Faster matrix multiplication

A study to finding bounds on
the rank of
matrix multiplication tensors

Owen M. de Heer
4951913

TUDelft

Faster matrix multiplication

A study to finding bounds on
the rank of
matrix multiplication tensors

by

Owen M. de Heer
4951913

to obtain the degree of Bachelor of Science
at the Delft University of Technology,
to be defended publicly on Wednesday July 5, 2023 at 17:00 PM.

Student number:	4951913	
Project duration:	April 24, 2023 – July 5, 2023	
Thesis committee:	Prof. dr. D. Gijswijt,	TU Delft, supervisor
	dr. ir. W. Caspers,	TU Delft

Abstract

Matrix multiplication with the standard algorithm has an algebraic complexity of $\mathcal{O}(n^3)$ for $n \times n$ matrices, but in 1969 Strassen [26] found another algorithm for multiplying 2×2 matrices with which he showed that matrix multiplication can be done with a complexity of $\mathcal{O}(n^{2.81})$ by applying his algorithm recursively for large matrices. We present a historical overview of the best known bounds on the matrix multiplication exponent, with the current best known bound of 2.371866 [10] and methods, used to find new algorithms for other matrix multiplications, such as alternating least squares (ALS) and SAT solving. After this we present a novel method with which we found a rank 23 decomposition of the $\langle 3,3,3\rangle$ matrix multiplication tensor that may be inequivalent to known decompositions. We also found decompositions for $\langle 2,2,2\rangle$, $\langle 2,2,3\rangle$, $\langle 2,2,4\rangle$, $\langle 2,3,3\rangle$ and $\langle 2,2,5\rangle$ that provide the same bounds as earlier found decompositions. Our method found many decompositions for the smaller tensors but only one for the larger due to time limits. The method is based on the alternating least squares method (ALS) with the modification that we 'push' the coefficients towards integer (or rational) numbers. After a number of iterations and rounding the result this sometimes yields exact decompositions so that we can bound the matrix multiplication exponent.

Contents

1 Introduction

In mathematics one often finds oneself formulating a problem as a set of linear equations or representing data in an array of values. This can all be represented with matrices and solved by performing operations on these matrices. This all falls under the study of linear algebra which is fundamental in all sorts of mathematical and other technical engineering. These problems often deal with matrices of enormous sizes and the processing of these can take a while. One of these processes is matrix multiplication which is used for other processes as well such as inversion or solving systems of equations. The standard method for multiplying $n \times n$ matrices requires $\mathcal{O}(n^3)$ algebraic operations but in 1969 Strassen [26] found another algorithm to multiply two 2×2 matrices that uses only 7 multiplications but requires 18 additions which leads to a complexity of $\mathcal{O}(n^{2.81})$, when applied recursively for large matrices, that will be proven in Chapter 2.

First as a reminder the standard way to multiply two 2×2 matrices is

$$\begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix} \cdot \begin{bmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{bmatrix} = \begin{bmatrix} C_{11} & C_{12} \\ C_{21} & C_{22} \end{bmatrix} = \begin{bmatrix} A_{11} \cdot B_{11} + A_{12} \cdot B_{21} & A_{11} \cdot B_{12} + A_{12} \cdot B_{22} \\ A_{21} \cdot B_{11} + A_{22} \cdot B_{21} & A_{21} \cdot B_{12} + A_{22} \cdot B_{22} \end{bmatrix} \tag{1.1}$$

which has an algebraic complexity of $8 + 4 = 12$ for the 8 multiplications and 4 additions.

But Strassen found that by defining the following $M_1, \dots, M_7$ he could write the elements in C as below.

$$\begin{aligned}
M_1 &= (A_{11} + A_{22}) \cdot (B_{11} + B_{22}) \\
M_2 &= (A_{21} + A_{22}) \cdot B_{11} \\
M_3 &= A_{11} \cdot (B_{12} - B_{22}) \\
M_4 &= A_{22} \cdot (B_{21} - B_{11}) \\
M_5 &= (A_{11} + A_{12}) \cdot B_{22} \\
M_6 &= (A_{21} - A_{11}) \cdot (B_{11} + B_{12}) \\
M_7 &= (A_{12} - A_{22}) \cdot (B_{21} + B_{22}) \\
\\
C_{11} &= M_1 + M_4 - M_5 + M_7 \\
C_{12} &= M_3 + M_5 \\
C_{21} &= M_2 + M_4 \\
C_{22} &= M_1 - M_2 + M_3 + M_6
\end{aligned} \tag{1.2}$$

Note that when the multiplications and additions are simplified these C_{ij} are the same as defined in equation (1.1) and therefore this is a valid 2×2 matrix multiplication algorithm as well.

This algorithm has an algebraic complexity of $7 + 18 = 25$ for the 7 multiplications and 18 additions. This is not better than the 12 of the standard algorithm, but the magic happens when applying this method recursively for block matrices that have sizes of powers of two. In fact it is precisely this recursion and the fact that we only need 7 multiplications that leads to the complexity $\mathcal{O}(n^{\log_2(7)}) = \mathcal{O}(n^{2.81})$.

To see this consider multiplying two 8×8 matrices $A \cdot B = C$. First divide them in four 2×2 block matrices where the blocks are 4×4 in size. Then apply Strassen's algorithm to these 2×2 block matrices. For the multiplications of the blocks divide these again, now into four 2×2 blocks and apply Strassen's algorithm again to those block matrices. This division is demonstrated below for matrix A. For larger matrix multiplications repeat this all the way down. By doing this it turns out that for large enough matrices, by recursively applying Strassen's algorithm, fewer algebraic operations are required than when using the standard algorithm. This will be proven in the Proposition 2.2.1.

$$A = \left[\begin{array}{cccc|cccc}
A_{11} & \multicolumn{1}{c|}{A_{12}} & A_{13} & A_{14} & A_{15} & A_{16} & A_{17} & A_{18} \\
A_{21} & \multicolumn{1}{c|}{A_{22}} & A_{23} & A_{24} & A_{25} & A_{26} & A_{27} & A_{28} \\
\cline{1-4}
A_{31} & \multicolumn{1}{c|}{A_{32}} & A_{33} & A_{34} & A_{35} & A_{36} & A_{37} & A_{38} \\
A_{41} & \multicolumn{1}{c|}{A_{42}} & A_{43} & A_{44} & A_{45} & A_{46} & A_{47} & A_{48} \\
\hline
A_{51} & A_{52} & A_{53} & A_{54} & A_{55} & A_{56} & A_{57} & A_{58} \\
A_{61} & A_{62} & A_{63} & A_{64} & A_{65} & A_{66} & A_{67} & A_{68} \\
A_{71} & A_{72} & A_{73} & A_{74} & A_{75} & A_{76} & A_{77} & A_{78} \\
A_{81} & A_{82} & A_{83} & A_{84} & A_{85} & A_{86} & A_{87} & A_{88}
\end{array}\right]$$

One may start to wonder if maybe there is an even better algorithm that brings the exponent of matrix multiplication even lower. Before investigating this it is important to formalise and generalise some things which will be done in Chapter 2.

1.1. Thesis structure

In Chapter 2 we will start with some preliminaries, definitions and theorems where we define exactly what this exponent is, what a tensor is, how it can relate to matrix multiplication and how its decomposition improves the bound on the exponent. With this knowledge we will present a historical overview of all the improvements over time from Strassen's simple method in 1969 to more convoluted and almost incomprehensible methods later on to gain marginal improvements. We also present a list of some of the methods used to find decompositions for specific tensors to lower the bound on their rank from ALS to SAT solving. Then we present the methods we attempted to implement in Chapter 4 and construct a novel method we created based on a modification on ALS where we want the coefficients to be integer or half-integer. With this method we are able to rediscover known bounds for many small matrix multiplication tensors from $\langle 2,2,2\rangle$ to $\langle 3,3,3\rangle$, $\langle 2,2,4\rangle$, $\langle 2,2,5\rangle$ and $\langle 2,3,4\rangle$. We were however not able to improve any of the known bounds on these.

2 Preliminaries

This chapter is based on the tutorial handout by Le Gall [17] with additions of proofs and examples.
We start by proving that Strassen's algorithm does indeed improve the algebraic complexity of matrix multiplication. This means that when one wants to multiply large matrices (say 1024×1024) it is faster to use Strassen's algorithm than the standard one, assuming no overhead computation. After this we generalise this method to bilinear algorithms and further to tensor decomposition. This will be the basis for any further research.

2.1. Algebraic complexity

Definition 2.1.1. The ***algebraic complexity*** of an algorithm is the number of algebraic operations it requires, i.e. the number of additions/subtractions and multiplications/divisions.

Consider again the multiplication of two 2×2 matrices A and B resulting in the 2×2 matrix C: $A \cdot B = C$ where

$$\begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix} \cdot \begin{bmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{bmatrix} = \begin{bmatrix} C_{11} & C_{12} \\ C_{21} & C_{22} \end{bmatrix} = \begin{bmatrix} A_{11} \cdot B_{11} + A_{12} \cdot B_{21} & A_{11} \cdot B_{12} + A_{12} \cdot B_{22} \\ A_{21} \cdot B_{11} + A_{22} \cdot B_{21} & A_{21} \cdot B_{12} + A_{22} \cdot B_{22} \end{bmatrix}. \tag{2.1}$$

We can write this as

$$\begin{aligned} M_1 &= A_{11} \cdot B_{11} \\ M_2 &= A_{12} \cdot B_{21} \\ M_3 &= A_{11} \cdot B_{12} \\ M_4 &= A_{12} \cdot B_{22} \\ M_5 &= A_{21} \cdot B_{11} \\ M_6 &= A_{22} \cdot B_{21} \\ M_7 &= A_{21} \cdot B_{12} \\ M_8 &= A_{22} \cdot B_{22} \end{aligned}$$

$$\begin{aligned} C_{11} &= M_1 + M_2 \\ C_{11} &= M_3 + M_4 \\ C_{11} &= M_5 + M_6 \\ C_{11} &= M_7 + M_8 \end{aligned}$$

The reason for which will become clear in section 2.3.

It is clear that this has 8 multiplications and 4 additions and so an algebraic complexity of $8 + 4 = 12$. In general to multiply two $n \times n$ matrices the standard algorithm has an algebraic complexity of $n^3 + (n-1)n^2$ which is of the order $\mathcal{O}(n^3)$.

Definition 2.1.2. Let $\boldsymbol{C(n)}$ be the smallest algebraic complexity for multiplying two $n \times n$ matrices for some positive integer n.

Definition 2.1.3. We define $\boldsymbol{\omega}$ by

$$\omega = \inf\{\alpha : C(n) \text{ is } \mathcal{O}(n^{\alpha})\}.$$

That is, ω is the lowest exponent such that we can multiply some large $n \times n$ matrices with $\mathcal{O}(n^{\omega+\epsilon})$ operations for all $\epsilon > 0$. It is clear that $2 \leq \omega \leq 3$ and by the result of Strassen even that $\omega \leq \log_2(7) \leq 2.81$.

2.2. Strassen's algorithm

$$\begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix} \cdot \begin{bmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{bmatrix} = \begin{bmatrix} C_{11} & C_{12} \\ C_{21} & C_{22} \end{bmatrix} \tag{2.2}$$

$$\begin{aligned}
M_1 &= (A_{11} + A_{22}) \cdot (B_{11} + B_{22}) \\
M_2 &= (A_{21} + A_{22}) \cdot B_{11} \\
M_3 &= A_{11} \cdot (B_{12} - B_{22}) \\
M_4 &= A_{22} \cdot (B_{21} - B_{11}) \\
M_5 &= (A_{11} + A_{12}) \cdot B_{22} \\
M_6 &= (A_{21} - A_{11}) \cdot (B_{11} + B_{12}) \\
M_7 &= (A_{12} - A_{22}) \cdot (B_{21} + B_{22}) \\
\\
C_{11} &= M_1 + M_4 - M_5 + M_7 \\
C_{12} &= M_3 + M_5 \\
C_{21} &= M_2 + M_4 \\
C_{22} &= M_1 - M_2 + M_3 + M_6
\end{aligned} \tag{2.3}$$

Proposition 2.2.1. Strassen's algorithm for multiplying $n \times n$ matrices has an algebraic complexity of $\mathcal{O}(n^{\log_2(7)}) = \mathcal{O}(n^{2.81})$.

