Complexity of Matrix Multiplication: The Power of Recursion

Okko Makkonen

November 2022

Abstract

The complexity of matrix multiplication represents the number of operations needed to compute a matrix product in the asymptotic limit. The first advance in asymptotic complexity was made in 1969 when Strassen introduced an algorithm that is capable of computing the product of two $N \times N$ matrices with $\mathcal{O}(N^{2.81})$ operations, which is better than the naive algorithm that takes $\mathcal{O}(N^3)$ operations. The current record is an algorithm that is able to compute the product using just $\mathcal{O}(N^{2.372})$ operations. We present the basic tools that are used to analyze this problem and present an algorithm that is even better than the Strassen algorithm. This includes studying the matrix multiplication tensor, its rank, and the so-called border rank.

1 Introduction and Strassen's algorithm

This note follows some great surveys on the topic: [Pan, 1984], [Stothers, 2010], and [Bläser, 2013].

We wish to compute the product of two $N \times N$ matrices $A \in K^{N\times N}$ and $B \in K^{N\times N}$ over some field K. The field will not be specified at this stage and some of the results could even be generalized to arbitrary rings. Matrix multiplication is defined by

$$(AB)_{ik} = \sum_{j=1}^{N} A_{ij}B_{jk}.$$

The obvious way to compute this requires N multiplications and $N-1$ additions for each term. As there are N^2 terms in the output, we need N^3 multiplications and $N^3 - N^2$ additions. In total, there are $2N^3 - N^2 \in \mathcal{O}(N^3)$ operations. For the simple case of $N = 2$, the naive algorithm requires 8 multiplications and 4 additions, which is a total of 12 operations.

In [Strassen, 1969] Strassen introduced the following algorithm for computing the product of two 2×2 matrices. Compute the following 7 products:

$$\begin{aligned}
\mathrm{I} &= (A_{11} + A_{22})(B_{11} + B_{22}), & \mathrm{V} &= (A_{11} + A_{12})B_{22},\\
\mathrm{II} &= (A_{21} + A_{22})B_{11}, & \mathrm{VI} &= (A_{21} - A_{11})(B_{11} + B_{12}),\\
\mathrm{III} &= A_{11}(B_{12} - B_{22}), & \mathrm{VII} &= (A_{12} - A_{22})(B_{21} + B_{22}),\\
\mathrm{IV} &= A_{22}(B_{21} - B_{11}).
\end{aligned}$$

Then,

$$\begin{aligned}
(AB)_{11} &= \mathrm{I} + \mathrm{IV} - \mathrm{V} + \mathrm{VII}, & (AB)_{12} &= \mathrm{III} + \mathrm{V},\\
(AB)_{21} &= \mathrm{II} + \mathrm{IV}, & (AB)_{22} &= \mathrm{I} + \mathrm{III} - \mathrm{II} + \mathrm{VI}.
\end{aligned}$$

This algorithm requires 18 additions and 7 multiplications, which is more operations than the naive algorithm for 2×2 matrices. There have been some improvements, which have brought the number of additions down a little bit, but all require more operations than the naive algorithm.

Notice that Strassen's algorithm works over any (noncommutative) ring, so we can apply it recursively. If $N = 2^k$, then we can compute the product of two $N \times N$ matrices with 18 additions and 7 multiplications over $2^{k-1} \times 2^{k-1}$ matrices. These products can be computed by again applying the Strassen's algorithm. Let us denote by A_k the number of additions and by M_k the number of multiplications. Then, $M_k = 7^k$ and $A_k = 7A_{k-1} + 18 \cdot 4^{k-1}$ and $A_0 = 0$. By setting $B_k = A_k - 7A_{k-1} = 18 \cdot 4^{k-1}$. Thus,

$$A_k = B_k + 7A_{k-1} = B_k + 7(B_{k-1} + 7A_{k-2}) = \cdots = \sum_{i=0}^{k-1} 7^i B_{k-i} = 6 \cdot (7^k - 4^k).$$

