E0309 Topics in Complexity Theory **Spring 2015**

Lecture 3: January 28, 2015

Lecturer: Neeraj Kayal *Scribe: Abhijat Sharma*

3.1 Matrix Multiplication

In the previous lecture, we discussed the complexity of *Matrix Vector Multiplication* ,which was the following problem. Given a fixed matrix $A \in \mathbb{F}^{m \times m}$,
Input: $x = (x_1, x_2, ..., x_m) \in \mathbb{F}^m$
Output: $y = A.x \in \mathbb{F}^m$
We were particularly interested in this problem when the matrix A was a *Vandermonde Matrix*, because that is the case we need to take care of during the *evaluation and interpolation* steps of polynomial multiplication.

In this lecture, we look at a much more general problem, *Matrix Multiplication* which is an integral part of various other complex computational problems. It is defined as follows:
Input: 2 matrices $X, Y \in \mathbb{F}^{n \times n}$
Output: $Z = X.Y$, the product matrix

One of the most direct applications of an efficient matrix multiplication algorithm is the *Graph Reachability* problem.
Input: A directed graph, $G = (V, E)$ represented as an adjacency matrix, A such that

$$A_{ij} = \begin{cases} 1 & \text{if } (i,j) \in E \\ 0 & \text{otherwise} \end{cases}$$

Output: The reachability matrix B such that

$$B_{ij} = \begin{cases} 1 & \text{if there is a path in } G \text{ from vertex } i \text{ to vertex } j \\ 0 & \text{otherwise} \end{cases}$$

where A_{ij} and B_{ij} refer to the jth entry in the ith row of matrices A and B respectively. There is a simple algorithm to solve the above problem, once we observe that if we compute the matrix A^k (for any integer k), then the element $(A^k)_{ij}$ will be a non-zero quantity iff there is a path from i to j of length at-most k. Thus, all we need to do is compute the matrix A^n from A by repeated squaring, which would require $\log n$ matrix multiplications, and as the length of any path can be at-most n, the required output matrix B can be computed from the matrix A^n as follows: for all $i, j \in [n]$, $B_{ij} = 1$ iff $(A^n)_{ij} \neq 0$. Hence, the complexity of graph reachability would be $\log n$ times the complexity of matrix multiplication.

Remark: Also, it is worthwhile to note here that matrix multiplication reduces to matrix powering. Suppose we have two $n \times n$ matrices A and B. We create a $2n \times 2n$ matrix $C = \begin{bmatrix} 0 & A \\ B & 0 \end{bmatrix}$, i.e the upper right sub-matrix of C would have all the entries of A, the lower left sub-matrix will have the entries of matrix B and all the other entries in C will be zero. We can compute both products AB and BA by just computing the square of this matrix C.

$$\begin{bmatrix} 0 & A \\ B & 0 \end{bmatrix} \begin{bmatrix} 0 & A \\ B & 0 \end{bmatrix} = \begin{bmatrix} AB & 0 \\ 0 & BA \end{bmatrix}$$

Let us look at an upper bound, how high can the complexity of multiplying two $n \times n$ matrices be. Observe that there are $2n^2$ inputs (n^2 in each input matrix), and the output matrix contains n^2 output elemants. The most trivial algorithm involves computing each output independently, by the following formula:

$$z_{ij} = \sum_{k=1}^{n} x_{ik}.y_{kj}$$

where z_{ij} refers to the usual notation of the entry in the output matrix Z, corresponding to row i and column j. Thus, each output computes n multiplications, and then adds them up, requiring total $O(n)$ operations. As there are n^2 such outputs, the overall complexity of this trivial algorithm is $O(n^3)$. For a long time, it was thought that this was optimal, but in 1970, Strassen came up with a substantially improved algorithm which computed the product of two $n \times n$ matrix in $O(n^{\log_2 7})$ operations.

