MATH 6122: PROJECT REPORT: THE FOURIER TRANSFORM ON FINITE GROUPS: THEORY, COMPUTATION, AND AN APPLICATION

YIAN YAO AND WEIWEI ZHANG

Abstract. We report on the Fourier analysis on finite groups. In the theoretical part, we start by a brief introduction of the Fourier analysis on the real domain, followed by the discussion on finite abelian groups. After this, we delve into the Fourier analysis on all finite groups with the aid of the representation theory. In the computational part, we report on two fast Fourier transform(FFT) algorithms developed by Cooley-Tukey and Diaconis-Rockmore, and specifically focus on the S_n groups. Some theoretical results $\hat{f}_e(y_j)$s are provided to speed up the algorithm. Finally, in the applicational part of the report, we explore the use of Fourier transforms in multi-object tracking problems in machine learning, where an efficient algorithm for approximately maintaining and updating a distribution over permutations is provided with the aid of Clausen's FFT.

keywords: Fourier Transform on Finite Groups, Representation Theory, Cooley-Tukey fast Fourier Transform, Multi-object Tracking

1. Introduction

The Fourier Transform(FT) is an extremely powerful mathematical tool with large applications in many areas. The basic idea of FT is decomposing a function into its constituent frequencies, such functions can be signals, audios or other functions of time, and represent the function by the linear combination of sine and cosine waves. This idea of moving function into the frequency domain allows people to filter out certain frequencies for dimension reduction or noise canceling.

The principle of Fourier analysis is not only applicable when the time domain for the function is real(i.e. $\mathbb{R}$), but also when the domain is a finite group and the basic idea is pretty similar. For the finite group case, we will depend on the representation theory to build a set of suitable basis in the frequency domain.

The Fourier transform on finite groups has applications in object tracking problems in machine learning, since it allows a faster method to store and update information. Multi-object trackers can be viewed as a permutation between different tracks and real-world objects. A classical example is the air traffic control, where the tracks for the aircraft are often visible on radar, the identity of the aircraft is rarely revealed unless the pilot report by video. The typical way of such data association problem is using a probability distribution to represent the possibility of the corresponding objects' identities. However, maintaining such distribution can be very costly when the number of objects is large.

This report is organized as follows: In section 2, we discuss the theory of Fourier Transform when the domain is $\mathbb{R}$, finite abelian groups, and finite groups in general. In section 3, we study the Cooley-Tukey fast Fourier Transform and Diaconis-Rockmore fast Fourier

The code in section 3 is available in my github page.

Transform. Moreover, a concrete algorithm is provided when the domain is S_n. In section 4, we explore a target tracking algorithm with the aid of the Fourier Transform.

This report is mainly based on [Dan18](section 2, 3), [KHJ07](section 4).

2. The Theory of Fourier Transform

In this section, we explore the theoretical derivation of the Fourier Transform. We start by a brief introduction of the Fourier Transform in $L^2(\mathbb{X})$, where $\mathbb{X}$ is a measurable space, and then generate the definition into finite abelian groups by the group characters. After that, we recap some facts in the representation theory and use them to arrive at a proper definition for the Fourier Transform for all finite groups. In order to avoid redundancy, we will omit some of the definitions and lemmas. One can refer to the original paper [Dan18] for more details.

2.1. Fourier Theory on Measurable Space.

In this subsection, we recall briefly the Fourier Transform in a measurable space $\mathbb{X}$.

Recall $L^2(\mathbb{X}) \triangleq \{f \mid (\int_{\mathbb{X}} |f|^2)^{\frac{1}{2}} < \infty\}$ is a Hilbert space if one unifies functions that are equivalent almost everywhere, with the inner product defined as $\langle f, g\rangle = \int_{\mathbb{X}} f\overline{g}$. This allows us to define the orthogonality between functions, and moreover, finding a proper set of orthonormal basis.

In fact, when $\mathbb{X} = [0, 1]$, the orthonormal topological basis of $L^2([0, 1])$ are $e_k(x) = e^{-2\pi ikx}$. We can then write f by the series representation $f = \sum_{k=-\infty}^{\infty} \langle f, e_k\rangle e_k$, with the image of f under the new basis $\{e_k\}_{k=-\infty}^{\infty}$ is

$$\hat{f}(k) \triangleq \langle f, e_k\rangle = \int_0^1 f(x)e^{-2\pi ikx}dx, \tag{1}$$

which gives us the definition of the Fourier Transform.

