Practical Implementation of Matrix Multiplication Algorithm From SUSP

Khai Dong, Matthew Anderson (Advisor)

November 15, 2023

Abstract

In 2003, Cohn and Umans [7] gave a matrix multiplication framework and conjectured an upper bound of $\mathcal{O}(n^{2+o(1)})$ for their framework. Since this framework conjectured the lowest possible upperbound for matrix multiplication, it has been an active area of research [2, 3]. Continuing the efforts of Anderson et al. [2, 3], we have fully implemented this algorithm and the related computation task, verified its actual runtime against Cohn and Umans' conjecture, and derive the break-even threshold of Cohn-Umans matrix multiplication against the naive one.

Contents

List of Figures

List of Tables

1 Introduction

Matrix multiplication is important in various fields due to its versatility in a wide range of theoretical and empirical works. It served as the foundation of linear algebra, where, in example, it represents systems of equations, linear transformations, and other criteria. In empirical applications, matrix multiplication is used in machine learning to train mathematical models which ultimately translate into image and speech recognition, weather prediction, fraud detection, and many other useful applications [1] .

Because of this great importance, improving the runtime of matrix multiplication is a nontrivial task as speeding up this specific operation speeds up a wide array of related tasks. Over the years, researchers have developed many algorithms to perform faster matrix multiplication. This research aimed to implement one of such algorithms which conjectured $\mathcal{O}(n^{2+o(1)})$ runtime. This algorithm was first introduced by Cohn and Umans [7] in 2003, and their subsequent work [6] further elaborated on to find the unexplored parameters to allow for efficient algorithms they conjectured.

1.1 Prior Work

In 1969, Strassen [20] developed a $\mathcal{O}(n^{2.81})$ matrix multiplication algorithm, improving the runtime of multiplying matrices from $\mathcal{O}(n^3)$. More elaboration on this $\mathcal{O}$-notation is available in Section 2.2, and matrix multiplication researchers aim to reduce the exponent of n to the smallest possible. For brevity, we define ω to be the exponent of the runtime $O(n^\omega)$, where, in the case of Strassen [20]'s algorithm, $\omega = 2.81$. In 1987, Coppersmith and Winograd [8] made improvements upon Strassen [20]'s algorithm, yielding $\omega = 2.38$. This improvement raised the question of what is the lowest ω achievable.

Another approach was proposed by Cohn and Umans [7] in 2003, conjecturing $\omega = 2 + o(1)$ which is the smallest ω could be. However, this framework is largely unexplored and unimplemented and thresholds of matrix sizes where Cohn-Umans algorithm gives improvements was unclear.

1.2 Our Results

We fully implements Cohn-Umans matrix multiplication algorithm and procedures to compute the Wedderburn Decomposition that is required to use Cohn-Umans algorithm. We also attempts to give a better upperbound for ω from our analysis and derives the break-even threshold for this algorithm compared to the naive one, and creates a simulation the runtime of this framework for some set of parameters. The remaining of this report elaborated on Section 2, this report goes through the methodology of this research. Section 2 goes into the necessary background for this research. Section 3 goes into how the Cohn Umans algorithm works and the related computations. Section 4 goes into our implementation, specifically, the challenges involve and the design choices made. Section 5 goes into analyzing the runtime of the implemented algorithm. Section 6 concludes our findings, gives the limitation of our work, and proposes paths forward.

2 Preliminaries

This section presents some background information to build up to Cohn Umans matrix multiplication framework ranging from basic linear algebra and graduate-level abstract algebra to algorithm theory. We noted that the abstract algebra and representation used is based on *Abstract Algebra* by Dummit and Foote [13] and algorithm theory is based on *Introduction to Algorithms* by Cormen et al. [9]. Since this section only provides math and computer science backgrounds, skipping ahead and referring back to this section as needed are recommended. This section assumed that the readers know the basic ideas of sets, functions, and linear algebra.

2.1 Matrices and Matrix Multiplication

A **matrix** is a rectangular array of numbers or symbolic expressions arranged into rows and columns. A matrix with m rows and n columns has $m \times n$ entries where m and n are arbitrary positive integers. An example of a 2×3 matrix with 6 entries of integers is

$$\begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix}$$

Given this definition, it is possible to have a matrix with matrix entries. These matrices are called **block matrices**. For example, we have

$$\begin{bmatrix} \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix} & \begin{bmatrix} 2 & 6 & 6 \\ 3 & 4 & 6 \end{bmatrix} \\ \begin{bmatrix} 2 & 6 & 6 \\ 3 & 4 & 6 \end{bmatrix} & \begin{bmatrix} 2 & 4 & 0 \\ 1 & 9 & 6 \end{bmatrix} \end{bmatrix} \mapsto \begin{bmatrix} 1 & 2 & 3 & 2 & 6 & 6 \\ 4 & 5 & 6 & 3 & 4 & 6 \\ 2 & 6 & 6 & 2 & 4 & 0 \\ 3 & 4 & 6 & 1 & 9 & 6 \end{bmatrix}$$

as a matrix with entries are matrices with integers entries which is equivalent to a matrix of scalars. We can also go from the matrix of scalar to the block matrices by **subdividing** the matrix of scalars into a 2×2 block matrix.

Matrix multiplication between two matrices is defined when the number of columns in the first matrix is equal to the number of rows in the second matrix. In more mathematical terms, the multiplication of matrix A of size $m \times n$ and matrix B of size $n \times p$ is defined as A and B both having n columns and rows, respectively. The resulting matrix C is of size $m \times p$. This process of multiplying two matrices involves taking the dot products of the first matrix's rows with the second matrix's columns. For example, we have,

$$\begin{bmatrix} 1 & 2 \end{bmatrix} \cdot \begin{bmatrix} 3 & 5 \\ 4 & 6 \end{bmatrix} = \begin{bmatrix} 1 \cdot 3 + 2 \cdot 4 & 1 \cdot 5 + 2 \cdot 6 \end{bmatrix} = \begin{bmatrix} 11 & 17 \end{bmatrix}$$

$$\begin{bmatrix} 1 & 2 \\ 0 & 0 \end{bmatrix} \cdot \begin{bmatrix} 3 & 5 \\ 4 & 6 \end{bmatrix} = \begin{bmatrix} 1 \cdot 3 + 2 \cdot 4 & 1 \cdot 5 + 2 \cdot 6 \\ 0 \cdot 3 + 0 \cdot 4 & 0 \cdot 5 + 0 \cdot 6 \end{bmatrix} = \begin{bmatrix} 11 & 17 \\ 0 & 0 \end{bmatrix}$$

Generally, we can observe that matrix multiplication of non-square matrices can be reduced to the problem of multiplying square matrices by adding entries 0 in the missing rows and columns. Therefore, it is sufficient to solely focus on multiplications of square matrices.

Let A and B be $n \times n$ square matrices to be of the form

$$A = \begin{bmatrix} a_{1,1} & a_{1,2} & \dots & a_{1,n} \\ a_{2,1} & a_{2,2} & \dots & a_{2,n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n,1} & a_{n,2} & \dots & a_{n,n} \end{bmatrix} \qquad B = \begin{bmatrix} b_{1,1} & b_{1,2} & \dots & b_{1,n} \\ b_{2,1} & b_{2,2} & \dots & b_{2,n} \\ \vdots & \vdots & \ddots & \vdots \\ b_{n,1} & b_{n,2} & \dots & b_{n,n} \end{bmatrix}$$

The product of these 2 matrices, $C = A \cdot B$, is defined as an $n \times n$ matrix, with entries

$$c_{i,j} = \sum_{k=1}^{n} a_{i,k} \cdot b_{k,j}$$

where $a_{i,j}$, $b_{i,j}$, and $c_{i,j}$ are entries at i^{th} row and j^{th} column of matrices A, B, and C, respectively. Naively, following the definition, each entry of C takes n multiplications and $n-1$ sums, which is a total of $2n-1$ operations. Accordingly, the computation of the whole matrix C costs $n^2 \cdot (2n-1)$ operations. For the sake of simplicity, we round this number of operations to n^3 in our analysis.

2.2 Growth of Functions and $\mathcal{O}$-notation

In comparing the efficiency of algorithms, it is the case that comparing the number of operations used or the actual runtime for some input sizes is not enough: one algorithm could be faster or slower than others for a different set of inputs. However, it is usually the case that if the input size grows larger, the runtime of the algorithm would scale with it. Therefore, this section goes into how to directly compare the algorithm using **asymptotic bound** with the usage of $\mathcal{O}$-notation.

Given function $g(n)$, we define $\Theta(g(n)))$ by the *set of functions*

$$\Theta(g(n)) = \{f(n) : \text{there exists positive constants } c_1, c_2, \text{ and } n_0 \text{ such that}$$
$$0 \leq c_1 g(n) \leq f(n) \leq c_2 g(n) \text{ for all } n \geq n_0 \geq 0\}$$

Since $\Theta(g(n))$ is a set of functions, we write $f(n) \in \Theta(g(n))$. Another alternative notation is $f(n) = \Theta(g(n))$. Among computer scientists, $\mathcal{O}(g(n))$ is usually used in place of $\Theta(g(n))$, although they mean slightly different things. For more details, please refer to Cormen et al. [9].

Using this notation, let $T(n)$ be a function of the worst-case running time of the previously described matrix multiplication algorithm (in Section 2.1) in the number of operations. We know, for matrix of size $n \times n$, $T(n) = n^2 \cdot (2n-1) = 2n^3 - n^2 = \mathcal{O}(n^3)$ (with $g(n) = n^3$, $c_1 = 1$, $c_2 = 3$, and $n_0 = 4$).
It is also the case that as $n \to \infty$, n^3 term will be much larger than the n^2 term and will cause $T(n)$ to behave almost as a function with a singular cubic term. This means, as the input size grows large enough, the only relevant growth would be that of the highest order (in this case n^3). In this research, we are concerned with an algorithm whose runtime $\mathcal{O}(n^\omega)$ where $\omega < 3$.

2.3 Recursive algorithms, Recurrences, and Master Theorem

A **recursive algorithm** refers to an algorithm that calls itself with smaller inputs than the original input. It is noted that some algorithms could be converted into an equivalent recursive one instead. An example could be calculating the sum of an array of n given integers whose recursive approach is described in Algorithm 1.

To measure the runtime of a recursive algorithm, a **recurrence** is used. A **recurrence** is an equation that

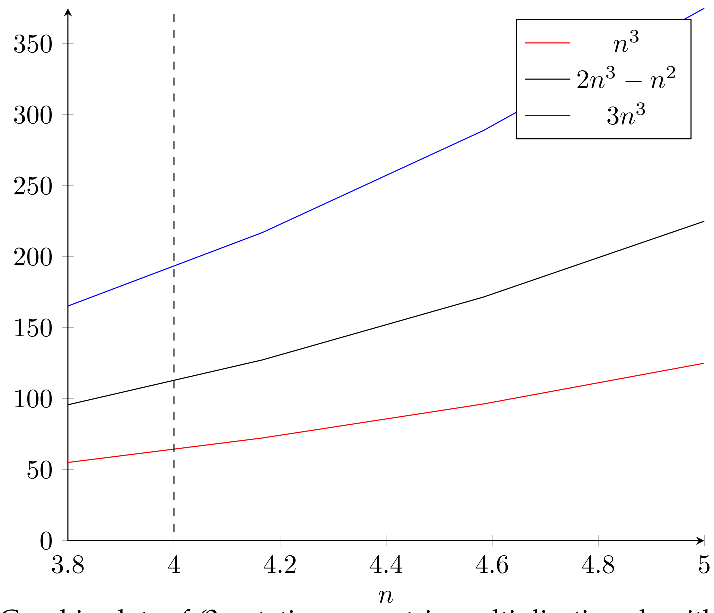

Figure 1: Graphic plots of $\mathcal{O}$-notation on matrix multiplication algorithm example

Algorithm 1: Sum of $A[1 \ldots n]$

```
Data: $A[1 \ldots n]$  $(n \geq 0)$
Result: $\sum_{i=1}^{j} A[i]$
function SUM($A[1 \ldots n]$):
    if $n = 0$ then
        return 0
    else
        return SUM ($A[1 \ldots n-1]$) + $A[n]$
    end
```

described a function in terms of its value on a different input (usually, smaller inputs, but in some rare cases, larger inputs are also used). The runtime of Algorithm 1 can be described by the following recurrence:

$$T(n) = \begin{cases} 0 & \text{if } n = 0 \\ T(n-1) + 1 & \text{if } n > 0 \end{cases}$$

where "+1" term is the cost of taking a sum between 2 integers. However, we can observe that this recurrence does not give us an easy way to decide if $T(n)$ belongs to $\mathcal{O}(g(n))$ for which $g(n)$. One way of solving this would be to continuously substitute $T(n-1)$ with its supposed value according to the recurrence until we see a pattern. However, for more complex recurrence, this is not easy. However, we have Theorem 1 that is useful to determine $\mathcal{O}$-class of recurrence.

Theorem 1 *(Master Theorem) Let $a \geq 1$ and $b > 1$ be constants, let $f(n)$ be a function, and let $T(n)$ be defined by*

$$T(n) = aT(\frac{n}{b}) + f(n),$$

where $f(n) = \mathcal{O}(n^d)$ for some constant $d \geq 0$, then

$$T(n) = \begin{cases} \mathcal{O}(n^d) & \text{if } a < b^d \\ \mathcal{O}(n^d \log n) & \text{if } a = b^d \\ \mathcal{O}(n^{\log_b a}) & \text{if } a > b^d \end{cases}$$

2.4 Strassen [20]'s Matrix Multiplication

This section presents Strassen's recursive algorithm for multiplying matrices which run in $\mathcal{O}(n^{2.81})$ time which outperforms the naive matrix multiplication described in Section 2.1 for sufficiently large n.

The main improvement of this algorithm from the naive one stems from multiplying 2×2 matrices using only 7 multiplication instead of 8 as in the naive algorithm, and this construction only takes $\mathcal{O}(n^2)$ scalar additions. For details on how this construction work, please refer to Cormen et al. [9]. Strassen's algorithm then works as described in Algorithm 2.

