149	246
SmallGroup(13, 1)	13	12	1509	1160	266
SmallGroup(14, 1)	14	12	1755	1487	326
SmallGroup(15, 1)	15	12	2124	1838	378
SmallGroup(16, 13)	16	15	6145	5615	1171
SmallGroup(17, 1)	17	16	3564	2971	650
SmallGroup(18, 2)	18	16	5972	5344	1076
SmallGroup(19, 1)	19	18	5280	4447	998
SmallGroup(20, 5)	20	16	5698	5096	1023
SmallGroup(21, 1)	21	18	7134	6215	1301
SmallGroup(22, 2)	22	20	8393	7435	1598
SmallGroup(23, 1)	23	22	10362	9015	2096
SmallGroup(24, 12)	24	4	12230	10925	2224
SmallGroup(25, 1)	25	24	20668	19025	4465
SmallGroup(26, 2)	26	24	15890	13872	2939
SmallGroup(27, 5)	27	26	36444	33612	7641
SmallGroup(28, 2)	28	24	20485	18627	4236
SmallGroup(29, 1)	29	28	24873	22186	5470
SmallGroup(30, 3)	30	24	40211	36496	9846
SmallGroup(31, 1)	31	30	32360	29041	7035
SmallGroup(32, 30)	32	31	94929	90622	20760
SmallGroup(33, 1)	33	30	38096	35095	8120
SmallGroup(34, 1)	34	32	42906	39635	9095
SmallGroup(35, 1)	35	30	48037	44211	9540
SmallGroup(36, 13)	36	32	108007	100196	24856
SmallGroup(37, 1)	37	36	62266	55884	13574
SmallGroup(38, 1)	38	36	66639	61478	14178
SmallGroup(39, 2)	39	36	72689	68611	14751
SmallGroup(40, 3)	40	32	118378	111898	29001
SmallGroup(41, 1)	41	40	92266	84462	20340
SmallGroup(42, 4)	42	36	95276	88060	18465
SmallGroup(43, 1)	43	42	112666	104453	26012
SmallGroup(44, 2)	44	40	116645	108871	23956
SmallGroup(45, 2)	45	36	186502	176615	45191
SmallGroup(46, 2)	46	44	142507	133275	31847
SmallGroup(47, 1)	47	46	161498	148449	37464
SmallGroup(48, 34)	48	32	283286	261922	60727
SmallGroup(49, 2)	49	48	274387	260002	60450
SmallGroup(50, 2)	50	48	294518	278919	60294

Table A.52: Jacobson radical experiment, Friedl and Rónyai algorithm, run 2.

Group	dim A	dim $J(A)$	Group	Mult. Table	Matrix
SmallGroup(2, 1)	2	1	39	9	5
SmallGroup(3, 1)	3	2	271	26	9
SmallGroup(4, 1)	4	3	88	50	15
SmallGroup(5, 1)	5	4	197	65	19
SmallGroup(6, 1)	6	4	153	100	25
SmallGroup(7, 1)	7	6	300	160	41
SmallGroup(8, 5)	8	7	523	448	97
SmallGroup(9, 2)	9	8	672	513	106
SmallGroup(10, 1)	10	8	575	473	106
SmallGroup(11, 1)	11	10	925	662	154
SmallGroup(12, 1)	12	8	1346	1150	238
SmallGroup(13, 1)	13	12	1506	1170	267
SmallGroup(14, 1)	14	12	1731	1502	317
SmallGroup(15, 1)	15	12	2117	1861	380
SmallGroup(16, 13)	16	15	6102	5675	1166
SmallGroup(17, 1)	17	16	3552	2943	654
SmallGroup(18, 2)	18	16	5966	5350	1060
SmallGroup(19, 1)	19	18	5249	4462	1012
SmallGroup(20, 5)	20	16	5754	5102	1006
SmallGroup(21, 1)	21	18	7093	6152	1290
SmallGroup(22, 2)	22	20	8376	7450	1608
SmallGroup(23, 1)	23	22	10382	8958	2074
SmallGroup(24, 12)	24	4	12056	10867	2183
SmallGroup(25, 1)	25	24	20286	18761	4425
SmallGroup(26, 2)	26	24	15644	13660	2896
SmallGroup(27, 5)	27	26	36136	33311	7603
SmallGroup(28, 2)	28	24	20370	18638	4209
SmallGroup(29, 1)	29	28	24754	22046	5433
SmallGroup(30, 3)	30	24	40021	36344	9800
SmallGroup(31, 1)	31	30	32339	28929	7031
SmallGroup(32, 30)	32	31	94658	90216	20567
SmallGroup(33, 1)	33	30	37795	34990	8066
SmallGroup(34, 1)	34	32	42446	39232	9137
SmallGroup(35, 1)	35	30	47872	43531	9412
SmallGroup(36, 13)	36	32	107280	99499	24812
SmallGroup(37, 1)	37	36	62376	55708	13521
SmallGroup(38, 1)	38	36	66576	61746	14145
SmallGroup(39, 2)	39	36	73597	68492	14873
SmallGroup(40, 3)	40	32	119047	112087	29013
SmallGroup(41, 1)	41	40	92496	84745	20209
SmallGroup(42, 4)	42	36	96079	88604	18505
SmallGroup(43, 1)	43	42	113670	103995	25992
SmallGroup(44, 2)	44	40	117520	110207	23994
SmallGroup(45, 2)	45	36	187454	176080	45278
SmallGroup(46, 2)	46	44	142421	133673	32007
SmallGroup(47, 1)	47	46	162416	149079	37430
SmallGroup(48, 34)	48	32	283885	263206	60486
SmallGroup(49, 2)	49	48	273854	259027	59977
SmallGroup(50, 2)	50	48	293737	278382	60137

