

4/14/2009: Fast Fourier Transform

Advanced Algorithms (CSE 794)

Instructor: Tamal K. Dey

Scribe: Steven Martin *

April 30, 2009

1 Introduction

Fourier Transforms transform signals between a signal basis and a frequency basis. While this is often applied for the purposes of signal analysis, it can also be used to efficiently convolve signals, taking advantage of the observation that convolution in terms of the signal basis amounts to mere multiplication in the frequency basis. This lecture explains how to apply the discrete fourier transform to polynomial multiplication, and how to evaluate the discrete fourier transform in $O(N\log N)$ time using the Cooley-Tukey Fast Fourier Transform(FFT) algorithm.

2 Applying the Discrete Fourier Transform to Polynomial Multiplication

We evaluate a point-value pair representation of the polynomials with sufficient sampling, transform the result to frequency domain with the FFT, multiply those coefficients pointwise, then transform the result back to the original domain with the inverse FFT.

Given: p_0, p_1, \dots, p_n and q_0, q_1, \dots, q_n coefficients of the two polynomials p and q :

Step 1: Evaluate Compute the point-value pair representation of the two polynomials:

$$\begin{aligned} P &= \{(x, p(x_i)) | 0 \leq i \leq 2n - 1\} \\ Q &= \{(x, q(x_i)) | 0 \leq i \leq 2n - 1\} \end{aligned}$$

Then, using these $2n$ sample points, compute the DFT, resulting in two frequency domain vectors:

$$\begin{aligned} P_f(i) &= \text{DFT}(P_{0..2n-1})_i | 0 \leq i \leq 2n - 1 \\ Q_f(i) &= \text{DFT}(Q_{0..2n-1})_i | 0 \leq i \leq 2n - 1 \end{aligned}$$

Step 2: Pointwise Multiply Perform pointwise multiplication between P_f and Q_f to produce a resulting frequency domain sample vector Y_f .

$$Y_f(i) = P_f(i)Q_f(i) | 0 \leq i \leq 2n - 1$$

Because this operation is being done in the frequency domain, pointwise multiplication of the point-value polynomial samples can be done rather than convolution.

*Department of Computer and Information Sciences, The Ohio State University, Columbus, Ohio, USA.

Step 3: Interpolate Transform frequency domain samples from the pointwise multiplication back to the input domain:

$$Y(i) = \text{IDFT}(Y_{f_{0..2n-1}})_i | 0 \leq i \leq 2n - 1$$

The result Y is the point-value pair representation of the multiplication of the two polynomials, $y = p \times q$:

$$Y = \{(x, y(x_i)) | 0 \leq i \leq 2n - 1\}$$

For this process to be efficient, we need an efficient algorithm for the discrete fourier transform(DFT) and its inverse(IDFT)

3 Background

3.1 Complex Numbers

$$\begin{aligned} (a, b) &= a + ib \\ &= |A|(\cos\alpha + i\sin\alpha) \\ &= |A|\text{Exp}(i\alpha) \end{aligned}$$

where

$$\begin{aligned} A &= (a^2 + b^2)^{\frac{1}{2}} \\ \alpha &= \cos^{-1} \frac{a}{A} \\ \alpha &= \sin^{-1} \frac{b}{A} \end{aligned}$$

3.2 Roots of Unity

Definitions:

$\omega^n = 1$, ω is the n^{th} root of unity(1).

$\omega_n = \text{Exp}(i\frac{2\pi}{n}) = n^{\text{th}}$ principle root of unity

$\omega_n^0, \omega_n^1 = \omega_n, \omega_n^2, \dots, \omega_n^{n-1}$ are the roots of unity.