Proof. First consider the case where $n = 2^k$ for some k. Let $f(k)$ be the algebraic complexity for multiplying two $2^k \times 2^k$ matrices using Strassen's algorithm. Because there are 7 multiplications of blocks that have sizes $2^{k-1} \times 2^{k-1}$ and 18 additions of these blocks we obtain the recursive formula

$$f(k) = 7f(k-1) + 18\left(2^{k-1}\right)^2.$$

We claim that this is solved by

$$f(k) = 7 \cdot 7^k - 6 \cdot 4^k \tag{2.4}$$

which we prove by induction. As shown in Chapter 1 we have

$$f(1) = 25 = 49 - 24 = 7 \cdot 7^1 - 6 \cdot 4^1$$

satisfying (2.4) as a base case. Now suppose

$$f(j) = 7 \cdot 7^j - 6 \cdot 4^j$$

for some $j \geq 1$. Then

$$\begin{aligned}
f(j+1) &= 7f(j) + 18\left(2^j\right)^2 \\
&= 7\left(7 \cdot 7^j - 6 \cdot 4^j\right) + 18\left(2^j\right)^2 \\
&= 7 \cdot 7^{j+1} - 42\left(2^j\right)^2 + 18\left(2^j\right)^2 \\
&= 7 \cdot 7^{j+1} - 6 \cdot 4^{j+1}.
\end{aligned} \tag{2.5}$$

This proves the claim. Now we note that equation (2.4) is $\mathcal{O}(7^k)$ and as $n = 2^k$

$$7^k = 7^{\log_2(n)} = \left(2^{\log_2(7)}\right)^{\log_2(n)} = \left(2^{\log_2(n)}\right)^{\log_2(7)} = n^{\log_2(7)}$$

it follows that

$$f(k) = \mathcal{O}(7^k) = \mathcal{O}(n^{\log_2(7)}) = \mathcal{O}(n^{2.81}).$$

Now in the case that n is not a power of 2 the result follows from the fact that we can bound n from below and from above by powers of 2. Denote $g(n)$ the algebraic complexity of multiplying two $n \times n$ matrices. As the number of computations is strictly increasing, because multiplying larger matrices requires more algebraic operations, we know that for some integer k

$$g(n) \leq f(k+1) = \mathcal{O}(n^{\log_2(7)})$$

So $g(n) = \mathcal{O}(n^{\log_2(7)})$ as well. □

Note that this can also be used for non-square matrices by zero-padding them to a size $2^{k+1} \times 2^{k+1}$ for some k.

2.3. Bilinear algorithms

Definition 2.3.1. We define ***bilinear algorithms*** as those that first create t products of linear combinations of A and linear combinations of B and then creates the elements of C as linear combinations of those products. The ***bilinear complexity*** is the number of products t.

$$\begin{aligned} M_1 &= (\text{linear combination of } a_{ij}\text{'s}) \cdot (\text{linear combination of } b_{ij}\text{'s}) \\ &\vdots \\ M_t &= (\text{linear combination of } a_{ij}\text{'s}) \cdot (\text{linear combination of } b_{ij}\text{'s}) \end{aligned}$$

And each c_{ij} is then a linear combination of these $M_1, \ldots, M_t$

It is easily validated that the standard algorithm and Strassen's are bilinear algorithms with bilinear complexities 8 and 7 respectively.

With this definition we can state a proposition that gives a way to improve the bound on ω by finding bilinear algorithms for multiplying $n \times n$ matrices with bilinear complexity t for some n and t. The proof for the following proposition is similar to that of proposition 2.2.1 as this proposition is a more general statement of it.

Proposition 2.3.2. Let m be a positive integer. Suppose that there exists a bilinear algorithm that computes the product of two $m \times m$ matrices with bilinear complexity t. Then

$$\omega \leq \log_m(t)$$

Proof. We consider the multiplication of two $n \times n$ matrices for arbitrary n. First consider the case where $n = m^k$ for some k. Let $f(k)$ be the algebraic complexity for multiplying two $m^k \times m^k$ matrices using this algorithm. Because there are t multiplications of blocks that have sizes $m^{k-1} \times m^{k-1}$ and C additions of these blocks, where C is bounded above by $3m^{k-1}$ that represents the case when all coefficients for the linear combinations of the elements of the blocks are nonzero, we obtain the recursive formula

$$f(k) = t \cdot f(k-1) + C\left(m^{k-1}\right)^2.$$

We claim that this is solved by

$$f(k) = \left(1 + \frac{C}{t - m^2}\right) t^k - \frac{C}{t - m^2} \cdot \left(m^2\right)^k \tag{2.6}$$

which we prove by induction.
By the assumptions we have that

$$\begin{aligned} f(1) &= t + C \\ &= \frac{(t-m^2)(t+C)}{t-m^2} \\ &= \frac{t^2 - tm^2 + Ct - Cm^2}{t-m^2} \\ &= t + \frac{Ct}{t-m^2} - \frac{Cm^2}{t-m^2} \\ &= \left(1 + \frac{C}{t-m^2}\right) t^1 - \frac{C}{t-m^2}\left(m^2\right)^1 \end{aligned}$$

satisfying (2.6) as a base case. Now suppose

$$f(j) = \left(1 + \frac{C}{t-m^2}\right) t^j - \frac{C}{t-m^2}\left(m^2\right)^j$$

for some $j \geq 1$. Then

$$\begin{aligned} f(j+1) &= t \cdot f(j) + C\left(m^j\right)^2 \\ &= t \cdot f(j) + C\left(m^2\right)^j \\ &= t\left(\left(1 + \frac{C}{t-m^2}\right) t^j - \frac{C}{t-m^2}\left(m^2\right)^j\right) + C\left(m^2\right)^j \\ &= \left(1 + \frac{C}{t-m^2}\right) t^{j+1} - \frac{t \cdot C\left(m^2\right)^j}{t-m^2} + \frac{C\left(m^2\right)^j}{t-m^2}(t-m^2) \qquad (2.7) \\ &= \left(1 + \frac{C}{t-m^2}\right) t^{j+1} + \frac{-t \cdot C\left(m^2\right)^j + t \cdot C\left(m^2\right)^j - C\left(m^2\right)^j m^2}{t-m^2} \\ &= \left(1 + \frac{C}{t-m^2}\right) t^{j+1} - \frac{C}{t-m^2}\left(m^2\right)^{j+1} \end{aligned}$$

This proves the claim. Now we note that equation 2.6 is $\mathcal{O}(t^k)$ and as $n = m^k$

$$t^k = t^{\log_m(n)} = \left(m^{\log_m(t)}\right)^{\log_m(n)} = \left(m^{\log_m(n)}\right)^{\log_m(t)} = n^{\log_m(t)}$$

it follows that

$$f(k) = \mathcal{O}(t^k) = \mathcal{O}(n^{\log_m(t)}).$$

Now in the case that n is not a power of m the result follows from the fact that we can bound n from below and from above by powers of m. Denote $g(n)$ the algebraic complexity of multiplying two $n \times n$ matrices. As the number of computations is strictly increasing, because multiplying larger matrices requires more algebraic operations, we know that for some integer k

$$g(n) \leq f(k+1) = \mathcal{O}(n^{\log_m(t)})$$

So $g(n) = \mathcal{O}(n^{\log_m(t)})$ as well. □

This can even be generalised to finding bilinear algorithms for non-square matrix multiplications, but first we will generalise the problem even further before stating one of the fundamental theorems for improving the bound on ω that also holds for non-square matrix multiplication.

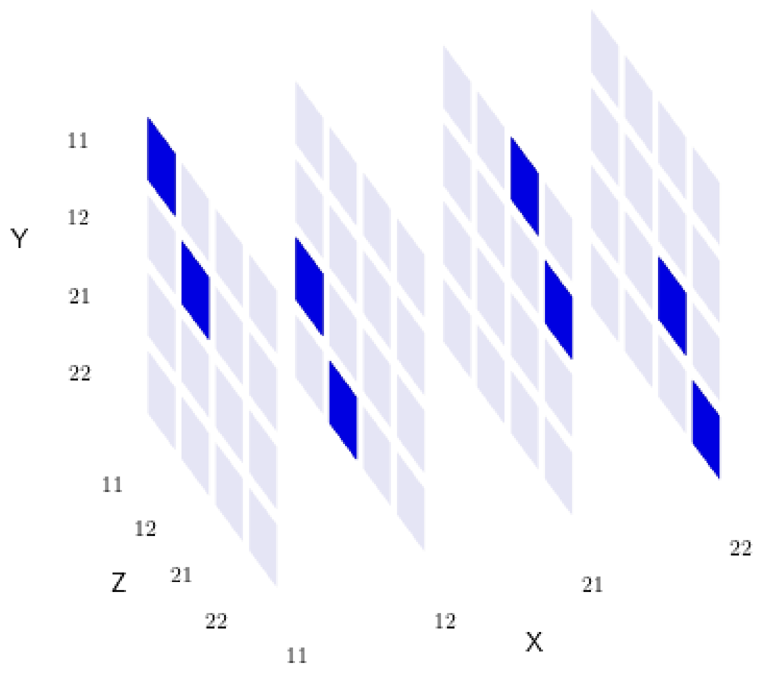

Figure 2.1: Matrix multiplication tensor <2,2,2>

2.4. Tensors

Definition 2.4.1. Given three finite-dimensional vector spaces U, V and W over the field $\mathbb{F}$. Take bases $\{x_1,\dots,x_{\dim(U)}\}$, $\{y_1,\dots,y_{\dim(V)}\}$ and $\{z_1,\dots,z_{\dim(W)}\}$ of U,V and W respectively. A ***tensor*** over (U,V,W) is an element of $U\otimes V\otimes W$, or a formal sum

$$T=\sum_{u=1}^{\dim(U)}\sum_{v=1}^{\dim(V)}\sum_{w=1}^{\dim(W)} d_{uvw}x_u\otimes y_v\otimes z_w$$

with coefficients $d_{uvw}\in\mathbb{F}$ for each $(u,v,w)\in\{1,\dots,\dim(U)\}\times\{1,\dots,\dim(V)\}\times\{1,\dots,\dim(W)\}$.

Definition 2.4.2. We define the following tensor as $\langle m,n,p\rangle$ and call it the matrix multiplication tensor.

$$\langle m,n,p\rangle=\sum_{i=1}^{m}\sum_{j=1}^{p}\sum_{k=1}^{n} a_{ik}\otimes b_{kj}\otimes c_{ij}$$

where a_{ik}, b_{kj} and c_{ij} are base vectors of vector spaces with dimensions mn, np and mp respectively.

We call this the matrix multiplication tensor because $T_{ijk}\in\{0,1\}$ indicates whether $A_{ik}\cdot B_{kj}$ contributes to C_{ij}.

