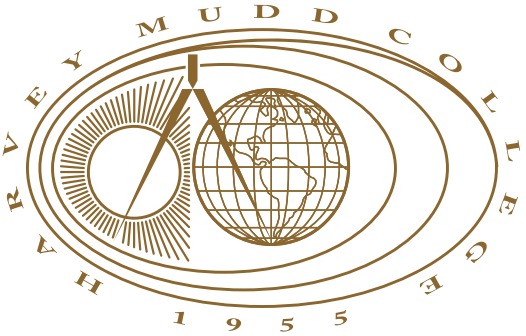
2005

Decimation-in-Frequency Fast Fourier Transforms for the Symmetric Group

Eric Malm
Harvey Mudd College



Decimation-in-frequency Fast Fourier Transforms for the Symmetric Group

Eric J. Malm

Michael Orrison, Advisor

Shahriar Shahriari, Reader

April 27, 2005

HARVEY MUDD COLLEGE

Department of Mathematics

Abstract

In this thesis, we present a new class of algorithms that determine fast Fourier transforms for a given finite group G. These algorithms use eigenspace projections determined by a chain of subgroups of G, and rely on a path-algebraic approach to the representation theory of finite groups developed by Ram (26). Applying this framework to the symmetric group, S_n, yields a class of fast Fourier transforms that we conjecture to run in $O(n^2 n!)$ time. We also discuss several future directions for this research.

Contents

Abstract **iii**

Acknowledgments **xi**

1 Introduction **1**
1.1 Introduction 1
1.2 Group-Theoretical Fourier Transforms 3
1.3 Algorithmic Approaches to FFTs 9
1.4 FFTs for the Symmetric Group 12
1.5 Applications 13
1.6 Open Questions 14

2 Character Graphs and Seminormal Representations **17**
2.1 Character Graphs 17
2.2 Path Algebras 19
2.3 Seminormal Matrix Representations 21
2.4 Applications to MC-Groups 24

3 Representation Theory of the Symmetric Group **27**
3.1 Constructions of Irreducible Representations 27
3.2 Reformulation of Path-Algebraic Techniques 31
3.3 Seminormal Matrix Representations 35
3.4 Computation and Examples of Representations 39
3.5 Conclusions and Generalizations 41

4 Decimation-In-Frequency Algorithm Theory **43**
4.1 The DFT as a Change of Basis 43
4.2 Path Algebras, DFTs and FFTs 45
4.3 Bimodules and Opposite Algebras 46
4.4 Double-Coset Branchings and Bases 48

4.5 Projections and Minimum Rank Decompositions 52
4.6 Decimation-in-Frequency Algorithms 53
4.7 Bases and Regular Representations 62
4.8 Computation of Double-Coset Projections 66
4.9 Conclusion 70

5 Fast Fourier Transforms for the Symmetric Group 73
5.1 Computation of Coset Bases 74
5.2 Separating Elements 76
5.3 Eigenvalue List Completion 77
5.4 Computation of Final Permutation Matrix 80
5.5 Computation of Scaling Matrix 81

6 Initial Implementation and Results 85
6.1 *Mathematica* Implementation 85
6.2 Precomputation 86
6.3 Evaluation 87
6.4 Multiplication and Convolution 90

7 Future Directions and Conclusions 93
7.1 Double-Coset Bases and Module Decompositions 93
7.2 Row Reduction and Choice of Basis 94
7.3 Efficiency of Precomputation 94
7.4 Efficiency of Evaluation 95
7.5 MATLAB and GAP Implementations 95
7.6 Parallel Implementations 95

A Computational Examples 97
A.1 $\mathbb{C}S_3$ with Idempotents 98
A.2 $\mathbb{C}S_3$ with Jucys-Murphy Elements 105

B Tabulation of Double Coset Irreducibles 109
B.1 Double-Coset Modules in $\mathbb{C}S_3$ 110
B.2 Double-Coset Modules in $\mathbb{C}S_4$ 110
B.3 Double-Coset Modules in $\mathbb{C}S_5$ 111

C Mathematica Code: FFT Generation Algorithm 115

Bibliography 137

List of Figures

2.1 Character Graph for $\mathbb{Z}/6\mathbb{Z}$ 19
2.2 Character Graph for S_3 21

4.1 Double-Coset Branching for S_3 50

6.1 Graphical Representation of Factorization 89

List of Tables

3.1 Action of Transpositions on $(3,2)$-Tableaux 42

5.1 Eigenvalue Completion 80

6.1 Precomputation times 86
6.2 FFT Evaluation Operation Counts 88
6.3 FFT Inverse Operation Counts 89
6.4 Convolution Operation Counts 90

A.1 Character Tables for S_2, S_3. 98

B.1 (S_2, S_2)-Double Cosets in $\mathbb{C}S_3$. 110
B.2 (S_2, S_2)-Double Cosets in $\mathbb{C}S_4$. 110
B.3 (S_3, S_2)-Double Cosets in $\mathbb{C}S_4$. 111
B.4 (S_3, S_3)-Double Cosets in $\mathbb{C}S_4$. 111
B.5 (S_2, S_2)-Double Cosets in $\mathbb{C}S_5$. 112
B.6 (S_3, S_2)-Double Cosets in $\mathbb{C}S_5$. 112
B.7 (S_3, S_3)-Double Cosets in $\mathbb{C}S_4$. 113
B.8 (S_4, S_3)-Double Cosets in $\mathbb{C}S_4$. 113
B.9 (S_4, S_4)-Double Cosets in $\mathbb{C}S_4$. 114

Acknowledgments

I would like to thank my thesis advisor, Michael Orrison, for his insight, support, and guidance on this project, and my second reader, Shahriar Shahriari, for his helpful commentary. Additionally, I would like to thank Lisa Lambeth for her patience and her willingness to endure early drafts of chapters, Claire Connelly for her excellent thesis class and LaTeX assistance, and my friends and parents for their support and encouragement.

Chapter 1

Introduction

1.1 Introduction

Spectral analysis methods play a crucial role in mathematics today and in the pure and applied sciences. For example, the study of Fourier series concerns itself with the decomposition of a periodic function f into a series of complex exponentials (13):

$$f(x) = \sum_{n=-\infty}^{\infty} c_n e^{inx}. \tag{1.1}$$

The amplitudes c_n of these exponentials then constitute the spectrum of the function. Such decompositions have applications to linear partial differential equations, where the exponential functions act as a basis of eigenfunctions of the linear operator associated with the equation. The eigenvalue spectrum of the operator then determines how the solution to the equation depends on the initial and boundary conditions (13; 17). This series decomposition can be extended to nonperiodic functions by allowing a continuous distribution of frequencies for the constituent complex exponentials, so that a function $f(x)$ decomposes as

$$f(x) = \int_{-\infty}^{\infty} g(\omega) e^{i\omega x}\, d\omega. \tag{1.2}$$

The spectrum of amplitudes $g(\omega)$ is then the Fourier transform of the function $f(x)$. This terminology also illustrates that the Fourier transform itself is a map from one space of functions to another space of functions, possibly over a different domain.

Spectral methods, and the Fourier transform in particular, have significant applications in the physical sciences. In quantum mechanics, for instance, the eigenvalue spectrum of a Hermitian operator such as the Hamiltonian or the angular momentum operator determines the range of observable values for the physical quantity corresponding to that operator. The position-space wave function $\psi(x)$ of a particle describes the distribution of its possible positions states, and can be rewritten in terms of its momentum-space wave function $\hat{\psi}(p)$ in what is in fact a Fourier transform on $\mathbb{R}$ (31):

$$\psi(x) = \frac{1}{\sqrt{2\pi\hbar}} \int_{-\infty}^{\infty} \hat{\psi}(p) e^{ipx/\hbar}\, dp.$$

The physical significance of these transforms arises from the natural duality between quantities such as position and momentum and energy and time. This same duality underlies the famous Heisenberg uncertainty relations $\Delta x \Delta p \geq \hbar/2$ and $\Delta E \Delta t \geq \hbar/2$.

Spectral methods are also of major significance in engineering and the applied sciences. For example, much of modern signal processing concerns determining not only the spectrum of a given input signal, but also how that spectrum will change when the signal passes through particular systems and how to design systems that amplify or isolate certain portions of the spectrum. Of particular importance to signal processing is the Discrete Fourier Transform (DFT), which converts a function on N evenly spaced points to N amplitudes associated to certain frequencies. In keeping with the terminology above, these amplitudes are called the Fourier coefficients of the function. This transform allows discrete samples of a continuous signal to yield some information about the spectrum of that signal. Because computational methods operate primarily on such discrete data, the DFT has become ubiquitous in modern signal processing.

Naïve implementations of the DFT require $O(N)$ operations for the construction of each coefficient, resulting in an overall $O(N^2)$ algorithm for the DFT. This $O(N^2)$ complexity severely limits the application of the DFT to large data sets and motivates the search for more efficient implementations of the transform. Any such efficient implementation of the DFT is called a Fast Fourier Transform (FFT). Such FFTs trace back even to Gauss, who determined an efficient interpolation of a planetary orbit between n points from its interpolation on two sets of $n/2$ points. Modern FFTs are derived from the algorithm that Cooley and Tukey developed in the 1960s (6; 27), which computes a DFT on $N = pq$ points first by p transforms of length q and then by q transforms of length p, for a total of $pq(p+q)$ operations.

Applied recursively in the case where $N = 2^n$, this algorithm yields a complexity of $O(N \log N)$ for the N-point DFT, a significant improvement over the naïve $O(N^2)$ implementation.

1.2 Group-Theoretical Fourier Transforms

As discussed above, given a continuous periodic function $f : \mathbb{R} \to \mathbb{C}$, we can determine some of its frequency spectrum by sampling the function at N points $x_0, x_1, \ldots, x_{N-1}$ over one of its periods and computing the DFT on those points. One of the key features of the DFT is that the magnitudes and relative phases of the coefficients it yields do not depend on the time period over which the function was sampled. In particular, this indicates that the DFT is invariant under cyclic shifts of the points $X = \{x_0, x_1, \ldots, x_{N-1}\}$. Such shifts correspond to the action of the group $\mathbb{Z}/N\mathbb{Z}$ on this set X. We obtain the same shifting action if we replace the N points of X with the corresponding elements of $\mathbb{Z}/N\mathbb{Z}$, however. Hence, this function f on the points of X can instead be thought of as a function on the elements of $\mathbb{Z}/N\mathbb{Z}$. Finally, we can treat the function $f : \mathbb{Z}/N\mathbb{Z} \to \mathbb{C}$ as an element of the group algebra $\mathbb{C}(\mathbb{Z}/N\mathbb{Z})$, where the coefficient of $g \in \mathbb{Z}/N\mathbb{Z}$ is $f(g)$. This mapping of the function $f : X \to \mathbb{C}$ into $\mathbb{C}(\mathbb{Z}/n\mathbb{Z})$ provides a representation-theoretic interpretation of the DFT and an avenue for its generalization to arbitrary groups.

Suppose G is a finite group with h conjugacy classes, and let M be a $\mathbb{C}G$-module, so that M is a representation for G. Let $\hat{G}$ denote the set of h equivalence classes of irreducible representations of G, and let $U_1, \ldots, U_h$ be representatives of these equivalence classes. By Maschke's Theorem, M decomposes as

$$M = \bigoplus_{i=1}^{h} M_i, \tag{1.3}$$

where each M_i is isomorphic to a direct sum of n_i isomorphic copies of U_i, so that $M_i \cong n_i U_i$. These M_i are referred to as the *isotypic components* of M and are unique for each $\mathbb{C}G$-module M.

In particular, $\mathbb{C}G$ is itself a $\mathbb{C}G$-module by left multiplication, and so it too decomposes into a direct sum of its isotypic components. The isotypic components in this decomposition correspond to the minimal two-sided ideals of $\mathbb{C}G$, while the irreducible representations correspond to its minimal left ideals. Thus, each element $f \in \mathbb{C}G$ can be written as a unique sum of elements in the representations constituting this direct sum, and these

elements provide a generalization of the Fourier coefficients obtained by the usual DFT.

These coefficients can be written explicitly by considering each irreducible representation U_i as a vector space over $\mathbb{C}$ of dimension equal to the degree d_i of the representation. Then each component of $f \in \mathbb{C}G$ corresponds to a vector in $\mathbb{C}^{d_i}$, and each isotypic component of f corresponds to a matrix in $\mathbb{C}^{d_i \times d_i}$. Moreover, our module isomorphism $\mathbb{C}G = \bigoplus_{i=1}^{h} M_i \cong \bigoplus_{i=1}^{h} d_i U_i$ extends to an algebra isomorphism to yield Wedderburn's Theorem (6; 11):

Theorem 1.1 *The group algebra $\mathbb{C}G$ of a finite group G is isomorphic to an algebra of block diagonal matrices:*

$$\mathbb{C}G \cong \bigoplus_{i=1}^{h} \mathbb{C}^{d_i \times d_i}. \tag{1.4}$$

■

This isomorphism provides the generalization of the DFT that we seek:

Definition 1.2 Every $\mathbb{C}$-algebra-isomorphism $D : \mathbb{C}G \to \bigoplus_{i=1}^{h} \mathbb{C}^{d_i \times d_i}$ is called a *discrete Fourier transform* (DFT) for $\mathbb{C}G$, or simply for G. The coefficients of the matrix $D(f)$ are called the *Fourier coefficients* of f. ■

Because we have a choice of basis for each $\mathbb{C}^{d_i \times d_i}$, such isomorphisms are not unique. Combined with a choice of basis for $\mathbb{C}G$, we can reformulate the DFT as a $|G| \times |G|$ matrix from the coordinate representation of $f \in \mathbb{C}G$ to the coordinate representation of its transform $D(f) \in \bigoplus_{i=1}^{h} \mathbb{C}^{d_i \times d_i}$. This matrix form also indicates that the DFT is a linear transformation from the input function to the coefficients.

A DFT for G is also equivalent to a complete set of inequivalent irreducible matrix representations for G: each block in the matrix algebra yields such a representation, and given a collection $\{\rho_j\}_{j=1}^{h}$ of matrix representations such that each ρ_j corresponds to a distinct irreducible type, their direct sum determines a DFT. Hence, given such a matrix representation ρ of G associated to the jth irreducible type and an element $f = \sum_{g \in G} f_g g \in \mathbb{C}G$, we can compute the Fourier coefficients in block j of the matrix algebra for the corresponding DFT as

$$\hat{f} = \sum_{g \in G} f_g \rho(g). \tag{1.5}$$

We use such matrix representations below to construct a discrete Fourier transform for the symmetric group.

1.2.1 Abelian Groups

We now illustrate how this concept of generalized Fourier transforms encompasses the familiar cases of spectral analysis discussed in Section 1.1, all of which then correspond to Fourier transforms on abelian groups. In the case of a finite abelian group G such as $\mathbb{Z}/N\mathbb{Z}$, each irreducible representation of G has degree 1, so the isotypic subspaces of $\mathbb{C}G$ are all one-dimensional. Therefore, Wedderburn's Theorem states that

$$\mathbb{C}G \cong \bigoplus_{i=1}^{|G|} \mathbb{C}^{1\times 1} \cong \mathbb{C}^{|G|}, \tag{1.6}$$

and the Fourier coefficients of $f \in \mathbb{C}G$ are simply the coordinates in $\mathbb{C}^{|G|}$ of the image of f under this isomorphism.

We can also reformulate the Cooley-Tukey FFT on $N = pq$ points in terms of a factorization of the group $\mathbb{Z}/N\mathbb{Z}$. The analysis presented here closely follows that presented by Maslen and Rockmore (20). The N irreducible representations of $\mathbb{Z}/N\mathbb{Z}$ are all one-dimensional and hence are equal to the irreducible characters of the group. These characters are given by $\zeta_k(j) = \omega^{-jk}$, where ω is a primitive Nth-root of unity. Thus, by Equation 1.5, the kth Fourier coefficient of an element $f = \sum_j f_j j \in \mathbb{C}(\mathbb{Z}/N\mathbb{Z})$ is given by

$$X_k = \sum_{j=0}^{N-1} f_j \zeta_k(j) = \sum_{j=0}^{N-1} f_j \omega^{-jk}. \tag{1.7}$$

We can reindex the sum into a double sum by the chain of subgroups $1 < \mathbb{Z}/p\mathbb{Z} < \mathbb{Z}/N\mathbb{Z}$. Each $j \in \mathbb{Z}/N\mathbb{Z}$ can be written as an element of a coset of the subgroup $q\mathbb{Z}/N\mathbb{Z} \cong \mathbb{Z}/p\mathbb{Z}$, so that $j = i_1 q + i_2$. Let A be a transversal of $q\mathbb{Z}/N\mathbb{Z}$ in $\mathbb{Z}/N\mathbb{Z}$. Then the sum above becomes

$$\begin{aligned} X_k &= \sum_{a\in A} \sum_{b\in q\mathbb{Z}/N\mathbb{Z}} \zeta_k(a+b) f_{a+b} \\ &= \sum_{a\in A} \sum_{b\in q\mathbb{Z}/N\mathbb{Z}} \zeta_k(a)\zeta_k(b) f_{a+b} \\ &= \sum_{a\in A} \zeta_k(a) \sum_{b\in q\mathbb{Z}/N\mathbb{Z}} \zeta_k(b) f_{a+b}. \end{aligned}$$

We note that, in the inner sum, ζ_k acts only on elements of the subgroup $q\mathbb{Z}/n\mathbb{Z}$, so we need consider only the restrictions $(\zeta_k{\downarrow}_{q\mathbb{Z}/N\mathbb{Z}})$ of the characters ζ_k to this subgroup. Since

$$\zeta_k(i_1 q) = \omega^{-i_1 qk} = (\omega^q)^{-i_1 k},$$

and ω^q is a primitive pth-root of unity, these restrictions correspond exactly to the p irreducible characters χ_m of $\mathbb{Z}/p\mathbb{Z}$. Consequently, we need compute the inner sum only for $k \in \mathbb{Z}/p\mathbb{Z}$.

We now reformulate the calculation of the Fourier coefficients in two stages, at the expense of some storage space:

- We compute and store $f_1(a,k) = \sum_{b \in q\mathbb{Z}/N\mathbb{Z}} \zeta_k(b) f_{a+b}$ for all $a \in A$ and for all $k \in \mathbb{Z}/p\mathbb{Z}$. Each sum requires p operations, for a total of p^2q operations.
- We then compute $\sum_{a \in A} \zeta_k(a) f_1(a,k)$ for all $k \in \mathbb{Z}/N\mathbb{Z}$ to obtain the Fourier coefficients. Each sum requires q operations, for a total of $Nq = pq^2$ operations.

The final operation count for these two stages is then $pq(p+q)$ operations, while the total count for the one-stage calculations given by Equation (1.7) is $N^2 = p^2q^2$. This algorithm therefore represents a significant improvement in complexity, and ultimately leads to the $O(N \log N)$ running time of the Cooley-Tukey FFT.

Abelian groups other than the cyclic groups afford similar DFTs and FFTs. We consider an example from Maslen and Rockmore (20), called the 2^n-factorial design, which corresponds to functions on $(\mathbb{Z}/2\mathbb{Z})^n$. Such a set of data might arise from the effects of n independent factors on the growth of a crop of plants, where each factor can assume either a high value or a low value. The irreducible representations of $(\mathbb{Z}/2\mathbb{Z})^n$ are, as above, all one-dimensional, and are given by $\chi_v(w) = (-1)^{\langle v,w \rangle}$, where $v, w \in (\mathbb{Z}/2\mathbb{Z})^n$ and $\langle v,w \rangle$ represents the usual inner product taken mod 2. We decompose the computation of the Fourier coefficients by Equation (1.5) into n stages according to the chain of subgroups

$$1 < \mathbb{Z}/2\mathbb{Z} < (\mathbb{Z}/2\mathbb{Z})^2 < \cdots < (\mathbb{Z}/2\mathbb{Z})^{n-1} < (\mathbb{Z}/2\mathbb{Z})^n$$

in a manner similar to the separation performed in the Cooley-Tukey FFT to yield a transform in $3 \cdot 2^n \log 2^n$ operations. The DFT under consideration here is equivalent the well-known Walsh-Hadamard transform, and the FFT algorithm we generate then represents a sparse factorization of the DFT matrix associated with the transform.

Much work has already been done on FFTs for abelian groups. The key result, presented by Clausen and Baum (6), is that the combination of the methods of Cooley and Tukey and the so-called chirp-z and Rader transforms yields FFTs for all abelian groups G in fewer than $8|G| \log |G|$ operations.

We can even extend this group theoretic formulation to infinite groups, under certain restrictions detailed below. Consider a function $f : \mathbb{R} \to \mathbb{C}$ that is 2π-periodic. We reformulate f as a function on the unit circle S^1, which is a compact group. The associated group algebra also has irreducible degree-one representations, although in this case there are an infinite number of them, indexed by $\mathbb{Z}$. Representing the elements of S^1 by $\theta \in [0, 2\pi)$, the nth irreducible representation is given by $\chi_n(\theta) = \frac{1}{2\pi} e^{-in\theta}$. Thus, the Fourier coefficients f_n are determined by the integral

$$f_n = \int_0^{2\pi} f(\theta)\chi_n(\theta)\, d\theta = \frac{1}{2\pi} \int_0^{2\pi} f(\theta) e^{-in\theta}\, d\theta. \tag{1.8}$$

In general, this integral exists because S^1 is compact and thus has finite volume under the measure $d\theta$. The inverse transform is as specified in Equation (1.1). We often require that the function f be band-limited, so that $f_n = 0$ for $|n| > B$ for some B. This restriction allows us to replace the infinite sum in Equation (1.1) with a finite sum from $-B$ to B. In this case, there exists a sampling method that allows the exact computation of the coefficients from $2B + 1$ samples of the function f (27); the finite-case FFT then makes such computations efficient.

Finally, the Fourier transform over $\mathbb{R}$ detailed above in Equation (1.2) provides an example of a transform over a noncompact abelian group. As above, the Fourier coefficients are given by an integral, although this time over $\mathbb{R}$:

$$\hat{f}(\omega) = \frac{1}{2\pi} \int_{-\infty}^{\infty} f(x) e^{-i\omega x}\, dx. \tag{1.9}$$

This result reflects the infinite number of irreducible representations of $\mathbb{R}$, as in the S^1 case, although now they are indexed by $\mathbb{R}$ instead of by $\mathbb{Z}$. In order that these coefficients $\hat{f}(\omega)$ exist, we must place certain restrictions on f. These include that it be band-limited (27) or that it belong to a special class of functions that ultimately decay faster than e^{-x^2} (17). In the band-limited case, the Fourier coefficients have finite support, so as in the S^1 case, there exist sampling methods that then admit judicious use of the FFT to create efficient transforms.

1.2.2 Nonabelian Groups

We now explore the generalization of these Fourier transforms to the case of nonabelian groups G. Among the finite groups of particular interest are the symmetric groups S_n, the dihedral groups D_{2n}, solvable and supersolvable

groups, and finite groups of Lie type, including several types of matrix groups over finite fields (16; 20; 27). Such groups have applications that include the analysis of ranked data and the construction of error-correcting codes. Section 1.5 discusses potential applications of spectral analysis on these groups in more detail.

Fourier transforms on finite nonabelian groups are even useful for understanding or manipulating the corresponding group algebras, as multiplication of elements in the group algebra corresponds to multiplication of the elements in the group (6). Hence, this multiplication can be computed through two DFTs, a matrix multiplication in the transform space, and an inverse DFT. Implemented naïvely, either approach requires $O(|G|^2)$ operations, but an FFT algorithm can reduce the complexity of the transform approach to at worst $O(|G|^{3/2})$ and in some cases $O(|G| \log^c |G|)$ for some constant $c \geq 1$, depending on the efficiency of the FFT algorithm itself.

Such applications therefore motivate us to determine how efficient the FFTs for a given group can be. Such questions are usually stated in terms of the complexity $L_s(G)$ of a group G, which is defined to be the minimum of the complexities of all the possible DFT matrices associated with G (4; 6; 20). A number of bounds on the complexity of nonabelian FFTs have already been established. Clausen (4) states that, for a finite group G, the complexity $L_s(G)$ of a generalized FFT on G is bounded above by

$$L_s(G) \leq \min_{\mathcal{C}} \{(s(\mathcal{C}) - l(\mathcal{C})) \cdot |G| + 7\sqrt{q(\mathcal{C})}|G|^{3/2}\},$$

where the minimum is taken over all possible chains $\mathcal{C}$ of subgroups $1 = G_0 < \cdots < G_n = G$ of G, where $l(\mathcal{C})$ is the length n of the chain, and where q and s are the maximum and sum, respectively, of the indices $[G_{i+1} : G_i]$ determined by the chain. While this is a significant improvement over the trivial bound of $2|G|^2$ operations, the existence of $O(|G| \log |G|)$ FFTs for abelian groups demonstrates that this is by no means a sharp bound. Also of interest are lower bounds on the complexity of an FFT, so that we can determine when we have an optimal algorithm. Clausen and Baum (6) state that $O(|G|)$ is the best lower bound that has been proved so far in computational models that allow arbitrarily large multiplications, although if limits are placed on those multiplications the lower complexity bound grows to $O(|G| \log |G|)$.

Better complexity bounds have been determined for specific families of groups. Clausen and Baum (6) prove that if G is a solvable group with a monomial DFT, then its complexity is less than $8.5|G| \log |G|$. Since all supersolvable groups meet this criterion, this result applies to them as well.

Maslen (18) proves that the symmetric group S_n has an FFT that can be evaluated in $O(|S_n| \log^2 |S_n|)$ operations. Further discussion of FFTs for the symmetric group occurs in Section 1.4. Additonally, Maslen and Rockmore (20) demonstrate that the complexity of $GL_n(F_q)$ is bounded above by $\frac{1}{2}2^{2q}q^{2n-2}|GL_n(F_q)|$.

As in the commutative case, these Fourier transforms on finite groups can be extended to a compact group G provided certain constraints apply (27). In particular, the irreducible representations of G must be finite-dimensional, and square-integrable functions must have a countable number of coefficients so that the Fourier decomposition converges. The Fourier coefficients are then computed as integrals over the group with respect to the Haar measure. Among the groups that meet these criteria are the classical compact Lie groups, such as $O(n)$, $SO(n)$, $U(n)$, $SU(n)$, and $Sp(n)$. Maslen (27) has made progress on bounds for transforms of band-limited functions on $U(n)$, $SU(n)$, and $Sp(n)$. Driscoll and Healy, Jr. (9), furthermore, treat the 2-sphere S^2 as a homogeneous space of $SO(3)$ to construct an FFT that yields a spherical harmonic decomposition for a band-limited function on S^2.

In the noncompact case, Chirikjian (3) has made some progress with respect to Fourier transforms for the Euclidean motion group $SE(3) = SO(3) \rtimes R^3$, although a general theory of generalized FFTs on noncompact nonabelian groups has not yet been developed. Section 1.6 addresses current open questions in this and other aspects of FFT research.

1.3 Algorithmic Approaches to FFTs

1.3.1 Decimation-in-Time Algorithms

We now discuss different methods of constructing FFT algorithms. The majority of current FFT algorithms employ a *decimation-in-time* or *separation of variables* approach, in which the elements of the group G are factored according to a particular chain of subgroups $1 = G_0 < G_1 < \cdots < G_n = G$. As in the Cooley-Tukey case, the frequencies are then computed through a series of nested sums. The factorization that the subgroup chain affords reduces the total number of operations that must be performed to compute the sums at each stage.

Such algorithms typically produce the DFT corresponding to the seminormal matrix representations for G adapted to the chain of subgroups used to factor the group (6). By Maschke's Theorem, the restriction of a

matrix representation ρ of G to a subgroup G_i in this chain will decompose into a direct sum of representations for G_i. The feature of a seminormal representation, however, is that the representing matrices are partitioned into matrix direct sums under these restrictions, eliminating the need for a further change of basis to bring them into block-diagonal form. We explore the construction of these seminormal representations in general and for the symmetric group in Chapters 2 and 3, respectively.

Decimation-in-time algorithms rely upon writing elements of the group G as elements of the double cosets of G_{n-1}, where the coset representatives are drawn from some fixed transversal (20). These representatives are subsequently represented as elements of the double cosets of G_{n-2} and so on until an entire factorization of the group with respect to this chain is reached. Then, just as in the algebraic approach to the Cooley-Tukey algorithm presented in Section 1.2.1, the computation of the Fourier coefficients can be approached in stages relating to the chosen chain of subgroups. Finally, the seminormal basis ensures that the representations in the sum will restrict to direct sums of representations of subgroups, reducing the number of terms in each sum.

1.3.2 Decimation-in-Frequency Algorithms

Decimation-in-frequency algorithms present an approach to these FFTs that is essentially dual to the decimation-in-time approach. In particular, seminormal representations of the group adapted to a suitable chain of subgroups are still used, but in these algorithms the frequency space is decomposed systematically according to the irreducible representations of the chosen chain of subgroups.

We illustrate this approach with an algebraic description of the Gentleman-Sande FFT (19; 27). In order to do so, we first discuss the notion of a separating set for a representation M of a group G. Recall that M decomposes into isotypic subspaces M_i, as in Equation (1.3). Consider a set of simultaneously diagonalizable linear transformations $\{T_1, \ldots, T_k\}$ on M such that the eigenspaces of the T_js are direct sums of the isotypic components of M. Then applying each T_j to each M_i yields a list $c_i = (\lambda_{i1}, \ldots, \lambda_{ik})$ of the eigenvalues of the T_js on M_i. If $c_i = c_j$ implies that $M_i = M_j$, we say that the T_is form a *separating set* for M. In this case, the T_is suffice to distinguish among the isotypic components of M.

Consider now the case of a group algebra $\mathbb{C}G$ acting on itself as a left $\mathbb{C}G$-module. Then $\mathbb{C}G$ is also a representation of G, as discussed in Section 1.2, and the elements of $\mathbb{C}G$ act as linear transformations of $\mathbb{C}G$. Thus,

we can represent a separating set for $\mathbb{C}G$ as a collection of elements of $\mathbb{C}G$. One such separating set is the collection of centrally primitive idempotents $\{e_1, \ldots, e_h\}$ that correspond to the two-sided ideals of $\mathbb{C}G$, as e_i has eigenvalue 1 on the ith isotypic component and 0 elsewhere. These idempotents form a basis for the space of class sums in $\mathbb{C}G$, so any linear combination of class sums is also a diagonalizable linear transform on $\mathbb{C}G$, and any set of them can be diagonalized simultaneously (29). This result also indicates that the set of class sums is a separating set for $\mathbb{C}G$ (19).

We now address the DFT from a separating set perspective. We borrow the algebraic formulation of the conventional DFT as an isomorphism of $\mathbb{C}(\mathbb{Z}/N\mathbb{Z})$ from Section 1.2.1. Since each irreducible representation of $\mathbb{C}(\mathbb{Z}/N\mathbb{Z})$ is one-dimensional, so are the isotypic subspaces of $\mathbb{C}(\mathbb{Z}/N\mathbb{Z})$. Hence, these isotypics correspond to the Fourier coefficients. The representations are given by $\zeta_j(i) = \omega^{-ij}$, where ω is a primitive Nth root of unity. Furthermore, the conjugacy class sum $T_1 = \bar{1}$ separates these isotypic components, since $\zeta_j(\bar{1}) = \omega^{-j}$ and each of these is distinct for distinct j. Thus, each isotypic component V_j corresponds to the eigenspace of T_1 with eigenvalue ω^{-j}. To isolate the Fourier coefficients of $f \in \mathbb{C}(\mathbb{Z}/N\mathbb{Z})$, we then compute the projections of f onto these spaces. Doing so by the projection formula

$$f_i = \left(\frac{d_i}{|G|} \sum_{g \in G} \chi_i(g)^* \rho(g) \right) f \tag{1.10}$$

given in (19) or (29) requires $O(N)$ operations for each of the N coefficients, however, which yields an $O(N^2)$ algorithm.

The Gentleman-Sande FFT, and decimation-in-frequency algorithms in general, take advantage of a chain of subgroups of G to compute the Fourier coefficients in a series of projections, just as the decimation-in-time algorithms construct the coefficients in a series of sums. The idea in the Gentleman-Sande FFT is first to consider the effect of the class sum $T_q = \bar{q}$, which is a separating set for $\mathbb{C}(\mathbb{Z}/N\mathbb{Z})$ as a $\mathbb{C}(q\mathbb{Z}/N\mathbb{Z}) \cong \mathbb{C}(\mathbb{Z}/p\mathbb{Z})$-module. Thus, we first project $f \in \mathbb{C}(\mathbb{Z}/N\mathbb{Z})$ onto the eigenspaces $W_0, \ldots, W_{q-1}$ of T_q, each of which then consists of a direct sum of q isotypic subspaces of $\mathbb{C}(\mathbb{Z}/N\mathbb{Z})$:

$$W_k = V_k \oplus V_{k+p} \oplus \cdots \oplus V_{k+(q-1)p}. \tag{1.11}$$

Each projection then takes only $O(pq)$ operations to compute, for a total of $O(p^2q)$ operations. Then the projections via T_1 onto the $N = pq$ isotypics of $\mathbb{C}(\mathbb{Z}/N\mathbb{Z})$ as a $\mathbb{C}(\mathbb{Z}/N\mathbb{Z})$-module take only $O(q)$ operations per coefficient, for a total of $O(pq^2)$ operations (19). The overall complexity is $O(pq(p+q))$, the same as for the Cooley-Tukey FFT.

Decimation-in-frequency algorithms for a finite group G follow a pattern similar to that of the Gentleman-Sande FFT. If G is nonabelian, however, some of the isotypic subspaces of $\mathbb{C}G$ will have dimension greater than one and will no longer correspond directly to the Fourier coefficients of $\mathbb{C}G$. Consequently, the class sums of elements from G alone will not suffice to distinguish the Fourier coefficients, as they do in the case of abelian groups. Nevertheless, just as we use the class sum $\bar{q}$ in the Gentleman-Sande FFT above to improve the efficiency of the computation, we can introduce additional separating elements corresponding to the subgroup chain to produce the desired efficient decomposition into one-dimensional Fourier coefficient spaces. We discuss more general formulations of this approach in Chapter 4.

1.3.3 Convolution Algorithms

Other algorithms have been used in the case of the Cooley-Tukey FFT to increase the efficiency of transforms on $\mathbb{Z}/p\mathbb{Z}$ for large prime p. These groups have no nontrivial subgroups, so they are susceptible to neither the decimation-in-time nor the decimation-in-frequency approaches described above. One useful algorithm in this setting is the Rader transform (6; 20), which relates the DFT on p points to a convolution on $(\mathbb{Z}/p\mathbb{Z})^\times$, which is a cyclic group of order $p-1$. If $p-1$ contains a number of small prime factors, these convolutions themselves are efficient by the usual Cooley-Tukey methods and in turn provide an FFT for these p points. Similarly, the chirp-z transform uses a different change of variables to relate the DFT to a convolution on a larger cyclic group. If this cyclic group has order equal to a power of 2, this convolution can again be performed efficiently by Cooley-Tukey.

1.4 FFTs for the Symmetric Group

The symmetric group presents several features that make it ideal for the study of its fast Fourier tranforms. First, its representation theory is well understood and has recently undergone a significant reformulation (23; 26). We present the relevant features of this theory in Chapters 2 and 3. In addition, as discussed below in Section 1.5, there are known applications of DFTs on the symmetric group to frequency analysis of voting data and other ranked preference information. Moreover, since $|S_n| = n!$, the size of the group grows exponentially with increasing n, making efficient DFT al-

gorithms necessary for even moderate values of n. Finally, many common computation packages such as *Mathematica*, GAP, and MATLAB present sophisticated manipulation of permutations and combinatorial objects associated with them.