In total, we get $6 \cdot (7^k - 4^k)$ additions and 7^k multiplications. As $7^k = N^{\log 7/ \log 2}$ we get that the multiplication can be done using just $\mathcal{O}(N^{\log 7/ \log 2})$ operations if N is a power of two. If N is not a power of two, we can round N up to the nearest power of two, say 2^k. We can compute the $N \times N$ product by computing the $2^k \times 2^k$ product by padding with zeros. As $2^k \leq 2N$, we get that the product can be computed using $\mathcal{O}(N^{\log 7/ \log 2})$ operations. As $\log 7/ \log 2 \approx 2.807 < 3$, we have now found an asymptotically faster algorithm for computing the matrix product.

Notice in the above recursive definition that **what matters for the complexity is the number of multiplications** in the Strassen's algorithm, not the number of additions. This is because in the first step, the additions cost 4^{k-1} operations, but the multiplications cost at least 7^k operations. Therefore, we will be interested in minimizing the number of multiplications we do. Even scalar multiplications will be free compared to matrix multiplications. In [Winograd, 1971] it was shown that the Strassen algorithm is optimal for multiplying 2×2 matrices.

Definition 1 (Matrix multiplication exponent)**.** The matrix multiplication exponent is defined as

$$\omega = \inf\{\beta \mid \text{two } N \times N \text{ matrices can be multiplied using } \mathcal{O}(N^\beta) \text{ operations}\}.$$

Notice that the matrix multiplication exponent is defined as the infimum, which means that it is not necessarily the case that the product can be computed using $\mathcal{O}(N^\omega)$ operations. However, the product can be computed using $\mathcal{O}(N^{\omega+\varepsilon})$ operations for all $\varepsilon > 0$. Here we are only interested in computing the product of square matrices. A similar exponent can be defined also for nonsquare matrices. It still turns out that the products of nonsquare matrices is interesting for this note.

The naive algorithm gives us the trivial upper bound of $\omega \leq 3$. On the other hand, Strassen's algorithm gives us the better bound $\omega \leq \log 7/ \log 2$. The trivial lower bound of ω is given by the fact that the output contains N^2 entries, so $\omega \geq 2$. There have been some lower bounds ([Bläser, 1999]), but none give $\omega > 2$. It is an open problem whether $\omega = 2$ or $\omega > 2$.

If we have an algorithm for computing the product of two $n \times n$ matrices for some fixed $n > 1$, which uses r multiplications, then we can recursively apply this algorithm for matrices of size $N = n^k$. This gives us a total of r^k multiplications. Therefore, we can give an upper bound on the matrix multiplication exponent.

Theorem 1 (The power of recursion)**.** *If we have an algorithm for computing the product of two $n \times n$ matrices for $n > 1$, which uses at most r multiplications, then we have*

$$\omega \leq \log r/ \log n.$$

The idea of the above theorem is that **if you have an algorithm for multiplying small square matrices, then ω is determined by the number of multiplications in that algorithm**. Strassen's algorithm is an example of this, since the base algorithm uses $r = 7$ multiplications to compute the product of matrices of order $n = 2$.

2 The matrix multiplication tensor and its rank

Let U, V, W be vector spaces and $\varphi\colon U \times V \to W$ a bilinear map. Then

$$\varphi(u,v) = \varphi(\sum_i u_i x_i, \sum_j v_j y_j) = \sum_i \sum_j u_i v_j \varphi(x_i, y_j),$$

where $\{x_i\}_i$ and $\{y_j\}_j$ are bases for U and V. This means that φ is determined by its action on the basis elements, *i.e.*, $w_{ij} = \varphi(x_i, y_j)$ determine φ uniquely. On the other hand, $f_i\colon u \mapsto u_i$ and $g_j\colon v \mapsto v_j$ are linear functionals on U and V. Therefore, φ is uniquely associated with a tensor in the space