Definition 2.4.3. A rank 1 tensor is one that can be written as the tensor product of elements of U,V and W. That is, there exists $(a,b,c)\in(U,V,W)$ such that

$$T=a\otimes b\otimes c.$$

Definition 2.4.4. Let T be a tensor over (U,V,W). The tensor rank of T, denoted $R(T)$, is defined as the minimal integer t such that T can be written as

$$T=\sum_{s=1}^{t}\left[\left(\sum_{u=1}^{\dim(U)}\alpha_{us}x_u\right)\otimes\left(\sum_{v=1}^{\dim(V)}\beta_{vs}y_v\right)\otimes\left(\sum_{w=1}^{\dim(W)}\gamma_{ws}z_w\right)\right]$$

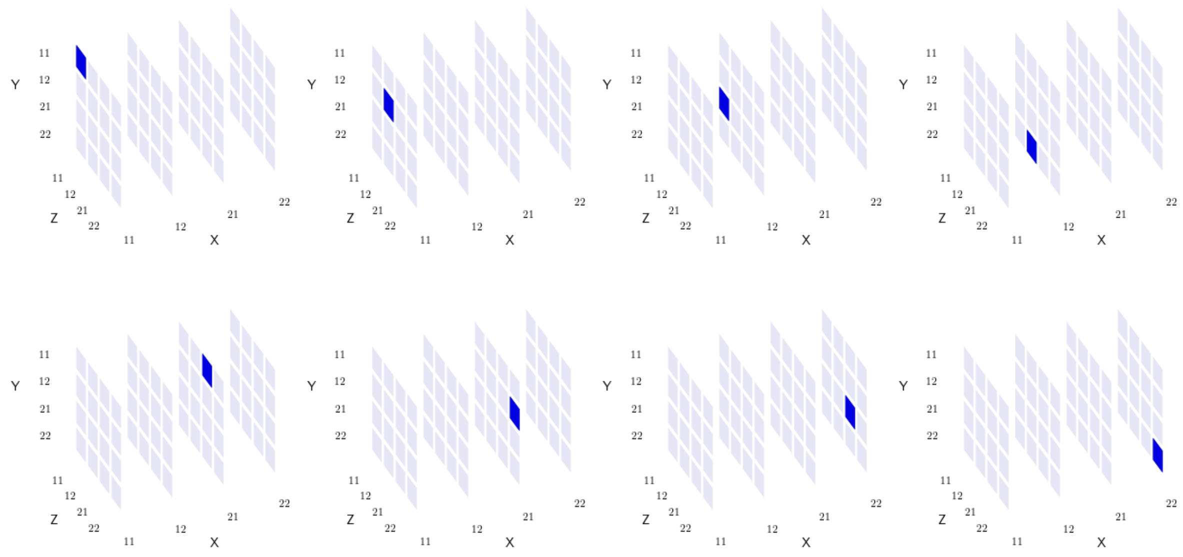

Figure 2.2: Standard matrix multiplication tensor decomposition

for some constants $\alpha_{us}, \beta_{vs}, \gamma_{ws} \in \mathbb{F}$ and where x_u, y_v, z_w are base vectors for U, V, W. Note this is the sum of t rank 1 tensors.

A tensor can also be written in terms of coefficient matrices. When considering a decomposition of an $n \times m \times p$ tensor T over (U, V, W) with rank t, one can define the coefficient matrices A, B and C of sizes $nm \times t$, $mp \times t$ and $np \times t$ respectively with columns $a(s), b(s)$ and $c(s)$ in U, V and W respectively, such that

$$T = \sum_{s=1}^{t} a(s) \otimes b(s) \otimes c(s).$$

The tensor representing the 2×2 matrix multiplication is given in Figure 2.1. In Figures 2.2 and 2.3 the tensor decompositions from the standard algorithm and that of Strassen are shown. Note how the sum of these rank 1 tensors add up to the $< 2, 2, 2 >$ tensor.

Note now that from a tensor rank decomposition one can make a bilinear algorithm with bilinear complexity precisely t. which means that in order to find a better bound on omega or find a faster multiplication algorithm one can find a lower rank tensor decomposition.

The coefficient matrices for Strassen's algorithm are as follows

$$A = \begin{bmatrix} 1 & 0 & 1 & 0 & 1 & -1 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 1 \\ 0 & 1 & 0 & 0 & 0 & 0 & -1 \\ 1 & 1 & 0 & 1 & 0 & 0 & -1 \end{bmatrix} B = \begin{bmatrix} 1 & 1 & 0 & -1 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 1 \\ 1 & 0 & -1 & 0 & 1 & 0 & 1 \end{bmatrix} C = \begin{bmatrix} 1 & 0 & 0 & 1 & -1 & 0 & 1 \\ 0 & 0 & 1 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 & 0 & 0 \\ 1 & -1 & 1 & 0 & 0 & 1 & 0 \end{bmatrix}$$

Note how the tensor products of corresponding columns correspond to a rank 1 tensor in the decomposition in figure 2.3.

Theorem 2.4.5. Let m, n, p, t be positive integers. If $R(\langle m, n, p \rangle) \le t$, then

$$(mnp)^{\omega/3} \le t$$

This theorem presented by Le Gall [17] is a generalisation of proposition 2.3.2 and shows that a decomposition for any matrix multiplication tensor (even non-square) with a lower rank may improve the upper bound on ω as

$$\omega \le 3 \log_{mnp}(t).$$

Using this theorem Strassen's result follows as well with $m = n = p = 2$ and $t = 7$.

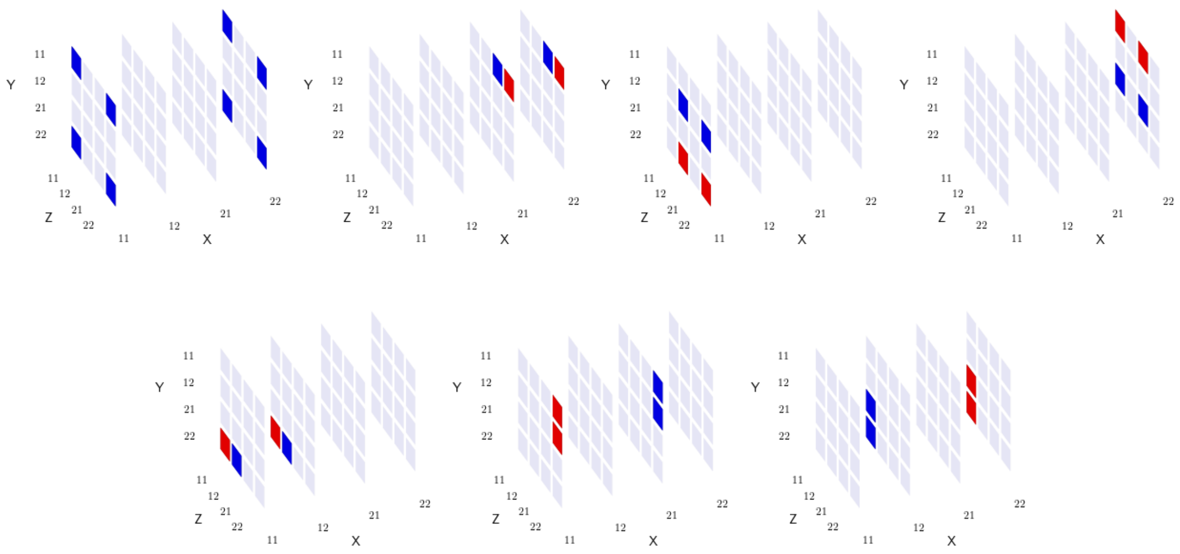

Figure 2.3: Strassen's tensor decomposition

3 Historical overview

In this chapter we present an overview of some of the methods/ techniques used to improve the bound on ω as well as techniques used to improve the bounds of some small tensors. As the focus of this paper is on bounding the ranks of small tensors we will not go into much detail with the presented results. We refer to the respective papers for further explanation on these complex and convoluted methods.

3.1. ω bound improvements

As stated in chapter 2 it all started in 1969 with Volker Strassen [26] when he showed that

$$\omega \leq \log_2(7) \leq 2.81.$$

10 years later Pan [20] constructed a new technique of trilinear operations of aggregating, uniting and cancelling with which he found a new bound on ω of

$$\omega \leq \log_{70}(143640) \leq 2.795.$$

A year later in 1979 Bini et al. [2] found an approximate 2×2 matrix multiplication where one of the elements is 0 with 5 multiplications. They then used this in some clever way to construct a proper 12×12 matrix multiplication algorithm with 1000 multiplications and thus showed that

$$\omega \leq \log_{12}(1000) \leq 2.780.$$

Then in the same year using Schönhage's asumptotic sum inequality [23], Pan [21] came back and showed that

$$\omega \leq 3\log_{436}(196) \leq 2.61.$$

Schönhage [23] used his theorem to show

$$\omega \leq 3\log_{110}(52) \leq 2.522.$$

Using the same construction as Schönhage but further optimised, Romani [22] found a slightly better bound

$$\omega \leq 2.52.$$

After this, in 1982, Coppersmith and Winograd [5] showed that there always exists a better algorithm for matrix multiplication that has a better asymptotic complexity. This means that ω is a limit point and improvements on the bound must always be possible. They then presented a new construction that creates new algorithms from old ones and showed

$$\omega \leq 2.496.$$

However, they stated that it seemed that this iterating process improves only marginally and therefore cannot be used indefinitely.

After this Strassen [25] came back in the picture and constructed a trilinear form which is not a matrix multiplication at all. however, as there are matrix multiplications in this form, by taking tensor powers and operating on the result he creates several disjoint matrix products and use a theorem by Schönhage to show

$$\omega \leq 2.4785.$$

This method will be called the laser method. Coppersmith and Winograd [6] built on this idea and find that

$$\omega \leq 2.376.$$

In 2011 Stothers [8] further builds on the results by the aforementioned authors and gives the next lower upper bound on ω

$$\omega \leq 2.37369.$$

The next to improve the bound is Williams [27] who developed an automated approach to design matrix multiplication algorithms based on constructions similar to Coppersmith and Winograd.

$$\omega \leq 2.37294.$$

In 2014 Le Gall [18] used a method based on convex optimisation and applied it to powers of the Winograd-Coppersmith construction to obtain the bound

$$\omega \leq 2.3729.$$

In 2021 Alman and Williams [1] refined Strassen's laser method to obtain a small improvement of

$$\omega \leq 2.37286.$$

This method however has some limitations as they showed that when applied to the Coppersmith-Winograd construction this cannot ever give a better bound than $\omega \leq 3725$.

Then this limit was obliterated by Duan et al. who used asymmetric hashing to prove

$$\omega \leq 2.371866.$$

In figure 3.1 we plotted a graph of all these improvements over time. It can be seen that the latest improvements are all very marginal, indicating how difficult it is to make any real improvements to lowering the upper bound om ω.

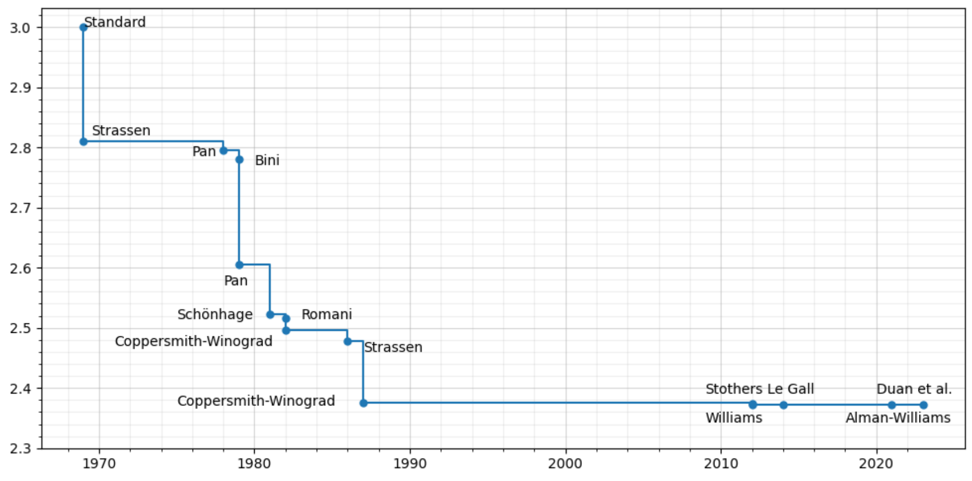

Figure 3.1: Upper bound on ω through the years

3.2. Bounding small tensors' rank

In this section we present an overview of some of the known bounds and results on small matrix multiplication tensors, where by small we mean matrix multiplication tensors like $\langle 2,2,2\rangle$ and $\langle 3,3,3\rangle$.