To date, significant work has been done on FFTs on the symmetric group from a decimation-in-time perspective. Clausen and Baum produced the first promising results in 1989 with a proof that the complexity of S_n is bounded above by $\frac{1}{2}(n^3 + n^2)n!$ operations (4). In 1993, they provided an explicit implementation of both a DFT and an inverse DFT for S_n, each requiring that number of operations (7). Their results arise from a sparse factorization of the DFT matrix based on Young's seminormal form at each S_i in the chain $S_1 < S_2 < \cdots < S_n$ (7; 20).

Maslen's 1998 paper (18) improves upon this bound with a decimation-in-time algorithm yielding a DFT for S_n in fewer than $\frac{3}{4}n(n-1)n!$ operations. His method relies on a separation of variables at the scalar level, rather than the matrix separation that Clausen and Baum employ. The commutativity of these scalars allows more sophisticated rearrangement of the sums involved in constructing the Fourier coefficients. This rearrangement entails a more complex indexing scheme based on the paths through a graph of irreducible representations for the subgroup chain (called a character graph and discussed in detail in Chapter 2) rather than only on the subgroup chain itself.

1.5 Applications

Applications for generalized Fourier transforms exist in engineering, mathematics, and the physical and social sciences. For example, Fourier transforms on the symmetric group have natural applications to the spectral analysis of ranked data. Each voter effectively creates a permutation in S_n by ranking their n candidates, so that the final tallies of votes yield a function on S_n which can be analyzed using the generalized Fourier transforms described above. Diaconis (8) identifies the decomposition of $\mathbb{C}S_n$ into its isotypic components as the key to understanding the effects of candidates on ranking preferences. Such transforms have been applied to partially ranked data as well (27).

The symmetric group is not the only finite group on which Fourier analysis presents applications. The group $SL_2(\mathbb{F}_p)$ of two-by-two matrices with determinant one over the finite field $\mathbb{F}_p$ has applications in coding theory, particularly with respect to low-density parity check codes, and

in graph theory (27). Maslen, Orrison, and Rockmore (19) discuss applications of generalized Fourier analysis to the study of distance-transitive graphs; while their analysis includes examples that relate primarily to the symmetric group, other groups could also be used in this context. In addition, transforms on other finite groups may yield lossy data compression algorithms with better performance than such standards as JPEG, which is based on the Discrete Cosine Transform (6). Finally, such transforms have applications to quantum mechanics and quantum computing. In particular, Shor's quantum factoring algorithm relies on transforms on the cyclic group $(\mathbb{Z}/n\mathbb{Z})^{\times}$, and it is conjectured that generalized FFTs may provide an efficient quantum algorithm for the graph isomorphism problem (27). 0.25pt

Fourier transforms on nonabelian compact groups also have significant applications. The spherical harmonics are orthogonal functions on the unit sphere S^2 that yield a series decomposition for functions on S^2 analogous to that provided by the Fourier transform on S^1. Such decompositions have applications in physics, where they play a key role in describing the distributions of electrons in atomic orbitals (12; 31). In addition, any frequency analysis of spherically distributed data rests on these spherical harmonic functions. Such analysis arises in global circulation modeling, control theory, and computer vision models, for example (20). As mentioned above, Driscoll and Healy (9) present an efficient algorithm for the computation of spherical harmonics for band-limited functions on S^2 through the analysis of FFTs on the group $SO(3)$, which acts transitively on S^2.

There even exist applications of generalized Fourier transforms for noncompact groups. Chirikjian and Kyatkin (3) present their analysis of transforms on the Euclidean motion group $SE(3)$ in order to describe the configuration space for certain robotic arms. Such transforms may also apply to the configuration space of proteins as they fold into their appropriate forms and hence may provide a convenient means of describing these folded states (27).

1.6 Open Questions

We conclude with a number of open questions and directions for future development in the fields of generalized FFTs and noncommutative harmonic analysis. Many of these derive from papers by Maslen and Rockmore (20; 27).

- Although certain groups present $O(|G| \log |G|)$ or $O(|G| \log^2 |G|)$ FFT

algorithms, there exists no universal $O(|G| \log^c |G|)$ bound on the complexity of generalized FFTs for finite groups. One approach to this problem may involve the determination of FFTs for all the groups in the classification of finite simple groups. In particular, this goal requires better FFTs for finite groups of Lie type and for matrix groups.

- It remains to be seen if decimation-in-frequency algorithms can be generalized to match the level of progress that has been made with decimation-in-time algorithms. These decimation-in-frequency formulations are particularly appealing because their theory more closely reflects the module-theoretic underpinnings of group representation theory.

- Similarly, no noncommutative analogues of the important Rader and chirp-z transforms are currently known. If they exist, such analogues may relate transforms between groups that have no nontrivial group-subgroup relationship.

- FFTs for groups seem to rely mainly on the semisimplicity of the group algebra $\mathbb{C}G$. Because of this result, it seems likely that there exist FFTs for band-limited functions on all semisimple Lie groups.

- Much of the theory of generalized Fourier transforms on noncompact groups such as $SE(n)$ is in its initial stages. Such transforms would require the development of suitable sampling algorithms for these groups as a first step.

- The recursive nature of many of the known FFTs algorithms suggests that there exist effective parallel implementations of these algorithms. Decimation-in-frequency algorithms in particular would seem to admit parallel implementations because of their explicit separation of frequency space.

Chapter 2

Character Graphs and Seminormal Representations

Fast Fourier transforms for a group G frequently depend on a seminormal matrix representation for the group with respect to a chain $1 = G_0 < G_1 < \cdots < G_n = G$ of its subgroups (6). Such seminormal representations are partitioned (rather than merely decomposed as direct sums) when restricted to subgroups in the chain. We present the concept of a character graph for such a chain of subgroups and use it to construct these seminormal matrix representations. Much of this follows from Ram (26), although Okounkov and Vershik (23) use similar techniques to determine seminormal representations for S_n.

2.1 Character Graphs

We first develop the notion of a character graph for a chain of subgroups.

Definition 2.1 Let G be a finite group.

- Let $\hat{G}$ denote an index set for the isomorphism classes of irreducible representations of G.
- For $\mathbb{C}G$-modules M and N, let their *intertwining number* be given by

$$i(M, N) = \langle M, N \rangle = \dim_{\mathbb{C}} \mathrm{Hom}_{\mathbb{C}G}(M, N),$$

  the dimension of the space of $\mathbb{C}G$-module homomorphisms from M to N. ■

We note that, by Schur's Lemma, if M is irreducible, then $i(M, N)$ gives the multiplicity of M in N.

Let H be a subgroup of G, and let M^λ be an irreducible representation of G of type $\lambda \in \hat{G}$. Then its restriction to H decomposes as

$$M^\lambda \downarrow_H^G = \bigoplus_{\mu \in \hat{H}} c_\mu^\lambda M^\mu,$$

where M^μ is an irreducible representation of type $\mu \in \hat{H}$, and c_μ^λ is its multiplicity $i(M^\mu, M^\lambda \downarrow_H^G)$ in M^λ.

Definition 2.2 Let G be a group and $\mathcal{C}$ a chain of subgroups $1 = G_0 < G_1 < \cdots < G_n = G$. The *character graph* $\Gamma = \Gamma(\mathcal{C})$ for this chain is a multigraph, graded by $\mathbb{N}$, such that the vertices at the ith level of Γ correspond to the elements of $\hat{G}_i$, there is a unique vertex $\emptyset$ for G_0, and if $\rho \in \hat{G}_i$ and $\mu \in \hat{G}_{i-1}$, then there exist c_μ^ρ edges between ρ and μ.

In addition, we define several spaces of paths through this graph:

- $\Lambda(\lambda \to \mu)$ is the set of paths from λ to μ,
- $\Lambda(\lambda)$ is the set of paths from $\emptyset$ to λ,
- $\Lambda(\lambda \to s)$ is the set of paths from λ to any $\mu \in \hat{G}_s$,
- $\Lambda(m)$ is the set of paths from $\emptyset$ to any $\mu \in \hat{G}_m$,
- $\Lambda = \Lambda(n)$, where n is the height of the subgroup chain,
- $\Omega(\lambda)$ is the set of pairs (S, T) of paths such that $S, T \in \Lambda(\lambda)$,
- $\Omega(m)$ is the set of pairs (S, T) of paths such that $S, T \in \Lambda(\lambda)$ for some $\lambda \in \hat{G}_m$.

In general, we denote a path $L \in \Lambda$ by $(\lambda^{(0)} \xrightarrow{e_1} \ldots \xrightarrow{e_n} \lambda^{(n)})$. We omit the edge labels on there arrows if the $\lambda^{(i)}$s suffice to determine the path or if they are unimportant for a given application. ■

Example 2.3 Consider the subgroup chain $1 < \mathbb{Z}/2\mathbb{Z} < \mathbb{Z}/6\mathbb{Z}$. As in Section 1.2.1, we denote the irreducible representations of $\mathbb{Z}/N\mathbb{Z}$ as ζ_k for $k = 0, \ldots, N-1$. The character graph Γ for this chain is depicted in Figure 2.1. Some example paths in Γ are $S = (\emptyset \to \zeta_0 \to \zeta_2)$ and $T = (\emptyset \to \zeta_1 \to \zeta_5)$, and the total number of paths in the diagram is $|\Lambda| = 6$. Since exactly one path travels to each irreducible type of $\mathbb{Z}/6\mathbb{Z}$, $|\Omega(2)| = 6$ as well. ■

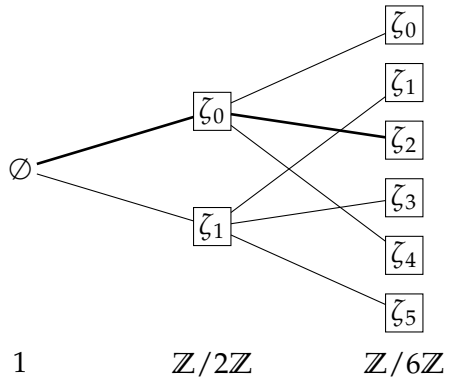


Figure 2.1: Character graph for $1 < \mathbb{Z}/2\mathbb{Z} < \mathbb{Z}/6\mathbb{Z}$. The path $(\emptyset \to \zeta_0 \to \zeta_2)$ is highlighted. Note that the restrictions of these irreducibles are all multiplicity free, so we have no multiple edges.

2.2 Path Algebras

These paths in the character graph lead naturally to a series of algebras over $\mathbb{C}$.

Definition 2.4 Let Γ be a character graph for a subgroup chain $1 = G_0 < G_1 < \cdots < G_n = G$. For $0 \leq m < n$, define the $\mathbb{C}$-algebra P_m with basis E_{ST} for each pair of paths $(S, T) \in \Omega(m)$, and with a multiplication on these basis elements given by

$$E_{ST}E_{PQ} = \delta_{TP}E_{SQ}, \tag{2.1}$$

for all $(S, T), (P, Q) \in \Omega(m)$. Here, δ_{ij} is the Kronecker delta, defined to be 1 if $i = j$ and 0 otherwise.

For each $\lambda \in \hat{G}_m$, define V^λ to be a $\mathbb{C}$-vector space with basis $\{v_L \mid L \in \Lambda(\lambda)\}$. With multiplication defined by $E_{ST}v_L = \delta_{TL}v_S$, V^λ is then a P_m-module. ■

These V^λ modules defined above then form a complete set of representatives for the classes of inequivalent irreducible representations for the path algebra P_m. Furthermore, in the basis specified above for the V^λ spaces, the basis elements of P_m that terminate at λ correspond to the standard basis of the matrix algebra $\mathbb{C}^{d_\lambda \times d_\lambda}$, where d_λ is the degree of the irreducible type λ. Thus,

$$P_m \cong \bigoplus_{\lambda \in \hat{G}_m} \mathbb{C}^{d_\lambda \times d_\lambda}.$$

We also give an inclusion of P_m in P_n for $m < n$. Given paths $T = (\lambda \to \ldots \to \mu)$ and $S = (\mu \to \ldots \to \nu)$, their concatenation is $T * S = (\lambda \to \ldots \to \mu \to \ldots \to \nu)$. Then given an element $E_{PQ} \in P_m$, we define E_{PQ} as an element of P_n by

$$E_{PQ} = \sum_{T \in \Lambda(\lambda \to n)} E_{P*T,Q*T}.$$

Furthermore, this inclusion affords a natural restriction mechanism for P_n-algebras. Suppose that $\lambda \in \hat{G}_m$ and that V^λ is the irreducible P_m-module as constructed above. Then the restriction of V^λ to P_{m-1} decomposes as

$$V^\lambda \downarrow_{P_{m-1}}^{P_m} \cong \bigoplus_{\mu \in \hat{G}_{m-1}} \bigoplus_{e \in \Lambda(\mu \to \lambda)} V^\mu.$$

Thus, for each edge e that connects a G_{m-1} irreducible type μ to λ, V^λ contains an isomorphic copy of V^μ.

Example 2.5 We illustrate some of these concepts with the subgroup chain $S_1 < S_2 < S_3$. As discussed in Chapter 3, the irreducible representation types of S_n correspond to partitions of n. Thus, the character graph is as depicted in Figure 2.2.

The paths in this diagram are

$$\begin{aligned}\Lambda(3) = \{&(\emptyset \to (1) \to (2) \to (3)), \quad (\emptyset \to (1) \to (2) \to (2,1)),\\ &(\emptyset \to (1) \to (1,1) \to (2,1)), \quad (\emptyset \to (1) \to (1,1) \to (1,1,1))\},\end{aligned}$$

which we enumerate as T_1 through T_4. Then a basis for the path algebra P_3 is $\{E_{T_1T_1}, E_{T_2T_2}, E_{T_2T_3}, E_{T_3T_2}, E_{T_3T_3}, E_{T_4T_4}\}$, and we have

$$P_3 \cong \mathbb{C}^{1\times 1} \oplus \mathbb{C}^{2\times 2} \oplus \mathbb{C}^{1\times 1}.$$

The P_3-modules with bases given by $\{v_{T_1}\}$, $\{v_{T_2}, v_{T_3}\}$, and $\{v_{T_4}\}$ are then irreducible representations of P_3.

The paths terminating at the second level of the diagram are

$$\Lambda(2) = \{U_1 = (\emptyset \to (1) \to (2)), U_2 = (\emptyset \to (1) \to (1,1))\},$$

and hence basis elements for P_2 are $\{E_{U_1U_1}, E_{U_2U_2}\}$. These P_2-elements embed in P_3 as

$$E_{U_1U_1} = E_{T_1T_1} + E_{T_2T_2} \quad \text{and} \quad E_{U_2U_2} = E_{T_3T_3} + E_{T_4T_4}.$$

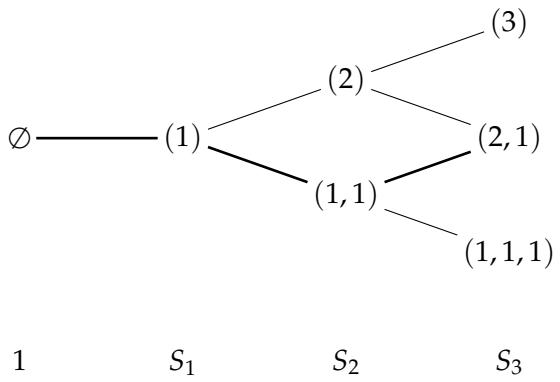


Figure 2.2: Character graph for $1 < S_1 < S_2 < S_3$, with the path $(\emptyset \to (1) \to (1,1) \to (2,1))$ highlighted. As in Figure 2.1, the restrictions here are all multiplicity free.

The P_2-modules with bases given by $\{v_{U_1}\}$ and $\{v_{U_2}\}$ are irreducible representations for P_2. Finally, observing that there are two edges connecting the second level of the diagram to $(2,1)$ in the third level, we have that

$$V^{(2,1)}{\downarrow}_{P_2}^{P_3} = V^{(2)} \oplus V^{(1,1)}.$$

In particular, if $\{v_{T_2}, v_{T_3}\}$ is a basis for $V^{(2,1)}$, then $\{v_{T_2}\}$ and $\{v_{T_3}\}$ are bases for $V^{(2)}$ and $V^{(1,1)}$, respectively. Here, the basis for $V^{(2,1)}$ is partitioned into bases for these components of the restriction. ■

As illustrated in the above example, the basis $\{v_L\}$ for V^λ is partitioned into bases for the V^μs upon restriction. Because of this partitioning, this basis is said to be a *seminormal basis* adapted to the subalgebra chain $P_0 \subset \ldots \subset P_n$. Such bases are also called *Gel'fand-Tsetlin bases* or *adapted bases*, the latter name deriving from the adaptation of the basis to the specified chain of subgroups (20; 26).

2.3 Seminormal Matrix Representations

We now relate these seminormal representations of path algebras to seminormal matrix representations of the $\mathbb{C}G_i$s. In particular, we see that the partitioning behavior of the V^λs under restrictions gives a decomposition of matrix representations for P_{m-1} in P_m into direct sums of matrices, and

we seek a similar direct sum decomposition for representations of elements of $\mathbb{C}G_{i-1}$ in $\mathbb{C}G_i$.

Consequently, we wish to determine an algebra-isomorphism $\Phi : P_n \to \mathbb{C}G$ such that $\Phi(P_i) = \mathbb{C}G_i$ for all $i \leq n$. Wedderburn's Theorem guarantees that such an isomorphism exists, as both P_n and $\mathbb{C}G$ are isomorphic as $\mathbb{C}$-algebras to $\bigoplus_{\lambda \in \hat{G}_n} \mathbb{C}^{d_\lambda \times d_\lambda}$. Such an isomorphism then affords an action of $\alpha \in \mathbb{C}G$ on the V^λs by

$$\alpha v_L = \Phi^{-1}(\alpha) v_L.$$

Given such an isomorphism, we also define $e_{ML} = \Phi(E_{ML})$ for each $(M, L) \in \Omega(n)$.

To determine the properties of such an isomorphism, we focus on sets of central elements in the group algebras. Ram (26) states the following key lemma, which follows from Schur's Lemma and the centrality of the elements under consideration.

Lemma 2.6 (Ram (26: 1.9)) *Let $z_{k,j}$ be a central element in $\mathbb{C}G_k$, let*

$$L = (\lambda^{(0)} \to \ldots \to \lambda^{(n)}) \in \Lambda(n)$$

be a path in Γ, and let $\chi^{\lambda^{(k)}}$ be the irreducible character of G_k indexed by $\lambda^{(k)} \in \hat{G}_k$. Then for any choice of Φ as defined above,

$$z_{k,j} v_L = c_{k,j}(\lambda^{(k)}) v_L, \quad \text{where } c_{k,j}(\lambda^{(k)}) = \frac{\chi^{\lambda^{(k)}}(z_{k,j})}{\chi^{\lambda^{(k)}}(1)}.$$

■

We note that this scalar $c_{k,j}$ depends only on the irreducible type for G_k that L contains, and not on the rest of the path. Since $z_{k,j}$ scales v_L by $c_{k,j}$, $c_{k,j}$ is the eigenvalue of $z_{k,j}$ associated to the irreducible representation type $\lambda^{(k)}$ containing v_L. Having determined that the central group algebra elements thus act as scalars on the seminormal basis vectors, we now assign their eigenvalues as weights to the corresponding paths in the character graph.

Definition 2.7 For each $1 \leq k \leq n$, let $Z_k = \{z_{k,j}\}_{j=1}^{r_k}$ be a collection of central elements of $\mathbb{C}G_k$. For each $\mu \in \hat{G}_k$, let $c_k(\mu) = (c_{k,1}(\mu), \ldots, c_{k,r_k}(\mu))$, so that $c_k(\mu)$ is a list of the eigenvalues of the $z_{k,j}$s corresponding to μ.

Let $L = (\lambda^{(0)} \to \ldots \to \lambda^{(n)}) \in \Lambda$. Then the *weight* of L is

$$\mathrm{wt}(L) = (c_0(\lambda^{(0)}), \ldots, (c_n(\lambda^{(n)})).$$

■

If these path-weights suffice to distinguish paths in the character graph Γ, then they constrain the choice of isomorphism significantly.

Proposition 2.8 (Ram (26: 1.12)) *Assume that* wt *is injective on* Λ*, so that paths in* Γ *are distinguished by their weights. Then for each* $L \in \Lambda$*, the* $\mathbb{C}G$*-element* $e_{LL} = \Phi(E_{LL})$ *is determined uniquely by the* $z_{k,j}$*s and by the constants* $c_{k,j}(\mu)$ *for* $\mu \in \hat{G}_k$*,* $0 \leq k \leq n$*. Furthermore, if* M *and* L *are distinct paths, then* $e_{ML} = \Phi(E_{ML})$ *is determined up to a constant by these elements.*

Proof: Let $L = (\lambda^{(0)}, \ldots, \lambda^{(n)})$ be a path in Γ, and for each $0 \leq k \leq n$ and each $1 \leq j \leq r_k$, let

$$p_{k,j}(\lambda^{(k)}) = \prod_{c_{k,j}(\mu) \neq c_{k,j}(\lambda^{(k)})} \frac{z_{k,j} - c_{k,j}(\mu)}{c_{k,j}(\lambda^{(k)}) - c_{k,j}(\mu)},$$

where we take the product over all $c_{k,j}(\mu)$ with $\mu \in \hat{G}_k$ such that $c_{k,j}(\mu) \neq c_{k,j}(\lambda^{(k)})$. Thus, by Lemma 2.6, if $M = (\mu^{(0)} \to \ldots \to \mu^{(n)})$ is a path in Γ, then for any Φ,

$$\Phi^{-1}(p_{k,j}(\lambda^{(k)}))v_M = \begin{cases} v_M & \text{if } c_{k,j}(\lambda^{(k)}) = c_{k,j}(\mu^{(k)}), \\ 0 & \text{otherwise.} \end{cases}$$

In essence, these $p_{k,j}(\lambda^{(k)})$ elements act as the identity only on the vectors corresponding to paths that pass through nodes at level k with weight $c_{k,j}(\lambda^{(k)})$. Thus, their product, which we define to be

$$e_{LL} = \prod_{k,j} p_{k,j}(\lambda^{(k)}),$$

is the identity only on all paths with the same weight as L. We show that this e_{LL} element coincides with $\Phi(E_{LL})$. By the injectivity of wt, we have that

$$\Phi^{-1}(e_{LL})v_M = \delta_{\mathrm{wt}(L)\,\mathrm{wt}(M)}v_M = \delta_{LM}v_M = \delta_{LM}v_L = E_{LL}v_M.$$

Hence, by the injectivity of Φ, e_{LL} is unique.

Let M and L be distinct paths, and let $a \in \mathbb{C}G$ be such that $e_{MM}ae_{LL} \neq 0$. Then e_{ML} must equal a constant times the element $e_{MM}ae_{LL} \in \mathbb{C}G$, so because e_{LL} and e_{MM} are uniquely determined, so is e_{ML}, up to this choice of constant. ∎

Thus, while we still have some freedom in the choice of the seminormal matrix representations of G, they are constrained entirely for the idempotents of $\mathbb{C}G$ and up to a constant for the remaining elements of $\mathbb{C}G$. In fact,

there are further restrictions on these constants that result from the multiplication structure on P_n: if Φ and Φ' are two isomorphisms from P_n to $\mathbb{C}G$ such that $\Phi(E_{ML}) = e_{ML}$ and $\Phi'(E_{ML}) = e'_{ML}$, then $e_{ML} = \kappa_{ML} e'_{ML}$, and these κ_{ML} constants must satisfy

$$\kappa_{ML}\kappa_{LM} = 1 \quad \text{and} \quad \kappa_{ML}\kappa_{LN} = \kappa_{MN}$$

for all $M, L, N \in \Lambda$ as a direct consequence of the multiplication given by Equation (2.1).

2.4 Applications to MC-Groups

We note that for each group G_i, its centrally primitive idempotents distinguish among the irreducible representations of G_i, and hence also distinguish among the vertices at level i in the character graph Γ. Other bases for the center of $\mathbb{C}G_i$, such as the set of all conjugacy class sums in G_i, then provide alternat choices for the set Z_k.

In the case in which the restrictions of irreducible representations of G_i yield multiplicity-free decompositions into irreducibles of G_{i-1}, the path weights do suffice to distinguish paths, as each path is uniquely determined by its list of the eigenvalues of the $z_{k,j}$s at each of its vertices. Fortunately, several classes of groups exhibit such chains of subgroups, including many of the Weyl groups. In fact, the branchings for the chains

$$\begin{aligned} &S_1 < S_2 < \cdots < S_n, \\ &WB_1 < WB_3 < \cdots < WB_n, \\ &WB_2 < WB_3 < WF_4, \\ &WD_5 < WE_6 < WE_7 \end{aligned}$$

are all multiplicity-free. There also exist such chains for supersolvable groups (5).

Definition 2.9 Let G be a group which exhibits a chain of subgroups $G_0 < G_1 < \cdots < G_n = G$, such that the restriction branching rules at each subgroup in the chain are multiplicity-free. Then G is said to have a *multiplicity-free character graph* and is called an *MC-group*. ■

To conclude, the path-algebraic constructions presented in this chapter give an explicit construction of seminormal matrix representations for an

MC-group G adapted to the chain of its subgroups that exhibits multiplicity-free restrictions. As mentioned above, these seminormal matrix representations are particularly useful because their blocks decompose into matrix direct sums upon restriction to subgroups. Thus, this path-algebraic construction also confers an indexing of the matrix coefficients by pairs of paths through the character graph for the subgroup chain. We develop these seminormal matrix representations for the MC-group S_n in the next chapter.

Chapter 3

Representation Theory of the Symmetric Group

Having developed a general framework for the construction of seminormal matrix representations for a group adapted to a particular chain of subgroups, we now apply it to the symmetric group S_n with the subgroup chain $\mathcal{S}$ given by $S_1 < S_2 < \ldots < S_n$. The relation of the irreducible representations of S_n to partitions of n leads to a compelling combinatorial description of the representation theory for S_n. In conjunction with the framework from Chapter 2, these combinatorics afford an intuitive construction of the seminormal matrix representations for S_n that we require for FFT algorithms.

3.1 Constructions of Irreducible Representations

Before discussing the application of path-algebraic group representations to the symmetric group, we give an overview of the classical construction of the irreducible representations of S_n. Much of the classical work on the representations of the symmetric group was performed by Alfred Young in the late 1920s (28). James and Kerber (14) modernizes Young's approach significantly and remains a canonical reference on this classical characterization of these representations. The following material draws on their analysis.

At the center of Young's formulation are several combinatorial objects, the definitions of which we take largely from Sagan (28). While we use some of these definitions only later in this chapter, we elect to consolidate them here for convenience. Throughout, let n be a positive integer.

Definition 3.1 A *composition* λ of n is a sequence $\lambda = (\lambda_1, \lambda_2, \dots)$ of non-negative integers such that $\sum_i \lambda_i = n$. The λ_is are called the *parts* of λ. We typically truncate compositions at their last positive entry.

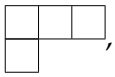
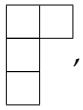
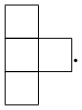
A *partition* of n is a composition λ such that its parts are weakly decreasing, that is, $\lambda_{i+1} \leq \lambda_i$ for all $i = 1, 2, \dots$. If λ is a partition of n, we write $\lambda \vdash n$. ■

Example 3.2 Let $n = 3$. Then $(2,1)$, $(0,1,2)$, and $(0,1,0,1,1)$ are all compositions of n, but only $(2,1)$ is a partition. The partitions of n are given by (3), $(2,1)$, and $(1,1,1)$. ■

Definition 3.3 If $\lambda = (\lambda_1, \lambda_2, \dots, \lambda_k)$ is a composition of n, then the *Ferrers diagram* of λ is an array of n boxes in k left-aligned rows, with row i having λ_i boxes. If $\lambda \vdash n$, then both it and the corresponding diagram are *proper*.

If t is the Ferrers diagram of a composition λ, and b is a box in the ith row and jth column of t, then the *content* of b is $\mathrm{ct}(b) = j - i$. ■

Example 3.4 Let $n = 4$, and let $\lambda = (3,1)$, $\mu = (2,1,1)$ and $\nu = (1,2,1)$. Then the Ferrers diagrams of these compositions are

$\lambda =$  , $\mu =$  , and $\nu =$  .

Reading left to right and top to bottom, their boxes have content values $(0,1,2;-1)$, $(0,1;-1;-2)$, and $(0;-1,0;-2)$, respectively. ■

Definition 3.5 A *Young tableau of shape* λ is an array t obtained by placing the numbers $1, 2, \dots, n$ in the boxes of the Ferrers diagram for the partition λ. The *shape* of t, denoted $\mathrm{sh}\, t$, is the partition λ.

Let $t_{i,j}$ denote the entry in the box in row i and column j of t, and let $t[k]$ denote the box of t that contains k.

A Young tableau t is *standard* if the entries in its rows and columns are strictly increasing. Let T^λ denote the set of all tableaux of shape λ, and let T^λ_s denote the set of all such standard tableaux. Define $f^\lambda = |T^\lambda_s|$. ■

Example 3.6 Let $n = 3$, and let $\lambda = (2,1)$. Then the Ferrers diagram of λ is

and we can enumerate the elements of T^λ as

$$t_1 = \begin{array}{|c|c|}\hline 1 & 2 \\ \hline 3 \\ \cline{1-1}\end{array},\ t_2 = \begin{array}{|c|c|}\hline 1 & 3 \\ \hline 2 \\ \cline{1-1}\end{array},\ t_3 = \begin{array}{|c|c|}\hline 2 & 1 \\ \hline 3 \\ \cline{1-1}\end{array},\ t_4 = \begin{array}{|c|c|}\hline 2 & 3 \\ \hline 1 \\ \cline{1-1}\end{array},\ t_5 = \begin{array}{|c|c|}\hline 1 & 3 \\ \hline 2 \\ \cline{1-1}\end{array},\ t_6 = \begin{array}{|c|c|}\hline 2 & 3 \\ \hline 1 \\ \cline{1-1}\end{array}.$$

Of these, only

$$\begin{array}{|c|c|}\hline 1 & 2 \\ \hline 3 \\ \cline{1-1}\end{array} \quad \text{and} \quad \begin{array}{|c|c|}\hline 1 & 3 \\ \hline 2 \\ \cline{1-1}\end{array}$$

are standard tableaux.

The other partitions of $n = 3$ are (3) and $(1,1,1)$, which have the standard tableaux

$$\begin{array}{|c|c|c|}\hline 1 & 2 & 3 \\ \hline\end{array} \quad \text{and} \quad \begin{array}{|c|}\hline 1 \\ \hline 2 \\ \hline 3 \\ \hline\end{array}$$

respectively, so we have $f^{(3)} = 1$, $f^{(2,1)} = 2$, and $f^{(1,1,1)} = 1$.

In general, if λ is a composition of n, there are $n!$ distinct λ-tableaux, since each permutation of $\{1, 2, \ldots, n\}$ uniquely determines a tableau. ■

Definition 3.7 If $\pi \in S_n$ and $t = (t_{i,j})$ is a tableau of shape λ, then we define πt to be the tableau with ijth entry $\pi(t_{i,j})$. This gives an action of S_n on T^λ that extends by linearity to make the $\mathbb{C}$-vector space $\mathbb{C}T^\lambda$ a $\mathbb{C}S_n$-module. ■

Example 3.8 Continuing Example 3.6, we see that, for example, if $\pi = (2\,3)$,

$$\pi \begin{array}{|c|c|}\hline 1 & 2 \\ \hline 3 \\ \cline{1-1}\end{array} = \begin{array}{|c|c|}\hline 1 & 3 \\ \hline 2 \\ \cline{1-1}\end{array}$$

since π exchanges 2 and 3. Extending by linearity, we see that this gives $\mathbb{C}T^\lambda$ a $\mathbb{C}S_n$-module structure, so that, for example,

$$(1 - (1\,2\,3)) \left(3 \begin{array}{|c|c|}\hline 1 & 2 \\ \hline 3 \\ \cline{1-1}\end{array} - \begin{array}{|c|c|}\hline 1 & 3 \\ \hline 2 \\ \cline{1-1}\end{array} \right) = 3 \begin{array}{|c|c|}\hline 1 & 2 \\ \hline 3 \\ \cline{1-1}\end{array} - 3 \begin{array}{|c|c|}\hline 2 & 3 \\ \hline 1 \\ \cline{1-1}\end{array} - \begin{array}{|c|c|}\hline 1 & 3 \\ \hline 2 \\ \cline{1-1}\end{array} + \begin{array}{|c|c|}\hline 2 & 1 \\ \hline 3 \\ \cline{1-1}\end{array}.$$ ■

Definition 3.9 Let $\lambda = (\lambda_1, \lambda_2, \ldots, \lambda_k) \vdash n$. Then the corresponding *Young subgroup* of S_n is

$$S_\lambda = S_{\{1,2,\ldots,\lambda_1\}} \times S_{\{\lambda_1+1,\ldots,\lambda_2\}} \times \cdots \times S_{\{n-\lambda_k+1,\ldots,n\}}.$$ ■

We note that, for a general $\lambda = (\lambda_1, \lambda_2, \ldots, \lambda_k) \vdash n$, we have the group isomorphism

$$S_\lambda \cong S_{\lambda_1} \times S_{\lambda_2} \times \cdots \times S_{\lambda_k}.$$

Example 3.10 Let $n = 9$ and let $\lambda = (3,3,2,1) \vdash n$. Then the Young subgroup S_λ of S_n is $S_{\{1,2,3\}} \times S_{\{4,5,6\}} \times S_{\{7,8\}} \times S_{\{9\}}$ and is isomorphic to $S_3 \times S_3 \times S_2 \times S_1$. ■

With this combinatorial machinery in place, we describe (without proof) Young's construction of the irreducible representations of S_n. Let α be a partition of n, and let α' be the complementary partition, such that the ith row of the Young diagram associated to α' contains the number of boxes in the ith column of α. We can construct such a partition graphically by taking the transpose of the Young diagram associated with α. Let ι denote the trivial representation of the Young subgroup S_α, and let ϵ denote the alternating representation of $S_{\alpha'}$. Inducing these representations to S_n gives $\iota\uparrow_{S_\alpha}^{S_n}$ and $\epsilon\uparrow_{S_{\alpha'}}^{S_n}$. We then have

$$i(\iota\uparrow_{S_\alpha}^{S_n}, \epsilon\uparrow_{S_{\alpha'}}^{S_n}) = 1,$$

where i is the intertwining number defined in Definition 2.1. Since this quantity equals one, these two induced representations share a single copy of an irreducible representation of S_n, which we represent by $[\alpha]$. These representations in fact determine all such irreducibles up to isomorphism:

Theorem 3.11 (James and Kerber (14: Thm 2.1.11)) *$\{[\alpha] \mid \alpha \vdash n\}$ is the complete set of equivalence classes of ordinary irreducible representations of S_n.* ■

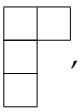
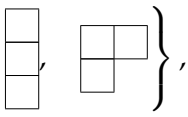
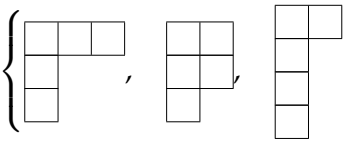
Thus, if $\alpha, \beta \vdash n$ are distinct partitions of n, then $[\alpha]$ and $[\beta]$ are nonisomorphic representations of S_n. The *Specht modules*, discussed extensively in Sagan (28), provide a different complete set of inequivalent irreducible representations for S_n. The Specht module isomorphic to $[\lambda]$ is denoted S^λ. Sagan (28) specifies the effects of the restriction of these modules to S_{n-1} or their induction to S_{n+1}. Before we state these results, we define two families of partitions associated to $\lambda \vdash n$.

Definition 3.12 If $\lambda \vdash n$, then denote by λ^- the set of all partitions μ of $n-1$ such that μ has a part that, when incremented, changes μ to λ. Similarly, let λ^+ denote the set of all partitions μ of $n+1$ such that μ has a part that, when decremented, changes μ to λ. ■

We can also formulate these notions in terms of Ferrers diagrams: if $\lambda \vdash n$, then the set λ^- consists of those proper shapes μ corresponding to the diagrams formed by removing a box from the diagram of shape λ, and the set λ^+ likewise consists of those proper shapes μ corresponding to the diagrams formed by adding a box to the diagram of shape λ.