By adapting (1), for any funciton $f \in L^2(\mathbb{R})$, we have:

$$\hat{f}(y) = \int_{-\infty}^{\infty} f(x)e^{-2\pi iyx}dx. \tag{2}$$

2.2. Fourier Theory on Finite Abelian Groups.

Starting from this section, we extend the Fourier theory to any function $f : G \to \mathbb{C}$, where G is some finite group. In this subsection, we restrict G to be a finite Abelian Group.

Similarly to 2.1, we define $L^2(G)$ with the inner product $\langle f_a, f_b\rangle = \sum_{g\in G} f_a(g)\overline{f_b(g)}$ where $f_a, f_b \in L^2(G)$. Unlike in 2.1, the finiteness of G guarantees the integralbility and ease us from the concern of equivalence classes. In other words, $L^2(G)$ is a Hilbert space.

In order to write f in terms of Fourier series, we recall:

Definition. A group character is a homomorphism from a group G to the multiplicative group of S_1. More spercifically, $\chi : G \to S_1$ is a group character if χ satisfies $\chi(gh) = \chi(g)\chi(h),\ \forall g, h \in G$. The dual space of G is the set of characters of G, and is denoted by $\hat{G}$.

By this definition, it is easy to show that

Lemma 1. *If χ is a character of G, then $\chi(g)$ is a $|G|$-th root of unity for all $g \in G$.*

and

Lemma 2. *The set of group characters of G is orthogonal in $L^2(G)$.*

Proof. Suppose $\chi_a \neq \chi_b$, then for some $h \in G$, we have

$$\begin{aligned}\chi_a(h)\langle\chi_a, \chi_b\rangle &= \sum_{g\in G} \chi_a(h)\chi_a(g)\overline{\chi_b(g)} = \sum_{g\in G} \chi_a(gh)\overline{\chi_b(g)} = \sum_{g\in G} \chi_a(g)\overline{\chi_b(h^{-1}g)} \\ &= \sum_{g\in G} \chi_a(g)\chi_b(h)\overline{\chi_b(g)} = \chi_b(h)\langle\chi_a, \chi_b\rangle.\end{aligned}$$

□

We can then show

Theorem 3. $G \cong \hat{G}$.

This proof is rather long and we will refer the details to [Dan18]. The general idea is first show $|G| = \left|\hat{G}\right|$. To show this equality, one can start by assuming G is cyclic, and then generate it to finite abelian group by $G = \mathbb{Z}/n_1\mathbb{Z} \times \mathbb{Z}/n_2\mathbb{Z} \times \cdots \times \mathbb{Z}/n_k\mathbb{Z}$ and consider $\chi_g \triangleq \chi_{g_1}(x_1)\cdots\chi_{g_k}(x_k)$ where each χ_{g_i} is defined in the cyclic case, and $x = (x_1, \cdots, x_k) \in G$. After showing the equality of the order, we have $G \cong \hat{G}$ by noticing the map $g \to \chi_g$ is a isomorphism.

By Lemma 1, 2 and 3, we can arrive the following theorem:

Theorem 4. *The set of characters of a finite abelian group G is a basis for $L^2(G)$.*

By observing the delta function forms a basis of $L^2(G)$ with the same cardinality $|G|$.

Above all, we can rewrite $f \in L^2(G)$ under the new basis $\{\chi_g\}_{g\in G}$ as follows:

$$f = \sum_{g\in G} \langle f, \chi_g\rangle\chi_g, \text{and}$$

which gives us the Fourier transform of f:

$$\hat{f}(\chi) \triangleq \langle f, \chi\rangle = \sum_{g\in G} f(g)\overline{\chi(g)}. \tag{3}$$

2.3. **Fourier Theory on Non-Abelian Groups.** Because there is no clear way of writing functions under the basis of $L^2(G)$ where G is any finite group. We now delve into the representation theory to look for the new basis.

2.3.1. *Some facts in representation theory.* For simplicity, we skip most of the proofs in this subsubsection as we have seen most of the results in class.

