Fourier Transforms and Theoretical Computer Science

February 5, 1999

Lecturer: Umesh Vazirani

Scribe: Steve Chien

Today's topics: The Discrete Fourier Transform=20

- DFT over $\mathbf{Z}_2^n$ and the mixing time of random walks.
- DFT over $\mathbf{Z}_n$ and applications to polynomial/integer = multiplication.

1 DFT over $\mathbf{Z}_2^n$

We consider the group $\mathbf{Z}_2^n$ and functions $f : \mathbf{Z}_2^n \rightarrow \mathbf{C}$. = These functions form a vector space of dimension 2^n, and a natural basis to take would be that comprised = of the indicator functions $\{f_x : f_x(y) = 3D1 \Leftrightarrow x = 3Dy\}$. Another basis is the parity basis, = which we can define as the set=20 $\{\chi_x : \chi_x(y) = 3D(-1)^{x \cdot y}\}$, where $x \cdot y$ is the inner = product of x and y. The effect of applying χ_x is to determine the parity of those bits of y = selected by the 1 bits in x.

If we define $\chi_x \cdot \chi_y$ as $\sum_z \chi_x(z)\chi_y(z)$, then we = see that the χ_x are orthogonal. It is=20 obvious that $\forall x \chi_x \cdot \chi_x \neq 0$. If $x \neq z$, then = $\chi_x \cdot \chi_y = 3D\sum_y = 20(-1)^{x \cdot y}(-1)^{z \cdot y} = 3D\sum_y (-1)^{(x+z) \cdot y}$. = Since $x + z$ must be nonzero in at least one of its=20 bits, we see that as y varies, we will achieve exact cancellation and = end up with 0. Thus, by a counting argument, the 2^n parity functions form a basis. =20

Furthermore, since these functions $\{\chi_x\}$ are homomorphisms, they = are in fact the Fourier basis. (Another way to say this is that they are the *characters* of $\mathbf{Z}_2^n$, a term that = comes form group representation theory.) The Fourier transform then takes us from the basis of the indicator = functions to the parity basis: $\hat{f}(y) = 3D\sum_z (-1)^{y \cdot z} f(z)$.

There are several nice properties that come from these definitions.

Convolution

The first is that convolution in the indicator basis is equivalent to = multiplication in the Fourier basis. Recall that the convolution of two functions f and g is defined as $f* = g(x) = 3D\sum_y f(y)g(x-y)$ (over an arbitrary group, we replace the $g(x-y)$ with $g(y^{-1}x)$). = We can quickly verify that $f \hat{*} g(x) = 3D\hat{f}(x)\hat{g}(x)$:

$$\begin{aligned} f \hat{*} g(x) &= 3D \quad \sum_z \sum_y f(y)g(z-y)(-1)^{x \cdot z} \\ &= 3D \quad \sum_y f(y)(-1)^{x \cdot y} \sum_z g(z-y)(-1)^{x \cdot (z-y)} = \\ &= 3D \quad \hat{f}(x)\hat{g}(x) \end{aligned}$$

Plancherel's Theorem

$$\sum_x f(x)g(x) = 3D = \frac{1}{2^n} \sum_x \hat{f}(x)\hat{g}(x)$$

Subspaces of $\mathbf{Z}_2^n$

Consider $\mathbf{Z}_2^n$ as a vector space, and a subspace $X \subseteq \mathbf{Z}_2^n$. = Then define $X^\perp$ (pronounced "x-perp") as $\{y \in \mathbf{Z}_2^n \ : x \cdot y = 3D0 \forall x \in X\}$. Consider now = the indicator function=20 $f(x) = 3D \begin{cases} 1 & \text{if } x \in X \\ 0 & \text{otherwise} \end{cases}$ We claim that the Fourier transform of f behaves well:=20 $\hat{f}(y) = 3D \begin{cases} 1 & \text{if } y \in X^\perp \\ 0 & \text{otherwise} \end{cases}$.

Proof. Note that $\hat{f}(y) = 3D \sum_{x \in = X} (-1)^{x \cdot y}$. If $y \in X^\perp$, then $\forall x x \cdot y = 3D0$ and so $\hat{f}(y) = 3D|X|$. If $y \notin = X^\perp$ then $\exists z \in X$ such that $z \cdot y = 3D1$. We can then divide X into two equal sets, = $X' = 3D\{x : x \cdot y = 3D0\}$ and $X' + Z$. From this we can conclude that $\hat{f}(y) = 3D0$ as needed.