The recurrence that described the runtime of this algorithm is

Algorithm 2: Strassen's Matrix Multiplication

```
Data: $A, B \in \mathbb{C}^{n \times n}$
Result: $C = A \cdot B$
function MATMUL($A, B$):
    if $n = 1$ then
        return $A \cdot B$ ;
    else
        $A' \leftarrow$ Subdivision $A$ into 2 × 2 block matrices ;
        $B' \leftarrow$ Subdivision $B$ into 2 × 2 block matrices ;
        $C' \leftarrow A' \cdot B'$ as 2 × 2 matrices with 7 multiplications ;
        (Noted that the entries of $A'$ and $B'$ are matrices, so we recursively call MATMUL where
          applied) ;
        return $C$;
    end
```

$$T(n) = 7T(\frac{n}{2}) + \mathcal{O}(n^2)$$

where we break a $n \times n$ matrix into a 2×2 block matrix, each entry of the block matrices has dimension $\frac{n}{2} \times \frac{n}{2}$. By Theorem 1, we have that $T(n) = \mathcal{O}(n^{\log_2 7}) = \mathcal{O}(n^{2.81})$.

Following Strassen's algorithm, we can observe that if we have a $\mathcal{O}(n^2)$ construction that reduces the number of multiplications made in the 2×2 case, we would get a better matrix multiplication algorithm. More generally, if we can do multiplication of $m \times n$ and $n \times r$ matrices using only $\mathcal{O}(mn + nr + mr)$ scalar additions and less than $m * n * r$ multiplication, we get a better matrix multiplication. Moreover, the less multiplication we used, the better the algorithm gets.
Cohn and Umans [7]'s framework follows this notion to generate a better matrix multiplication algorithm using underlying abstract algebra and representation theory, and, hopefully, generating a $\mathcal{O}(n^2)$ algorithm for some m, n, and r using some relatively small number of multiplications.

2.5 Groups, Rings, and Fields

A **group** $(G, *)$ is a non-empty set of elements G defined under an binary operation $*$ such that the following (also known as the **group axioms**) hold:

- Closure: For all $a, b \in G$, $a * b \in G$.
- Associativity: For all $a, b, c \in G$, $(a * b) * c = a * (b * c)$.
- Identity: There exists $e \in G$ such that for all $a \in G$, $e * a = a * e = a$. This e is called the *identity element* of the group.
- Inverse: For all $a \in G$, there exists $b \in G$ such that $a * b = b * a = e$ where e is the identity element.

A group is called *abelian* if for all $a, b \in G$, $a * b = b * a$. In another term, every element in G *commutes* through $*$ or $*$ is *commutative* in G.

A simple example of a group is $(\mathbb{Z}, +)$ where $\mathbb{Z}$ is the set of all integers:

- $a, b \in \mathbb{Z}$, then $a + b \in \mathbb{Z}$.
- $+$ is associative by basic math.
- 0 is the identity element.
- For $a \in \mathbb{Z}$, $-a \in \mathbb{Z}$ where $a + (-a) = (-a) + a = 0$.

More exotic examples include the Dihedral group of order $2n$ describing the symmetries of an n-gon, the Klein four-group. Another important example is the **symmetric group** of degree n (denoted S_n) which is the set of all permutations of numbers from 1 to n. When $*$ is clear from context, we will refer to the group $(G, *)$ as G.

A **subgroup** H of G is a non-empty subset of group G under the same binary operation $*$ as group G such that H satisfies all the group axioms.

A **ring** R is a set of elements with 2 binary operations $+$ and $*$ (addition and multiplication) such that the following axioms hold:

- $(R, +)$ is an abelian group.
- $*$ is associative.
- Distributive: for all $a, b, c \in R$,

$$(a + b) * c = a * c + b * c \text{ and } c * (a + b) = c * a + c * b$$

The ring R is *commutative* if multiplication is commutative.
The ring R is said to have an *identity* or a ring with *unity* if there exists $1 \in R$ such that for all $a \in R$, $1 * a = a * 1 = a$.
A ring R with unity, where $1 \neq 0$, is called a **division ring** if every non-zero element $a \in R$ has a multiplicative inverse i.e. there exists $b \in a$ such that $a * b = b * a = 1$. A commutative division ring is a **field**.
A basic example of fields (or rings) that is used in this research is $\mathbb{C}$, the set of complex numbers.
Similar to groups, $S \subseteq R$ is a **subring** if S is a ring under the same operations as R.

Two rings R and S are said to be **isomorphic**, denoted $R \cong S$, if there exists an a bijection $\phi : R \to S$ such that for all $a, b \in R$, $\phi(a) +_S \phi(b) = \phi(a +_R b)$ and $\phi(a) *_S \phi(b) = \phi(a *_R b)$. In this case, ϕ is named a **ring isomorphism**. Since fields are also rings, **field isomorphisms** have the same requirement as ring isomorphisms.
The left **ideal** generated by $a \in R$ of a ring R is the set of elements

$$aR = \{a * r \mid r \in R\}.$$

Similarly, we can define the right ideal and the double-sided ideal generated by a respectively as

$$Ra = \{r * a \mid r \in R\} \quad \text{and} \quad aRa = \{a * r * a \mid r \in R\}$$

2.6 Polynomial Rings and Group Algebra

Let R be a ring and x be some indeterminate. The **polynomial ring** $R[x]$ in x over ring R is the set of all formal sums of the form:

$$a_n x^n + a_{n-1} x^{n-1} + a_{n-1} x^{n-1} + \cdots + a_0$$

or, in other notation, $\sum_{i=0}^{n} a_i x^i$ for some $n \geq 0$ and $a_n \neq 0$. In this case, n is also known as the **degree** of the polynomial. The polynomial is called **monic** if $a_n = 1$ and is called **irreducible** if the polynomials can not be factored into multiplicative terms. Addition in $R[x]$ is defined "componentwise" as

$$\sum_{i=0}^{n} a_i x^i + \sum_{i=0}^{n} b_i x^i = \sum_{i=0}^{n} (a_i +_R b_i) x^i,$$

and multiplication is defined the same way as multiplying polynomials as

$$\sum_{i=0}^{n} a_i x^i * \sum_{j=0}^{n} b_j x^j = \sum_{i=0}^{n} \sum_{j=0}^{n} (a_i *_R b_j) x^{i+j}.$$

Since $R[x]$ is named a "polynomial ring", we can already notice that we can prove that $R[x]$ is a ring. The proof is omitted in this report, but we can formulate Theorem 2.

Theorem 2 *Given a ring R and indeterminate x, $R[x]$ is a ring.*

Now, using $R[x]$ as our ring and y as our indeterminate, we can define the polynomial ring $R[x][y]$, also denoted $R[x, y]$, in a similar fashion and get another ring.

Corollary 2.1 *Given a ring R and a finite list of indeterminates x_1, x_2, ..., x_n, $R[x_1, x_2, \ldots, x_n]$ is a ring.*

Following Corollary 2.1, we define $R[G]$, the polynomial ring using elements of a finite group $G = \{g_1, g_2, \ldots, g_n\}$ as the indeterminates. Alternatively, this ring $R[G]$ is called **group algebra** of group G over R. Since G is a group, closure applies i.e. doing multiplication on elements of group G always gives an element of the group. This means elements of $R[G]$ are formal sums of the form:

$$a_1 g_1 + a_2 g_2 + \cdots + a_n g_n$$

where $a_i \in R$. This group algebra $R[G]$ has some special property when R is a field. This will be discuss in Section 2.8.

Since $R[G]$ is also a ring, we can define a **subalgebra** $\mathcal{A}' \subseteq R[G]$ where $\mathcal{A}'$ is a subring of $R[G]$.

2.7 Field Extensions, Defining Polynomials, and Algebraic Closure of Fields

Let $\mathbb{K}$ be a field containing field $\mathbb{F}$, then $\mathbb{K}$ is said to be a **field extension** of $\mathbb{F}$. Alternatively, the field $\mathbb{F}$ is called the **base field** of $\mathbb{K}$.

Let $\alpha \in \mathbb{K}$ is said to be **algebraic** over $\mathbb{F}$ if α is a root of some nonzero polynomial $f(x) \in \mathbb{F}[x]$, the polynomial ring in x over $\mathbb{F}$. If α is not algebraic over $\mathbb{F}$, α is said to be **transcendental** over $\mathbb{F}$. If every element in $\mathbb{K}$ is algebraic over $\mathbb{F}$, $\mathbb{K}$ is an **algebraic extension** of $\mathbb{F}$.

Theorem 3 *Let α be algebraic over $\mathbb{F}$. Then, there exists a monic irreducible polynomial $m_{\alpha,\mathbb{F}}(x) \in \mathbb{F}[x]$ where $m_{\alpha,\mathbb{F}}(\alpha) = 0$.*

This monic irreducible polynomial $m_{\alpha,\mathbb{F}}(x)$ is also known as a **defining polynomial** of α over $\mathbb{F}$. Noted that α is not necessary in $\mathbb{F}$: we can also use $\alpha \in R[G]$ for some group G, and $m_{\alpha,\mathbb{F}}(x)$ is still defined. Observed that $m_{\alpha,\mathbb{F}}(x)$ will not factor over $\mathbb{F}$, but it would factor over $\mathbb{K}$ because $\alpha \in \mathbb{K}$. Here, we present the idea of an **algebraic extension**.

The **algebraic extension** of $\mathbb{F}$ generated by an algebraic element α, denoted $\mathbb{F}[\alpha]$, is a set of elements of the form:

$$f_0 + f_1\alpha + f_1\alpha^2 + \cdots + f_n\alpha^n$$

where $f_i \in \mathbb{F}$. Similarly to multi-indeterminate polynomial rings, we can extend $\mathbb{F}$ by a set of algebraic elements, or maybe, all of $\mathbb{F}$'s algebraic elements, then all polynomial $f(x) \in \mathbb{F}(x)$ will factor into polynomials of degree 1, or in another term, **completely split**. This field is called the **algebraic closure** of $\mathbb{F}$, denoted $\overline{\mathbb{F}}$.

Theorem 4 *Let $\mathbb{F}$ be a field. Then $\overline{\mathbb{F}}$ is **algebraically closed**, meaning every polynomial $f(x) \in \overline{\mathbb{F}}[x]$ has roots in $\overline{\mathbb{F}}$.*

This also means the only algebraic extension of $\overline{\mathbb{F}}$ is $\overline{\mathbb{F}}$ itself i.e. $\overline{\overline{\mathbb{F}}} = \overline{\mathbb{F}}$.

2.8 Representation Theory

Representation theory is a branch of mathematics that studies the abstract algebra structure we mentioned in the previous sections by representing their elements as linear algebra objects (vectors, vector spaces, linear transformation, and matrices). In this research, we rely on the representation theory to pursue our theoretical approach and represent abstract algebra objects in our implementation.

2.8.1 Representation of Group Algebra

Let $G = \{g_1, g_2, \dots, g_n\}$ be a finite group and $R[G]$ be a group algebra of group G over some ring R with unity. Recall from Section 2.6 that elements of a group algebra are of the form:

$$\alpha = a_1g_1 + a_2g_2 + \cdots + a_ng_n$$

$$\beta = b_1g_1 + b_2g_2 + \cdots + b_ng_n$$

where a_i's and b_i's are elements of R. We observed that we can represent α and β as vectors admitting some ordering of G as follows:

$$\vec{\alpha} = \begin{bmatrix} a_1 \\ a_2 \\ \vdots \\ a_n \end{bmatrix} \quad \text{and} \quad \vec{\beta} = \begin{bmatrix} b_1 \\ b_2 \\ \vdots \\ b_n \end{bmatrix}$$

where the entries of the vectors α and $\vec{\alpha}, \vec{\beta} \in R^n$. Indeed, we can also define $+$ and $*$ in R^n such that $\alpha \mapsto \vec{\alpha}$ be an isomorphism. For $+$, the construction is rather simple and intuitive as a pointwise sum

$$\alpha + \beta = (a_1 + b_1)g_1 + (a_2 + b_2)g_2 + \cdots + (a_n + b_n)g_n$$

$$\mapsto \vec{\alpha} + \vec{\beta} = \begin{bmatrix} a_1 + b_1 \\ a_2 + b_2 \\ \vdots \\ a_n + b_n \end{bmatrix}$$

However, it is not as simple for $*$ because the underlying logic from group G:

$$\alpha * \beta = \sum_{i=1}^{n} a_ig_i \sum_{j=1}^{n} b_jg_j = \sum_{i=1}^{n}\sum_{j=1}^{n} (a_i * b_j) * (g_i * g_j)$$

where there is not set value of $(g_i * g_j)$ for all groups G. We observe that to define $*$, we have to somehow encapsulate the underlying logic of group G. Here, we introduce the idea of **structural constants** which gives a discrete representation of group G. For any group G, the **structural constants** are the tensor $C = \mathbb{C}^{n \times n \times n}$ where

$$A(i,j,k) = \begin{cases} 1 & \text{if } g_i * g_j = g_k \\ 0 & \text{otherwise} \end{cases}$$

Noted that this representation is sparse since it is likely that there are more 0 entries than 1 entries. However, we use this representation for a reason. Having the structural constants C, we rewrite the $*$ in $R[G]$ as follows:

$$\begin{aligned}\alpha * \beta &= \sum_{i=1}^{n}\sum_{j=1}^{n}(a_i * b_j) * (g_i * g_j)\\ &= \sum_{i=1}^{n}\sum_{j=1}^{n}(a_i * b_j)\sum_{k=1}^{n} C(i,j,k) * g_k\end{aligned}$$

Moreover, g_k as a group algebra element can be represented as

$$\vec{g_k} = \begin{bmatrix} 0 \\ \vdots \\ 1 \\ \vdots \\ 0 \end{bmatrix}$$

where $\vec{g_k}$ has a single 1 entry at index k and 0 everywhere else. Hence, we have

$$\vec{\alpha} * \vec{\beta} = \sum_{i=1}^{n}\sum_{j=1}^{n}(a_i * b_j)\sum_{k=1}^{n} C(i,j,k)\vec{g_k}$$

Observe that the right-hand side gives a vector in R^n. Hence, from $R[G]$ we got the equivalent representation of R^n under defined $+$ and $*$. Any algebra $\mathcal{A}$ whose elements can be defined this way in a finite vector is called **finite dimensional** over R. Also, by using

$$\vec{g_k} = \begin{bmatrix} 0 \\ \vdots \\ 1 \\ \vdots \\ 0 \end{bmatrix},$$

like with linear algebra, $\{\vec{g_k}\}_{k=[1...n]}$ is called the canonical **basis** of R^n (and by extension, $\{g_k\}_{k=[1...n]}$ is that of $R[G]$). Hence, we can do a **change of basis** to give different bases of R^n (and by extension, $R[G]$). Nevertheless, this change of basis makes our previous structural constants unusable since with C, we are assuming $\{g_k\}_{k=[1...n]}$ being the basis. Another set of structural constants, C' can be constructed in the similar manner, but C' is not guaranteed to contain only 0s and 1s; hence, C and C' are defined to be a 3D tensor.

