Matrix Multiplication I

Yuval Filmus

February 2, 2012

These notes are based on a lecture given at the Toronto Student Seminar on February 2, 2012. The material is taken mostly from the book *Algebraic Complexity Theory* [ACT] and the lecture notes by Bläser and Bendun [Blä]. Starred sections are the ones I didn't have time to cover.

1 Problem statement

This lecture discusses the problem of multiplying two square matrices. We will be working in the algebraic complexity model. For us, an algorithm for multiplying two $n \times n$ matrices will mean a sequence of steps, where step l is a statement of the form

- $t_l \leftarrow r$ for any $r \in \mathbb{R}$
- $t_l \leftarrow a_{ij}$ or $t_l \leftarrow b_{ij}$ for $i, j \in \{1, \ldots, n\}$
- $t_l \leftarrow t_p \circ t_q$, where $p, q < l$ and $\circ \in \{+, -, \cdot, /\}$
- $c_{ij} \leftarrow t_p$ for $p < l$

We will say that such an algorithm computes the product $C = AB$ if at the end of the program, $c_{ik} = \sum_j a_{ij} b_{jk}$. The *running-time* or *complexity* of the algorithm is the total number of steps, disregarding input and output steps. Our model is *non-uniform.*

As an example, consider Strassen's algorithm for multiplying two 2×2 matrices, as copied from Wikipedia:

$$\begin{aligned}
m_1 &= (a_{11} + a_{22})(b_{11} + b_{22}) \\
m_2 &= (a_{21} + a_{22})b_{11} \\
m_3 &= a_{11}(b_{12} - b_{22}) \\
m_4 &= a_{22}(b_{21} - b_{11}) \\
m_5 &= (a_{11} + a_{12})b_{22} \\
m_6 &= (a_{21} - a_{11})(b_{11} + b_{12}) \\
m_7 &= (a_{12} - a_{22})(b_{21} + b_{22}) \\
c_{11} &= m_1 + m_4 - m_5 + m_7 \\
c_{12} &= m_3 + m_5 \\
c_{21} &= m_2 + m_4 \\
c_{22} &= m_1 - m_2 + m_3 + m_6
\end{aligned}$$

In our model (using some benign shortcuts), it will look like this:

$$\begin{aligned}
t_1 &\leftarrow a_{11} + a_{22} \\
t_2 &\leftarrow b_{11} + b_{22} \\
t_3 &\leftarrow a_{21} + a_{22} \\
t_4 &\leftarrow b_{12} - b_{22} \\
t_5 &\leftarrow b_{21} - b_{11} \\
t_6 &\leftarrow a_{11} + a_{12} \\
t_7 &\leftarrow a_{21} - a_{11} \\
t_8 &\leftarrow b_{11} + b_{12} \\
t_9 &\leftarrow a_{12} - a_{22} \\
t_{10} &\leftarrow b_{21} + b_{22} \\
t_{11} &\leftarrow t_1 \cdot t_2 \\
t_{12} &\leftarrow t_3 \cdot b_{11} \\
t_{13} &\leftarrow a_{11} \cdot t_4 \\
t_{14} &\leftarrow a_{22} \cdot t_5 \\
t_{16} &\leftarrow t_6 \cdot b_{22} \\
t_{17} &\leftarrow t_7 \cdot t_8 \\
t_{18} &\leftarrow t_9 \cdot t_{10} \\
t_{19} &\leftarrow t_{11} + t_{14} \\
t_{20} &\leftarrow t_{19} - t_{15} \\
c_{11} &\leftarrow t_{20} + t_{17} \\
c_{12} &\leftarrow t_{13} + t_{15} \\
c_{21} &\leftarrow t_{12} + t_{14} \\
t_{21} &\leftarrow t_{11} - t_{12} \\
t_{22} &\leftarrow t_{21} + t_{13} \\
c_{22} &\leftarrow t_{22} + t_{16}
\end{aligned}$$

The total complexity of this algorithm is 26 operations. Later we will write such algorithms in a much better way.

Strassen's algorithm can be used to multiply two $2^n \times 2^n$ matrices. The key is to write the matrices in block form:

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

Each of the blocks is a $2^{n-1} \times 2^{n-1}$ matrix. Applying Strassen's algorithm to the big matrices, there are some additions and subtractions that we already know how to do, and 7 multiplications of two $2^{n-1} \times 2^{n-1}$ matrices. For the latter, we apply the algorithm recursively. Eventually, everything will reduce to 7^n scalar multiplications, and countless[1] additions and subtractions. This construction, called *tensoring*, is the basic tool used in matrix multiplication algorithms.

[1]Not really countless, as we show in the sequel.

Once we show that the running-time is dominated by the number of scalar multiplications, we can conclude that two matrices can be multiplied in time $O(n^{\log_2 7})$. This shows that $\omega \leq \log_2 7 \approx 2.81$. Here ω, also known as *the exponent of matrix multiplication*, is defined as the infimum of all α such that two $n \times n$ matrices can be multiplied in time $O(n^\alpha)$ (the constant can depend on α).