Example 3.13 Consider $\lambda = (2,1,1) \vdash 4$. Then $\lambda^- = \{(1,1,1),(2,1)\}$, and $\lambda^+ = \{(3,1,1),(2,2,1),(2,1,1,1)\}$. Represented as Ferrers diagrams, these

partitions are

$\lambda =$  , $\lambda^- =$  , $\lambda^+ =$  ■

Theorem 3.14 (Sagan (28: Thm 2.8.3)) *If $\lambda \vdash n$, then*

$$S^\lambda{\downarrow}_{S_{n-1}}^{S_n} = \bigoplus_{\mu \in \lambda^-} S^\mu \quad \textit{and} \quad S^\lambda{\uparrow}_{S_n}^{S_{n+1}} = \bigoplus_{\mu \in \lambda^+} S^\mu. \qquad \blacksquare$$

This characterization of the Specht module inductions and restrictions explicitly shows the multiplicity-free branching that the subgroup chain $S_1 < S_2 < \cdots < S_n$ exhibits.

3.2 Reformulation of Path-Algebraic Techniques

Using the path-algebraic techniques introduced in Chapter 2, we determine seminormal bases for these irreducible representations of S_n. We first relate the paths in the character graph $\Gamma = \Gamma(\mathcal{S})$ to standard tableaux. From above, the irreducible types for S_k are in bijective correspondence with the partitions of k, so we use these partitions as the vertices at level k of Γ (as we alluded to in Example 2.5).

Proposition 3.15 *Let n be a positive integer, and let $\lambda \vdash n$. There is a bijection from T_s^λ to $\Lambda(\lambda)$ given as follows: Given a standard tableau t with $\operatorname{sh} t = \lambda$, let $\lambda^{(0)} = \emptyset$, and for $1 \leq i \leq n$ let $\lambda^{(i)}$ be the shape of the boxes in t that contain the numbers $1, \ldots, i$. Then*

$$t \mapsto (\lambda^{(0)} \to \lambda^{(1)} \to \ldots \to \lambda^{(n)}).$$

Proof: Since this chain of subgroups affords multiplicity-free restrictions, there is at most one edge between vertices at adjacent levels of the diagram, and so an element $T \in \Lambda(\lambda)$ is specified entirely by the list of partitions $(\mu^{(0)}, \mu^{(1)}, \ldots, \mu^{(n)})$ that constitutes the vertices of T.

We now show this map takes a standard tableau t of shape λ to a path in $\Lambda(\lambda)$. First, we show that $\lambda^{(k)} \vdash k$. Consider the position of k in t. Since t is standard, the boxes above and to the left of the box with k must contain

integers less than k. This property holds for each $j < k$, too, so considering only these boxes containing $j \leq k$ must give a proper shape. Thus, each $\lambda^{(k)}$ is a partition of k for each $0 \leq k \leq n$.

Now consider $\lambda^{(k)}$ and $\lambda^{(k-1)}$ for $1 \leq k \leq n$. Since $\lambda^{(k)} \vdash k$ and $\lambda^{(k-1)} \vdash (k-1)$, and since their shapes differ only by a box, by Theorem 3.14 $S^{\lambda^{(k)}}\downarrow^{S_k}_{S_{k-1}}$ contains an isomorphic copy of $S^{\lambda^{(k-1)}}$. Thus, there is an edge connecting $\lambda^{(k)}$ and $\lambda^{(k-1)}$ in Γ. Lastly, $\lambda^{(k)} = \lambda$ since $\operatorname{sh} t = \lambda$. Hence, t maps to an element of $\Lambda(\lambda)$.

Suppose $t, t' \in T^\lambda_s$ are distinct. Then there exists some minimal k for which $t[k] \neq t'[k]$, so their corresponding lists of partitions differ at k. Thus, this map is injective.

Finally, suppose $T \in \Lambda(\lambda)$, such that $T = (\mu^{(0)}, \mu^{(1)}, \ldots, \mu^{(n)})$. By Theorem 3.14, $\mu^{(k)}$ is obtained from $\mu^{(k-1)}$ by adding a box to the shape $\mu^{(k-1)}$ such that the shape remains proper. Thus, we construct a standard tableau t from T by placing k in the box added when moving from $\mu^{(k-1)}$ to $\mu^{(k)}$. Then t maps back to T, so this map is surjective. ∎

By this proposition, we may identify paths through the character graph Γ terminating in λ with standard tableaux of shape λ. Since $\lambda^{(0)} = \emptyset$ in all cases, we typically drop it from the list of vertices of Γ. We give some examples of this identification.

Example 3.16 Consider the standard tableau

$$t = \begin{array}{|c|c|c|}\hline 1 & 3 & 4 \\ \hline 2 & 5 \\ \cline{1-2}\end{array}$$

of shape $(3, 2)$. By Proposition 3.15, t corresponds to the path

$$\emptyset \to \ydiagram{1} \to \ydiagram{1,1} \to \ydiagram{2,1} \to \ydiagram{3,1} \to \ydiagram{3,2}.$$

As another example, the four paths P_1 through P_4 through the character graph of Example 2.5 correspond to the four standard tableaux of S_3 presented in Example 3.6.

Finally, recall from Section 2.2 that, for $\lambda \vdash n$, the set $\{v_L \mid L \in \Lambda(\lambda)\}$ forms a seminormal basis for the irreducible P_n-module V^λ. With this identification, we can index these seminormal basis vectors with the standard tableau of shape λ. Since these V^λ are also irreducibles for $\mathbb{C}S_n$ by any $\mathbb{C}$-algebra isomorphism Φ, this gives a different proof of the following theorem. ∎

Theorem 3.17 (Sagan (28: 2.5.2)) *If λ is a partition for n and S^λ the corresponding irreducible representation for S_n, then the standard tableaux of shape λ correspond to a basis for S^λ, and* $\dim_{\mathbb{C}} S^\lambda = f^\lambda$. ■

Again following Ram (26) for much of the remaining material in this section, we define the following elements of $\mathbb{C}S_n$:

Definition 3.18 Define $s_1 = z_1 = m_1 = 0$. For $2 \leq k \leq n$, define $s_k = (k-1\ k)$,

$$z_k = \sum_{k=1}^{n} \sum_{j=1}^{k-1} (j\,k),$$

and $m_k = z_k - z_{k-1}$. ■

We note some significant properties of these elements of $\mathbb{C}S_n$:

- The s_ks are the simple transpositions $(1\,2), (2\,3), \ldots$ and generate S_n.
- The z_ks are the class sums of transpositions of S_k and hence are central in $\mathbb{C}S_k$.
- The m_ks are the differences of these class sums and are used extensively in the construction of the seminormal matrix representations below. Jucys (15) and Murphy (21; 22) independently identified these elements in their constructions of Young's seminormal representations of S_n, and they are now called *Jucys-Murphy elements* in their honor (23; 26). The first few such elements for S_n are $(1\,2)$, $(1\,3) + (2\,3)$, and $(1\,4) + (2\,4) + (3\,4)$. In general, we have

$$m_k = \sum_{j=1}^{k-1} (j\,k).$$

The next key result concerns the eigenvalues of z_k acting on the irreducible representations of S_k, where our action is taken to be that specified in Lemma 2.6.

Proposition 3.19 (Ram (26: 3.8)) *Let $k \leq n$, and let $\mu \vdash k$. Let χ^μ be the character of the irreducible representation S^μ. Then*

$$\frac{\chi^\mu(z_k)}{\chi^\mu(1)} = \sum_{b \in \mu} \mathrm{ct}(b),$$

where the sum is over all the boxes in the shape μ. ■

This result then specifies the weights $c_k(\mu)$ for any choice of isomorphism Φ from the path algebra P_n on Γ to $\mathbb{C}S_n$. Furthermore, this result allows us to distinguish paths in Γ by the weights assigned by these z_ks.

Definition 3.20 Recalling the identification of paths in Γ with standard tableaux, we define the *weight* of a standard tableau $L = (\lambda^{(1)}, \ldots, \lambda^{(n)})$ for S_n to be

$$\mathrm{wt}(L) = (c_1(\lambda^{(1)}), \ldots, c_n(\lambda^{(n)})).$$

We define the *differential weight* of L to be

$$\widetilde{\mathrm{wt}}(L) = (c_1(\lambda^{(1)}), c_2(\lambda^{(2)}) - c_1(\lambda^{(1)}), \ldots, c_n(\lambda^{(n)}) - c_{n-1}(\lambda^{(n-1)})). \quad \blacksquare$$

We note that the weight and the differential weight of a tableau determine each other uniquely. By Proposition 3.19, $c_k(\lambda^{(k)}) - c_{k-1}(\lambda^{(k-1)}) = \mathrm{ct}(L[k])$, so that

$$\widetilde{\mathrm{wt}}(L) = (\mathrm{ct}(L[1]), \mathrm{ct}(L[2]), \ldots, \mathrm{ct}(L[n])).$$

Furthermore, for any isomorphism $\Phi : P_n \to \mathbb{C}S_n$, these content values determine the action of z_k and m_k on the seminormal basis vectors v_L, where L is a standard tableau. In particular, if $L = (\lambda^{(1)}, \ldots, \lambda^{(n)})$, then

$$z_k v_L = c_k(\lambda^{(k)}) v_L = \left(\sum_{b \in \lambda^{(k)}} \mathrm{ct}(b) \right) v_L \tag{3.1}$$

and

$$m_k v_L = (z_k - z_{k-1}) v_L = c_k(\lambda^{(k)}) v_L - c_{k-1}(\lambda^{(k-1)}) v_L = \mathrm{ct}(L[k]) v_L. \tag{3.2}$$

Let $e_{ML} \in \mathbb{C}G$ be as defined in Proposition 2.8. Then, for N a standard tableau of size n, we have

$$m_k e_{ML} v_N = m_k \delta_{LN} v_M = \mathrm{ct}(M[k]) \delta_{LN} v_M = \mathrm{ct}(M[k]) e_{ML} v_N, \tag{3.3}$$

$$\begin{aligned} e_{ML} m_k v_N &= e_{ML} \, \mathrm{ct}(N[k]) v_N = \mathrm{ct}(N[k]) \delta_{LN} v_M \\ &= \mathrm{ct}(L[k]) \delta_{LN} v_M = \mathrm{ct}(L[k]) e_{ML} v_N, \end{aligned} \tag{3.4}$$

where $\mathrm{ct}(N[k])\delta_{LN} = \mathrm{ct}(L[k])\delta_{LN}$. Thus, for m_k acting from the left, e_{ML} is an eigenvector with eigenvalue $\mathrm{ct}(M[k])$, while for m_k acting from the right, e_{ML} is an eigenvector with eigenvalue $\mathrm{ct}(L[k])$.

Proposition 3.21 (Ram (26: 3.9)) *Each standard tableau $L = (\lambda^{(1)}, \ldots, \lambda^{(k)})$ is distinguished by its weight.*

Proof: Two boxes b and b' have the same content only if they lie on the same diagonal. Hence, if $\lambda^{(k)}$ is a partition, then each of the boxes b that can be added to $\lambda^{(k)}$ to obtain a new partition in $\lambda^{(k)+}$ has a different content $\mathrm{ct}(b)$. Therefore, the shape $\lambda^{(k+1)}$ in L is completely determined by $\mathrm{ct}(b)$ and $\lambda^{(k)}$, so L is completely determinted by $\widetilde{\mathrm{wt}}(L)$, and thus by $\mathrm{wt}(L)$. ∎

Corollary 3.21.1 *The space $\mathbb{C}e_{ML}$ is distinguished by the differential weights $\widetilde{\mathrm{wt}}(M)$ and $\widetilde{\mathrm{wt}}(L)$, which correspond to the left and right eigenvalues of $\{m_k\}_{k=1}^n$ on e_{ML}, respectively.*

Proof: By Equations (3.3) and (3.4), the left and right eigenvalues of $\{m_k\}_{k=1}^n$ on $\mathbb{C}e_{ML}$ give $\widetilde{\mathrm{wt}}(M)$ and $\widetilde{\mathrm{wt}}(L)$, which by Proposition 3.21 suffice to distinguish M and L and hence to identify $\mathbb{C}e_{ML}$. ∎

By this result, the conclusions of Proposition 2.8 apply, and the isomorphism $P_n \to \mathbb{C}S_n$ is almost uniquely determined. In the next section, we specify such an isomorphism and determine matrix representations adapted to these for the generators s_k of S_n.

3.3 Seminormal Matrix Representations

While we have identified the standard tableaux of shape $\lambda \vdash n$ as corresponding to a seminormal basis for the irreducible of type λ, we lack an ordering of this basis. Fortunately, there exists one on the standard tableaux based on the relative positions of the letters in the boxes. Since this order derives from the placement of the letters in descending order, it is called the *last-letter order* on these tableaux (14).

The last-letter order is specified as follows. Suppose that s and t are two standard tableaux of shape $\lambda \vdash n$, and let $k \leq n$ be the last letter on which they differ. Then $s < t$ iff the row index of $s[k]$ is less than that of $t[k]$.

Because two standard tableaux always each contain the numbers 1 to n and because there exists some box on which distinct tableaux differ, this procedure introduces a total order onto the tableaux of each shape. Thus, the last-letter order gives a natural order to the seminormal basis vectors of each irreducible representation of S_n. We let $\{t_i^\lambda\}_{i=1}^{f^\lambda}$ denote the standard tableaux of shape λ, ordered so that $t_i^\lambda < t_j^\lambda$ iff $i < j$.

Example 3.22 As an example, the five standard tableaux of shape $(3,2)$ are ordered as follows:

$$\begin{array}{|c|c|c|}\hline 1&3&5\\\hline 2&4\\\cline{1-2}\end{array} < \begin{array}{|c|c|c|}\hline 1&2&5\\\hline 3&4\\\cline{1-2}\end{array} < \begin{array}{|c|c|c|}\hline 1&3&4\\\hline 2&5\\\cline{1-2}\end{array} < \begin{array}{|c|c|c|}\hline 1&2&4\\\hline 3&5\\\cline{1-2}\end{array} < \begin{array}{|c|c|c|}\hline 1&2&3\\\hline 4&5\\\cline{1-2}\end{array}. \tag{3.5}$$

In particular, the tableaux are ordered as they are constructed last letter first: the "greatest" three tableaux have 5 in a lower row than the "least" two tableaux, while, among these three greater tableaux, the tableau with its 4 in the second row is greater than the two that place the 4 in the first row, and so on. ■

We reformulate this last-letter order in terms of the content of boxes in the tableaux (and hence in terms of their differential weights):

Proposition 3.23 *Let $\lambda \vdash n$, and let $t, t' \in T_s^\lambda$. Let $\widetilde{\mathrm{wt}}_r(t)$ be the reverse of $\widetilde{\mathrm{wt}}(t)$, so that $\widetilde{\mathrm{wt}}_r(t) = (\mathrm{ct}(t[n]), \mathrm{ct}(t[n-1]), \ldots, \mathrm{ct}(t[1]))$. Then $t < t'$ in the last-letter order iff $\widetilde{\mathrm{wt}}_r(t) > \widetilde{\mathrm{wt}}_r(t')$ in the lexicographic order.*

Proof: This result follows from translating the description of the last-letter sequence given above into one that uses box content.

Suppose $t < t'$, and let k be the largest integer such that $t[k] \neq t'[k]$. Since $t < t'$, k occurs in a higher row in t than it does in t'. Let s and s' be the Young tableaux obtained from t and t' by removing boxes $k+1$ through n. These boxes occur in identical positions in t and t', so s and s' have the same shape, μ. Furthermore, $t[k]$ and $t'[k]$ must be outer corners of μ, as they are the last boxes to be added to form s and s'. Thus, since the row index of $t[k]$ is less than that of $t'[k]$, the column index of $t[k]$ is larger than that of $t'[k]$, and so $\mathrm{ct}(t[k]) > \mathrm{ct}(t'[k])$. Since $\widetilde{\mathrm{wt}}_r(t)$ and $\widetilde{\mathrm{wt}}_r(t')$ agree up to the $t[k]$ and $t'[k]$ entries, $\widetilde{\mathrm{wt}}_r(t) > \widetilde{\mathrm{wt}}_r(t')$ in the lexicographic order.

Conversely, if $\widetilde{\mathrm{wt}}_r(t) > \widetilde{\mathrm{wt}}_r(t')$, let k be the entry in the box corresponding to the first place at which these lists disagree. Then $\mathrm{ct}(t[k]) > \mathrm{ct}(t'[k])$, so by the same outer-corner argument, the row index of $t[k]$ is less than that of $t'[k]$, and $t < t'$ in the last-letter order. ■

Example 3.24 We illustrate this correspondence with the standard tableaux of Example 3.22. The reverses of the differential weights for these tableaux are

$$\widetilde{\mathrm{wt}}_r\left(\begin{array}{|c|c|c|}\hline 1&3&5\\\hline 2&4\\\cline{1-2}\end{array}\right) = (2,0,1,-1,0),$$

$$\widetilde{\mathrm{wt}}_r\left(\begin{array}{|c|c|c|}\hline 1&2&5\\\hline 3&4\\\cline{1-2}\end{array}\right) = (2,0,-1,1,0),$$

$$\widetilde{\mathrm{wt}}_r\left(\begin{array}{|c|c|c|}\hline 1&3&4\\\hline 2&5\\\cline{1-2}\end{array}\right) = (0,2,1,-1,0),$$

$$\widetilde{\mathrm{wt}}_r\left(\begin{array}{|c|c|c|}\hline 1&2&4\\\hline 3&5\\\cline{1-2}\end{array}\right) = (0,2,-1,1,0),$$

$$\widetilde{\mathrm{wt}}_r\left(\begin{array}{|c|c|c|}\hline 1&2&3\\\hline 4&5\\\cline{1-2}\end{array}\right) = (0,-1,2,1,0),$$

which are easily seen to be in decreasing lexicographic order. ∎

We now specify the seminormal matrix representations for the generators s_k of S_n. For a choice of algebra isomorphism $\Phi : P_n \to \mathbb{C}S_n$, and for standard tableaux L, M of the same shape, let $(s_k)_{ML}$ denote the coefficient of v_M in $s_k v_L = \Phi^{-1}(s_k)v_L$. Since the s_ks generate $\mathbb{C}S_n$, the choice of these coefficients completely determines Φ, although not all such choices determine Φ as an algebra isomorphism. One choice of these coefficients is given below.

Theorem 3.25 (Ram (26: 3.26)) *Let L be a standard tableau of shape $\lambda \vdash n$, and for $2 \le k \le n$ define*

$$c(s_k, L) = \frac{1}{\mathrm{ct}(L[k]) - \mathrm{ct}(L[k-1])}.$$

We recall that the s_ks act on T_λ by interchanging boxes k and $k-1$, and note that even if L is standard, $s_k L$ may not be. Defining

$$(s_k)_{ML} = \begin{cases} c(s_k, L), & M = L, \\ 1 + c(s_k, L), & M = s_k L, \\ 0, & \textit{otherwise} \end{cases}$$

for each k and each pair of $L, M \in T_s^\lambda$ gives an action of $\mathbb{C}S_n$ on V^λ consistent with the choice of some isomorphism $\Phi : P_n \to \mathbb{C}S_n$.

Proof: For the sake of simplicity, we show this for $k = n$; the case for $k < n$ is similar. We show this result in stages: we first show $(s_n)_{ML} = 0$ unless $M = L$ or $M = s_n L$, and then we specify values for $(s_n)_{LL}$ and $(s_n)_{s_n L, L}$ (should $s_n L$ be standard).

Let $L = (\lambda^{(1)}, \ldots, \lambda^{(n)})$ be a standard tableau of shape $\lambda \vdash n$. As in Proposition 2.8, define for each $1 \le k \le n$

$$p_k(\lambda^{(k)}) = \prod_{c_k(\lambda^{(k)}) \neq c_k(\mu)} \frac{z_k - c_k(\mu)}{c_k(\lambda^{(k)}) - c_k(\mu)},$$

and define

$$p_L^* = \prod_{k=1}^{n-2} p_k(\lambda^{(k)}).$$

Suppose $M = (\mu^{(1)}, \ldots, \mu^{(n)}) \in T_s^\lambda$. Then

$$p_L^* v_M = \begin{cases} v_M, & \mu^{(k)} = \lambda^{(k)} \text{for } 1 \leq k \leq n-2, \\ 0, & \text{otherwise.} \end{cases}$$

Since $\lambda^{(n-2)}$ and $\lambda^{(n)}$ differ by only two boxes, we have only two choices for the order in which to add them to $\lambda^{(n-2)}$ to produce $\lambda^{(n)}$. Hence, there are only two tableaux M, L and $s_n L$, such that $\mu^{(k)} = \lambda^{(k)}$ for $1 \leq k \leq n-2$. Of these, L is standard, but $s_n L$ may not be.

Since each p_k term in p_L^* is an element of $\mathbb{C}S_{n-2}$, p_L^* commutes with s_n. Thus, we have that

$$s_n v_L = s_n p_L^* v_L = p_L^* s_n v_L = p_L^* \sum_{T \in T_\lambda^s} (s_n)_{TL} v_T = \sum_{T \in T_\lambda^s} (s_n)_{TL} p_L^* v_T.$$

Thus, if $s_n L$ is standard,

$$s_n v_L = (s_n)_{LL} v_L + (s_n)_{s_n L, L} v_{s_n L},$$

and if not, $s_n v_L = (s_n)_{LL} v_L$. In this case, we define $(s_n)_{s_n L, L} = 0$.

We now specify a value for $(s_n)_{LL}$. It is well-known (Murphy (21: (2.3)), Ram (26)) and in any case straightforward to verify that

$$s_n m_{n-1} = m_n s_n - 1.$$

Rewriting this as $1 = m_n s_n - s_n m_{n-1}$ and applying this equation to v_L with $M = s_n L$ yields

$$\begin{aligned} v_L &= (m_n s_n - s_n m_{n-1}) v_L \\ &= m_n((s_n)_{LL} v_L + (s_n)_{ML} v_M) - \mathrm{ct}(L[n-1]) s_n v_L \\ &= (\mathrm{ct}(L[n]) - \mathrm{ct}(L[n-1]))(s_n)_{LL} v_L, \end{aligned}$$

so that

$$(s_n)_{LL} = \frac{1}{\mathrm{ct}(L[n]) - \mathrm{ct}(L[n-1])} = c(s_n, L),$$

as desired.

Suppose that $M = s_n L$ is standard. We determine a value for $(s_n)_{ML}$ that yields a consistent choice of isomorphism Φ. We note that $(s_n)_{MM} = -(s_n)_{LL}$, since $M = s_n L$ is formed from L by interchanging boxes $n-1$ and

n and so $c(s_n, M) = -c(s_n, L)$. Applying both sides of $s_n^2 = 1$ to v_L gives

$$\begin{aligned} v_L &= s_n s_n v_L \\ &= s_n((s_n)_{LL} v_L + (s_n)_{ML} v_M) \\ &= (s_n)_{LL}^2 v_L + (s_n)_{ML}(s_n)_{LL} v_M + (s_n)_{MM}(s_n)_{ML} v_M + (s_n)_{LM}(s_n)_{ML} v_L \\ &= ((s_n)_{LL}^2 + (s_n)_{LM}(s_n)_{ML}) v_L + ((s_n)_{LL} + (s_n)_{MM})(s_n)_{ML} v_M. \end{aligned}$$

Since the coefficient on v_L in this last expression must equal 1, we have that

$$(s_n)_{LM}(s_n)_{ML} = 1 - (s_n)_{LL}^2 = (1 - (s_n)_{LL})(1 + (s_n)_{LL}),$$

and we are free to choose $(s_n)_{ML} = 1 + (s_n)_{LL}$ and $(s_n)_{LM} = 1 - (s_n)_{LL}$. This choice of coefficient is also consistent with $(s_n)_{MM} = -(s_n)_{LL}$, as then $(s_n)_{LM} = 1 + (s_n)_{MM} = 1 - (s_n)_{LL}$ and $(s_n)_{ML} = 1 - (s_n)_{MM} = 1 + (s_n)_{LL}$.

The proof for $k < n$ is similar; the principal change is to omit p_{k-1} and p_k from the product defining p_L^*, rather than p_{n-1} and p_n. ∎

As a result of this theorem, we can determine matrix representations σ^λ for each $\lambda \vdash n$ by their values $\sigma^\lambda(s_k)$ on the generators s_k of S_n. Picking $\{t_i^\lambda\}$ as an ordered basis for V^λ gives the coefficients of $\sigma^\lambda(s_k)$ to be

$$(\sigma^\lambda(s_k))_{ij} = (s_k)_{t_i^\lambda t_j^\lambda}. \tag{3.6}$$

Furthermore, since the σ^λs specify a complete set of matrix representations for the irreducibles, they afford an algebra-homomorphism $D : \mathbb{C}S_n \to \bigoplus_{\lambda \vdash n} \mathbb{C}^{d_\lambda \times d_\lambda}$ such that

$$D(s_k) = \bigoplus_{\lambda \vdash n} \sigma^\lambda(s_k).$$

Hence, D is a DFT for S_n, and it is the one for which we elect to develop FFTs. Consequently, unless otherwise specified, we take this D to be our DFT for S_n.

3.4 Computation and Examples of Representations

We give a more algorithmic construction of these σ^λ based on the action of s_k on T_s^λ. This action gives orbits of size at most 2, since $s_n^2 = 1$. For a similar algorithm yielding a slightly different form of the generating matrices (with 1s below the diagonal), see Clausen and Baum (7: §2) or James and Kerber (14: 3.2.29).

Algorithm 3.26 Fix $\lambda \vdash n$ and s_k, and consider the action of s_k on the t_i^λ. We observe three important cases of the orbit of t_i^λ in T_s^λ:

- If $k-1$ and k lie in the same row of t_i^λ, then $s_k t_i^\lambda$ is not standard. Since $\mathrm{ct}(t_i^\lambda[k]) - \mathrm{ct}(t_i^\lambda[k-1]) = 1$, we set $\sigma_{ii}^\lambda = c(s_k, t_i^\lambda) = 1$.
- If $k-1$ and k lie in the same column of t_i^λ, then $s_k t_i^\lambda$ is again not standard. Since $\mathrm{ct}(t_i^\lambda[k]) - \mathrm{ct}(t_i^\lambda[k-1]) = -1$, we set $\sigma_{ii}^\lambda = -1$.
- If $k-1$ and k lie in separate rows and separate columns, boxes k and $k-1$ may be interchanged freely, so $t_j^\lambda = s_k t_i^\lambda$ is standard. Without loss of generality, suppose that $t_i^\lambda < t_j^\lambda$. Then, since t_i^λ and t_j^λ differ only in the positions of k and $k-1$, it must be the case that the row index of k in t_i^λ is lower than that of k in t_j^λ. Thus, we have $\mathrm{ct}(t_i^\lambda[k]) > \mathrm{ct}(t_i^\lambda[k-1])$.

  Let $d = \mathrm{ct}(t_i^\lambda[k]) - \mathrm{ct}(t_i^\lambda[k-1])$; then $d > 0$, and we set

$$\begin{pmatrix} \sigma_{ii}^\lambda & \sigma_{ij}^\lambda \\ \sigma_{ji}^\lambda & \sigma_{jj}^\lambda \end{pmatrix} = \begin{pmatrix} d^{-1} & 1+d^{-1} \\ 1-d^{-1} & -d^{-1} \end{pmatrix}.$$

Set all remaining elements of σ^λ to 0. This procedure completely specifies $\sigma^\lambda(s_k)$. ∎

Example 3.27 We apply Algorithm 3.26 to the standard tableaux of S_3 (listed in Example 3.6) to construct $D(s_k)$ for s_2 and s_3. In their last-letter orders, these tableaux are

$$t_1^{(3)} = \begin{array}{|c|c|c|}\hline 1&2&3\\\hline\end{array}, \qquad t_1^{(2,1)} = \begin{array}{|c|c|}\hline 1&3\\\hline 2\\\cline{1-1}\end{array}, \qquad t_2^{(2,1)} = \begin{array}{|c|c|}\hline 1&2\\\hline 3\\\cline{1-1}\end{array}, \qquad t_1^{(1,1,1)} = \begin{array}{|c|}\hline 1\\\hline 2\\\hline 3\\\hline\end{array}.$$

Applying the algorithm, we have that

$$D(s_2) = \sigma^{(3)}(s_2) \oplus \sigma^{(2,1)}(s_2) \oplus \sigma^{(1,1,1)}(s_2) = \begin{pmatrix} 1 & & & \\ & -1 & 0 & \\ & 0 & 1 & \\ & & & -1 \end{pmatrix}, \tag{3.7}$$

$$D(s_3) = \sigma^{(3)}(s_3) \oplus \sigma^{(2,1)}(s_3) \oplus \sigma^{(1,1,1)}(s_3) = \begin{pmatrix} 1 & & & \\ & \frac{1}{2} & \frac{3}{2} & \\ & \frac{1}{2} & -\frac{1}{2} & \\ & & & -1 \end{pmatrix}. \tag{3.8}$$

∎

Example 3.28 We construct the block $\sigma^{(3,2)}$ of the seminormal matrix representations for the transpositions $(1\,2)$, $(2\,3)$, $(3\,4)$, and $(4\,5)$ using the standard tableaux for $\alpha = (3,2)$ presented in Equation (3.5). Table 3.1 displays the effects of this action on these tableaux. Using the algorithm above, we have that

$$\sigma^{(3,2)}((1\,2)) = \begin{pmatrix} -1 & & & & & \\ & 1 & & & & \\ & & -1 & & & \\ & & & 1 & & \\ & & & & 1 & \\ & & & & & 1 \end{pmatrix}, \quad \sigma^{(3,2)}((2\,3)) = \begin{pmatrix} \frac{1}{2} & \frac{3}{2} & & & & \\ \frac{1}{2} & -\frac{1}{2} & & & & \\ & & \frac{1}{2} & \frac{3}{2} & & \\ & & \frac{1}{2} & -\frac{1}{2} & & \\ & & & & 1 & \\ & & & & & 1 \end{pmatrix},$$

$$\sigma^{(3,2)}((3\,4)) = \begin{pmatrix} -1 & & & & & \\ & 1 & & & & \\ & & 1 & & & \\ & & & \frac{1}{3} & \frac{4}{3} & \\ & & & \frac{2}{3} & -\frac{1}{3} & \\ & & & & & 1 \end{pmatrix}, \quad \sigma^{(3,2)}((4\,5)) = \begin{pmatrix} \frac{1}{2} & & \frac{3}{2} & & & \\ & \frac{1}{2} & & & \frac{3}{2} & \\ \frac{1}{2} & & -\frac{1}{2} & & & \\ & \frac{1}{2} & & -\frac{1}{2} & & \\ & & & & 1 & \\ & & & & & 1 \end{pmatrix}.$$

As noted above, these matrices allow the construction of seminormal matrix representations for all $f \in \mathbb{C}S_5$, as this map $\sigma^{(3,2)}$ is an algebra homomorphism of $\mathbb{C}S_5$. ■

3.5 Conclusions and Generalizations

To summarize the above results, for $f \in \mathbb{C}S_n$, the matrix coefficients $\sigma^{\lambda}_{ij}(f)$ are the coefficients of $e_{t_i^\lambda, t_j^\lambda}$ in f and, by Definition 1.2, are also the Fourier coefficients of f. Hence, we have the following equivalent ways of identifying the coefficients of the matrix $D(f)$:

- By pairs of paths in the character graph $\Gamma(\mathcal{S})$,
- By pairs of standard tableaux $(t_i^\lambda, t_j^\lambda)$, where i and j specify the index of the tableaux in the last-letter order,
- By pairs of lists of eigenvalues corresponding to the left and right actions of the Jucys-Murphy elements $m_2, \ldots, m_n$ on the spaces $\mathbb{C}e_{ML}$.

We will interchange freely between these descriptions of the Fourier spaces.

	(3,2)-tableaux in last letter order				
σ	$\begin{array}{\|c\|c\|c\|}\hline 1&3&5\\\hline 2&4\\\cline{1-2}\end{array}$	$\begin{array}{\|c\|c\|c\|}\hline 1&2&5\\\hline 3&4\\\cline{1-2}\end{array}$	$\begin{array}{\|c\|c\|c\|}\hline 1&3&4\\\hline 2&5\\\cline{1-2}\end{array}$	$\begin{array}{\|c\|c\|c\|}\hline 1&2&4\\\hline 3&5\\\cline{1-2}\end{array}$	$\begin{array}{\|c\|c\|c\|}\hline 1&2&3\\\hline 4&5\\\cline{1-2}\end{array}$
$(1\,2)$	C	R	C	R	R
$(2\,3)$	$\begin{array}{\|c\|c\|c\|}\hline 1&2&5\\\hline 3&4\\\cline{1-2}\end{array}$	$\begin{array}{\|c\|c\|c\|}\hline 1&3&5\\\hline 2&4\\\cline{1-2}\end{array}$	$\begin{array}{\|c\|c\|c\|}\hline 1&2&4\\\hline 3&5\\\cline{1-2}\end{array}$	$\begin{array}{\|c\|c\|c\|}\hline 1&3&4\\\hline 2&5\\\cline{1-2}\end{array}$	R
$(3\,4)$	C	R	R	$\begin{array}{\|c\|c\|c\|}\hline 1&2&3\\\hline 4&5\\\cline{1-2}\end{array}$	$\begin{array}{\|c\|c\|c\|}\hline 1&2&4\\\hline 3&5\\\cline{1-2}\end{array}$
$(4\,5)$	$\begin{array}{\|c\|c\|c\|}\hline 1&3&4\\\hline 2&5\\\cline{1-2}\end{array}$	$\begin{array}{\|c\|c\|c\|}\hline 1&2&4\\\hline 3&5\\\cline{1-2}\end{array}$	$\begin{array}{\|c\|c\|c\|}\hline 1&3&5\\\hline 2&4\\\cline{1-2}\end{array}$	$\begin{array}{\|c\|c\|c\|}\hline 1&2&5\\\hline 3&4\\\cline{1-2}\end{array}$	R

Table 3.1: Table of action of transpositions on the standard tableaux of shape $(3,2)$. An R denotes that the boxes to be interchanged lie in the same row, and a C that they lie in the same column.

Finally, this path algebra affords a generalization of the tableaux used in the construction of the symmetric group to other MC-groups. Ram (26) explicitly constructs such a system of tableaux for WB_n, while Clausen (5) explores this notion for supersolvable groups as well. In Ram's formulation, for example, the irreducibles of WB_n are indexed by pairs of partitions whose sizes sum to n. Consequently, similar seminormal matrix representations can be developed for these groups, as well as the MC-groups noted in Section 2.4.

Chapter 4

Decimation-In-Frequency Algorithm Theory

We now discuss general algorithms for the construction of decimation-in-frequency fast Fourier transform algorithms. As discussed in Section 1.3, these algorithms differ from earlier, decimation-in-time ones in that they focus on producing successively finer decompositions of the frequency space, rather than on a factorization of the group by double-coset representatives that have sparse seminormal matrix representations. In particular, we develop the idea of the DFT as a change of basis on the group algebra $\mathbb{C}G$, and then formulate the decimation-in-frequency FFT as a sparse factorization of the change-of-basis matrix.