Table A.53: Jacobson radical experiment, Friedl and Rónyai algorithm, run 3.

Group	dim A	dim $J(A)$	Group	Mult. Table	Matrix
SmallGroup(2, 1)	2	1	39	10	5
SmallGroup(3, 1)	3	2	278	27	9
SmallGroup(4, 1)	4	3	88	49	14
SmallGroup(5, 1)	5	4	198	64	19
SmallGroup(6, 1)	6	4	155	101	25
SmallGroup(7, 1)	7	6	300	154	41
SmallGroup(8, 5)	8	7	523	447	96
SmallGroup(9, 2)	9	8	678	513	114
SmallGroup(10, 1)	10	8	576	474	105
SmallGroup(11, 1)	11	10	917	661	154
SmallGroup(12, 1)	12	8	1349	1152	246
SmallGroup(13, 1)	13	12	1508	1143	265
SmallGroup(14, 1)	14	12	1712	1481	322
SmallGroup(15, 1)	15	12	2107	1836	376
SmallGroup(16, 13)	16	15	6079	5629	1157
SmallGroup(17, 1)	17	16	3498	2928	647
SmallGroup(18, 2)	18	16	5891	5283	1072
SmallGroup(19, 1)	19	18	5202	4427	996
SmallGroup(20, 5)	20	16	5686	5061	1018
SmallGroup(21, 1)	21	18	7013	6111	1282
SmallGroup(22, 2)	22	20	8305	7392	1594
SmallGroup(23, 1)	23	22	10300	8878	2060
SmallGroup(24, 12)	24	4	12075	10863	2199
SmallGroup(25, 1)	25	24	20276	18750	4407
SmallGroup(26, 2)	26	24	15630	13675	2892
SmallGroup(27, 5)	27	26	36025	33014	7558
SmallGroup(28, 2)	28	24	20163	18452	4175
SmallGroup(29, 1)	29	28	24651	21924	5367
SmallGroup(30, 3)	30	24	39795	36149	9778
SmallGroup(31, 1)	31	30	32207	28691	6988
SmallGroup(32, 30)	32	31	94177	89637	20412
SmallGroup(33, 1)	33	30	37694	34909	8048
SmallGroup(34, 1)	34	32	42551	39344	9070
SmallGroup(35, 1)	35	30	47899	43763	9418
SmallGroup(36, 13)	36	32	107163	98814	24763
SmallGroup(37, 1)	37	36	61945	55520	13533
SmallGroup(38, 1)	38	36	66379	61209	14150
SmallGroup(39, 2)	39	36	72478	67568	14734
SmallGroup(40, 3)	40	32	117972	111067	28904
SmallGroup(41, 1)	41	40	92207	84116	20191
SmallGroup(42, 4)	42	36	94968	87512	18403
SmallGroup(43, 1)	43	42	112541	103638	25928
SmallGroup(44, 2)	44	40	116515	108704	23890
SmallGroup(45, 2)	45	36	186031	175166	45090
SmallGroup(46, 2)	46	44	142207	132708	31740
SmallGroup(47, 1)	47	46	161391	147860	37364
SmallGroup(48, 34)	48	32	280862	261594	60862
SmallGroup(49, 2)	49	48	275224	258909	59919
SmallGroup(50, 2)	50	48	294571	280023	60273