Furthermore, since $\{g_k\}_{k=[1...n]}$ is a basis of $R[G]$, we say $\dim(R[G]) = n$ or $R[G]$ has **dimension** $|G|$. From this definition, we can observe that $\dim(R[G])$ is independent of the basis. This is also true for the subalgebra.

Since this research only deals with group algebra $R[G]$ of a finite group G with R being a field, every algebra we encounter here is finite-dimensional over a given field.

2.8.2 Wedderburn-Artin Theorem

In this section, we introduce an important result in representation theory, beginning with Wedderburn-Artin Theorem.

Theorem 5 *(Wedderburn-Artin) A **semisimple** group algebra $\mathbb{F}[G]$ of a finite group G can be decomposed into the direct product of full matrix algebras:*

$$\mathbb{F}[G] \cong \mathbb{D}_1^{d_1 \times d_1} \oplus \mathbb{D}_2^{d_2 \times d_2} \oplus \cdots \oplus \mathbb{D}_k^{d_k \times d_k} \tag{1}$$

*where d_i is the order of matrix algebra $\mathbb{D}_i^{d_i \times d_i}$ and $\mathbb{D}_i$ is an algebraic extension of $\mathbb{F}$. k, d_i's and $\mathbb{D}_i$'s are uniquely determined by G and $\mathbb{F}$ up to permutation. The d_i's are also called the **character degrees** of G.*

Here, a **matrix algebra** $\mathbb{D}_i^{d_i \times d_i}$ is a collection of $d_i \times d_i$ matrices with entries in $\mathbb{D}_i$. An element of $\mathbb{Q}^{2\times 2}$ could be:

$$\begin{bmatrix} \frac{1}{2} & 2 \\ 4 & 0 \end{bmatrix}.$$

However, for Theorem 5 to apply, we first need $\mathbb{F}[G]$ to be semisimple. Theorem 6 proves the be helpful here.

Theorem 6 *Let $\mathbb{F}$ be a field and G be a group, then $\mathbb{F}[G]$ is semisimple.*

Now, consider $\mathbb{F}[G]$ to be semisimple, Equation 3 indicates that there exists an isomorphism $\Phi : \mathbb{F}[G] \to \bigoplus_{i=1}^{k} \mathbb{D}_i^{d_i \times d_i}$. Informally, Φ maps a group algebra element to a set of square matrices of determined sizes, and multiplying in the group algebra is the same as multiplying these square matrices pointwise. In the contribution of Wedderburn, Φ is also called the **Wedderburn Decomposition**. Noted that Φ is not unique.

For more elaboration, an example of a decomposition is as follows:

Consider the semisimple group algebra $\mathbb{C}[G]$ with some group G over the complex numbers $\mathbb{C}$ with the following decomposition:

$$\mathbb{C}[G] \cong \mathbb{C}^{2\times 2} \oplus \mathbb{C}^{3\times 3}.$$

Noted that the D_is in the Wedderburn decomposition of $\mathbb{C}[G]$ are just $\mathbb{C}$ since $\mathbb{C}$ is algebraically closed.

For $C \in \mathbb{C}[G]$, we have that

$$\Phi(C) = A \oplus B = \begin{bmatrix} A & 0 \\ 0 & B \end{bmatrix} = \begin{bmatrix} \begin{bmatrix} a_{1,1} & a_{1,2} \\ a_{2,1} & a_{2,2} \end{bmatrix} & 0 \\ 0 & \begin{bmatrix} b_{1,1} & b_{1,2} & b_{1,3} \\ b_{2,1} & b_{2,2} & b_{2,3} \\ b_{3,1} & b_{3,2} & b_{3,3} \end{bmatrix} \end{bmatrix} = \begin{bmatrix} a_{1,1} & a_{1,2} & 0 & 0 & 0 \\ a_{2,1} & a_{2,2} & 0 & 0 & 0 \\ 0 & 0 & b_{1,1} & b_{1,2} & b_{1,3} \\ 0 & 0 & b_{2,1} & b_{2,2} & b_{2,3} \\ 0 & 0 & b_{3,1} & b_{3,2} & b_{3,3} \end{bmatrix}$$

where A and B are 2×2 and 3×3 matrices with entries in $\mathbb{C}$. Similarly, with $C' \in \mathbb{C}[G]$, we have a similar decomposition:

$$\Phi(C') = A' \oplus B' = \begin{bmatrix} A' & 0 \\ 0 & B' \end{bmatrix} = \begin{bmatrix} \begin{bmatrix} a'_{1,1} & a'_{1,2} \\ a'_{2,1} & a'_{2,2} \end{bmatrix} & 0 \\ 0 & \begin{bmatrix} b'_{1,1} & b'_{1,2} & b'_{1,3} \\ b'_{2,1} & b'_{2,2} & b'_{2,3} \\ b'_{3,1} & b'_{3,2} & b'_{3,3} \end{bmatrix} \end{bmatrix} = \begin{bmatrix} a'_{1,1} & a'_{1,2} & 0 & 0 & 0 \\ a'_{2,1} & a'_{2,2} & 0 & 0 & 0 \\ 0 & 0 & b'_{1,1} & b'_{1,2} & b'_{1,3} \\ 0 & 0 & b'_{2,1} & b'_{2,2} & b'_{2,3} \\ 0 & 0 & b'_{3,1} & b'_{3,2} & b'_{3,3} \end{bmatrix}$$

It is the case that $\Phi(C) \cdot \Phi(C') = \Phi(C \cdot C') = \begin{bmatrix} A \cdot A' & 0 \\ 0 & B \cdot B' \end{bmatrix}$ by naively multiplying out the 2 matrices in the decomposition.

Generally, we have that if $\Phi(A) = A_1 \oplus A_2 \oplus \cdots \oplus A_k$ and $\Phi(B) = B_1 \oplus B_2 \oplus \cdots \oplus B_k$ such that A_i and B_i are matrices of dimension $d_i \times d_i$,

$$\Phi(A) \cdot \Phi(B) = \Phi(A \cdot B) = \begin{bmatrix} A_1 \cdot B_1 & 0 & \dots & 0 \\ 0 & A_2 \cdot B_2 & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & A_k \cdot B_k \end{bmatrix}$$

This can be proven by naively multiplying out the matrices. This multiplying in the decomposition provides some speed up, which will be explained in the next subsection. We noted that Φ is an isomorphism (a bijection), so there exists its inverse Φ^{-1}, and we can obtain $A \cdot B$ by applying Φ^{-1} on $\Phi(A \cdot B)$:

$$\Phi^{-1}\left(\begin{bmatrix} A_1 \cdot B_1 & 0 & \dots & 0 \\ 0 & A_2 \cdot B_2 & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & A_k \cdot B_k \end{bmatrix}\right) = A \cdot B$$

As a side note, in case of $\mathbb{C}$, or any algebraically closed field, the equality

$$\dim(\mathbb{C}[G]) = |G| = \dim\left(\bigoplus_{i=1}^{k} \mathbb{C}^{d_i \times d_i}\right) = \sum_{i=1}^{k} d_i^2, \tag{2}$$

holds for any group G. Hence, we have Corollary 6.1.

Corollary 6.1 *Let G be a finite group and $\mathbb{F}$ be algebraically closed. Then, we have*

$$\mathbb{F}[G] \cong \mathbb{F}^{d_1 \times d_1} \oplus \mathbb{F}^{d_2 \times d_2} \oplus \cdots \oplus \mathbb{F}^{d_k \times d_k} \tag{3}$$

and

$$|G| = \sum_{i=1}^{k} d_i^2. \tag{4}$$

The proof for Corollary 6.1 is trivial since the only algebraic field extension of $\mathbb{F}$ is $\mathbb{F}$ given an algebraically closed field $\mathbb{F}$.

3 Cohn and Umans Matrix Multiplication

As Section 2 gives us sufficient math and computer science background to talk about Cohn and Umans matrix multiplication framework, this section first gives an overview of the matrix multiplication algorithm and then goes into the specifics regarding how to perform the steps and how the information in Section 2 is applied.

3.1 Framework Overview

Let A and B be $m \times n$ and $n \times r$ matrices over some algebraically closed field $\mathbb{F}$, and G be a group that satisfies the **triple product property** (TPP) [7] that **realize** $< m', n', r' >$ (enable the multiplication of matrices with scalar entries of size $m' \times n'$ and $n' \times r'$). Let $C = A \cdot B$. We have the following approach:

Step 1: Subdivide A and B into $m' \times n'$ and $n' \times r'$ block matrices, A' and B', respectively. Append missing 0 entries to make m' divide m, n' divide n, and r' divide r. Details regarding subdividing a matrix is provided in Section 2.1.

Step 2: Treating entries of A' and B' as scalars, embed A' and B' into $\mathbb{F}[G]$, denoted $\overline{A}, \overline{B}$, respectively.

Step 3: Compute the Wedderburn Decompositions of $\overline{A}$ and $\overline{B}$, $\Phi(\overline{A})$ and $\Phi(\overline{B})$, respectively. Note that here, the matrices in the Wedderburn Decomposition are of sizes $(d_i \cdot \lceil \frac{m}{m'} \rceil) \times (d_i \cdot \lceil \frac{n}{n'} \rceil)$ and $(d_i \cdot \lceil \frac{n}{n'} \rceil) \times (d_i \cdot \lceil \frac{r}{r'} \rceil)$, so matrix multiplication holds. Regarding Φ, please refer to Section 2.8.2.

Step 4: Compute $\Phi(\overline{A}) \cdot \Phi(\overline{B}) = \Phi(\overline{A} \cdot \overline{B})$ as point-wise product as described in Section 2.8. Notice that each product is that of 2 matrices, so we can send them back to Step 1 or do the naive matrix multiplication, whichever is more efficient.

Step 5: Compute $\Phi^{-1}(\Phi(\overline{A} \cdot \overline{B})) = \overline{A} \cdot \overline{B} = \overline{F} \in \mathbb{C}[G]$.

Step 6: Unembed $\overline{C}$ to its respective block matrix C' of size $m' \times r'$.

Step 7: Assemble C' and crop into size $n \times r$ to obtain C.

Steps 4 and 5 hold since the mapping in Step 1 [7] and Φ are invertible. The improvement of the run-time comes from doing the point-wise product in Step 4 as described in Section 2.8. Referring back to Algorithm 2 where doing less multiplication gives us efficiency, this algorithm uses k multiplications at Step 4. If $k < m'n'r'$, we gain efficiency. Put simply, the smaller k is, the better the algorithm. Therefore, this framework opens up the research line of finding group G with triple product property realizing $< m', n', r' >$ where k is small enough to give $\mathcal{O}(n^{2+o(1)})$ matrix multiplication algorithm.

However, before multiplying matrices using this framework, we observed that this framework first requires group G satisfying TPP and Wedderburn Decomposition Φ of $\mathbb{F}[G]$. For group G, in 2005, Cohn et al. [6] gives the construction for mathematical objects called **Strong Unique Solvable Puzzles** (SUSPs) which generates the necessary group G. The constructions for finding, verifying, and generating finite groups G from SUSPs have been implemented by Anderson et al. [2]. The computation and structure of Φ are the rigor of this research and are described in Section 3.4.

3.2 Embeddings of Matrices into the Group Algebra

A finite group G satisfying TPP also comes with subgroups T, U, and V that realize $< |T|, |U|, |V| >$. Let $|T| = m$, $|U| = n$, and $|V| = r$, and A and B be $m \times n$ and $n \times r$ matrices, respectively. Since G is finite, T,

U, and V are finite, we gives them arbitrary orderings

$$T = \{t_1, t_2, \ldots, t_m\}$$
$$U = \{u_1, u_2, \ldots, u_n\}$$
$$V = \{v_1, v_2, \ldots, v_r\}$$

be some orderings of elements of T, U, and V. Then, the embedding of A and B into group algebra $\mathbb{F}[G]$ is as follows:

$$\overline{A} = \sum_{i=1}^{m} \sum_{j=1}^{n} A_{i,j}(t_i^{-1}u_j) \quad \text{and} \quad \overline{B} = \sum_{j=1}^{n} \sum_{k=1}^{r} B_{j,k}(u_j^{-1}v_k)$$

It is a fact that

$$\overline{C} = \overline{A} \cdot \overline{B} = \sum_{i=1}^{m} \sum_{k=1}^{r} C_{i,k}(t_i^{-1}v_k).$$

So to unembed $\overline{C}$ into matrix C, we map the coefficients of $\overline{C}$ to the right entries. This construction is rather simple, and Anderson et al. [2] has provided a working implementation. Noted that doing multiplication in the group algebra and unemebed C gives the correct product of A and B. However, doing this way does not give improvement since taking multiplication in the algebra takes $\mathcal{O}(|G|^2)$ multiplication, naively, and, $\mathcal{O}(|G| \log |G|)$ using a Fourier Transform. Since $|G| \geq mnr$, there is no improvement in multiplying matrices in the embedding.

