Discrete Fourier Transform (lecture notes)

Alexander Tiskin

1 Definitions

Given a natural number $n \geq 1$, a complex number ω is called

- a *root of unity* of degree n, if $\omega^n = 1$
- in particular, a *primitive root of unity* of degree n, if $\omega, \omega^2, \ldots, \omega^{n-1} \neq 1$, and $\omega^n = 1$

All roots of unity of degree n are of the form[1] $\mathrm{e}^{2\pi \mathrm{i} k/n}$, where $k = 0, 1, \ldots, n-1$. A root of unity for a given $k > 0$ is primitive, if k is relatively prime with n. The *principal root of unity* of degree n is the primitive root $\mathrm{e}^{2\pi \mathrm{i}/n}$, corresponding to $k = 1$.

Let us fix a particular degree n and a primitive root of unity ω of degree n. The *Discrete Fourier Transform (DFT)* problem is defined as the matrix-vector product $F_{\omega,n} \cdot a = b$ over complex numbers, where the matrix is the special $n \times n$ *Fourier matrix* $F_{\omega,n} = \left[\omega^{ij}\right]_{i,j=0}^{n-1}$, the n-vector a is given as input, and the n-vector b is produced as output:

$$\begin{bmatrix} 1 & 1 & 1 & \cdots & 1 \\ 1 & \omega & \omega^2 & \cdots & \omega^{n-1} \\ 1 & \omega^2 & \omega^4 & \cdots & \omega^{n-2} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & \omega^{n-1} & \omega^{n-2} & \cdots & \omega \end{bmatrix} \cdot \begin{bmatrix} a_0 \\ a_1 \\ a_2 \\ \vdots \\ a_{n-1} \end{bmatrix} = \begin{bmatrix} b_0 \\ b_1 \\ b_2 \\ \vdots \\ b_{n-1} \end{bmatrix}$$

$$\sum_{j=0}^{n-1} \omega^{ij} a_j = b_i \qquad i, j = 0, \ldots, n-1$$

Direct computation of the DFT by the above definition requires $O(n^2)$ operations to evaluate the matrix-vector product.

The Fourier matrix $F_{\omega,n}$ is always nonsingular, therefore the output vector b uniquely determines the input vector a. The *inverse DFT* problem is given vector b as input, and asks to find the corresponding vector a. The inverse of the Fourier matrix is given by $(F_{\omega,n})^{-1} = 1/n \cdot F_{\omega^{-1},n}$ (this can be checked by direct multiplication). Therefore, the

[1] We use upright i for the imaginary unit, and italic i (alongside j, k, etc.) for an integer index.

inverse DFT corresponds to matrix-vector multiplication $1/n \cdot F_{\omega^{-1},n} \cdot b = a$, which is itself a DFT problem, up to a change of the primitive of unity from ω to ω^{-1} and scaling by a constant $1/n$. Therefore, any algorithm for DFT also solves the inverse DFT.

The DFT is a fundamental concept in many engineering applications. In particular, in digital signal processing it transforms a vector a of a signal's amplitude over time to a vector b of its frequency components. The DFT can also be used as an algorithmic tool for fast multiplication of polynomials and long integers.

2 Fast Fourier Transform, the "four-step" version

The *Fast Fourier Transform (FFT)* algorithm computes the DFT by divide-and-conquer, solving it on smaller subproblems, and then combining their solutions to a solution of the original problem.

The *four-step FFT* is the most symmetric version of FFT. It decomposes a DFT instance of degree n into $2n^{1/2}$ subproblems, each of which is a DFT instance of degree $n^{1/2}$. Assume that $n = 4^r$, and let $m = n^{1/2} = 2^r$. Let $A_{u,v} = a_{mu+v}$, $B_{s,t} = b_{ms+t}$, where $s, t, u, v = 0, \ldots, m-1$. Matrices A, B are n-vectors a, b, written out as $m \times m$ matrices:

$$A = \begin{bmatrix} a_0 & a_1 & \cdots & a_{m-1} \\ a_m & a_{m+1} & \cdots & a_{2m-1} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n-m} & a_{n-m+1} & \cdots & a_{n-1} \end{bmatrix} \qquad B = \begin{bmatrix} b_0 & b_1 & \cdots & b_{m-1} \\ b_m & b_{m+1} & \cdots & b_{2m-1} \\ \vdots & \vdots & \ddots & \vdots \\ b_{n-m} & b_{n-m+1} & \cdots & b_{n-1} \end{bmatrix}$$