Table A.54: Jacobson radical experiment, Friedl and Rónyai algorithm, run 4.

Group	dim A	dim $J(A)$	Group	Mult. Table	Matrix
SmallGroup(2, 1)	2	1	39	10	6
SmallGroup(3, 1)	3	2	279	26	10
SmallGroup(4, 1)	4	3	88	50	15
SmallGroup(5, 1)	5	4	198	64	18
SmallGroup(6, 1)	6	4	154	101	26
SmallGroup(7, 1)	7	6	301	162	41
SmallGroup(8, 5)	8	7	524	449	97
SmallGroup(9, 2)	9	8	672	513	115
SmallGroup(10, 1)	10	8	575	465	105
SmallGroup(11, 1)	11	10	924	661	154
SmallGroup(12, 1)	12	8	1347	1149	246
SmallGroup(13, 1)	13	12	1498	1157	265
SmallGroup(14, 1)	14	12	1703	1479	321
SmallGroup(15, 1)	15	12	2087	1837	369
SmallGroup(16, 13)	16	15	6077	5636	1155
SmallGroup(17, 1)	17	16	3520	2930	648
SmallGroup(18, 2)	18	16	5919	5283	1073
SmallGroup(19, 1)	19	18	5192	4436	988
SmallGroup(20, 5)	20	16	5697	5076	1018
SmallGroup(21, 1)	21	18	7026	6128	1283
SmallGroup(22, 2)	22	20	8305	7382	1593
SmallGroup(23, 1)	23	22	10291	8882	2036
SmallGroup(24, 12)	24	4	12062	10866	2199
SmallGroup(25, 1)	25	24	20309	18746	4423
SmallGroup(26, 2)	26	24	15671	13667	2870
SmallGroup(27, 5)	27	26	36124	33062	7551
SmallGroup(28, 2)	28	24	20139	18426	4189
SmallGroup(29, 1)	29	28	24629	21934	5397
SmallGroup(30, 3)	30	24	39815	36184	9761
SmallGroup(31, 1)	31	30	32180	28711	6989
SmallGroup(32, 30)	32	31	94091	89610	20450
SmallGroup(33, 1)	33	30	37582	34652	8027
SmallGroup(34, 1)	34	32	42288	39102	9064
SmallGroup(35, 1)	35	30	47597	43481	9433
SmallGroup(36, 13)	36	32	106530	98796	24803
SmallGroup(37, 1)	37	36	61977	55547	13541
SmallGroup(38, 1)	38	36	66408	61273	14173
SmallGroup(39, 2)	39	36	72536	67611	14723
SmallGroup(40, 3)	40	32	117823	111047	28878
SmallGroup(41, 1)	41	40	92059	84130	20186
SmallGroup(42, 4)	42	36	94973	87370	18377
SmallGroup(43, 1)	43	42	112516	103472	25911
SmallGroup(44, 2)	44	40	116390	108548	23902
SmallGroup(45, 2)	45	36	186082	175118	45122
SmallGroup(46, 2)	46	44	142090	132605	31706
SmallGroup(47, 1)	47	46	161230	147750	37390
SmallGroup(48, 34)	48	32	280156	261102	60600
SmallGroup(49, 2)	49	48	273774	258862	59974
SmallGroup(50, 2)	50	48	294579	278220	60145

Table A.55: Jacobson radical experiment, Friedl and Rónyai algorithm, run 5.