$$U^* \otimes V^* \otimes W.$$

In fact, we have an isomorphism that sends the map $(u,v) \mapsto f(u)g(v)w$ to the simple tensor $f \otimes g \otimes w$. In general, if $\varphi_1\colon U_1 \times V_1 \to W_1$ and $\varphi_2\colon U_2 \times V_2 \to W_2$ are bilinear, then

$$\begin{aligned} \varphi\colon (U_1 \oplus U_2) \times (V_1 \oplus V_2) &\to W_1 \oplus W_2 \\ (u_1 \oplus u_2, v_1 \oplus v_2) &\mapsto \varphi_1(u_1, v_1) \oplus \varphi_2(u_2, v_2) \end{aligned}$$

is bilinear. Thus, there is a correspondence between such bilinear maps and the tensors in

$$(U_1^* \oplus U_2^*) \otimes (V_1^* \oplus V_2^*) \otimes (W_1 \oplus W_2).$$

We also have the embeddings of $U_1^* \otimes V_1^* \otimes W_1$ and $U_2^* \otimes V_2^* \otimes W_2$ to this space by mapping the simple tensors

$$\begin{aligned} f_1 \otimes g_1 \otimes w_1 &\mapsto (f_1 \oplus 0) \otimes (g_1 \oplus 0) \otimes (w_1 \oplus 0) \\ f_2 \otimes g_2 \otimes w_2 &\mapsto (0 \oplus f_2) \otimes (0 \oplus g_2) \otimes (0 \oplus w_2). \end{aligned}$$

For brevity, we will write

$$s \odot t = \underbrace{t \oplus \cdots \oplus t}_{s \text{ times}}.$$

Similarly, we may write

$$(U_1^* \otimes V_1^* \otimes W_1) \otimes (U_2^* \otimes V_2^* \otimes W_2) \cong (U_1^* \otimes U_2^*) \otimes (V_1^* \otimes V_2^*) \otimes (W_1 \otimes W_2),$$

which allows us to define the tensor product of tensors.

The rank of a tensor is defined in the usual way, *i.e.*, the rank of $t \in U^* \otimes V^* \otimes W$ is the smallest r such that

$$t = \sum_{i=1}^{r} f_i \otimes g_i \otimes w_i$$

for some $f_i \in U^*$, $g_i \in V^*$ and $w_i \in W$. We denote the rank of t by $R(t)$. We have the following properties of the rank, which can easily be proven by writing out the rank decomposition of t_1 and t_2.

Lemma 1. *If t, t_1 and t_2 are tensors, then*

$$\begin{aligned} R(t_1 \oplus t_2) &\leq R(t_1) + R(t_2) \\ R(s \odot t) &\leq sR(t) \\ R(t_1 \otimes t_2) &\leq R(t_1)R(t_2). \end{aligned}$$

We can interpret the rank decomposition of the tensor as follows. The bilinear map φ can be computed by computing the linear forms $f_i(u)$ and $g_i(u)$, then computing their products $f_i(u)g_i(u)$ and finally computing the sum of $f_i(u)g_i(u)w_i$. If we only count the number of "bilinear multiplications" (*i.e.*, not scalar multiplications), then we need $R(t)$ multiplications to compute $\varphi(u, v)$, where t is the tensor associated to φ.

We set $U = K^{m\times n}$, $V = K^{n\times p}$ and $W = K^{m\times p}$ and consider the map

$$\begin{aligned}\varphi\colon K^{m\times n} \times K^{n\times p} &\to K^{m\times p}\\ (A, B) &\mapsto AB.\end{aligned}$$

This is a bilinear map. The tensor associated to the matrix multiplication is called the matrix multiplication tensor, and we denote this tensor by $\langle m, n, p\rangle$. Using the definition of matrix multiplication we can write the matrix multiplication tensor as

$$\langle m, n, p\rangle = \sum_{i=1}^{m}\sum_{j=1}^{n}\sum_{k=1}^{p} \pi_{ij} \otimes \tau_{jk} \otimes E_{ik},$$

where $\pi_{ij}(A) = A_{ij}$, $\tau_{jk}(B) = B_{jk}$ and E_{ik} is the elementary matrix with a 1 in the position indexed by (i, k).