2 Normal form

We spent much of the previous section on defining the algebraic complexity model. Now it's time to show that we can forget about it and concentrate on algorithms of a very specific form. This is the result of the following two theorems.

Theorem 1. *Suppose one can multiply two $n \times n$ matrices in running-time T. Then there is an algorithm in the following normal form, for $M = 2T$:*

- *For $1 \leq i \leq M$, compute α_i, a linear combination of entries of A.*
- *For $1 \leq i \leq M$, compute β_i, a linear combination of entries of B.*
- *For $1 \leq i \leq M$, compute $p_i = \alpha_i \beta_i$.*
- *For $1 \leq j, k \leq n$, compute c_{jk} as a linear combination of the p_i.*

All these linear combinations are fixed (don't depend on A, B).

Our original presentation of Strassen's algorithm is in this form. You might ask what's the point of using this transformation. After all, for each original operation we now have two multiplications, and there are also lots of linear combinations to compute, each requiring up to n^2 multiplications and additions. This is explained by the following theorem.

Theorem 2. *Suppose there is an algorithm in normal form that multiplies two $n \times n$ matrices with $M = n^\alpha$. Then $\omega \leq \alpha$.*

So asymptotically, all the extra additions and multiplications don't really count. Coppersmith and Winograd [CW] showed that in fact $\omega < \alpha$ (we will comment on this later on).

Together, the two theorems show that we can forget about the algebraic complexity model and concentrate on normal-form algorithms, trying to reduce M.

2.1 Proofs*

The idea of the proof of Theorem 1 is that it's enough to keep track of the linear and quadratic parts of all computations involved, since higher degree parts don't matter.

Proof of Theorem 1. **1.** Divisions are confusing, so let's ignore them for now. We will prove something very similar to the statement of the theorem, with $M = T$ and α_i, β_i both linear combinations of *both* input matrices.

If there are no divisions, then each quantity t_i computed during the algorithm is a multivariate polynomial in the input variables. We can group it into its homogeneous parts. The dth homogeneous part $P^{(d)}$ of a polynomial P consists of all those monomials with total degree exactly d. For example, if $P = 1 + 2a + b + a^2 + 3ab$ then

$$\begin{aligned} P^{(0)} &= 1 \\ P^{(1)} &= 2a + b \\ P^{(2)} &= a^2 + 3ab \end{aligned}$$

Every polynomial is the sum of its homogeneous parts. The outputs c_{ij} are all quadratic, i.e. homogeneous of degree 2. So we're not really interested in $t_i^{(d)}$ for $d > 2$. In order to compute $t_i^{(2)}$ for all i, it is enough to compute the constant, linear and quadratic parts of everything (i.e. $t_i^{(0)}, t_i^{(1)}, t_i^{(2)}$). We will convert an arbitrary program into a new program computing these parts for all of the original variables.

Replace a statement $t_i \leftarrow r$ with

$$\begin{aligned} t_i^{(0)} &\leftarrow r \\ t_i^{(1)} &\leftarrow 0 \\ t_i^{(2)} &\leftarrow 0 \end{aligned}$$

Replace a statement $t_i \leftarrow a_{jk}$ with

$$\begin{aligned} t_i^{(0)} &\leftarrow 0 \\ t_i^{(1)} &\leftarrow a_{jk} \\ t_i^{(2)} &\leftarrow 0 \end{aligned}$$

Replace a statement $t_i \leftarrow t_j \pm t_k$ with

$$\begin{aligned} t_i^{(0)} &\leftarrow t_j^{(0)} \pm t_k^{(0)} \\ t_i^{(1)} &\leftarrow t_j^{(1)} \pm t_k^{(1)} \\ t_i^{(2)} &\leftarrow t_j^{(2)} \pm t_k^{(2)} \end{aligned}$$

Replace a statement $t_i \leftarrow t_j \cdot t_k$ with

$$\begin{aligned} t_i^{(0)} &\leftarrow t_j^{(0)} t_k^{(0)} \\ t_i^{(1)} &\leftarrow t_j^{(0)} t_k^{(1)} + t_j^{(1)} t_k^{(0)} \\ t_i^{(2)} &\leftarrow t_j^{(0)} t_k^{(2)} + t_j^{(1)} t_k^{(1)} + t_j^{(2)} t_k^{(0)} \end{aligned}$$

These statements are not basic, but that's not going to matter.

Finally, replace an output statement $c_{ij} \leftarrow t_k$ with

$$c_{ij} \leftarrow t_k^{(2)}.$$

It's easy to prove by induction that the new program also computes matrix multiplication. Another induction shows that:

- All constant parts $t_i^{(0)}$ are constant (don't depend on the inputs).
- All linear parts $t_i^{(1)}$ are linear combinations of inputs.
- All quadratic parts $t_i^{(2)}$ are linear combinations of products $t_j^{(1)}t_k^{(1)}$ appearing up to step i in the original program.

From this it's easy to extract the (modified) normal form.