After Strassen found his rank 7 decomposition for $\langle 2,2,2\rangle$, it was Hopcraft [13] who showed that this is minimal, meaning that $\text{rank}(\langle 2,2,2\rangle) = 7$. They also showed that for any integer n the tensor $\langle 2,2,n\rangle$ has minimal rank is $\lceil \frac{7n}{2} \rceil$ as well as that the rank of $\langle 2,3,3\rangle$ is 15.

For $\langle 3,3,3\rangle$ the exact rank is still unknown. It was shown in 2003 by Bläser [3] that

$$19 \leq \text{rank}(\langle 3,3,3\rangle).$$

In 1976 Laderman [16] already showed that

$$\text{rank}(\langle 3,3,3\rangle) \leq 23$$

after finding an explicit decomposition by merging solutions of smaller systems of equations.

After this Johnson and McLoughlin [14] found two whole families of algorithms that are pairwise inequivalent to each other and Laderman's. They did this using the alternating least squares method (ALS). To find exact coefficients they used transformations of the coefficients to fix some of the coefficients and continue until all are rational and thus an exact solution is obtained.

In 2011 Courtois et al [7] converted the problem to a SAT problem and found a new unique algorithm within a day on a single CPU.

Then in 2013 Oh et al [19] devised a rounding method to enhance the rationality of solutions. The basis for this method is the ALS method as well. After some iterations they round some coeficients to rationals and continue from there. They also proposed a naming convention to easily distinguish solutions by looking at the distribution of rank 1, 2 or 3 matrices in the coefficients. They found up to 31 algorithms and showed inequivalence.

This converting to a SAT problem proved promising as in 2021 Heule et al [12] found 17,000 new algorithms with coefficients in $\{-1,0,1\}$ which means that these solutions work over any field.

Another approach by Kauers [15] was to use flip graphs with which he also found a new solution of rank 23 for $\langle 3,3,3\rangle$. They also improved the bound on $\langle 5,5,5\rangle$ from 98 found by[24] to 97.

The most recent innovation was made by Fawzi et al [11] when they converted the problem to a single-player game, TensorGame, and used machine learning to teach an AI to play this game. With this approach they were able to rediscover many algorithms and bounds and even lower the bound on $\langle 3,4,5\rangle$.

4 Methods

In this chapter we will explore the different methods we have tried to use to find new decompositions. As most methods presented in section 3.2 and our methods are based on ALS we will describe this algorithm in detail first. After that we consider the problem as an ILP problem and as a combination of these (AILP). Then we present our method with which we were able to rediscover many of the known best bounds for small tensors.

4.1. Alternating least squares

4.1.1. Algorithm description

The problem one attempts to solve is the following objective function: For a given tensor T one tries to find coefficient matrices A, B and C with columns $a(s), b(s)$ and $c(s)$ to minimise the error

$$\Upsilon(A,B,C) = ||T - \sum_{s=1}^{t} a(s) \otimes b(s) \otimes c(s)||_2^2.$$

Before stating the ALS method we first need some definitions.

Definition 4.1.1. The ***unfolded tensor*** $T_{KI \times J}$ of an $I \times J \times K$ tensor is given by

$$T_{KI\times J} = \begin{bmatrix} T_{::1} \\ \vdots \\ T_{::K} \end{bmatrix}.$$

Which is a stack of the $kI \times J$ slices of T. Similarly for $T_{IJ\times K}$ and $T_{JK\times I}$.

Definition 4.1.2. Let two matrices A and H be given be of sizes $I \times J$ and $K \times L$ respectively. Then the ***Kronecker product*** is defined as the $IK \times JL$ matrix:

$$A \otimes H = \begin{pmatrix} a_{11}H & a_{12}H & \cdots & \\ a_{21}H & a_{22}H & \cdots & \\ \vdots & \vdots & & \end{pmatrix}.$$

Definition 4.1.3. Let two matrices A and B be given be of sizes $I \times J$ and $K \times J$ respectively. Then the ***Khatri-Rao product*** is defined as the $IK \times J$ matrix:

$$A \odot B = \begin{pmatrix} a_1 \otimes b_1 & a_2 \otimes b_2 & \cdots & \end{pmatrix}.$$

Which is also referred to as the column-wise Kronecker product.

With these we can see how the ALS method works.

First we need to redefine the objective function. We do this with respect to A, B and C. These are all equivalent.

$$\begin{aligned}\Upsilon(A,B,C) &= ||T_{JK\times I} - (B\odot C)A^T||_2^2 \\ &= ||T_{KI\times J} - (C\odot A)B^T||_2^2 \\ &= ||T_{IJ\times K} - (A\odot B)C^T||_2^2\end{aligned} \tag{4.1}$$

We define the functions that find the coefficients that minimize this objective by

$$\begin{aligned}A^T &= (B\odot C)^\dagger T_{JK\times I} = f_A(B,C) \\ B^T &= (C\odot A)^\dagger T_{KI\times J} = f_B(C,A) \\ C^T &= (A\odot B)^\dagger T_{IJ\times K} = f_C(A,B).\end{aligned} \tag{4.2}$$

A description of the pseudocode for ALS is given below in Algorithm 1.

Algorithm 1 Alternating least squares

```
1: while True do
2:     Initialise $A, B, C$
3:     while True do
4:         $A \leftarrow f_A(B,C)$
5:         $B \leftarrow f_B(C,A)$
6:         $C \leftarrow f_C(A,B)$
7:         if Conergence then
8:             Break
9:         end if
10:    end while
11:    if Close enough then
12:        return $A, B, C$
13:    end if
14: end while
```

Now what ALS does is iteratively fix B and C and solves the least squares problem 4.1 for A using f_A and then similarly for B and then C. By doing so, hoping to converge to a good approximation for the tensor T.

More details for this method can be found in [4]

4.2. Integer linear programming

4.2.1. Method description

First we will state the non-linear minimisation problem before incorporating techniques to linearise the problem.

$$\text{Minimize} \sum_{i,j,k=1}^{N^2} \left| T_{ijk} - \sum_{s=1}^{t} A_{ir} B_{jr} C_{kr} \right|$$

subject to

$$A_{ir}, B_{jr}, C_{kr} \in \{-1, 0, 1\}.$$

That is, we want to find coefficient matrices A, B, C such that the tensor created from these is as close to a given tensor as possible. When this objective is exactly 0 we have found a decomposition of T with rank t.

To linearise this problem we introduce a few auxiliary variables and use linear constraints to model the non-linear problem.

We define new variables and write

$$A_{ir} = A^1_{ir}(2A^2_{ir} - 1)$$

$$A^1_{ir}, A^2_{ir} \in \{-0, 1\}$$

and similar for B and C. With this we can rewrite the product $A_{ir}B_{jr}C_{kr}$, introduce new variables and write

$$\begin{aligned} A_{ir}B_{jr}C_{kr} &= A^1_{ir}(2A^2_{ir} - 1)B^1_{jr}(2B^2_{jr} - 1)C^1_{kr}(2C^2_{kr} - 1) \\ &= \left(A^1_{ir}B^1_{jr}C^1_{kr}\right)\left((2A^2_{ir} - 1)(2B^2_{jr} - 1)(2C^2_{kr} - 1)\right) \\ &= X_{ijkr}(2Y_{ijkr} - 1) \\ &= Z_{ijkr}. \end{aligned}$$

This results in the following ILP

$$\text{Minimize} \sum_{i,j,k=1}^{n^2} E_{ijk}$$

subject to

$$
\begin{aligned}
E_{ijk} &\geq \left(\sum_{s=1}^{t} Z_{ijkr}\right) - T_{ijk} \\
E_{ijk} &\geq T_{ijk} - \left(\sum_{s=1}^{t} Z_{ijkr}\right) \\
Z_{ijkr} &\leq -X_{ijkr} + 2Y_{ijkr} \\
Z_{ijkr} &\leq X_{ijkr} \\
Z_{ijkr} &\geq X_{ijkr} + 2Y_{ijkr} - 2 \\
Z_{ijkr} &\geq -X_{ijkr} \\
Y_{ijkr} &\leq A^2_{ir} + B^2_{jr} + C^2_{kr} \\
Y_{ijkr} &\leq -A^2_{ir} - B^2_{jr} + C^2_{kr} + 2 \\
Y_{ijkr} &\leq A^2_{ir} - B^2_{jr} - C^2_{kr} + 2 \\
Y_{ijkr} &\leq -A^2_{ir} + B^2_{jr} - C^2_{kr} + 2 \\
Y_{ijkr} &\geq A^2_{ir} - B^2_{jr} - C^2_{kr} \\
Y_{ijkr} &\geq -A^2_{ir} + B^2_{jr} - C^2_{kr} \\
Y_{ijkr} &\geq -A^2_{ir} - B^2_{jr} + C^2_{kr} \\
Y_{ijkr} &\geq A^2_{ir} + B^2_{jr} + C^2_{kr} - 2 \\
X_{ijkr} &\leq A^1_{ir} \\
X_{ijkr} &\leq B^1_{jr} \\
X_{ijkr} &\leq C^1_{kr} \\
X_{ijkr} &\geq A^1_{ir} + B^1_{jr} + C^1_{kr} - 2 \\
A^1_{ir} &\in \{0,1\} \\
A^2_{ir} &\in \{0,1\} \\
B^1_{jr} &\in \{0,1\} \\
B^2_{jr} &\in \{0,1\} \\
C^1_{kr} &\in \{0,1\} \\
C^2_{kr} &\in \{0,1\} \\
X_{ijkr} &\in \{0,1\} \\
Y_{ijkr} &\in \{0,1\} \\
Z_{ijkr} &\in \{-1,0,1\} \\
E_{ijkr} &\in \{-t-1,0,t+1\}
\end{aligned}
$$

Where all inequalities are constructed such that all variables behave exactly as defined above.

4.2.2. Results

This problem was implemented in python and solved with the Gurobi solver.

This program could solve the $\langle 2,2,2\rangle$ tensor decomposition with rank 7 within approximately 10 minutes. This is equivalent to Strassen's algorithm as this was proven to be essentially unique by de Groote [9]. Searching for larger tensors took over 8 hours after which the program was stopped.

While this method does find exact solutions, it does so with exponentially increasing time with respect to the size of the tensor. Further optimisations for the number of variables and constraints, using a better solver or a faster computer could improve the capabilities of this approach. However, even with these improvements it is unlikely that significant progress can be made using this approach.

After this two other similar methods were attempted to combine ALS with ILP by means of an alternating ILP (AILP) and a quadratic AILP (QAILP). With AILP it works the same as ALS but instead of finding the least squares solution for the A system we solve an ILP where B and C have been kept fixed similar to ALS. Then the solution to this updates A and repeats for B and C alternating until convergence. This method did not have good convergence likely due to the fact that coefficients where jumping around too much as the problem is not continuous and thus we tried QAILP. This method linearised the problem for A and B and kept C constant and alternated with this. This gave more control to the optimisation and had better convergence, but took too long on larger tensors. Therefore we concluded that these methods did not seem very promising and we will not go in further detail with these.

4.3. Integer alternating least squares

4.3.1. Method description

This method has certain characteristics of the ALS method and gradient descent and produces integer solutions. It is similar to the procedures of Oh et al [19] and Johnson and McLoughlin[14] but is aimed to be a smoother approach than simply rounding and fixing values in order to find integer solutions near real ones.

The algorithm starts by initialising the coefficient matrices randomly. After that it fixes B and C and finds the A that brings the produced tensor closest to the desired tensor. It then takes a small step from A to this new A. Before updating A it pushes the values slightly towards integers and clamps the coefficients to the range $[-1,1]$. Then it repeats the same procedure for B and C and repeats until a stopping criterion is met. The pseudocode is shown in algorithm 2.