Properties:

(i) $\omega_{dn}^{dk} = \omega_n^k = e^{i\frac{2\pi dk}{dn}} = e^{i\frac{2\pi k}{n}} = \omega_n^k$

(ii) $\omega_n^{n/2} = \omega_{2n} = -1$ for n even

(iii) $(\omega_n^k)^2 = \omega_{n/2}^k$ for n even

(iv) $\sum_{j=0}^{n-1} (\omega_n^k)^j = 0$ for $n \geq 1$ and $k \geq 1$ not divisible by n

Proof.

$$\begin{aligned} \omega_n^k \sum_{j=0}^{n-1} (\omega_n^k)^j - \sum_{j=0}^{n-1} (\omega_n^k)^j &= \sum_{j=1}^n (\omega_n^k)^j - \sum_{j=0}^{n-1} (\omega_n^k)^j \\ &= (\omega_n^k)^n - (\omega_n^k)^0 \\ &= 0 \end{aligned}$$

So, $\sum_{j=0}^{n-1} (\omega_n^k)^j = \frac{0}{\omega_n^k - 1} \neq 0$ because n is not a multiple of k . □

4 Discrete Fourier Transform

The DFT changes a discrete signal from an input basis to a frequency basis.

Define: degree bound n of $p =$ degree of p is at least $n + 1$

Evaluate p at $x_j = \omega_n$ for $j = 0..n - 1$. Then, $p(x) = p_0 + p_1x + \dots + p_{n-1}x^{n-1}$.

Define: $(y_0, y_1, \dots, y_{n-1})$ so that $y_k = p(\omega_n^k) = \sum_{j=0}^{n-1} p_j \omega_n^{kj}$

The vector $(y_0, y_1, \dots, y_{n-1})$ is the DFT of the polynomial $p = (p_0, p_1, \dots, p_{n-1})$, thus $y = \text{DFT}(p)$. Naively implemented, this algorithm is a general convolution between two n element signals and requires $O(n^2)$ operations. However, a divide and conquer approach can be taken to achieve $O(n \log n)$ operations. Such approaches are called fast fourier transforms (FFTs)

5 Fast Fourier Transform

The general approach of a FFT is to factor the DFT to avoid redundant computation. Many variants exist on this, and different dimensionalities of input signals may yield different possibilities. In this lecture the most common 1D factorization was considered, the Cooley-Tukey algorithm.

We split the input signal into two, one comprised of the even coefficients, one comprised of the odd coefficients. The DFT (or FFT) is then performed on those signals and the results are combined with addition and scaling.

Assume: n is a power of 2

Let:

$$\begin{aligned} p(x) &= p^{[0]}(x^2) + xp^{[1]}(x^2) \\ p^{[0]}(x) &= p_0 + p_2x + p_4x^2 + \dots + p_{n-2}x^{(n-2)/2} \\ p^{[1]}(x) &= p_1 + p_3x + p_5x^2 + \dots + p_{n-1}x^{(n-2)/2} \\ y^{[0]} &= \text{DFT}(p^{[0]}) \\ y^{[1]} &= \text{DFT}(p^{[1]}) \end{aligned}$$

This means that...

$$\begin{aligned} y_k^{[0]} &= p^{[0]}(\omega_{n/2}^k) \text{ for } 0 \leq k \leq (n-2)/2 \\ y_k^{[1]} &= p^{[1]}(\omega_{n/2}^k) \text{ for } 0 \leq k \leq (n-2)/2 \end{aligned}$$

We can compute $y = \text{DFT}(p)$ as follows (using the above formulas). For $0 \leq k \leq (n-2)/2$:

$$\begin{aligned} y_k &= p(\omega_n^k) = p^{[0]}(\omega_n^{2k}) + \omega_n^k p^{[1]}(\omega_n^{2k}) \\ &= y_k^{[0]} + \omega_n^k y_k^{[1]} \end{aligned}$$

and

$$\begin{aligned} y_{n/2+k} &= p(\omega_n^{n/2+k}) \\ &= p^{[0]}(\omega_n^{n+2k}) + \omega_n^{n/2+k} p^{[1]}(\omega_n^{n+2k}) \\ &= p^{[0]}(\omega_n^{2k}) - \omega_n^k p^{[1]}(\omega_n^{2k}) \\ &= y_k^{[0]} - \omega_n^k y_k^{[1]} \end{aligned}$$