3.3 Strong Unique Solvable Puzzles (SUSPs) and Simplifiable SUSPs

A **Strong Unique Solvable Puzzle (SUSP)** of *width* q is a subset $U \subseteq \{1, 2, 3\}^q$ such that or all permutation $\pi_1, \pi_2, \pi_3 \in Sym(U)$ (the group of permutations of elements of U, isomorphic to $S_{|U|}$ ($|U| =$ # of elements of U) by admitting an ordering for elements of U), either $\pi_1 = \pi_2 = \pi_3$ or there exists $u \in U$ and $i \in [1 \ldots k]$ such that exactly 2 of the following holds: $(\pi_1(u))_i = 1$, $(\pi_2(u))_i = 2$, and $(\pi_3(u))_i = 3$. The *size* $s = |U|$ of a puzzle is the number of elements in U.

For brevity, going forward, this report refers to a SUSP as a *puzzle* and the puzzle of size s and width q as a $s \times q$ puzzle. The implementation for this generation has been done in detail by Anderson et al. [2]. Specifically, group G generated by a puzzle P of width q and size s along with an integer $p \geq 3$ realizing

$< m', n', r' >$ where

$$|G| = s! \cdot p^{s \cdot q},$$

$$m' = s! \cdot (p-1)^{n_1},$$

$$n' = s! \cdot (p-1)^{n_2},$$

$$r' = s! \cdot (p-1)^{n_3},$$

n_1, n_2, n_3 are the numbers of 1, 2, and 3 in puzzle P, respectively.

It is also worth noting that for any puzzle P, we want to determine the value of p such that the matrix multiplication algorithm generated a minimum ω for that puzzle. To determine ω given the group G generated by puzzle P, Cohn et al. [6] gave Theorem 7, Corollary 7.1, and Corollary 7.2 [6, Theorem 1.8, Corollary 1.9, and Corollary 3.6].

Theorem 7 *Suppose G realizes $< m, n, r >$ and the character degrees of G are $\{d_i\}$. Then, $(mnr)^{\omega/3} \leq \sum_i d_i^{\omega}$.*

Corollary 7.1 *Suppose G realizes $< m, n, r >$ and has largest character degree d, then $(mnr)^{\omega/3} \leq d^{\omega-2}|G|$.*

Corollary 7.2 *If P is a $s \times q$ puzzle and $p \geq 3$ is an integer, then*

$$\omega \leq \frac{3 \log p}{\log(p-1)} - \frac{3 \log s!}{sq \log(p-1)}.$$

A simple example of a SUSP is the following 2×2 puzzle P:

1	3
2	1

By Theorem 7 and Corollary 7.1, puzzle P yields $\omega \leq 2.919992$ and $\omega \leq 2.917891$, respectively at $p = 9$. However, finding and verifying SUSPs are hard tasks; it remains an open question if the latter is coNP-complete.

This introduces us to a subset of SUSPs called **simplifiable SUSPs**. This first appeared in Anderson and Le [3]. The details on this object are not the focus of this project; however, it gives useful results that allow us to simplify the asymptotic analysis.

Theorem 8 *[3, Lemma 9] The ω's can be achieved by SUSPs can be achieved by SUSPs that are simplifiable.*

Theorem 9 *If there is a simplifiable $s \times q$ puzzle, then there exists a balanced simplifiable puzzle of size $s^3 \times 3q$.*

Theorem 8 means that we can get all ω's possible using only the simplifiable SUSPs, and Theorem 9 gives that for analysis, we may assume that we always have a **balanced** SUSP where $n_1 = n_2 = n_3$, implying $m' = n' = r'$. Theorem 9 came up in a discussion with Anderson about [3]. For brevity, this report refers to simplifiable SUSPs as simplifiable puzzles.

3.4 Computation of Wedderburn Decomposition and Matrix Explicit Isomorphism Problem

Firstly, Peirce Decomposition gives existence of **idempotents** $e_1, e_2, \ldots, e_k \in \mathcal{A}$ such that

$$\mathcal{A} \cong e_1\mathcal{A}e_1 \oplus e_2\mathcal{A}e_2 \oplus \cdots \oplus e_k\mathcal{A}e_k$$

where each $e_i\mathcal{A}e_i \cong \mathbb{F}^{d_i \times d_i}$ and is a finite dimensional subalgebra of $\mathcal{A}$ with identity e_i. The procedure to find these idempotents is not the rigor of this research since there has been works into finding them [10, 12]. For brevity, the process of finding these idempotents is referred as **splitting the algebra**. The natural isomorphism that map A to $\bigoplus_{i=1}^{k} e_i\mathcal{A}e_i$ is

$$f(a) = e_1ae_1 \oplus e_2ae_2 \oplus \cdots \oplus e_kae_k.$$

So, our last step is to find the isomorphism

$$\phi_i : e_i\mathcal{A}e_i \to \mathbb{F}^{d_i \times d_i}$$

knowing $\mathcal{A}$, e_i and d_i. The problem in this last step is also known as the **matrix explicit isomorphism problem**. We have been able to solve this problem and give the Wedderburn Decomposition Φ.

Firstly, we reform the **matrix explicit isomorphism problem** as follows: Given a finite-dimensional algebra $\mathcal{A}$ over an algebraically closed field $\mathbb{F}$ and $d \in \mathbb{N}$ such that $\mathcal{A} \cong \mathbb{F}^{d \times d}$. Find an isomorphism $\phi : \mathcal{A} \to \mathbb{F}^{d \times d}$. Ivanyos et al. [19] gives that we can further reduce the problem into finding an element $c \in \mathcal{A}$ such that $rank(\phi(c)) = 1$. Ivanyos et al. [19] also shows that the rank of an element is actually independent of the isomorphism, so, for brevity, we will use $rank(c)$ instead of $rank(\phi(c))$.

3.4.1 Rank 1 Element Admit An Isomorphism

Let $c \in \mathcal{A}$ such that $rank(c) = 1$ and $\mathcal{A}$ is a finite dimensional algebra with basis $\{a_1, a_2, \ldots, a_{d^2}\}$. The construction uses the following chain of isomorphisms

$$\mathcal{A} \cong \mathrm{End}(\mathcal{A}c, \mathcal{A}c) \cong (\mathbb{F}^{d\times d})^{op} \cong \mathbb{F}^{d\times d}$$

where $\mathrm{End}(\mathcal{A}c, \mathcal{A}c)$ is the set of all linear transformations from $\mathcal{A}c$ to $\mathcal{A}c$ and $(\mathbb{F}^{d\times d})^{op}$ is the matrix algebra but with the matrix multiplication reverse i.e. $A \cdot_{op} B = B \cdot A$.
Omitting all of the proofs and the exact isomorphism, Algorithm 3 described the process of generating the isomorphism ϕ.

Algorithm 3: Generate Isomorphism from Rank 1 Element

```
Data: $\mathcal{A} \cong \mathbb{F}^{d\times d}$ as basis $\{a_1, a_2, \ldots, a_{d^2}\}$
Data: $c \in \mathcal{A}$ where $rank(c) = 1$
Result: An Isomorphism from $\mathcal{A}$ to $\mathbb{F}^{d\times d}$
1 Derived basis $\{b_1, b_2, \ldots, b_d\}$ of $\mathcal{A}c$. for $i \in [1 \ldots d^2]$ do
2     $A_i \leftarrow$ Linear transformation matrix induced by $a_i$ on $\mathcal{A}e$ using basis $\{b_1, b_2, \ldots, b_d\}$;
3     $A_i \leftarrow A_i^T$
4 end
5 return $\phi(a_i) = A_i$ for all $i \in [1 \ldots d^2]$.
```

The proof for Algorithm 3 can be constructed with the proof of Theorem 5 with some slight adjustments knowing $\mathcal{A}c$ is a d-dimensional vector space over $\mathbb{F}$. For constructive proof of Theorem 5, refer to Drozd and Kirichenko [11]. This result is summarized in Theorem 10.
Note that it is sufficient to return $a_i \mapsto A_i$ for all i since $\{a_1, a_2, \ldots, a_n\}$ is a basis for $\mathcal{A}$, implying $\{A_i\}$ is a basis for $\mathbb{F}^{d\times d}$. We can map elements from $\mathcal{A}$ to $\mathbb{F}^{d\times d}$ by solving a linear system.

Theorem 10 *Given algebra $\mathcal{A} \cong \mathbb{F}^{d\times d}$, Algorithm 3 solve the **matrix explicit isomorphism problem** in polynomial time in d given a rank 1 element c.*

3.4.2 Finding A Rank 1 Element

So now we need a way to find the rank 1 element to construct the isomorphism. Eberly [16] gives a randomized polynomial-time algorithm with an oracle for polynomial factoring to achieve this task. This task has been derandomized by Graaf [17]. However, since the density of the failure case is at 0 for the infinite field and all algebraically-closed field is infinite, for the sake of simplicity, we decided to go with the randomized algorithm instead. To see how this probability is calculated, refer to Eberly [15, Lemma 2.1].

Noted that Algorithm 4 is a **randomized** algorithm which means it always returns a correct answer and reports failure in a low probability. Due to the small chance of failure, re-running the algorithm until it is successful is sufficient. At Line 2, an oracle for factoring polynomials over $\mathbb{F}$ is used. In our implementation, we outsourced this operation to SageMath. Here, we also used a subroutine provided by Dubinsky [12] to find the defining polynomial $f(x)$ of α. The proof for why this algorithm gives a rank 1 element is in Eberly [14].

Algorithm 4: Find Rank 1 Element

```
Data: $\mathcal{A} \cong \mathbb{F}^{d\times d}$ as basis $\{a_1, a_2, \dots, a_{d^2}\}$
Result: $c \in \mathcal{A}$ where $rank(c) = 1$
1  $\alpha \leftarrow$ random element of $\mathcal{A}$ ;
2  $f(x) = (x-\theta_1)(x-\theta_2)\dots(x-\theta_k) \leftarrow$ defining polynomial of $\alpha$ over $\mathbb{F}$ (refer to Section 2.7) ;
3  if $k \neq d$ or $f(x)$ has repeated roots then
4      return FAIL
5  end
6  $\theta_i \leftarrow$ a random root of $f(x)$ ;
7  $h(x) \leftarrow (x-\theta_1)\dots(x-\theta_{i-1})(x-\theta_{i+1})\dots(x-\theta_k)$ i.e. remove $(x-\theta_i)$ factor from $f(x)$;
8  $r \leftarrow$ remainder of $h(x)$ divided by $x-\theta_i$ ($r \in \mathbb{F}$) ;
9  $c \leftarrow h(\alpha)r^{-1}$;
10 return $c$
```

3.5 Wedderburn Decomposition as a Linear Transformation

Now, as we have solved the explicit isomorphism problem, the last thing to do is assemble the Wedderburn Decomposition Φ of group algebra $\mathcal{A} = \mathbb{F}[G]$ of group $G = \{g_1, g_2, \dots, g_n\}$ over an algebraically closed field $\mathbb{F}$, mapping from

$$\mathcal{A} \mapsto e_1\mathcal{A}e_1 \oplus e_2\mathcal{A}e_2 \oplus \dots \oplus e_k\mathcal{A}e_k \mapsto \mathbb{F}^{d_1\times d_1} \oplus \mathbb{F}^{d_2\times d_2} \oplus \dots \oplus \mathbb{F}^{d_k\times d_k}$$

Firstly, from Section 3.4, we have a natural isomorphism $f : \mathcal{A} \to e_1\mathcal{A}e_1 \oplus e_2\mathcal{A}e_2 \oplus \dots \oplus e_k\mathcal{A}e_k$

$$f : \alpha \mapsto e_1\alpha e_1 \oplus e_2\alpha e_2 \oplus \dots \oplus e_k\alpha e_k,$$

and its inverse f^{-1} can be constructed as follows:

$$f^{-1} : e_1\alpha e_1 \oplus e_2\alpha e_2 \oplus \dots \oplus e_k\alpha e_k \mapsto e_1\alpha e_1 + e_2\alpha e_2 + \dots + e_k\alpha e_k = \alpha$$

Then, for any $i \in [0 \dots k]$, we can construct a basis $\{b_{i,1}, b_{i,2}, \dots, b_{i,d_i^2}\}$ by making $\{e_ig_1e_i, e_ig_2e_i, \dots, e_ig_ke_i\}$ linearly independent, and Algorithms 3 and 4 give the isomorphism $\phi : e_i\mathcal{A}e_i \to \mathbb{F}^{d_i\times d_i}$ which maps $e_i\alpha e_i$

to its corresponding matrix in $\mathbb{F}^{d_i \times d_i}$. Furthermore, mapping a matrix A_i in $\mathbb{F}^{d_i \times d_i}$ can be done by solving for the α_i's in the linear system:

$$A_i = \alpha_1 \phi_i(b_{i,1}) + \alpha_2 \phi_i(b_{i,2}) + \cdots + \alpha_{d_i^2} \phi_i(b_{i,d_i^2}) \tag{5}$$

Indeed, since ϕ is an isomorphism, we have

$$\begin{aligned} A_i &= \alpha_1 \phi_i(b_{i,1}) + \alpha_2 \phi_i(b_{i,2}) + \cdots + \alpha_{d_i^2} \phi_i(b_{i,d_i^2}) \\ &= \phi_i(\alpha_1 b_{i,1}) + \phi_i(\alpha_2 b_{i,2}) + \cdots + \phi_i(\alpha_{d_i^2} b_{i,d_i^2}) \\ &= \phi_i(\alpha_1 b_{i,1} + \alpha_2 b_{i,2} + \cdots + \alpha_{d_i^2} b_{i,d_i^2}) = \phi_i(e_i \alpha e_i) \text{ for some } \alpha \in \mathcal{A}. \end{aligned}$$

