Algebraic Algorithms

1. Fast matrix multiplication
2. Inverting matrices
3. Fast Fourier Transform

1. Matrix multiplication

Let A, B be $(n \times n)$ matrices. Then

$$C = A \cdot B \qquad \text{with} \qquad c_{ij} = \sum_{k=1}^{n} a_{ik} \cdot b_{k_j}.$$

Conventional methods:

iteratively: Compute c_{ij} according to the above formula.
<u>Outcome:</u> n multiplications, $n-1$ additions i.e. $2n-1$ arithmetic operations for each pair i, j.
$\Rightarrow n^2(2n-1) = O(n^3)$ operations.

recursive: (Divide and Conquer)
Split A, B into 4 $\left(\frac{n}{2} \times \frac{n}{2}\right)$ matrices. (W.l.o.g. let $n = 2^k$).

$$A \cdot B = \begin{pmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{pmatrix} \cdot \begin{pmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{pmatrix} = \begin{pmatrix} C_{11} & C_{12} \\ C_{21} & C_{22} \end{pmatrix} = C$$

where:

$$\begin{aligned} C_{11} &= A_{11}B_{11} + A_{12}B_{21} \\ C_{12} &= A_{11}B_{12} + A_{12}B_{22} \\ C_{21} &= A_{21}B_{11} + A_{22}B_{21} \\ C_{22} &= A_{21}B_{12} + A_{22}B_{22}. \end{aligned}$$

<u>Outcome:</u>

$$\begin{aligned} T(n) &= 8 \cdot T\left(\frac{n}{2}\right) + 4\left(\frac{n}{2}\right)^2 \\ &= O\left(n^{\log_2 8}\right) = O(n^3). \end{aligned}$$

Strassen's algorithm

<u>Idea:</u> Use Divide and Conquer approach for computing C_{ij}, $i, j \in \{1, 2\}$, but with a smaller number ($m \leq 7$) of multiplications and more ($a \geq 4$) additions:

$$T(n) = m \cdot T\left(\frac{n}{2}\right) + a\left(\frac{n}{2}\right)^2 = O\left(n^{\log_2 m}\right),$$

Theorem 1 *Two $(n \times n)$ matrices can be multiplied by using 7 multiplications and 18 additions.*

Corollary 1 *The running time of the Strassen's algorithm is $O\left(n^{\log_2 7}\right) \approx O(n^{2.81})$.*

Remark 1 *There exist better algorithms with running time $O(n^{2.38})$. The lower bound is $\Omega(n^2)$, since n^2 matrix elements need to be computed.*

Proof of Theorem 1:

Let $n = 2^k$. Denote:

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

One has:

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

If $2^{k-1} < n < 2^k$ the we fill out the matrices A, B up to $(2^k \times 2^k)$ matrices. This increases the running time up to $d \cdot n^{\log_2 7}$ for a constant $d > 0$. $\square$

Remark 2 *The constant d is pretty large. As it follows from practice, the conventional method is better for $n <\approx 1000$.*

Remark 3 *The exact value of the minimum running time necessary to multiply two matrices in presently unknown.*

2. Computing the matrix inverse

Let A be an $(n \times n)$ matrix. We compute the matrix A^{-1} such that $A \cdot A^{-1} = A^{-1} \cdot A = I_n$. If A^{-1} does exist, the matrix A is called nonsingular. In in this case A^{-1} is defined uniquely.

Denote by $M(n)$ the time for the matrix multiplication and by $I(n)$ the time for computing the inverse.