Resulting algorithm:

```

function FFT(p: polynomial): point-values
  n=n(p);
  if n=1 then return p;
  else p0=(p[0], p[2], ..., p[n-2]);
        p1=(p[1], p[3], ..., p[n-1]);
        y0=FFT(p[0]);
        y1=FFT(p[1]);
        w:=1;
        wn:=exp(i*2*PI/n);
        for k:=0 to (n/2)/2 do
          y[k]:=y0[k]+w*y0[k];
          y[n/2+k]:=y1[k]-w*y1[k];
          w:=w*wn;
        endfor
        return y;
  endif

```

Time analysis: $T(n) = 2T(n/2) + \theta(n)$ which is $O(n \log n)$

This gives us what we need to handle step 1 of our polynomial multiplication technique, but we now need to compute the inverse DFT for step 3.

6 Inverse DFT via the Inverse FFT

Proposition 1. Let V be a $n \times n$ vandermonde matrix where $V_{i,j} = \omega^{ij}$. Then, we define the DFT as $y = \text{DFT}(r) = Vr$. Using this, the IDFT can be defined as $r = \text{IDFT}(y) = V^{-1}y$. V^{-1} is $V_{i,j}^{-1} = \omega^{-ij}$.

Proof.

$$\begin{aligned} V_n^{-1}V_n &= \frac{1}{n} \left(\sum_{j=0}^{n-1} \omega_n^{-jk} * \omega_n^{jl} \right) \text{ where } k = 0, \dots, n-1, l = 0, \dots, n-1 \\ &= (1/n) \left(\sum_{j=0}^{n-1} \omega_n^{j(l-k)} \right) \text{ where } k = 0, \dots, n-1, l = 0, \dots, n-1 \end{aligned}$$

We then know via the properties of the roots of unity that $\omega_n^{j(l-k)} = 0$ if $l \neq k$, 1 otherwise.

Thus, $VV^{-1} = I$. □

Using this observation we need to make only two changes to the above FFT algorithm to apply it for computing the IDFT:

- Use a negative power in the exponential calculation
- Multiply the results by $\frac{1}{n}$

Rewriting the above algorithm with these changes in mind:

```

function IFFT_prenorm(p: polynomial): point-values
    n=n(p);
    if n=1 then return p;
    else p0=(p[0], p[2], ..., p[n-2]);
        p1=(p[1], p[3], ..., p[n-1]);
        y0=FFT(p[0]);
        y1=FFT(p[1]);
        w:=1;
        wn:=exp(-i*2*PI/n);
        for k:=0 to (n-2)/2 do
            y[k]:=y0[k]+w*y1[k];
            y[n/2+k]:=y1[k]-w*y0[k];
            w:=w*wn;
        endfor
        return y;
    endif

```

```

function IFFT(p: polynomial): point-values
    y=IFFT_prenorm(p);
    return y/n(y);

```

7 Other notes

Often times in practice the forward and inverse transform results are multiplied by $\frac{1}{n^{1/2}}$, resulting in unitary forward and inverse transforms; the only difference between the forward and inverse transforms in that case will be the sign of the exponential.

Additionally, in the case of having polynomials with real coefficients, the above transform can be further simplified. Well known packages for computing fourier transforms such as FFTW provide specialized algorithms for real and complex transformations.

8 Conclusion

Theorem 1. Let $p(x)$ and $q(x)$ be two polynomials with degree bound $n = 2^k$. Then $r = DFT_{2n}^{-1}(DFT_{2n}(p) * DFT_{2n}(q))$ is such that $r(x) = p(x) * q(x)$ and it can be computed in $O(n \log n)$ time.