3.2 Strassen's Algorithm (1969)

The Strassen's algorithm for matrix multiplication is a remarkable example of the recursive technique: *Divide and Conquer*. Assume that n is a power of 2, we divide each of the two input $n \times n$ matrices X and Y into four smaller $n/2 \times n/2$ sub-matrices. Also, we compute the output matrix Z as 4 smaller $n/2 \times n/2$ matrices Z_1, Z_2, Z_3, Z_4. Then, the equation $Z = X.Y$ becomes

$$\begin{bmatrix} Z_{11} & Z_{12} \\ Z_{21} & Z_{22} \end{bmatrix} = \begin{bmatrix} X_{11} & X_{12} \\ X_{21} & X_{22} \end{bmatrix} \begin{bmatrix} Y_{11} & Y_{12} \\ Y_{21} & Y_{22} \end{bmatrix} = \begin{bmatrix} X_{11}.Y_{11} + X_{12}.Y_{21} & X_{11}.Y_{12} + X_{12}.Y_{22} \\ X_{21}.Y_{11} + X_{22}.Y_{21} & X_{21}.Y_{12} + X_{22}.Y_{22} \end{bmatrix}$$

where all the X_{ij}'s and Y_{ij}'s are $n/2 \times n/2$ matrices. Thus, each of the four parts of the output matrix requires two smaller multiplications and then addition of the two obtained products. So, to compute the complete product matrix Z, we need eight recursive multiplications of $n/2 \times n/2$ matrices and $O(n^2)$ additions. Hence, if $T(n)$ denotes the time complexity of multiplying two $n \times n$ matrices, it can be expressed as the following recurrence:

$$T(n) = 8.T(n/2) + O(n^2)$$

On solving the above recurrence, we get $T(n) = O(n^3)$, which is no faster than the trivial algorithm.

However, suppose we could compute the four sub-matrices Z_1, Z_2, Z_3, Z_4, using less than eight recursive multiplications, say seven. Then, assuming additions and scalar multiplications amount to $O(n^2)$ operations, the above recurrence will become:

$$T(n) = 7.T(n/2) + O(n^2)$$

which yields the result, $T(n) = O(n^{\log_2 7}) = O(n^{2.81})$ which is a definite improvement in the earlier asymptotic time complexity. This is exactly how the Strassen's algorithm works. In every intermediate recursive step, it starts with 8 inputs, $X_{11}, X_{12}, X_{21}, X_{22}, Y_{11}, Y_{12}, Y_{21}, Y_{22}$, and computes the following 7 products of half the size:

$$\begin{aligned} M_1 &= (X_{11} + X_{22}).(Y_{11} + Y_{22}) \\ M_2 &= (X_{21} + X_{22}).Y_{11} \\ M_3 &= X_{11}.(Y_{12} - Y_{22}) \\ M_4 &= X_{22}(Y_{21} - Y_{11}) \\ M_5 &= (X_{11} + X_{12}).Y_{22} \\ M_6 &= (X_{21} - X_{11}).(Y_{11} + Y_{12}) \\ M_7 &= (X_{12} - X_{22}).(Y_{21} + Y_{22}) \end{aligned}$$

Claim 3.1 *All the required outputs $Z_{11}, Z_{12}, Z_{21}, Z_{22}$ belong to the linear subspace spanned by $M_1, M_2, ..., M_7$.*

Proof: When we consider the linear subspace spanned by the matrices $M_1, M_2, ..., M_7$, we know that it contains only those matrices which can be obtained usng linear combinations of these 7 matrices. We show that our output sub-matrices obey the above constraint:

$$Z_{11} = M_1 + M_4 - M_5 + M_7$$
$$Z_{12} = M_3 + M_5$$
$$Z_{21} = M_2 + M_4$$
$$Z_{22} = M_1 - M_2 + M_3 + M_6$$

The above equations are obtained by solving a simple system of linear equations, with variables as the coefficients of the M_i's above. ■

Now, having seen that reducing the number of intermediate recursive multiplications from 8 to 7 leads to a significant improvement in the time complexity of the algorithm, it is natural to ask whether it can be further optimized by computing less than 7 multiplications. Interestingly, it has been proved that we need at-least 7 multiplications for computing the product of two 2×2 matrices[2]. It is also interesting to note that the above 7 products $M_1, M_2, ..., M_7$ form the only way of achieving this improvement, and there is no other way of multiplying the intermediate sub-matrices such that $Z_{11}, Z_{12}, Z_{21}, Z_{22}$ are obtained by simple linear combinations of the products. We now present a possible explanation of this lower bound of 7 multiplications, by showing how computation of polynomials is related to the concept of tensor-rank.