Group	dim A	dim $J(A)$	Group	Mult. Table	Matrix
SmallGroup(2, 1)	2	1	39	10	6
SmallGroup(3, 1)	3	2	270	26	8
SmallGroup(4, 1)	4	3	87	49	15
SmallGroup(5, 1)	5	4	198	64	19
SmallGroup(6, 1)	6	4	154	100	26
SmallGroup(7, 1)	7	6	300	161	41
SmallGroup(8, 5)	8	7	516	447	97
SmallGroup(9, 2)	9	8	671	513	114
SmallGroup(10, 1)	10	8	568	474	106
SmallGroup(11, 1)	11	10	927	661	154
SmallGroup(12, 1)	12	8	1348	1151	239
SmallGroup(13, 1)	13	12	1499	1159	265
SmallGroup(14, 1)	14	12	1704	1480	321
SmallGroup(15, 1)	15	12	2104	1835	369
SmallGroup(16, 13)	16	15	6071	5633	1156
SmallGroup(17, 1)	17	16	3512	2913	647
SmallGroup(18, 2)	18	16	5911	5300	1057
SmallGroup(19, 1)	19	18	5210	4410	995
SmallGroup(20, 5)	20	16	5692	5070	1003
SmallGroup(21, 1)	21	18	7032	6126	1283
SmallGroup(22, 2)	22	20	8295	7391	1594
SmallGroup(23, 1)	23	22	10313	8870	2067
SmallGroup(24, 12)	24	4	12088	10861	2199
SmallGroup(25, 1)	25	24	20312	18733	4429
SmallGroup(26, 2)	26	24	15684	13697	2900
SmallGroup(27, 5)	27	26	35982	33038	7557
SmallGroup(28, 2)	28	24	20145	18438	4190
SmallGroup(29, 1)	29	28	24623	21933	5396
SmallGroup(30, 3)	30	24	39802	36258	9824
SmallGroup(31, 1)	31	30	32508	29055	7044
SmallGroup(32, 30)	32	31	94523	89779	20465
SmallGroup(33, 1)	33	30	37747	34941	8095
SmallGroup(34, 1)	34	32	42547	39100	9031
SmallGroup(35, 1)	35	30	47583	43849	9479
SmallGroup(36, 13)	36	32	107507	99802	24781
SmallGroup(37, 1)	37	36	62485	56230	13516
SmallGroup(38, 1)	38	36	66421	61631	14272
SmallGroup(39, 2)	39	36	73389	67719	14864
SmallGroup(40, 3)	40	32	118100	111029	29092
SmallGroup(41, 1)	41	40	92201	84371	20236
SmallGroup(42, 4)	42	36	95377	87433	18421
SmallGroup(43, 1)	43	42	112593	103592	25943
SmallGroup(44, 2)	44	40	118108	108615	23906
SmallGroup(45, 2)	45	36	185974	175595	45136
SmallGroup(46, 2)	46	44	142679	133380	31747
SmallGroup(47, 1)	47	46	161938	149104	37441
SmallGroup(48, 34)	48	32	283685	261168	60523
SmallGroup(49, 2)	49	48	273715	259098	59992
SmallGroup(50, 2)	50	48	294271	278207	60093

A.5.2 Cohen, Ivanyos, Wales

This is the data for RadicalPrimeCIW.

Table A.56: Jacobson radical experiment, Cohen, et al. algorithm, run 1.