However, we can see this representation of the Wedderburn Decomposition is rather complex with several complex mathematical objects and operations as functions and solving a linear system. Hence, we want to aim for a less complex representation. Bremner [4] gives such representation where Φ is represented as a **linear transformation**. This involves finding element $e^i_{j,k} \in e_i \mathcal{A} e_i$, where $\phi(e^i_{j,k}) = E_{j,k} \in \mathbb{F}^{d_i \times d_i}$ is a matrix with all zero except for the 1 entry at j^{th} row and the k^{th} column, for all possible j and k. We note that this can be done by solving d_i^2 linear system in the form of Equation 5. This means with the same i, mirroring the matrix units, the $E^i_{j,k}$'s satisfy the **matrix unit relations**:

$$E^i_{j,k} \cdot E^i_{l,m} = \delta_{k,l} E^i_{j,m}$$

where

$$\delta_{k,l} = \begin{cases} 1 & \text{if } k = l \\ 0 & \text{otherwise} \end{cases}.$$

Then, the inverse of the Wedderburn Decomposition Φ^{-1} can be described as a linear transformation with the following matrix, by "listing" the $E^i_{j,k}$:

$$M^{-1} = \begin{bmatrix} | & | & & | & & | & & | \\ \overrightarrow{E^1_{1,1}} & \overrightarrow{E^1_{1,2}} & \ldots & \overrightarrow{E^1_{2,1}} & \ldots & \overrightarrow{E^1_{d_1,d_1}} & \ldots & \overrightarrow{E^k_{d_k,d_k}} \\ | & | & & | & & | & & | \end{bmatrix} \tag{6}$$

where $\overrightarrow{E^i_{j,k}}$ is the vector form of $E^i_{j,k}$ with respective to the canonical basis of $\mathbb{F}[G]$ being $\{g_1, g_2, \dots, g_{|G|}\}$, the elements of group G. Observe that by Corollary 6.1, M^{-1} has $|G| = \sum_{i=1}^{k} d_i^2$ columns. Moreover, since $\mathbb{F}[G]$ has dimension $|G|$, M^{-1} is a $|G| \times |G|$ matrix. It is also a fact that M^{-1} is invertible. Hence, we can obtain the linear transformation matrix M of Φ by taking the inverse of M^{-1} i.e. $M = (M^{-1})^{-1}$.
Then to apply the Wedderburn Decomposition on a group algebra element α, we do the following

$$\overline{\Phi(\alpha)} = M \cdot \vec{\alpha}$$

where $\vec{\alpha}$ is the vector representation of α in respective to the canonical basis and $\overline{\Phi(\alpha)}$ is the flatten form of the set of matrices $\Phi(\alpha)$. To obtain $\Phi(\alpha)$, we unflatten $\overline{\Phi(\alpha)}$ by writing its entries into the set of matrices in the same order as we do for M^{-1} (see Equation 6). By the reverse process, we applied the inverse of the Wedderburn by first flattening the set of matrices, then taking the product M^{-1} with the flattened vector:

$$\alpha = M^{-1} \cdot \overline{\Phi(\alpha)}.$$

Using this representation, we only have to keep track of M and M^{-1} to apply/inverse the Wedderburn Decomposition and the $E^i_{j,k}$'s up to its order used to construct M^{-1} to unflatten $\overline{\Phi(\alpha)}$.

4 Implementation

This section demonstrates how the aforementioned pseudo-code algorithms are translated into a functional codebase.

4.1 Computation System

Since the final implementation of this research relies on some of the implemented subroutines provided by Dubinsky [12], the implementation is done in the same computation system, SageMath. SageMath is an open-source computational system built upon Python that unifies various mathematical software packages into one interface. It provides a comprehensive environment for mathematical computing, covering a wide range of areas including but not limited to abstract algebra, representation theory, calculus, number theory, combinatorics, and graph theory. Since SageMath provides the needed data structures and some of the algorithms, SageMath proves to be most ideal. For this research, we use SageMath 10.0.

4.2 Choice of Fields

Firstly, to apply our framework, we must first choose an algebraically closed field to base our implementation in. Noted that Cohn and Umans [7]'s original framework was based in the field of complex numbers $\mathbb{C}$, which is an algebraically-closed field. However, by first using $\mathbb{C}$, we ran into the problem of numerical instability where we can not represent transcendental elements (i.e. π and e) using a finite number of bits. This caused us to rely on taking an approximation in floating point instead. Since our algorithm requires some error-sensitive operations like taking the reduced row echelon form (RREF) of a matrix, using $\mathbb{C}$ proved to be not effective and caused a huge deviation in our computation of the Wedderburn Computation. Here, SageMath provides us with a solution: $\overline{\mathbb{Q}}$, the fields of algebraic numbers. By Theorem 4, $\overline{\mathbb{Q}}$ is algebraically closed, so our framework and algorithms still apply. Since SageMath's $\overline{\mathbb{Q}}$ provides exact computation, the deviation coming from error tolerance goes away. We theorize that our program would for algebraically-closed fields given some adjustments in the splitting procedure used.

4.3 Algebra Splitting

Our implementation uses the splitting procedure provided by SageMath from [10]. However, this procedure provided by SageMath only works for $\overline{\mathbb{Q}}$. Dubinsky [12] implemented another algorithm that would theoretically work for any field. However, there needs to be some adjustments to his implementation to make it efficient since it involves creating a n^3 tensor described in Section 2.8.1 where n is the size of the group G. For larger groups, we do not have enough RAM or time to create such tensor. It is also outside the scope of this research to improve his implementation.

4.4 Wedderburn Decomposition Structure

We represent the Wedderburn Decomposition as described in Section 3.5. In our implementation, it is sufficient to keep track of the d_i's up to an ordering since we decide to construct M^{-1} by placing the elements on the same row next to each other with increasing column indexes, and rows of the same matrices next to each other in increasing column index (first d_1^2 entries are entries of the first matrix, and so on). Since this representation only requires the implementation to keep track of 2 matrices (M and M^{-1}), and an ordering of the d_i's as a list, and all of these data structures are provided within SageMath along with the required operations, our efforts to implement Cohn-Umans matrix multiplication is greatly simplified.

4.5 $\overline{\mathbb{Q}}$ Runtime Issue

Nevertheless, since $\overline{\mathbb{Q}}$ keeps track of the algebraic components (non-rational parts) ($i.e.\sqrt{2}$), the complexity of the symbolic expressions increase as more computations are performed. More details on how $\overline{\mathbb{Q}}$ works are provided in [5]. This causes serious runtime issues in the computation of Wedderburn Decomposition especially in parts where equal comparisons of elements in $\overline{\mathbb{Q}}$ are used. Specifically, due to the complexity of M^{-1}, the matrix of a linear transformation of Φ^{-1}, in $\overline{\mathbb{Q}}$, taking the there inverse of M^{-1} to obtain M is not efficient where the methods used in this case all required some form of equal comparison. To solve the runtime issues caused by $\overline{\mathbb{Q}}$, we decided to take the approximation of the computation where the exact calculation is not required. Here, matrix M^{-1} was approximated into a floating point number field (with complex components) with some precision, then we take the inverse of the approximated M^{-1} to obtain the approximated M. Then, the remaining of the matrix multiplication framework uses this approximation of M and M^{-1} to obtain an approximation of the matrix product.

4.6 Approximation and Precision of Wedderburn Decomposition

However, by taking the approximation, we run into another problem is that in the case where the matrix M^{-1} is ill-conditioned where taking the inverse with standard precision of 64 bits is not sufficient. Since $\overline{\mathbb{Q}}$ still provide us with the exact M^{-1}, we solve this problem by taking a better approximation of M^{-1}, and then take the inverse of M^{-1}. For the matrices M^{-1} we computed, using 256 bits of precision is sufficient. This number could be smaller, but there is not enough time in this research to figure out the global minima. However, ultimately, the precision depends on the condition number of M^{-1}. Since the runtime is still acceptable in the scope of a computation task, we decided to devote more time to the runtime and error analysis of the matrix multiplication function.

4.7 Error Propagation of Recursive Matrix Multiplication

Then, since there are errors in the Wedderburn Decomposition Φ and its inverse Φ^{-1}, continuous application of Φ and Φ^{-1} propagates errors as the recursion depth increases. We solve this by running an experiment to see what precision gives us the best approximation. Given group G realize $< m, n, r >$ The setup of the experiment is as follows:

1. Increase the precision range from 32 to 256 bits with a 32-bit increment.
2. Measure the distance between the result of the recursive algorithm and the actual result using the Euclidean norm for matrices of different sizes.

3. The sizes of the matrix used are of size $m^s \times n^s$ and $n^s \times r^s$ for easy determination of the recursion depth (just s in this case)

We then proceed to plot these data to see how much the error changes with increments in precision and recursion depth. Noted that since numbers coming from the data is really large, we rely on log-plotting for a viewable figure. From Figure 2, we can see that the algorithm only gives tolerable results (below 64-bit standard) starting from 192-bit precision, which is way higher than the standard 64-bit precision, and the errors stack up rather quickly: increasing recursion depth 2 to 3 causes the error to be $\approx 10^{20}$ times larger. We hope to improve this but for the scope of this project, the algorithm is set to run in 256-bit precision for tolerable results. The raw data used in Figure 2 is in Table A1a.

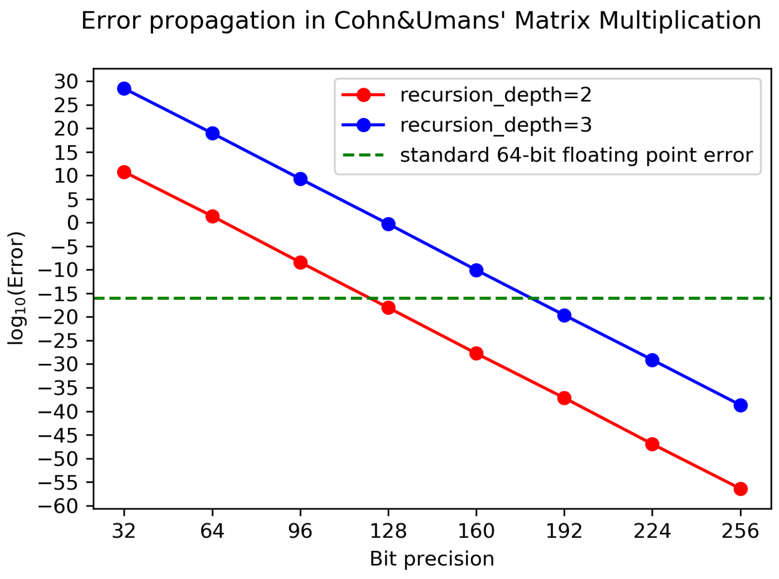

Figure 2: Error propagation of Cohn & Umans's matrix multiplication

5 Runtime Analysis

This section goes into analyzing the runtime of the algorithm and compares them against the runtime of that Cohn et al. [6] gave. We have 2 objectives in this section:

- To experimentally measure the runtime of our implementation.
- To determine for the threshold for the size of matrices where Cohn-Umans matrix multiplication is faster than the naive one
- To verify Cohn & Umans' runtime analysis.

5.1 Experimental Runtime

We run an experiment to determine the actual runtime of the implementation. This reveals the inefficiency of our implementation and by extension Cohn-Umans' matrix multiplication. The experiment setup is as follows:

- We use the puzzle
$$P = \begin{array}{|c|c|} \hline 1 & 3 \\ \hline 2 & 1 \\ \hline \end{array},$$
and $p = 3$. By Theorem 7, this gives $\omega \leq 3$ which will not demonstrate fast multiplication, but this is the smallest group we have that enables matrix multiplication (which would mean that for small matrices, this group would yield the most reasonable runtime).

- We set the base case to be when one of the sizes of the matrices is smaller than the largest character degree of the generated groups. In this case, $\max\{d_i\} = 2$. This makes it possible for the algorithm to go through the recursion instead of direct naive matrix multiplication.

- We multiply matrices of size $m^k \times n^k$ and $n^k \times r^k$ where the generated group realizes $< m, n, r >$. In this case, $m = 8$, $n = 4$, and $r = 4$. For each k, we multiply 4 pairs of matrices.

- For the sake of saving some times, we will only use 64-bit precision.

Table 1 shows our results. The raw data is in Appendix A1b.

	Runtime (s)
$k = 2$	629.6519 ± 4.1718
$k = 3$	$62054.3111 \pm 15492.9139$

Table 1: Experimental Runtime of Cohn-Umans' matrix multiplication

Using a profiler, we identified the significant parts that contribute to the inefficiency. One of which is applying the Wedderburn Decomposition and its inverse to our group algebra elements and block matrices. This is due to the size of the group being proportional to the size of Wedderburn Decomposition.
Another factor is the large branching factor of the algorithm being the number of character degrees of the group (54 in this case) makes it highly inefficient to multiply small matrices. This shows using a larger puzzle for better ω would just lead to the algorithm being even more inefficient since both the cost of applying Wedderburn Decomposition and the branching factor grows. This leads us to approach analyzing its runtime through analysis.

However, this slow runtime could be due to our implementation being inefficient in certain parts. Some of which could be

- The size of the Wedderburn Decomposition is the size of the group squared.
- Applying and inversing the Wedderburn Decomposition to the group algebra elements is inefficient since there are parts of the groups that are not used in the (un)embeddings.
- The cost of copying and cropping the submatrices.

This means there is more work needed in optimizing the implementation for better experimental runtime.

5.2 Asymptotic Analysis

We first attempted to solve the algorithm's recurrence to derive the exponent ω.