2.3.2. *In search of an new basis.* The new basis is formed by observing that the matrix entries corresponding to a representation are orthogonal. To show this, we first need the following lemma:

Lemma 5. *Let (ρ, V) be an irreducible representation, and let $L : V \to V$ intertwines ρ with itself. Then $L = xI$ where x is a scalar, and I is identity linear transformation.*

Proof. In this case, L is an invertible square matrix. Let x be an eigenvalue, and W be the corresponding eigenspace. For $w \in W$, we have

$$L(\rho(g)(w)) = \rho(g)(L(w)) = \rho(g)(xw) = x\rho(g)(w).$$

which implies that $\rho(g)(w) \in W$. Since W induces a subrepresentation of ρ, and ρ is irreducible, we ahve W is either $\{0\}$ or V, thus $L = xI$. □

Similar to 2.1, 2.2, we define $L^2(G)$ with the inner product $\langle \rho_{n,m}, \pi_{i,j} \rangle = \sum_{g \in G} \rho_{n,m} \overline{\pi_{i,j}}$, and denote the dual space of G, the set of all irreducible representations, as $\hat{G}$. With the help of this lemma, we can show that

Theorem 6. *Let $(\rho, \mathbb{C}^a)$ be an irreducible representation of G. Then $\langle \rho_{n,m}, \rho_{i,j} \rangle \neq 0$ if and only if $n = i$ and $m = j$.*

Proof. Define $N = \sum_{g \in G} \rho(g) M \rho(g^{-1})$, where M is any linear mapping $\mathbb{C}^a \to \mathbb{C}^a$. Then one can show that N intertwines ρ with itself [Dan18]. By the aboeve lemma, we have $N = xI$ for some $x \in \mathbb{C}$. Since the trace of a matrix preserves under conjugation, we have $|G|\, tr(M) = tr(xI) = xa$.

In particular, let $M(m, n) = 1$ for $m = i$ or $n = j$, and otherwise 0, we have $\sum_{g \in G} \rho_{n,m}(g) \overline{\rho_{i,j}(g)} = \langle \rho_{n,m}, \rho_{i,j} \rangle$.

Thus, $x \neq 0 \iff tr(M) \neq 0 \iff m = j$, we have $\langle \rho_{n,m}, \rho_{i,j} \rangle \neq 0 \iff m = j$ and $n = i$. □

To demostrate that the matrix entries of an irreducible representation indeed forms an orthogonal basis, we need the following theorem:

Theorem 7. *Let G be a finite group, then $\sum_{\rho \in \hat{G}} d_\rho^2 = |G|$.*

whose proof is rather complicated, requires discussions related to left regular representations, and we refer to [Ter99] for those who are interested.

By the above discussion, we can define the Fourier transform for all finite groups as

$$\hat{f}(\rho) = \sum_{g \in G} f(g) \rho(g). \tag{4}$$

Remark. The definition of the Fourier Transform in 3 is actually a special case of 4, since the abelian group characters are one-dimensional representations. Thus the two definitions are consistant.

3. The Algorithm for Computing Fourier Transforms

In this section, we will discuss the numerical algorithms of the Fourier transform. Define a function $f \in L^2(\mathbb{X})$, where $\mathbb{X}$ can be either $\mathbb{R}$ or any finite group, by sampling f at n points $x_0, x_1, \cdots, x_{n-1}$, we can compute

$$\hat{f}(y_j) = \sum_{i=0}^{n-1} f(x_i) e^{\frac{-2\pi i x_i y_i}{n}}. \tag{5}$$

The pseudo-code for this naive implementation is as follows:

function SLOWFT(f) ▷ f is an array storing sampled values of f
    $n \leftarrow$ length(f)
    G = SymmetricGroup(n) ▷ Constructing S_n
    groupSize = len(G)
    parts = Partitions(n).list() ▷ Determining all partitions
    numParts = len(parts)
    ftResult = ∅

```
    for i in range(numParts) do
        rep = SymmetricGroupRepresentation(parts[i], "orthogonal")
        idRep = rep(G[0])
        ftResult[str(parts[i])] = idRep
        for j in range(1, groupSize) do
            ftResult[str(parts[i])] += (func(G[j]))*rep(G[j])
        end for
    end for
    return ftResult
end function
```

Remark. The commands in the pseudo code are actually valid code in python with the package "Sage" imported.

As one may have already noticed, using this formula directly is rather slow: with time complexity $O(n^2)$, as computing each value requires $O(n)$ calculations. The two algorithms described below will provide a more efficient algorithm by taking advantage of the symmetries.