If $R(\langle m, n, p\rangle) = r$, then we can compute the product of a $m \times n$ matrix with an $n \times p$ matrix using r multiplications. However, this is not necessarily optimal, since we only consider the linear forms f_i, g_i as depending on the entries of A and the entries of B. If we allowed mixed linear forms, then we could use fewer multiplications. In any case, we have the following inequalities regarding the number of multiplications and the rank of the tensor

$$\#\text{multiplications} \leq R(\langle m, n, p\rangle) \leq 2 \cdot \#\text{multiplications}.$$

There are some cases where the number of multiplications required is strictly less than the rank of the associated tensor. Asymptotically we should not see a difference, so we can define the matrix multiplication exponent using the rank. The following proposition states that the two definitions match.

Proposition 1. *The matrix multiplication exponent can be written using the rank as*

$$\omega = \inf\{\beta \mid R(\langle N, N, N\rangle) \in \mathcal{O}(N^\beta)\}.$$

Studying the rank is much more well-behaved, so this is a good choice. We can now look at the direct sum and tensor product of two such matrix multiplication tensors. Firstly, the direct sum of $\langle m, n, p\rangle$ and $\langle m', n', p'\rangle$ corresponds to the bilinear map

$$(A, X), (B, Y) \mapsto (AB, XY).$$

Second, the tensor product of $\langle m, n, p\rangle$ and $\langle m', n', p'\rangle$ corresponds to the matrix multiplication tensor $\langle mm', nn', pp'\rangle$. This corresponds to the well-known property $(A \otimes X)(B \otimes Y) = AB \otimes XY$. We can write the properties of the rank in terms of the matrix multiplication tensor as follows.

Lemma 2. *We have the following properties for the matrix multiplication tensors*

$$\begin{aligned}&\langle m, n, p\rangle \otimes \langle m', n', p'\rangle = \langle mm', nn', pp'\rangle,\\ &R(\langle m, n, p\rangle) = R(\langle n, m, p\rangle) = \cdots = R(\langle p, n, m\rangle)\\ &R(\langle m, n, p\rangle) \leq R(\langle m', n', p'\rangle) \qquad \text{if } m \leq m', n \leq n', p \leq p'.\end{aligned}$$

It is obvious that $R(\langle m,n,p\rangle) = R(\langle p,n,m\rangle)$, since $B^TA^T = (AB)^T$, *i.e.*, the product of a $p \times n$ and a $n \times m$ matrix can be computed by computing the product of a $m \times n$ matrix and a $n \times p$ matrix. Furthermore, by constructing a simple isomorphism, we may deduce that $R(\langle m,n,p\rangle) = R(\langle p,m,n\rangle)$. Thus, $R(\langle m,n,p\rangle)$ is constant for all permutations of m,n,p. The last one can be obtained by embedding the smaller matrix multiplication in the larger one by padding with zeros.

These should be interpreted as:

- **matrix product can be computed by partitioning**,
- **switching the dimensions does not affect the complexity**, and
- **computing a large matrix product is at least as difficult as computing a small matrix product**.

We can now prove a better version of Theorem 1.

Theorem 2 (The power of recursion)**.** *If $R(\langle m,n,p\rangle) \leq r$ for some m,n,p such that $mnp > 1$, then*

$$\omega \leq 3 \log r / \log mnp.$$

Proof. Using the properties of the matrix multiplication tensor and its rank, we get

$$\begin{aligned} R(\langle mnp, mnp, mnp\rangle) &= R(\langle m,n,p\rangle \otimes \langle p,m,n\rangle \otimes \langle n,p,m\rangle) \\ &\leq R(\langle m,n,p\rangle)R(\langle p,m,n\rangle)R(\langle n,p,m\rangle) \\ &= R(\langle m,n,p\rangle)^3 \\ &\leq r^3. \end{aligned}$$