Algorithm 2 Integer alternating least squares (IALS)

```
while True do
    Initialise $A, B, C$
    while True do
        $\mu = g$(step count)
        $A_{\text{new}} \leftarrow$ Least squares$(B, C)$
        $A \leftarrow (1-\mu) \cdot A + \mu \cdot f(A_{\text{new}})$
        $B_{\text{new}} \leftarrow$ Least squares $(C, A)$
        $B \leftarrow (1-\mu) \cdot B + \mu \cdot f(B_{\text{new}})$
        $C_{\text{new}} \leftarrow$ Least squares $(A, B)$
        $C \leftarrow (1-\mu) \cdot A + \mu \cdot f(C_{\text{new}})$
        if Stop criterion then
            Break
        end if
    end while
    if Found then
        return $A, B, C$
    end if
end while
```

4.3.2. Implementation

By experimentation we found the following parameters to give the fastest and best results.

We say we found a solution if after rounding A, B, C to the nearest half we find that these perfectly decompose the desired tensor. Note that we don't round to integers because with the function f that we chose, there is an equilibrium at ± 0.5 and sometimes coefficients converge here.

The stop criterion is met when the following is true

$$\Delta A, \Delta B, \Delta C \leq \text{threshold}$$

and

$$i \leq \text{max steps}$$

where with the L_2 norm we define

$$\Delta A = \frac{||A_n - A_{n-1}||_2^2}{||A_{n-1}||_2^2}$$

$$\Delta B = \frac{||B_n - B_{n-1}||_2^2}{||B_{n-1}||_2^2}$$

$$\Delta C = \frac{||C_n - C_{n-1}||_2^2}{||C_{n-1}||_2^2}$$

$$f_\gamma(x) = x * (-4(1-\gamma)x^4 + 5(1-\gamma)x^2 + \gamma).$$

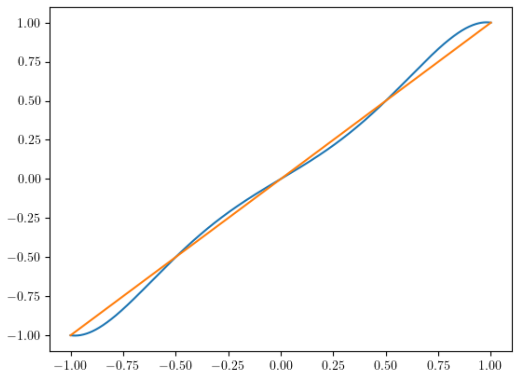

Figure 4.1: Function to push values towards integers displayed for $\gamma = 0.8$.

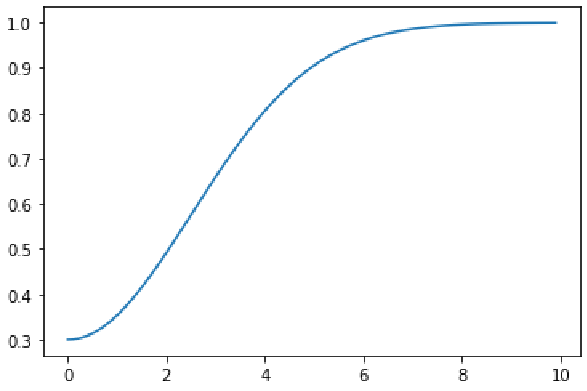

Figure 4.2: Step size over iterations $g_{0.08,0.7}$

and i is the number of steps from initialisation.

To push values towards integers we used the following function that is applied elementwise on the new coefficient matrices after clipping them to the range $[-1,1]$.

$$f_\gamma(x) = x * (-4(1-\gamma)x^4 + 5(1-\gamma)x^2 + \gamma).$$

This effect can be seen when compared to the graph of $y = x$ in figure 4.1.

For the step-size μ we found that increasing it over time yields better results than keeping it fixed. For this we choose the function

$$g_{\alpha,\beta}(i) = 1 - \beta \mathrm{e}^{-\alpha i^2}$$

which is evaluated at the i'th step. The graph is plotted in 4.2

5 Results

With the functions defined above we ran the program with the following parameters

$$\begin{aligned} &f_{0.9}(x)\\ &g_{0.08,0.7}(i)\\ \text{threshold} &= 0.0035\\ \text{max steps} &= 100 \end{aligned}$$

These parameters were found experimentally by looking for 100 rank 7 decompositions, starting from different seeds, of the $\langle 2,2,2\rangle$ tensor and seeing if increasing or decreasing a value changes the average runtime. These values seemed to be at least a local minimum.

With this we were able to find decompositions for all the tensors $\langle 2,2,2\rangle$, $\langle 2,2,3\rangle$, $\langle 2,2,4\rangle$, $\langle 2,3,3\rangle$, $\langle 2,2,5\rangle$, $\langle 3,3,3\rangle$ within a time frame ranging from tenths of seconds to 2 hours. In tables 5.2, 5.3, 5.4, 5.5, 5.7 we present some of our findings. In table 5.1 we present an overview of some of the best bounds discovered. With this method we did not find any better bounds for these tensors or any bound for larger tensors.

The coefficients are arranged in matrix forms. Specifically for the $\langle 3,3,3\rangle$ tensor the coefficients are arranged in 3×3 grids so that we can find the rank of these matrices. Our algorithm has 52 rank 1, 17 rank 2 and 0 rank 3 matrices. With this classification we can compare our algorithm with others based on the observation by [19] that different classifications are unique decompositions. They have also found an algorithm with the same distribution as ours. This does not mean they are equivalent, however we will not investigate this.

Table 5.1: Best known bounds

Tensor	Best known bound	Method	Our bound
$\langle 2,2,2\rangle$	7	Strassen [26]	7
$\langle 2,2,3\rangle$	11	$\langle 2,2,2\rangle + \langle 2,2,1\rangle$	11
$\langle 2,2,4\rangle$	14	$\langle 2,2,2\rangle + \langle 2,2,2\rangle$	14
$\langle 2,3,3\rangle$	15	Hopcraft and Kerr [13]	15
$\langle 2,2,5\rangle$	18	$(\langle 2,2,3\rangle + \langle 2,2,2\rangle$	18
$\langle 2,3,4\rangle$	20	Hopcraft and Kerr [13]	20
$\langle 3,3,3\rangle$	23	Laderman [16]	23

Table 5.2: Rank 11 decomposition of $\langle 2,2,3\rangle$.

s	A_s	B_s	C_s	s	A_s	B_s	C_s
1	$\begin{matrix}0 & 1\\ 0 & 0\end{matrix}$	$\begin{matrix}-1 & 0 & 0\\ -1 & 0 & 0\end{matrix}$	$\begin{matrix}-1 & 1 & 0\\ 0 & 0 & 0\end{matrix}$	7	$\begin{matrix}0 & 1\\ 0 & 0\end{matrix}$	$\begin{matrix}-1 & 0 & 0\\ -1 & 0 & 0\end{matrix}$	$\begin{matrix}-1 & 1 & 0\\ 0 & 0 & 0\end{matrix}$
2	$\begin{matrix}0 & 0\\ 0 & -1\end{matrix}$	$\begin{matrix}0 & 0 & 0\\ -1 & -1 & 0\end{matrix}$	$\begin{matrix}0 & 0 & 0\\ 1 & 0 & 0\end{matrix}$	8	$\begin{matrix}0 & 0\\ 0 & -1\end{matrix}$	$\begin{matrix}0 & 0 & 0\\ -1 & -1 & 0\end{matrix}$	$\begin{matrix}0 & 0 & 0\\ 1 & 0 & 0\end{matrix}$
3	$\begin{matrix}-1 & 0\\ -1 & 0\end{matrix}$	$\begin{matrix}0 & -1 & 1\\ 0 & 0 & 0\end{matrix}$	$\begin{matrix}0 & 0 & 0\\ 0 & 0 & -1\end{matrix}$	9	$\begin{matrix}-1 & 0\\ -1 & 0\end{matrix}$	$\begin{matrix}0 & -1 & 1\\ 0 & 0 & 0\end{matrix}$	$\begin{matrix}0 & 0 & 0\\ 0 & 0 & -1\end{matrix}$
4	$\begin{matrix}0 & 1\\ 0 & 0\end{matrix}$	$\begin{matrix}1 & 0 & 0\\ 1 & 1 & -1\end{matrix}$	$\begin{matrix}0 & 1 & 0\\ 0 & 0 & 0\end{matrix}$	10	$\begin{matrix}0 & 1\\ 0 & 0\end{matrix}$	$\begin{matrix}1 & 0 & 0\\ 1 & 1 & -1\end{matrix}$	$\begin{matrix}0 & 1 & 0\\ 0 & 0 & 0\end{matrix}$
5	$\begin{matrix}-1 & 1\\ -1 & 1\end{matrix}$	$\begin{matrix}0 & 0 & 0\\ 0 & 0 & 1\end{matrix}$	$\begin{matrix}0 & 0 & 0\\ -1 & 1 & 1\end{matrix}$	11	$\begin{matrix}-1 & 1\\ -1 & 1\end{matrix}$	$\begin{matrix}0 & 0 & 0\\ 0 & 0 & 1\end{matrix}$	$\begin{matrix}0 & 0 & 0\\ -1 & 1 & 1\end{matrix}$
6	$\begin{matrix}-1 & 1\\ -1 & 1\end{matrix}$	$\begin{matrix}0 & 0 & 0\\ 0 & 0 & 1\end{matrix}$	$\begin{matrix}0 & 0 & 0\\ -1 & 1 & 1\end{matrix}$				

Table 5.3: Rank 14 decomposition of $\langle 2,2,4\rangle$.

s	A_s	B_s	C_s	s	A_s	B_s	C_s
1	$\begin{matrix}0 & 0\\ 0 & 0\end{matrix}$	$\begin{matrix}1 & 1 & -1 & 0\\ 0 & -1 & 1 & 0\end{matrix}$	$\begin{matrix}1 & 0 & -1 & 0\\ 1 & 0 & -1 & 0\end{matrix}$	8	$\begin{matrix}0 & 0\\ 0 & 1\end{matrix}$	$\begin{matrix}-1 & 0 & -1 & -1\\ -1 & 0 & -1 & -1\end{matrix}$	$\begin{matrix}0 & 0 & 0 & 0\\ -1 & 1 & 1 & -1\end{matrix}$
2	$\begin{matrix}0 & 0\\ 1 & 1\end{matrix}$	$\begin{matrix}1 & 0 & 1 & 0\\ 0 & 0 & -1 & -1\end{matrix}$	$\begin{matrix}0 & 0 & 0 & 0\\ 0 & 0 & 1 & -1\end{matrix}$	9	$\begin{matrix}1 & 0\\ -1 & 0\end{matrix}$	$\begin{matrix}0 & 0 & 1 & 1\\ 0 & 0 & 1 & 1\end{matrix}$	$\begin{matrix}0 & 0 & 0 & 0\\ 1 & 0 & -1 & 0\end{matrix}$
3	$\begin{matrix}1 & 0\\ -1 & 1\end{matrix}$	$\begin{matrix}-1 & 0 & -1 & -1\\ 0 & 0 & 0-1 & -1\end{matrix}$	$\begin{matrix}0 & 0 & 0 & -1\\ 1 & -1 & -1 & 0\end{matrix}$	10	$\begin{matrix}-1 & 1\\ 1 & -1\end{matrix}$	$\begin{matrix}0 & 0 & 0 & 0\\ 0 & 0 & 1 & 1\end{matrix}$	$\begin{matrix}0 & 0 & 0 & 1\\ 0 & 0 & 0 & 0\end{matrix}$
4	$\begin{matrix}0 & 1\\ 0 & 0\end{matrix}$	$\begin{matrix}0 & 0 & 0 & 0\\ -1 & 0 & 0 & 0\end{matrix}$	$\begin{matrix}-1 & 1 & 1 & -1\\ 0 & 0 & 0 & 0\end{matrix}$	11	$\begin{matrix}1 & 0\\ 0 & 1\end{matrix}$	$\begin{matrix}1 & 0 & 1 & 0\\ 0 & 1 & -1 & 0\end{matrix}$	$\begin{matrix}0 & 1 & 1 & -1\\ 0 & 1 & 0 & 0\end{matrix}$
5	$\begin{matrix}0 & 0\\ 0 & 1\end{matrix}$	$\begin{matrix}-1 & 0 & -1 & 0\\ 1 & 0 & 1 & 0\end{matrix}$	$\begin{matrix}0 & 1 & 1 & -1\\ 0 & 1 & 1 & -1\end{matrix}$	12	$\begin{matrix}1 & 1\\ 0 & 0\end{matrix}$	$\begin{matrix}0 & 0 & 0 & 0\\ 0 & 0 & 1 & -1\end{matrix}$	$\begin{matrix}0 & 0 & -1 & 1\\ 0 & 0 & 0 & 0\end{matrix}$
6	$\begin{matrix}0 & 0\\ 1 & -1\end{matrix}$	$\begin{matrix}1 & 0 & 1 & 1\\ 0 & 0 & 0 & 0\end{matrix}$	$\begin{matrix}0 & 0 & 0 & 1\\ 0 & 0 & 0 & 1\end{matrix}$	13	$\begin{matrix}1 & 0\\ 0 & 0\end{matrix}$	$\begin{matrix}-1 & 1 & -1 & 1\\ 0 & -1 & 1 & 0\end{matrix}$	$\begin{matrix}-1 & 1 & 0 & 0\\ -1 & 1 & 0 & 0\end{matrix}$
7	$\begin{matrix}0 & 1\\ 0 & -1\end{matrix}$	$\begin{matrix}0 & 0 & 0 & 0\\ 1 & 1 & 0 & 0\end{matrix}$	$\begin{matrix}0 & 1 & 1 & -1\\ 0 & 0 & 0 & 0\end{matrix}$	14	$\begin{matrix}-1 & 0\\ 1 & 0\end{matrix}$	$\begin{matrix}1 & 1 & 1 & 1\\ 0 & 0 & 1 & 1\end{matrix}$	$\begin{matrix}0 & 0 & 0 & 0\\ 0 & 1 & 0 & 0\end{matrix}$