2. We've almost reached our normal form. The only problem is that α_i and β_i may each depend on both parts of the input. We can always write $\alpha_i = \alpha_i^A + \alpha_i^B$, $\beta_i = \beta_i^A + \beta_i^B$, separating the two parts. Since
$$\alpha_i\beta_i = \alpha_i^A\beta_i^A + \alpha_i^A\beta_i^B + \alpha_i^B\beta_i^A + \alpha_i^B\beta_i^B,$$
we can separate the two inputs at the cost of quadrupling M. Nothing bad would happen if we drop $\alpha_i^A\beta_i^A$ and $\alpha_i^B\beta_i^B$, since at the end we don't need these terms which are quadratic in the entries of one of the matrices. So it's really enough only to double M.

3. Now it's time to handle divisions. Every rational function can be extended to a formal power series (given that the denominator is not zero at the origin[2]). So it's natural to do the same thing as before, adding the following rule for statements $t_j \leftarrow t_i/t_k$:
$$\begin{aligned}
t_j^{(0)} &\leftarrow t_i^{(0)}/t_k^{(0)}\\
t_j^{(1)} &\leftarrow (t_i^{(1)} - t_j^{(0)}t_k^{(1)})/t_k^{(0)}\\
t_j^{(2)} &\leftarrow (t_i^{(2)} - t_j^{(0)}t_k^{(2)} + t_j^{(1)}t_k^{(1)})/t_k^{(0)}.
\end{aligned}$$
We got these statements from reversing what we got for multiplication above. The rest of the proof goes through. □

To prove Theorem 2, we follow the same technique we used to show that Strassen's algorithm applied recursively to $2^n \times 2^n$ matrices results in 7^n scalar multiplications, only this time we also account for the rest of the operations. We get a recurrence whose solution is $O(n^\alpha)$.

Proof of Theorem 2. Like we did for Strassen's algorithm, we can extend the given algorithm to an algorithm for multiplying two $n^k \times n^k$ matrices. Denote its running time by $T(k)$, so $T(1) = n^\alpha$. What is $T(k+1)$? The first thing we have to do is compute M linear combinations in the entries of A, namely the α_i. Each linear combination takes roughly $3n^{2k}$ operations, for a total of $3Mn^{2k}$. The same number of operations is needed to compute the M linear combinations β_i. Next, we multiply these using $MT(k)$ operations. Finally, we compute n^2 linear combinations, one for each entry in the target matrix. So
$$T(k+1) \le n^\alpha T(k) + (6M + n^2)n^{2k}.$$
One can show that necessarily $\alpha > 2$ (in other words, you cannot multiply two $n \times n$ matrices with $M = n^2$), and so the solution to the implicit recurrence is
$$T(k) = O(n^{\alpha k}).$$
In terms of the matrix size $N = n^k$, this is $T(k) = O(N^\alpha)$. □

[2]That happens in our case since the algorithm should correctly compute the product of two zero matrices.

3 Tensor notation

There's a neat way to write algorithms in normal form. Start by writing α_t as a sum of a_{ij} and β_t as a sum of b_{ij}. For Strassen's algorithm, for example, we'd have $\alpha_1 = a_{11} + a_{22}$, $\beta_1 = b_{11} + b_{22}$, and so on. Next, each c_{ij} is a linear combination

$$c_{ij} = \sum_{t=1}^{M} C_t^{ij} \alpha_t \beta_t.$$

Define $\gamma_t = \sum_t C_t^{ij} c_{ij}$, where we think of c_{ij} as formal variables. For example, in Strassen's algorithm we have $\gamma_1 = c_{11} + c_{22}$. Let's look at the expression $\sum_{t=1}^{M} \alpha_t \beta_t \gamma_t$. What is the coefficient of c_{ij}? It's the value that c_{ij} gets at the end of the algorithm, i.e. $\sum_k a_{ik} b_{kj}$. So in a formal sense,

$$\sum_{t=1}^{M} \alpha_t \beta_t \gamma_t = \sum_{i,j,k=1}^{n} a_{ik} b_{kj} c_{ij}.$$

Here's Strassen's algorithm in this form:

$$\begin{aligned}
&\sum_{i,j,k=1}^{2} a_{ik} b_{kj} c_{ij} = \\
&(a_{11} + a_{22})(b_{11} + b_{22})(c_{11} + c_{22}) + \\
&(a_{21} + a_{22}) b_{11} (c_{21} - c_{22}) + \\
&a_{11}(b_{12} - b_{22})(c_{12} + c_{22}) + \\
&a_{22}(b_{21} - b_{11})(c_{11} + c_{21}) + \\
&(a_{11} + a_{12}) b_{22} (-c_{11} + c_{12}) + \\
&(a_{21} - a_{11})(b_{11} + b_{12}) c_{22} + \\
&(a_{12} - a_{22})(b_{21} + b_{22}) c_{11}
\end{aligned}$$

We denote the left-hand side by $\langle 2, 2, 2 \rangle$. In general,

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

corresponds to multiplying an $n \times m$ matrix by an $m \times p$ matrix, obtaining an $n \times p$ matrix as a result. We call these new objects *tensors*. They are generalizations of matrices (see below).