Then we may use Theorem 1 to deduce that

$$\omega \leq \log r^3 / \log mnp = 3 \log r / \log mnp.$$

□

The idea of the above theorem is that **if you have an algorithm for multiplying small matrices, then ω is determined by the number of multiplications in that algorithm, even if the matrices are not square**. In particular, we get the following algorithm for multiplying square matrices of order mnp. Partition the matrices first to blocks of size $np \times mp$ and $mp \times mn$. The full product can be computed using r multiplications of the blocks. Partition the blocks further to blocks of size $p \times m$ and $m \times n$. These products can be computed using r block products. The final block products can be computed with r multiplications. Therefore, the total number of multiplications is at most r^3.

3 Disjoint matrix multiplication

We will now focus on the problem of computing two matrix products AB and XY simultaneously. This problem is known as disjoint matrix multiplication, since we assume no dependencies between the entries of the different matrices. We start by noting the equalities

$$(a+x)(b+y) - xb - (a+x)y = ab$$
$$(a+x)(b+y) - xb - a(b+y) = xy.$$

Using this we may write $a = A_{ij}$, $b = B_{jk}$, $x = X_{jk}$, $y = Y_{ki}$. Then,

$$(AB)_{ik} = \sum_{j=1}^{n}(A_{ij}+X_{jk})(B_{jk}+Y_{jk}) - \sum_{j=1}^{n} X_{jk}B_{jk} - \left[\sum_{j=1}^{n}(A_{ij}+X_{jk})\right]Y_{ki}$$
$$(XY)_{ji} = \sum_{k=1}^{p}(A_{ij}+X_{jk})(B_{jk}+Y_{jk}) - \sum_{k=1}^{p} X_{jk}B_{jk} - A_{ij}\left[\sum_{k=1}^{p}(B_{jk}+Y_{ki})\right].$$

We may compute these sums by computing the products

$$\begin{aligned}
(A_{ij}+X_{jk})(B_{jk}+Y_{jk}) &\qquad \text{for all } i,j,k\\
X_{jk}B_{jk} &\qquad \text{for all } j,k\\
\left[\sum_{j=1}^{n}(A_{ij}+X_{jk})\right]Y_{ki} &\qquad \text{for all } k,i\\
A_{ij}\left[\sum_{k=1}^{p}(B_{jk}+Y_{ki})\right] &\qquad \text{for all } i,j.
\end{aligned}$$

This means that we need to compute a total of $mnp + mn + mp + np$ products to compute both AB and XY. This computation will give us a decomposition of the corresponding tensor

$$\langle m,n,p\rangle \oplus \langle n,p,m\rangle$$

into a sum of $mnp + mn + mp + np$ simple tensors. As we are computing two equally hard products, we would wish that we can compute one of the problems using half the number of computations. In other words, we wish for the following property of the ranks

$$R(\langle m,n,p\rangle \oplus \langle n,p,m\rangle) = R(\langle m,n,p\rangle) + R(\langle n,p,m\rangle).$$

However, we do not have a proof of such a property. The general version of this property is known as Strassen's conjecture, which is still open. We would still like to utilize such an idea, but without requiring the proof of Strassen's conjecture.

Theorem 3 (The power of recursion)**.** *If $R(s \odot \langle m,n,p\rangle) \leq r$ and $r = qs$ for some m,n,p such that $mnp > 1$, then*

$$\omega \leq 3 \log \frac{r}{s} / \log mnp.$$

We would want to show that $R(\langle m,n,p\rangle) \leq q$, which would give us the result. However, it will be easier to show that

$$R(s \odot \langle m^h, n^h, p^h\rangle) \leq q^h s.$$

In the end we will show that this gives us the result asymptotically as $h \to \infty$.