5.2.1 Algorithm's Recurrence

By following the steps of the recursive algorithm, given a group G realizing $< m', n', r' >$ with character degrees d_i's ($i = [1 \dots k]$), the recurrence of this recursive algorithm multiplying $m \times n$ and $n \times r$ matrices is as follows:

$$\begin{aligned} T(m,n,r) &= (mn + nr + mr) && \text{(subdividing and re-assembling the matrices)} \\ &+ |G|(m'n' + n'r' + m'r') && \text{(embed and unemebed the block matrices)} \\ &+ |G|^2(mn + nr + mr) && \text{(applying the Wedderburn Decomposition and its inverse)} \\ &+ \sum_{i=1}^{k} T(\frac{m}{m'} \cdot d_i, \frac{n}{n'} \cdot d_i, \frac{r}{r'} \cdot d_i) && \text{(recursive calls)} \end{aligned}$$

Since group G, d_i's, k, m', n', and r' are constants, let

$$\begin{aligned} \alpha &= (|G|^2 + 1) \\ \beta &= |G|(m'n' + n'r' + m'r') \\ f(m,n,r) &= \alpha(mn + nr + mr) + \beta \end{aligned}$$

we have a more simplified version of the recurrence

$$T(m,n,r) = f(m,n,r) + \sum_{i=1}^{k} T(\frac{m}{m'} \cdot d_i, \frac{n}{n'} \cdot d_i, \frac{r}{r'} \cdot d_i) \tag{7}$$

Then, the recurrence also needs a base case. For the case of this algorithm, the base case is unknown and is what this section aims to find. Hence, we let

$$T(m,n,r) = mnr, \quad \text{if } m \leq m_0 \text{ and } n \leq n_0 \text{ and } r \leq r_0, \tag{8}$$

where naive matrix multiplication is matrices smaller than $m_0 \times n_0$ and $n_0 \times r_0$ are more efficient than using Cohn and Umans algorithm. In research, we also aim to find m_0, n_0, and r_0 for group G.

5.2.2 Recurrence's Solution

We first solve Equation 7 using back substitution

$$\begin{aligned}
T(m,n,r) &= f(m,n,r) + \sum_{i=1}^{k} T(\frac{m}{m'} \cdot d_i, \frac{n}{n'} \cdot d_i, \frac{r}{r'} \cdot d_i) \\
&= f(m,n,r) + \sum_{i=1}^{k} \left[f(\frac{m}{m'} \cdot d_i, \frac{n}{n'} \cdot d_i, \frac{r}{r'} \cdot d_i) + \sum_{j=1}^{k} T\big(\frac{m}{(m')^2} \cdot d_i \cdot d_j, \frac{n}{(n')^2} \cdot d_i \cdot d_j, \frac{r}{(r')^2} \cdot d_i \cdot d_j\big) \right] \\
&= f(m,n,r) + \sum_{i=1}^{k} f(\frac{m}{m'} \cdot d_i, \frac{n}{n'} \cdot d_i, \frac{r}{r'} \cdot d_i) + \sum_{i=1}^{k}\sum_{j=1}^{k} T\big(\frac{m}{(m')^2} \cdot d_i \cdot d_j, \frac{n}{(n')^2} \cdot d_i \cdot d_j, \frac{r}{(r')^2} \cdot d_i \cdot d_j\big) \\
&= f(m,n,r) + \sum_{i=1}^{k} f(\frac{m}{m'} \cdot d_i, \frac{n}{n'} \cdot d_i, \frac{r}{r'} \cdot d_i) + \sum_{i=1}^{k}\sum_{j=1}^{k} f(\frac{m}{(m')^2} \cdot d_i \cdot d_j, \frac{n}{(n')^2} \cdot d_i \cdot d_j, \frac{r}{(r')^2} \cdot d_i \cdot d_j) \\
&\qquad + \sum_{i=1}^{k}\sum_{j=1}^{k}\sum_{l=1}^{k} T\big(\frac{m}{(m')^3} \cdot d_i \cdot d_j \cdot d_l, \frac{n}{(n')^3} \cdot d_i \cdot d_j \cdot d_l, \frac{r}{(r')^3} \cdot d_i \cdot d_j \cdot d_l\big) \\
&= f(m,n,r) + \sum_{i=1}^{k} f(\frac{m}{m'} \cdot d_i, \frac{n}{n'} \cdot d_i, \frac{r}{r'} \cdot d_i) + \sum_{i=1}^{k}\sum_{j=1}^{k} f(\frac{m}{(m')^2} \cdot d_i \cdot d_j, \frac{n}{(n')^2} \cdot d_i \cdot d_j, \frac{r}{(r')^2} \cdot d_i \cdot d_j) \\
&\qquad + \sum_{i=1}^{k}\sum_{j=1}^{k}\sum_{l=1}^{k} f(\frac{m}{(m')^3} \cdot d_i \cdot d_j \cdot d_l, \frac{n}{(n')^3} \cdot d_i \cdot d_j \cdot d_l, \frac{r}{(r')^3} \cdot d_i \cdot d_j \cdot d_l) + \dots
\end{aligned}$$

Since we care more about squares matrices, we evaluate $T(n,n,n)$. For brevity, we denote $T(n) = T(n,n,n)$. Observe that if we were to solve $T(n)$ with arbitrary m', n', and r', it is indeterminate when we will reach the base case. However, Theorem 9 gives us that we can assume $m' = n' = r'$. Because we assume the matrices are square and n will be divided by the same n', we can also assume $m_0 = n_0 = r_0$.

Then, we want to determine the maximum number of recursion depth of this algorithm before the naive matrix multiplication, δ. Firstly, let $d = \max\{d_i\}$. We observe at recursion depth δ, the biggest matrix generated would be of size

$$\frac{n}{(n')^\delta} \cdot d^\delta = n \cdot (\frac{d}{n'})^\delta \leq n_0.$$

This means

$$\delta \geq \log_{n'/d} \frac{n}{n_0}.$$

Assuming $\delta = \lceil \log_{n'/d} \frac{n}{n_0} \rceil$ and multiplying matrices smaller than $n_0 \times n_0$ is done in n_0^3, we have that

$$\begin{aligned}
T(n) &\leq f(n) + \sum_{i=1}^{k} f(\frac{n}{n'} \cdot d_i) + \sum_{i=1}^{k}\sum_{j=1}^{k} f(\frac{n}{(n')^2} \cdot d_i \cdot d_j) + \sum_{i=1}^{k}\sum_{j=1}^{k}\sum_{l=1}^{k} T(\frac{n}{(n')^3} \cdot d_i \cdot d_j \cdot d_l) && (9)\\
&= 3\alpha n^2 + \beta + \sum_{i=1}^{k} \left[3\alpha \frac{n^2 d_i^2}{(n')^2} + \beta \right] + \sum_{i=1}^{k}\sum_{j=1}^{k} \left[3\alpha \frac{n^2 d_i^2 d_j^2}{(n')^4} + \beta \right] + \sum_{i=1}^{k}\sum_{j=1}^{k}\sum_{l=1}^{k} T(\frac{n}{(n')^3} \cdot d_i \cdot d_j \cdot d_l) && (10)\\
&\leq \left[3\alpha n^2 + \beta \right] + \left[3\alpha (\frac{n}{n'})^2 \sum_{i=1}^{k} d_i^2 + \sum_{i=1}^{k} \beta \right] + \left[3\alpha (\frac{n}{(n')^2})^2 \sum_{i=1}^{k}\sum_{j=1}^{k} d_i^2 d_j^2 + \sum_{i=1}^{k}\sum_{j=1}^{k} \beta \right] + \dots && (11)
\end{aligned}$$

Now observe that from Corollary 6.1, we have $\sum_{i=1}^{k} d_i^2 = |G|$. Furthermore, it is the case that

$$\sum_{i=1}^{k}\sum_{j=1}^{k} \cdots \sum_{l=1}^{k} d_i^2 d_j^2 \dots d_l^2 = \sum_{i=1}^{k} d_i^2 \sum_{j=1}^{k} d_j^2 \cdots \sum_{l=1}^{k} d_l^2 = |G| \cdot |G| \dots |G|.$$

Hence, we can further $T(n)$ as follows:

$$\begin{aligned}
T(n) &\leq \sum_{i=0}^{\delta} 3\alpha (\frac{n}{(n')^i})^2 |G|^i + \beta \cdot k^\delta + \sum_{i=1}^{k}\sum_{j=1}^{k} \cdots \sum_{l=1}^{k} n_0^3 && (12)\\
&= 3\alpha n^2 \sum_{i=0}^{\delta} \frac{|G|^i}{(n')^{2i}} + \beta \cdot k^\delta + n_0^3 \cdot k^\delta && (13)\\
&= 3\alpha n^2 \sum_{i=0}^{\delta} \left(\frac{|G|}{(n')^2} \right)^i + (\beta + n_0^3) \cdot k^\delta && (14)
\end{aligned}$$

To further reduce $T(n)$, we consider the following geometric sum formula:

$$\sum_{i=0}^{b} a^i = \begin{cases} b+1 & \text{if } a = 1 \\ \frac{a^{b+1}-1}{a-1} & \text{otherwise} \end{cases} \tag{15}$$

and a result in logarithm

$$a^{\log_b n} = n^{\log_b a}. \tag{16}$$

Here, we have 2 cases: $|G| = (n')^2$ or $|G| > (n')^2$. If $|G| = (n')^2$,

$$\begin{aligned} T(n) &\leq 3\alpha n^2(\delta + 1) + (\beta + n_0^3) \cdot k^\delta \qquad (17) \\ &= 3(|G|^2 + 1)n^2(\lceil \log_{n'/d}(\frac{n}{n_0}) \rceil + 1) + (\beta + n_0^3) \cdot k^{\lceil \log_{n'/d}(n/n_0) \rceil} \qquad (18) \\ &\approx 3(|G|^2 + 1)n^2 \log_{n'/d}(n) + (\beta + n_0^3) \cdot n^{\log_{n'/d}(k)} \qquad (19) \end{aligned}$$

where $T(n) \in \mathcal{O}(n^2 \log n)$ with sufficiently small k. This means that given a family of group of size $(n')^2$, we can get infinitely close to $\mathcal{O}(n^2)$. However, in practical application, we have not found any group G such that $|G| = (n')^2$, so further analysis on this case is not part of this report. Considering $|G| \neq (n')^2$, we have that

$$\begin{aligned} T(n) &\leq 3\alpha n^2 \frac{(|G|/(n')^2)^{\delta+1} - 1}{(|G|/(n')^2) - 1} + (\beta + n_0^3) \cdot k^\delta \qquad (20) \\ &\approx 3\alpha n^2 \left(\frac{|G|}{(n')^2} \right)^\delta + (\beta + n_0^3) \cdot k^\delta \qquad (21) \\ &= 3\alpha n^2 \left(\frac{|G|}{(n')^2} \right)^{\log_{n'/d}(n/n_0)} + (\beta + n_0^3) \cdot k^{\log_{n'/d}(n/n_0)} \qquad (22) \\ &= 3\alpha n^2 \left(\frac{n}{n_0} \right)^{\log_{n'/d}(|G|/(n')^2)} + (\beta + n_0^3) \cdot \left(\frac{n}{n_0} \right)^{\log_{n'/d} k} \qquad (23) \\ &= 3(|G|^2 + 1)n^2 \left(\frac{n}{n_0} \right)^{\log_{n'/d}(|G|/(n')^2)} + (3|G|(n')^2 + n_0^3) \cdot \left(\frac{n}{n_0} \right)^{\log_{n'/d} k} \qquad (24) \\ &= \frac{3(|G|^2 + 1)}{n_0^{\log_{n'/d}(|G|/(n')^2)}} \cdot n^{2 + \log_{n'/d}(|G|/(n')^2)} + \frac{(3|G|(n')^2 + n_0^3)}{n_0^{\log_{n'/d} k}} \cdot n^{\log_{n'/d} k} \qquad (25) \end{aligned}$$

We observe that $T(n) \in \mathcal{O}(n^\omega)$ where

$$\omega \leq \max\{2 + \log_{n'/d}(|G|/(n')^2), \log_{n'/d} k\}. \qquad (26)$$

We then try to solve for n_0 by solving

$$\begin{aligned} & T(n_0) \leq n_0^2 \cdot (2n_0 - 1) \qquad (27) \\ \iff & \frac{3(|G|^2 + 1)}{n_0^{\log_{n'/d}(|G|/(n')^2)}} \cdot n_0^{2 + \log_{n'/d}(|G|/(n')^2)} + \frac{(3|G|(n')^2 + n_0^3)}{n_0^{\log_{n'/d} k}} \cdot n_0^{\log_{n'/d} k} \leq 2n_0^3 - n_0^2 \qquad (28) \\ \iff & 3(|G|^2 + 1)n_0^2 + (3|G|(n')^2 + n_0^3) \leq 2n_0^3 - n_0^2 \qquad (29) \\ \iff & n_0^3 - (3|G|^2 + 4)n_0^2 - 3|G|(n')^2 \geq 0 \qquad (30) \end{aligned}$$

Using SageMath, assume $n_0 \geq 0$, we solve this inequality to obtain

$$\begin{aligned} n_0 \geq & \frac{\frac{2}{\sqrt[3]{4}}\left(3\,|G|^2+4\right)^2}{3\left(2\left(3\,|G|^2+4\right)^3+81\,|G|(n')^2+9n'\,\sqrt{(108\,|G|^6+432\,|G|^4+81\,|G|(n')^2+576\,|G|^2+256)|G|}\right)^{\frac{1}{3}}}+|G|^2 \\ & +\frac{1}{3\sqrt[3]{2}}\left(2\left(3\,|G|^2+4\right)^3+81\,|G|(n')^2+9\,\sqrt{(108\,|G|^6+432\,|G|^4+81\,|G|(n')^2+576\,|G|^2+256)|G|}n'\right)^{\frac{1}{3}}+\frac{4}{3} \end{aligned} \tag{31}$$

Using this formula on the puzzle

$$P = \begin{array}{|c|c|}\hline 1 & 3 \\ \hline 2 & 1 \\ \hline \end{array},$$

we get the values for n_0 as in Table 2.

p	$n_0 \approx$
8	$2.013 \cdot 10^8$
9	$5.165 \cdot 10^8$
10	$1.200 \cdot 10^9$
11	$2.572 \cdot 10^9$

Table 2: Some n_0's for puzzle P

Noted that these values are relatively large: with $n_0 \approx 5 \cdot 10^8$, multiply two matrices of this size takes approximately $3.96 \cdot 10^9$ years on an modern CPU which can do 10^9 arithmetic operations in 1 second. However, it is also true that Cohn & Umans give better performance for matrices larger than n_0. Noted that this value n_0 is from the assumption that no padding is needed for the matrices involved. More analysis on the runtime for different matrix sizes is to be carried out since the paddings may kill any performance the algorithm provided or further increasing n_0. However, this is not in the scope of this research.