Theorem 2 *If* $I(3n) = O(I(n))$ *then:*

$$M(n) = O(I(n)).$$

Proof. Given matrices A, B construct the matrix

$$D = \begin{pmatrix} I_n & A & 0 \\ 0 & I_n & B \\ 0 & 0 & I_n \end{pmatrix}.$$

One has:

$$D^{-1} = \begin{pmatrix} I_n & -A & AB \\ 0 & I_n & -B \\ 0 & 0 & I_n \end{pmatrix}.$$

Since D is computable in $\Theta(n^2)$ time, $I(n) = \Omega(n^2)$ and D^{-1} is computable in time $O(I(3n)) = O(I(n))$ we have the Theorem. $\square$

Remark 4 *Since* $I(n) = \Theta\left(n^c \log^d n\right)$ *for some constants* $c > 0$ *and* $d \geq 0$*, one has* $I(3n) = O(I(n))$*.*

Theorem 3 *If* $M(n) = O(M(n+k))$ *for* $0 \le k \le n$ *then:*

$$I(n) = O(M(n)).$$

Proof. W.l.o.g. we assume $n = 2^p$. Indeed, if $2^{p-1} < n < 2^p$ then for $k = 2^p - n$:

$$\begin{pmatrix} A & 0 \\ 0 & I_k \end{pmatrix}^{-1} = \begin{pmatrix} A^{-1} & 0 \\ 0 & I_k \end{pmatrix}.$$

Therefore, the running time increases in a constant factor only.

An $(n \times n)$ matrix A is called <u>positive-definite</u> if $x^T A x > 0$ for any n-dimensional vector $x \neq 0$.

First assume A is symmetric and positive-definite. Split A into 4 $(n/2 \times n/2)$ matrices:

$$A = \begin{pmatrix} B & C^T \\ C & D \end{pmatrix}$$

and set

$$S = D - CB^{-1}C^T.$$

The matrices B and S are symmetric and positive definite, so they are non-singular (Lemmas 28.9 – 28.11 in the book).

Therefore, one has:

$$A^{-1} = \begin{pmatrix} B^{-1} + B^{-1}C^T S^{-1} C B^{-1} & -B^{-1}C^T S^{-1} \\ -S^{-1}CB^{-1} & S^{-1} \end{pmatrix}.$$

To compute A^{-1} in this approach one needs to compute the matrices $E = C \cdot B^{-1}$ and also the matrices

$$E \cdot C^T \qquad S^{-1} \cdot E \qquad E^T \cdot (S^{-1}E).$$

Hence:

$$\begin{aligned} I(n) &\le 2I(n/2) + 4M(n/2) + O(n^2) \\ &= 2I(n/2) + O(M(n)) = O(M(n)). \end{aligned}$$

If A is not symmetric, consider the matrix $A^T A$. For any non-singular matrix A the matrix $A^T A$ is symmetric and positive-definite (see Theorem 28.6 in the book).

Since $((A^T A)^{-1} A^T) A = (A^T A)^{-1} \cdot (A^T A) = I_n$, one has:

$$A^{-1} = (A^T A)^{-1} \cdot A^T.$$

To compute A^{-1} we first construct $A^T A$ and then compute $(A^T A)^{-1}$ by the above method.

Therefore, A^{-1} can be computed in time $O(M(n))$. $\square$

Remark 5 *It follows from Theorem 3 that the system of linear equations $Ax = b$ with a non-singular matrix A can be solved in time $O(M(n))$: construct A^{-1} and then compute $x = A^{-1}b$.*

3. Polynomials and FFT

Let $A(x) = \sum_{j=0}^{n-1} a_j x^j$ be a polynomial, where $a_j \in C$ (complex numbers), $j = 0, \ldots, n-1$.

For $B(x) = \sum_{j=0}^{n-1} b_j x^j$ consider

$$
\begin{aligned}
C(x) &= A(x) + B(x) = \sum_{j=0}^{n-1} c_j x^j \\
D(x) &= A(x) \cdot B(x) = \sum_{j=0}^{2n-2} d_j x^j
\end{aligned}
$$

where $c_j = a_j + b_j$ and $d_j = \sum_{k=0}^{j} a_k b_{j-k}$, $j = 0, \ldots, 2n-2$.

The polynomial $C(x)$ can be computed in time $\Theta(n)$. However, one needs to perform $\sum_{j=0}^{2n-2} j = O(n^2)$ steps to compute $D(x)$ according to the above formula.

Point-value representation of $A(x)$:

$$\{(x_0, y_0),\ (x_1, y_1), \ldots, (x_{n-1}, y_{n-1})\}, \quad y_k = A(x_k)$$

(we assume that all x_k are distinct).