Group	dim A	dim $J(A)$	Group	Mult. Table	Matrix
SmallGroup(2, 1)	2	1	23	15	12
SmallGroup(3, 1)	3	2	312	33	24
SmallGroup(4, 1)	4	3	88	47	34
SmallGroup(5, 1)	5	4	236	86	61
SmallGroup(6, 1)	6	4	230	156	101
SmallGroup(7, 1)	7	6	351	172	126
SmallGroup(8, 5)	8	7	331	271	195
SmallGroup(9, 2)	9	8	536	412	294
SmallGroup(10, 1)	10	8	663	555	398
SmallGroup(11, 1)	11	10	819	576	387
SmallGroup(12, 1)	12	8	1580	1370	984
SmallGroup(13, 1)	13	12	1225	916	614
SmallGroup(14, 1)	14	12	1658	1427	1011
SmallGroup(15, 1)	15	12	1748	1509	1015
SmallGroup(16, 13)	16	15	2097	1916	1332
SmallGroup(17, 1)	17	16	2553	2030	1347
SmallGroup(18, 2)	18	16	3034	2639	1789
SmallGroup(19, 1)	19	18	3530	2868	1893
SmallGroup(20, 5)	20	16	4368	3852	2619
SmallGroup(21, 1)	21	18	5402	4789	3395
SmallGroup(22, 2)	22	20	5455	4816	3173
SmallGroup(23, 1)	23	22	6402	5326	3481
SmallGroup(24, 12)	24	4	9122	8200	6723
SmallGroup(25, 1)	25	24	8064	7339	5615
SmallGroup(26, 2)	26	24	9600	8404	6232
SmallGroup(27, 5)	27	26	12611	11374	8896
SmallGroup(28, 2)	28	24	12151	10854	8006
SmallGroup(29, 1)	29	28	13729	11711	8340
SmallGroup(30, 3)	30	24	30933	28305	22559
SmallGroup(31, 1)	31	30	18148	14843	11194
SmallGroup(32, 30)	32	31	20597	19125	15236
SmallGroup(33, 1)	33	30	22195	20453	14912
SmallGroup(34, 1)	34	32	28170	25567	19673
SmallGroup(35, 1)	35	30	27171	24569	17831
SmallGroup(36, 13)	36	32	67280	63064	51100
SmallGroup(37, 1)	37	36	33147	29443	21043
SmallGroup(38, 1)	38	36	41665	38109	29342
SmallGroup(39, 2)	39	36	40013	35897	26374
SmallGroup(40, 3)	40	32	82629	76505	59293
SmallGroup(41, 1)	41	40	48206	43001	31508
SmallGroup(42, 4)	42	36	56035	51671	39511
SmallGroup(43, 1)	43	42	57777	51314	37594
SmallGroup(44, 2)	44	40	63365	57561	42172
SmallGroup(45, 2)	45	36	124897	117457	90801
SmallGroup(46, 2)	46	44	74263	68236	49871
SmallGroup(47, 1)	47	46	81455	72983	53277
SmallGroup(48, 34)	48	32	144803	135412	105974
SmallGroup(49, 2)	49	48	98446	91456	68085
SmallGroup(50, 2)	50	48	100277	94676	69449

Table A.57: Jacobson radical experiment, Cohen, et al. algorithm, run 2.

Group	dim A	dim $J(A)$	Group	Mult. Table	Matrix
SmallGroup(2, 1)	2	1	22	15	11
SmallGroup(3, 1)	3	2	303	33	25
SmallGroup(4, 1)	4	3	87	47	34
SmallGroup(5, 1)	5	4	239	86	61
SmallGroup(6, 1)	6	4	229	155	109
SmallGroup(7, 1)	7	6	351	180	127
SmallGroup(8, 5)	8	7	323	273	195
SmallGroup(9, 2)	9	8	561	412	294
SmallGroup(10, 1)	10	8	664	555	397
SmallGroup(11, 1)	11	10	819	568	389
SmallGroup(12, 1)	12	8	1582	1361	985
SmallGroup(13, 1)	13	12	1228	900	614
SmallGroup(14, 1)	14	12	1651	1426	1023
SmallGroup(15, 1)	15	12	1735	1508	1020
SmallGroup(16, 13)	16	15	2105	1910	1339
SmallGroup(17, 1)	17	16	2551	2039	1341
SmallGroup(18, 2)	18	16	3035	2652	1781
SmallGroup(19, 1)	19	18	3529	2857	1895
SmallGroup(20, 5)	20	16	4391	3827	2604
SmallGroup(21, 1)	21	18	5389	4782	3387
SmallGroup(22, 2)	22	20	5467	4803	3173
SmallGroup(23, 1)	23	22	6387	5320	3490
SmallGroup(24, 12)	24	4	9122	8181	6695
SmallGroup(25, 1)	25	24	8051	7141	5539
SmallGroup(26, 2)	26	24	9365	8342	6227
SmallGroup(27, 5)	27	26	12596	11334	8923
SmallGroup(28, 2)	28	24	12162	10817	8008
SmallGroup(29, 1)	29	28	13711	11746	8342
SmallGroup(30, 3)	30	24	31113	28565	22617
SmallGroup(31, 1)	31	30	18128	14802	11203
SmallGroup(32, 30)	32	31	20579	19092	15191
SmallGroup(33, 1)	33	30	22474	20166	14756
SmallGroup(34, 1)	34	32	28250	25819	19559
SmallGroup(35, 1)	35	30	27093	24468	17823
SmallGroup(36, 13)	36	32	67064	63007	51216
SmallGroup(37, 1)	37	36	33093	29388	21071
SmallGroup(38, 1)	38	36	41566	38119	29292
SmallGroup(39, 2)	39	36	40010	35774	26373
SmallGroup(40, 3)	40	32	82445	76597	59551
SmallGroup(41, 1)	41	40	48545	43079	31557
SmallGroup(42, 4)	42	36	55954	51507	39621
SmallGroup(43, 1)	43	42	58572	51728	37636
SmallGroup(44, 2)	44	40	63554	57779	42221
SmallGroup(45, 2)	45	36	125823	117025	90953
SmallGroup(46, 2)	46	44	74170	68694	49727
SmallGroup(47, 1)	47	46	81889	73284	53415
SmallGroup(48, 34)	48	32	145785	136164	106538
SmallGroup(49, 2)	49	48	99039	91932	68615
SmallGroup(50, 2)	50	48	101017	94670	69403