The minimal M such that a tensor T can be represented as a sum $\sum_{i=1}^{M} \alpha_i \beta_i \gamma_i$, where $\alpha_i, \beta_i, \gamma_i$ are linear combinations of entries of A, B, C (respectively) is known as the *rank* of T, denoted $R(T)$. Strassen's algorithm shows that $R(\langle 2, 2, 2 \rangle) \leq 7$ (we actually have equality in this case). In earlier literature, the rank is also called the number of *essential multiplications*.

How does tensor rank generalize matrix rank? Encode a matrix A of dimension $n \times m$ as

$$\sum_{i=1}^{n} \sum_{j=1}^{m} a_{ij} x_i y_j.$$

The rank of this tensor is the minimal R such that

$$\sum_{i=1}^{n}\sum_{j=1}^{m} a_{ij}x_iy_j = \sum_{k=1}^{R}\left(\sum_{i=1}^{n} b_k^i x_i\right)\left(\sum_{j=1}^{m} c_k^j y_j\right).$$

In terms of matrices, this corresponds to a representation

$$A = \sum_{k=1}^{R} b_k c_k,$$

where b_k is a column vector and c_k is a row vector (so these are all outer products). This expresses each row of A as a linear combination of the row vectors c_k, and so R is the row rank of A. Switching the roles, we see that R is also the column rank of A. The symmetric form of this representation thus immediately implies that the row rank equals the column rank.

We can rephrase Theorem 2 using our new terminology:

$$\omega \leq \log_n R(\langle n, n, n\rangle). \tag{1}$$

As we've already commented, the inequality is actually strict.

Why did we bother describing this tensor notation? The new notation shows the symmetry among the variables a_{ij}, b_{ij}, c_{ij}. Starting with the tensor for $\langle n, m, p\rangle$, if we switch a with b then we get the tensor $\langle p, m, n\rangle$ (we also need to switch some of the indices). Going over all permutations, we find out that

$$R(\langle n, m, p\rangle) = R(\langle n, p, m\rangle) = R(\langle m, n, p\rangle) = R(\langle m, p, n\rangle) = R(\langle p, n, m\rangle) = R(\langle p, m, n\rangle). \tag{2}$$

Suppose we know that $R(\langle n_1, m_1, p_1\rangle) \leq R_1$ and $R(\langle n_2, m_2, p_2\rangle) \leq R_2$. What can we say about $R(\langle n_1n_2, m_1m_2, p_1p_2\rangle)$? We can think of the $n_1n_2 \times m_1m_2$ matrix A as an $n_1 \times n_2$ matrix whose entries are $n_2 \times m_2$ matrices. The matrices B, C can be decomposed analogously. We then apply the algorithm showing that $R(\langle n_1, m_1, p_1\rangle) \leq R_1$. Each time we have to multiply two elements, which are now matrices, we use the algorithm showing $R(\langle n_2, m_2, p_2\rangle) \leq R_2$. This will reduce everything to computations of R_1R_2 matrix products. So

$$R(\langle n_1n_2, m_1m_2, p_1p_2\rangle) \leq R(\langle n_1, m_1, p_1\rangle)R(\langle n_2, m_2, p_2\rangle). \tag{3}$$

Let's apply this together with the symmetry relations:

$$R(\langle nmp, nmp, nmp\rangle) \leq R(\langle n, m, p\rangle)^3.$$

This has the implication that

$$\omega \leq 3\log_{nmp} R(\langle n, m, p\rangle). \tag{4}$$

This generalizes (1).

We'll see (and use) even more general forms below.

4 Border rank

Bini came up with the following identity:

$$\begin{aligned}
&\epsilon(a_{11}b_{11}c_{11} + a_{11}b_{12}c_{21} + a_{12}b_{21}c_{11} + a_{12}b_{22}c_{21} + a_{21}b_{11}c_{12} + a_{21}b_{12}c_{22}) + \\
&\epsilon^2(a_{11}b_{22}c_{21} + a_{11}b_{11}c_{12} + a_{12}b_{21}c_{22} + a_{21}b_{21}c_{22}) = \\
&(a_{12} + \epsilon a_{11})(b_{12} + \epsilon b_{22})c_{21} + \\
&(a_{21} + \epsilon a_{11})b_{11}(c_{11} + \epsilon c_{12}) - \\
&a_{12}b_{12}(c_{11} + c_{21} + \epsilon c_{22}) - \\
&a_{21}(b_{11} + b_{12} + \epsilon b_{21})c_{11} + \\
&(a_{12} + a_{21})(b_{12} + \epsilon b_{21})(c_{11} + \epsilon c_{22}).
\end{aligned}$$

On the left-hand side, we have ϵ times the tensor corresponding to the following partial matrix multiplication:

$$\begin{bmatrix} a_{11} & a_{12} \\ a_{21} & 0 \end{bmatrix} \begin{bmatrix} b_{11} & b_{12} \\ b_{21} & b_{22} \end{bmatrix} = \begin{bmatrix} c_{11} & c_{12} \\ c_{21} & c_{22} \end{bmatrix}.$$

From a computational point of view, here's how you'd use this identity to compute this partial matrix multiplication approximately (if you could calculate with infinite precision). Pick $\epsilon > 0$ very small, and use the identity. Divide the result by ϵ. We obtain roughly the result. By picking ϵ infinitesimal, we can compute the matrix multiplication exactly. But there's a way to do it even without nonstandard analysis, as we'll see below.