3.1. **The Cooley-Tukey Fast Fourier Transform.** The Cooley-Tukey Fast Fourier Transform can be traced back to Gauss, it computes the Fourier transform on a finite cyclic group of order n, where n is a power of 2. This probably is the most commonly used FFT and has been implemented and compressed in many numerical computing packages.

The basic idea behind the Cooley-Tukey FFT is the "devide and conquer" technique, which is achieved by divide the problem in several sub-problems with smaller size, and solve the sub-problems using the same techniques recursively. In our problem, We can split (5) into two separate sums, one for the even terms and the other for the odd:

$$\begin{aligned}\hat{f}(y_j) &= \sum_{i=0}^{\frac{n}{2}-1} f(x_{2i}) e^{\frac{-2\pi i(2x_i)y_i}{n}} + \sum_{i=0}^{\frac{n}{2}-1} f(x_{2i+1}) e^{\frac{-2\pi i(2*x_i+1)y_i}{n}} \\ &= \sum_{i=0}^{\frac{n}{2}-1} f(x_{2i}) e^{\frac{-2\pi i x_i y_i}{\frac{n}{2}}} + e^{-\frac{2\pi i j}{n}} \sum_{i=0}^{\frac{n}{2}-1} f(x_{2i+1}) e^{\frac{-2\pi i x_i y_i}{\frac{n}{2}}} \end{aligned} \tag{6}$$

To make it more clear, we can rewrite the above formular as the iteration scheme as follows:

$$\hat{f}(y_j) = \hat{f}_e(y_j) + e^{-\frac{2\pi i j}{n}} \hat{f}_o(y_j).$$

where $\hat{f}_e(y_j)$ and $\hat{f}_o(y_j)$ are the even and odd terms of $\hat{f}(y_j)$, and they can futher divided until there is only one term in the summation. Thus, for calculating $\hat{f}(y_j)$, we will need $\log_2(n)$ layers, and thus $\log_2(n)$ operations. This means that the total time complexity for the Cooley-Tukey FFT is $O(n \log_2 n)$, which is a significant improvement than the naive Fourier Transform.

The pseudo code for the Cooley-Tukey FFT is as follows:

```
function FFT(f)                          ▷ f is an array storing sampled values of f
    n ← length(f)
    if n = 1 then
        return f                         ▷ base case, in which f̂ = f
    else
```

```
        evens ← splitEvens(f)          ▷ splitEvens: helper: get even indexed elements
        odds ← splitOdds(f)            ▷ splitOdds: helper: get odd indexed elements
        t_evens ← FFT(evens, n/2)      ▷ Compute FT recursively
        t_odds ← FFT(odds, n/2)
    end if
    f̂ ← ∅
    for j ∈ {1, 2, ··· , n/2 − 1} do
        f̂[j] ← t_evens[j] + e^{−2πij/n} t_odds[j]
        f̂[j + n/2] ← t_evens[j] − e^{−2πij/n} t_odds[j]
    end for
    return f̂
end function
```

3.2. **The Diaconis-Rockmore Fast Fourier Transform.** Although the Cooley-Tukey FFT 3.1 is quick and easy to use, it has limitations since it can only work for cyclic groups and when n is a power of 2. A more general approach, developed by Diaconis and Rockmore, can work on all finite groups. The general idea behind Diaconis-Rockmore FFT is still divided and conquer, but instead of dividing the summation into even and odd terms(which is only applicable when n is a power of 2), they consider a series of subgroups of G such that $G \supset H_1 \supset H_2 \supset \cdots \supset H_n = id$.

More spercifically, suppose H is a subgroup of G, with $[G : H] = n$. Then we have that there exists a set of elements $\{g_1, \cdots, g_n\}$ where $g_i \in G$ such that $\sqcup_{i=1}^{n} g_i H = G$. We call this set a coset representation of H in G. By the above discussion, and recall

$$\begin{aligned} \hat{f}(\rho) &= \sum_{g\in G} f(g)\rho(g) = \sum_{i=1}^{n}\sum_{h\in H} f(g_i h)\rho(g_i h) \\ &= \sum_{i=1}^{n} \rho(g_i) \sum_{h\in H} f_i(h)\rho(h) = \sum_{i=1}^{n} \rho(g_i)\hat{f}_i(\rho\mid_H) \end{aligned} \tag{7}$$

where $f_i(h) \triangleq f(g_i h)$, and the second equality on the second line is achieved by observing that $\sum_{h\in H} f_i(h)\rho(h)$ is the Fourier transform of f_i on the representation ρ restriced to H. To resolve the issue that $\rho\mid_H$ is not guarenteed to be irreducible, we know that it can be decomposed into the direct sum of finitely many irreducible representations. Thus, we can summarize the general steps for the Diaconis-Rockmore FFT as follows:

(1) Choose a subgroup H of G, and determine the corresponding left transversal $\{g_1, \cdots, g_n\}$
(2) For each g_i, apply the Fourier Transform of f_i on some set of irreducible representations of H.
(3) For each $\rho \in \hat{G}$, construct $\hat{f}_i(\rho\mid_H)$.
(4) Sum over the $\hat{f}_i(\rho\mid_H)$ and use (7) to find the Fourier Transform of f.

Remark. By letting $G = \mathbb{Z}/2^n\mathbb{Z}$, and use the chain $\mathbb{Z}/2^n\mathbb{Z} \supset \mathbb{Z}/2^{n-1}\mathbb{Z} \supset \cdots \mathbb{Z}/2\mathbb{Z} \supset\supset id$, we can see that the Diaconis-Rockmore FFT reduces to be the Cooley-Tukey FFT.

3.2.1. *The Diaconis-Rockmore FFT in S_n.* As discussion above, the Diaconis-Rockmore FFT is rather complicated and the detailed calculation varies when the structure of G varies. To make the algorithm more clear, in this subsubsection, we focus on the Fourier Transform on S_n. The choice of S_n is not random since according to Cayley's theorem, every finite group

of order n is isomorphism to a subgroup of S_n, and there is a natural choice of the chain of subsets as $G = S_n \supset S_{n-1} \supset \cdots \supset S_1 = id$.

One of the hard part of the Diaconis-Rockmore FFT is finding the irreducible representations of S_n, we shall benefits from the following results to ease the calculation.

Definition. An integer partition of a number n is set $\{\lambda_1, \cdots, \lambda_k\}$ where each $\lambda_i \in \mathbb{N}$, $\lambda_1 \geq \cdots \geq \lambda_k$ and $\lambda_1 + \cdots + \lambda_k = n$.

Theorem 8. *The set of integer partitions of n is in bijection with $\hat{S}_n$, the set of irreducible representations of S_n [Dia88].*

Theorem 9 (The Branching Theorem [Dia88])**.** *Let ρ be an irreducible representation on S_n, represented by the partition $\{\lambda_1, \cdots, \lambda_k\}$ of n. We know that $\rho\,|_{S_{n-1}}$ splits into a direct sum over a collection of irreducible representations, which can be created by subtracting 1 from any λ_i while maintaining the decreasing order of the integer partition.*

By the aid of the Branching Theorem, we now can index the irreducible representation of S_n by the partitions of n, and we can write down the pseudo code for the Diaconis-Rockmore FFT on S_n. We start by implementing some helper functions.

Remark. The commands in the following pseudo codes are actually valid code in python with the package "Sage" imported.

We first write the helper function getCosetReps, which gives us the representatives of the coset of S_n/S_{n-1}. We will use the following lemma to make the calculation easier:

Lemma 10. *Let $r = (1, 2, \cdots, n) \in S_n$, then $\{r, r^2, \cdots, r^n = e\}$ is a set of coset representatives of S_n/S_{n-1}.*

Proof. Suppose $r^a S_{n-1} = r^b S_{n-1}$ for some a, b, then $r^{a-b} \in S_{n-1}$. Since r doesnot fix the element n, we must have $a = b$, which finishes the proof. □

Now the function for the coset representation can simply be written as:

function nCycle(n)
  return $[n] + list(range(1, n))$
end function
function getCosetReps(G, n)
  $H = G$.subgroup([nCycle[n]])
  return H.list()
end function

To mimic $f_i : h \to f(g_i h_i)$, we have

function buildFunc(f, g)
  function newFunc(h)
    return $f(gh)$
  end function
  return newFunc ▷ This is returning the reference of the function.
end function