We have

$$B_{s,t} = \sum_{u,v} \omega^{(ms+t)(mu+v)} A_{u,v} = \sum_{u,v} \omega^{msv+tv+mtu} A_{u,v} =$$
$$\sum_{v} \Big((\omega^m)^{sv} \cdot \omega^{tv} \cdot \sum_{u} (\omega^m)^{tu} A_{u,v} \Big)$$

where $s, t, u, v = 0, \ldots, m-1$.

Define the *twiddle-factor matrix* $T_{\omega,m} = \left[\omega^{tv}\right]_{t,v=0}^{m-1}$ (note that it forms the top-left corner $m \times m$ block in the $n \times n$ Fourier matrix $F_{\omega,m}$). We have obtained

$$B = F_{\omega^m,m} \cdot (T_{\omega,m} \circ (F_{\omega^m,m} \cdot A)^T)$$

where

- symbol '$\cdot$' denotes standard matrix product: $A \cdot B = C$ defined as $\sum_j A_{i,j} B_{j,k} = C_{i,k}$ for all i, k
- symbol '$\circ$' denotes Hadamard (elementwise) matrix product: $A \circ B = C$ defined as $A_{i,j} B_{i,j} = C_{i,j}$ for all i, j

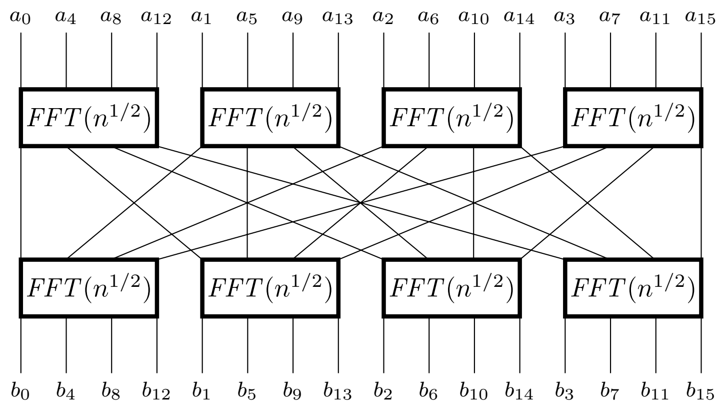

Figure 1: The four-step FFT for $n = 16$ (divide-and-conquer)

- symbol 'T' denotes matrix transposition: $A^T = B$ defined as $A_{i,j} = B_{j,i}$ for all i, j

Observe that the $m \times m$ matrix-matrix product $F_{\omega^m,m} \cdot A$ performs m independent DFTs with primitive root of unity ω^m of degree m on each column of matrix A. We thus compute the DFT of degree n by divide-and-conquer in four steps:

- m independent DFTs of degree m (multiplication by $F_{\omega^m,m}$)
- transposition and twiddle-factor scaling (Hadamard multiplication by $T_{\omega,m}$)
- m independent DFTs of degree m (multiplication by $F_{\omega^m,m}$)

We have reduced the DFT of size $n = 4^r$ to $2m$ DFTs of size $m = n^{1/2} = 2^r$, which are combined by $O(n)$ operations involved in matrix transposition and twiddle-factor scaling.

The base for the divide-and-conquer is provided by the DFT of degree 2:

$$F_{-1,2} \cdot a = \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix} \cdot \begin{bmatrix} a_0 \\ a_1 \end{bmatrix} = \begin{bmatrix} a_0 + a_1 \\ a_0 - a_1 \end{bmatrix}$$

The divide-and-conquer procedure will go through $\log_2 r = O(\log \log n)$ levels before reaching its base. We have the following recurrence for the overall running time:

$$T(n) = O(n) + 2 \cdot n^{1/2} \cdot T(n^{1/2}) =$$
$$O(1 \cdot n \cdot 1 + 2 \cdot n^{1/2} \cdot n^{1/2} + 4 \cdot n^{3/4} \cdot n^{1/4} + \cdots + \log n \cdot n \cdot 1) =$$
$$O(n + 2n + 4n + \cdots + \log n \cdot n) = O(n \log n)$$

The structure of the four-step FFT is shown in Figures 1, 2 (for $n = 16$). This structure is traditionally called a *butterfly*.