Mixing Time of Random Walks over $\mathbf{Z}_2^d$

[Insert reference.]

Consider a random walk over $\mathbf{Z}_2^d$ in which a particle begins at the = origin and at each time step will either remain at its current position or move to one of its d neighbors, all with = equal probability $\frac{1}{d+1}$. We wish to know how quickly the particle's position as a probability distribution = approaches uniform.

More formally, the probability distribution of the particle's position = after one time step can be described as=20

$$Q(x) = 3D \begin{cases} \frac{1}{d+1} & \text{if } x = 3D0 \text{ or } x = 3De_i \\ 0 & \text{otherwise} \end{cases}$$

Phrasing it this way allows to see the happy fact that after two steps, = the probability distribution is simply the convolution of $Q(x)$ with itself, which by definition is $Q * Q(x) == 3D \sum_y Q(y)Q(x-y)$. One way to read this is that the probability of the particle being at x after two steps is = exactly the sum over all y of the=20 probability of both being at y after one step, $Q(y)$ and moving from = y to x in the second step, $Q(x-y)$. ¿From this we conclude that after n steps, the probability distribution = is the convolution power of Q, denoted by Q^{*n}.

Using the above fact about convolutions and the Fourier transform, we = can see that $\widehat{Q^{*n}} = 3D\hat{Q}^n$. We wish to determine how close to uniform this is. Notice that if $U(x) == 3D\frac{1}{2^d}$ is the uniform distribution, then the Fourier transform of it is=20 $\hat{U}(x) = 3D \begin{cases} 1 & \text{if } x = 3D0 \\ 0 & \text{otherwise} \end{cases}$.

We now prove the following theorem:

Theorem. $||Q^{*k} - U||_1 \leq \sqrt{2(e^{e^{-c}} - 1)}$ = for $k = 3D\frac{(d+1)\ln d}{4} + cd$.

In fact this is a very tight bound; a lower bound of $\frac{d \ln d}{4} + = cd$ can be proven.

To prove this theorem we first need the following lemma:

Lemma. For any function $P(x)$ such that $\hat{P}(0) = 3D1$ then = $||P^{*k} - U||_1^2 \leq \sum_{x \neq 0} |\hat{P}(x)|^{2k}$.

Proof. The proof relies on the Cauchy-Schwarz inequality, whose = results are often disastrous but fortunately not so in this case, as well as Plancherel's result.

$$||P^{*k} - U||_1^2 \quad = 3D \quad \sum_y (|P^{*k}(y) - U(y)|)^2 =$$

$$\leq \quad 2^d \sum_y |P^{*k}(y) - U(y)|^2 \text{ by } = \text{Cauchy-Schwarz}$$

Now, since $\hat{U}(0) = 3D\hat{P}(0) = 3D1$ but $\hat{U}(x) = 3D0 =$ otherwise, using Plancherel's theorem we can write

$$\begin{aligned} \|P^{*k} - U\|_1^2 &\leq \sum_{y\neq 0} (\hat{P^{*=k}}(y))^2 \\ &\leq \sum_{y\neq 0} [\hat{P}(y)]^{2k} \end{aligned}$$

which is exactly what we need.

Proof of Theorem.=20 What we need to do now is to calculate the actual Fourier coefficients = and apply our lemma. Fortunately, our task is made much easier by the structure of $Q(x)$, namely the = properties that $Q(x)$ is nonzero for only $d+1$ values of x, and that these values of x have simple structure.

$$\begin{aligned} \hat{Q}(x) &= 3D \quad \sum_y (-1)^{x\cdot y} Q(y) \\ &= 3D \quad \frac{1}{d+1}[1 + \#0\text{'s in } x - \#1\text{'s in } = x] \\ &= 3D \quad 1 - \frac{2|x|}{d+1} \text{where } |x| = 3D\#1\text{'s in } = x \end{aligned}$$

The second equality comes about since $Q(y)$ is nonzero only for $y = 3D = 0$ or $y = 3De_i$. When $y = 3D0$,=20 $(-1)^{x\cdot y}$ is 1, and when $y = 3De_i$, then $(-1)^{x\cdot =y}$ is 1 when the ith bit of x is 0, and -1 otherwise.