Proof. In the case $h = 1$ we clearly have

$$R(s \odot \langle m^1, n^1, p^1 \rangle) \leq r = q^1 s.$$

Let us proceed by induction on h. We have that

$$\begin{aligned} R(s \odot \langle m^{h+1}, n^{h+1}, p^{h+1} \rangle) &= R((s \odot \langle m, n, p \rangle) \otimes \langle m^h, n^h, p^h \rangle) \\ &\leq R(r \odot \langle m^h, n^h, p^h \rangle) \\ &= R(q \odot (s \odot \langle m^h, n^h, p^h \rangle)) \\ &\leq qR(s \odot \langle m^h, n^h, p^h \rangle) \\ &\leq qq^h s = q^{h+1} s. \end{aligned}$$

Thus,

$$R(\langle m^h, n^h, p^h \rangle) \leq R(s \odot \langle m^h, n^h, p^h \rangle) \leq q^h s.$$

By Theorem 2 we get that for all h

$$\omega \leq 3 \log(q^h s) / \log(mnp)^h.$$

As ω is defined as the infimum, we must also have that ω is less than the limit of the upper bounds. Therefore,

$$\omega \leq \lim_{h \to \infty} 3 \frac{h \log q + \log s}{h \log mnp} = \lim_{h \to \infty} 3 \frac{\log q}{\log mnp} + \frac{3 \log s}{h \log mnp} = 3 \log q / \log mnp.$$

□

The idea of the above theorem is that **if you have an algorithm for multiplying s small matrix products, then ω is determined by the number of multiplications in that algorithm divided by the number of products**. Actually, we do not even need $s \mid r$ by approximating appropriately. The bound is obtained by accumulating the accelerating power of the disjoint matrix multiplication task by using recursion. The cost of computing multiple products goes to zero as the sizes of the matrices grow. We are only computing unnecessary products in the first recursive step, since the rest require r multiplications, which can be peformed by computing the q many s-fold products using the algorithm.

From this result it should be clear that the algorithms obtained in this way are not useful in practice, since their accelerating power will become useful only in the limit. We have also not shown that the matrix multiplication can be computed in $\mathcal{O}(N^\beta)$ operations, where β is the upper bound in the theorem. Our obtained upper bound is a limiting upper bound. If $\omega > 0$, then ω will be limiting in this sense by using a similar argument as in the above theorem.

We can now utilize our result that

$$R(2 \odot \langle n, n, n \rangle) \leq n^3 + 3n^2.$$

Theorem 3 immediately gives us that

$$\omega \leq \log \frac{n^3 + 3n^2}{2} / \log n.$$

In particular, $n = 13$ gives us $\omega \leq \log 1352 / \log 13 \leq 2.811$. This is slightly worse than the Strassen's bound. It seems that all the work we did was wasted, since we were not able to improve on Strassen's bound. However, we are still missing one tool.

4 Approximate algorithms and the border rank

Let us now concentrate on the field $K = \mathbb{R}$, but these methods will eventually work for any field. We start this section with the same equalities as in the previous sections, but we replace y with λy. Thus,

$$(a+x)(b+\lambda y) - xb - \lambda(a+x)y = ab$$
$$\lambda^{-1}\Big[(a+x)(b+\lambda y) - xb - a(b+\lambda y)\Big] = xy.$$

We can compute these products with 4 multiplications just like before. However, $\lambda(a+x)y \to 0$ as $\lambda \to 0$, so we can approximate the products using just 3 multiplications by discarding the term $\lambda(a+x)y$. Therefore, we can approximate the tensor $\langle m,n,p\rangle \oplus \langle n,p,m\rangle$ with a sequence of tensors of rank at most $mnp + mn + np$. The following definition was first given in [Bini et al., 1979].

Definition 2 (Border rank)**.** Let $d \in \mathbb{N}$. We define $R_d(t)$ to be the smallest such r such that

$$\lambda^d t + \mathcal{O}(\lambda^{d+1}) = t_d,$$

where t_d is a tensor of rank at most r over the ring $K[\lambda]$. Then the border rank of t is $\underline{R}(t) = \min_d R_d(t)$.