Table A.58: Jacobson radical experiment, Cohen, et al. algorithm, run 3.

Group	dim A	dim $J(A)$	Group	Mult. Table	Matrix
SmallGroup(2, 1)	2	1	23	15	12
SmallGroup(3, 1)	3	2	302	32	24
SmallGroup(4, 1)	4	3	87	48	33
SmallGroup(5, 1)	5	4	238	86	61
SmallGroup(6, 1)	6	4	222	155	109
SmallGroup(7, 1)	7	6	350	181	126
SmallGroup(8, 5)	8	7	333	272	187
SmallGroup(9, 2)	9	8	552	412	294
SmallGroup(10, 1)	10	8	648	554	397
SmallGroup(11, 1)	11	10	817	574	389
SmallGroup(12, 1)	12	8	1581	1362	983
SmallGroup(13, 1)	13	12	1226	908	613
SmallGroup(14, 1)	14	12	1656	1410	1023
SmallGroup(15, 1)	15	12	1748	1509	1019
SmallGroup(16, 13)	16	15	2080	1915	1339
SmallGroup(17, 1)	17	16	2545	2036	1349
SmallGroup(18, 2)	18	16	3035	2650	1787
SmallGroup(19, 1)	19	18	3510	2868	1894
SmallGroup(20, 5)	20	16	4372	3828	2592
SmallGroup(21, 1)	21	18	5395	4793	3370
SmallGroup(22, 2)	22	20	5476	4789	3162
SmallGroup(23, 1)	23	22	6379	5324	3480
SmallGroup(24, 12)	24	4	9095	8183	6696
SmallGroup(25, 1)	25	24	8067	7148	5538
SmallGroup(26, 2)	26	24	9375	8354	6240
SmallGroup(27, 5)	27	26	12568	11312	8919
SmallGroup(28, 2)	28	24	12133	10804	7998
SmallGroup(29, 1)	29	28	13702	11735	8345
SmallGroup(30, 3)	30	24	30869	28590	22717
SmallGroup(31, 1)	31	30	18180	14847	11195
SmallGroup(32, 30)	32	31	20724	19113	15221
SmallGroup(33, 1)	33	30	22147	20071	14758
SmallGroup(34, 1)	34	32	28112	25510	19577
SmallGroup(35, 1)	35	30	27065	24448	17814
SmallGroup(36, 13)	36	32	67020	64200	51450
SmallGroup(37, 1)	37	36	33098	29580	21123
SmallGroup(38, 1)	38	36	41594	38120	29272
SmallGroup(39, 2)	39	36	39940	35800	26324
SmallGroup(40, 3)	40	32	82418	76857	59377
SmallGroup(41, 1)	41	40	48204	42859	31510
SmallGroup(42, 4)	42	36	55899	51538	39531
SmallGroup(43, 1)	43	42	57710	51189	37616
SmallGroup(44, 2)	44	40	63469	57655	42377
SmallGroup(45, 2)	45	36	124882	115474	90238
SmallGroup(46, 2)	46	44	73568	67875	49455
SmallGroup(47, 1)	47	46	81000	72455	53120
SmallGroup(48, 34)	48	32	144232	134463	105806
SmallGroup(49, 2)	49	48	97743	90684	67970
SmallGroup(50, 2)	50	48	100055	93882	68963

Table A.59: Jacobson radical experiment, Cohen, et al. algorithm, run 4.