Since

$$A(x_k) = a_0 + x_k(a_1 + x_k(a_2 + \cdots + x_k(a_{n-2} + x_k a_{n-1})) \cdots),$$

the point-value representation can be computed in $O(n^2)$ time.

Lemma 1 *For any point set*

$$\{(x_0, y_0),\ (x_1, y_1), \ldots, (x_{n-1}, y_{n-1})\}$$

there exist only one polynomial $A(x)$ of degree $\leq n$ such that $y_k = A(x_k)$ for $k = 0, \ldots, n$.

Proof.
We show that the system of linear equations

$$\begin{pmatrix} 1 & x_0 & x_0^2 & \cdots & x_0^{n-1} \\ 1 & x_1 & x_1^2 & \cdots & x_1^{n-1} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & x_{n-1} & x_{n-1}^2 & \cdots & x_{n-1}^{n-1} \end{pmatrix} \begin{pmatrix} a_0 \\ a_1 \\ \vdots \\ a_{n-1} \end{pmatrix} = \begin{pmatrix} y_0 \\ y_1 \\ \vdots \\ y_{n-1} \end{pmatrix}$$

has a unique solution $(a_0, a_1, \ldots, a_n)$.

The determinant of this system is the Vandermonde determinant which equals $\prod_{j<k} (x_k - x_j) \neq 0$ for distinct x_k, x_j. $\square$

The point-value representation allows to compute $C(x)$ and $D(x)$ faster. Since $C(x) = A(x) + B(x)$ and $D(x) = A(x) \cdot B(x)$, for any fixed x we can evaluate $C(x)$ and $D(x)$ in $O(n)$ time.

However, to compute $D(x)$ we need to know the values of $A(x)$ and $B(x)$ in $2n$ points (not just in n points).

Idea: Use the point-value representation for computing $D(x)$, if the coefficients of $A(x)$ and $B(x)$ are given.