4.1 The DFT as a Change of Basis

Suppose G is a finite group, so that by Wedderburn's Theorem there exists an isomorphism

$$D : \mathbb{C}G \to \bigoplus_{\lambda \in \hat{G}} \mathbb{C}^{d_\lambda \times d_\lambda}.$$

Let $E_{\lambda,ij}$ be the usual matrix basis element consisting of the matrix with a 1 in the ith row and jth column of block λ in the matrix algebra, and let $u_{\lambda,ij}$ be its preimage $D^{-1}(E_{\lambda,ij})$ in $\mathbb{C}G$. Furthermore, let $U_{\lambda,ij} = \mathbb{C}u_{\lambda,ij}$ be the space spanned by $u_{k,ij}$. Then, by this isomorphism D,

$$\mathbb{C}G = \bigoplus_{\lambda \in \hat{G}} \bigoplus_{j=1}^{d_\lambda} \bigoplus_{i=1}^{d_\lambda} U_{\lambda,ij}.$$

D maps $U_{\lambda,ij}$ onto the space spanned by $E_{\lambda,ij}$, so each of these $U_{\lambda,ij}$s corresponds to one of these Fourier coefficient spaces in the matrix algebra. Thus, $\mathbb{C}G$ can be viewed both as a direct sum of the spaces spanned by the group elements of G and as a direct sum of these Fourier spaces $U_{\lambda,ij}$; the former corresponds to the time domain of the classical DFT, while the latter corresponds to the frequency domain. Consequently,

$$\mathcal{T} = \{g \mid g \in G\} \quad \text{and} \quad \mathcal{F} = \{E_{\lambda,ij} \mid \lambda \in \hat{G}, i \leq i, j, \leq d_\lambda\}$$

are bases for the group algebra and the matrix algebra, respectively, that correspond to these decompositions of the spaces. We recall from Dummit and Foote (11) the defintion and notation for matrix representations of linear transformations:

Definition 4.1 If V and W are vector spaces with bases $B = \{v_i\}$ and $E = \{w_j\}$, respectively, and if $\phi \in \mathrm{Hom}(V, W)$, then the *matrix of ϕ with respect to B and E* is denoted $[\phi]_B^E$. The ijth coefficient of this matrix is the coefficient of w_i in $\phi(v_j)$. ■

Under this formulation, we define the *DFT* matrix for D to be $[D]_{\mathcal{T}}^{\mathcal{F}}$. This matrix then transforms a coordinate vector in the group-element basis $\mathcal{T}$ to a vector consisting of the Fourier coefficients of D.

Example 4.2 We illustrate this decomposition explicitly for S_3. We recall that S_3 has three classes of irreducible representations, two of which are one-dimensional and the third of which is two-dimensional. Thus,

$$\mathbb{C}S_3 \cong \mathbb{C}^{1\times 1} \oplus \mathbb{C}^{2\times 2} \oplus \mathbb{C}^{1\times 1}.$$

We can then write $\mathbb{C}S_3$ as

$$\mathbb{C}S_3 = U_{1,11} \oplus U_{2,11} \oplus U_{2,12} \oplus U_{2,21} \oplus U_{2,22} \oplus U_{3,11},$$

where, for example, the spaces $U_{1,11}, U_{2,11} \oplus U_{2,21}, U_{2,12} \oplus U_{2,22}$, and $U_{3,11}$ form irreducible left $\mathbb{C}S_3$-modules, since they correspond to the four different columns of the matrix algebra. Computing the images of the group elements under the DFT using the algorithm detailed in Section 3.4, the matrix representation of the seminormal DFT on $\mathbb{C}S_3$ is the change of basis

matrix

$$D = \begin{pmatrix} 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & -1 & \frac{1}{2} & -\frac{1}{2} & \frac{1}{2} & -\frac{1}{2} \\ 0 & 0 & -\frac{3}{2} & \frac{3}{2} & \frac{3}{2} & -\frac{3}{2} \\ 0 & 0 & -\frac{1}{2} & -\frac{1}{2} & \frac{1}{2} & \frac{1}{2} \\ 1 & 1 & -\frac{1}{2} & -\frac{1}{2} & -\frac{1}{2} & -\frac{1}{2} \\ 1 & -1 & -1 & 1 & -1 & 1 \end{pmatrix}. \tag{4.1}$$

Such matrices are precisely the ones we wish to factor in our algorithm. ■

4.2 Path Algebras, DFTs and FFTs

We relate this change-of-basis formulation of the DFT to the path algebra representations of $\mathbb{C}G$ developed in Chapter 2. As above, let G have a chain of subgroups $\mathcal{C}$ given by $1 = G_0 < G_1 < \cdots < G_n = G$, let P_n be the associated path algebra, and let $Z_k = \{z_{k,j}\}_{j=1}^{r_k}$ be sets of central elements for $\mathbb{C}G_k$ for each $k \leq n$. Let an isomorphism $\Phi : P_n \to \mathbb{C}G$ be fixed.

Suppose $(L, M) \in \Omega(n)$ is a pair of paths such that $L = (\lambda^{(0)} \to \ldots \to \lambda^{(n)})$ and $M = (\mu^{(0)} \to \ldots \to \mu^{(n)})$. Then, with e_{ML} and v_N as defined in Chapter 2, proceeding as in Equations (3.3) and (3.4) yields that

$$z_{k,j}e_{ML} = c_{k,j}(\mu^{(k)})e_{ML} \qquad \text{and} \qquad e_{ML}z_{k,j} = c_{k,j}(\lambda^{(k)})e_{ML}. \tag{4.2}$$

Thus, the group elements e_{ML} are eigenvectors for the $z_{k,j}$s. Note that $z_{k,j}$ is central in $\mathbb{C}G_k$ and not necessarily in $\mathbb{C}G$, so that the left and right eigenvalues $c_{k,j}(\mu^{(k)})$ and $c_{k,j}(\lambda^{(k)})$ need not coincide except for $k = n$ (since $\lambda^{(n)}$ and $\mu^{(n)}$ must be equal).

Since each e_{ML} is an eigenvector for the $z_{k,j}$s, $\mathbb{C}G$ decomposes into a direct sum of eigenspaces for the $z_{k,j}$s indexed by pairs of path weights. If all paths in $\Lambda(n)$ can be distinguished by such weights, then the simultaneous left and right action of the $z_{k,j}$s decomposes $\mathbb{C}G$ into the one-dimensional spaces spanned by the e_{ML}.

As noted in Chapter 2, fixing Φ and selecting $\{v_N\}_{N \in \Lambda(\lambda)}$ as bases for the irreducible representations V^λ of P_n gives a seminormal DFT D for $\mathbb{C}G$, in which case the e_{ML}s correspond to the $u_{\lambda,ij}$s defined above, and the one-dimensional eigenspaces to the $U_{\lambda,ij}$s. Hence, the Fourier coefficients of $f \in \mathbb{C}G$ are given by the coefficients of the e_{ML} in f, which are determined by the projections of f into the eigenspaces of the $z_{k,j}$s and by the choice

of the e_{ML} as eigenbases for these spaces. While the projections of f into these eigenspaces are determined uniquely, the corresponding coefficients are not. We can rescale these coefficients so that they are consistent with the seminormal matrix representations on the generators, however. Such methods are explored further in Section 4.6.

If the paths in $\Lambda(n)$ cannot be separated by the $z_{k,j}$s, then much of the above analysis applies, except that the resulting eigenspaces will not all be one-dimensional. In that case, checking that these coefficients are consistent with the values of the predetermined seminormal DFT D requires more complicated linear transformations on the sets of Fourier coefficients corresponding to the eigenspaces, but is still theoretically feasible. In fact, for the one-dimensional eigenspaces, these linear transformations reduce to the scalings discussed above, as these are the only invertible linear transformations on one-dimensional spaces.

Additionally, we note that we can also use the successive differences $m_{k,j} = z_{k,j} - z_{k-1,j}$ of the central elements to accomplish the same separation of eigenspaces, as the lists of the eigenvalues of the $z_{k,j}$s and the $m_{k,j}$s determine each other. When it is not important to distinguish between the two sets of elements, we use $z_{k,j}$ to refer to either collection, and call them simply *separating elements*, in keeping with the terminology of separating sets introduced in Section 1.3.2. We employ such elements for the symmetric group in Chapter 5 to separate Fourier coefficient spaces.

Finally, this eigenspace approach affords a means of determining a fast Fourier transform algorithm for G: rather than projecting elements of $\mathbb{C}G$ into these one-dimensional Fourier spaces immediately, we project them into successively smaller eigenspaces by the left and right action of an appropriate set of $z_{k,j}$ elements or their differences $m_{k,j}$, with a new set acting on the left and then the right at each stage of the transform. Because at each stage we subdivide the eigenspaces of $\mathbb{C}G$ into smaller eigenspaces, this procedure realizes a *decimation-in-frequency* FFT for G.

4.3 Bimodules and Opposite Algebras

Before we discuss such decimation-in-frequency approaches to fast Fourier transforms on G, we develop some additional theoretical material. In particular, we discuss

- opposite algebras and their relation to bimodules,
- an encoding of the double coset structure of G with respect to a chain

of its subgroups,

- minimum rank decompositions of matrices.

Much of this material on opposite algebras and tensor products is drawn from Adkins and Weintraub (1), Drozd and Kirichenko (10), and Sagan (28). We first define the notion of an opposite algebra.

Definition 4.3 Let A be a $\mathbb{C}$-algebra. Let the *opposite algebra* of A, denoted A^o, has the same elements as A, but with the element $a \in A$ denoted a^o in A^o and with multiplication defined by $a^o b^o = (ba)^o$. ■

As Drozd and Kirichenko (10: §4.2) point out, if A and B are algebras, any (A, B)-bimodule M becomes a left $A \otimes B^o$-module by setting $(a \otimes b^o)m = amb$ for $a \in A$, $b \in B$, $m \in M$. Furthermore, if M is a left $A \otimes B^o$-module, setting $am = (a \otimes 1)m$ and $mb = (1 \otimes b^o)m$ for $a \in A$, $b \in B$ and $m \in M$ makes M into an (A, B)-bimodule. Hence, the two concepts are equivalent, and the study of such bimodules thus reduces to the study of tensor products and opposite algebras. Unless otherwise stated, we assume that all tensor products are over $\mathbb{C}$. We also typically drop the ^{o} on opposite algebra elements in tensor products if it is clear from context.

We note that the usual algebra multiplication on $\mathbb{C}G$ makes it into a $(\mathbb{C}G, \mathbb{C}G)$-bimodule, so by this correspondence it is also a left $(\mathbb{C}G \otimes \mathbb{C}G^o)$-module. Furthermore, if H and K are subgroups of G, then by restriction $\mathbb{C}G$ is also a left $(\mathbb{C}H \otimes \mathbb{C}K^o)$-module.

We state two results about the irreducible representations of opposite algebras and tensor products of group algebras.

Proposition 4.4 *The isomorphism classes of simple left $\mathbb{C}G$-modules coincide with those of its opposite algebra $\mathbb{C}G$.*

Proof: As noted in Adkins and Weintraub (1: 3.1.2), the antiautomorphism $\phi(g) = g^{-1}$ on G extends by linearity to an algebra antiautomorphism $\phi : \mathbb{C}G \to \mathbb{C}G$. Similarly, the opposite map from $\mathbb{C}G$ to $\mathbb{C}G^o$ given by $\alpha \to \alpha^o$ for each $\alpha \in \mathbb{C}G$ gives an antiisomorphism of these algebras. Hence, their composition $\sigma : \mathbb{C}G \to \mathbb{C}G^o$ given by $\sigma(\alpha) = \phi(\alpha)^o$ is an algebra isomorphism. Since the algebras are isomorphic, their isomorphism classes of simple left modules must coincide. ■

Let A and B be $\mathbb{C}$-algebras, and let M be an A-module and N a B-module. Their tensor product $M \otimes N$ over $\mathbb{C}$ is a $A \otimes B$-module, where multiplication for simple tensors is given by

$$(a \otimes b)(m \otimes n) = (am \otimes bn)$$

and extends to all tensors by linearity over $\mathbb{C}$.

Proposition 4.5 *Let G and H be groups.*

1. *If X and Y are irreducible representations for G and H, respectively, then $X \otimes Y$ is an irreducible representation of $G \times H$.*
2. *If $\{X_i\}$ and $\{Y_i\}$ are complete lists of inequivalent irreducible representations for G and H, then $\{X_i \otimes Y_j\}_{i=1,j=1}^{m,n}$ is a complete list of inequivalent irreducible representations for $G \times H$.* ■

By the algebra isomorphisms

$$\mathbb{C}(G \times H) \cong \mathbb{C}G \otimes \mathbb{C}H \cong \mathbb{C}G \otimes \mathbb{C}H^o$$

where the first is given by extending $(g, h) \mapsto g \otimes h$ by linearity and the second by $\sigma : \mathbb{C}H \to \mathbb{C}H^o$ as defined in Proposition 4.4. By this result and by Proposition 4.5, the irreducible types of $\mathbb{C}G \otimes \mathbb{C}H^o$ are the same as those of $\mathbb{C}(G \times H)$. Finally, by the semisimplicity of $\mathbb{C}(G \times H)$, any $(\mathbb{C}G \otimes \mathbb{C}H^o)$-module decomposes into a direct sum of irreducibles.

Example 4.6 Considered as a $(\mathbb{C}G \otimes \mathbb{C}G^o)$-module, $\mathbb{C}G$ can be written as a direct sum of tensor products of its irreducibles. Furthermore, taking $G_0 < G_1 < \cdots < G_n = G$ to be a chain of subgroups of G, $\mathbb{C}G$ considered as a $(\mathbb{C}G_i \otimes \mathbb{C}G_j^o)$-module decomposes into a direct sum of tensor products of the irreducibles of G_i and G_j.

The decimation-in-frequency strategy proposed above in Section 4.2 then corresponds to the successive decomposition of $\mathbb{C}G$ with respect to the chain of algebras

$$\mathbb{C}G_0 \otimes \mathbb{C}G_0^o \subset \mathbb{C}G_1 \otimes \mathbb{C}G_0^o \subset \mathbb{C}G_1 \otimes \mathbb{C}G_1^o \subset \mathbb{C}G_2 \otimes \mathbb{C}G_1^o \subset \cdots \subset \mathbb{C}G_n \otimes \mathbb{C}G_n^o \tag{4.3}$$

where we alternately increment the indices of the left and right G_is. ■

4.4 Double-Coset Branchings and Bases

We can also encode the chains of tensor algebras such as that presented in Example 4.6 as a chain of pairs of subgroups for G. This motives the following definition:

Definition 4.7 Let G be a finite group, and let $H_0 \leq H_1 \leq \cdots \leq H_n$ and $K_0 \leq K_1 \leq \cdots \leq K_n$ be chains of subgroups of G of equal length. Then the sequence $\{(H_j, K_j)\}_{j=0}^n$ is a *chain of subgroup pairs* for G. ■

Example 4.8 The chain of group algebra tensor products presented in Equation (4.3) corresponds to the chain

$$(G_0, G_0) < (G_1, G_0) < (G_1, G_1) < \cdots < (G_n, G_n)$$

of subgroup pairs for the group G. ■

Let $\mathcal{P} = \{(H_j, K_j)\}$ be a chain of subgroup pairs for G. We note that the action of the corresponding group algebra $\mathbb{C}H_j \otimes \mathbb{C}K_j^o$ on $\mathbb{C}G$ makes $\mathbb{C}G$ into a $(\mathbb{C}H_j \otimes \mathbb{C}K_j^o)$-module by restriction, and hence it will decompose into a direct sum of such modules. Moreover, the $(\mathbb{C}H_j \otimes \mathbb{C}K_j^o)$-modules generated by the standard basis for $\mathbb{C}G$ (i.e., the elements of G considered as elements of $\mathbb{C}G$) are precisely those spanned by the double cosets of H_j and K_j in G. We refer to such modules as (H_j, K_j)-*double-coset modules*. Furthermore, because $H_{j-1} \leq H_j$ and $K_{j-1} \leq K_j$, the double cosets of H_j and K_j are a disjoint union of double cosets of H_{j-1} and K_{j-1}, so these (H_j, K_j)-double-coset modules decompose into direct sums of (H_{j-1}, K_{j-1})-double-coset modules.

Example 4.9 Consider the chain $\mathcal{P}$

$$(S_1, S_1) < (S_2, S_1) < (S_2, S_2) < (S_3, S_2) < (S_3, S_3)$$

of subgroup pairs of S_3. Since

$$D_{1,1} = \{1, (1\,2)\} \quad \text{and} \quad D_{1,2} = \{(1\,3), (1\,3\,2), (2\,3), (1\,2\,3)\}$$

are the two (S_2, S_2)-double cosets in S_3, $\mathbb{C}S_3$ decomposes into two double-coset modules, $\mathbb{C}D_{1,1}$ and $\mathbb{C}D_{1,2}$, which are spanned by these double cosets.

Furthermore, since the (S_2, S_1)-double cosets in S_3 are $D_{2,1} = D_{1,1}$, $D_{2,2} = \{(1\,3), (1\,3\,2)\}$, and $D_{2,3} = \{(2\,3), (1\,2\,3)\}$, as $\mathbb{C}S_2 \otimes \mathbb{C}S_1^o$-modules, $\mathbb{C}D_{1,1} = \mathbb{C}D_{2,1}$, while $\mathbb{C}D_{1,2} = \mathbb{C}D_{2,2} \oplus \mathbb{C}D_{2,3}$.

We remark that these double-coset modules, while cyclic, are not necessarily irreducible. They are, however, the smallest subalgebra modules that can be written with bases consisting of subsets of the standard group-element basis for $\mathbb{C}G$. ■

We encode this branching of double cosets associated to pairs of subgroups in a chain of subgroup pairs as follows.

Definition 4.10 Let G be a finite group, and let $\mathcal{P}$ be a chain of subgroup pairs given by $\{(H_j, K_j)\}_{j=0}^n$.

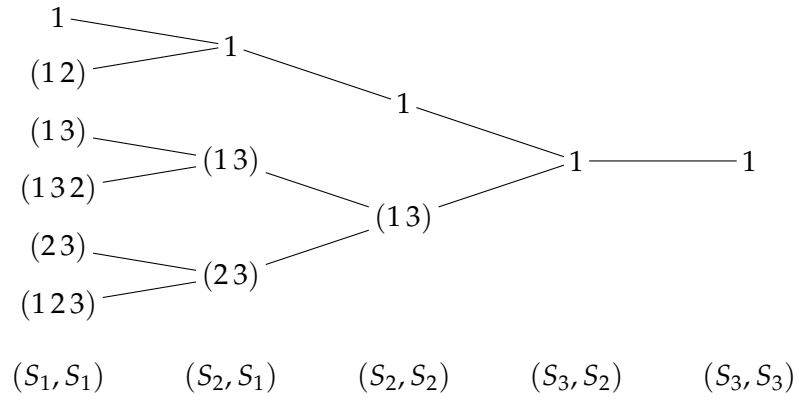


Figure 4.1: Graphical depiction of double-coset branching for S_3. The indexing sets for the double cosets are taken to be arbitrary choices of representatives from the double cosets.

- For each $0 \leq j \leq n$, let I_j be an index set for the (H_j, K_j)-double cosets in G.
- For each $1 \leq j \leq n$, let $s_j : I_{j-1} \to I_j$ be a function such that for each index $i \in I_{j-1}$, $s_j(i)$ gives the index of the (H_j, K_j)-double coset containing the ith (H_{j-1}, K_{j-1})-double coset.

We define the *double-coset branching* associated to $\mathcal{P}$ to be the collection of the I_js and the s_js, and denote it by $\mathcal{B}(\mathcal{P})$. The s_j are then called the *successor functions* for the branching. ■

These successor functions s_j are well defined precisely because of this partitioning behavior of the double cosets for the pairs of subgroups in the chain.

Example 4.11 The chain of subgroup pairs for S_3 in Example 4.9 corresponds to the double-coset branching depicted in Figure 4.1. Here, particular choices of coset representatives are used to index the double cosets at each stage. Then, for example, $s_2((2\,3)) = (1\,3)$, since the (S_2, S_1)-double coset corresponding to $(2\,3)$ is part of the (S_2, S_2)-double coset corresponding to $(1\,3)$. ■

Closely associated with such branchings are group-element bases for $\mathbb{C}G$ that respect the branching.

Definition 4.12 Let G be a finite group, and consider a chain $\mathcal{P}$ of subgroup pairs for G. A *double-coset basis* for $\mathbb{C}G$ with respect to $\mathcal{P}$ is an ordered list $L = (g_1, \ldots, g_k)$ of the elements of G such that, for each pair $(H, K) \in \mathcal{P}$, L is partitioned into sublists that correspond to the double cosets of H and K in G. ■

Thus, a double-coset basis is one for which this action induces a partition of the basis into group-element bases for the double cosets in contiguous blocks. The computation of such bases relies in part on the determination of double-coset transversals for each pair of subgroups in the pairing chain above. Methods for constructing such transversals are discussed in Brown et al. (2). When $K \leq H \leq G$ and when there is a right transversal of H in G that consists of orbits of the elements of G under the action of K by conjugation, the computation of (H, K)-double cosets can be simplified considerably. We discuss such computation with respect to the symmetric group in Chapter 5.

Example 4.13 We exhibit a double-coset basis for S_3 with respect to the chain of subgroup pairs

$$(S_1, S_1) < (S_2, S_1) < (S_2, S_2) < (S_3, S_2) < (S_3, S_3).$$

One such double coset basis is

$$D = (1, (1\,2), (1\,3), (1\,2\,3), (2\,3), (1\,3\,2)),$$

as $S_2 = \{1, (1\,2)\}$ and $S_2(1\,3)S_2 = \{(1\,3), (1\,2\,3), (2\,3), (1\,3\,2)\}$ are the double cosets of S_2 and S_2 in S_3. ■

Finally, we note that an order chosen on each of the indexing sets I_j for a branching $\mathcal{B}(\mathcal{P})$ determines a double-coset basis for G. For each $g \in G$, let $p_j(g)$ denote the index of the double coset of H_j and K_j containing g. Defining

$$L(g) = (p_n(g), p_{n-1}(g), \ldots, p_1(g), p_0(g)),$$

we see that sorting G by the lexicographic order on the $L(g)$ gives a double-coset order for G with respect to $\mathcal{P}$. We note that $p_j(g) = (s_j \circ s_{j-1} \circ \cdots \circ s_1)(p_0(g))$, so we can compute such lists, and hence the corresponding double-coset basis, given only the double-coset branching for the chain and the initial locations of the elements of G in the smallest double cosets for the chain.

4.5 Projections and Minimum Rank Decompositions

We now discuss the eigenspace projection operators associated to each of the separating elements $z_{k,j}$, and efficient means of constructing and storing these operators. We note that, for any choice of isomorphism Φ, the e_{ML} form simultaneous left and right eigenbases for $\mathbb{C}G$ with respect to each $z_{k,j}$. Hence, for any choice of basis B for $\mathbb{C}G$, the matrix representation $[z_{k,j} \otimes 1]_B^B$ is diagonalizable, and so can be written

$$[z_{k,j} \otimes 1]_B^B = \sum_\lambda \lambda P_\lambda, \tag{4.4}$$

where λ ranges over the eigenvalues of $z_{k,j}$, and P_λ is the projection matrix that projects into the eigenspace W^λ associated to λ. Furthermore, we can write these projections in a factored form: if E_λ is a eigenbasis for W^λ, then it is a fact of elementary linear algebra that

$$P_\lambda = [E_\lambda]_B(([E_\lambda]_B)^T[E_\lambda]_B)^{-1}([E_\lambda]_B)^T. \tag{4.5}$$

The same can be done for the projections associated with the eigenspaces of $1 \otimes z_{k,j}$.

We also note that a general $m \times n$ matrix A of rank r can be factored into $A = DC$, where D is a fully ranked $m \times r$ matrix and C is a fully ranked $r \times n$ matrix. The algorithm below gives one means of accomplishing this using row-reduction.

Algorithm 4.14 Let A be an $m \times n$ matrix of rank r. Row reduce the augmented matrix $[A \mid I_m]$ to $[B \mid E]$ using E. Then

$$B = \begin{pmatrix} c_1 \\ \vdots \\ c_r \\ 0 \\ \vdots \\ 0 \end{pmatrix} \quad \text{and} \quad E^{-1} = \begin{pmatrix} d_1 & d_2 & \cdots & d_r & d_{r+1} & \cdots & d_m \end{pmatrix},$$

where the c_i are $1 \times n$ row vectors and the d_i are $m \times 1$ column vectors. Set

$$C = \begin{pmatrix} c_1 \\ \vdots \\ c_r \end{pmatrix} \quad \text{and} \quad D = \begin{pmatrix} d_1 & d_2 & \cdots & d_r \end{pmatrix}.$$

Then $A = DC$, and C and D are fully ranked and of size $m \times r$ and $r \times n$, respectively.

Proof: By the row reduction, B takes the form above, with r nonzero rows, E is invertible, and $A = E^{-1}B$. Expanding this expression by outer products gives

$$A = E^{-1}B = d_1 \cdot c_1 + \cdots + d_r \cdot c_r + d_{r+1} \cdot 0 + \cdots + d_n \cdot 0 = DC. \qquad \blacksquare$$

4.6 Decimation-in-Frequency Algorithms

With these tools at our disposal, we can now specify a general algorithm for the precomputation of a decimation-in-frequency fast Fourier transform. Let G be a finite group.

- Let $1 = G_0 < G_1 < \cdots < G_n = G$ be a chain of its subgroups, denoted $\mathcal{C}$.
- Let $Z_k = \{z_{k,j}\}_{j=1}^{r_k}$ be a set of separating elements for this chain $\mathcal{C}$, where each $Z_k \subset \mathbb{C}G_k$.
- Let B be a group-element basis for $\mathbb{C}G$.
- Let D be a seminormal DFT for G adapted to $\mathcal{C}$, and let $\mathcal{F}$ be the corresponding Fourier basis for $\bigoplus_{\lambda \in \hat{G}} \mathbb{C}^{d_\lambda \times d_\lambda}$. We place the elements of $\mathcal{F}$ in row-major order, treating elements of $\bigoplus_{\lambda \in \hat{G}} \mathbb{C}^{d_\lambda \times d_\lambda}$ as direct sums of matrices.

Our decimation-in-frequency FFT algorithm produces a sparse factorization of the matrix that, for $f \in \mathbb{C}G$, takes $[f]_B$ to its coordinates in an eigenbasis for the separating elements $z_{k,j}$. With an additional factor to transform these eigenspace coordinates into those that are consistent with a seminormal DFT D for G, this algorithm gives a sparse factorization of the corresponding DFT matrix $[D]_B^{\mathcal{F}}$.

Because we wish to determine the change of basis matrix from the group-element basis B for $\mathbb{C}G$ to these eigenbasis, our principal strategy will be to focus on the changes of basis induced by the eigenspace projections for the $z_{k,j}$. Specifically, for each $k \leq n$, we project based on the left action of the $z_{k,j}$ on $\mathbb{C}G$, then on the right action of the $z_{k,j}$.

We structure this alternating action as follows.

- Let $\mathcal{P}$ be the chain of subgroup pairs

$$(G_0, G_0) < (G_1, G_0) < (G_1, G_1) < (G_2, G_1) < \cdots < (G_n, G_n).$$

- Let $\mathcal{B}(\mathcal{P})$ be the double-coset branching associated to $\mathcal{P}$. Let I_j be an (ordered) index set for the double cosets of the jth pair in $\mathcal{P}$, and let $s_j : I_{j-1} \to I_j$ be the successor functions that determine the branching.
- Let $P_0, \ldots, P_{2n}$ denote the elements of $\mathcal{P}$, and let $\mathbb{C}P_0, \ldots, \mathbb{C}P_{2n}$ denote the corresponding tensor products of group algebras. Thus, for k between 1 and n, we have $P_{2k-1} = (G_k, G_{k-1})$, and $P_{2k} = (G_k, G_k)$. Likewise, $\mathbb{C}P_{2k-1} = \mathbb{C}G_k \otimes \mathbb{C}G_{k-1}^o$, and $\mathbb{C}P_{2k} = \mathbb{C}G_k \otimes \mathbb{C}G_k^o$.
- For k between 1 and n, let $t_{2k-1,j}$ denote $z_{k,j} \otimes 1$ and $t_{2k,j} = 1 \otimes z_{k,j}$. Let $r'_{2k-1} = r'_{2k} = r_k$, so that for each m between 1 and $2n$ r'_m gives the number of $t_{m,j}$ elements.

As discussed above, when treated as a $\mathbb{C}P_m$-module, $\mathbb{C}G$ decomposes as a direct sum of the spaces spanned by the P_m-double cosets, each of which is a $\mathbb{C}P_m$-module in its own right. Consequently, we can further subdivide the relevant projection operators and eigenbases into operators and bases that relate only to a single double-coset module. In fact, since we start with a group-element basis on $\mathbb{C}G$, and since the double-coset modules are precisely the cyclic modules generated by these basis elements, this decomposition of $\mathbb{C}G$ by double cosets is the finest decomposition that gives block-matrix forms for the matrix representations of the projections and eigenbases with respect to B. Keeping the eigenbases and projections localized to double-coset submodules also has computational advantages that we explore below.

If B is a double-coset basis for $\mathbb{C}G$ with respect to $\mathcal{P}$, these projection matrices with respect to B are exactly direct sums of these matrix blocks; if B is not, these matrices are these direct sums conjugated by the permutation change of basis taking B to a double-coset basis.

Example 4.15 Consider S_3 with the double-coset basis D of Example 4.13. Let $z_2 = \frac{1}{2}(1 + (1\,2))$, one of the centrally primitive idempotents for $\mathbb{C}S_2$. With respect to D, we have

$$[z_2 \otimes 1]_D^D = \begin{pmatrix} \frac{1}{2} & \frac{1}{2} & 0 & 0 & 0 & 0 \\ \frac{1}{2} & \frac{1}{2} & 0 & 0 & 0 & 0 \\ 0 & 0 & \frac{1}{2} & \frac{1}{2} & 0 & 0 \\ 0 & 0 & \frac{1}{2} & \frac{1}{2} & 0 & 0 \\ 0 & 0 & 0 & 0 & \frac{1}{2} & \frac{1}{2} \\ 0 & 0 & 0 & 0 & \frac{1}{2} & \frac{1}{2} \end{pmatrix} \text{ and } [1 \otimes z_2]_D^D = \begin{pmatrix} \frac{1}{2} & \frac{1}{2} & 0 & 0 & 0 & 0 \\ \frac{1}{2} & \frac{1}{2} & 0 & 0 & 0 & 0 \\ 0 & 0 & \frac{1}{2} & 0 & 0 & \frac{1}{2} \\ 0 & 0 & 0 & \frac{1}{2} & \frac{1}{2} & 0 \\ 0 & 0 & 0 & \frac{1}{2} & \frac{1}{2} & 0 \\ 0 & 0 & \frac{1}{2} & 0 & 0 & \frac{1}{2} \end{pmatrix}$$

We observe that $[z_2 \otimes 1]_D^D$ is a matrix direct sum of projection matrices that act on the $(\mathbb{C}S_2 \otimes \mathbb{C}S_1^o)$-double-coset submodules of $\mathbb{C}S_3$, while $[1 \otimes z_2]_D^D$ is a matrix direct sum of projection matrices that act on the $(\mathbb{C}S_2 \otimes \mathbb{C}S_2^o)$-double-coset submodules of $\mathbb{C}S_3$. Because of this block decomposition, it is possible to construct eigenbases for these projections such that their basis vectors lie in these double-coset submodules. In particular, reasonable choices for such eigenbases in this case are

$$\left\{ \begin{pmatrix} 1 \\ 1 \\ 0 \\ 0 \\ 0 \\ 0 \end{pmatrix}, \begin{pmatrix} 1 \\ -1 \\ 0 \\ 0 \\ 0 \\ 0 \end{pmatrix}, \begin{pmatrix} 0 \\ 0 \\ 1 \\ 1 \\ 0 \\ 0 \end{pmatrix}, \begin{pmatrix} 0 \\ 0 \\ 1 \\ -1 \\ 0 \\ 0 \end{pmatrix}, \begin{pmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 1 \\ 1 \end{pmatrix}, \begin{pmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 1 \\ -1 \end{pmatrix} \right\}$$

and

$$\left\{ \begin{pmatrix} 1 \\ 1 \\ 0 \\ 0 \\ 0 \\ 0 \end{pmatrix}, \begin{pmatrix} 1 \\ -1 \\ 0 \\ 0 \\ 0 \\ 0 \end{pmatrix}, \begin{pmatrix} 0 \\ 0 \\ 1 \\ 1 \\ 1 \\ 1 \end{pmatrix}, \begin{pmatrix} 0 \\ 0 \\ 1 \\ 1 \\ -1 \\ -1 \end{pmatrix}, \begin{pmatrix} 0 \\ 0 \\ 1 \\ -1 \\ 1 \\ -1 \end{pmatrix}, \begin{pmatrix} 0 \\ 0 \\ 1 \\ -1 \\ -1 \\ 1 \end{pmatrix} \right\}$$

■

As a result of this reducibility of the spaces with respect to double-coset modules, we introduce two final notational items.

Suppose that $T = \{t_1, \ldots, t_k\}$ is a set of separating elements for $\mathbb{C}G$, and that $T \subset \mathbb{C}P_m$. For each $j \leq k$, let λ_j be an eigenvalue of t_j, and let $\lambda = (\lambda_1, \ldots, \lambda_k)$. For $i \in I_m$, define $W^{\lambda,i}$ to be

$$W^{\lambda,i} = \mathbb{C}D_i \cap \bigcap_{j=1}^{k} W^{\lambda_j},$$

where $\mathbb{C}D_i$ is the double-coset module in $\mathbb{C}G$ corresponding to i, and W^{λ_j} is the eigenspace of t_j corresponding to λ_j in $\mathbb{C}G$. Then $W^{\lambda,i}$ is the simultaneous eigenspace of the t_j for these eigenvalues λ_j in this double-coset module.

Consider $\mathbb{C}P_m$ for $m \leq 2n$. Let λ be one of the eigenvalues of the separating element $t_{m,j} \in \mathbb{C}P_m$, with corresponding eigenspace W in $\mathbb{C}G$. Since $W = \bigoplus_{i \in I_m} W^i$, where W^i is contained in the ith P_m-double-coset module,

there exist bases $E_{m,j}^{\lambda,i}$ for the W^i that also reside in these double-coset modules. Hence, a basis $E_{m,j}^{\lambda}$ for W is given by

$$E_{m,j}^{\lambda} = \bigcup_{i \in I_m} E_{m,j}^{\lambda,i}.$$

We call the $E_{m,j}^{\lambda,i}$ double-coset adapted eigenbases for the eigenspace projections of the $t_{k,j}$.

With this framework in mind, we describe a collection of algorithms for the computation of a sparse factorization of the DFT matrix associated with the seminormal DFT D. Our first algorithm transforms bases for simultaneous eigenspaces of a set of separating elements T into those that are also eigenbases for the next separating element, t, and hence provides the main step in the precomputation algorithm.

Algorithm 4.16 Fix $m \leq 2n$, and let the algebra $\mathbb{C}P_m$ and group-element basis B for $\mathbb{C}G$ be as above. Let T be a set of separating elements in $\mathbb{C}P_m$, and let t be a new separating element of $\mathbb{C}P_m$. Suppose we have the following:

- Let $S = (S_j)_{j=1}^r$ be a list of lists S_j such that each list $S_j = (i_j, \lambda_j)$, where $i_j \in I_m$ and λ_j is a list of the eigenvalues of the elements of T. Moreover, let each entry in S be unique, and let each list (i_j, λ_j) such that W^{λ_j, i_j} is nontrivial be included in S, so that S is complete.
- Let $D = (D_j)_{j=1}^r$, where D_j is a matrix whose columns are the B-coordinates of a basis for W^{λ_j, i_j} as defined above.
- For each eigenvalue μ of t and each double-coset index $i \in I_m$, let $W^{\mu,i}$ be the eigenspace for μ in the ith double-coset module of $\mathbb{C}G$. Let $E_{\mu,i}$ be the coordinates with respect to B of an eigenbasis for $W^{\mu,i}$.

Let D' and S' be empty lists.