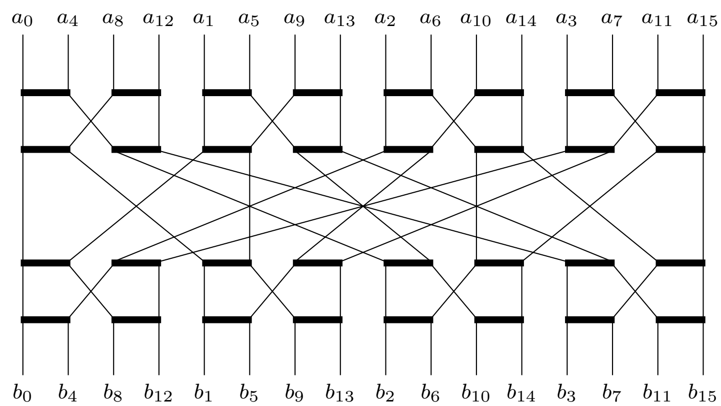

Figure 2: The four-step FFT for $n = 16$ (fully expanded)

3 Fast Fourier Transform, traditional version

A more traditional version of FFT exists in two “complementary” variants: *FFT with decimation in time (FFT-DIT)* and *FFT with decimation in frequency (FFT-DIF)*. Both decompose a DFT instance of degree n into two DFT subproblems of degree $n/2$ (solved by divide-and-conquer), and $n/2$ further DFT subproblems of degree 2 (solved directly). We describe FFT-DIT, and outline briefly the changes required to obtain FFT-DIF.

Both FFT-DIT and FFT-DIF only need to assume that $n = 2^r$. For FFT-DIT, let $A_{u,v} = a_{2u+v}$, $B_{s,t} = b_{ns/2+t}$, where $t, u = 0, \ldots, n/2 - 1$, $s, v = 0, 1$. Matrices A, B are n-vectors a, b, written out as an $n/2 \times 2$ and a $2 \times n/2$ matrix, respectively:

$$A = \begin{bmatrix} a_0 & a_1 \\ a_2 & a_3 \\ \vdots & \vdots \\ a_{n-2} & a_{n-1} \end{bmatrix} \qquad B = \begin{bmatrix} b_0 & b_1 & \ldots & b_{n/2-1} \\ b_{n/2} & b_{n/2+1} & \ldots & b_{n-1} \end{bmatrix}$$

(FFT-DIF does the opposite, writing A, B as a $2 \times n/2$ and an $n/2 \times 2$ matrix, respectively.) Similarly to the four-step FFT, we have

$$B_{s,t} = \sum_{u,v} \omega^{(ns/2+t)(2u+v)} A_{u,v} = \sum_{u,v} \omega^{nsv/2+tv+2tu} A_{u,v} =$$
$$\sum_{v} \big((-1)^{sv} \cdot \omega^{tv} \cdot \sum_{u} (\omega^2)^{tu} A_{u,v} \big)$$

where $t, u = 0, \ldots, n/2 - 1$, $s, v = 0, 1$.

Denote the "even half" ($v = 0$) and the "odd half" ($v = 1$) of vector a by

$$a_{even} = \begin{bmatrix} a_0 \\ a_2 \\ \vdots \\ a_{n-2} \end{bmatrix} \qquad a_{odd} = \begin{bmatrix} a_1 \\ a_3 \\ \vdots \\ a_{n-1} \end{bmatrix}$$

and the "lower half" ($s = 0$) and the "upper half" ($s = 1$) of vector b by

$$b_{lower} = \begin{bmatrix} b_0 & b_1 & \ldots & b_{n/2-1} \end{bmatrix} \qquad b_{upper} = \begin{bmatrix} b_{n/2} & b_{n/2+1} & \ldots & b_{n-1} \end{bmatrix}$$

We have

$$b_{lower} = (F_{w^2,n/2} a_{even})^T + \begin{bmatrix} 1 & \omega & \omega^2 & \ldots & \omega^{n/2} \end{bmatrix} \circ (F_{w^2,n/2} a_{odd})^T$$
$$b_{upper} = (F_{w^2,n/2} a_{even})^T - \begin{bmatrix} 1 & \omega & \omega^2 & \ldots & \omega^{n/2} \end{bmatrix} \circ (F_{w^2,n/2} a_{odd})^T$$

We have reduced the DFT of size $n = 2^r$ to two DFTs of size $n/2$, which are combined by $O(n)$ operations, including a computation of $n/2$ DFTs of size 2.