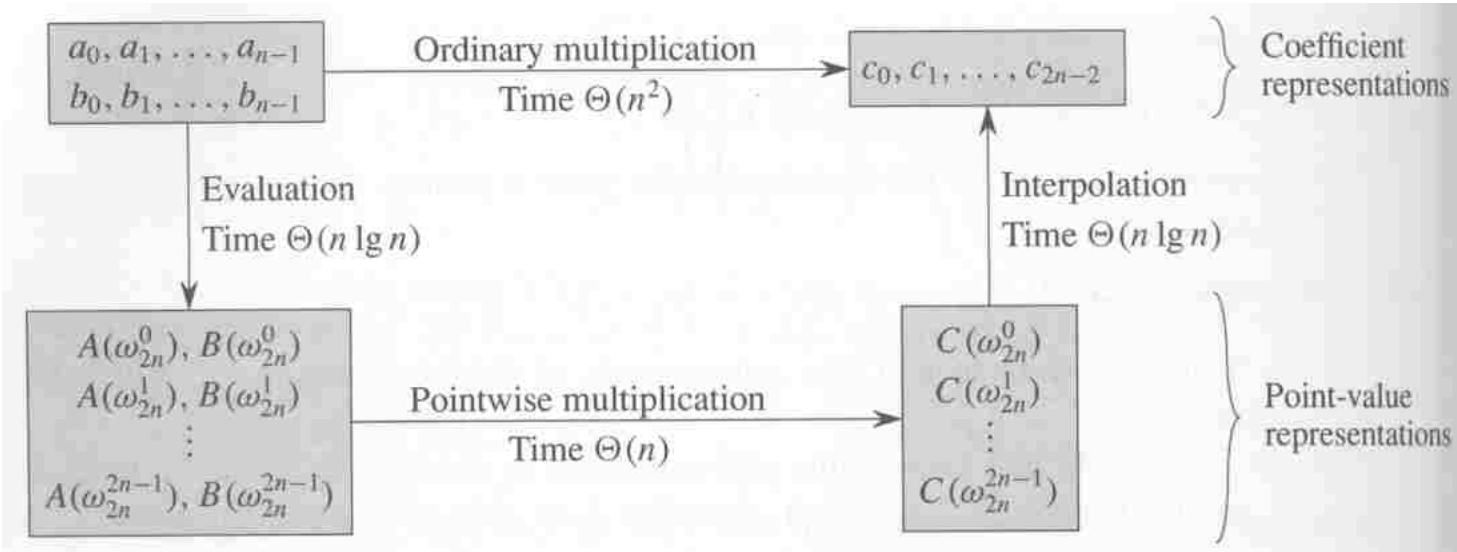

Figure 1: A graphical outline of our approach

Let $i = \sqrt{-1}$. Then $e^{iu} = \cos(u) + i \sin(u)$.

We compute the polynomials $A(x)$ and $B(x)$ for (complex) numbers ω_{2n}^k, where $\omega_{2n} = e^{2\pi i/2n}$.

The numbers $\{\omega_{2n}^k \mid k = 0, \ldots, 2n-1\}$ satisfy the equality $\omega^{2n} = 1$ and form a multiplicative group:

$\omega_{2n}^{2n} = \omega_{2n}^0 = 1 \Rightarrow \omega_{2n}^j \omega_{2n}^k = \omega_{2n}^{(j+k) \bmod 2n}$ and $\omega_{2n}^{-1} = \omega_{2n}^{2n-1}$.

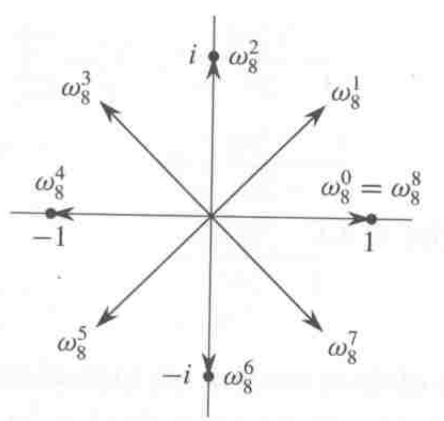

Figure 2: The numbers ω_8^k, $k = 0, \ldots, 7$, on the complex plain.

Lemma 2 *For all* $m \geq 0$, $k \geq 0$ *and* $d > 0$ *one has:* $\omega_{dm}^{dk} = \omega_m^k$.

Proof. $\omega_{dm}^{dk} = \left(e^{2\pi i/dm}\right)^{dk} = \left(e^{2\pi i/m}\right)^k = \omega_m^k$. $\square$

In particular: $\omega_{2n}^n = \omega_2 = -1$.

Lemma 3 *It holds:*

$$\{\omega^2 \mid \omega^{2n} = 1\} = \{\omega \mid \omega^n = 1\}.$$

Proof. Since $(\omega_{2n}^k)^2 = \omega_n^k$, Lemma 2 implies

$$(\omega_{2n}^{k+n})^2 = \omega_{2n}^{2k+2n} = \omega_{2n}^{2k}\omega_{2n}^{2n} = \omega_{2n}^{2k} = (\omega_{2n}^k)^2. \qquad \square$$

Let $A(x) = a_0 + a_1 x + a_2 x^2 + \cdots + a_{n-1}x^{n-1}$ and

$$y_k = A(\omega_n^k) = \sum_{j=0}^{n-1} a_j \cdot \omega_n^{kj}, \qquad k = 0, \ldots, n-1.$$

The vector $\vec{y} = (y_0, y_1, \ldots, y_{n-1})$ is called <u>Discrete Fourier Transform</u> (DFT) of vector $\vec{a} = (a_0, a_1, \ldots, a_{n-1})$.

We compute the vector $\vec{y}$ by using the Fast Fourier Transform (FFT). For this denote

$$\begin{aligned} A_0(x) &= a_0 + a_2 x + a_4 x^2 + \cdots + a_{n-2} x^{\lceil n/2 \rceil - 1} \\ A_1(x) &= a_1 + a_3 x + a_5 x^2 + \cdots + a_{n-1} x^{\lfloor n/2 \rfloor - 1}. \end{aligned}$$

Therefore:

$$A(x) = A_0(x^2) + x \cdot A_1(x^2).$$

W.l.o.g assume that n is a power of 2.