1. For each $E_{\lambda,i}$, compute and store $(E_{\lambda,i}^T E_{\lambda,i})^{-1}$ (these can also be precomputed and provided to the algorithm).
2. For each $j = 1$ to r,
   2.1. Set i to be the double-coset index i_j of D_j, as determined by the list $S_j = (i_j, \lambda_j)$.
   2.2. For each eigenvalue μ of t,

2.2.1. Compute the matrix $E_{\mu,i}^T D_j$.

2.2.2. Decompose $E_{\mu,i}^T D_j$ into $\tilde{D}_{\mu j}$ and $C_{\mu j}$ according to a minimum-rank decomposition algorithm such as Algorithm 4.14.

2.2.3. Compute $D_{\mu j} = E_{\mu,i}(E_{\mu,i}^T E_{\mu,i})^{-1}\tilde{D}_{\mu j}$.

2.2.4. Append $D_{\mu j}$ to D'.

2.2.5. Append $(i_j, \lambda_{j,\mu})$ to S', where $\lambda_{j,\mu}$ is λ_j with μ appended.

2.3. Concatenate the columns of the $C_{\mu j}$s in order to form the matrix $\bar{C}_j$.

3. Set the matrix F to be the direct sum $\bigoplus_{j=1}^{r} \bar{C}_j$ of the $\bar{C}_j$ matrices.

Return F, D', S'. ∎

Theorem 4.17 *In Algorithm 4.16 above, $S' = (S'_j)_{j=1}^{r'}$ is a complete list of lists of double-coset indices for P_m-double cosets and eigenvalues for each of the elements in $T \cup \{t\}$. Writing $S'_j = (i'_j, \lambda'_j)$, each D'_j in $D' = (D'_j)_{j=1}^{r'}$ consists of the coordinates with respect to B of a basis for $W^{\lambda'_j, i'_j}$. Finally, F is the change of basis matrix $[1]_D^{D'}$ that changes coordinates from D to D'.*

Proof: Since t and the elements of T are separating elements of $\mathbb{C}P_m$ as defined above, there exists a basis of $\mathbb{C}G$ that diagonalizes all of these elements simultaneously. The eigenbases in D already diagonalize the elements of T by hypothesis. This algorithm changes the basis of each D_j into one that partitions into eigenbases for both T and t.

We note that $P_{\mu,i} = E_{\mu,i}(E_{\mu,i}^T E_{\mu,i})^{-1} E_{\mu,i}^T$ projects $\mathbb{C}G$ onto $W^{\mu,i}$. Since the sum $\sum_\mu P_{\mu,i}$ of these projections is the identity on the ith double-coset module of $\mathbb{C}G$, we then have

$$\begin{aligned} D_j &= \sum_\mu E_{\mu,i}(E_{\mu,i}^T E_{\mu,i})^{-1} E_{\mu,i}^T D_j \\ &= \sum_\mu E_{\mu,i}(E_{\mu,i}^T E_{\mu,i})^{-1} \tilde{D}_{\mu j} C_{\mu j} \\ &= \sum_\mu D_{\mu j} C_{\mu j}, \end{aligned}$$

where $C_{\mu j}$, $\tilde{D}_{\mu j}$, and $D_{\mu j}$ are as defined above. Expanding this via outer products gives

$$D_j = \sum_\mu D_{\mu j} C_{\mu j} = \begin{pmatrix} D_{\mu_1 j} & \cdots & D_{\mu_k j} \end{pmatrix} \begin{pmatrix} C_{\mu_1 j} \\ \vdots \\ C_{\mu_k j} \end{pmatrix} = \begin{pmatrix} D_{\mu_1 j} & \cdots & D_{\mu_k j} \end{pmatrix} \bar{C}_j.$$

Then $\begin{pmatrix} D_{\mu_1 j} & \cdots & D_{\mu_m j} \end{pmatrix}$ is precisely the list of blocks appended to D'. Furthermore, each $D_{\mu j}$ has a double-coset/eigenvalue list given by $(i_j, \lambda_{j,\mu})$, and since the μ were unique, so are these new lists.

Since S_j is unique for each j, these new lists $S_{\mu j}$ that constitute S' are also all unique, so each $D_{\mu j}$ in D' is a complete eigenbasis for the corresponding intersections of eigenspaces for T and t.

Let $[D]$ and $[D']$ represent the columns of D and D', respectively, concatenated to form $|G| \times |G|$ matrices. By construction, $[D']F = [D]$. But $[D] = [1]_D^B$ and $[D'] = [1]_{D'}^B$, so

$$F = [1]_B^{D'}[1]_{D'}^B F = [1]_B^{D'}[D]F = [1]_B^{D'}[D] = [1]_B^{D'}[1]_D^B = [1]_D^{D'}. \qquad \blacksquare$$

We note that we need keep only those elements of D' and S' that correspond to nontrivial eigenspaces of $\mathbb{C}G$, as the others will correspond to empty lists in D'.

The decimation-in-frequency algorithm then essentially consists of repeated application of Algorithm 4.16, as well as a grouping together of double-coset bases at each new pair in the chain of subgroup pairs.

Algorithm 4.18 Let any symbols not defined here explicitly be as defined earlier in this section.

- Let $B_{0,k} = ([g_k]_B)$, the coordinates of the kth element of B with respect to B, and let B_0 be the list $(B_{0,k})_{k=1}^{|G|}$.
- Let F be an empty list, which we use to store the factors of the seminormal DFT matrix.
- Let S_0 be a list of $|G|$ lists, the kth element of which consists of $(p_0(B_{0,k}))$, the index of the (G_0, G_0)-double coset containing $B_{0,k}$.

1. For each $m = 1$ to $2n - 1$,
   1.1. Apply the successor function s_m to the first element of each list in S_{m-1}, and set S'_{m-1} to the result.
   1.2. Sort B_{m-1} according to the lexicographic order on the corresponding lists in S'_{m-1}. Concatenate those elements of B_{m-1} with identical lists in S'_{m-1}. Set $B_{m,0}$ to this sorted, concatenated B_{m-1}, and set $S_{m,0}$ to the sorted S'_{m-1} with duplicate lists removed. Set P_m to the (permutation) change-of-basis matrix determined by this reordering, and append it to F.

1.3. For $j = 1$ to r'_m, compute the factor $F_{m,j}$, the new eigenbasis $B_{m,j}$, and the new eigenvalue list $S_{m,j}$ from $B_{m,j-1}$, the coordinates $\{[E^{i,\lambda}_{m,j}]_B\}$ of the projection eigenbases, and $S_{m,j-1}$ according to Algorithm 4.16.

1.4. Append $F_{m,j}$ to F, and set $S_m = S_{m,r'_m}$ and $B_m = B_{m,r'_m}$.

2. Compute a coefficient permutation matrix P_{DFT} as described below using S and the order on the Fourier basis $\mathcal{F}$ and append P_{DFT} to F.

3. Compute a scaling matrix S_{DFT} and possibly a second permutation matrix P'_{DFT} from F and D as described below. ■

Theorem 4.19 *In Algorithm 4.18 above, B_{2n} consists of blocks of coordinates with respect to the basis B of bases for the simultaneous eigenspaces of the separating elements $t_{i,j}$.*

Furthermore, $F = (P_1, F_{1,1}, \ldots, P_{2n-1}, \ldots, F_{2n-1,r'_{2n-1}}, P_{DFT}, S_{DFT}, P'_{DFT})$ is a sparse factorization of the DFT matrix associated to D, such that

$$P'_{DFT} F_{DFT} P_{DFT} F_{2n-1,r'_{2n-1}} \cdots P_{2n-1} \cdots F_{1,1} P_1 = [D]^{\mathcal{F}}_B. \tag{4.6}$$

Proof: We proceed by induction on m, first showing that at each stage S_m contains unique lists and that the elements of B_m correspond to the $W^{\lambda,i}$, where $(i, \lambda) \in S_m$.

The initial basis B_0 trivially consists of blocks of eigenbases that diagonalize the $z_{0,j}$, which can consist only of complex scalars since $G_0 = 1$. Furthermore, the lists in S_0 uniquely identify the blocks in B_0, since each element of G determines its own (G_0, G_0)-double-coset.

Suppose that B_{m-1} is as specified above. The sorting and concatenation operation produces bases and guarantees that elements of S'_{m-1} are unique. Repeated application of Algorithm 4.16 for the $t_{m,j}$ then produces unique elements in S_m and blocks in B_m that give bases for the corresponding spaces. Since each block represents a complete eigenbasis, the lists in S_m are unique. Thus, after stage $m = 2n$, B_{2n} consists of the desired simultaneous eigenbases.

Each application of Algorithm 4.16 provides the change of basis matrix $F_{m,j}$ that takes the coordinates of a vector in the $B_{m,j-1}$ basis to those in the $B_{m,j}$ basis. Since the P_m matrices give the change of basis that corresponds to combining bases in the next set of double-coset modules, their product with the $F_{m,j}$ matrices gives the change of basis into B_{2n}.

We note, however, that the factors from the $m = 2n$ stage are trivial. The only double coset of G_n and G_{n-1} in G is G, as is the only double coset of G_n and G_n in G, so no basis vectors in B_{2n-1} are grouped together after the application of the successor function to their double coset indices. Furthermore, regardless of whether the separating elements $z_{n,j}$ are central elements or are differences of them, the eigenvalues of $t_{2n,j} = 1 \otimes z_{n.j}$ on each of the spaces in B_{2n-1} are determined by the previously computed eigenvalues on each of these spaces. Hence, they perform no additional separation of these spaces, and so the $F_{2n,j}$ factors are all trivial as well. Thus, we may terminate the process after the computation of the $m = 2n - 1$ factors.

Since the Fourier basis $\mathcal{F}$ associated with the seminormal DFT D also diagonalizes these $t_{i,j}$ separating elements, there exists a change of basis on each block in B_{2n} that takes it to the Fourier basis. By the description below, P_{DFT} and F_{DFT} effect this final change of basis. ■

We elect not to give general algorithms for the computation of P_T and F_T above, as in general they will depend on how the lists of separating-element eigenvalues in S_{2n} relate to the indexing of the entries in the matrix algebra $\bigoplus_{\lambda \in \hat{G}} \mathbb{C}^{d_\lambda \times d_\lambda}$. Instead, we describe this in general below; specific algorithms for the symmetric group are found in in Chapter 5.

In general, the coordinates of the vector should be ordered to correspond with the row-major order of the Fourier coefficients from the seminormal DFT D. Hence, given a relation between the eigenvalues of the separating elements and the row and column indices for the matrix algebra, the blocks in B_{2n} can be rearranged to match this order. Thus, if the separating elements can separate rows 1, 2 and 3, and 4, and likewise for the columns, we permute the coefficients to the order $(1,1)$, $(1,2/3)$, $(1,4)$, $(2/3,1)$, $(2/3,2/3)$, $(2/3,4)$, and so on. This reordering of basis vectors generates the permutation matrix P_{DFT}.

Once these blocks are sorted, we determine linear transformations that map these coefficients to the Fourier coefficients for the seminormal DFT. For each block we desire a matrix A such that $Ab = s$, where b is a vector of coefficients corresponding to that block and s is the correct seminormal representation of the coefficients. Thus, for a block of size d, a set $\{b_1, \ldots, b_d\}$ of d linearly independent coordinate vectors and their corresponding seminormal coordinates $\{s_1, \ldots, s_d\}$, we have

$$A = A \begin{pmatrix} b_1 & \cdots & b_d \end{pmatrix} \begin{pmatrix} b_1 & \cdots & b_d \end{pmatrix}^{-1} = \begin{pmatrix} s_1 & \cdots & s_d \end{pmatrix} \begin{pmatrix} b_1 & \cdots & b_d \end{pmatrix}^{-1}.$$

We typically obtain these linearly independent coordinate vectors through the computation of the DFT (without the final scaling) and the seminormal

DFT for a sufficient number of group elements. Taking the appropriate direct sum of these A matrices gives the scaling matrix S_{DFT}.

Finally, now that the coefficients are transformed to the correct seminormal ones, we may permute them once again to correspond to the row-major order of the seminormal coefficients, encoding this further rearrangement of the basis vectors as P'_{DFT}.

Example 4.20 We illustrate the computation of this transform for S_3 using the centrally primitive idempotents of $\mathbb{C}S_2$ and $\mathbb{C}S_3$ as separating elements in Appendix A. ■

Additionally, we wish to have an efficient inverse transform, so that we can recover elements of the group algebra $\mathbb{C}G$ from their matrix algebra representations. Fortunately, the block-diagonal and permutation structures of the sparse factors of the DFT matrix that Algorithms 4.16 and 4.18 produce make the computation of this efficient inverse transform straightforward.

Algorithm 4.21 Let $F = (P_1, F_{1,1}, \ldots, P_{2n-1}, \ldots, F_{2n,r'_{2n}}, P_{\text{DFT}}, S_{\text{DFT}}, P'_{\text{DFT}})$ be the list of sparse factors of $[D]_B^{\mathcal{F}}$ generated by Algorithms 4.16 and 4.18. Define

$$I = ((P'_{\text{DFT}})^T, S_{\text{DFT}}^{-1}, P_{\text{DFT}}^T, F_{2n,r'_n}^{-1}, \ldots, P_{2n-1}^T, \ldots, F_{1,1}^{-1}, P_1^T).$$ ■

Theorem 4.22 *In Algorithm 4.21 above, the product of the elements of I is the matrix representation of the inverse DFT D^{-1}:*

$$[D^{-1}]_{\mathcal{F}}^{B} = P_1^T F_{1,1}^{-1} \cdots P_{2n-1}^T \cdots F_{2n,r'_n}^{-1} P_{DFT}^T S_{DFT}^{-1} (P'_{DFT})^T.$$

Furthermore, the number of nonzero entries in each matrix is bounded above by the block sizes of the matrices in F.

Proof: Since $[D^{-1}]_{\mathcal{F}}^{B}[D]_B^{\mathcal{F}} = [D^{-1} \circ D]_B^B = I_{|G|}$, the matrix representation of D^{-1} is simply the inverse of the matrix representation of D. Hence, the inverses of the factors of $[D]_B^{\mathcal{F}}$ give a factorization of this inverse transform matrix. Furthermore, the inverses of the permutation matrices are simply their transposes, while the inverses of the block-diagonal $F_{i,j}$ and S_{DFT} matrices exhibit the same block diagonal structure. ■

4.7 Bases and Regular Representations

In order to implement these transforms on $\mathbb{C}G$ in terms of linear algebraic computations, we must select a group-element basis for $\mathbb{C}G$. In particular, we specify two important classes of group-element bases for $\mathbb{C}G$.

While a double-coset basis for $\mathbb{C}G$ gives particularly nice block-diagonal forms for the regular representations of subalgebra elements acting on $\mathbb{C}G$, we can improve upon the computation of such forms through *left-* and *right-coset bases*:

Definition 4.23 Let G be a group and $1 = G_0 < G_1 < \ldots < G_n = G$ be a chain of its subgroups, denoted $\mathcal{C}$. A *left-coset basis* for $\mathbb{C}G$ with respect to $\mathcal{C}$ is an ordered list $L = (g_1, \ldots, g_k)$ of the elements of G such that, for each $G_i \in \mathcal{C}$, L splits into sublists that correspond to the left cosets of G_i in G. Similarly, a *right-coset basis* for $\mathbb{C}G$ with respect to $\mathcal{C}$ is an ordered list $L = (g_1, \ldots, g_k)$ of the elements of G such that, for each $G_i \in \mathcal{C}$, L splits into sublists that correspond to the right cosets of G_i in G. ■

We give a straightforward algorithm for the recursive computation of such bases.

Algorithm 4.24 Let G and $\mathcal{C}$ be as above, and for each $k < n$ let T_k be a left transversal of G_k in G_{k+1}. Let $L_0 = (1)$, the identity element of G. Given L_k, construct L_{k+1} by

$$L_{k+1} = t_1 L_k \,\#\, t_2 L_k \,\#\, \cdots \,\#\, t_{|T_k|} L_k,$$

where $t_1, \ldots, t_{|T_k|}$ are the distinct elements of T_k, and where # denotes the concatenation of lists.

Similarly, for right transversals T'_k, set $L'_0 = L_0$ and construct L'_{k+1} by

$$L'_{k+1} = L'_k t_1 \,\#\, L'_k t_2 \,\#\, \cdots \,\#\, L'_k t_{|T'_k|},$$

where $t_1, \ldots, t_{|T'_k|}$ are the distinct elements of T'_k. ■

Proposition 4.25 *The L_n and L'_n produced by Algorithm 4.24 are left-coset and right-coset bases for G adapted to $\mathcal{C}$, respectively.*

Proof: Fix $k \leq n$ and let $t^i \in T_i$ for each $k < i \leq n$. Then L_k lists the elements of G_k, and so $t^n t^{n-1} \cdots t^{k+1} L_k$ is a left coset of G_k. By its construction, L_n is a concatenation of these lists of left coset elements and hence partitions into left cosets of L_k. The right-hand case is similar. ■

Example 4.26 Consider the subgroup chain $S_1 < S_2 < S_3$ for S_3. Picking $T_2 = \{1, (1\,2)\}$ and $T_3 = \{1, (1\,3), (2\,3)\}$ gives $L_1 = (1)$, $L_2 = (1, (1\,2))$, and

$$\begin{aligned} L_3 &= (1, (1\,2), (1\,3), (1\,3)(1\,2), (2\,3), (2\,3)(1\,2)) \\ &= (1, (1\,2), (1\,3), (1\,2\,3), (2\,3), (1\,3\,2)) \end{aligned}$$

as left-coset bases for this chain. Picking $T_2' = T_2$ and $T_3' = T_3$ gives the right-coset bases $L_1' = (1)$, $L_2' = (1, (1\,2))$, and

$$\begin{aligned} L_3' &= (1, (1\,2), (1\,3), (1\,2)(1\,3), (2\,3), (1\,2)(2\,3)) \\ &= (1, (1\,2), (1\,3), (1\,3\,2), (2\,3), (1\,2\,3)). \end{aligned}$$

Consider the subgroup chain $1 < \mathbb{Z}/2\mathbb{Z} < \mathbb{Z}/6\mathbb{Z}$ for $\mathbb{Z}/6\mathbb{Z}$. Picking $T_2 = \{\bar{0}, \bar{3}\}$ and $T_3 = \{\bar{0}, \bar{1}, \bar{2}\}$ gives the left-coset basis

$$L_3 = (\bar{0}, \bar{3}, \bar{1}, \bar{4}, \bar{2}, \bar{5}).$$

Since $\mathbb{Z}/6\mathbb{Z}$ is abelian, this basis equals the corresponding right-coset basis. ■

Additionally, when using bases for $\mathbb{C}G$ constructed according to Algorithm 4.24, the corresponding matrix representations of elements of $\mathbb{C}G_i \otimes \mathbb{C}G_j^o$ acting on $\mathbb{C}G$ take a particularly simple form.

Proposition 4.27 *Let G be a group, H a subgroup of G, and let B be a right-coset basis for $\mathbb{C}G$ adapted to $1 < H < G$ constructed by Algorithm 4.24. Let S be the group-element basis for $\mathbb{C}H$ used in this construction, and let T be the right transversal of H in G used. For $\alpha \in \mathbb{C}H$, consider α acting from the left on both $\mathbb{C}H$ and $\mathbb{C}G$. Then the matrix representations of these actions with respect to S and B are related by*

$$[\alpha]_B^B = \bigoplus_{i=1}^{[G:H]} [\alpha]_S^S = I_{[G:H]} \otimes [\alpha]_S^S, \tag{4.7}$$

where $I_{[G:H]}$ is the identity matrix of size $[G : H]$.

Proof: Consider first $g \in H$ and $\sigma \in G$. Then $S = \{h_1, \ldots, h_m\}$ is a basis for $\mathbb{C}H$, and $S\sigma = \{h_1\sigma, \ldots, h_m\sigma\}$ is a basis for the left $\mathbb{C}H$-module $\mathbb{C}H\sigma$. Furthermore, $gh_j = h_k$ if and only if $gh_j\sigma = h_k\sigma$, so the matrix representations of g on $\mathbb{C}H$ with respect to S and on $\mathbb{C}H\sigma$ with respect to $S\sigma$ are identical permutation matrices:

$$[g]_S^S = [g]_{S\sigma}^{S\sigma}.$$

By the partitioning of B into right cosets of H, we have

$$[g]_B^B = \bigoplus_{t\in T}[g]_{St}^{St} = \bigoplus_{t\in T}[g]_S^S = \bigoplus_{i=1}^{[G:H]}[g]_S^S$$

where the last step follows because $|T| = [G : H]$.

Now pick $\alpha \in \mathbb{C}H$, so that $\alpha = \sum_{h\in H}\alpha_h h$. By linearity,

$$[\alpha]_B^B = \sum_{h\ inH}\alpha_h[h]_B^B = \sum_{h\in H}\alpha_h\bigoplus_{i=1}^{[G:H]}[h]_S^S = \bigoplus_{i=1}^{[G:H]}\left[\sum_{h\in H}\alpha_h h\right]_S^S = \bigoplus_{i=1}^{[G:H]}[\alpha]_S^S.$$

Finally,

$$\bigoplus_{i=1}^{[G:H]}[\alpha]_S^S = I_{[G:H]}\otimes[\alpha]_S^S$$

by the definition of the Kronecker product. ∎

Since $\alpha \in \mathbb{C}H$ and $\alpha\otimes 1 \in (\mathbb{C}H\otimes\mathbb{C}K^o)$ act on $\mathbb{C}G$ from the left identically, we then have

$$[\alpha\otimes 1]_B^B = \bigoplus_{i=1}^{[G:H]}[\alpha\otimes 1]_S^S = I_{[G:H]}\otimes[\alpha\otimes 1]_S^S, \tag{4.8}$$

so that this structural decomposition applies to the tensor algebras we wish to act on $\mathbb{C}G$. Consequently, because we tend to act on the left and then on the right, we base much of the initial construction of matrix representations on a right-coset basis for G.

We now relate the matrix representation of $1\otimes\alpha^o$ with respect to a group-element basis B to that of $\alpha\otimes 1$ through an *inverse basis* for B.

Definition 4.28 Let G be a finite group, and $B = (b_1, b_2, \ldots, b_n)$ a group-element basis for $\mathbb{C}G$. Then the *inverse basis* of B is given by

$$B^I = (b_1^{-1}, b_2^{-1}, \ldots, b_n^{-1}).$$ ∎

We note that, for all $h \in G$, $[h]_B = [h^{-1}]_{B^I}$. Since B and B^I are always group-element bases, and hence are permutations of each other, the change-of-basis matrix $[1]_B^{B^I}$ is a permutation matrix. Moreover, for $h \in G$ considered as an element of $\mathbb{C}G$,

$$[1]_B^{B^I}[1]_B^{B^I}[h]_B = [1]_B^{B^I}[h]_{B^I} = [1]_B^{B^I}[h^{-1}]_B = [h^{-1}]_{B^I} = [h]_B.$$

Because this holds for all $h \in G$, which span $\mathbb{C}G$, $[1]_B^{B^I}[1]_B^{B^I} = I_{|G|}$, so $[1]_B^{B^I}$ equals its own inverse. Since $[1]_B^{B^I}$ is a permutation matrix, $([1]_B^{B^I})^{-1} = ([1]_B^{B^I})^T$, so $[1]_B^{B^I}$ is symmetric.

Example 4.29 Consider the left-coset basis L_3 of S_3 of Example 4.26:

$$L_3 = (1, (1\,2), (1\,3), (1\,2\,3), (2\,3), (1\,3\,2))$$

Its inverse basis is given by

$$L_3^I = (1, (1\,2), (1\,3), (1\,3\,2), (2\,3), (1\,2\,3))$$

which in this case coincides with the corresponding right-coset basis L_3'.

The inverse basis of the left-coset basis $L_3 = (\bar{0}, \bar{3}, \bar{1}, \bar{4}, \bar{2}, \bar{5})$ for $\mathbb{Z}/6\mathbb{Z}$ is

$$L_3^I = (\bar{0}, \bar{3}, \bar{5}, \bar{2}, \bar{4}, \bar{1}).$$

Since the left- and right-coset bases for $\mathbb{Z}/6\mathbb{Z}$ are identical, this inverse basis does not in this case equal the right-coset basis. ■

Proposition 4.30 *Let $\mathbb{C}G$ be considered as a $(\mathbb{C}G \otimes \mathbb{C}G^o)$-module, and let B be a group-element basis for $\mathbb{C}G$. Let C be the corresponding inverse basis for B, and let $P = [1]_B^C$. Then for $\alpha \in \mathbb{C}G$,*

$$[1 \otimes \alpha^o]_B^B = P([\alpha \otimes 1]_B^B)^T P. \tag{4.9}$$

Proof: Fix $g \in G$. For each $h \in G$, we have

$$\begin{aligned}[1 \otimes g^o]_B^B[h]_B = [hg]_B &= [(hg)^{-1}]_C \\ &= [g^{-1}h^{-1}]_C = [g^{-1} \otimes 1]_C^C[h^{-1}]_C = [g^{-1} \otimes 1]_C^C[h]_B.\end{aligned}$$

This result extends by linearlity to hold for all $\sum_{h\in G} \alpha_h h \in \mathbb{C}G$, so $[1 \otimes g^o]_B^B = [g^{-1} \otimes 1]_C^C$. We then have that

$$[g^{-1} \otimes 1]_C^C = ([g \otimes 1]_C^C)^{-1} = ([g \otimes 1]_C^C)^T,$$

where the last step follows because g simply permutes the elements of C, and hence its matrix representation with respect to C is a permutation matrix. Finally,

$$([g \otimes 1]_C^C)^T = (P[g \otimes 1]_B^B P^{-1})^T = P([g \otimes 1]_B^B)^T P$$

by change of basis, and because, as noted above, P is a symmetric permutation matrix.

This result extends by linearity to show that

$$[1 \otimes \alpha^o]_B^B = P([\alpha \otimes 1]_B^B)^T P$$

for all $\alpha \in \mathbb{C}G$. ■

As a consequence of this result, given the regular representations of all $g \in G$ with respect to a particular basis B consisting of the elements of G, as well as the change-of-basis matrix between B and its inverse basis, we can construct the matrix representation of any element $\alpha \in (\mathbb{C}G \otimes \mathbb{C}G^o)$ with respect to this basis B.

4.8 Computation of Double-Coset Projections

We use the above results to compute eigenspace projection operators specific to these double-coset modules. We first note that if B is a group-element basis and C its inverse basis, applying Proposition 4.30 to $[1 \otimes z_{k,j}^o]_B^B$ with $P_B^C = [1]_B^C$ gives

$$[1 \otimes z_{k,j}^o]_B^B = P_B^C([z_{k,j} \otimes 1]_B^B)^T P_B^C = P_B^C \left(\sum_\lambda \lambda P_\lambda^T \right) P_B^C = \sum_\lambda \lambda P_B^C P_\lambda P_B^C.$$

Hence, the projection matrices for the eigenspaces of $1 \otimes z_{k,j}^o$ with respect to B are given by $P_B^C P_\lambda P_B^C$, so it suffices to compute bases only for the projections of $[z_{k,j} \otimes 1]$.

We recall that if the separating elements up to $z_{k,j}$ are contained in the algebra $\mathbb{C}G_k \otimes \mathbb{C}G_{k-1}^o$, then the eigenspace projections of $z_{k,j}$ may be restricted to the double-coset modules associated with this algebra. By Proposition 4.27, we can compute these double-coset-specific projections for $z_{k,j}$ from the bases of the eigenspaces of $z_{k,j}$ acting on $\mathbb{C}G_k$ with respect to its right-coset basis B_k:

Algorithm 4.31 Let λ be an eigenvalue of $z_{k,j} \in \mathbb{C}G_k$, and let E denote a basis for the corresponding eigenspace W^λ in $\mathbb{C}G_k$. Let $s = [G : G_k]$, the index of G_k in G. Let $B = (g_1, \ldots, g_{|G|})$ be a right-coset basis for $\mathbb{C}G$ generated according to Algorithm 4.24, and let B_k be the intermediate basis for $\mathbb{C}G_k$. Let $m = 2k - 1$.

1. Compute the list $L = (p_m(g_{|G_k|}), p_m(g_{2|G_k|}), \ldots, p_m(g_{|G|}))$ and enumerate its elements L_1 through L_s.
2. For $j = 1$ to s, if $L_j = i \in I_m$, then append $\mathbf{e}_j \otimes [E]_{B_k}$ to E^i, where $\mathbf{e}_j$ is the column vector of length s with a one in the jth entry and zeroes elsewhere. ■

Proposition 4.32 *The E^i for $i \in I_{2k-1}$ computed in Algorithm 4.31 constitute the coordinates of bases for the eigenspaces of $z_{k,j} \otimes 1$ in $\mathbb{C}G$ associated with λ such that E^i is contained in the ith (G_k, G_{k-1})-double-coset submodule of $\mathbb{C}G$.*

Proof: The elements of L indicate which (G_k, G_{k-1})-double coset will contain the jth right coset of G_k. Tensoring $[E]_{B_k}$ with $\mathbf{e}_j$ is equivalent to right-multiplying E by the jth coset representative in the transversal of G_k in G, and the process that builds the E^i guarantees that the E^i contain all of the coordinates for basis vectors that lie in its constituent right cosets. ■

The computation of the projections for $1 \otimes z_{k,j}$ involve the inverse basis as well because of the introduction of the P_B^C factor in the projection.

Algorithm 4.33 Let λ be an eigenvalue of $z_{k,j} \in \mathbb{C}G_k$, and let E denote a basis for the corresponding eigenspace W^λ in $\mathbb{C}G_k$. Let $s = [G : G_k]$, the index of G_k in G. Let $B = (g_1, \ldots, g_{|G|})$ be a right-coset basis for $\mathbb{C}G$ generated according to Algorithm 4.24, let B_k be the intermediate basis for $\mathbb{C}G_k$, and let C be the inverse basis of B. Let $m = 2k$.

1. Compute the list $L = (p_m(g_{|G_k|}^{-1}), p_m(g_{2|G_k|}^{-1}), \ldots, p_m(g_{|G|}^{-1}))$ and enumerate its elements L_1 through L_s.
2. For $j = 1$ to s, if $L_j = i \in I_m$, then append $\mathbf{e}_j \otimes [E]_{B_k}$ to E^i, where $\mathbf{e}_j$ is the column vector of length s with a one in the jth entry and zeroes elsewhere. ■

Proposition 4.34 *The E^i for $i \in I_{2k}$ computed in Algorithm 4.33 constitute the coordinates of bases for the eigenspaces of $1 \otimes z_{k,j}$ in $\mathbb{C}G$ associated with λ such that $[1]_C^B E^i$ is contained in the ith (G_k, G_k)-double-coset submodule of $\mathbb{C}G$.*

Proof: As above, the construction of the E^i is a matter of collecting the $[1]_C^B(\mathbf{e}_j \otimes [E]_{B_k})$ blocks whose nonzero coordinates lie in the ith double coset of G_k and G_k. Suppose $\alpha = \sum_{g \in G_k} \alpha_g g \in \mathbb{C}G_k$. Then

$$\mathbf{e}_j \otimes [\alpha]_{B_k} = \left[\sum_{g \in G_k} \alpha_g g t_j \right]_B$$

where t_j is the jth element of the transversal T of G_k in G used to construct the right-coset basis B. Multiplication by $[1]_C^B$ then yields

$$[1]_C^B(\mathbf{e}_j \otimes [\alpha]_{B_k}) = [1]_C^B \left[\sum_{g \in G_k} \alpha_g g t_j \right]_B = \left[\sum_{g \in G_k} \alpha_g g t_j \right]_C$$
$$= \left[\sum_{g \in G_k} \alpha_g t_j^{-1} g^{-1} \right]_B = \left[\sum_{g \in G_k} \alpha_{g^{-1}} t_j^{-1} g \right]_B .$$

Thus, after this multiplication, this vector corresponds to the left coset $t_j^{-1}G_k$. We note, however, that $G_k \sigma G_k$ is a disjoint sum both of left cosets of G_k and of right cosets of G_k, so we need only determine which left cosets of G_k are contained in $G_k t_j^{-1} G_k$.

The inverse basis $B^I = (g_1^{-1}, g_2^{-1}, \ldots, g_{|G|}^{-1})$ of B partitions into left cosets of G_k, so the elements $g_{|G_k|}^{-1}, g_{2|G_k|}^{-1}, \ldots$ are "samples" of these left cosets. Hence, the elements of L determine which double coset of G_k and G_k the block $[1]_C^B(\mathbf{e}_j \otimes [E]_{B_k})$ corresponds to, so we concatenate the blocks $\mathbf{e}_j \otimes [E]_{B_k}$ according to these indices. ■

Thus, the projections of the $z_{k,j}$ acting on either side of $\mathbb{C}G$ are determined by the left action of $z_{k,j}$ on $\mathbb{C}G_k$, the double coset branching for the chain of subgroup pairs we employ in the construction of the DFT matrix factorization, and the permutation change-of-basis matrices from the basis B to its inverse basis and to the double-coset basis.

Example 4.35 We construct the projection matrices for $e_2 = \frac{1}{2}(1 + (1\,2))$ acting on $\mathbb{C}S_3$ from the left and from the right, with respect to the right-coset basis L_3' specified in Example 4.26 (which is also a double-coset basis). Note that bases for the eigenspaces of e_2 acting on $\mathbb{C}S_2$ from the left are

$$E_1 = \begin{pmatrix} 1 \\ 1 \end{pmatrix} \quad \text{and} \quad E_2 = \begin{pmatrix} 1 \\ -1 \end{pmatrix}.$$

Indexing the double cosets in S_3 by the positive integers and applying Al-

gorithm 4.31 yields $L = (1, 2, 3)$, so that

$$E_{1,1} = \begin{pmatrix} 1 \\ 1 \\ 0 \\ 0 \\ 0 \\ 0 \end{pmatrix}, \qquad E_{1,2} = \begin{pmatrix} 0 \\ 0 \\ 1 \\ 1 \\ 0 \\ 0 \end{pmatrix}, \qquad E_{1,3} = \begin{pmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 1 \\ 1 \end{pmatrix},$$

$$E_{2,1} = \begin{pmatrix} 1 \\ -1 \\ 0 \\ 0 \\ 0 \\ 0 \end{pmatrix}, \qquad E_{2,1} = \begin{pmatrix} 0 \\ 0 \\ 1 \\ -1 \\ 0 \\ 0 \end{pmatrix}, \qquad E_{2,1} = \begin{pmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 1 \\ -1 \end{pmatrix}.$$

are bases for the projections specific to the double cosets of S_2 and S_1. The projection matrices $E(E^TE)^{-1}E^T$ are then of the form

$$P_{1,1} = \begin{pmatrix} 1 \\ 1 \\ 0 \\ 0 \\ 0 \\ 0 \end{pmatrix} (2)^{-1} \begin{pmatrix} 1 & 1 & 0 & 0 & 0 & 0 \end{pmatrix} = \begin{pmatrix} \frac{1}{2} & \frac{1}{2} & 0 & 0 & 0 & 0 \\ \frac{1}{2} & \frac{1}{2} & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix},$$

$$P_{2,3} = \begin{pmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 1 \\ -1 \end{pmatrix} (2)^{-1} \begin{pmatrix} 0 & 0 & 0 & 0 & 1 & -1 \end{pmatrix} = \begin{pmatrix} 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & \frac{1}{2} & -\frac{1}{2} \\ 0 & 0 & 0 & 0 & -\frac{1}{2} & \frac{1}{2} \end{pmatrix},$$

in the right-coset basis B and thus, in this case, in the double-coset basis. Note that the only nonzero entries in each matrix occur in the rows and columns corresponding to the appropriate double coset.