Applying the lemma, we obtain

$$\begin{aligned} \|Q^{*k} - U\|_1^2 &= 3D \quad \sum_{x\neq 0} (1 - = \frac{2|x|}{d+1})^{2k} \\ &= 3D \quad \sum_{j=3D1}^{d} \binom{d}{j\ =} = 20(1 - \frac{2j}{d+1})^{2k} \\ &\leq \quad \sum_{j=3D1}^{d} \frac{d^j}{j!} = e^{\frac{(-2j)(2k)}{d+1}} \end{aligned}$$

Considering only the e term and substituting in for k, we see that=20

$$e^{\frac{(-2j)(2k)}{d+1}} = 3De^{-\frac{4j}{d+1}[\frac{(d+1)\ln d}{4} += cd]} = 3De^{-j\ln d} e^{\frac{-4jcd}{d+1}} \leq d^{-j}e^{-cj}$$

Substituting this back in, we find that

$$\|Q^{*k} - U\|_1^2 \quad \leq \quad \sum_{j=3D1}^{d} \frac{d^j}{j!} = d^{-j}e^{-cj}$$

$$
\begin{aligned}
&\leq \sum_{j=3D1}^{d} \frac{e^{-cj}}{j!} \\
&\leq \sum_{j=3D1}^{\infty} \frac{e^{-cj}}{j!} \\
&\leq e^{e^{-c}} - 1
\end{aligned}
$$

This finishes the proof.

2 DFT over $\mathbf{Z}_n$

We now consider the Discrete Fourier Transform over the group $\mathbf{Z}_n$. = Following the same principle as before, we consider the functions $f : \mathbf{Z}_n \rightarrow \mathbf{C}$. This time the = homomorphisms, and thus the proper Fourier=20 basis functions, are $\{\chi^k(j) = 3D\omega^{jk}\}$, where $\omega == 3De^{2\pi i/n}$ is a primitive nth root of unity. (Once again, these are the characters of $\mathbf{Z}_n$.) Consequently, = we can once again define the Fourier transform of f as $\hat{f}(x) = 3D\sum_y \omega^{xy} f(y)$.

Notice that the Fourier transform in this case has an elegant matrix = representation:

$$
\begin{bmatrix} \hat{f}(1) \\ \hat{f}(\omega) \\ = \hat{f}(\omega^2) \\ \vdots \\ \hat{f}(\omega^n) \end{bmatrix} = 3D \underbrace{\begin{bmatrix} 1 & 1 & 1 & \cdots & 1 \\ = 201 & \omega & \omega^2 & \cdots & \omega^{n-1} \\ 1 & \omega^2 & \omega^4 & \cdots & \omega^{2(n-1)} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & \omega^{n-1} & \omega^{2(n-1)} & \cdots & \omega^{(n-1)^2} \end{bmatrix}}_{F} \begin{bmatrix} f(1) \\ f(\omega) \\ f(\omega^2) \\ \vdots \\ = f(\omega^{n-1}) \end{bmatrix}
$$

In other words, the transformation matrix F can be expressed as $= \left[\{\omega^{jk}\}\right]$. Also important is that F is invertible and that F^{-1} has a nice = form, namely that=20 $F^{-1} = 3D\frac{1}{n}F^t = 3D\frac{1}{n}\left[\{\omega^{-jk}\} =\right]$. We also have the usual property concerning convolution: $\widehat{f * g}(x) = 3D\hat{f}(x)\hat{g}(x)$. = (These last two properties are fairly easy exercises.)

FFT

Calculating the $\hat{f}$ vector using straightforward matrix = multiplication would require time $O(n^2)$, but the well-known FFT algorithm requires only time $O(n \log n)$. FFT = relies on the fact that a clever rearrangement of the columns of the F matrix significantly reduces the number of = needed operations. More precisely, we move the columns representing the even powers of ω to the left, = and those representing the odd powers to the right, and rearrange the f vector to place the even values on top = and the odd values on bottom. This gives=20 us the following:

$$
\begin{bmatrix} \hat{f}(1) \\ \hat{f}(\omega) \\ = \hat{f}(\omega^2) \\ \vdots \\ \hat{f}(\omega^n) \end{bmatrix} = 3D = 20 \begin{bmatrix} \{\omega^{2jk}\} = 3DA & \{\omega^j\omega^{2jk}\} = 3DB = \\ = 20\{\omega^{2(j+n/2)k}\} = 3DA & \{\omega^{j+n/2}\omega^{2(j+n/2)k}\} == 3D-B \end{bmatrix} =
$$

$$20\begin{bmatrix} \{f(\omega^{2j})\} = 3Df_0 = \\ \\ \{f(\omega^{2j\prime+1})\} = 3Df_1 \end{bmatrix} = 3D = 20 \begin{bmatrix} Af_0 + Bf_1 \\ \\ = Af_0 - Bf_1 \end{bmatrix}$$

The operations now required include the recursive calls of computing = each of Af_0 and Bf_1 and then=20 performing the additions. The total running time now becomes $T(n) = 3D = 2T(n/2) + O(n) = 3DO(n \log n)$.