The base for the divide-and-conquer is provided by defining the DFT of degree 1 as the identity function $F_{1,1} a = a$.

The divide-and-conquer procedure will go through $\log_2 n$ levels before reaching its base. We have the following recurrence for the overall running time:

$$T(n) = O(n) + 2T(n/2) = O(1 \cdot n + 2 \cdot n/2 + 4 \cdot n/4 + \ldots + n \cdot 1) =$$
$$O(n + n + \ldots + n) = O(n \log n)$$

4 Polynomial multiplication by Fast Fourier Transform

We consider a polynomial of degree $n - 1$ to be given by a vector of its n coefficients. Consider the *polynomial multiplication problem*: given two polynomials of degree $n - 1$ over real or complex numbers

$$a(x) = \sum_{i=0}^{n-1} a_i x^i \qquad b(x) = \sum_{i=0}^{n-1} b_i x^i$$

obtain their product, which is a polynomial of degree $2n - 2$:

$$ab(x) = a(x) \cdot b(x) = \sum_{k=0}^{2n-2} \sum_{i=0}^{i} a_i b_{k-i} x^k$$

Direct computation of the product's coefficients by the above formula requires $O(n^2)$ operations.

An alternative method for polynomial multiplication involves evaluation and interpolation of polynomials for multiple arguments. Given a polynomial of degree $n-1$ represented by a column vector a of its n coefficients, and n argument values $x_0, x_1, \ldots, x_{n-1}$, the vector of polynomial's values at the given arguments corresponds to matrix-vector product

$$\begin{bmatrix} 1 & x_0 & x_0^2 & \cdots & x_0^{n-1} \\ 1 & x_1 & x_1^2 & \cdots & x_1^{n-1} \\ 1 & x_2 & x_2^2 & \cdots & x_2^{n-1} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & x_{n-1} & x_{n-1}^2 & \cdots & x_{n-1}^{n-1} \end{bmatrix} \cdot \begin{bmatrix} a_0 \\ a_1 \\ a_2 \\ \vdots \\ a_{n-1} \end{bmatrix} = \begin{bmatrix} a(x_0) \\ a(x_1) \\ a(x_2) \\ \vdots \\ a(x_{n-1}) \end{bmatrix}$$

$$\sum_{j=0}^{n-1} x_i^j a_j = a(x_i) \qquad i, j = 0, \ldots, n-1$$

If all arguments $x_0, x_1, \ldots, x_{n-1}$ are distinct, then the above matrix is nonsingular, and therefore the polynomial's $n-1$ coefficients are determined uniquely by its $n-1$ values. Therefore, the polynomial multiplication problem can be solved as follows:

- pick $N \geq 2n-1$ distinct complex numbers $x_0, x_1, \ldots, x_{N-1}$
- evaluate each of the polynomials a, b on x_i, obtaining $a(x_i)$, $b(x_i)$, for all $i = 0, 1, \ldots, N-1$
- obtain pairwise products $ab(x_i) = a(x_i) \cdot b(x_i)$, which determine uniquely the coefficients of the polynomial ab
- interpolate the coefficients of the polynomial ab from its values

For arbitrarily chosen argument values x_i, the described method does not give a computational advantage over direct computation of the product's coefficients. However, if we choose N to be the smallest power of 2 no less than $2n-1$, and the arguments to be $x_i = \omega^i = \mathrm{e}^{2\pi \mathrm{i} i/N}$, $i = 0, 1, \ldots, N-1$, then the multiple evaluation step corresponds to the DFT of degree N, and the interpolation step to the inverse DFT of degree N. Using the FFT algorithm, we can perform both these steps, and therefore obtain the solution to the polynomial multiplication problem, in $O(n \log n)$ operations.