First, let's see how you'd use the identity for multiplying large square matrices. Putting together two copies of the identity, we get an identity equating $\epsilon\langle 3,2,2\rangle + O(\epsilon^2)$ with a sum of 10 terms (the big-O notation hides terms which grow like ϵ^3 or smaller). Symmetrizing, we get an identity equating $\epsilon^3\langle 12,12,12\rangle + O(\epsilon^4)$ with a sum of 10^3 terms. Taking the Nth tensor power, we get an identity equating $\epsilon^{3N}\langle 12^N, 12^N, 12^N\rangle + O(\epsilon^{3N+1})$ with a sum of 10^{3N} terms.

What do we do with the new identity? We're only really interested in all terms with coefficient ϵ^{3N}. How do we isolate them? Consider the product

$$(a_{12} + \epsilon a_{11})(b_{12} + \epsilon b_{22})c_{21}.$$

If we wanted to compute only the coefficient of ϵ, we would need to compute $a_{11}b_{12}c_{21} + a_{12}b_{22}c_{21}$. The general situation is similar. For each given term $\alpha_i\beta_i\gamma_i$, in order to compute the coefficient of ϵ^{3N}, we will take terms corresponding to coefficients $d_\alpha, d_\beta, d_\gamma$ from $\alpha_i, \beta_i, \gamma_i$ (respectively) such that $d_\alpha + d_\beta + d_\gamma = 6N$, and sum all of these. There are at most $\binom{3N+2}{2} = O(N^2)$ such terms. So

$$\omega \le \log_{12^N} O(N^2 10^{3N}) \to 3\log_{12} 10 \approx 2.78.$$

The polynomial factor $O(N^2)$ doesn't really make any difference in the limit. In other words, the fact that the identity was only approximate makes no difference.

Bini's identity (after taking two copies of it) shows that $\underline{\mathrm{R}}(\langle 3,2,2\rangle) \le 10$. The *border rank* $\underline{\mathrm{R}}(T)$ of a tensor T is exactly the minimum M such that $\epsilon^d T + O(\epsilon^{d+1}) = \sum_i \alpha_i\beta_i\gamma_i$ for some d. Our argument shows in general that

$$\omega \le 3\log_{nmp} \underline{\mathrm{R}}(\langle n,m,p\rangle). \tag{5}$$

Border rank satisfies many of the properties of rank. For example, it's symmetric 2 and submultiplicative 3. We can think of it as a generalization of rank which is "good enough" to obtain bounds on ω.

Bini's particular identity can be leveraged even more, showing that $\omega \leq 3\log_6 5 \approx 2.70$, using the methods of §6.

In earlier literature, these identities are known as λ-computations (so ϵ is replaced with λ). Also, negative powers of λ are used so that the left-hand side has an error term of magnitude $O(\lambda)$.

5 Schönhage's τ theorem

Schönhage discovered the following surprising identity:

$$\begin{aligned}
&\epsilon^2 \left(\sum_{i=1}^{4} \sum_{j=1}^{4} A_i B_j C_{ji} + \sum_{i=1}^{3} \sum_{j=1}^{3} X_{3i+j} Y_{3i+j} Z \right) + O(\epsilon^3) = \\
&\sum_{i=1}^{3} \sum_{j=1}^{3} (A_i + \epsilon X_{3i+j})(B_j + \epsilon Y_{3i+j})(\epsilon^2 C_{ji} + Z) + \\
&\sum_{i=1}^{3} A_i (B_4 - \epsilon \sum_{j=1}^{3} Y_{3i+j})(\epsilon^2 C_{4i} + Z) + \sum_{j=1}^{3} (A_4 - \epsilon \sum_{i=1}^{3} X_{3i+j}) B_j (\epsilon^2 C_{j4} + Z) + \\
&A_4 B_4 (\epsilon^2 C_{44} + Z) - \left(\sum_{i=1}^{4} A_i \right) \left(\sum_{j=1}^{4} B_j \right) Z.
\end{aligned}$$

This identity shows that

$$\underline{\mathrm{R}}(\langle 4,1,4 \rangle \oplus \langle 1,9,1 \rangle) \leq 17.$$

(In fact there is equality.) The identity shows how to compute the outer product $\langle 4,1,4 \rangle$ of two vectors of length 4 along with the inner product $\langle 1,9,1 \rangle$ of two other vectors of length 9 (the symbol $\oplus$ emphasizes the fact that the two products concern different variables) with only $4^2 + 1$ multiplications. This is surprising, since easy arguments show that $\underline{\mathrm{R}}(\langle 4,1,4 \rangle) = 16$, and with one more multiplication it is suddenly possible to compute along an extra (long) inner product.