3.3 Tensors and Tensor-Rank

Given a polynomial in $2n$ variables, $p(x_1, x_2, ..., x_n, y_1, ..., y_n) = \sum_{i,j \in [n]} \alpha_{ij} x_i y_j$ (such a polynomial is called a *bilinear polynomial*), we ask what is the fewest number of multiplications required to compute p, such that every multiplication has one multiplicand as a linear combination of x_i's, and other multiplicand as a linear combination of y_j's. We define the $n \times n$ matrix corresponding to the polynomial p, $M(p)$ such that every entry in $M(p)$ corresponds to a coefficient of the polynomial. Precisely, the jth entry of the ith column in $M(p)$, $(M(p))_{ij} = \alpha_{ij}$.

Claim 3.2 *Minimum number of such multiplications required to compute p is equal to* **rank** *of the matrix $M(p)$*[1].

Before we prove the above claim, let us first define the notion of matrix rank as follows: rank of a matrix M, $rank(M) = r$ iff M can be expressed as the sum of r matrices, $M = M_1 + M_2 + ... + M_r$, where each matrix M_k is a **rank one** matrix, i.e it can be expressed as a *tensor product* of two vectors $u_k, v_k \in \mathbb{F}^n$. (Tensor product, $M_k = u_k \otimes v_k$ implies $M_k(a, b) = u_k(a).v_k(b)$ for $a, b \in [n]$)

Note that the above notion complies with the familiar notion of matrix rank, being equal to the number of linearly independent columns (or rows). By that notion, a rank one matrix implies that all columns are constant multiples of a particular column, say u, then the matrix can be written as $M = [\ v_1.u\ v_2.u\ ...\ v_n.u]$, where $v_1, v_2, ...v_n$ are scalars. Observe that this is the same as writing M as a tensor product $M = u \otimes v$, where $v = (v_1, v_2, ..., v_n)$ is a vector, and that is how we have now defined rank one matrices. Similarly, by the previous notion, if a matrix has rank r, every column is a linear combination of r fixed columns, say $u_1, u_2, ..., u_r$, and therefore can be written as a sum of r rank-one matrices, each corresponding to a particular u_i, exactly as defined by the new notion.

Proof:[Proof of claim 3.2]
Let r be the rank of the matrix $M(p)$, then by the above definition of rank, $M(p) = M_1 + M_2 + ... + M_r$, where each M_k is a tensor-product, $M_k = u_k \otimes v_k$. Now, consider the polynomial p_k $(k = 1, 2, ..., r)$ corresponding to the matrix M_k, just as p corresponds to the matrix $M(p)$. Observe that the entry $(M_k)_{ij} = u_k(i).v_k(j)$, is equal to the coefficient of $x_i y_j$ in polynomial p_k. Hence, p_k can be written as a product of two linear polynomials

$$p_k = (u_k \odot \underline{X}).(v_k \odot \underline{Y})$$

where $\underline{X} = (x_1, x_2, ..., x_n)$ and $\underline{Y} = (y_1, y_2, ..., y_n)$ are vectors and $\odot$ refers to dot-product. As $u_k \odot \underline{X}$ and $v_k \odot \underline{Y}$ are only linear combinations of the variables $x_1, x_2, ..., x_n, y_1, ..., y_n$, only one multiplication is required to compute the polynomial p_k. Now, because of the way these matrices are related to their corresponding bilinear polynomials, if $M(p) = M_1 + M_2 + ... + M_r$, it directly implies $p = p_1 + p_2 + ... + p_r$ (every entry of matrix corresponds to coefficient of a particular monomial). Thus, from the above correlation, if every p_k requires 1 multiplication, then r multiplications would suffice for the overall computation of the polynomial p. Thus, if the minimum number of multiplications required is s (say), The above argument implies that $s \leq r$.

On the other hand, s is the minimum number of required multiplications, then we can write

$$p = q_1 + q_2 + ... + q_s$$

where every q_i is a product of a linear form in $\underline{X}$ and a linear form in $\underline{Y}$. Rewriting the above equation in terms of the matrices corresponding to bilinear polynomials (as defined above), we get $M(p) = M(q_1) + M(q_2) + ... + M(q_s)$, where $M(q_i)$ is a rank-one matrix for all $i = 1, 2, ..., s$. However, if $M(p)$ has rank r, then r is the minimum number of rank-one matrices $M_1, M_2, ..., M_r$ such that $M_1 + M_2 + ... + M_r = M$. Thus, it is clear to see that, by definition of matrix-rank, $s \geq r$. Combining the above two arguments, we get $s = r$ which completes the proof of claim 3.2.