5.2.3 Recurrence's Solution in Terms of Puzzles' Parameters

Given a balanced $s \times q$ puzzle P, Section 3.3 gives us that

$$|G| = s! \cdot p^{s \cdot q}$$
$$m' = n' = r' = s! \cdot (p-1)^{\frac{s \cdot q}{3}}$$

Moreover, to determine the possible value of d, from Huppert [18], we have Theorem 11, and Cohn et al. [6, Section 3.2] gives any group G generated have a subgroup H where $|H| = p^{s \cdot q}$.

Theorem 11 *[18, Proposition 2.6] If G has an abelian subgroup A, then all the character degrees of G are less than or equal to the index $[G : A] = |G|/|A|$.*

Hence, by Theorem 11, $d \leq |G|/|H| = \frac{s! \cdot p^{s \cdot q}}{p^{s \cdot q}} = s!$. By Equation 26, $T(n) = \mathcal{O}(n^{\omega})$ where $\omega \leq \max\{2 + \log_{n'/d}(|G|/(n')^2), \log_{n'/d} k\}$. We have

$$2 + \log_{n'/d}(|G|/(n')^2) = 2 + \log_{(s! \cdot (p-1)^{\frac{s \cdot q}{3}})/s!} \left[\frac{s! \cdot p^{s \cdot q}}{(s!)^2 \cdot (p-1)^{\frac{2s \cdot q}{3}}}\right] \tag{32}$$

$$= 2 + \log_{(p-1)^{\frac{s \cdot q}{3}}} \frac{p^{s \cdot q}}{s! \cdot (p-1)^{\frac{2s \cdot q}{3}}} \tag{33}$$

$$= 2 + \log_{p-1} \left(\frac{p^{s \cdot q}}{s! \cdot (p-1)^{\frac{2s \cdot q}{3}}}\right)^{\frac{3}{s \cdot q}} \tag{34}$$

$$= 2 + \log_{p-1} \frac{p^3}{(p-1)^2} - \log_{p-1} \sqrt[\frac{s \cdot q}{3}]{s!} \tag{35}$$

$$= 2 + \log_{p-1} p^3 - \log_{p-1}(p-1)^2 - \log_{p-1} \sqrt[\frac{s \cdot q}{3}]{s!} \tag{36}$$

$$= \log_{p-1} p^3 - \log_{p-1} \sqrt[\frac{s \cdot q}{3}]{s!} = \log_{p-1} \frac{p^3}{\sqrt[\frac{s \cdot q}{3}]{s!}} \tag{37}$$

and

$$\log_{n'/d} k = \log_{(p-1)^{\frac{s \cdot q}{3}}} k = \log_{p-1} \sqrt[\frac{s \cdot q}{3}]{k}. \tag{38}$$

Hence, we have

$$\omega \leq log_{p-1} \max\{\frac{p^3}{\sqrt[\frac{s \cdot q}{3}]{s!}}, \sqrt[\frac{s \cdot q}{3}]{k}\}. \tag{39}$$

Furthermore, for k, we are unable to come up with an explicit upper bound besides $|G|$; however, the lower bound for k is that

$$k \geq \frac{|G|}{d^2} = \frac{s! \cdot p^{s \cdot q}}{(s!)^2} = \frac{p^{s \cdot q}}{s!}$$

Using $k = p^{s \cdot q}$, we have

$$\omega \leq log_{p-1} \max\{\frac{p^3}{\sqrt[\frac{s \cdot q}{3}]{s!}}, \sqrt[\frac{s \cdot q}{3}]{\frac{p^{s \cdot q}}{s!}}\} = log_{p-1} \max\{\frac{p^3}{\sqrt[\frac{s \cdot q}{3}]{s!}}, \frac{p^3}{\sqrt[\frac{s \cdot q}{3}]{s!}}\} = \log_{p-1} \frac{p^3}{\sqrt[\frac{s \cdot q}{3}]{s!}}. \tag{40}$$

Furthermore, we have that

$$\log_{p-1} \frac{p^3}{\sqrt[\frac{s \cdot q}{3}]{s!}} = \log_{p-1} p^3 - \log_{p-1} \sqrt[\frac{s \cdot q}{3}]{s!} = \frac{\log p^3}{\log(p-1)} - \frac{\log \sqrt[\frac{s \cdot q}{3}]{s!}}{\log(p-1)} = \frac{3 \log p}{\log(p-1)} - \frac{3 \log s!}{sq \log(p-1)} \tag{41}$$

Firstly, we notice that this is the same result as Corollary 7.2. Also, if a group G has only character degree d, to derive the runtime of the puzzle the first term is sufficient. However, this is not usually the case. Furthermore, we test Equation 39 to see which terms in the $\max$ operation decide ω; Table 3 shows this attempt with a 2×2 puzzle P with several values for p.

$$P = \begin{array}{|c|c|} \hline 1 & 3 \\ \hline 2 & 1 \\ \hline \end{array}$$

	$log_{p-1} \frac{p^3}{\sqrt[\frac{s \cdot q}{3}]{s!}}$	$log_{p-1} \sqrt[\frac{s \cdot q}{3}]{k}$
$p = 7$	2.96795979207917	2.99283342390225
$p = 8$	2.93870929364118	2.95636537711459
$p = 9$	2.91992500144231	2.93304185641585
$p = 10$	2.90725625384478	2.91734584682942
$p = 11$	2.89840555872669	2.90638279486103
$p = 12$	2.89206106814289	2.89851025113160
$p = 13$	2.88742745630979	2.89273824793205
$p = 14$	2.88399908633937	2.88844074289190
$p = 15$	2.88144184385539	2.88520603645213
$p = 16$	2.87952777911042	2.88275444119794
$p = 17$	2.87809713093775	2.88089067025888
$p = 18$	2.87703532999695	2.87947513614257
$p = 19$	2.87625857447851	2.87840602556709
$p = 20$	2.87570450437356	2.87760775696893
$p = 21$	2.87532601086926	2.87702334720466
$p = 22$	2.87508703046391	2.87660924367445
$p = 23$	2.87495962817090	2.87633175203413

Table 3: Comparison of terms in Equation 39

We can see that the term in k is slightly above the other term in some cases. This means this term has significance and can not be discarded in Equation 39. However, it is also the case that the 2 terms converge toward each other. This means if we can get a good approximation for k in terms of s, p, and q, we can potentially figure out the p where this upperbound for ω reaches its global minima. Table 3 is part of the data in the last 2 columns of Table A2.

5.2.4 Comparing to Cohn & Umans' Results

Referring back to Section 3.3, Theorem 7 and Corollary 7.1 gives us 2 ways to determine the upperbound for ω while Corollary 7.2 is part of our solution. Here, we aim to compare our upperbound for ω from Equation 39. Here, since the group generated by the 3×3 puzzles is too large to be generated in a reasonable time, we decided to assume that the puzzle

$$P = \begin{array}{|c|c|} \hline 1 & 3 \\ \hline 2 & 1 \\ \hline \end{array}$$

is balanced since Equation 39 does not include n_1, n_2, and n_3 terms. For better results, this procedure should have been done with a balanced puzzle instead; however, the limitation on computing resources places a limit on what can be done.

Sadly, from Figure 3, our bound does not give a better bound than those of Cohn & Umans. As such, more improvements and eases of assumptions would need to be made to give us a better upperbound of ω. However, looking up to Table 3, the first term's values are better on the order of 10^{-4} than Corollary 7.1. Furthermore, as p increases, the k term slowly converged towards the other term. This means, we may be able to figure out the minimum ω from the puzzle alone using Equation 40 or Corollary 7.2 i.e.

$$\omega \leq \log_{p-1} \frac{p^3}{\sqrt[\frac{s \cdot q}{3}]{s!}}$$

We then attempt to minimize $\log_{p-1} \frac{p^3}{\sqrt[\frac{s \cdot q}{3}]{s!}}$ using some basic calculus.

$$\frac{d}{dp} \log_{p-1} \frac{p^3}{\sqrt[\frac{s \cdot q}{3}]{s!}} = \frac{3}{\ln(p-1)p} - \frac{3\ln(p) - \ln(\sqrt[\frac{s \cdot q}{3}]{s!})}{\ln^2(p-1)(p-1)} = 0 \tag{42}$$

$$\tag{43}$$

However, solving this has been proven to be hard. Therefore, we find it easier to plug in the formula while increasing the value of p til the minima is reached.

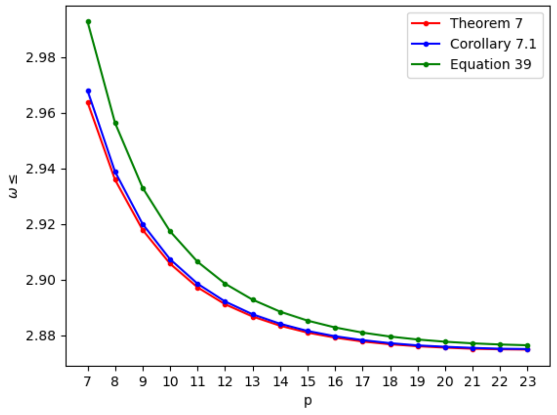

Figure 3: Graphical comparisons of ω from Theorem 7, Corollary 7.1, and Equation 39.

5.3 Simulation of Cohn & Umans Matrix Multiplication

This section involves introducing a technique in approximate the runtime on Cohn & Umans' matrix multiplication algorithm via simulation as empirical experiments are not possible in the scope of this project since the lower bound of the sizes of matrices where this algorithm is more efficient than the naive one is large (in the order of $|G|^2$, Section 5.2.2). Hence, we based our simulation on Equation 7. To further simplify the recurrence, we observed that the $\{d_i\}$ are not unique, and there are repeated characters degrees. For example, for the SUSP

$$P = \begin{array}{|c|c|}\hline 1 & 3 \\ \hline 2 & 1 \\ \hline \end{array}$$

any group generated from puzzle P only has characters degrees 1 and 2. Hence, we define

$$c_G(d) = \text{the number of character degree } d \text{ of group } G.$$

Then, we have

$$T(m,n,r) = f(m,n,r) + \sum_{d \in \{d_i\}} c_G(d) \cdot T(\frac{m}{m'} \cdot d, \frac{n}{n'} \cdot d, \frac{r}{r'} \cdot d) \tag{44}$$

This has proven to be helpful in both simulating the number of operations and solving the recurrence since it is normally the case that $k >> |\{d_i\}|$. With this more data analysis can be carried out in a reasonable time. To test our theoretical works, we use this simulation to calculate the number of operations used at $n >> n_0$ using puzzle P. Table 4 shows the derived ω for $p = 7, 8, 9$. For each p, we have 10 data points.

p	ω	$\pm$
7	2.977882814587704	0.024268302689252657
8	2.933323860044298	0.023762556059874554
9	2.930549267234582	0.023879395279609056

Table 4: ω derived from simulation data

Table 4 is consistent with Table 3. However, it is also the case that the simulation assumes the sizes of matrices used in Table 4 perfectly divides m', n', and r' at every recursion depth. Our attempt to run this simulation without this assumption does not give $\omega \leq 3$ (Table 5). This shows that the paddings cause the recursion depth to go up, thus, make the algorithm slower than it could perform. This means Cohn-Umans matrix multiplication only yields performance for multiplication of matrices of sizes $(m')^s \times (n')^s$

p	ω	$\pm$
7	3.9165251977068283	0.020915332916487832
8	3.8517514701784883	0.024369180838071974
9	3.841768882593626	0.025825165591696144

Table 5: ω derived from simulation data (with matrix paddings involved)

and $(n')^s \times (r')^s$ in this experiment.

At further inspection of our simulation, we realized the paddings actually allow the recursion depth δ to go above what we were theorizing in Section 5.2.2 in the case where the matrices' sizes are "uneven" to $(m')^s \times (n')^s$ and $(n')^s \times (r')^s$ which cause δ to increase up to the order of

$$\max\{\log_{m'/d}(m), \log_{n'/d}(n) \log_{r'/d} r\}$$

where group G realizes $< m', n', r' >$ and m, n, r are the sizes of the sides of input matrices. This heavily impacts the runtime in the case of "uneven" input matrices as the recursion tree grows more than we anticipated. From our experiments, in the worst case, this could come up to multiples of the size of the anticipated depth. To prevent this from happening, we decide to add another base case for the matrix multiplication algorithm

$$T(m, n, r) = mnr, \quad \text{if} \min\{m, n, r\} < d. \tag{45}$$

This means if one of the sides of the matrices has hit its recursion depth, and can not be reduced any further, then we stop the recurrence and just do naive matrix multiplication. By adding the extra basecase to the algorithm, we get results as shown in Table 6. This results are comparable to that of Table 4. More analysis is required to confirm this, but it shows how we can achieve comparable ω in the "uneven" cases.

p	ω	$\pm$
7	2.9497220826087873	0.020710468169938902
8	2.978316222654991	0.024954822802836497
9	2.95564347379515	0.026880794958700098

Table 6: ω derived from simulation data (with matrix paddings and extra basecase involved)

The raw data for Tables 4, 5, and 6 are Tables A3, A4, and A5, respectively.

6 Conclusion

This section is to summarize our advancements in the Cohn & Umans matrix multiplication framework.