The function for computing all the partitions as the representatives of S_{n-1} is a typical programming exercise problem in leetcode. The implementation can be as follows

function getSubReps(partition)

```
    newPartList = []
    k = len(partition)
    for i in range(k - 1) do
        if partition[i] ¿ partition[i + 1] then                ▷ Check if valid partition
            newPart = partition.copy()
            newPart[i] -= 1
            newPartList.append(newPart)
        end if
    end for
    lastPart = partition.copy()
    if lastPart[k - 1] ¿ 1 then:
        lastPart[k - 1] -= 1
    else
        del lastPart[-1]
    end if
    newPartList.append(lastPart)
    return newPartList
end function
```

Above all, we can finally write down the code for the Fast Fourier Transform:

```
function FASTFT(f)
    n = len(f)
    G = SymmetricGroup(n)
    if n = 1 then
        return {"[1]": matrix([f(G[0])])}                      ▷ Returning a dictionary
    else
        cosetReps = getSubReps(G, n)
        recursed = ∅
        for i in range(n) do
            funcG = buildFunc(f, cosetReps[i])
            recursed[i] = fastFT(funcG)                        ▷ recursive step
        end for
        ftRestricted = ∅
        parts = Partitions(n).list()
        numParts = len(parts)
        for i in range(n) do
            ftRestricted[i] =                                  ▷ An empty dictionary
            for j in range(numParts) do                        ▷ Building retricted transform
                blockMat = makeBlockFT(recursed[i], parts[j])
                ftRestricted[i][str(parts[j])] = blockMap
            end for
        end for
        ftResult =                                             ▷ computing final FT results
        for j in range(numParts) do
            rep = SymmetricGroupRepresentation(parts[j], "orthogonal")
            for i in range(n) do
                if i == 0 then
```

```
                start = rep(cosetReps[i])                    ▷ Initialize the matrix
                Start *= ftRestricted[i][str(parts[j])]
                ftResult[str(parts[j])] = start
            else
                addOn = rep(cosetReps[i])
                addOn *= ftRestricted[i][str(parts[j])]
                ftResult[str(parts[j])] += addOn
            end if
        end for
    end for
end if
return ftResult
end function
```

Finally, we attached the performance of the naive FT and the FFT on S_n as follows:

n	1	2	3	4	5	6	7	8
slowFT	0.0004	0.0020	0.0111	0.1050	4.2183	389.92	55102	-
fastFT	0.0004	0.0018	0.0110	0.1053	4.1372	69.22	1685.0	5804.6

As shown in the table, the FFT has a much better performance when n is large. In fact, the time complexity for the FFT is approximately $O(n \log_2 n)$, where this $n = |G|$, which is similar to the Cooley-Tukey algorithm.

4. Applications in Multi-object Tracking Problem

Essentially, the multi-object tracking problem is a data association problem which can be modeled by a permutation $\sigma \in S_n$, where $\sigma(i) = j$ signifies that target i is associated with track j. A probabilly distribution over permutations is represented by an $n!$ dimensional vector where each component $p(\sigma)$ is a probability density function.

4.1. Canonical Projection for Dimension Reduction.

By the above discussion, when n is big, maintaining such $n!$ information is very costly. To resolve this, we need to find a suitable canonical projection to reduce the dimension of the problem. Such projection(from $\mathbb{R}^{n!}$ to W(a subspace)) has to satisfies the following criteria:

- The projection can be reversed with minimal loss of information.
- The projection should be symmetric in the sense that each component $p(\sigma)$ suffers the same loss during the projection.
- It should be possible to apply observation and noise update directly to the projected distribution.

By the discussion in section 2, representation theory allows us to extend the concept of Fourier transformation to any finite groups, and the map $f \to \hat{f}(\rho)$ can be regarded as a projection of f onto a d_ρ^2 dimensional subspace V_ρ, where d_ρ is the dimension of the representation. Since $\hat{f}_\rho$, so-called isotypic components, captures the variation of f at a different level of complexity, we shall use this idea to reduce the dimension.

We first transform S_n into the Cayley graph, more specifically, two permutations are connected if they only differ by a transposition (i, j) for some i, j. The Laplacian of the graph is $\Delta = D - A$, where D is the degree matrix, and A, is the adjacency matrix. According to general spectral theory, the orthogonal eigenvectors $v_1, v_2, \cdots, v_n$ of Δ ordered

by their eigenvalues $\alpha_1, \alpha_2, \cdots, \alpha_n$ corresponds to the increasingly complex functions on the Cayley graph.