■

From the above discussion, we have a definite lower bound for the number of multiplications required to compute any bilinear polynomial. Now, consider the case when we have a bunch of bilinear polynomials, $p_1(\underline{X}, \underline{Y}), p_2(\underline{X}, \underline{Y}), ..., p_n(\underline{X}, \underline{Y})$ where $\underline{X} = (x_1, x_2, ..., x_n)$ and $\underline{Y} = (y_1, y_2, ..., y_n)$ are vectors. This introduces us to the concept of **tensors**, which can be thought to be a generalization of matrices for higher dimensions.

Formally, a tensor is a function $T : [n]^r \to \mathbb{F}$. It is easy to see that a matrix is a special case of a tensor where $r = 2$. Thus, just like we defined a matrix corresponding to one polynomial, we define a 3-dimensional tensor $M : [n]^3 \to \mathbb{F}$, corresponding to the bunch of polynomials $p_1, p_2, ..., p_n$ such that $M(i, j, k)$ is equal to the coefficient of the monomial $x_i y_j$ in the polynomial p_k. Similar to the correspondence between matrix-rank and a polynomial, we make the following claim:

Claim 3.3 *If m is the minimum number of multiplications (i.e product of two linear forms $l_1(\underline{X}, \underline{Y})$ and $l_2(\underline{X}, \underline{Y})$) required to compute the bunch of polynomials $p_1, p_2, ..., p_n$, (meaning, every p_i is a linear combination of these m products) and r is the tensor-rank of M, then*

$$r/2 \leq m \leq r$$

where *tensor-rank* is only a generalization of matrix-rank, defined similarly, as follows: The tensor $M : [n]^3 \to \mathbb{F}$ is said to have rank r if r is the minimum number such that M can be expressed as a sum of r rank one tensors, $M = M_1 + M_2 + ... + M_r$, and M_i is rank-one if it is a tensor product of 3 vectors, $M_i = u_i \otimes v_i \otimes w_i$ for all i $(M_i(a, b, c) = u_i(a).v_i(b).w_i(c)$ for $a, b, c \in [n])$.

Proof:[Proof of Claim 3.3]
We will abuse notation slightly and write the tensor M equivalently as the following degree-3 polynomial in $3n$ variables $x_1, ..., x_n, y_1, ..., y_n, z_1, ..., z_n$:

$$M = z_1p_1 + z_2p_2 + ... + z_np_n$$

This polynomial is just an equivalent/alternate representation of the tensor M defined earlier because every term in the above expression is a monomial of the form $x_iy_jz_k$, and the coefficient of this monomial is exactly the entry $M(i, j, k)$. We know that the above tensor M has rank r, and therefore it can be written as a sum of r rank-one tensors as follows:

$$M = \sum_{l=1}^{r} u_l \otimes v_l \otimes w_l$$

Let M_l denote the lth term in the above sum, i.e $M_l = u_l \otimes v_l \otimes w_l$ and $M_l(i, j, k) = u_l(i).v_l(j).w_l(k)$. On rewriting the tensor M_l as a degree 3-polynomial like before, we get

$$\begin{aligned} M_l &= (\sum_{i\in[n]} u_l(i).x_i).(\sum_{j\in[n]} v_l(j)y_j).(\sum_{k\in[n]} w_l(k).z_k) \\ &= (u_l \odot \underline{X}).(v_l \odot \underline{Y}).(w_l \odot \underline{Z}) \end{aligned}$$

which implies

$$M = \sum_{l=1}^{r} (u_l \odot \underline{X}).(v_l \odot \underline{Y}).(w_l \odot \underline{Z})$$

expanding $w_l \odot \underline{Z}$, and collecating coefficients of z_k together,

$$= \sum_{k=1}^{n} z_k \sum_{l=1}^{r} w_l(k).(u_l \odot \underline{X}).(v_l \odot \underline{Y})$$

which finally means

$$\sum_{k=1}^{n} z_kp_k = \sum_{k=1}^{n} z_k \sum_{l=1}^{r} w_l(k).(u_l \odot \underline{X}).(v_l \odot \underline{Y})$$

where old notations have been borrowed for dot-product and other vectors denoting variable-sets. From the final equation, we can easily see that every p_k $(k = 1, 2, ..., n)$ belongs to the linear span of $\{(u_l \odot \underline{X}).(v_l \odot \underline{Y}), l \in [r]\}$. So, $m \leq r$.