Over $\mathbb{R}$ or $\mathbb{C}$ this can be thought of as obtaining t as the limit of tensors with rank at most r. For order 2 tensors, *i.e.*, matrices, the border rank and the rank coincide, since the rank is semi-continuous. We have a few properties of the border rank.

Lemma 3. *Let t, t_1, t_2 be a tensors and $d, d_1, d_2 \in \mathbb{N}$. Then*

$$R(t) = R_0(t) \geq R_1(t) \geq \cdots \geq \underline{R}(t),$$
$$R_d(t_1 \oplus t_2) \leq R_d(t_1) + R_d(t_2),$$
$$R_d(s \odot t) \leq sR_d(t)$$
$$R_{d_1+d_2}(t_1 \otimes t_2) \leq R_{d_1}(t_1)R_{d_2}(t_2).$$

These are clear by representing the tensors using the representation in the definition of $R_d(t)$. Even though the border rank represents approximation of the tensor, we can still gain useful information about the rank of the tensor by the following lemma.

Lemma 4. *Let t be a tensor and $d \in \mathbb{N}$. Then $R(t) \leq c_d R_d(t)$, where c_d is a polynomial in d.*

Proof. Let $r = R_d(t)$. Then we can write the approximation

$$\begin{aligned}
\lambda^d t + \mathcal{O}(\lambda^{d+1}) &= \sum_{i=1}^{r} f_i \otimes g_i \otimes w_i \\
&= \sum_{i=1}^{r} \left(\sum_{\alpha=0}^{d} \lambda^\alpha f_i^{(\alpha)}\right) \otimes \left(\sum_{\beta=0}^{d} \lambda^\beta g_i^{(\beta)}\right) \otimes \left(\sum_{\gamma=0}^{d} \lambda^\gamma w_i^{(\gamma)}\right) \\
&= \sum_{\alpha=0}^{d}\sum_{\beta=0}^{d}\sum_{\gamma=0}^{d} \lambda^{\alpha+\beta+\gamma} \sum_{i=1}^{r} f_i^{(\alpha)} \otimes g_i^{(\beta)} \otimes w_i^{(\gamma)}.
\end{aligned}$$

By comparing coefficients we get that

$$t = \sum_{\substack{\alpha,\beta,\gamma \\ \alpha+\beta+\gamma=d}} \sum_{i=1}^{r} f_i^{(\alpha)} \otimes g_i^{(\beta)} \otimes w_i^{(\gamma)}$$

There are at most $(d+1)^2$ ways to choose α, β, γ such that $\alpha + \beta + \gamma = d$, since we may choose α, β freely, which fixes γ. Therefore, $R(t) \leq (d+1)^2 r$ □

The above lemma will not give us a good algorithm for computing a product if we have an approximate algorithm, since we are multiplying the rank by a potentially large number. We will remedy this fact by using the same trick that we did in Theorem 3, *i.e.*, we accumulate the accelerating power of the approximate algorithm.

Theorem 4 (The power of recursion)**.** *If* $\underline{R}(s \odot \langle m, n, p \rangle) \leq r$ *and* $r = qs$ *for some* m, n, p *such that* $mnp > 1$*, then*

$$\omega \leq 3 \log \frac{r}{s} / \log mnp.$$

Proof. Let d be such that $R_d(s \odot \langle m, n, p, \rangle) \leq r$. By doing the same argument as in the proof of Theorem 3 we get that

$$R_{dh}(s \odot \langle m^h, n^h, p^h \rangle) \leq q^h s.$$

Thus, we may use the relation between the rank and the border rank to obtain

$$R(s \odot \langle m^h, n^h, p^h) \leq (dh+1)^2 R_{dh}(s \odot \langle m^h, n^h, p^h \rangle) = (dh+1)^2 q^h s$$

for large enough dh by Lemma 3. Hence, by Theorem 3 we get that

$$\omega \leq \lim_{h \to \infty} 3 \log \left((dh+1)^2 q^h s \right) / \log(mnp)^h = \frac{3 \log q}{\log mnp} + \lim_{h \to \infty} \frac{3 \log (dh+1)^2 s}{h \log mnp} = 3 \log q / \log mnp.$$

The last limit comes from the fact that the logarithm of a polynomial is strictly sublinear. □

The idea of the above theorem is that **if you have an algorithm for multiplying s small matrix products, then ω is determined by the number of multiplications in that algorithm divided by the number of products, even if the algorithm can only approximate the products arbitrarily well**.