Table 5.4: Rank 15 decomposition of $\langle 2,3,3\rangle$.

s	A_s	B_s	C_s	s	A_s	B_s	C_s
1	$\begin{matrix} 0 & 1 & 0 \\ 0 & -1 & 0 \end{matrix}$	$\begin{matrix} -1 & 0 & 0 \\ 1 & 0 & 0 \\ 1 & 0 & 0 \end{matrix}$	$\begin{matrix} 1 & 0 & 0 \\ 0 & 0 & 0 \end{matrix}$	9	$\begin{matrix} 0 & 1 & 0 \\ 0 & -1 & 0 \end{matrix}$	$\begin{matrix} -1 & 0 & 0 \\ 1 & 0 & 0 \\ 1 & 0 & 0 \end{matrix}$	$\begin{matrix} 1 & 0 & 0 \\ 0 & 0 & 0 \end{matrix}$
2	$\begin{matrix} 0 & 1 & -1 \\ 0 & 0 & 0 \end{matrix}$	$\begin{matrix} 0 & 0 & 0 \\ 0 & 1 & -1 \\ 0 & 0 & 0 \end{matrix}$	$\begin{matrix} -1 & 1 & 0 \\ -1 & 1 & 0 \end{matrix}$	10	$\begin{matrix} 0 & 1 & -1 \\ 0 & 0 & 0 \end{matrix}$	$\begin{matrix} 0 & 0 & 0 \\ 0 & 1 & -1 \\ 0 & 0 & 0 \end{matrix}$	$\begin{matrix} -1 & 1 & 0 \\ -1 & 1 & 0 \end{matrix}$
3	$\begin{matrix} 1 & 1 & 0 \\ 1 & 1 & 0 \end{matrix}$	$\begin{matrix} 0 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & 0 & 0 \end{matrix}$	$\begin{matrix} -1 & 1 & 1 \\ -1 & 1 & 1 \end{matrix}$	11	$\begin{matrix} 1 & 1 & 0 \\ 1 & 1 & 0 \end{matrix}$	$\begin{matrix} 0 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & 0 & 0 \end{matrix}$	$\begin{matrix} -1 & 1 & 1 \\ -1 & 1 & 1 \end{matrix}$
4	$\begin{matrix} 0 & 0 & 0 \\ 1 & 0 & 0 \end{matrix}$	$\begin{matrix} 1 & 1 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{matrix}$	$\begin{matrix} 0 & -1 & -1 \\ 0 & 1 & 1 \end{matrix}$	12	$\begin{matrix} 0 & 0 & 0 \\ 1 & 0 & 0 \end{matrix}$	$\begin{matrix} 1 & 1 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{matrix}$	$\begin{matrix} 0 & -1 & -1 \\ 0 & 1 & 1 \end{matrix}$
5	$\begin{matrix} 0 & 0 & 0 \\ 0 & 0 & 0 \end{matrix}$	$\begin{matrix} -1 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{matrix}$	$\begin{matrix} -1 & 1 & 1 \\ 1 & -1 & -1 \end{matrix}$	13	$\begin{matrix} 0 & 0 & 0 \\ 0 & 0 & 0 \end{matrix}$	$\begin{matrix} -1 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{matrix}$	$\begin{matrix} -1 & 1 & 1 \\ 1 & -1 & -1 \end{matrix}$
6	$\begin{matrix} 0 & -1 & 1 \\ 0 & 1 & -1 \end{matrix}$	$\begin{matrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 1 & 0 & 0 \end{matrix}$	$\begin{matrix} 0 & 0 & 0 \\ -1 & 1 & 0 \end{matrix}$	14	$\begin{matrix} 0 & -1 & 1 \\ 0 & 1 & -1 \end{matrix}$	$\begin{matrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 1 & 0 & 0 \end{matrix}$	$\begin{matrix} 0 & 0 & 0 \\ -1 & 1 & 0 \end{matrix}$
7	$\begin{matrix} 1 & 0 & 1 \\ 1 & 0 & 1 \end{matrix}$	$\begin{matrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 1 \end{matrix}$	$\begin{matrix} 0 & 0 & 0 \\ 0 & 0 & 1 \end{matrix}$	15	$\begin{matrix} 1 & 0 & 1 \\ 1 & 0 & 1 \end{matrix}$	$\begin{matrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 1 \end{matrix}$	$\begin{matrix} 0 & 0 & 0 \\ 0 & 0 & 1 \end{matrix}$
8	$\begin{matrix} 1 & 0 & 1 \\ 1 & 0 & 1 \end{matrix}$	$\begin{matrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 1 \end{matrix}$	$\begin{matrix} 0 & 0 & 0 \\ 0 & 0 & 1 \end{matrix}$				

Table 5.5: Rank 18 decomposition of $\langle 2,2,5\rangle$.

s	A_s	B_s	C_s	s	A_s	B_s	C_s
1	$\begin{matrix} 1 & 0 \\ 0 & 0 \end{matrix}$	$\begin{matrix} 1 & 0 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 & 0 \end{matrix}$	$\begin{matrix} 0 & -1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 \end{matrix}$	10	$\begin{matrix} 1 & 0 \\ 0 & 0 \end{matrix}$	$\begin{matrix} 1 & 0 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 & 0 \end{matrix}$	$\begin{matrix} 0 & -1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 \end{matrix}$
2	$\begin{matrix} 1 & -1 \\ 0 & 1 \end{matrix}$	$\begin{matrix} -1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & -1 \end{matrix}$	$\begin{matrix} -1 & 0 & 0 & 0 & 0 \\ -1 & 0 & 1 & 0 & 0 \end{matrix}$	11	$\begin{matrix} 1 & -1 \\ 0 & 1 \end{matrix}$	$\begin{matrix} -1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & -1 \end{matrix}$	$\begin{matrix} -1 & 0 & 0 & 0 & 0 \\ -1 & 0 & 1 & 0 & 0 \end{matrix}$
3	$\begin{matrix} 1 & -1 \\ -1 & 1 \end{matrix}$	$\begin{matrix} -1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{matrix}$	$\begin{matrix} 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & -1 & -1 & 0 \end{matrix}$	12	$\begin{matrix} 1 & -1 \\ -1 & 1 \end{matrix}$	$\begin{matrix} -1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{matrix}$	$\begin{matrix} 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & -1 & -1 & 0 \end{matrix}$
4	$\begin{matrix} 0 & 0 \\ 0 & -1 \end{matrix}$	$\begin{matrix} 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \end{matrix}$	$\begin{matrix} 1 & -1 & -1 & 0 & -1 \\ 1 & -1 & -1 & 0 & -1 \end{matrix}$	13	$\begin{matrix} 0 & 0 \\ 0 & -1 \end{matrix}$	$\begin{matrix} 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \end{matrix}$	$\begin{matrix} 1 & -1 & -1 & 0 & -1 \\ 1 & -1 & -1 & 0 & -1 \end{matrix}$
5	$\begin{matrix} -1 & 0 \\ 0 & 0 \end{matrix}$	$\begin{matrix} 0 & 0 & -1 & 0 & 1 \\ 0 & 0 & -1 & 0 & 1 \end{matrix}$	$\begin{matrix} 0 & 0 & 0 & 0 & -1 \\ 0 & 0 & 0 & 0 & 0 \end{matrix}$	14	$\begin{matrix} -1 & 0 \\ 0 & 0 \end{matrix}$	$\begin{matrix} 0 & 0 & -1 & 0 & 1 \\ 0 & 0 & -1 & 0 & 1 \end{matrix}$	$\begin{matrix} 0 & 0 & 0 & 0 & -1 \\ 0 & 0 & 0 & 0 & 0 \end{matrix}$
6	$\begin{matrix} 0 & 0 \\ 0 & 1 \end{matrix}$	$\begin{matrix} 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & -1 \end{matrix}$	$\begin{matrix} 0 & 0 & -1 & 0 & -1 \\ 0 & 0 & -1 & 0 & -1 \end{matrix}$	15	$\begin{matrix} 0 & 0 \\ 0 & 1 \end{matrix}$	$\begin{matrix} 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & -1 \end{matrix}$	$\begin{matrix} 0 & 0 & -1 & 0 & -1 \\ 0 & 0 & -1 & 0 & -1 \end{matrix}$
7	$\begin{matrix} 1 & 0 \\ 0 & 0 \end{matrix}$	$\begin{matrix} -1 & 0 & -1 & 1 & 0 \\ -1 & 0 & -1 & 1 & 0 \end{matrix}$	$\begin{matrix} 0 & -1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{matrix}$	16	$\begin{matrix} 1 & 0 \\ 0 & 0 \end{matrix}$	$\begin{matrix} -1 & 0 & -1 & 1 & 0 \\ -1 & 0 & -1 & 1 & 0 \end{matrix}$	$\begin{matrix} 0 & -1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{matrix}$
8	$\begin{matrix} 0 & 0 \\ 1 & 0 \end{matrix}$	$\begin{matrix} 1 & -1 & 1 & 0 & -1 \\ 0 & 0 & 0 & 0 & 0 \end{matrix}$	$\begin{matrix} 0 & 0 & 0 & 0 & 0 \\ 0 & -1 & -1 & 1 & -1 \end{matrix}$	17	$\begin{matrix} 0 & 0 \\ 1 & 0 \end{matrix}$	$\begin{matrix} 1 & -1 & 1 & 0 & -1 \\ 0 & 0 & 0 & 0 & 0 \end{matrix}$	$\begin{matrix} 0 & 0 & 0 & 0 & 0 \\ 0 & -1 & -1 & 1 & -1 \end{matrix}$
9	$\begin{matrix} 0 & 1 \\ 0 & -1 \end{matrix}$	$\begin{matrix} -1 & 0 & 0 & 0 & 0 \\ -1 & 0 & 0 & 0 & 0 \end{matrix}$	$\begin{matrix} -1 & 1 & 1 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 \end{matrix}$	18	$\begin{matrix} 0 & 1 \\ 0 & -1 \end{matrix}$	$\begin{matrix} -1 & 0 & 0 & 0 & 0 \\ -1 & 0 & 0 & 0 & 0 \end{matrix}$	$\begin{matrix} -1 & 1 & 1 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 \end{matrix}$