How do we use this identity? Schönhage invented his τ-theorem (also: asymptotic sum inequality) to answer this question. We will approach his solution through a series of simple steps. For simplicity, we will only explicitly consider rank, but everything we do also works with border rank, which is how we state our results.

1. Suppose we had an identity showing that $R(k \odot \langle n,n,n \rangle) \leq M$ (here $k \odot \langle n,n,n \rangle$ is the direct sum of k copies of $\langle n,n,n \rangle$ with disjoint variables). How would we use it to multiply square matrices? Suppose we wanted to multiply two $n^T \times n^T$ matrices, where T is very large. We apply our new algorithm recursively. In the first level we can't really take advantage of the full abilities of our algorithm, since we only have one matrix product to compute. In the second level, we already have M of these, so we need to apply our algorithm recursively only $\lceil M/k \rceil$ times. As we go along, we will be able to group the products we need to compute at each level to groups of size k, and leverage our algorithm almost perfectly. So asymptotically, our algorithm behaves as if we had a way to multiply two $n \times n$ matrices using M/k products (note that M/k need not be integral).

The details work out, showing

$$\omega \leq \log_n \frac{\underline{\mathrm{R}}(k \odot \langle n, n, n \rangle)}{k}. \tag{6}$$

2. A simple symmetrization argument shows that

$$\omega \leq 3 \log_{nmp} \frac{\underline{\mathrm{R}}(k \odot \langle n, m, p \rangle)}{k}. \tag{7}$$

3. Consider now Schönhage's identity, showing

$$\underline{\mathrm{R}}(\langle 4, 1, 4 \rangle \oplus \langle 1, 9, 1 \rangle) \leq 17.$$

Compute the Nth tensor power:

$$\underline{\mathrm{R}} \left(\bigoplus_{i=0}^{N} c_i \odot \langle n_i, m_i, p_i \rangle \right) \leq 17^N,$$

where

$$c_i = \binom{N}{i}, \quad n_i = p_i = 4^i, \quad m_i = 9^{N-i}.$$

Our goal is to obtain a bound using (7). If we were aiming at a bound $\omega \leq 3\tau$, then we'd need $17^N \approx c_i(n_i m_i p_i)^\tau$. It's natural to define accordingly the *volume* of a tensor $\langle n, m, p \rangle$ by $V_\tau(\langle n, m, p \rangle) = (nmp)^\tau$, and extend the definition additively to direct sums. This notion of volume is multiplicative, i.e. $V_\tau(T_1 T_2) = V_\tau(T_1) V_\tau(T_2)$, so

$$V_\tau \left(\bigoplus_{i=0}^{N} c_i \odot \langle n_i, m_i, p_i \rangle \right) = V_\tau(\langle 4, 1, 4 \rangle \oplus \langle 1, 9, 1 \rangle)^N = (16^\tau + 9^\tau)^N.$$

Since there are only $N + 1$ terms on the left, there must be some term satisfying

$$c_i (n_i m_i p_i)^\tau \geq \frac{(16^\tau + 9^\tau)^N}{N + 1}.$$

We want this to be roughly equal to 17^N, so we choose τ so that

$$16^\tau + 9^\tau = 17.$$

With this value of τ, formula (7) implies that

$$\omega \leq 3 \log_{n_i m_i p_i} \frac{17^N}{c_i} \approx 3\tau.$$

In the limit, we actually get $\omega \leq 3\tau$. In our case, $\tau \approx 0.85$ and $3\tau \approx 2.55$.

This proof generalizes to give Schönhage's τ-theorem:

$$\sum_i (n_i m_i p_i)^\tau = \underline{\mathrm{R}} \left(\bigoplus_i \langle n_i, m_i, p_i \rangle \right) \text{ implies } \omega \leq 3\tau. \tag{8}$$

An alternative form shows why it's worthy of its other name, the asymptotic sum inequality:

$$\sum_i (n_i m_i p_i)^{\omega/3} \leq \underline{\mathrm{R}}\left(\bigoplus_i \langle n_i, m_i, p_i\rangle\right). \tag{9}$$

Coppersmith and Winograd [CW, Corollary 3.2] showed that

$$R(\langle n,m,p\rangle \oplus \langle 1, R(\langle n,m,p\rangle) - m(n+p-1), 1\rangle) \leq R(\langle n,m,p\rangle) + m.$$

This shows that starting with an algorithm for $\langle n,m,p\rangle$, we can always obtain a better algorithm (yielding a better ω through the asymptotic sum inequality). In other words, no single algorithm for $\langle n,m,p\rangle$ can yield the optimal ω. A similar result of theirs from the same paper (Corollary 3.5) implies that the inequality in (8) is always strict.

6 Fast multiplication of rectangular matrices*

For this section, we change our focus and concentrate on multiplication problems of the type $\langle n,n,n^\alpha\rangle$. We want to find a value of α such that $R(\langle n,n,n^\alpha\rangle) = \tilde{O}(n^2)$.