Group	dim A	dim $J(A)$	Group	Mult. Table	Matrix
SmallGroup(2, 1)	2	1	23	14	11
SmallGroup(3, 1)	3	2	305	31	24
SmallGroup(4, 1)	4	3	88	49	34
SmallGroup(5, 1)	5	4	238	85	61
SmallGroup(6, 1)	6	4	231	155	108
SmallGroup(7, 1)	7	6	350	181	126
SmallGroup(8, 5)	8	7	332	272	195
SmallGroup(9, 2)	9	8	552	412	295
SmallGroup(10, 1)	10	8	665	555	381
SmallGroup(11, 1)	11	10	817	575	388
SmallGroup(12, 1)	12	8	1574	1367	985
SmallGroup(13, 1)	13	12	1228	914	613
SmallGroup(14, 1)	14	12	1659	1422	1014
SmallGroup(15, 1)	15	12	1748	1501	1019
SmallGroup(16, 13)	16	15	2102	1913	1340
SmallGroup(17, 1)	17	16	2544	2020	1348
SmallGroup(18, 2)	18	16	3032	2699	1815
SmallGroup(19, 1)	19	18	3591	2875	1942
SmallGroup(20, 5)	20	16	4455	3906	2648
SmallGroup(21, 1)	21	18	5482	4858	3443
SmallGroup(22, 2)	22	20	5536	4872	3217
SmallGroup(23, 1)	23	22	6508	5410	3539
SmallGroup(24, 12)	24	4	9233	8216	6708
SmallGroup(25, 1)	25	24	8066	7134	5601
SmallGroup(26, 2)	26	24	9504	8367	6227
SmallGroup(27, 5)	27	26	12617	11517	8968
SmallGroup(28, 2)	28	24	12338	10837	8012
SmallGroup(29, 1)	29	28	13785	11933	8414
SmallGroup(30, 3)	30	24	31395	28801	22914
SmallGroup(31, 1)	31	30	18396	14787	11167
SmallGroup(32, 30)	32	31	20558	19073	15227
SmallGroup(33, 1)	33	30	22125	20059	14768
SmallGroup(34, 1)	34	32	27866	25535	19591
SmallGroup(35, 1)	35	30	27063	24475	17839
SmallGroup(36, 13)	36	32	67099	63101	51381
SmallGroup(37, 1)	37	36	33680	29910	21063
SmallGroup(38, 1)	38	36	41652	38340	29363
SmallGroup(39, 2)	39	36	39999	35939	26371
SmallGroup(40, 3)	40	32	82433	76486	59149
SmallGroup(41, 1)	41	40	48095	42897	31901
SmallGroup(42, 4)	42	36	57704	52446	39826
SmallGroup(43, 1)	43	42	57668	51972	38712
SmallGroup(44, 2)	44	40	65210	57420	42980
SmallGroup(45, 2)	45	36	127169	117944	91996
SmallGroup(46, 2)	46	44	77172	67860	49526
SmallGroup(47, 1)	47	46	81002	72529	53167
SmallGroup(48, 34)	48	32	144228	134317	105857
SmallGroup(49, 2)	49	48	97692	90591	67961
SmallGroup(50, 2)	50	48	99682	93459	68961

Table A.60: Jacobson radical experiment, Cohen, et al. algorithm, run 5.