Table 5.6: Rank 20 decomposition of $\langle 2,3,4\rangle$.

s	A_s	B_s	C_s	s	A_s	B_s	C_s
1	$\begin{matrix} 0 & 0 & 0 \\ 0 & 1 & 0 \end{matrix}$	$\begin{matrix} 0 & 0 & 0 & 1 \\ 1 & 0 & 1 & 1 \\ 0 & 0 & 0 & -1 \end{matrix}$	$\begin{matrix} 0 & 0 & 0 & 0 \\ -1 & 1 & 1 & 1 \end{matrix}$	11	$\begin{matrix} 0 & 0 & 0 \\ 0 & 1 & 0 \end{matrix}$	$\begin{matrix} 0 & 0 & 0 & 1 \\ 1 & 0 & 1 & 1 \\ 0 & 0 & 0 & -1 \end{matrix}$	$\begin{matrix} 0 & 0 & 0 & 0 \\ -1 & 1 & 1 & 1 \end{matrix}$
2	$\begin{matrix} -1 & 0 & -1 \\ 0 & 0 & 0 \end{matrix}$	$\begin{matrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & -1 \end{matrix}$	$\begin{matrix} 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \end{matrix}$	12	$\begin{matrix} -1 & 0 & -1 \\ 0 & 0 & 0 \end{matrix}$	$\begin{matrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & -1 \end{matrix}$	$\begin{matrix} 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \end{matrix}$
3	$\begin{matrix} 0 & 0 & 0 \\ 0 & 1 & 0 \end{matrix}$	$\begin{matrix} 0 & 0 & 0 & 0 \\ -1 & 0 & -1 & 0 \\ 0 & 0 & 0 & 0 \end{matrix}$	$\begin{matrix} 0 & 0 & 0 & 0 \\ -1 & 0 & 0 & 1 \end{matrix}$	13	$\begin{matrix} 0 & 0 & 0 \\ 0 & 1 & 0 \end{matrix}$	$\begin{matrix} 0 & 0 & 0 & 0 \\ -1 & 0 & -1 & 0 \\ 0 & 0 & 0 & 0 \end{matrix}$	$\begin{matrix} 0 & 0 & 0 & 0 \\ -1 & 0 & 0 & 1 \end{matrix}$
4	$\begin{matrix} 0 & 0 & 0 \\ -1 & 0 & 0 \end{matrix}$	$\begin{matrix} -1 & 0 & -1 & 0 \\ 0 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 \end{matrix}$	$\begin{matrix} 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 \end{matrix}$	14	$\begin{matrix} 0 & 0 & 0 \\ -1 & 0 & 0 \end{matrix}$	$\begin{matrix} -1 & 0 & -1 & 0 \\ 0 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 \end{matrix}$	$\begin{matrix} 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 \end{matrix}$
5	$\begin{matrix} 1 & -1 & 0 \\ 0 & 0 & 0 \end{matrix}$	$\begin{matrix} 0 & -1 & 0 & 1 \\ 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & -1 \end{matrix}$	$\begin{matrix} 0 & -1 & 0 & 0 \\ 0 & -1 & 0 & 0 \end{matrix}$	15	$\begin{matrix} 1 & -1 & 0 \\ 0 & 0 & 0 \end{matrix}$	$\begin{matrix} 0 & -1 & 0 & 1 \\ 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & -1 \end{matrix}$	$\begin{matrix} 0 & -1 & 0 & 0 \\ 0 & -1 & 0 & 0 \end{matrix}$
6	$\begin{matrix} 0 & -1 & 0 \\ 0 & 0 & 0 \end{matrix}$	$\begin{matrix} 0 & 0 & 0 & 0 \\ -1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{matrix}$	$\begin{matrix} 1 & 0 & 0 & -1 \\ 0 & 0 & 0 & 0 \end{matrix}$	16	$\begin{matrix} 0 & -1 & 0 \\ 0 & 0 & 0 \end{matrix}$	$\begin{matrix} 0 & 0 & 0 & 0 \\ -1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{matrix}$	$\begin{matrix} 1 & 0 & 0 & -1 \\ 0 & 0 & 0 & 0 \end{matrix}$
7	$\begin{matrix} -1 & 0 & 0 \\ 1 & 0 & 0 \end{matrix}$	$\begin{matrix} 0 & 1 & -1 & -1 \\ -1 & 1 & -1 & -1 \\ 0 & -1 & 0 & 1 \end{matrix}$	$\begin{matrix} 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 \end{matrix}$	17	$\begin{matrix} -1 & 0 & 0 \\ 1 & 0 & 0 \end{matrix}$	$\begin{matrix} 0 & 1 & -1 & -1 \\ -1 & 1 & -1 & -1 \\ 0 & -1 & 0 & 1 \end{matrix}$	$\begin{matrix} 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 \end{matrix}$
8	$\begin{matrix} 1 & -1 & 0 \\ -1 & 0 & 0 \end{matrix}$	$\begin{matrix} 0 & 1 & 0 & -1 \\ -1 & 1 & -1 & -1 \\ 0 & -1 & 0 & 1 \end{matrix}$	$\begin{matrix} 0 & 0 & 1 & 0 \\ 0 & -1 & 0 & 0 \end{matrix}$	18	$\begin{matrix} 1 & -1 & 0 \\ -1 & 0 & 0 \end{matrix}$	$\begin{matrix} 0 & 1 & 0 & -1 \\ -1 & 1 & -1 & -1 \\ 0 & -1 & 0 & 1 \end{matrix}$	$\begin{matrix} 0 & 0 & 1 & 0 \\ 0 & -1 & 0 & 0 \end{matrix}$
9	$\begin{matrix} 0 & -1 & 0 \\ 0 & 0 & 0 \end{matrix}$	$\begin{matrix} 0 & -1 & 0 & 1 \\ 1 & -1 & 0 & 1 \\ 0 & 1 & 0 & -1 \end{matrix}$	$\begin{matrix} 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 \end{matrix}$	19	$\begin{matrix} 0 & -1 & 0 \\ 0 & 0 & 0 \end{matrix}$	$\begin{matrix} 0 & -1 & 0 & 1 \\ 1 & -1 & 0 & 1 \\ 0 & 1 & 0 & -1 \end{matrix}$	$\begin{matrix} 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 \end{matrix}$
10	$\begin{matrix} 0 & 0 & -1 \\ 0 & 0 & 0 \end{matrix}$	$\begin{matrix} -1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 \end{matrix}$	$\begin{matrix} -1 & 0 & 1 & 0 \\ -1 & 0 & 1 & 0 \end{matrix}$	20	$\begin{matrix} 0 & 0 & -1 \\ 0 & 0 & 0 \end{matrix}$	$\begin{matrix} -1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 \end{matrix}$	$\begin{matrix} -1 & 0 & 1 & 0 \\ -1 & 0 & 1 & 0 \end{matrix}$

Table 5.7: A [52, 17, 0]-algorithm for ⟨3, 3, 3⟩

s	A_s	B_s	C_s	s	A_s	B_s	C_s
1	$\begin{matrix}0&1&0\\0&0&0\\0&1&0\end{matrix}$	$\begin{matrix}0&0&0\\0&1&0\\0&0&0\end{matrix}$	$\begin{matrix}-1&1&-1\\-1&1&-1\\0&0&0\end{matrix}$	13	$\begin{matrix}0&0&1\\1&0&0\\0&0&0\end{matrix}$	$\begin{matrix}0&1&0\\0&0&0\\-1&0&1\end{matrix}$	$\begin{matrix}0&0&1\\0&1&0\\0&0&0\end{matrix}$
2	$\begin{matrix}0&0&1\\0&0&0\\0&0&1\end{matrix}$	$\begin{matrix}0&0&0\\0&0&0\\0&1&0\end{matrix}$	$\begin{matrix}0&0&0\\0&0&0\\0&1&0\end{matrix}$	14	$\begin{matrix}0&0&0\\1&-1&0\\-1&1&0\end{matrix}$	$\begin{matrix}1&0&0\\0&0&0\\0&0&0\end{matrix}$	$\begin{matrix}1&0&0\\0&0&0\\-1&0&0\end{matrix}$
3	$\begin{matrix}0&-1&0\\1&0&0\\0&0&0\end{matrix}$	$\begin{matrix}1&0&0\\1&0&-1\\0&0&0\end{matrix}$	$\begin{matrix}0&0&1\\1&0&1\\1&0&0\end{matrix}$	15	$\begin{matrix}0&0&0\\0&0&0\\1&0&0\end{matrix}$	$\begin{matrix}0&0&1\\0&0&0\\0&0&0\end{matrix}$	$\begin{matrix}0&0&0\\0&0&0\\0&0&1\end{matrix}$
4	$\begin{matrix}1&0&0\\0&0&0\\1&0&0\end{matrix}$	$\begin{matrix}0&1&0\\0&0&0\\0&0&0\end{matrix}$	$\begin{matrix}0&0&0\\0&0&0\\0&1&0\end{matrix}$	16	$\begin{matrix}0&0&0\\0&0&0\\0&0&1\end{matrix}$	$\begin{matrix}0&0&0\\0&0&1\\0&0&1\end{matrix}$	$\begin{matrix}0&0&0\\0&0&0\\0&0&1\end{matrix}$
5	$\begin{matrix}0&-1&1\\0&1&-1\\0&-1&1\end{matrix}$	$\begin{matrix}0&0&0\\0&0&0\\1&0&0\end{matrix}$	$\begin{matrix}0&0&0\\-1&0&-1\\0&0&0\end{matrix}$	17	$\begin{matrix}0&0&0\\1&0&0\\0&0&0\end{matrix}$	$\begin{matrix}1&0&-1\\1&0&-1\\-1&0&1\end{matrix}$	$\begin{matrix}0&0&-1\\0&0&-1\\0&0&0\end{matrix}$
6	$\begin{matrix}0&-1&0\\0&1&0\\0&-1&1\end{matrix}$	$\begin{matrix}0&0&0\\0&0&1\\1&0&0\end{matrix}$	$\begin{matrix}0&0&0\\1&0&1\\1&0&0\end{matrix}$	18	$\begin{matrix}0&-1&0\\0&1&0\\0&0&0\end{matrix}$	$\begin{matrix}0&0&0\\0&1&0\\0&0&0\end{matrix}$	$\begin{matrix}1&-1&1\\1&0&1\\0&1&0\end{matrix}$
7	$\begin{matrix}0&0&0\\-1&1&0\\0&0&0\end{matrix}$	$\begin{matrix}0&0&0\\1&0&-1\\0&0&0\end{matrix}$	$\begin{matrix}0&0&0\\1&0&0\\1&0&0\end{matrix}$	19	$\begin{matrix}0&0&0\\0&0&0\\0&1&-1\end{matrix}$	$\begin{matrix}0&0&0\\0&0&1\\0&0&0\end{matrix}$	$\begin{matrix}0&0&0\\1&0&1\\1&0&1\end{matrix}$
8	$\begin{matrix}0&1&1\\0&-1&0\\0&1&0\end{matrix}$	$\begin{matrix}1&0&0\\1&1&0\\1&0&0\end{matrix}$	$\begin{matrix}1&-1&1\\1&-1&1\\0&1&0\end{matrix}$	20	$\begin{matrix}1&0&0\\-1&0&0\\0&0&0\end{matrix}$	$\begin{matrix}0&1&1\\0&0&0\\0&0&0\end{matrix}$	$\begin{matrix}0&0&1\\0&0&0\\0&0&0\end{matrix}$
9	$\begin{matrix}0&0&0\\1&0&1\\0&0&0\end{matrix}$	$\begin{matrix}0&0&0\\0&0&0\\1&0&-1\end{matrix}$	$\begin{matrix}0&0&0\\0&1&-1\\0&0&0\end{matrix}$	21	$\begin{matrix}1&-1&0\\-1&1&0\\1&-1&0\end{matrix}$	$\begin{matrix}1&0&0\\0&0&0\\0&0&0\end{matrix}$	$\begin{matrix}1&0&0\\0&0&0\\0&0&0\end{matrix}$
10	$\begin{matrix}-1&0&-1\\0&0&0\\0&0&0\end{matrix}$	$\begin{matrix}0&-1&0\\0&0&0\\0&0&0\end{matrix}$	$\begin{matrix}0&1&-1\\0&0&0\\0&-1&0\end{matrix}$	22	$\begin{matrix}0&-1&0\\0&1&0\\0&-1&0\end{matrix}$	$\begin{matrix}1&0&0\\1&0&0\\1&0&0\end{matrix}$	$\begin{matrix}1&-1&1\\1&-1&1\\-1&1&0\end{matrix}$
11	$\begin{matrix}0&0&1\\0&0&0\\0&0&0\end{matrix}$	$\begin{matrix}1&-1&0\\1&1&0\\1&1&0\end{matrix}$	$\begin{matrix}0&1&0\\0&1&0\\0&-1&0\end{matrix}$	23	$\begin{matrix}0&0&-1\\0&0&1\\0&0&0\end{matrix}$	$\begin{matrix}0&0&0\\0&0&0\\-1&1&1\end{matrix}$	$\begin{matrix}0&0&0\\0&1&0\\0&0&0\end{matrix}$
12	$\begin{matrix}0&1&-1\\0&0&0\\0&0&0\end{matrix}$	$\begin{matrix}1&0&0\\1&1&0\\0&0&0\end{matrix}$	$\begin{matrix}1&0&1\\1&0&1\\0&0&0\end{matrix}$				