We can use the above theorem to see that

$$\omega \leq \log \frac{n^3 + 2n^2}{2} / \log n$$

for all $n > 1$. In particular, if we choose $n = 6$, then $\omega \leq \log 144 / \log 6 \leq 2.774 < \log 7 / \log 2$. This means that we have produced an algorithm that is asymptotically faster than the Strassen algorithm.

5 Conclusions

Using the tools we were able to come up with an algorithm that can show that $\omega \leq 2.774$. The same techniques can be used to show that $\omega \leq 2.67$ [Pan, 1984]. Much more advanced methods can be utilized, but are too complicated for this short note. These include partial matrix multiplication, Schönhage's τ-theorem, Strassen's Laser method, Coppersmith and Winograd tensor, *etc.* The current record is at $\omega < 2.37188$, which was achieved recently in October 2022 [Duan et al., 2022].

There has also been research into explaining some of these algorithms in a more understandable way. In particular, explaining the construction of the Strassen algorithm has been studied in [Landsberg, 2008], [Burichenko, 2014], [Grochow and Moore, 2017], [Ikenmeyer and Lysikov, 2019] using secant varieties, Segre isomorphisms, unitary 2-designs, and by considering different bases.

References

[Bini et al., 1979] Bini, D., Capovani, M., Lotti, G., and Romani, F. (1979). $O(n^{2.7799})$ complexity for matrix multiplication. Strassen algorithm is not optimal. *Inform. Process. Lett*, 8(5):234–235.

[Bläser, 1999] Bläser, M. (1999). A $5/2n^2$-lower bound for the rank of $n \times n$-matrix multiplication over arbitrary fields. In *40th Annual Symposium on Foundations of Computer Science (Cat. No. 99CB37039)*, pages 45–50. IEEE.

[Bläser, 2013] Bläser, M. (2013). Fast matrix multiplication. *Theory of Computing*, pages 1–60.

[Burichenko, 2014] Burichenko, V. P. (2014). On symmetries of the Strassen algorithm. *arXiv preprint arXiv:1408.6273*.

[Duan et al., 2022] Duan, R., Wu, H., and Zhou, R. (2022). Faster matrix multiplication via asymmetric hashing. *arXiv preprint arXiv:2210.10173*.

[Grochow and Moore, 2017] Grochow, J. A. and Moore, C. (2017). Designing Strassen's algorithm. *arXiv preprint arXiv:1708.09398*.

[Ikenmeyer and Lysikov, 2019] Ikenmeyer, C. and Lysikov, V. (2019). Strassen's 2×2 matrix multiplication algorithm: a conceptual perspective. *Annali dell' Università di Ferrara*, 65(2):241–248.

[Landsberg, 2008] Landsberg, J. (2008). Geometry and the complexity of matrix multiplication. *Bulletin of the American Mathematical Society*, 45(2):247–284.

[Pan, 1984] Pan, V. (1984). How can we speed up matrix multiplication? *SIAM review*, 26(3):393–415.

[Stothers, 2010] Stothers, A. J. (2010). *On the complexity of matrix multiplication*. PhD thesis, The University of Edinburgh.

[Strassen, 1969] Strassen, V. (1969). Gaussian elimination is not optimal. *Numerische mathematik*, 13(4):354–356.

[Winograd, 1971] Winograd, S. (1971). On multiplication of 2×2 matrices. *Linear algebra and its applications*, 4(4):381–388.