Algorithm 1 FFT($\vec{a}$);

1. m :=length(a) $\quad\quad$ $//m$ is a power of 2
2. **if** $m = 1$ **then** **return** a
3. $\omega_m := e^{2\pi i/m}$
4. $\omega := 1$
5. $a^0 := (a_0, a_2, \ldots, a_{m-2})$
6. $a^1 := (a_1, a_3, \ldots, a_{m-1})$
7. $\vec{y}^0 := FFT(\vec{a}^0)$
8. $\vec{y}^1 := FFT(\vec{a}^1)$
9. **for** $k = 0$ **to** $m/2 - 1$ **do**
10. $\quad y_k := y_k^0 + \omega \cdot y_k^1$
11. $\quad y_{k+m/2} := y_k^0 - \omega \cdot y_k^1$
12. $\quad \omega := \omega \cdot \omega_m$
13. **return** y

Line 2: basis of the recursion. $y_0 = a_0 \omega_1^0 = a_0$.

Lines 7-8: by Lemma 3 we have

$$\begin{aligned} y_k^0 &= A_0(\omega_{m/2}^k) = A_0(\omega_m^{2k}) \\ y_k^1 &= A_1(\omega_{m/2}^k) = A_1(\omega_m^{2k}) \end{aligned}$$

Line 10: for $k = 0, \ldots, m/2 - 1$ it holds:

$$\begin{aligned} y_k &= y_k^0 + \omega_m^k \cdot y_k^1 \\ &= A_0(\omega_m^{2k}) + \omega_m^k \cdot A_1(\omega_m^{2k}) \\ &= A(\omega_m^k). \end{aligned}$$

Line 11: for $k = 0, \ldots, m/2 - 1$ it holds

$$\begin{aligned} y_{k+m/2} &= y_k^0 - \omega_m^k \cdot y_k^1 \\ &= y_k^0 + \omega_m^{k+m/2} \cdot y_k^1 \qquad (\omega_m^{m/2} = \omega_{2m}^m = \omega_2 = -1) \\ &= A_0(\omega_{m/2}^k) + \omega_m^{k+m/2} \cdot A_1(\omega_{m/2}^k) \\ &= A_0(\omega_m^{2k}) + \omega_m^{k+m/2} \cdot A_1(\omega_m^{2k}) \\ &= A_0(\omega_m^{2k+m}) + \omega_m^{k+m/2} \cdot A_1(\omega_m^{2k+m}) \\ &= A(\omega_m^{k+m}). \end{aligned}$$

For the running time of the FFT-algorithm one has:

$$T(m) = 2 \cdot T(m/2) + \Theta(m) = \Theta(m \log_2 m).$$

The last problem is to convert the point-value representation of a polynomial into its coefficient representation. For this note that the numbers ω_n^k, $k = 0, \ldots, n-1$ satisfy the following equation:

$$\begin{pmatrix} 1 & 1 & 1 & 1 & \cdots & 1 \\ 1 & \omega_n & \omega_n^2 & \omega_n^3 & \cdots & \omega_n^{n-1} \\ 1 & \omega_n^2 & \omega_n^4 & \omega_n^6 & \cdots & \omega_n^{2(n-1)} \\ 1 & \omega_n^3 & \omega_n^6 & \omega_n^9 & \cdots & \omega_n^{3(n-1)} \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & \omega_n^{n-1} & \omega_n^{2(n-1)} & \omega_n^{3(n-1)} & \cdots & \omega_n^{(n-1)(n-1)} \end{pmatrix} \begin{pmatrix} a_0 \\ a_1 \\ a_2 \\ a_3 \\ \vdots \\ a_{n-1} \end{pmatrix} = \begin{pmatrix} y_0 \\ y_1 \\ y_2 \\ y_3 \\ \vdots \\ y_{n-1} \end{pmatrix}$$

$$= (y_0, y_1, y_2, y_3, \ldots, y_{n-1})^T = \vec{y}^T.$$

In other words, $\vec{y} = V_n \vec{a}$, so $\vec{a} = V_n^{-1} \vec{y}$.