Now, we need to prove the lower bound on the required multiplications m. We have represented the tensor M as the polynomial

$$M = \sum_{k=1}^{n} z_kp_k(\underline{X}, \underline{Y}) \tag{3.1}$$

And by definition, m is the minimum number of polynomials $q_1(\underline{X}, \underline{Y}), ..., q_m(\underline{X}, \underline{Y})$ (each q_i is a product of two linear forms in the variables $x_1, x_2, ..., x_n, y_1, ..., y_n$), such that for all k, $p_k = \sum_{i=1}^{m} c_{ik}q_i(\underline{X}, \underline{Y})$ (a linear combination of the q_is). So, by replacing the expansion of every p_k in equation 3.1, we can rewrite M as

$$M = \sum_{i=1}^{m} f(z)q_i(\underline{X}, \underline{Y}) = \sum_{i=1}^{m} f(z)g_i(\underline{X}, \underline{Y})h_i(\underline{X}, \underline{Y}) \tag{3.2}$$

where g_i and h_i are linear polynomials in variables $x_1, x_2, ..., x_n, y_1, ..., y_n$ and $q_i = g_i.h_i$ for all i.

Further, we break $g_i(\underline{X}, \underline{Y})$ into $g1_i(\underline{X})$ and $g2_i(\underline{Y})$, and $h_i(\underline{X}, \underline{Y})$ into $h1_i(\underline{X})$ and $h2_i(\underline{Y})$ similarly. Substituting this in equation 3.2, we get

$$M = \sum_{i=1}^{m} f(z)(g1_i(\underline{X}) + g2_i(\underline{Y}))(h1_i(\underline{X}) + h2_i(\underline{Y})) \tag{3.3}$$

$$= \sum_{i=1}^{m} f(z)g1_i(\underline{X})h2_i(\underline{Y}) + \sum_{i=1}^{m} f(z)g2_i(\underline{Y})h1_i(\underline{Y}) \tag{3.4}$$

and the monomials of the form $x_i x_j$ and $y_i y_j$ cancel out because the left hand side polynomial (tensor M) contains only monomials of the form $z_k x_i y_j$. From the above equation 3.4, it is clear that the tensor cannot be a sum of more than $2m$ rank-one tensor, and hence proved that $r \leq 2m \Rightarrow r/2 \leq m$.

■

3.3.1 Tensor Rank and Matrix Multiplication

From the above discussion, it is easy to observe that in the process of computing the product of two $n \times n$ matrices, using a Strassen-inspired divide and conquer recursive algorithm, we essentially compute four polynomials in each step of recursion, $Z_{11}, Z_{12}, Z_{21}, Z_{22}$, in the 8 variables $X_{11}, X_{12}, X_{21}, X_{22}, Y_{11}, Y_{12}, Y_{21}, Y_{22}$. Therefore, by claim 3.3, the minimum number of computations in this process corresponds to the rank of a $(4 \times 4 \times 4)$ tensor ($T : [n]^3 \to \mathbb{F}$ for $n = 4$). Tensors of the form $T : [n]^k \to \mathbb{F}$ are known as *order-k* tensors. Hence, for analyzing matrix multiplication, we would be concerned primarily with order-3 tensors.