For the right-action eigenspace projections of e_2, we compute $L = (1, 2, 2)$

from the inverse basis, and then construct the eigenspace bases

$$E'_{1,1} = \begin{pmatrix} 1 \\ 1 \\ 0 \\ 0 \\ 0 \\ 0 \end{pmatrix}, \quad E'_{1,2} = \begin{pmatrix} 0 & 0 \\ 0 & 0 \\ 1 & 0 \\ 1 & 0 \\ 0 & 1 \\ 0 & 1 \end{pmatrix}, \quad E'_{2,1} = \begin{pmatrix} 1 \\ -1 \\ 0 \\ 0 \\ 0 \\ 0 \end{pmatrix}, \quad E'_{2,2} = \begin{pmatrix} 0 & 0 \\ 0 & 0 \\ 1 & 0 \\ -1 & 0 \\ 0 & 1 \\ 0 & -1 \end{pmatrix}.$$

Multiplication by P, the change of basis matrix from L_3 to its inverse L'_3, gives the bases

$$E''_{1,1} = \begin{pmatrix} 1 \\ 1 \\ 0 \\ 0 \\ 0 \\ 0 \end{pmatrix}, \quad E''_{1,2} = \begin{pmatrix} 0 & 0 \\ 0 & 0 \\ 1 & 0 \\ 0 & 1 \\ 0 & 1 \\ 1 & 0 \end{pmatrix}, \quad E''_{2,1} = \begin{pmatrix} 1 \\ -1 \\ 0 \\ 0 \\ 0 \\ 0 \end{pmatrix}, \quad E''_{2,2} = \begin{pmatrix} 0 & 0 \\ 0 & 0 \\ 1 & 0 \\ 0 & -1 \\ 0 & 1 \\ -1 & 0 \end{pmatrix}.$$

These yield projection matrices of the form

$$P''_{1,2} = \begin{pmatrix} 0 & 0 \\ 0 & 0 \\ 1 & 0 \\ 0 & 1 \\ 0 & 1 \\ 1 & 0 \end{pmatrix} \begin{pmatrix} 2 & 0 \\ 0 & 2 \end{pmatrix}^{-1} \begin{pmatrix} 0 & 0 & 1 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 & 1 & 0 \end{pmatrix} = \begin{pmatrix} 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & \frac{1}{2} & 0 & 0 & \frac{1}{2} \\ 0 & 0 & 0 & \frac{1}{2} & \frac{1}{2} & 0 \\ 0 & 0 & 0 & \frac{1}{2} & \frac{1}{2} & 0 \\ 0 & 0 & \frac{1}{2} & 0 & 0 & \frac{1}{2} \end{pmatrix}.$$

In each case, these projections take the form of the blocks along the diagonals of the matrices in Example 4.15. ■

Since the computation of eigenspace bases for $z_{k,j}$ in $\mathbb{C}G_k$ is faster than in $\mathbb{C}G$, these construction algorithms allow us to compute the factored forms of the double-coset eigenspace projections more efficiently.

4.9 Conclusion

In this chapter, we have exhibited a general framework for the precomputation of a decimation-in-frequency fast Fourier transform based on eigenspace projections. We note that while the separation into double-coset specific projections and eigenspaces is not strictly necessary, it produces much

smaller block sizes in the change-of-basis matrices that Algorithm 4.16 produces, as well as smaller matrices to decompose into minimal-rank factorizations.

This algorithm works best when there are relatively few separating elements $z_{k,j}$ that are required to separate paths and when these elements suffice to separate all the paths in the character graph (so that the resulting spaces are all one-dimensional). Such is the case with the symmetric group, which makes it an ideal candidate for an initial test of this factorization algorithm. We present the specifics of this computation in the next chapter.

Chapter 5

Fast Fourier Transforms for the Symmetric Group

We combine the representation theory of S_n from Chapter 3 with the decimation-in-frequency FFT framework from Chapter 4 to develop decimation-in-frequency FFT algorithms for the symmetric group. We also describe several improvements to the general algorithm that result from the particular structure of S_n and its seminormal matrix representations.

As indicated in Chapter 3, we select the subgroup chain $\mathcal{C}$ of S_n given by

$$S_1 < S_2 < \cdots < S_n,$$

and hence the corresponding chain $\mathcal{P}$ of subgroup pairs is

$$(S_1, S_1) < (S_2, S_1) < (S_2, S_2) < (S_3, S_2) < \cdots < (S_n, S_n).$$

We also have a seminormal DFT D for S_n, namely that given in Section 3.3. In order to employ the decimation-in-frequency algorithms developed in Chapter 4, we must now specify a right-coset basis for S_n adapted to this chain $\mathcal{C}$, a branching order $\mathcal{B}(\mathcal{P})$ for the subgroup pair chain $\mathcal{P}$ and corresponding double-coset basis B, and separating elements for each S_k. Additionally, we develop some improvements to the computation of the change-of-basis factors, and specify how to construct the final permutation and scaling matrices.

Ultimately, our algorithms produce the sparse matrix factorization

$$[D]_B^{\mathcal{F}} = S_{\mathrm{DFT}} P_{\mathrm{DFT}} F_{2n-1} P_{2n-1} \cdots P_3 F_2$$

of the DFT matrix $[D]_B^{\mathcal{F}}$.

5.1 Computation of Coset Bases

To determine a right-coset basis for S_n adapted to this chain $\mathcal{C}$, we require a right transversal of S_{k-1} in S_k for $k \leq n$. We note, however, that the set of transpositions $\{1, (1\,k), (2\,k), \ldots, (k-1\;k)\}$ is both a left and a right transversal for S_{k-1} in S_k, so we take these as our coset representatives. Thus, the right transversals of S_k in S_n are of the form

$$(i_{k+1}\;k+1)(i_{k+2}\;k+2)\cdots(i_n\;n), \tag{5.1}$$

where we let each number i_m range from 1 to $m-1$ for all m between $k+1$ and n.

Moreover, these choices of coset representatives give a convenient recursive construction of the (S_k, S_k)- and (S_k, S_{k-1})-double cosets in S_n. Suppose that $K < H < G$, and that we wish to compute the double cosets of H and K in G. For $\sigma \in G$, we have that

$$\begin{aligned} H\sigma K &= \{\rho\sigma\tau \mid \rho \in H, \tau \in K\} \\ &= \{\rho\tau \cdot \tau^{-1}\sigma\tau \mid \rho \in H, \tau \in K\} \\ &= \{\rho \cdot \tau^{-1}\sigma\tau \mid \rho \in H, \tau \in K\} \end{aligned}$$

where the last step is a reindexing that follows from $K < H$. Thus, $H\sigma K$ is determined by the right cosets of H in G by all the K-conjugates of σ. Since conjugation by S_k or S_{k-1} changes only the first letter of each transposition in the elements of the form shown in Equation (5.1), the right transversal given by all such elements is a union of conjugacy classes. Hence, we can specify the corresponding double cosets with only these conjugacy classes.

Example 5.1 Example 4.26 illustrates left- and right-coset bases for S_3 constructed using these transposition transversals.

The right transversal of S_{n-1} in S_n is given by $\{1, (1\,n), \ldots, (n-1\;n)\}$. The elements conjugate under S_{n-1} are $\{1\}$ and $\{(1\,n), \ldots, (n-1\;n)\}$, and hence there are two double cosets of S_{n-1} and S_{n-1} in S_n, one equal to S_{n-1} and the other equal to $S_n - S_{n-1}$.

Likewise, under conjugation by S_{n-2}, this transversal yields the conjugacy classes $\{1\}$, $\{(1\,n), \ldots, (n-2\,n)\}$, and $\{(n-1\,n)\}$, so for $n \geq 3$ there are three double cosets of S_{n-1} and S_{n-2} in S_n, two of size $(n-1)!$ and a third of size $(n-2)(n-1)!$. ∎

This partitioning of the transversal also gives a recursive algorithm for the construction of a double-coset branching order and a double-coset basis

for $\mathbb{C}S_n$. In fact, we specify this for a general chain of subgroups of a finite group G.

Algorithm 5.2 Let G be a finite group, let $\mathcal{C}$ be the chain $1 = G_0 < G_1 < \cdots < G_n = G$ of subgroups of G, and let $\mathcal{P}$ be the associated alternating chain of subgroups pairs. For each k in $1, \ldots, n$, let T_k be a right transversal of S_{k-1} in G_k such that T_k is a union of conjugacy classes under conjugation by G_{k-1}.

Set $L = \{1\}$.

1. For k from n to 1,
   - 1.1. Replace each group element σ in L with the sequence of group elements $t_1\sigma, t_2\sigma, \ldots, t_{|T_k|}\sigma$.
   - 1.2. Partition each group-element subset in L into its conjugacy classes under G_{k-1}.
   - 1.3. If $k > 1$, partition each group-element subset in L into its conjugacy classes under G_{k-2}.

Then L is a nested set of the elements of G, such that the sets at each level correspond to double cosets for the pairs of subgroups in $\mathcal{P}$. ■

Example 5.3 We illustrate this process with S_3 and the transversals $T_2 = \{1, (1\,2)\}$ and $T_3 = \{1, (1\,3), (2\,3)\}$. Starting with $L = \{1\}$, left-multiplying by T_3 gives $L = \{1, (1\,3), (2\,3)\}$. Then partitioning by S_2-conjugacy classes produces

$$L = \{\{1\}, \{(1\,3), (2\,3)\}\}$$

and then by S_1 classes gives

$$L = \{\{\{1\}\}, \{\{(1\,3)\}, \{(2\,3)\}\}\}.$$

Next, left multiplying by T_2 gives

$$L = \{\{\{1, (1\,2)\}\}, \{\{(1\,3), (1\,2)(1\,3)\}, \{(2\,3), (1\,2)(2\,3)\}\}\},$$

and the final partitioning by S_1 conjugacy classes gives

$$L = \{\{\{\{1\}, \{(1\,2)\}\}\}, \{\{\{(1\,3)\}, \{(1\,2)(1\,3)\}\}, \{\{(2\,3)\}, \{(1\,2)(2\,3)\}\}\}\}.$$

Viewed as a tree, L gives the double-coset branching structure on S_3 depicted in Figure 4.1. ■

We note that a total order on the elements of G gives a natural order to the double cosets in this branching, where the index i of each double coset D is taken to be the least element in D, and where the index set I inherits the total order on G. Thus, this total order on G determines the double-coset basis order.

The right-coset basis generated by the (ordered) transposition transversals $\{1, (k-1\ k), \ldots, (2\,k), (1\,k)\}$ gives just such a total order on S_n. Taken together with the double-coset branching, this order determines a double-coset basis for $\mathbb{C}S_n$. One benefit of the resulting order on this basis is that the larger double cosets in S_n tend to form near the end of the order, while the first $k!$ elements in the order still give a double-coset basis for $S_k < S_n$ because 1 is the first element of each transversal.

5.2 Separating Elements

Having constructed coset bases for S_n, we now select separating elements for the FFT precomputation. We have two reasonable choices of candidates: the centrally primitive idempotents for $\mathbb{C}S_k$, $1 \leq k \leq n$, and the Jucys-Murphy elements m_k introduced in Chapter 3.

5.2.1 Centrally Primitive Idempotents

Since we seek to distinguish the paths in the multiplicity-free character graph $\Gamma(\mathcal{C})$ for S_n, a natural choice of separating elements for each k is the set of centrally primitive idempotents for $\mathbb{C}S_k$, as these distinguish between the vertices at level k of Γ.

Moreover, since the idempotents are themselves orthogonal projections, rather than have each idempotent act individually as a separating element, with the two eigenvalues 0 and 1, we consolidate them into a single central separating element z'_k whose eigenspaces are the spaces into which the idempotents project. Denote the centrally primitive idempotents for $\mathbb{C}S_k$ by $e_1^{(k)}, \ldots, e_{h_k}^{(k)}$, and let

$$z'_k = \sum_{i=1}^{h_k} i e_i^{(k)};$$

then the eigenspace associated to the eigenvalue i of z'_k is the space into which $e_i^{(k)}$ projects.

While this approach is initially appealing in its simplicity, it faces several computational problems: the number of idempotents for $\mathbb{C}S_n$ grows as

the number of partitions of n, which grows exponentially for large n (for example, $n = 20$ has 627 partitions (30) and hence S_{20} has 627 centrally primitive idempotents). Furthermore, computation of the idempotents for S_n requires precomputation of the character tables of S_n.

5.2.2 Jucys-Murphy Elements

The Jucys-Murphy elements described above in Chapter 3 provide an alternate means of specifying these representations and hence of separating these Fourier spaces. By Equation (3.2) and Proposition 3.21, the eigenvalues of the Jucys-Murphy elements $m_2, \ldots, m_k$ suffice to distinguish seminormal basis vectors for representations of S_k. Hence, projecting the frequency spaces into the eigenspaces associated with the left and right actions of these Jucys-Murphy elements accomplishes the same separation that we achieve with the idempotents.

Furthermore, the Jucys-Murphy elements make up for the deficiencies of these idempotents:

- Since the eigenvalues of m_k are the contents of boxes added to partitions of $k-1$, they range from at least $-k+1$ to at most $k+1$. Thus, the number of projection operators at each S_k stage is bounded by $2k$, rather than by the number of partitions of k.
- The matrix representations of these elements in the group element basis consist of the sum of $k-1$ transposition matrices, which themselves consist of simple, symmetric permutation matrices. Thus, the Jucys-Murphy matrices are symmetric and easy to construct.
- These matrices can be stored efficiently in a sparse format, as each contains only $k \cdot n!$ ones.

Appendix A illustrates the use of these Jucys-Murphy elements in the computation of an FFT for S_3.

5.3 Eigenvalue List Completion

Another benefit of these Jucys-Murphy elements is that their eigenvalues on a space $\mathbb{C}e_{ML}$ yield important combinatorial information about the standard tableaux indexes M and L of the space. In particular, the eigenvalue of $m_k \otimes 1$ on this space is $\mathrm{ct}(M[k])$, while that of $1 \otimes m_k$ is $\mathrm{ct}(L[k])$.

Suppose that, at stage s of Algorithm 4.18, the space B with eigenvalue list $(\lambda_1, \lambda_2, \ldots, \lambda_s)$ is one-dimensional, so that it has only one vector in its basis. Then there is a unique way to complete this eigenvalue list so that $(\lambda_1, \lambda_3, \ldots, \lambda_{2n-1})$ and $(\lambda_2, \lambda_4, \ldots, \lambda_{2n})$ yield the standard tableaux that index this space. Since these tableaux have the same shape, their corresponding lists of content values are permutations of each other. Hence, if the full lists are uniquely specified by the partials, the remaining content values λ_{s+1} through λ_{2n} must be taken from the list $(\lambda_1, \lambda_2, \ldots, \lambda_s)$.

Given two such partial lists L and M, we give an algorithm to determine the full lists that they determine.

Algorithm 5.4 Let $L = (\lambda_1, \lambda_2, \ldots, \lambda_l)$ and $M = (\mu_1, \mu_2, \ldots, \mu_m)$ be partial lists of eigenvalues for a one-dimensional eigenspace of the Jucys-Murphy elements.

1. For each $n \in \mathbb{Z}$, set $K_L(n) = \{i \mid \lambda_i = n\}$ and $K_M(n) = \{i \mid \mu_i = n\}$.
2. Set $C(n) = \min\{|K_L(n)|, |K_M(n)|\}$.
3. Set $D_L(n)$ to $K_L(n)$ with the least $C(n)$ elements removed, and likewise set $D_M(n)$ to $K_M(n)$ with the least $C(n)$ elements removed.
4. Set $D_L = \bigcup_{n\in\mathbb{Z}} D_L(n)$ and $D_M = \bigcup_{n\in\mathbb{Z}} D_M(n)$.
5. Set S_L to the list $(\lambda_j)_{j\in D_L}$, where the j values increase monotonically. Likewise, set S_M to the list $(\mu_j)_{j\in D_L}$, where the j values increase monotonically.
6. Set $\bar{L} = L\#S_M$ and $\bar{M} = M\#S_L$, where as above # represents list concatenation.

Then $\bar{L}$ and $\bar{M}$ are the unique complete eigenvalue lists for this space.

Proof: This completion algorithm relies on the hook structure of the standard tableaux. Any Ferrers diagram λ can be decomposed into a set of nested hooks, each of which with a box of content 0 at the corner, boxes of negative content value below the corner, and boxes of positive content value to the right of the corner. If a box in a tableau is the kth such box of its content in the order of boxes added to construct that tableau, then that box lies in the kth hook of the shape. Hence, a box's content value and the number of boxes with its content value added to the tableau before it completely specified its position in the shape.

Consequently, if L has l boxes of content n and M has m, they share $C(n) = \min\{l, m\}$ boxes of that content value, which are located in hooks 1 through $C(n)$. Removing the first $C(n)$ indices of these content-n boxes to form $D_L(n)$ then gives the indices of the boxes of content n in L that are not present in M. The union D_L of all such sets $D_L(n)$ then gives the indices of all the boxes in L that are not present in M.

Since these boxes indexed by D_L were added to the boxes common to L and M to form L, they can also be added to M to form the shape consisting of the union of the boxes in L and M. By the definition of a standard tableaux, adding these boxes in the order they appear in the tableau keeps the corresponding partition proper, so adding them in that same order to M keeps the partition proper at each stage and hence guarantees that the new tableau $M\#S_L$ is standard. Likewise, $L\#S_m$ is standard.

Finally, since the space that L and M index is one-dimensional, $M\#S_L$ and $L\#S_M$ are the only possible valid completions of these eigenvalue lists.■

Example 5.5 We give an example of such a completion process. Consider the standard tableaux

$$L = \begin{array}{|c|c|c|}\hline 1&2&5\\\hline 3&4&6\\\hline\end{array} \quad \text{and} \quad M = \begin{array}{|c|c|}\hline 1&4\\\hline 2&5\\\hline 3&6\\\hline\end{array}.$$

It is clear that there is precisely one way to add boxes to these tableaux so that they have a common shape equal to the union of their two shapes. The resulting tableaux are

$$\bar{L} = \begin{array}{|c|c|c|}\hline 1&2&5\\\hline 3&4&6\\\hline 7&8\\\cline{1-2}\end{array} \quad \text{and} \quad \bar{M} = \begin{array}{|c|c|c|}\hline 1&4&7\\\hline 2&5&8\\\hline 3&6\\\cline{1-2}\end{array}.$$

We construct these using Algorithm 5.4. As lists of content, these tableaux are $L = (0, 1, -1, 0, 2, 1)$ and $M = (0, -1, -2, 1, 0, -1)$. Table 5.1 presents the intermediate information constructed in this algorithm, from which we see that $D_L = \{5, 6\}$ and $D_M = \{3, 6\}$. Hence, $S_L = (2, 1)$ and $S_M = (-2, -1)$, so that

$$\bar{L} = L\#S_M = (0, 1, -1, 0, 2, 1, -2, -1),$$
$$\bar{M} = M\#S_L = (0, -1, -2, 1, 0, -1, 2, 1).$$

These content lists then correspond to the completed tableaux of shape $(3, 3, 2)$ above. ■

n	$K_L(n)$	$K_M(n)$	$C(n)$	$D_L(n)$	$D_M(n)$
-2	$\emptyset$	$\{3\}$	0	$\emptyset$	$\{3\}$
-1	$\{3\}$	$\{2,6\}$	1	$\emptyset$	$\{6\}$
0	$\{1,4\}$	$\{1,5\}$	2	$\emptyset$	$\emptyset$
1	$\{2,6\}$	$\{4\}$	1	$\{6\}$	$\emptyset$
2	$\{5\}$	$\emptyset$	0	$\{5\}$	$\emptyset$

Table 5.1: Table of eigenvalue completion data from Algorithm 5.4 applied to the standard tableaux L and M of Example 5.5.

We note that, since we have already reduced this space to one dimension, there will be no further separation of the space by the projection operators: the space is known to be in the eigenspaces of the projections associated to the remaining eigenvalues in the completed list. Hence, we can modify Algorithm 4.16 to bypass projections for such spaces altogether. We still need the completed list of eigenvalues, however, to index the space later.

5.4 Computation of Final Permutation Matrix

By Proposition 3.23, the last letter order on standard tableaux is also determined by the lexicographic order on their associated content lists, and hence by the lists of Jucys-Murphy eigenvalues. Moreover, these content lists determine the shapes λ of the tableaux, and tableaux of the same shape have the same content lists but in a different order. The following algorithm then yields the permutation matrix that puts the Fourier coefficients into the order corresponding to the row-major order in the matrix algebra.

Algorithm 5.6 Given the list of Jucys-Murphy eigenvalue lists for the basis vectors resulting from the first part of Algorithm 4.18,

1. Tag each list L of eigenvalues with the index of the basis element in the original order.
2. Separate each list L into left and right eigenvalue lists L_L and L_R.
3. Group the pairs of lists by sorted L_L list.
4. Concatenate L_R and L_L and reverse this list to form S, and sort each group by descending lexicographic order on the S lists.

This process puts the coefficients in row-major order by matrix block. We have then permuted the list of coefficient indices, which we use to construct a permutation matrix P_{DFT} that permutes the coefficients into this order. ■

In particular, this row-major order makes converting from the vector of seminormal coordinates to the seminormal matrix representation convenient, and vice versa. Furthermore, this irreducible-representation partitioning process gives the degrees of each of the irreducible representations, which we require to convert between the matrix and the vector forms.

5.5 Computation of Scaling Matrix

Because of Proposition 2.8 and the fact that the content list for a tableau completely determines that tableau, the diagonal elements in the matrix algebra are also determined completely, regardless of the choice of seminormal DFT. Hence, rescaling the diagonal coefficients so that they are 1 on the identity element of S_n gives the correct scaling for all of these diagonal coefficients.

Additionally, Algorithm 3.26 specifies the off-diagonal matrix elements in terms of the diagonal ones, and hence fixes the remaining degrees of freedom in our choice of DFT. Thus, we can construct the appropriate scaling matrix for this seminormal DFT without computing its value on the generators of S_n beforehand. This scaling process involves three stages:

- Scaling the diagonal elements to be 1 on the identity.
- Constructing the seminormal matrix representations of the generators from the now-correctly scaled diagonal coefficients and Algorithm 3.26.
- Determining the remaining scalings by making the images of other group elements consistent with the corresponding products of the generators' matrices.

We then have the following algorithm to generate this scaling matrix S_{DFT}.

Algorithm 5.7 Given the list of lists of left and right eigenvalues generated by Algorithm 5.6, and the partial matrix factorization $P_{\text{DFT}}, F_{2n-1}, \ldots, F_2$ of the DFT matrix generated by the first two steps in Algorithm 4.18,

1. Let S be the coefficient scaling matrix, initialized to the $n! \times n!$ identity matrix. Index its diagonal elements by $s_{\lambda,ij}$, where $s_{\lambda,ij}$ scales the ijth Fourier coefficient in block λ.

2. Let L be the list of indices for coefficients that we have yet to scale; initialize L to $1, \ldots, n!$.

3. Ensure that the scalings for the image of $1 \in \mathbb{C}S_n$ are correct.

   3.1. Since 1 maps to the identity in the block matrix algebra, its correct image contains ones down the diagonal coordinates and zeroes elsewhere. Hence, if the coordinate $c_{\lambda,ij}$ is nonzero, we set $s_{\lambda,ij} = 1/c_{\lambda,ij}$.

   3.2. Remove the indices of the nonzero coordinates from L.

4. Construct the seminormal matrix representations of the generating transpositions $(1\,2)$, $(2\,3)$, $\ldots$, $(n-1\ \ n)$ from the coefficients under the partial factorization that lie on the matrix diagonal, which we now can scale correctly because they are covered by the scalings for the identity. Thus, for each k such that $1 \leq k < n$,

   4.1. Rewrite the image of $(k\ k+1)$ under the partial factorization,

   $$SP_{\text{DFT}}F_{2n-1}P_{2n-1}\cdots F_2[(k\ k+1)]_B,$$

   in block diagonal form to create the matrix C_k, with elements denoted $c_{\lambda,ij}$.

   4.2. Initialize a matrix σ_k to all zeros to store the correct seminormal representation of $(k\ k+1)$.

   4.3. For each nonzero element $c_{\lambda,ij}$ in C,

      4.3.1. If $i = j$, $c_{\lambda,ij}$ is on the diagonal and is scaled correctly. Hence, $\sigma_{\lambda,ij} = c_{\lambda,ij}$.

      4.3.2. If $i \neq j$, $c_{\lambda,ij}$ is off the diagonal, so $\sigma_{\lambda,ij} = 1 + c_{\lambda,ii}$.

      Store the σ_k thus generated.

   4.4. For each nonzero off-diagonal element of σ_k, set $s_{\lambda,ij} = \sigma_{\lambda,ij}/c_{\lambda,ij}$.

   4.5. Remove the scaled row indices from L.

5. We now compute the remaining scaling coefficients. While L is nonempty,

   5.1. Let i be the first index remaining in L. Compute the ith row of the partial factorization product by $\mathbf{e}_i^T \cdot SP_{\text{DFT}}F_{2n-1}\cdots F_2$, and let j be the index of the first nonzero entry in this row.

   5.2. Determine the permutation ρ such that $[\rho]_B = \mathbf{e}_j$, and decompose ρ into transpositions.

5.3. Compute the correct seminormal representation σ of ρ from the appropriate products of the σ_k matrices.

5.4. Compute the partial factorization image of ρ by

$$C = SP_{\mathrm{DFT}}F_{2n-1}P_{2n-1}\cdots F_2[\rho]_B.$$

5.5. For the nonzero coordinates $c_{\lambda,ij}$ corresponding to rows that have not been scaled yet, set $s_{\lambda,ij} = \sigma_{\lambda,ij}/c_{\lambda,ij}$.

5.6. Remove the scaled coefficient indices from L.

Set S_{DFT} to S. ■

Finally, we note that we do not need a second permutation matrix P'_{DFT} because the separated eigenspaces of the Jucys-Murphy elements are all one-dimensional and hence are put into the correct order by P_{DFT}. Thus, taken all together, we have the following procedure:

Algorithm 5.8 The application of Algorithms 4.18, 5.6, and 5.7 with the double-coset basis B determined by Algorithm 5.2 generates the sparse matrix factorization

$$[D]_B^{\mathcal{F}} = S_{\mathrm{DFT}}P_{\mathrm{DFT}}F_{2n-1}P_{2n-1}\cdots P_3F_2$$

of the DFT matrix $[D]_B^{\mathcal{F}}$ for the seminormal DFT $D : \mathbb{C}S_n \to \bigoplus_{\lambda\vdash n}\mathbb{C}^{d_\lambda\times d_\lambda}$ for S_n specified by Algorithm 3.26. ■

Chapter 6

Initial Implementation and Results

6.1 *Mathematica* Implementation

For the development of a prototype implementation of these decimation-in-frequency algorithms for the symmetric group, the symbolic computing program *Mathematica* 5.0 was used. This platform offers several advantages:

- The computational framework that *Mathematica* presented was more familiar than those of other symbolic computation programs such as Maple. *Mathematica* offers both procedural programming and functional programming, as well as sophisticated pattern matching. Furthermore, *Mathematica* is built around list structures and hence carries out list-processing algorithms efficiently.
- *Mathematica* provides calculations in exact arithmetic, whereas MATLAB natively supports only floating-point calculations.
- Like MATLAB, *Mathematica* supports sparse representations of vectors and matrices and fast algorithms for computations with sparse matrices.
- *Mathematica* features the *Combinatorica* package, which provides sophisticated, efficient functionality for working with permutations and permutations groups. Moreover, Pemmaraju and Skiena (24) document this package extensively. Since we are developing FFTs first for the symmetric group, this seems a natural set of features to employ.

n	3	4	5	6
Time/s	0.12	0.62	9.48	1710

Table 6.1: Precomputation times for $n = 3$ to 6.

- *Mathematica* also provides excellent features for importing and exporting data and graphics conveniently.

The *Mathematica* code presented in Appendix C implements Algorithms 4.18, 4.16, 5.6 and 5.7, as well as the left-, right-, and double-coset basis orders described in Section 5.1.

6.2 Precomputation

Most of the computational testing of this implementation was performed on a 900 MHz Pentium III processor with 256 MB of RAM. To date, we have been able to compute the factorization of the DFT matrix for S_n for $n \leq 6$. Table 6.1 lists the running times for the precomputation of the transform according to this algorithm. In particular, the increase in running time by a factor of 180 from $n = 5$ to $n = 6$ seems to indicate that this precomputation requires $O((n!)^3)$ time.

The two major expensive steps in the computation of the transform are in Algorithm 4.16, and are the multiplication of the $E^{\lambda,i}$ and the D_j, and the computation of the minimal rank decompositions of these products by row reduction into echelon form. Suppose H and K are the left and right subgroups of G under consideration. Because the $E^{\lambda,i}$ and the D_j pertaining to the ith double coset of H and K both have total rank equal to the size of the ith double-coset module, and because their nonzero entries are found entirely in the coefficients associated with this double coset, the computation of the products takes at most $\sum_{g \in T} |HgK|^3$ operations, where T is a double-coset transversal of H and K in G. When H and K are small, this reduces the time required for the multiplication of these products significantly in the early stages of the precomputation. When $H = G$, however, this yields on the order of $|G|^3 = (n!)^3$ operations for the symmetric group S_n.

The minimal rank decompositions also require extensive precomputation. Consider a space D_j of dimension s to be decomposed according to its double-coset specific projections. If the projection has rank r, then we must decompose an $r \times s$ matrix via row reduction, which takes $O(s^2(r+s))$

operations. Hence, to decompose these products for all the projections requires $O(s^2(|HgK| + 2ks)) \approx O(s^2|HgK|)$ operations, where HgK is the double coset in which our space lies. Assuming, then, that $s \approx n$, we require

$$\sum_{g \in T} \frac{|HgK|}{n}(n^2|HgK|) = n \sum_{g \in T} |HgK|^2$$

operations to decompose these products. As in the computation of the product matrices themselves, the worst case occurs when $H = G$, so that G itself is the only double coset in G. In this case, these row reductions take $O(n(n!)^2)$ operations to carry out.

Our eigenvalue completion techniques detailed in Section 5.3 afford significant increases in efficiency, particularly in the last stages of the computation. In fact, in the computation of the last change of basis factor F_{2n-1}, 40–50% of the spaces can be skipped because their projections are already known. Additionally, once we have completely decomposed a basis into smaller bases by eigenspace projections, we need not process that basis any longer, and can move to the next one. Likewise, once we have determined r eigenvectors for a projection of rank r, we have a full basis for its associated eigenspace, and so we can remove that projection from the set that acts on subsequent bases. These eigenvalue list, basis, and projection culling techniques improve the efficiency of the transform precomputation by approximately an order of magnitude.

6.3 Evaluation

More important than the efficiency of the precomputation of the transform is the efficiency of its evaluation. We note that we can measure this efficiency in several different ways. Clausen and Baum (6) measure complex additions and multiplications separately, and add them together to obtain a total count of operations, while Maslen (18) considers one complex multiplication with one complex addition a single complex operation. In either computational model, additions may incorporate sign changes, and so encompass any operation of the form $\pm z_1 \pm z_2$. Furthermore, Maslen considers the complexity T_G of a transform on G as well as its reduced complexity $t_G = \frac{1}{|G|} T_G$. We tabulate these operation counts for our transform in Table 6.2 and compare them with the operation counts for Maslen's decimation-in-time algorithm. We also depict the sparseness of the factorization in Figure 6.1. We see that, according to Maslen's definition of an

n	$\oplus$	$\otimes$	t_n^{full}	t_n^{DIF}	t_n^M	$\frac{1}{2}n(n-1)$
3	14	4	4.7	2.7	2.7	3
4	112	42	18.8	5.3	5.4	6
5	966	424	87.9	8.8	9.1	10
6	9278	4631	486.4	13.8	13.6	15

Table 6.2: Operation counts for the evaluation of the decimation-in-frequency FFT. In addition to the number of addition and multiplications $\oplus$ and $\otimes$, respectively, we compare our reduced complexity to that of Maslen (18). Here, t_n^{DFT} denotes the reduced complexity of our decimation-in-frequency algorithm for S_n, while t_n^M denotes Maslen's reduced complexity, and t_n^{full} the reduced complexity of the full DFT matrix. In each case, the reduced complexities are below the bound of $\frac{1}{2}n(n-1)$ that Maslen conjectures holds for his algorithm for all n.

operation, the operation counts for the decimation-in-frequency algorithm are approximately the same as those for Maslen's decimation-in-time algorithm, and in some cases are better.

Additionally, we compute similar operation counts for the inverse DFT that we obtain from Algorithm 4.21. In fact, the sizes of the blocks in the block-diagonal F_i factors bound the number of operations for both the transform and its inverse, since their factors have the same block-diagonal structure. Moreover, any 1×1 blocks are ones and hence are negligible. Table 6.3 presents both these inverse-FFT operation counts and the block-diagonal bounds. We also note that many of the blocks in the inverse transform factors contain fractions with the same denominators, so that a common scaling of each factor would yield more coefficients of ± 1 and hence fewer multiplications. Such a scaling would correspond to the $\frac{1}{N}$ factor frequently introduced into the DFT for $\mathbb{Z}/N\mathbb{Z}$.

As a result of these operation counts, we state the following conjecture about the efficiency of our decimation-in-frequency algorithm:

Conjecture 6.1 *The complexity of the evaluation of the decimation-in-frequency FFT for S_n computed by Algorithm 5.8 is $O(n^2 n!)$. The complexity of the inverse transform is also $O(n^2 n!)$.* ■

As discussed above, the establishment of this result would place the complexity of this FFT in the same class as Maslen's decimation-in-time algorithm (18), which requires $\frac{3}{4}n(n-1)n!$ operations to evaluate and which is to date the most efficient FFT known for S_n.

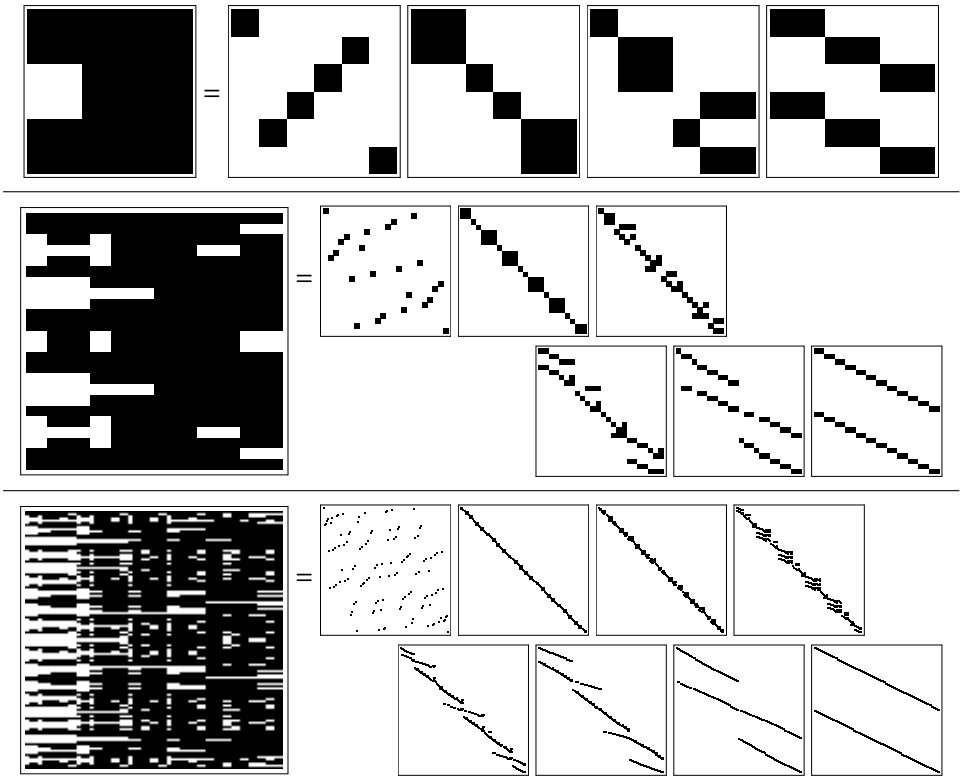


Figure 6.1: Graphical representation of DFT matrix factorization for $n = 3, 4, 5$. Each large box represents a matrix, and each black square a nonzero entry. The full DFT matrices on the left are relatively dense, while the factors on the right are sparse.

n	$\oplus$	$\otimes$	t_n^{full}	t_n^{DIF}	t_n^{block}	n^2
3	14	30	5.3	5.0	5.7	9
4	124	232	19.3	9.7	11.5	16
5	1170	1922	88.3	16.1	18.3	25
6	11884	17683	486.7	24.6	31.0	36

Table 6.3: Operation counts for the evaluation of the decimation-in-frequency inverse FFT, and complexity bounds from the block sizes. Here, t_n^{full} is the reduced complexity of the full inverse matrix, t_n^{DIF} that of the factored inverse transform, and t_n^{block} that of the bound given by the block sizes. We note that the last two are bounded above by n^2.

n	$(n!)^2$	$F_n = 2n^2 n!$	$M_n = \sum_{\lambda \vdash n} d_\lambda^3$	$(n!)^2/(F_n + M_n)$
3	36	108	10	0.31
4	576	768	64	0.69
5	14400	6000	596	2.18
6	518400	51840	8056	8.66
7	2.54×10^7	4.94×10^5	1.30×10^5	40.6
8	1.63×10^9	5.16×10^6	2.53×10^6	211
9	1.32×10^{11}	5.88×10^7	5.98×10^7	1110
10	1.32×10^{13}	7.25×10^8	1.72×10^9	5390

Table 6.4: Comparison of costs for group algebra multiplication and FFT-based matrix algebra multiplication. The cost for a naive element-by-element calculation in the group algebra is $(n!)^2$, while that of the forward and inverse FFTs is $F_n = 2n^2 n!$ and of the matrix algebra multiplication is $M_n = \sum_{\lambda \vdash n} d_\lambda^3$. For $n \geq 5$, performing the FFT and multiplying in the matrix algebra is more efficient than multiplying in the group ring. By $n = 10$, the FFT-based multiplication is faster than that in the group algebra by a factor of 5000. Character degrees taken from James and Kerber (14: App I.A).