Hence, we need to compute the matrix V_n^{-1}.

Theorem 4 *For* $V_n^{-1} = \{v_{jk}\}$ *one has:* $v_{jk} = \omega_n^{-kj}/n$.

Proof. We show that the matrix $W = \{v_{jk}\}$ satisfies $W \cdot V_n = I_n$. For this we compute the entry (j, j') of $W \cdot V_n$:

$$\begin{aligned} [W \cdot V_n]_{jj'} &= \sum_{k=0}^{n-1} (\omega_n^{-kj}/n)(\omega_n^{kj'}) \\ &= \sum_{k=0}^{n-1} \omega_n^{k(j'-j)}/n. \end{aligned}$$

For $j = j'$ we obviously have: $[W \cdot V_n]_{jj'} = 1$.

For $s = j' - j \neq 0$ we have:

$$\begin{aligned} n \cdot [W \cdot V_n]_{jj'} = \sum_{k=0}^{n-1} \omega_n^{ks} &= \frac{(\omega_n^s)^n - 1}{\omega_n^s - 1} \\ &= \frac{(\omega_n^n)^s - 1}{\omega_n^s - 1} \\ &= \frac{(1)^s - 1}{\omega_n^s - 1} = 0 \end{aligned}$$

since $-(n-1) < s < n-1$ and $s \neq 0 \Rightarrow \omega_n^s \neq 1$. $\square$

Therefore, for $\vec{a} = (a_0, \ldots, a_{n-1})$ one has:

$$a_j = \sum_{k=0}^{n-1} \left(\frac{1}{n} \cdot y_k \right) \cdot \omega_n^{-kj}, \qquad j = 0, \ldots, n-1.$$

In order to compute $\vec{a}$ we use the DFT and the FFT-algorithm (with $(1/n) \cdot \vec{y}$ instead of $\vec{a}$ and ω_n^{-1} instead of ω_n).

All this leads to the total running time $O(n \log n)$.

Therefore, for the polynomial

$$A(x) = \sum_{j=0}^{m-1} a_j \cdot x^j$$

we have the following formulas:

DFT: for $k = 0, \dots, m-1$

$$\begin{aligned}
y_k &= A(\omega_m^k) \\
&= \sum_{j=0}^{m-1} a_j \cdot \omega_m^{jk} \\
&= \sum_{j=0}^{m-1} a_j \cdot e^{\frac{2\pi i}{m} jk} \\
&= \sum_{j=0}^{m-1} a_j \cdot \left(\cos\left(\frac{2\pi}{m} jk\right) + i \cdot \sin\left(\frac{2\pi}{m} jk\right)\right) \\
&= \sum_{j=0}^{m-1} a_j \cdot \cos\left(\frac{2\pi}{m} jk\right) + i \cdot \sum_{j=0}^{m-1} a_j \cdot \sin\left(\frac{2\pi}{m} jk\right)
\end{aligned}$$

IDFT: for $j = 0, \dots, m-1$

$$\begin{aligned}
a_j &= \frac{1}{m} \sum_{k=0}^{m-1} y_k \cdot \omega_m^{-jk} \\
&= \frac{1}{m} \sum_{k=0}^{m-1} y_k \cdot e^{-\frac{2\pi i}{m} jk} \\
&= \frac{1}{m} \sum_{k=0}^{m-1} y_k \cdot \left(\cos\left(-\frac{2\pi}{m} jk\right) + i \cdot \sin\left(-\frac{2\pi}{m} jk\right)\right) \\
&= \frac{1}{m} \sum_{k=0}^{m-1} y_k \cdot \left(\cos\left(\frac{2\pi}{m} jk\right) - i \cdot \sin\left(\frac{2\pi}{m} jk\right)\right) \\
&= \frac{1}{m} \sum_{k=0}^{m-1} y_k \cdot \cos\left(\frac{2\pi}{m} jk\right) - i \cdot \frac{1}{m} \sum_{k=0}^{m-1} y_k \cdot \sin\left(\frac{2\pi}{m} jk\right)
\end{aligned}$$