6 Conclusion

As we have seen in chapter 3 researchers were able to find thousands of algorithms using SAT solvers. Others found a handful of algorithms by rounding extensions of ALS. We tried to devise a new method that extended ALS by pushing values slowly towards integers or rational numbers. Our method found decompositions for small tensors from $\langle 2,2,2 \rangle$ to $\langle 3,3,3 \rangle$, $\langle 2,2,5 \rangle$ and $\langle 2,3,4 \rangle$ that give the same bounds as earlier found results. We were not able to find any new bounds.

Although our method did not result in a new decompositions, it was able to find one for $\langle 3,3,3 \rangle$, this may be equivalent to another decomposition found by [19].

We found that the speed of finding results relies heavily on the parameters for the functions f and g and the functions themselves. The values used here were found by trial and error. The functions themselves as well. As the speed seemed heavily dependent on these, further research could be done to find better functions that make convergence faster and push towards integers faster. This might yield faster, better and more results. However we suspect this may still not be better to outperform the SAT solving methods. Again due to the complexity we chose not to go in much detail with these SAT solving methods, however these seem to be very promising.

Another method we think might be interesting is another rounding scheme. This start from a real solution and tries to round the solution in such a way that it may find an integer or rational solution. This would start by rounding the coefficient that is the closest to an integer. This in turn messes up the solution and so to find the next coefficient to round it looks for the coefficient that is closest to an integer but also restores a bit of the error made by the first rounding. Something like this continues until all coefficients are rounded and then this may result in a valid solution. Because of the complicated nature of this algorithm we did not attempt this as we wanted to find a simple algorithm.

For the $\langle 3,3,3 \rangle$ tensor it is our suspicion that there may not exist a rank 22 decomposition. Therefore we would recommend research to focus more on finding a tighter lower bound to prove that it indeed might not exist.

A

IALS.py

```python
import numpy as np
from scipy import linalg
import sys

import Tensors


def f_A(B, C, T):
    B_khatri_rao_C = linalg.khatri_rao(B, C)
    T_JKxI = np.vstack([T[:, j, :].transpose() for j in range(T.shape[1])])
    B_khatri_rao_C_pseudo_inverse = np.linalg.pinv(B_khatri_rao_C)

    res = np.matmul(B_khatri_rao_C_pseudo_inverse, T_JKxI).transpose()

    return res

def f_B(C, A, T):
    C_khatri_rao_A = linalg.khatri_rao(C, A)
    T_KIxJ = np.vstack([T[:, :, k] for k in range(T.shape[2])])
    C_khatri_rao_A_pseudo_inverse = np.linalg.pinv(C_khatri_rao_A)

    res = np.matmul(C_khatri_rao_A_pseudo_inverse, T_KIxJ).transpose()

    return res

def f_C(A, B, T):
    A_khatri_rao_B = linalg.khatri_rao(A, B)
    T_IJxK = np.vstack([T[i, :, :] for i in range(T.shape[0])])
    A_khatri_rao_B_pseudo_inverse = np.linalg.pinv(A_khatri_rao_B)

    res = np.matmul(A_khatri_rao_B_pseudo_inverse, T_IJxK).transpose()

    return res



def ALS(tensor, A, B, C, update_rule, step, max_iterations, stop_criterion):
    As = []
    Bs = []
    Cs = []

    for i in range(max_iterations):
        mu = step(i)

        new_A = f_A(B, C, tensor)
        A = update_rule(A, new_A, mu)

        new_B = f_B(C, A, tensor)
        B = update_rule(B, new_B, mu)
        new_C = f_C(A, B, tensor)
        C = update_rule(C, new_C, mu)

        As.append(A)
        Bs.append(B)
        Cs.append(C)

        if stop_criterion(As, Bs, Cs):
            break

    return As, Bs, Cs

def find_decomp(T, initialise, update_rule, step, max_iterations, stop_criterion, found_criterion):
    best = np.Infinity
    best_int = np.Infinity

    all_As = []
    all_Bs = []
    all_Cs = []

    i = 0
    while True:
        A, B, C = initialise()

        As, Bs, Cs = ALS(T, A, B, C, update_rule, step, max_iterations, stop_criterion)

        A, B, C = As[-1], Bs[-1], Cs[-1]

        tensor_round = Tensors.tensor(A, B, C)

        diff = tensor_round - T
        norm = np.linalg.norm(diff)

        A_round = np.round(A * 2) / 2
        B_round = np.round(B * 2) / 2
        C_round = np.round(C * 2) / 2

        tensor_round = Tensors.tensor(A_round, B_round, C_round)

        diff = tensor_round - T
        norm_round = np.sum(np.abs(diff))

        if norm < best:
            best = norm

        if norm_round < best_int:
            best_int = norm_round

        all_As.append(As)
        all_Bs.append(Bs)
        all_Cs.append(Cs)

        if found_criterion(As, Bs, Cs):
            sys.stdout.write(f"\rIteration {i}, best {best}, best {best_int}")
            return all_As, all_Bs, all_Cs

        sys.stdout.write(f"\rIteration {i}, best {best}, best {best_int}")

        i += 1
```

Bibliography

[1] Josh Alman and Virginia Vassilevska Williams. A refined laser method and faster matrix multiplication. In Proceedings of the 2021 ACM-SIAM Symposium on Discrete Algorithms (SODA), pages 522–539. SIAM, 2021.

[2] Dario Bini, Milvio Capovani, Francesco Romani, and Grazia Lotti. O (n2. 7799) complexity for n× n approximate matrix multiplication. Information Processing Letters, 8(5):234–235, 1979.

[3] Markus Bläser. On the complexity of the multiplication of matrices of small formats. Journal of Complexity, 19(1):43–60, 2003.

[4] Pierre Comon, Xavier Luciani, and André LF De Almeida. Tensor decompositions, alternating least squares and other tales. Journal of Chemometrics: A Journal of the Chemometrics Society, 23(7-8):393–405, 2009.

[5] Don Coppersmith and Shmuel Winograd. On the asymptotic complexity of matrix multiplication. SIAM Journal on Computing, 11(3):472–492, 1982.

[6] Don Coppersmith and Shmuel Winograd. Matrix multiplication via arithmetic progressions. In Proceedings of the nineteenth annual ACM symposium on Theory of computing, pages 1–6, 1987.

[7] Nicolas T Courtois, Gregory V Bard, and Daniel Hulme. A new general-purpose method to multiply 3x3 matrices using only 23 multiplications. arXiv preprint arXiv:1108.2830, 2011.

[8] Alexander Munro Davie and Andrew James Stothers. Improved bound for complexity of matrix multiplication. Proceedings of the Royal Society of Edinburgh Section A: Mathematics, 143(2):351–369, 2013.

[9] Hans F de Groote. On varieties of optimal algorithms for the computation of bilinear mappings ii. optimal algorithms for 2× 2-matrix multiplication. Theoretical Computer Science, 7(2):127–148, 1978.

[10] Ran Duan, Hongxun Wu, and Renfei Zhou. Faster matrix multiplication via asymmetric hashing. arXiv preprint arXiv:2210.10173, 2022.

[11] Alhussein Fawzi, Matej Balog, Aja Huang, Thomas Hubert, Bernardino Romera-Paredes, Mohammadamin Barekatain, Alexander Novikov, Francisco J R Ruiz, Julian Schrittwieser, Grzegorz Swirszcz, et al. Discovering faster matrix multiplication algorithms with reinforcement learning. Nature, 610 (7930):47–53, 2022.

[12] Marijn JH Heule, Manuel Kauers, and Martina Seidl. New ways to multiply 3× 3-matrices. Journal of Symbolic Computation, 104:899–916, 2021.

[13] John E Hopcroft and Leslie R Kerr. On minimizing the number of multiplications necessary for matrix multiplication. SIAM Journal on Applied Mathematics, 20(1):30–36, 1971.

[14] Rodney W Johnson and Aileen M McLoughlin. Noncommutative bilinear algorithms for 3*3 matrix multiplication. SIAM Journal on Computing, 15(2):595–603, 1986.

[15] Manuel Kauers and Jakob Moosbauer. Flip graphs for matrix multiplication. arXiv preprint arXiv:2212.01175, 2022.

[16] Julian D Laderman. A noncommutative algorithm for multiplying 3*3 matrices using 23 multiplications. 1976.

[17] François Le Gall. Algebraic complexity theory and matrix multiplication. In ISSAC, page 23, 2014.

[18] François Le Gall. Powers of tensors and fast matrix multiplication. In Proceedings of the 39th international symposium on symbolic and algebraic computation, pages 296–303, 2014.

[19] Jinsoo Oh, Jin Kim, and Byung-Ro Moon. On the inequivalence of bilinear algorithms for 3×3 matrix multiplication. Information Processing Letters, 113(17):640–645, 2013.

[20] V Ya Pan. Strassen's algorithm is not optimal trilinear technique of aggregating, uniting and canceling for constructing fast algorithms for matrix operations. In 19th Annual Symposium on Foundations of Computer Science (sfcs 1978), pages 166–176. IEEE, 1978.

[21] V Ya Pan. Field extension and trilinear aggregating, uniting and canceling for the acceleration of matrix multiplications. In 20th Annual Symposium on Foundations of Computer Science (sfcs 1979), pages 28–38. IEEE, 1979.

[22] Francesco Romani. Some properties of disjoint sums of tensors related to matrix multiplication. SIAM Journal on Computing, 11(2):263–267, 1982.

[23] Arnold Schönhage. Partial and total matrix multiplication. SIAM Journal on Computing, 10(3):434–455, 1981.

[24] Alexandre Sedoglavic and Alexey V Smirnov. The tensor rank of 5x5 matrices multiplication is bounded by 98 andits border rank by 89. In Proceedings of the 2021 on International Symposium on Symbolic and Algebraic Computation, pages 345–351, 2021.

[25] Volker Strassen. The asymptotic spectrum of tensors and the exponent of matrix multiplication. In 27th Annual Symposium on Foundations of Computer Science (sfcs 1986), pages 49–54. IEEE, 1986.

[26] Volker Strassen et al. Gaussian elimination is not optimal. Numerische mathematik, 13(4):354–356, 1969.

[27] Virginia Vassilevska Williams. Multiplying matrices faster than coppersmith-winograd. In Proceedings of the forty-fourth annual ACM symposium on Theory of computing, pages 887–898, 2012.