6.1 On Computation of Wedderburn Decomposition Φ

We have been able to implement and test the algorithm producing the Wedderburn Decomposition Φ on algebraically closed fields. However, this algorithm is randomized and has a fraction chance of failing. In the future, we might want to migrate to a deterministic algorithm for this problem. One of such is given by Ivanyos et al. [19]. Another problem we have is we are unable to efficiently compute Φ without taking an approximation which results in computation in high precision (256-bit precision instead of standard 64-bit) described in Section 4.6. The last problem is that the splitting procedure used by SageMath, which was introduced in 1962 [10], is not efficient for larger groups. Hence, we wish to produce a better implementation or use a completely different algorithm if possible.

6.2 On Runtime Analysis

We have been able to provide a lowerbound for, n_0, the size of the matrix where Cohn & Umans framework will be more efficient, and an upperbound for ω in Sections 5.2.2 and 5.2.3 in terms of the corresponding balanced puzzle. Going forward, we want to either find a tighter bound for ω and n_0 or test these against the implemented algorithm. The effort to test the algorithm is still impeded due to n_0 being in the order of the size of group square: for the smallest puzzle which gives $\omega < 3$, $n_0 \approx 5 \cdot 10^8$. Multiplying two matrices of this size takes approximately $3.96 \cdot 10^9$ years on a modern CPU that can do 10^9 arithmetic operations in 1 second. An efficient simulation technique can be used to obtain the exact number of operations the matrix multiplication used. We have run some experiments regarding this simulation whose results imply that improvements are only given with certain matrix sizes. Some modifications have been made to allow fast matrix multiplication in other cases, but we need verify on whether this actually works on general cases.

Acknowledgements

Thanks to Davenport Summer Research Fellowship for sponsoring my research, my advisor Matthew Anderson for advising and ”motivating” me in the process, Zachary Dubinsky for the helpful discussions and materials, and my colleagues for the continuous and helpful support.

References

[1] Charu C. Aggarwal. *Linear Algebra and Optimization for Machine Learning [electronic resource] : A Textbook / by Charu C. Aggarwal.* eng. 1st ed. 2020. Cham: Springer International Publishing, 2020. ISBN: 3-030-40344-0.

[2] Matthew Anderson, Zongliang Ji, and Anthony Xu. "Matrix Multiplication: Verifying Strong Uniquely Solvable Puzzles". In: June 2020, pp. 464–480. ISBN: 978-3-030-51824-0. DOI: `10.1007/978-3-030-51825-7_32`.

[3] Matthew Anderson and Vu Le. *Efficiently-Verifiable Strong Uniquely Solvable Puzzles and Matrix Multiplication*. 2023. eprint: `arXiv:2307.06463`.

[4] Murray R. Bremner. In: 3.1 (2011), pp. 47–66. DOI: `doi:10.1515/gcc.2011.003`. URL: `https://doi.org/10.1515/gcc.2011.003`.

[5] Witty Carl. *Field of Algebraic Numbers*. 2007. URL: `https://doc.sagemath.org/html/en/reference/number_fields/sage/rings/qqbar.html`.

[6] H. Cohn et al. "Group-theoretic algorithms for matrix multiplication". In: *46th Annual IEEE Symposium on Foundations of Computer Science (FOCS'05)* (2005), pp. 379–388.

[7] Henry Cohn and Christopher Umans. "A group-theoretic approach to fast matrix multiplication". In: *44th Annual IEEE Symposium on Foundations of Computer Science, 2003. Proceedings.* (2003), pp. 438–449.

[8] Don Coppersmith and Shmuel Winograd. "Matrix multiplication via arithmetic progressions". In: *Proceedings of the nineteenth annual ACM symposium on Theory of computing* (1987).

[9] Thomas H. Cormen et al. *Introduction to Algorithms*. 2nd ed. The MIT Press, 2001.

[10] Charles W. Curtis and Irving Reiner. *Representation Theory of Finite Groups and Associative Algebras*. AMS Chelsea Publishing, 1962. ISBN: 978-0-8218-4066-5. URL: `https://doi.org/10.1090/chel/356`.

[11] Yurij A. Drozd and Vladimir V. Kirichenko. *Finite dimensional algebras*. Springer Verlag, 1994.

[12] Zachary Dubinsky. *Fast Matrix Multiplication and The Wedderburn-Artin Theorem*. 2023.

[13] David S. Dummit and Richard M. Foote. *Abstract Algebra*. en. 3rd ed. Nashville, TN: John Wiley Sons, 2003. ISBN: 9780471433347.

[14] Wayne Eberly. "Computations for algebras and group representations". In: (1989).

[15] Wayne Eberly. "Decomposition of algebras over finite fields and number fields". In: *computational complexity* 1.2 (June 1991), pp. 183–210. ISSN: 1420-8954. DOI: 10.1007/BF01272520. URL: https://doi.org/10.1007/BF01272520.

[16] Wayne Eberly. "Decompositions of algebras over and ". In: *computational complexity* 1.3 (Sept. 1991), pp. 211–234. ISSN: 1420-8954. DOI: 10.1007/BF01200061. URL: https://doi.org/10.1007/BF01200061.

[17] Willem de Graaf. "Finding Splitting Elements and Maximal Tori in Matrix Algebras". In: (Sept. 1999).

[18] Bertram Huppert. *Character Theory of Finite Groups*. Berlin, New York: De Gruyter, 1998. ISBN: 9783110809237. DOI: doi:10.1515/9783110809237. URL: https://doi.org/10.1515/9783110809237.

[19] Gábor Ivanyos, Lajos Rónyai, and Josef Schicho. "Splitting full matrix algebras over algebraic number fields". In: *Journal of Algebra* 354.1 (2012), pp. 211–223. ISSN: 0021-8693. DOI: https://doi.org/10.1016/j.jalgebra.2012.01.008. URL: https://www.sciencedirect.com/science/article/pii/S0021869312000300.

[20] V. Strassen. "Gaussian Elimination is not Optimal." In: *Numerische Mathematik* 13 (1969), pp. 354–356. URL: http://eudml.org/doc/131927.

A Raw Data Tables

The following appendix provides supplementary data and detailed information to support the findings and analyses presented in the main text.

prec	recur_depth	err
32	2	5.7264×10^{10}
64	2	2.3558×10^{1}
96	2	3.9026×10^{-9}
128	2	9.3591×10^{-19}
160	2	1.8675×10^{-28}
192	2	6.8210×10^{-38}
224	2	1.1619×10^{-47}
256	2	3.9527×10^{-57}
32	3	2.7042×10^{28}
64	3	9.0478×10^{18}
96	3	1.9245×10^{9}
128	3	5.4285×10^{-1}
160	3	9.0050×10^{-11}
192	3	2.4477×10^{-20}
224	3	8.0423×10^{-30}
256	3	2.1653×10^{-39}

(a) Cohn-Umans Matrix Multiplication's Error Propagation

k	runtime
2	626.4389674663544
2	631.1938946247101
2	629.3493115901947
2	631.6254794597626
3	61967.19587850571
3	61964.51694512367
3	62018.92904090881
3	62266.60267663002

(b) Cohn-Umans' Matrix Multiplication Runtime

Table A1: Raw Experimental Data

p	omega-7	omega-7-1	omega-39	omega-39-log-p	omega-39-log-k
7	2.963 796	2.967 997	2.992 833	2.967 960	2.992 833
8	2.935 894	2.938 794	2.956 365	2.938 709	2.956 365
9	2.917 892	2.919 992	2.933 042	2.919 925	2.933 042
10	2.905 691	2.907 291	2.917 346	2.907 256	2.917 346
11	2.897 190	2.898 490	2.906 383	2.898 406	2.906 383
12	2.891 089	2.892 089	2.898 510	2.892 061	2.898 510
13	2.886 689	2.887 489	2.892 738	2.887 427	2.892 738
14	2.883 388	2.884 088	2.888 441	2.883 999	2.888 441
15	2.880 888	2.881 488	2.885 206	2.881 442	2.885 206
16	2.879 088	2.879 588	2.882 754	2.879 528	2.882 754
17	2.877 688	2.878 188	2.880 891	2.878 097	2.880 891
18	2.876 688	2.877 088	2.879 475	2.877 035	2.879 475
19	2.875 988	2.876 288	2.878 406	2.876 259	2.878 406
20	2.875 488	2.875 788	2.877 608	2.875 705	2.877 608
21	2.875 088	2.875 388	2.877 023	2.875 326	2.877 023
22	2.874 887	2.875 088	2.876 609	2.875 087	2.876 609
23	2.874 787	2.874 987	2.876 332	2.874 960	2.876 332

Table A2: Raw Data of Values of ω's by several methods.

p	n	num_oper
7	5.8676×10^{37}	1.8187×10^{119}
7	4.2247×10^{39}	4.6098×10^{124}
7	3.0418×10^{41}	8.6153×10^{129}
7	2.1901×10^{43}	1.5091×10^{135}
7	1.5769×10^{45}	2.6135×10^{140}
7	1.1353×10^{47}	8.3268×10^{146}
7	8.1745×10^{48}	2.2729×10^{152}
7	5.8856×10^{50}	4.3851×10^{157}
7	4.2376×10^{52}	7.7521×10^{162}
7	3.0511×10^{54}	1.3451×10^{168}
8	6.6695×10^{40}	5.9097×10^{126}
8	6.5361×10^{42}	6.7653×10^{133}
8	6.4054×10^{44}	3.7392×10^{139}
8	6.2773×10^{46}	1.5844×10^{145}
8	6.1518×10^{48}	6.4071×10^{150}
8	6.0287×10^{50}	2.5739×10^{156}
8	5.9082×10^{52}	2.6904×10^{163}
8	5.7900×10^{54}	1.5830×10^{169}
8	5.6742×10^{56}	6.8590×10^{174}
8	5.5607×10^{58}	2.7881×10^{180}
9	1.5136×10^{41}	1.6566×10^{128}
9	1.9374×10^{43}	1.3914×10^{134}
9	2.4798×10^{45}	1.1685×10^{140}
9	3.1742×10^{47}	3.6688×10^{147}
9	4.0629×10^{49}	4.0437×10^{153}
9	5.2006×10^{51}	3.5306×10^{159}
9	6.6567×10^{53}	2.9781×10^{165}
9	8.5206×10^{55}	2.5020×10^{171}
9	1.0906×10^{58}	2.1013×10^{177}
9	1.3960×10^{60}	6.0626×10^{184}

Table A5: Raw Data of Cohn-Umans Matrix Multiplication Simulation (With Paddings and Extra Basecase)

p	n	num_oper
7	1.1035×10^{23}	1.8254×10^{73}
7	7.9453×10^{24}	1.0491×10^{80}
7	5.7206×10^{26}	1.4144×10^{88}
7	4.1188×10^{28}	1.0805×10^{95}
7	2.9656×10^{30}	6.6218×10^{101}
7	2.1352×10^{32}	7.8742×10^{109}
7	1.5373×10^{34}	6.6090×10^{116}
7	1.1069×10^{36}	4.1804×10^{123}
7	7.9696×10^{37}	4.3870×10^{131}
7	5.7381×10^{39}	4.0557×10^{138}
8	1.4809×10^{25}	6.0401×10^{80}
8	1.4513×10^{27}	9.8288×10^{87}
8	1.4223×10^{29}	2.0174×10^{95}
8	1.3938×10^{31}	3.5108×10^{102}
8	1.3660×10^{33}	1.7436×10^{111}
8	1.3386×10^{35}	3.9244×10^{118}
8	1.3119×10^{37}	6.9200×10^{125}
8	1.2856×10^{39}	3.0944×10^{134}
8	1.2599×10^{41}	7.4956×10^{141}
8	1.2347×10^{43}	1.3558×10^{149}
9	1.0210×10^{25}	6.0220×10^{78}
9	1.3069×10^{27}	2.5907×10^{86}
9	1.6728×10^{29}	1.0980×10^{94}
9	2.1412×10^{31}	4.7972×10^{101}
9	2.7408×10^{33}	8.2126×10^{110}
9	3.5082×10^{35}	4.4172×10^{118}
9	4.4905×10^{37}	1.9572×10^{126}
9	5.7478×10^{39}	3.0852×10^{135}
9	7.3572×10^{41}	1.7689×10^{143}
9	9.4172×10^{43}	7.9738×10^{150}

Table A4: Raw Data of Cohn-Umans Matrix Multiplication Simulation (With Paddings)

p	n	num_oper
7	2.5446×10^{30}	1.2684×10^{97}
7	1.8321×10^{32}	2.2932×10^{102}
7	1.3191×10^{34}	3.9867×10^{107}
7	9.4977×10^{35}	6.8959×10^{112}
7	6.8383×10^{37}	2.5192×10^{119}
7	4.9236×10^{39}	6.3025×10^{124}
7	3.5450×10^{41}	1.1720×10^{130}
7	2.5524×10^{43}	2.0503×10^{135}
7	1.8377×10^{45}	3.5499×10^{140}
7	1.3232×10^{47}	1.1532×10^{147}
8	9.9384×10^{32}	4.1672×10^{104}
8	9.7396×10^{34}	1.7209×10^{110}
8	9.5448×10^{36}	6.9270×10^{115}
8	9.3539×10^{38}	2.7811×10^{121}
8	9.1668×10^{40}	3.2821×10^{128}
8	8.9835×10^{42}	1.7748×10^{134}
8	8.8038×10^{44}	7.4686×10^{139}
8	8.6278×10^{46}	3.0159×10^{145}
8	8.4552×10^{48}	1.2114×10^{151}
8	8.2861×10^{50}	4.8627×10^{156}
9	8.7902×10^{32}	6.5843×10^{104}
9	1.1251×10^{35}	6.6622×10^{110}
9	1.4402×10^{37}	5.7007×10^{116}
9	1.8434×10^{39}	4.7944×10^{122}
9	2.3596×10^{41}	4.0268×10^{128}
9	3.0203×10^{43}	3.3817×10^{134}
9	3.8660×10^{45}	1.0878×10^{142}
9	4.9485×10^{47}	1.1771×10^{148}
9	6.3340×10^{49}	1.0227×10^{154}
9	8.1076×10^{51}	8.6195×10^{159}

Table A3: Raw Data of Cohn-Umans Matrix Multiplication Simulation (No Paddings Involved)