Following Coppersmith's footsteps [Cop2], we consider another identity due to Schönhage:

$$\begin{aligned}
&\epsilon^2(a_{11}b_{11}c_{11} + a_{11}b_{12}c_{21} + a_{12}b_{21}c_{11} + a_{13}b_{31}c_{11} + a_{22}b_{21}c_{12} + a_{23}b_{31}c_{12}) + O(\epsilon^3) = \\
&(a_{11} + \epsilon^2 a_{12})(b_{21} + \epsilon^2 b_{11})c_{11} + \\
&(a_{11} + \epsilon^2 a_{13})b_{31}(c_{11} - \epsilon c_{21}) + \\
&(a_{11} + \epsilon^2 a_{22})(b_{21} - \epsilon b_{12})c_{12} + \\
&(a_{11} + \epsilon^2 a_{23})(b_{31} + \epsilon b_{12})(c_{12} + \epsilon c_{21}) - \\
&a_{11}(b_{21} + b_{31})(c_{11} + c_{12}).
\end{aligned}$$

This identity describes a fast way to approximately multiply a partial 2×3 matrix with a partial 3×2 matrix using only five multiplications:

$$\begin{pmatrix} a_{11} & a_{12} & a_{13} \\ 0 & a_{22} & a_{23} \end{pmatrix} \begin{pmatrix} b_{11} & b_{12} \\ b_{21} & 0 \\ b_{31} & 0 \end{pmatrix} = \begin{pmatrix} c_{11} & c_{12} \\ c_{21} & c_{22} \end{pmatrix}.$$

It is important here that this identity is tight: there is a trivial matching lower bound for the border rank, coming from the number of indeterminates in the A matrix.

Take this identity and tensor it N times to get a new identity involving 5^N multiplications, showing how to multiply a partial $2^N \times 3^N$ matrix with a partial $3^N \times 2^N$ matrix.

These matrices have a very complicated pattern of zeroes, but we will only be interested in the number of non-zero entries in each column of the first matrix, and the number of non-zero entries in each row of the second matrix. There are $N+1$ different types of such indices (does that sound familiar?). Type i involves $m_i = \binom{N}{i}2^i$ columns of the first matrix with $n_i = 2^i$ non-zero entries, and matching rows of the second matrix with $p_i = 2^{N-i}$ non-zero entries.

Suppose we group together all indices of type i, and zero all other entries in the two matrices. We get an identity for approximately computing AB, where A consists of m_i columns, each having

n_i non-zero entries, and B consists of m_i rows, each having p_i non-zero entries. So it's morally a tensor of type $\langle n_i, m_i, p_i \rangle$. In fact, we can actually encode $\langle n_i, m_i, p_i \rangle$ inside the product AB (see below). This shows that $\underline{\mathrm{R}}(\langle n_i, m_i, p_i \rangle) \leq 5^N$. Symmetrize once to get

$$\underline{\mathrm{R}}(\langle n_i m_i, n_i m_i, p_i^2 \rangle) \leq 5^{2N}.$$

We would like $n_i m_i \approx 5^N$. Since $n_i m_i = \binom{N}{i} 4^i$, $n_i m_i$ is maximized for $i = 4N/5$, at which point $n_i m_i = \Theta(5^N/\sqrt{N})$. For this i we have $p_i^2 = 4^{N/5} = 5^{\alpha N}$ for $\alpha = (\log_5 4)/5 \approx 0.17227$. Concluding,

$$\underline{\mathrm{R}}\left(\left\langle \frac{5^N}{\sqrt{N}}, \frac{5^N}{\sqrt{N}}, 5^{\alpha N} \right\rangle\right) \leq 5^{2N}.$$

Taking $n = 5^N/\sqrt{N}$,

$$\underline{\mathrm{R}}(\langle n, n, n^\alpha \rangle) = O(n^2 \log n).$$

This implies that

$$R(\langle n, n, n^\alpha \rangle) = O(n^2 \log^2 n).$$

The usual tensoring trick shows that one can multiply an $n \times n^\alpha$ matrix by an $n^\alpha \times n$ matrix using $O(n^2 \log^2 n)$ arithmetic operations. Moreover, as observed by Ryan Williams [Wil], we can describe all these transformations uniformly. Ryan Williams used fast matrix multiplication to solve SAT on certain "symmetric" circuits on all inputs very quickly. Later, Andreas Björklund found a more elementary dynamic programming accomplishing the same task, so that fast matrix multiplication is no longer needed.

6.1 Proof

It remains to prove that $\langle n, m, p \rangle$ matrix multiplication can be embedded in AB matrix multiplication if A consists of m columns, each having n non-zero entries, and B consists of m rows, each having p non-zero entries.

Let A' be an $n \times m$ matrix. We will construct a (constant) matrix T translating between A' and A: given A', we will be able to find A so that $A' = TA$. The same construction will also give a matrix S translating between an arbitrary $m \times p$ matrix B' and B, in the sense that $B' = BS$. We can then reduce $A'B'$ to AB using $A'B' = T(AB)S$.