6.4 Multiplication and Convolution

Because the elements of the matrix algebra $\bigoplus_{\lambda \vdash n} \mathbb{C}^{d_\lambda \times d_\lambda}$ consist of direct sums of $d_\lambda \times d_\lambda$ matrices, multiplication of generic elements of the algebra requires

$$\sum_{\lambda \vdash n} d_\lambda^3 \tag{6.1}$$

complex operations. Because $\sum_{\lambda \vdash n} d_\lambda^2 = |S_n| = n!$, we have that this sum is already bounded above by $(n!)^{3/2}$. Since multiplication of generic group algebra elements corresponds to a convolution of their coefficients and hence takes $|S_n|^2 = (n!)^2$ operations, this multiplication in the matrix algebra is automatically more efficient than in the group algebra. With an efficient FFT to convert between the two algebras, multiplication of group algebra elements is made efficient: to compute $\alpha\beta$ for $\alpha, \beta \in \mathbb{C}S_n$, we instead compute $D^{-1}(D(\alpha)D(\beta))$. Table 6.4 shows the operation counts for each for $n \leq 10$, assuming an FFT in $\frac{1}{2}n^2 n!$ operations and an inverse FFT in $n^2 n!$ operations.

By $n = 9$, the cost of the matrix algebra multiplication itself accounts for the majority of the total costs of the FFT-based multiplication. In par-

ticular, by this point, the maximum degree of an irreducible character for S_n, $\max_{\lambda \vdash n}$, is much greater than $2n^2$. Hence, for sufficiently large n, the complexity of the FFT-based multiplication is bounded by

$$2n^2 n! + \sum_{\lambda \vdash n} d_\lambda^3 \leq 2n^2 n! + \sum_{\lambda \vdash n} d_\lambda^2 \max_{\lambda \vdash n} d_\lambda = 2n^2 n! + \max_{\lambda} d_\lambda \sum_{\lambda \vdash n} d_\lambda^2$$
$$= (2n^2 + \max_{\lambda \vdash n} d_\lambda) n! \leq 2n! \max_{\lambda \vdash n} d_\lambda.$$

Thus, a lower bound for the ratio of the operation counts of the two methods is $n!/2 \max_{\lambda \vdash n} d_\lambda$. Since $\max_{\lambda \vdash n} d_\lambda$ grows much more slowly than $n!$ does, this result guarantees the efficiency of the FFT-based multiplication for arbitrarily large n.

Chapter 7

Future Directions and Conclusions

While we have established a general framework for decimation-in-frequency fast Fourier transforms and have applied it to obtain an initial implementation of a fast transform for S_n that we conjecture to be of $O(n^2 n!)$ complexity, there are still many directions for future research in this project. We indicate several of them below.

7.1 Double-Coset Bases and Module Decompositions

The double-coset basis chosen for S_n, or for a general finite group G, is selected arbitrarily. Moreover, because we employ row reduction to determine our bases for the eigenspaces, the basis vectors we obtain depend highly on the chosen double-coset order. If we continue to use such row-reduction techniques, then, it is worthwhile to investigate how different choices of double-coset basis for the same branching order affect the factorization we obtain.

As we remarked in Chapter 4, the double-coset modules for each subgroup pair of G decompose into direct sums of tensor products of irreducible representations for those subgroups. The identification of modules that are isomorphic in this sense would be a first step in determining a double-coset basis for G that produces identical blocks in the change-of-basis transform factors. To this end, Appendix B presents a tabulation of these decompositions for the double-coset modules of S_n.

7.2 Row Reduction and Choice of Basis

Related to these concerns over the double-coset basis is the issue of how to perform the minimum-rank decomposition that produces the eigenbases at each stage. Although the row reduction into echelon form in use in the current implementation is easy to describe and implement, and has the advantage of producing factors with many 1s and relatively few nonzero entries, it appears to have no natural relation to the structure of the spaces themselves. The determination of a more natural choice of basis for these eigenspaces could lead to their more efficient computation or to identical blocks in the factorization matrices. Such a new choice might also indicate how to project the bases into the appropriate eigenspaces without actually carrying out the projections themselves. This, in turn, would improve the efficiency of the precomputation algorithm significantly.

7.3 Efficiency of Precomputation

To compute transforms for the symmetric group S_n for much higher values of n, or for any finite group G of comparable order, we require both a reformulation of our initial implementation of the precomputation and more sophisticated computational techniques. In particular, attempts to calculate a transform for S_7 were hindered primarily by space restrictions, not time: the basis and projection data occupied most of the available virtual memory of the machine, and eventually the *Mathematica* kernel aborted the computation because of lack of memory. The computation proceeded through the decomposition by S_6, however, and based on the internal timing did so in a reasonable amount of computation time. Hence, precomputing projection operators and saving data to disk rather than keeping all structures in memory seem two reasonable first steps towards a more memory-efficient implementation of this algorithm.

Additionally, we may be hampered by the projection-based nature of this algorithm, particularly in the later stages of the precomputation. At each stage of the precomputation, we must compute the product of each basis vector with at least one projection matrix. In the last stage, where we often have dense basis vectors, the computation of such products requires at least $|G|^2$ operations. At present, it is not immediately clear how to improve the efficiency of such algorithms while remaining within our projection-based framework.

7.4 Efficiency of Evaluation

One key issue that future research should address is proving bounds on the complexity of our fast Fourier transform for S_n. Since we employ many of the same combinatorial tools that Maslen (18) uses in his decimation-in-time fast Fourier transform for S_n, such as paths through character graphs and Young tableaux, it may be possible to use some of his results to bound our complexity. In fact, because of the similar complexities of evaluation between our fast transform and Maslen's, it may be that the two transforms are equivalent, or at least closely related. If this relation can be shown, it may then be possible to adopt the same $O(n^2 n!)$ bounds that Maslen proves for his transform.

7.5 MATLAB and GAP Implementations

While *Mathematica* has provided an excellent platform for prototyping this algorithm, MATLAB and GAP are used more extensively than *Mathematica* is for signal processing and computational algebra, respectively. MATLAB in particular presents a natural platform on which to develop a more robust implementation of these algorithms, as it allows more careful control over memory management and algorithmic complexity while still providing high-level tools from linear algebra. Alternately, any GAP implementation may benefit from the use of AREP, a GAP package used to process representations of algebras abstractly. AREP is currently maintained as part of the SPIRAL project (25), which concerns the automatic generation of structured representations of matrices associated with linear digital signal processing transforms. One potential difficulty is that AREP is incompatible with the most recent versions of GAP, so some work is likely required to use it effectively in this context.

7.6 Parallel Implementations

These decimation-in-frequency algorithms seem ideal for parallel computation: the fundamental operation in the computation of these transforms is the successive separation of bases for $\mathbb{C}G$ into different eigenspaces, so each set of basis elements can be processed independently of the others. A distributed computing approach may be one solution to the space and time inefficiencies of the current precomputation algorithm.

Appendix A

Computational Examples

To illustrate the techniques presented in Chapters 4 and 5, we carry out the major steps in the computation of a fast Fourier transform for S_3 using both group algebra idempotents and Jucys-Murphy elements. In each case, $G = S_3$, the subgroup chain $\mathcal{C}$ is given by $S_1 < S_2 < S_3$, and the corresponding chain $\mathcal{P}$ of subgroup pairs is

$$(S_1, S_1) < (S_2, S_1) < (S_2, S_2) < (S_3, S_2) < (S_3, S_3).$$

Applying Algorithm 4.24 with the right transversals $T_2 = \{1, (1\,2)\}$ and $T_3 = \{1, (2\,3), (1\,3)\}$ yields the right-coset basis

$$R = \{1, (1\,2), (2\,3), (1\,2\,3), (1\,3), (1\,3\,2)\},$$

which has inverse basis

$$R' = \{1, (1\,2), (2\,3), (1\,3\,2), (1\,3), (1\,2\,3)\}.$$

Applying Algorithm 5.2 with the same transversals and this order given by the right-coset basis produces the double-coset branching

$$\mathcal{B}(\mathcal{P}) = \{\{\{\{1\}, \{(1\,2)\}\}\}, \{\{\{(2\,3)\}, \{(1\,2\,3)\}\}, \{\{(1\,3)\}, \{(1\,2\,3)\}\}\}\}.$$

and corresponding double-coset order B equal to B_R. For convenience, we also index these double cosets by the positive integers.

	Cycle Type	
	$(1,1)$	(2)
χ_1	1	1
χ_2	1	-1

	Cycle Type		
	$(1,1,1)$	$(2,1)$	(3)
χ_1	1	1	1
χ_2	2	0	-1
χ_3	1	-1	1

Table A.1: Character tables for S_2 and S_3 used in the computation of the centrally primitive idempotents.

A.1 $\mathbb{C}S_3$ with Idempotents

We first approach the decomposition of $\mathbb{C}S_3$ through the centrally primitive idempotents of $\mathbb{C}S_2$ and $\mathbb{C}S_3$. From the idempotent formula

$$e_i = \frac{d_i}{n} \sum_{g \in G} \chi_i(g^{-1}) g, \tag{A.1}$$

and from the character tables of S_2 and S_3 given in Table A.1, the idempotents for $\mathbb{C}S_2$ are

$$e_1^{(2)} = \frac{1}{2}(1 + (1\,2)), \qquad e_2^{(2)} = \frac{1}{2}(1 - (1\,2)),$$

and those for $\mathbb{C}S_3$ are

$$\begin{aligned} e_1^{(3)} &= \frac{1}{6}(1 + (1\,2) + (1\,3) + (1\,3\,2) + (2\,3) + (1\,2\,3)), \\ e_2^{(3)} &= \frac{1}{3}(2 - (1\,3\,2) - (1\,2\,3)), \\ e_3^{(3)} &= \frac{1}{6}(1 - (1\,2) - (1\,3) + (1\,3\,2) - (2\,3) + (1\,2\,3)). \end{aligned}$$

These idempotents are themselves projections into their eigenspaces of eigenvalue 1, so we need only determine bases for these eigenspaces. With respect to the $\mathbb{C}S_2$-basis R_2 contained in R, the matrix representations of these idempotents are

$$[e_1^{(2)} \otimes 1]_{R_2}^{R_2} = \frac{1}{2}\begin{pmatrix} 1 & 1 \\ 1 & 1 \end{pmatrix} \quad \text{and} \quad [e_2^{(2)} \otimes 1]_{R_2}^{R_2} = \frac{1}{2}\begin{pmatrix} 1 & -1 \\ -1 & 1 \end{pmatrix},$$

and hence suitable eigenvectors are $\begin{pmatrix} 1 & 1 \end{pmatrix}^T$ and $\begin{pmatrix} 1 & -1 \end{pmatrix}^T$. The computations to determine the eigenspaces specific to the (S_2, S_1)- and (S_2, S_2)-

double cosets are the same as in Example 4.35, and so are

$$[E_1^{1,1}]_B = \begin{pmatrix} 1 \\ 1 \\ 0 \\ 0 \\ 0 \\ 0 \end{pmatrix}, \qquad [E_1^{1,2}]_B = \begin{pmatrix} 0 \\ 0 \\ 1 \\ 1 \\ 0 \\ 0 \end{pmatrix}, \qquad [E_1^{1,3}]_B = \begin{pmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 1 \\ 1 \end{pmatrix},$$

$$[E_1^{2,1}]_B = \begin{pmatrix} 1 \\ -1 \\ 0 \\ 0 \\ 0 \\ 0 \end{pmatrix}, \qquad [E_1^{2,2}]_B = \begin{pmatrix} 0 \\ 0 \\ 1 \\ -1 \\ 0 \\ 0 \end{pmatrix}, \qquad [E_1^{2,3}]_B = \begin{pmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 1 \\ -1 \end{pmatrix}, \qquad \text{(A.2)}$$

and

$$[E_2^{1,1}]_B = \begin{pmatrix} 1 \\ 1 \\ 0 \\ 0 \\ 0 \\ 0 \end{pmatrix}, \quad [E_2^{1,2}]_B = \begin{pmatrix} 0 & 0 \\ 0 & 0 \\ 1 & 0 \\ 0 & 1 \\ 0 & 1 \\ 1 & 0 \end{pmatrix}, \quad [E_2^{2,1}]_B = \begin{pmatrix} 1 \\ -1 \\ 0 \\ 0 \\ 0 \\ 0 \end{pmatrix}, \quad [E_2^{2,2}]_B = \begin{pmatrix} 0 & 0 \\ 0 & 0 \\ 1 & 0 \\ 0 & -1 \\ 0 & 1 \\ -1 & 0 \end{pmatrix}. \qquad \text{(A.3)}$$

In each case, the associated inverse matrices are

$$(E^T E)^{-1} = \left(\tfrac{1}{2}\right) \quad \text{or} \quad (E^T E)^{-1} = \frac{1}{2}\begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}.$$

Similarly, from the matrix representations of the $\mathbb{C}S_3$ idempotents, we determine eigenspace bases to be

$$[E_3^1]_B = \begin{pmatrix} 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 1 \end{pmatrix}, \quad [E_3^2]_B = \begin{pmatrix} 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 \\ 0 & 0 & -1 & 0 \\ -1 & 0 & 0 & 0 \\ 0 & 0 & 0 & -1 \\ 0 & -1 & 0 & 0 \end{pmatrix}, \quad [E_3^3]_B = \begin{pmatrix} 1 \\ -1 \\ -1 \\ 1 \\ -1 \\ 1 \end{pmatrix}, \quad \text{(A.4)}$$

with inverse matrices

$$(([E_3^1]_B)^T[E_3^1]_B)^{-1} = (([E_3^3]_B)^T[E_3^3]_B)^{-1} = \left(\tfrac{1}{6}\right),$$
$$(([E_3^2]_B)^T[E_3^2]_B)^{-1} = \frac{1}{3}\begin{pmatrix} 2 & 0 & -1 & 0 \\ 0 & 2 & 0 & -1 \\ -1 & 0 & 2 & 0 \\ 0 & -1 & 0 & 2 \end{pmatrix}.$$

We now iterate our algorithm with these projections. We start with the bases

$$B_0 = \left(\begin{pmatrix}1\\0\\0\\0\\0\\0\end{pmatrix}, \begin{pmatrix}0\\1\\0\\0\\0\\0\end{pmatrix}, \begin{pmatrix}0\\0\\1\\0\\0\\0\end{pmatrix}, \begin{pmatrix}0\\0\\0\\1\\0\\0\end{pmatrix}, \begin{pmatrix}0\\0\\0\\0\\1\\0\end{pmatrix}, \begin{pmatrix}0\\0\\0\\0\\0\\1\end{pmatrix}\right),$$

indexed by $S_0 = (1,2,3,4,5,6)$, the integer double coset indices in our branching order. Applying the successor function s_1 to the elements of S_0 gives $S_0' = (1,1,2,2,3,3)$. Combining blocks with identical lists then yields

$$B_{1,0} = \left(\begin{pmatrix}1&0\\0&1\\0&0\\0&0\\0&0\\0&0\end{pmatrix}, \begin{pmatrix}0&0\\0&0\\1&0\\0&1\\0&0\\0&0\end{pmatrix}, \begin{pmatrix}0&0\\0&0\\0&0\\0&0\\1&0\\0&1\end{pmatrix}\right),$$

and pruning S_0' gives $S_{1,0} = (1,2,3)$. The permutation of these basis vectors to form new double-coset blocks is trivial, so the permutation matrix is as well, and we hence do not include it in the factorization. We now apply the factored projection matrices to these bases in $B_{1,0}$: for example,

$$([E_1^{1,1}]_B)^T(B_{1,0})_1 = \begin{pmatrix}1&1&0&0&0&0\end{pmatrix}\begin{pmatrix}1&0\\0&1\\0&0\\0&0\\0&0\\0&0\end{pmatrix} = \begin{pmatrix}1&1\end{pmatrix} = (1)\begin{pmatrix}1&1\end{pmatrix},$$

so the new eigenbasis is

$$[E_1^{1,1}]_B(([E_1^{1,1}]_B)^T[E_1^{1,1}]_B)^{-1}\,(1) = \frac{1}{2}\begin{pmatrix}1\\1\\0\\0\\0\\0\end{pmatrix}.$$

Applying all the projections in Equation (A.2) gives the first factor

$$F_1 = \begin{pmatrix}1 & 1 & & & & \\ 1 & -1 & & & & \\ & & 1 & 1 & & \\ & & 1 & -1 & & \\ & & & & 1 & 1 \\ & & & & 1 & -1\end{pmatrix}$$

as well as a new eigenbasis

$$B_1 = \left(\begin{pmatrix}\frac{1}{2}\\ \frac{1}{2}\\ 0\\ 0\\ 0\\ 0\end{pmatrix}, \begin{pmatrix}\frac{1}{2}\\ -\frac{1}{2}\\ 0\\ 0\\ 0\\ 0\end{pmatrix}, \begin{pmatrix}0\\ 0\\ \frac{1}{2}\\ \frac{1}{2}\\ 0\\ 0\end{pmatrix}, \begin{pmatrix}0\\ 0\\ \frac{1}{2}\\ -\frac{1}{2}\\ 0\\ 0\end{pmatrix}, \begin{pmatrix}0\\ 0\\ 0\\ 0\\ \frac{1}{2}\\ \frac{1}{2}\end{pmatrix}, \begin{pmatrix}0\\ 0\\ 0\\ 0\\ \frac{1}{2}\\ -\frac{1}{2}\end{pmatrix}\right)$$

indexed by the list $S_1 = ((1,1),(1,2),(2,1),(2,2),(3,1),(3,2))$. Applying the successor function s_2 to S_1, sorting, and concatenating then produces

$$B_{2,0} = \left(\begin{pmatrix}\frac{1}{2}\\ \frac{1}{2}\\ 0\\ 0\\ 0\\ 0\end{pmatrix}, \begin{pmatrix}\frac{1}{2}\\ -\frac{1}{2}\\ 0\\ 0\\ 0\\ 0\end{pmatrix}, \begin{pmatrix}0 & 0\\ 0 & 0\\ \frac{1}{2} & 0\\ \frac{1}{2} & 0\\ 0 & \frac{1}{2}\\ 0 & \frac{1}{2}\end{pmatrix}, \begin{pmatrix}0 & 0\\ 0 & 0\\ \frac{1}{2} & 0\\ -\frac{1}{2} & 0\\ 0 & \frac{1}{2}\\ 0 & -\frac{1}{2}\end{pmatrix}\right)$$

indexed by $S_{2,0} = ((1,1),(1,2),(2,1),(2,2))$, as well as the first nontrivial permutation factor

$$P_2 = \begin{pmatrix}1 & & & & & \\ & 1 & & & & \\ & & 1 & & & \\ & & & & 1 & \\ & & & 1 & & \\ & & & & & 1\end{pmatrix}.$$

Applying the projections in Equation (A.3) then yields products such as

$$([E_2^{1,2}]_B)^T(B_{2,0})_3 = \frac{1}{2}\begin{pmatrix}0&0&1&0&0&1\\0&0&0&1&1&0\end{pmatrix}\begin{pmatrix}0&0\\0&0\\1&0\\1&0\\0&1\\0&1\end{pmatrix} = \frac{1}{2}\begin{pmatrix}1&1\\1&1\end{pmatrix} = \begin{pmatrix}\frac{1}{2}\\\frac{1}{2}\end{pmatrix}\begin{pmatrix}1&1\end{pmatrix},$$

and corresponding eigenvectors such as

$$[E_2^{1,2}]_B(([E_2^{1,2}]_B)^T[E_2^{1,2}]_B)^{-1}\begin{pmatrix}\frac{1}{2}\\\frac{1}{2}\end{pmatrix} = \frac{1}{4}\begin{pmatrix}0&0\\0&0\\1&0\\0&1\\0&1\\1&0\end{pmatrix}\begin{pmatrix}1&0\\0&1\end{pmatrix}\begin{pmatrix}1\\1\end{pmatrix} = \frac{1}{4}\begin{pmatrix}0\\0\\1\\1\\1\\1\end{pmatrix}.$$

These calculations produce the second change-of-basis factor

$$F_2 = \begin{pmatrix}1&&&&&\\&1&&&&\\&&1&1&&\\&&1&-1&&\\&&&&1&1\\&&&&1&-1\end{pmatrix}$$

as well as the eigenbases

$$B_3 = \left(\begin{pmatrix}\frac{1}{2}\\\frac{1}{2}\\0\\0\\0\\0\end{pmatrix},\begin{pmatrix}\frac{1}{2}\\-\frac{1}{2}\\0\\0\\0\\0\end{pmatrix},\begin{pmatrix}0\\0\\\frac{1}{4}\\\frac{1}{4}\\\frac{1}{4}\\\frac{1}{4}\end{pmatrix},\begin{pmatrix}0\\0\\\frac{1}{4}\\\frac{1}{4}\\-\frac{1}{4}\\-\frac{1}{4}\end{pmatrix},\begin{pmatrix}0\\0\\\frac{1}{4}\\-\frac{1}{4}\\-\frac{1}{4}\\\frac{1}{4}\end{pmatrix},\begin{pmatrix}0\\0\\\frac{1}{4}\\-\frac{1}{4}\\\frac{1}{4}\\-\frac{1}{4}\end{pmatrix}\right),$$

indexed by $S_3 = ((1,1,1),(1,2,2),(2,1,1),(2,1,2),(2,2,1),(2,2,2))$. Applying s_3 to the elements of S_3, sorting, and concatenating one last time

produces

$$B_{3,0} = \left(\begin{pmatrix} \frac{1}{2} & 0 \\ \frac{1}{2} & 0 \\ 0 & \frac{1}{4} \\ 0 & \frac{1}{4} \\ 0 & \frac{1}{4} \\ 0 & \frac{1}{4} \end{pmatrix}, \begin{pmatrix} 0 \\ 0 \\ \frac{1}{4} \\ \frac{1}{4} \\ -\frac{1}{4} \\ -\frac{1}{4} \end{pmatrix}, \begin{pmatrix} 0 \\ 0 \\ \frac{1}{4} \\ -\frac{1}{4} \\ -\frac{1}{4} \\ \frac{1}{4} \end{pmatrix}, \begin{pmatrix} \frac{1}{2} & 0 \\ -\frac{1}{2} & 0 \\ 0 & \frac{1}{4} \\ 0 & -\frac{1}{4} \\ 0 & \frac{1}{4} \\ 0 & -\frac{1}{4} \end{pmatrix} \right),$$

indexed by $S_{3,0} = ((1,1,1),(1,1,2),(1,2,1),(1,2,2))$, as well as the second nontrivial permutation factor

$$P_3 = \begin{pmatrix} 1 & & & & & \\ & & 1 & & & \\ & & & 1 & & \\ & & & & 1 & \\ & 1 & & & & \\ & & & & & 1 \end{pmatrix}.$$

Applying the projections in Equation (A.4) then yields the factor

$$F_3 = \begin{pmatrix} 1 & 1 & & & & \\ 1 & -\frac{1}{2} & & & & \\ & & 1 & & & \\ & & & 1 & & \\ & & & & 1 & \frac{1}{2} \\ & & & & 1 & -1 \end{pmatrix}$$

as well as the eigenbases

$$B_4 = \left(\begin{pmatrix} \frac{1}{6} \\ \frac{1}{6} \\ \frac{1}{6} \\ \frac{1}{6} \\ \frac{1}{6} \\ \frac{1}{6} \end{pmatrix}, \begin{pmatrix} \frac{1}{3} \\ \frac{1}{3} \\ -\frac{1}{6} \\ -\frac{1}{6} \\ -\frac{1}{6} \\ -\frac{1}{6} \end{pmatrix}, \begin{pmatrix} 0 \\ 0 \\ \frac{1}{4} \\ \frac{1}{4} \\ -\frac{1}{4} \\ -\frac{1}{4} \end{pmatrix}, \begin{pmatrix} 0 \\ 0 \\ \frac{1}{4} \\ -\frac{1}{4} \\ -\frac{1}{4} \\ \frac{1}{4} \end{pmatrix}, \begin{pmatrix} \frac{1}{3} \\ -\frac{1}{3} \\ \frac{1}{6} \\ -\frac{1}{6} \\ \frac{1}{6} \\ -\frac{1}{6} \end{pmatrix}, \begin{pmatrix} \frac{1}{6} \\ -\frac{1}{6} \\ -\frac{1}{6} \\ \frac{1}{6} \\ -\frac{1}{6} \\ \frac{1}{6} \end{pmatrix} \right)$$

indexed by

$$S_4 = ((1,1,1,1),(1,1,1,2),(1,1,2,2),(1,2,1,2),(1,2,2,2),(1,2,2,3)).$$

We note that to put the coefficients in the order corresponding to the row-major order of the Fourier coefficients, we must reverse the order of the

coefficients for the second block. The permutation matrix for this change is given by

$$P_{\mathrm{DFT}} = \begin{pmatrix} 1 & & & & & \\ & & & 1 & & \\ & & 1 & & & \\ & 1 & & & & \\ 1 & & & & & \\ & & & & & 1 \end{pmatrix}.$$

We compute S_{DFT} according to Algorithm 5.7. We set

$$S = \begin{pmatrix} 1 & & & & & \\ & 1 & & & & \\ & & 1 & & & \\ & & & 1 & & \\ & & & & 1 & \\ & & & & & 1 \end{pmatrix}.$$

We have yet to scale the coefficients indexed by $L = \{1,2,3,4,5,6\}$. Since the image of the identity under the factorization is

$$P_{\mathrm{DFT}}F_3P_3F_2P_2F_1[1]_B = \begin{pmatrix} 1 \\ 1 \\ 0 \\ 0 \\ 1 \\ 1 \end{pmatrix} \mapsto \begin{pmatrix} 1 & & & \\ & 1 & 0 & \\ & 0 & 1 & \\ & & & 1 \end{pmatrix},$$

which is the identity in the matrix algebra, we need do no scaling of coefficients for the identity. Furthermore, since we know these rows are scaled correctly, we remove the indices $1,2,5,6$ from L to leave $L = \{3,4\}$. We then compute the image of the generator $(2\,3)$ under this transform to be

$$SP_{\mathrm{DFT}}F_3P_3F_2P_2F_1[(2\,3)]_B = \begin{pmatrix} 1 \\ \frac{1}{2} \\ 1 \\ 1 \\ -\frac{1}{2} \\ -1 \end{pmatrix} \mapsto \begin{pmatrix} 1 & & & \\ & \frac{1}{2} & 1 & \\ & 1 & -\frac{1}{2} & \\ & & & -1 \end{pmatrix}.$$

We observe that the correct coefficients in the second block should be

$$\sigma^2((2\,3)) = \begin{pmatrix} \frac{1}{2} & \frac{3}{2} \\ \frac{1}{2} & -\frac{1}{2} \end{pmatrix}.$$

Thus, we must scale the third coefficient by $\frac{3}{2}$ and the fourth by $\frac{1}{2}$, and remove 3 and 4 from L to obtain $L = \emptyset$. Since L is now empty, we terminate, with our scaling matrix determined to be

$$S_{\text{DFT}} = \begin{pmatrix} 1 & & & & & \\ & 1 & & & & \\ & & \frac{3}{2} & & & \\ & & & \frac{1}{2} & & \\ & & & & 1 & \\ & & & & & 1 \end{pmatrix}.$$

Then our factorization of the DFT D for S_3 is

$$[D]_B^{\mathcal{F}} = S_{\text{DFT}} P_{\text{DFT}} F_3 P_3 F_2 P_2 F_1.$$

This factorization of our transform then takes 14 additions and 4 multiplications to carry out, or a total of 16 complex operations under Maslen's (18) computational model.

A.2 $\mathbb{C}S_3$ with Jucys-Murphy Elements

We now approach this factorization through the Jucys-Murphy elements $m_2 = (1\,2)$ and $m_3 = (1\,3) + (2\,3)$. In the right-coset basis R_2 for $\mathbb{C}S_2$ contained in R, the matrix representation of m_2 is

$$[m_2 \otimes 1]_{R_2}^{R_2} = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix},$$

for which eigenvectors associated to the eigenvalues 1 and -1 are $\begin{pmatrix} 1 & 1 \end{pmatrix}^T$ and $\begin{pmatrix} 1 & -1 \end{pmatrix}^T$, respectively. Since these are the same as the eigenvectors determined for $\mathbb{C}S_2$ in Section A.1, the corresponding projection operators for the action of $m_2 \otimes 1$ and $1 \otimes m_2$ are the same as those shown in Equations (A.2) and (A.3).

The matrix representation of $m_3 \otimes 1$ with respect to R is

$$[m_3 \otimes 1]_R^R = \begin{pmatrix} 0 & 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 & 0 & 1 \\ 1 & 0 & 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 1 & 0 & 0 & 0 \end{pmatrix}$$

from which we calculate bases for its eigenspaces to be

$$[E_3^2]_B = \begin{pmatrix} 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 1 \end{pmatrix}, \qquad [E_3^1]_B = \begin{pmatrix} 1 & 1 \\ -1 & -1 \\ 0 & 1 \\ 0 & -1 \\ 1 & 0 \\ -1 & 0 \end{pmatrix},$$

$$[E_3^{-1}]_B = \begin{pmatrix} 1 & 1 \\ 1 & 1 \\ 0 & -1 \\ 0 & -1 \\ -1 & 0 \\ -1 & 0 \end{pmatrix}, \qquad [E_3^{-2}]_B = \begin{pmatrix} 1 \\ -1 \\ -1 \\ 1 \\ -1 \\ 1 \end{pmatrix}. \tag{A.5}$$

The inverse matrices $(E^T E)^{-1}$ for these eigenbases are

$$(([E_3^2]_B)^T [E_3^2]_B)^{-1} = (([E_3^{-2}]_B)^T [E_3^{-2}]_B)^{-1} = \left(\tfrac{1}{6}\right),$$
$$(([E_3^1]_B)^T [E_3^{1]_B})^{-1} = (([E_3^{-1}]_B)^T [E_3^{-1}]_B)^{-1} = \frac{1}{6}\begin{pmatrix} 2 & -1 \\ -1 & 2 \end{pmatrix}.$$

Since we have identical projection eigenbases in the first two stages of the computation, the factors F_1, P_2, F_2, P_3 are all the same as those computed above in Section A.1, although $S_{3,0}$ is slightly different because we append Jucys-Murphy eigenvalues rather than idempotent indices in the construction of the first two change-of-basis factors. Hence, we have $B_{3,0}$ as above, indexed by $S_{3,0} = ((1,1,1),(1,1,-1),(1,-1,1),(1,-1,-1))$, where we elect to sort the eigenvalues in decreasing order.

We note also that the second and third eigenbases $(B_{3,0})_2$ and $(B_{3,0})_3$ both have unique lists of eigenvalues and are one-dimensional. Hence, we may apply Algorithm 5.4 to determine that the full lists of eigenvalues are $((1,-1),(-1,1))$ and $((-1,1),(1,-1))$, respectively. Thus, $(B_{3,0})_2$ lies in the -1 eigenspace of $m_3 \otimes 1$, so we need not compute any of its projections in the computation of F_4.

Applying the projection operators determined by Equation (A.5) to the remaining bases $(B_{3,0})_1$ and $(B_{3,0})_4$ yields the factor F_4 and eigenbases B_4 from above, indexed now by the eigenvalue lists

$$S_4 = ((1,1,1,2),(1,1,1,-1),(1,1,-1,-1),$$
$$(1,-1,1,1),(1,-1,-1,1),(1,-1,-1,-2)).$$

Sorting these basis vectors according to Algorithm 5.6 and scaling them by Algorithm 5.7 produces the P_{DFT} and S_{DFT} matrices given above. Thus, in this case, the Jucys-Murphy elements provide the same factorization of $[D]_B^{\mathcal{F}}$, and bypassing two eigenbases in the computation of F_3 saves some operations in the precomputation step.

Appendix B

Tabulation of Double Coset Irreducibles

In this apendx, we tabulate the decompositions of the double-coset modules in $\mathbb{C}S_n$ for $n = 3, 4, 5$ into tensor products of irreducible representations. We consider only the (S_2, S_1)-double-coset modules through the (S_{n-1}, S_{n-1})-double-coset modules, as the (S_1, S_1)-double-coset modules are all trivial and the (S_n, S_{n-1})- and (S_n, S_n)-double-coset modules are all of S_n. Moreover, we note that each (S_2, S_1)-double-coset module M decomposes as

$$M \cong S^{(2)} \otimes S^{(1)} \oplus S^{(1,1)} \otimes S^{(1)}.$$

For convenience, we tabulate these decompositions in matrices, with the row and column indices corresponding to the partitions of n in dual (i.e., decreasing) lexicographic order. An m in the ijth position of the matrix associated to the module M then indicates that $S^{\lambda_i} \otimes S^{\lambda_j}$ has multiplicity m in M. These total orders on the partitions of $n = 2$ through $n = 5$ are given as follows:

n	Partition Order
2	$(2) < (1,1)$
3	$(3) < (2,1) < (1,1,1)$
4	$(4) < (3,1) < (2,2) < (2,1,1) < (1,1,1,1)$
5	$(5) < (4,1) < (3,2) < (3,1,1)$ $< (2,2,1) < (2,1,1,1) < (1,1,1,1,1)$

B.1 Double-Coset Modules in $\mathbb{C}S_3$

The decompositions of the double-coset modules for $\mathbb{C}S_5$ are shown in Table B.1.

σ	Multiplicites
1	$\begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}$
$(1\,3)$	$\begin{pmatrix} 1 & 1 \\ 1 & 1 \end{pmatrix}$

Table B.1: (S_2, S_2)-Double Cosets in $\mathbb{C}S_3$.

B.2 Double-Coset Modules in $\mathbb{C}S_4$

The decompositions of the double-coset modules for $\mathbb{C}S_5$ are shown in Tables B.2 through B.4.

σ	Multiplicites
1 $(3\,4)$	$\begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}$
$(1\,3)$ $(1\,4)$ $(1\,3)(1\,4)$ $(1\,3)(2\,4)$ $(1\,3)(3\,4)$	$\begin{pmatrix} 1 & 1 \\ 1 & 1 \end{pmatrix}$

Table B.2: (S_2, S_2)-Double Cosets in $\mathbb{C}S_4$.