Application: Polynomial multiplication

The FFT algorithm leads to an immediate application in polynomial = multiplication. Given two polynomials of degree $n-1$, $P(x) = 3Da_0 + a_1x + a_2x^2 + \cdots + = a_{n-1}x^{n-1}$ and $Q(x) = 3Db_0 + b_1x + = 20b_2x^2 + \cdots + b_{n-1}x^{n-1}$, we notice that the coefficients = of their product $P(x)Q(x)$ is the convolution of their original coefficients. Thus to compute the = product, all we need do is take the FFT of the original coefficients, multiply them, and take the inverse FFT, all = of which can be done in time $O(n \log n)$.

More explicitly, if we write $P(x)Q(x) = 3Dc_0 + c_1x + \cdots + = c_{2n-2}x^{2n-2}$, then our algorithm will first compute $\mathrm{FFT}(a_0, \ldots, a_{n-1}, 0, \ldots, 0)$ and $= \mathrm{FFT}(b_0, \ldots, b_{n-1}, 0, \ldots, 0)$ to obtain $(\hat{a}_0, \ldots, \hat{a}_{2n-2})$ and $= (\hat{b}_0, \ldots, \hat{b}_{2n-2})$. We then multiply these componentwise to obtain $(\hat{c}_0, \ldots, \hat{c}_{2n-2})$ and then = do the inverse FFT to obtain the $\{c_i\}$.

Application: Integer multiplication

A more interesting application of FFT is the integer multiplication = algorithm of Schönhage and Strassen, which allows us to multiply two n-bit integers in time $O(n \log n \log\log = n)$ rather than the naive $O(n^2)$. The=20 idea is to treat n-bit integers as polynomials and use the $O(n \log = n)$ polynomial multiplication algorithm. If we are still allowed to assume that primitive operations still take $= O(1)$ time, then we might imagine that given two n-bit integers A and B we might divide each of them into the = concatenation of $n/\log n$ pieces. For example, we would write $A = 3Da_{n-1}a_{n-2} \cdots a_1a_0$. We can = then write the polynomial $A(x) = 3Da_{n-1}x^{n-1} + \cdots + a_1x + a_0$ and realize that $A(n) = 3DA$. After = following a similar procedure for B, we would then apply the FFT polynomial multiplication algorithm to recover the $= c_i$. We can then evaluate the polynomial at n in time $O(n \log n)$, as the c_ix^i terms have small overlap.

The problem, of course, is that the primitive operations in the FFT take = longer than $O(1)$. Furthermore, this approach requires precise arithmetic computations. The solution to = these problems is to work in the ring $\mathbf{Z}_m$ and to divide the n-bit numbers in a different manner.

Fortunately for our purposes, it is not difficult to choose appropriate = values for m and n so that performing FFT is possible. If we let n and ω be some powers of 2 and = let $m = 3D\omega^{n/2} + 1$, then we have the following properties, which turn out to be sufficient for FFT: =20

- $\omega^n = 3D1$
- $\omega^0, \omega^1, \ldots, \omega^{n-1}$ are distinct
- $\forall 1 \leq k \leq n \sum_{j=3D0}^{n-1} \omega^{jk} = 3D0$
- n^{-1} exists.

Notice that the time to perform FFT has increased, however. Since our = addition operations are potentially=20 over n-bit numbers, we have an extra factor of n, so that the cost = of FFT is now $O(n^2 \log n)$. =20 (Note that as the only multiplications in FFT involve powers of $= \omega$, multiplications consist only of shifts and additions.)

The algorithm is now straightforward. We now divide each of our = n-numbers into $\sqrt{n}$ pieces each of size $\sqrt{n}$. As before, we do an FFT on these $\sqrt{n}$ values, perform = a componentwise multiplication, and then do the inverse FFT. Each FFT now takes time $O(n \log \sqrt{n}) = 3D = O(n \log n)$. If $T(n)$ is the time required=20 to multiply two n-bit integers, we can write the recurrence relation = $T(n) = 3D\sqrt{n}T(\sqrt{n}) + O(n \log n)$, which works out to $O(n \log n \log \log n)$.