How do we find the matrix T? We show one way to do this for small numbers, the general case being similar. Suppose that $n = 2$ and that A has three rows, so the matrix T has dimensions 2×3. Our matrix T will be

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

The matrix T corresponds to the linear transformation $T(x, y, z) = (x + y, x + z)$. This linear transformation has the property that, given the constraint that only two of x, y, z are non-zero, we can still encode any pair (a, b):

$$\begin{aligned} T(a, b-a, 0) &= (a, b) \\ T(a, 0, b-a) &= (a, b) \\ T(0, a, b) &= (a, b) \end{aligned}$$

Now take each column of the matrix A', and use the appropriate formula to encode it into a column of A. This encoding ensures that $A' = TA$. More generally, the property we need from the matrix T is the all square minors are non-zero. This is certainly the case for a random matrix T.

6.2 Alternative method

In the same paper describing the preceding construction, Coppersmith also described a slightly inferior construction giving only $\alpha \approx 0.1402$. However, the same method, combined with the identity and methods of [CW], gives a much better result, $\alpha \approx 0.29462$. This has recently been improved by Le Gall [Gal] to $\alpha \approx 0.30298$. These constructions apparently cannot be used for Williams' result since they only give $O(n^{2+\epsilon})$ algorithms for any $\epsilon > 0$.

The idea is to start with an identity by Schönhage which we have already considered (with different parameters):

$$\epsilon^2 \left(\sum_{i=1}^{3} \sum_{j=1}^{3} A_i B_j C_{ji} + \sum_{i=1}^{2} \sum_{j=1}^{2} X_{2i+j} Y_{2i+j} Z \right) + O(\epsilon^3) =$$
$$\sum_{i=1}^{2} \sum_{j=1}^{2} (A_i + \epsilon X_{2i+j})(B_j + \epsilon Y_{2i+j})(\epsilon^2 C_{ji} + Z) +$$
$$\sum_{i=1}^{2} A_i (B_3 - \epsilon \sum_{j=1}^{2} Y_{2i+j})(\epsilon^2 C_{3i} + Z) + \sum_{j=1}^{2} (A_3 - \epsilon \sum_{i=1}^{2} X_{2i+j}) B_j (\epsilon^2 C_{j3} + Z) +$$
$$A_3 B_3 (\epsilon^2 C_{33} + Z) - \left(\sum_{i=1}^{3} A_i \right) \left(\sum_{j=1}^{3} B_j \right) Z.$$

This identity shows that

$$\underline{\mathrm{R}}(\langle 3,1,3 \rangle \oplus \langle 1,4,1 \rangle) \leq 10.$$

Again, this identity is tight. The idea now is to take a high tensor power *without symmetrizing*:

$$\underline{\mathrm{R}} \left(\sum_{j=0}^{N} \binom{N}{j} \langle 3^j, 4^{N-j}, 3^j \rangle \right) \leq 10^N.$$

The idea now is to choose an appropriate j and zero everything else. Which j should we pick? The same method used for proving the asymptotic sum inequality shows that for large M,

$$\underline{\mathrm{R}}(\langle 3^{jM}, 4^{(N-j)M}, 3^{jM} \rangle) \lessapprox \frac{10^{NM}}{\binom{N}{j}^M}.$$

In order to get a tight bound, we would like the right-hand side to be approximately $(3^{jM})^2$. In other words, taking the Mth root, we want $\binom{N}{j} 9^j \approx 10^N$. When proving the asymptotic sum inequality, it was enough to comment that such a j can be found by taking the maximum over $\binom{N}{j} 9^j$, since the sum of these $N+1$ terms is 10^N. In this case we also need to know the value of j, which is roughly $j \approx 0.9N$. This shows that roughly speaking, $\underline{\mathrm{R}}(\langle 3^{0.9NM}, 4^{0.1NM}, 3^{0.9NM} \rangle) \lessapprox 9^{0.9NM}$. Putting $n = 3^{0.9NM}$, the other index is $4^{0.1NM} = n^\alpha$ for $\alpha = 0.1 \log 4 / 0.9 \log 3 \approx 0.1402$. When unrolling the construction, for technical reasons we only get a bound of the form $O(n^{2+\epsilon})$ rather than $\tilde{O}(n^2)$ as before.

References

[ACT] Peter Bürgisser, Michael Clausen and M. Amin Shokrollahi, *Algebraic Complexity Theory*, Springer, 1997.

[Blä] Markus Bläser, *Complexity of bilinear problems (lecture notes scribed by Fabian Bendun)*, `http://www-cc.cs.uni-saarland.de/teaching/SS09/ComplexityofBilinearProblems/script.pdf`, 2009.

[Cop] Dan Coppersmith, *Rapid multiplication of rectangular matrices*, SIAM J. Comput. 11:467–471, 1982.

[Cop2] Dan Coppersmith, *Rectangular matrix multiplication revisited*, J. Comp. 13:42–49, 1997.

[CW] Dan Coppersmith, Shmuel Winograd, *On the asymptotic complexity of matrix multiplication*, SIAM J. Comput. 5:618–623, 1976.

[Gal] Fran cois Le Gall, *Faster algorithms for rectangular matrix multiplication*, ArXiv, 2012.

[Wil] Ryan Williams, *Non-uniform ACC circuit lower bounds*, CCC 2011.