By the above discussion, since each isotypal subspace V_ρ is spanned by the eigenvectors of Δ with eigenvalue α_ρ, the proper subspace to project the probability distribution p is $W_{\alpha_{thres}} = \oplus_{\alpha_\rho \leq \alpha_{thres}} V_\rho$ for some threshold α_{thres}.

To apply the above low-pass-filter idea, we need to find out exactly the low frequency components. The discussion in Theorem 9 (or just use the Young diagram) give us a representative for the inequivalent irreducible representations over $\mathbb{C}$. The eigenvalues corresponding to the representations can be calculated by the following lemma:

Lemma 11. *The eigenvalue of Δ corresponding to the isotypal indexde by λ is*

$$\alpha_\lambda = \binom{n}{2}\left(1 - \frac{tr[\rho_\lambda([1,2])]}{d_\lambda}\right).$$

4.2. **Tracking Targets.** According to , a probabilistic data association algorithm must have the followint three components:

- A noise model describing how the distribution p degrades in the absence of observations.
- A rule for updating p when one or more targets are observed at specific tracks.
- An inference procedure for recovering the most likely assignment of targets to tracks.

4.2.1. *Naive Update Scheme.* Assume that with probability π, target o_i is at track r_j, then we have

$$p(O_{i\to j} \mid \sigma) = \begin{cases} \pi & \text{if } \sigma(i) = j, \\ \frac{1-\pi}{n-1} & \text{if otherwise.} \end{cases}$$

where $p(O_{i\to j}$ is the probability of target i is observed as track j. Thus we can have the following update rule:

$$p'(\sigma) = \begin{cases} \frac{1}{Z}\pi p(\sigma) & \text{for } \sigma(i) = j \\ \frac{1}{Z}\frac{1-\pi}{n-1} p(\sigma) & \text{for } \sigma(i) \neq j \end{cases} \tag{8}$$

where $Z = \pi p([\sigma(i) = j]) + \frac{1-\pi}{n-1}(1 - p([\sigma(i) \neq j])$ is the normalizer.

4.2.2. *Update Scheme with Fourier Transform.* Again, the naive update scheme is of little practically use if n is large. To resolve this, one can perform the same Fourier Transform procedure to compute the Fourier coefficient. By filtering out the high frequency terms and modifying the Naive update scheme, we can have a fast algorithm while maintaining most of the informations.

Remark. In [KHJ07], instead of using the Diaconis-Rockmore FFT, the authors used another metho called the Clausen's FFT, which results in the following equation:

$$\hat{f}(\rho)_\lambda = \sum_{j=1}^{n} \rho([j,n]) \oplus_{\lambda^-} \hat{f}_j(\rho_{\lambda^-})$$

where $[i,j]$ are permuations of the following form:

$$[p,q](i) = \begin{cases} i+1 & \text{if } p \leq i \leq q-1 \\ p & \text{if } i = q \\ i & \text{otherwise} \end{cases}$$

. The authors also "twisted" FFT to be

$$\hat{f}(\rho)_\lambda = \sum_{j=1}^{n} \rho([j,n]) \oplus_{\lambda^-} \hat{f}_j(\rho_{\lambda^-})\rho([j,n])^{-1}$$

with no explicit theoretical time complexity.

References

[Dan18] Rohan Dandavati. The fourier transform on finite groups: Theory and computation. 2018.

[Dia88] Persi Diaconis. *Chapter 7: Representation Theory of the Symmetric Group*, volume Volume 11 of *Lecture Notes–Monograph Series*, pages 131–140. Institute of Mathematical Statistics, Hayward, CA, 1988.

[KHJ07] Risi Kondor, Andrew Howard, and Tony Jebara. Multi-object tracking with representations of the symmetric group. 2:211–218, 21–24 Mar 2007.

[Ter99] Audrey Terras. *Fourier Analysis on Finite Groups and Applications*. London Mathematical Society Student Texts. Cambridge University Press, 1999.

School of Mathematics, Georgia Institute of Technology, 686 Cherry St., Atlanta GA 30332-160

E-mail address: yyao93@gatech.edu

School of Mathematics, Georgia Institute of Technology, 686 Cherry St., Atlanta GA 30332-160

E-mail address: wzhang393@gatech.edu