Group	dim A	dim $J(A)$	Group	Mult. Table	Matrix
SmallGroup(2, 1)	2	1	23	15	12
SmallGroup(3, 1)	3	2	305	33	24
SmallGroup(4, 1)	4	3	87	48	34
SmallGroup(5, 1)	5	4	238	85	61
SmallGroup(6, 1)	6	4	230	156	109
SmallGroup(7, 1)	7	6	344	180	126
SmallGroup(8, 5)	8	7	332	274	195
SmallGroup(9, 2)	9	8	560	405	294
SmallGroup(10, 1)	10	8	663	555	398
SmallGroup(11, 1)	11	10	810	576	388
SmallGroup(12, 1)	12	8	1580	1371	985
SmallGroup(13, 1)	13	12	1214	912	612
SmallGroup(14, 1)	14	12	1658	1419	1022
SmallGroup(15, 1)	15	12	1722	1498	1017
SmallGroup(16, 13)	16	15	2146	1952	1346
SmallGroup(17, 1)	17	16	2619	2055	1354
SmallGroup(18, 2)	18	16	3070	2695	1797
SmallGroup(19, 1)	19	18	3623	2961	1891
SmallGroup(20, 5)	20	16	4390	3826	2607
SmallGroup(21, 1)	21	18	5388	4771	3389
SmallGroup(22, 2)	22	20	5457	4792	3167
SmallGroup(23, 1)	23	22	6400	5330	3470
SmallGroup(24, 12)	24	4	9085	8167	6709
SmallGroup(25, 1)	25	24	8045	7131	5542
SmallGroup(26, 2)	26	24	9378	8357	6238
SmallGroup(27, 5)	27	26	12583	11312	8929
SmallGroup(28, 2)	28	24	12145	10807	8083
SmallGroup(29, 1)	29	28	13741	11709	8392
SmallGroup(30, 3)	30	24	30907	28353	22576
SmallGroup(31, 1)	31	30	18142	14779	11156
SmallGroup(32, 30)	32	31	20564	19077	15173
SmallGroup(33, 1)	33	30	22196	20082	14758
SmallGroup(34, 1)	34	32	27870	25510	19603
SmallGroup(35, 1)	35	30	27093	24543	17825
SmallGroup(36, 13)	36	32	67209	63240	51220
SmallGroup(37, 1)	37	36	33056	29659	21232
SmallGroup(38, 1)	38	36	42004	38376	29408
SmallGroup(39, 2)	39	36	40006	35816	26331
SmallGroup(40, 3)	40	32	83356	76672	59317
SmallGroup(41, 1)	41	40	48108	42869	31543
SmallGroup(42, 4)	42	36	55875	51520	39465
SmallGroup(43, 1)	43	42	57714	51221	37589
SmallGroup(44, 2)	44	40	63233	57450	42149
SmallGroup(45, 2)	45	36	124803	115422	90243
SmallGroup(46, 2)	46	44	73551	67880	49478
SmallGroup(47, 1)	47	46	81012	72480	53133
SmallGroup(48, 34)	48	32	144146	134376	105789
SmallGroup(49, 2)	49	48	97711	90727	67949
SmallGroup(50, 2)	50	48	99740	93480	68973

VITA

Craig Andrew Struble was born on September 20, 1970 in Kingston, NY. His family relocated to Richmond, VA shortly thereafter. Craig always enjoyed mathematical puzzles and started toying with computers in 1980 when his mother brought home a Commodore VIC-20. He attended Reams Road Elementary, Swift Creek Middle School, and was graduated from Monacan High School in 1988.

In 1993, Craig received a Bachelor of Science degree in Computer Science with a second major in Mathematics, *summa cum laude* from Virginia Polytechnic Institute and State University. During his undergraduate degree, he was a co-operative education student at Bell Northern Research and Northern Telecom (now Nortel Networks).

After graduation, Craig was employed at Pacific Northwest National Laboratory in 1993–1994. He worked on a number of different projects, including a multimedia sampler, a prototype user interface for a policy database, and visualization of high-dimensional data (now known as SPIRE).

In 1994, Craig returned to Virginia Polytechnic Institute and State University to pursue a Ph.D. in Computer Science. He received his Master of Science degree in Computer Science in 1996. Later in 1996, Craig accepted an internship at Sun Microsystems to work on collaborative software support in Java, in order to slow his progress. Craig later returned to school and married his beautiful wife, Cara Linette Cocking, on May 17, 1997. After several more years of pondering mathematics and computer science, Craig was prepared to graduate with his Ph.D. in Computer Science.