Unlike Matrix-rank, which is fairly easy to compute using techniques like Gaussian Elimination, Tensor-Rank computation has been proven to be an NP-Complete problem by Håstad[3]. However, we can try to get some upper bound on the rank of a 3-dimensional (order-3) tensor which we require in case of matrix multiplication. For that, we first observe that over a given finite field $\mathbb{F}$ of cardinality q (say), the total possible $(n \times n \times n)$ tensors are q^{n^3}. Among these, let us consider the number of possible rank-one tensors, i.e the number of different ways to choose 3 vectors u, v, w, each of length n. This gives us $3n$ *degrees of freedom* (number of values that can be arbitrarily chosen). However, observe that $2.2.2 = 1.1.8$, so choosing $3n$ field elements arbitrarily need not always produce a distinct tensor $u \otimes v \otimes w$. To take care of this, while choosing the 3 vectors ($u = (u_1, u_2, ..., u_n)$, $v = (v_1, v_2, ..., v_n)$ and $w = (w_1, w_2, ..., w_n)$), we restrict the first elements of u and v, $u_1 = v_1 = 1$, by dividing the entire vector appropriately. This gives us only $3n-2$ degrees of freedom and hence total possible rank-one vectors are q^{3n-2}. Now, given any random 3-dimensional tensor, if it is of rank r, it can be written as a sum of r rank one tensors. Therefore, $r(3n-2) = n^3$, which implies:

$$r = \frac{n^3}{3n-2}$$

Substituting $n = 4$ gives us the rank of a $(4 \times 4 \times 4)$ matrix to be approximately equal to 7, justifying Strassen's choice. Thus, counting arguments like the one show that with high probability, most order-3 tensors have rank $\Omega(n^2)$. However, it is an open problem to find an explicit three-dimensional tensor whose rank is $\omega(n)$.

Specifically, the rank of the matrix multiplication tensor corresponding to multiplication of two 2×2 matrices, represented as the polynomial, $(x_{11}y_{11} + x_{12}y_{21})z_1 + (x_{11}y_{12} + x_{12}y_{22})z_2 + (x_{21}y_{11} + x_{22}y_{21})z_3 + (x_{21}y_{12} + x_{22}y_{22})z_4$ has been proved to be 7 by Landsberg(2006), but for the tensor corresponding to 3×3

matrix multiplication, it is still unknown. The 3×3 matrix multiplication tensor corresponds to matrix multiplication when at every recursion step, all matrices are broken into 9 sub-matrices of size $(n/3 \times n/3)$, instead of four $(n/2 \times n/2)$ sub-matrices as in Strassen's algorithm. A trivial upper bound to rank of the above tensor is 27, and currently known upper bound is 23, and best known lower bound is 19[4]. There have been constant efforts to improve the complexity of multiplication of two $n \times n$ matrices, mostly based on techniques that try to generalise Strassen's algorithm. The current best known algorithm by Francois Le Gall[5] has complexity $O(n^{2.372})$, which is a generalisation of *Coppersmith-Winograd* algorithm.

3.4 Parallelization of Computation

We now introduce another topic of how the basic simple operations of integer addition and multiplication, can be computed by much time-efficient algorithms if we allow parallelization i.e the possibility of multiple operations being processed at the same instant of time. Also, to simulate parallel computation, we use the computational model of boolean circuits. We will formally introduce these models in the next lecture.

Now, let us just observe the process of addition of two integers, n-bit each.
Input: 2 integers $a = a_{n-1}a_{n-2}...a_1a_0$ and $b = b_{n-1}b_{n-2}...b_1b_0$.
Output: The sum of a and b, $c = c_nc_{n-1}...c_1c_0$.
We know that trivially, by a non-parallelized algorithm, this can be done in $O(n)$ time. We will prove in the next lecture how this can also be done in constant time, if we allow parallel computation. More formally, Integer Addition can be solved by *constant-depth* polynomial sized boolean circuits having *unbounded fan-in*. The terms used above will be explained in the next lecture.

3.5 References

[1] Amir Shpilka and Amir Yehudayaoff, Arithmetic Circuits: A survey of recent results and open questions, 2010

[2] J.M. Landsberg, Geometry and the Complexity of Matrix Multiplication, *Bulletin of the American Mathematical Society*, 2010

[3] Johan Håstad, Tensor Rank is NP-Complete, *Journal of Algorithms*, 1990

[4] Marcus Blaser, A $5/2n^2$-Lower Bound for the Multiplicative Complexity of $n \times n$-Matrix Multiplication , *Proceedings of the 40th Annual Symposium on Foundations of Computer Science (FOCS)*, New York, 1999

[5] Francois Le Gall, Power of Tensors and Fast Matrix Multiplication, *Proceedings of the 39th International Symposium on Symbolic and Algebraic Computation (ISSAC)*, 2014