σ	Multiplicites
1 $(3\,4)$	$\begin{pmatrix} 1 & 0 \\ 1 & 1 \\ 0 & 1 \end{pmatrix}$
$(1\,3)$	$\begin{pmatrix} 1 & 1 \\ 2 & 2 \\ 1 & 1 \end{pmatrix}$

Table B.3: (S_3, S_2)-Double Cosets in $\mathbb{C}S_4$.

σ	Multiplicites
1	$\begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}$
$(1\,3)$	$\begin{pmatrix} 1 & 1 & 0 \\ 1 & 2 & 1 \\ 0 & 1 & 1 \end{pmatrix}$

Table B.4: (S_3, S_3)-Double Cosets in $\mathbb{C}S_4$.

B.3 Double-Coset Modules in $\mathbb{C}S_5$

The decompositions of the double-coset modules for $\mathbb{C}S_5$ are shown in Tables B.5 through B.9.

σ		Multiplicites
1	(34)	$\begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}$
(35)	$(34)(35)$	
(45)	$(34)(45)$	
all others		$\begin{pmatrix} 1 & 1 \\ 1 & 1 \end{pmatrix}$

Table B.5: (S_2, S_2)-Double Cosets in $\mathbb{C}S_5$.

σ		Multiplicites
1	(34)	$\begin{pmatrix} 1 & 0 \\ 1 & 1 \\ 0 & 1 \end{pmatrix}$
(35)	$(34)(35)$	
(45)	$(34)(45)$	
(14)	(15)	$\begin{pmatrix} 1 & 1 \\ 2 & 2 \\ 1 & 1 \end{pmatrix}$
$(14)(15)$	$(14)(25)$	
$(14)(35)$	$(13)(54)$	
$(34)(15)$		

Table B.6: (S_3, S_2)-Double Cosets in $\mathbb{C}S_5$.

σ	Multiplicites
1 (45)	$\begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}$
(14) $\quad$ (15) $(14)(15)$ $\quad$ $(14)(45)$	$\begin{pmatrix} 1 & 1 & 0 \\ 1 & 2 & 1 \\ 0 & 1 & 1 \end{pmatrix}$
$(14)(25)$	$\begin{pmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{pmatrix}$

Table B.7: (S_3, S_3)-Double Cosets in $\mathbb{C}S_4$.

σ	Multiplicites
1 (45)	$\begin{pmatrix} 1 & 0 & 0 \\ 1 & 1 & 0 \\ 0 & 1 & 0 \\ 0 & 1 & 1 \\ 0 & 0 & 1 \end{pmatrix}$
(15)	$\begin{pmatrix} 1 & 1 & 0 \\ 2 & 3 & 1 \\ 1 & 2 & 1 \\ 1 & 3 & 2 \\ 0 & 1 & 1 \end{pmatrix}$

Table B.8: (S_4, S_3)-Double Cosets in $\mathbb{C}S_4$.

σ	Multiplicites
1	$\begin{pmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{pmatrix}$
(15)	$\begin{pmatrix} 1 & 1 & 0 & 0 & 0 \\ 1 & 2 & 1 & 1 & 0 \\ 0 & 1 & 1 & 1 & 0 \\ 0 & 1 & 1 & 2 & 1 \\ 0 & 0 & 0 & 1 & 1 \end{pmatrix}$

Table B.9: (S_4, S_4)-Double Cosets in $\mathbb{C}S_4$.

Appendix C

Mathematica Code: FFT Generation Algorithm

The following pages present the Mathematica code that constitutes the initial implementation of the decimation-in-frequency FFT precomputation algorithm specified in Algorithms 4.18, 4.16, 5.6 and 5.7. Sparse matrices are used throughout to improve efficiency in both memory and computation time.

```
(*
SymmetricGroupFFT Package
*)
BeginPackage[
    "SymmetricGroupFFT`", {"DiscreteMath`Combinatorica`",
      "LinearAlgebra`MatrixManipulation`",
      "LinearAlgebra`Orthogonalization`"}];
(*
Matrix Operations
*)
(*
Smart matrix operations that ignore {{}}
*)
SmartDot[(A_)?MatrixQ, (B_)?MatrixQ] :=
    If[A == {{}} || B == {{}}, {{}}, A.B];
SameColumnSize[l_List] := SameQ @@ (Dimensions[#1][[2]] &) /@ l;
MatrixNullQ[x_] = x == {{}};
ListNullQ[x_] = x == {};
SmartColumnAppend[(x__)?MatrixQ] :=
  Join @@ DeleteCases[{x}, _?MatrixNullQ] /;
    SameColumnSize[DeleteCases[{x}, _?MatrixNullQ]]
(*
Sparse matrix construction
*)
SparseIdentityMatrix[n_Integer] /; n > 0 :=
    SparseArray[{{i_, i_} -> 1}, {n, n}];
SparseDiagonalMatrix[(values_)?ListQ] :=
```

```
    SparseArray[
      Table[{i, i}, {i, 1, Length[values]}] -> values, {Length[values],
        Length[values]}];
(*
Sparse matrix direct sum
*)
MatrixDirectSumSparse[mat : {_?MatrixQ ..}] :=
    MatrixDirectSumSparseInner[DeleteCases[mat, _?MatrixNullQ]];
MatrixDirectSumSparseInner[mat : {_?MatrixQ ..}] :=
  SparseArray[
    Flatten[MapThread[
          Array[List, #1, #2 +
                1] &, ({#1, Most[FoldList[Plus, {0, 0}, #1]]} &)[
            Dimensions /@ mat]], 2] -> Flatten[mat]]
(*
Diagonal Inverse
*)
SparseDiagonalInverse[mat_] :=
    SparseArray[Table[{i, i} -> 1/mat[[i, i]], {i, 1, Length[mat]}]];
(*
Permutation matrices for defining representation
*)
(*
RowPermutationMatrix[p] generates a permutation matrix that moves row p[i] to \
row i or column i to p[i].
ColumnPermutationMatrix[p] generates a permutation matrix that moves row i to \
row p[i] or column p[i] to i.
*)
RowPermutationMatrix[(p_)?PermutationQ] :=
    SparseArray[
      MapThread[{#1, #2} &, {Range[1, Length[p]], p}] ->
        Table[1, {Length[p]}]];
ColumnPermutationMatrix[(p_)?PermutationQ] :=
    SparseArray[
      MapThread[{#1, #2} &, {p, Range[1, Length[p]]}] ->
        Table[1, {Length[p]}]];
(*
Row and Column Selection Matrices
*)
(*
RowSelectionMatrix[L, n] generates a matrix that selects rows L from an n-row \
matrix.
ColumnSelectionMatrix[L, n] generates a matrix that selects columns L from an \
n-column matrix.
*)
RowSelectionMatrix[l_List, ncols_Integer] :=
    SparseArray[
      MapThread[List, {Range[1, Length[l]], l}] ->
        Table[1, {Length[l]}], {Length[l], ncols}];
ColumnSelectionMatrix[l_List, nrows_Integer] :=
    SparseArray[
      MapThread[List, {l, Range[1, Length[l]]}] ->
        Table[1, {Length[l]}], {nrows, Length[l]}];
(*
Matrix Output
*)
RowToTeX[x_?VectorQ] := Module[{len},
      len = Length[x];
      StringJoin @@
```

```
        Take[Flatten[Map[{#, " & "} &, Map[ToString, Map[TeXForm, x]]]], {1,
            2len - 1}]
      ];
MatrixToTeX[x_?MatrixQ] := Module[{len},
      len = Length[x];
      "\\begin{pmatrix} " <>
        StringJoin @@
          Take[Flatten[Map[{#, " \\\\ "} &, Map[RowToTeX, x]]], {1,
              2 len - 1}] <>  "\\end{pmatrix}"
      ];
RowToGAP[x_?VectorQ] := Module[{len},
      len = Length[x];
      "[ " <>
        StringJoin @@
          Take[Flatten[
              Map[{#, ", "} &, Map[ToString, Map[InputForm, x]]], {1,
              2len - 1}] <>  " ]"
      ];
MatrixToGAP[x_?MatrixQ] := Module[{len},
      len = Length[x];
      "[ " <>
        StringJoin @@
          Take[Flatten[Map[{ #, ", "} &, Map[RowToGAP, x]]], {1,
              2 len - 1}] <>  " ]"
      ];
(*
Miscellaneous Utilities
*)
(*
"Unzip" list by every other entry
*)
splitList[(x_)?ListQ] :=
    Reap[(Sow[x[[#1]], Mod[#1, 2]] &) /@ Range[1, Length[x]]][[2]];
(*
"Rezip" two lists back together
*)
interleaveLists[l1_List, l2_List] := Flatten[MapThread[List, {l1, l2}]];
(*
Intersect sets with multiplicity
*)
MultiIntersection[l1_List, l2_List] :=
  Module[{nl, f}, f[x_] := {First[#], Length[#]} & /@ Split[Sort[x]];
    nl = Sort[Join[Flatten[Map[f, {l1, l2}], 1]]];
    nl = Split[nl, #[[1]] === #2[[1]] &];
    Flatten[Cases[nl, {{x_, m_}, {x_, n_}} :> Table[x, {m}]], 1]]
(*
Recover List Tails from Pairs of Lists
*)
CompleteJMEvalLists[l1_List, l2_List] :=
    Module[{nl, f, comelts, l1lookup, l2lookup, l1diff, l2diff},
      f[x_] := {First[#], Length[#]} & /@ Split[Sort[x]];
      nl = Sort[Join[Flatten[Map[f, {l1, l2}], 1]]];
      nl = Split[nl, #[[1]] === #2[[1]] &];
      Cases[nl, {{x_, m_}, {x_, n_}} :> (comelts[x] = m)];
      Cases[nl, {{x_, m_}} :> (comelts[x] = 0)];
      Reap[
          Apply[Sow, MapThread[List, {Range[1, Length[l1]], l1}],
            1], _, (l1lookup[#1] = #2) &][[2]];
      Reap[
```

```
          Apply[Sow, MapThread[List, {Range[1, Length[l2]], l2}],
            1], _, (l2lookup[#1] = #2) &][[2]];
      l1diff =
        l1[[Sort@Flatten[(Drop[l1lookup[#], comelts[#]]) & /@ Union[l1]]]];
      l2diff =
        l2[[Sort@Flatten[(Drop[l2lookup[#], comelts[#]]) & /@ Union[l2]]]];
      {Join[l1, l2diff], Join[l2, l1diff]}
      ];
(*
Symmetric Group Construction
*)
(*
Construction of Coset-Ordered Symmetric Group
*)
(*
Transposition[n, i, j] produces the permutation in Sn that swaps i and j.
JMTranspositionList[n, k] produces the list of permutations for m_k in S_n.
*)
Transposition[n_Integer, i_Integer, j_Integer] :=
    Table[k + KroneckerDelta[i, k]*(j - i) - KroneckerDelta[j, k]*(j - i), {k,
         1, n}];

JMTranspositionList[n_Integer, k_Integer] /; n > 1 && k > 1 && k <= n :=
    Table[Transposition[n, j, k], {j, 1, k - 1}];
(*
SymmetricGroupCoset[A, p, 0] = p A;
SymmetricGroupCoset[A, p, 1] = A p;
SymmetricGroupByCosets[n, k, o] constructs S_k ordered by order o, as \
length-n permutations.
Current orders:
0: Nested left-coset order, with coset representatives in the order 1, (k - 1 \
k), ..., (1 k)
1: Nested right-coset order, with coset representatives in the order 1, (k - \
1 k), ..., (1 k)
*)
SymmetricGroupCoset[subset : {_?PermutationQ ..}, (p_)?PermutationQ,
      0] := (Permute[p, #1] &) /@ subset;
SymmetricGroupCoset[subset : {_?PermutationQ ..}, (p_)?PermutationQ,
      1] := (Permute[#1, p] &) /@ subset;
SymmetricGroupByCosets[n_Integer, 1, side_Integer] /; n > 0 := {Range[1, n]};
SymmetricGroupByCosets[n_Integer, k_Integer, order_Integer] /;
      n > 0 && k > 1 && n >= k && (order == 0 || order == 1) :=
    Module[{SSubGroup}, SSubGroup = SymmetricGroupByCosets[n, k - 1, order];
      Flatten[(SymmetricGroupCoset[SSubGroup, #1, order] &) /@
          Table[Transposition[n, j, k], {j, k, 1, -1}], 1]];
(*
Permutation Ranking in Coset Order
*)
(*
ConstructSymmetricGroupLookups[n, 0/1] constructs the lookup table for S_n in \
the L/R coset order.
RankPermutationByCoset[p, 0/1] ranks p in the L/R coset order for S_n, where \
n = Length[p];
defaults to right coset order if second argument omitted.
UnrankPermutationByCoset[i, n, 0/1] generates p from its rank i in the L/R \
coset order for S_n.
*)
ConstructSymmetricGroupLookups[n_Integer,
      order_Integer] := (SymmetricGroupInverseLookup[n,
```

```
          order] = (RankPermutation[#1] + 1 &) /@
          SymmetricGroupByCosets[n, n, order];
      SymmetricGroupLookup[n, order] =
        InversePermutation[SymmetricGroupInverseLookup[n, order]];);

RankPermutationByCoset[(p_)?PermutationQ, order_Integer] :=
    SymmetricGroupLookup[Length[p], order][[RankPermutation[p] + 1]];
UnrankPermutationByCoset[index_Integer, n_Integer, order_Integer] :=
    UnrankPermutation[SymmetricGroupInverseLookup[n, order][[index]] - 1, n];
RankPermutationByCoset[(p_)?PermutationQ] = RankPermutationByCoset[p, 1];
(*
Construction of Double-Coset Ordered Symmetric Group
*)
(*
SymmetricCompare compares two permutations according to the specified order.
*)
SymmetricCompare[p_?PermutationQ, q_?PermutationQ,
      order_Integer] := (RankPermutationByCoset[p, order] <
        RankPermutationByCoset[q, order]);
(*
PartitionByConjugacy partitions a subset A of S_n by conjugacy classes under \
the action of S_k in S_n.
*)
PartitionByConjugacy[A : {_?PermutationQ ..}, k_Integer] :=
    Module[{Arest = Sort[A, SymmetricCompare[#1, #2, 1] &], a, aconjs,
        n = Length[First[A]], p = {},
        sym = SymmetricGroupByCosets[Length[First[A]], k, 1] },
      While[Length[Arest] > 0,
        a = First[Arest];
        aconjs =
          Sort[Union[(Permute[Permute[#, a], InversePermutation[#]] &) /@
                sym], SymmetricCompare[#1, #2, 1] &];
        AppendTo[p, aconjs];
        Arest =
          Sort[ Complement[Arest, aconjs], SymmetricCompare[#1, #2, 1] &];
        ];
      p
      ];
(*
PartitionSymmetricGroup partitions S_k in S_n by double cosets of the \
subgroup chain 1 < S_2 < ... < S_n.
*)
PartitionSymmetricGroupByDC[narg_Integer, karg_Integer] :=
    Module[{n = narg, k = karg, sympart, cosetreps},
      sympart = {UnrankPermutation[0, n]};
      sympart = Map[PartitionByConjugacy[#, k] &, sympart, {-3}];
      sympart = Map[PartitionByConjugacy[#, k - 1] &, sympart, {-3}];
      cosetreps = Table[Transposition[n, i, k], {i, k, 1, -1}];
      sympart =
        Map[Flatten[Outer[Permute, cosetreps, #, 1], 1] &, sympart, {-3}];
      While[k > 2,
        k--;
        sympart = Map[PartitionByConjugacy[#, k] &, sympart, {-3}];
        sympart = Map[PartitionByConjugacy[#, k - 1] &, sympart, {-3}];
        cosetreps = Table[Transposition[n, i, k], {i, k, 1, -1}];
        sympart =
          Map[Flatten[Outer[Permute, cosetreps, #, 1], 1] &, sympart, {-3}];
        ];
      (* put this line in if we want S1, S1 trivial double cosets *)
```

```
      (*sympart = Map[PartitionByConjugacy[#, 1] &, sympart, {-3}];*)
      sympart];
(*
SymmetricGroupByCosets computes S_k in S_n ordered by double cosets.
*)
SymmetricGroupByCosets[n_Integer, k_Integer, 2] :=
    Reap[Map[Sow, PartitionSymmetricGroupByDC[n, k], {-2}]][[2]][[1]];
(*
ResolveTreeLayer replaces leaf lists (smallests lists in tree) with their \
lengths, and Sows a list consisting of each leaf list's index, repeated for \
the length of the list.
This second list indicates which branches at level n of the tree merge at \
level n - 1.
*)
ResolveTreeLayer[tree_List] := Module[{temp, i = 0},
    temp = Map[Length, tree, {-2}];
    Sow[Flatten[Reap[Map[(i++; Sow[Table[i, {#}]]) &, Flatten[temp]]][[2]]]];
    temp
    ]
(*
SymmetricGroupDoubleCosetBranching determines which double cosets merge as \
they grow larger in the usual L/R subgroup chain pattern.
*)
SymmetricGroupDoubleCosetBranching[n_Integer, 2] :=
  Reap[Fold[ResolveTreeLayer[#1] &,
          Map[1 &, PartitionSymmetricGroupByDC[n, n], {-2}],
          Range[1, 2n - 3]]][[2]][[1]]
(*
ConstructSymmetricGroupDCBLookup constructs the branching lookup table for \
the double cosets of the usual subgroup chain in S_n.
*)
ConstructSymmetricGroupDCBLookup[n_Integer, 2] :=
    Module[{}, (SymmetricGroupDCBLookup[n, 2] =
            SymmetricGroupDoubleCosetBranching[n, 2]);];
(*
SymmetricGroupDCBNesting[index, n, o] computes the indices of the double \
cosets in which the specified permutation
*)
SymmetricGroupDCBNesting[index_Integer, n_Integer, order_Integer] :=
    FoldList[SymmetricGroupDCBLookup[n, order][[#2, #1]] &, index,
      Range[1, 2n - 3]];
(*
Matrices for Left and Right Regular Representations of Symmetric Group
*)
(*
PermutationRegRep[p, 0/1] gives the L/R regular permutation representation \
with respect to the lex. ordering from Combinatorica.
PermutationRegRepCoset[p, 0/1, o] gives the L/R regular perm repn with \
respect to order o.
JMMatrix[n, k, 0/1] gives the L/R reg perm repm of m_k on S_n with respect to \
the right coset basis.
*)
PermutationRegRep[(p_)?PermutationQ, 0] :=
    SparseArray[({RankPermutation[Permute[p, #1]] + 1,
                RankPermutation[#1] + 1} &) /@ SymmetricGroup[Length[p]] ->
        Table[1, {i, 1, Length[p]!}]];
PermutationRegRep[(p_)?PermutationQ, 1] :=
    SparseArray[({RankPermutation[Permute[#1, p]] + 1,
                RankPermutation[#1] + 1} &) /@ SymmetricGroup[Length[p]] ->
```

```
        Table[1, {i, 1, Length[p]!}]];

PermutationRegRepCoset[(p_)?PermutationQ, 0, order_Integer] :=
    SparseArray[({RankPermutationByCoset[Permute[p, #1], order],
                RankPermutationByCoset[#1, order]} &) /@
          SymmetricGroup[Length[p]] -> Table[1, {i, 1, Length[p]!}]];
PermutationRegRepCoset[(p_)?PermutationQ, 1, order_Integer] :=
    SparseArray[({RankPermutationByCoset[Permute[#1, p], order],
                RankPermutationByCoset[#1, order]} &) /@
          SymmetricGroup[Length[p]] -> Table[1, {i, 1, Length[p]!}]];
(*
Jucys-Murphy Regular Representations
*)
(*
JMMatrix[n, k, o] generates the regular representation for JM elt m_k acting \
on S_n on the L/R in the right-coset order.
*)
JMMatrix[n_Integer, k_Integer, side_Integer] /; n > 1 && k > 1 && k <= n :=
    Plus @@ (PermutationRegRepCoset[#1, side, 1] &) /@
        JMTranspositionList[n, k];
(*
JMEigensystem[n, k, 0/1] generates bases and eigenvalues for the eigenspaces \
of m_k for S_n acting on the L/R in right-coset order.
*)
JMEigensystem[n_Integer, k_Integer, side_Integer] :=
    Module[{jm, sparseidentity, evecs},
      jm = JMMatrix[n, k, side];
      sparseidentity = SparseIdentityMatrix[n!];
      evecs =
        Reverse[DeleteCases[(NullSpace[jm - #1*sparseidentity] &) /@
              Range[-k + 1, k - 1], _?ListNullQ]];
      {evecs, (({#1[[1]]}.jm.#1[[1]])[[1]]/Norm[#1[[1]]]^2 &) /@ evecs}
      ];
(*
Double Coset Matrices
*)
LeftCosetMatrix[n_Integer, k_Integer, order_Integer] :=
    Plus @@ (PermutationRegRepCoset[#, 0, order] & /@
          SymmetricGroupByCosets[n, k, order]);
RightCosetMatrix[n_Integer, k_Integer, order_Integer] :=
    Plus @@ (PermutationRegRepCoset[#, 1, order] & /@
          SymmetricGroupByCosets[n, k, order]);
DoubleCosetMatrix[n_Integer, k1_Integer, k2_Integer, order_Integer] :=
    LeftCosetMatrix[n, k1, order].RightCosetMatrix[n, k2, order];
(*
FFT Construction
*)
(*
FFT routines
*)
(*
Nonpartitioned routines
*)
(*
FFTFactorization[n] generates full DFT decomposiiton for Sn.
*)
FFTFactorization[n_Integer, order_Integer] /; n > 1 :=
    Module[{factorlist, devallist, irreddims},
      {devallist, factorlist} = FFTFactorizationToStage[n, 2*n - 3, order];
```

```
      devallist = ModifyFFTEigenvalueList[devallist];

      AppendTo[factorlist,
        RowPermutationMatrix[Flatten[Map[#1[[4]] &, devallist, {2}]]]];

      irreddims = Length /@ Union /@ Sort /@ (First /@ #1 &) /@ devallist;

      AppendTo[factorlist,
        FFTScalingMatrix[n, order, factorlist, irreddims]];
      {devallist, factorlist, irreddims}
      ];
(*
FFTFactorizationToStage[n, s, o] generates the DFT matrix decomposition to \
stage s for order o.
*)
FFTFactorizationToStage[n_Integer, stage_Integer, order_Integer] /;
      n >= 2 && stage > 0 && stage <= 2*n - 3 :=
    Module[{factorlist = {}, dlist = {SparseIdentityMatrix[n!]},
       devallist = Table[Infinity, {2n - 2}, {n!}], jmshortevecs,
       jmshortinverses, jmevecs, jmevals, jmevecsinverses, newfactor, lrperm,
        s, k, side},

      (* make left - coset/right - coset change of basis matrix *)
      lrperm =
        RowPermutationMatrix[
          RankPermutationByCoset[#,
                1] & /@ (UnrankPermutationByCoset[#, n, 0] & /@
                Range[1, n!])];

      (* make chosen - order/right - coset change of basis matrix *)
      basisperm =
        RowPermutationMatrix[
          RankPermutationByCoset[#,
                1] & /@ (UnrankPermutationByCoset[#, n, order] & /@
                Range[1, n!])];

      (* do stages 2 at a time in loop, so we can reuse L objects in R stage *)


      For[s = 1, s <= stage, s++,
        k = Floor[(s + 3)/2];
        side = Mod[s + 1, 2];
        If[side == 0,
          (* L stage *)
          Print["Stage ", k, "L"];

          {jmshortevecs, jmevals} = JMEigensystem[k, k, 0];

          jmevecs =
            MatrixDirectSumSparse /@ (Table[#1, {n!/k!}] & /@ jmshortevecs);
          Print["Generated projection eigenbases."];

          jmshortinverses = Inverse[#1.Transpose[#1]] & /@ jmshortevecs;

          jmevecsinverses =
            MatrixDirectSumSparse /@ (Table[#1, {n!/k!}] & /@
                  jmshortinverses);
          Print["Generated projection inverses."];
```

```
          (* change basis of eigenbasis vectors to chosen one *)
          jmevecs = #.Transpose[basisperm] & /@ jmevecs;
          Print["Generated left-action eigenbases."];,

          (* R stage, if needed *)
          Print["Stage ", k, "R"];
          (* convert eigenbasis vectors to right - action,
            then change basis to chosen one *)
          jmevecs = #.basisperm.lrperm.Transpose[basisperm] & /@ jmevecs;
          Print["Generated right-action eigenbases."];
          ];

        Print[
          Timing[{dlist, devallist[[s]], newfactor} =
                  FFTFactor[jmevecs, jmevals, jmevecsinverses, dlist,
                    devallist[[s]]];][[1]]];
        devallist = FFTFactorizationEvalRecovery[devallist];
        Print[MatrixForm[devallist]];
        AppendTo[factorlist, newfactor];

        Print[{First[#], Length[#]} & /@ Split@Sort@devallist[[s]]];
        StageStatistics[dlist, devallist];
        ];

      {devallist, factorlist}
      ];
(*
Partitioned Routines
*)
(*
FFTFactorizationDC[n] generates full DFT decomposiiton for Sn using \
double-coset partitions.
*)
FFTFactorizationDC[n_Integer, order_Integer] /; n > 1 :=
    Module[{factorlist, invfactorlist, newinvfactorlist = {}, devallist,
        irreddims, i},
      {devallist, factorlist, invfactorlist} =
        FFTFactorizationToStageDC[n, 2*n - 3, order];

      devallist = ModifyFFTEigenvalueList[devallist];

      AppendTo[factorlist,
        RowPermutationMatrix[Flatten[Map[#1[[4]] &, devallist, {2}]]]];
      newinvfactorlist = {invfactorlist[[1]], Transpose[factorlist[[2]]]};
      For[i = 2, i <= Length[invfactorlist], i++,
        AppendTo[newinvfactorlist,
          SparseArray[
            Fold[Dot[factorlist[[#2]], #1] &, invfactorlist[[i]],
              Range[1, 2i - 2]]]];
        AppendTo[newinvfactorlist, Transpose[factorlist[[2i]]]];
        ];

      irreddims = Length /@ Union /@ Sort /@ (First /@ #1 &) /@ devallist;

      AppendTo[factorlist,
        FFTScalingMatrix[n, order, factorlist, irreddims]];
      AppendTo[newinvfactorlist, SparseDiagonalInverse[Last[factorlist]]];
      {devallist, factorlist, newinvfactorlist, irreddims}
```

```
      ];
(*
FFTFactorizationToStageDC[n, s, o] generates the DFT matrix decomposition to \
stage s for nested-double-coset order o, using double-coset partitions of \
frequency blocks.
*)
FFTFactorizationToStageDC[n_Integer, stage_Integer, order_Integer] /;
      n >= 2 && stage > 0 && stage <= 2*n - 3 :=
    Module[{factorlist = {}, invfactorlist = {}, newfactor, lrperm,
        basisperm,
        dlist = {SparseIdentityMatrix[n!]},
        dlistflat = SparseIdentityMatrix[n!],
        devallist = Table[Infinity, {2n - 2}, {n!}], dcbranchlist, dcindices,
        dclookup, biglist, neworder,
        jmshortevecs, jmevecs, jmevals, jmevecsinverses, s, k, side, time},

      (* construct double coset branching pattern for S_n *)
      dcbranchlist =
        Transpose[
          Map[SymmetricGroupDCBNesting[#, n, order] &, Range[1, n!]]];

      (* make left - coset/right - coset change - of - basis matrix *)
      lrperm =
        RowPermutationMatrix[
          RankPermutationByCoset[#,
                1] & /@ (UnrankPermutationByCoset[#, n, 0] & /@
                Range[1, n!])];

      (* make right - coset to chosen order change - of - basis matrix *)
      basisperm =
        RowPermutationMatrix[
          RankPermutationByCoset[#,
                1] & /@ (UnrankPermutationByCoset[#, n, order] & /@
                Range[1, n!])];

      (* tests to see which branching order we need to pass to projection \
builder *)
      (*Print[MatrixForm[dcbranchlist.Transpose[basisperm]]];
        Print[MatrixForm[dcbranchlist.basisperm]];
        Print[MatrixForm[dcbranchlist.basisperm.lrperm]];*)

      (* do stages 2 at a time in loop, so we can reuse L objects in R stage *)


      For[s = 1, s <= stage, s++,
        k = Floor[(s + 3)/2];
        side = Mod[s + 1, 2];

        (* Sort decompressor columns by eigenvalue list and double coset,
          then split into finer operators to pass to FFTFactor *)
        (* TODO : explain steps in more detail *)
        dcbranchlist = Drop[dcbranchlist, 1];
        biglist = Join[-devallist, dcbranchlist, {Range[1, n!]}];
        biglist =
          MapThread[
            Join[#1, {#2}] &, {Transpose[biglist], Transpose[dlistflat]}, 1];
        neworder = Ordering[biglist];
        biglist =
          Split[Sort[
```

```
      biglist], #1[[Range[1, 2n - 1]]] == #2[[Range[1, 2n - 1]]] &];
dlist = SparseArray[Transpose[Part[#, All, -1]]] & /@ biglist;
(* pick out double coset index for each d operator *)
dclookup = Part[#, 1, 2n - 1] & /@ biglist;
dcbranchlist = dcbranchlist[[All, neworder]];
devallist = devallist[[All, neworder]];
newfactor = RowPermutationMatrix[neworder];
If[newfactor != SparseIdentityMatrix[n!],
  AppendTo[factorlist, RowPermutationMatrix[neworder]];, Null
  ];
(* (* debug statements *)
  Print[biglist];
  Print[devallist];
  Print[dcbranchlist];
  Print[MatrixForm /@ dlist];*)

(* generate projection operators for decompressors *)
If[side == 0,
  (* L stage *)
  Print["Stage ", k, "L"];

  dcindices = ({Sort[First[dcbranchlist]]}.basisperm)[[1]];

  time = Timing[{jmshortevecs, jmevals} =
          JMEigensystem[k, k, 0]][[1]];
  Print["Generated sets of eigenvectors: " <> ToString[time]];


  jmevecs =
    Map[#.Transpose[basisperm] &,
      FFTDoubleCosetProjectionEigenbases[n, k, jmshortevecs,
        dcindices], {2}];
  Print["Generated left-action partitioned eigenbases."];


  time = Timing[
        jmshortinverses =
          Inverse[#1.Transpose[#1]] & /@ jmshortevecs][[1]];
  Print["Generated inverses: " <> ToString[time]];


  jmevecsinverses =
    FFTDoubleCosetProjectionInverses[n, k, jmshortinverses,
      dcindices];
  Print["Generated left-action partitioned inverses."];
  ,

  (* R stage, if needed *)
  Print["Stage ", k, "R"];

  dcindices = ({Sort[First[dcbranchlist]]}.basisperm.lrperm)[[1]];


  jmevecs =
    Map[#.lrperm.Transpose[basisperm] &,
      FFTDoubleCosetProjectionEigenbases[n, k, jmshortevecs,
        dcindices], {2}];
  Print["Generated right-action partitioned eigenbases."];
```

```
          jmevecsinverses =
            FFTDoubleCosetProjectionInverses[n, k, jmshortinverses,
              dcindices];
          Print["Generated right-action partitioned inverses."];
          ];

        (* (* debug statements *)
          Print[Map[MatrixForm[#] &, jmevecs, {2}]];
          Print[Map[MatrixForm[#.basisperm] &, jmevecs, {2}]];
          Print[Map[MatrixForm[#.basisperm.lrperm] &, jmevecs, {2}]];*)

        (*Print[jmevecs = (AppendColumns @@ #) & /@ jmevecs];

          Print[jmevecsinverses =
              MatrixDirectSumSparse /@ jmevecsinverses];*)

        StageStatistics[dlist, devallist];

        Print[
          "Projected eigenbases: " <>
            ToString[
              Timing[{dlist, devallist[[s]], newfactor} =
                      FFTFactorDC[jmevecs, jmevals, jmevecsinverses, dlist,
                        devallist[[s]], dclookup];][[1]]]];
        devallist = FFTFactorizationEvalRecovery[devallist];
        AppendTo[factorlist, newfactor];

        (*Print[{First[#], Length[#]} & /@ Split@Sort@devallist[[s]]];*)
        (*Print[MatrixForm[devallist]];*)
        dlistflat = SparseArray[AppendRows @@ dlist];
        AppendTo[invfactorlist, dlistflat];
        ];

      {devallist, factorlist, invfactorlist}
      ];
(*
FFTDoubleCosetProjectionEigenbases produces the double-coset-partitioned \
projection operators for S_k in S_n from S_k-eigenbases and from the \
double-coset branching at that stage.
*)
FFTDoubleCosetProjectionEigenbases[n_Integer, k_Integer,
      jmshortevecs : {_?MatrixQ ..}, dcbranchingarg : {_Integer ..}] :=
    Module[{dcbranching = First /@ Partition[dcbranchingarg, k!],
        jmshortsizes = Length /@ jmshortevecs, rowlists, jmevecs, dcevecs},
      jmevecs =
        MatrixDirectSumSparse /@ (Table[#1, {n!/k!}] & /@ jmshortevecs);
      rowlists =
        Map[#[[2]] &,
          Sort[Reap[
                MapThread[
                  Sow[#1, #2] &, {Range[1, Length[dcbranching]],
                    dcbranching}], _, List[#1, #2] &][[2]]], 1];
      rowlists =
        Outer[Function[{x, y}, Map[Sequence @@ Range[x # - x + 1, x #] &, y]],
          jmshortsizes, rowlists, 1];
      dcevecs =
        MapThread[Function[{x, y}, Map[x[[#]] &, y, 1]], {jmevecs, rowlists},
          1];
```

```
      dcevecs
      ];
(*
FFTDoubleCosetProjectionInverses produces the double-coset-partitioned \
projection operators for S_k in S_n from S_k-eigenbases and from the \
double-coset branching at that stage.
*)
FFTDoubleCosetProjectionInverses[n_Integer, k_Integer,
      jmshortinverses : {_?MatrixQ ..}, dcbranchingarg : {_Integer ..}] :=
    Module[{dcbranching = Sort[First /@ Partition[dcbranchingarg, k!]],
        dcinverses},
      dcbranching = Length /@ Split[dcbranching];
      Outer[MatrixDirectSumSparse[Table[#1, {#2}]] &, jmshortinverses,
        dcbranching, 1]
      ];
(*
FFTFactorizationEvalRecovery recovers eigenvalues for 1D Fourier spaces once \
enough eigenvalues are known. Returns repopulated list of eigenvalues.
*)
FFTFactorizationEvalRecovery[devallistarg_] :=
    Module[{devallist = devallistarg, devallistsplit, listlen, onedims,
        reapresult},
      listlen = Length[First[devallist]];
      devallistsplit =
        Split[Sort[Transpose[Append[devallist, Range[1, listlen]]]],
          Most[#1] == Most[#2] &];
      reapresult =
        Reap[Map[(If[Length[#1] == 1, Sow[Last[First[#]]]]; #1) &,
              devallistsplit, 1]][[2]];
      If[reapresult != {},
        onedims =
          Intersection[First[reapresult] ,
            Flatten[Position[Last[devallist], Infinity]]];
        Map[(devallist[[All, #]] =
                interleaveLists @@
                  CompleteJMEvalLists @@
                    splitList[DeleteCases[devallist[[All, #]], Infinity]]) &,
          onedims];
        ];
      devallist
      ];
(*
StageStatistics provides stats on the current state of the factor algorithm \
from the eigenvalues and decompression operators.
*)
StageStatistics[dlistarg_, devallistarg_] :=
    Module[{dlist = dlistarg, devallist = Transpose[devallistarg], reapresult,
         onedims, i, completeevals},
      Print["Number of Decompressors: ", Length[dlist]];
      Print[
        "Subspace Dimensions: ", {First[#], Length[#]} & /@
          Split@Sort@(Length /@ First /@ dlist)];
      (*i = 0;
        reapresult =
          Reap[Map[(i += #1; If[#1 == 1, Sow[i]]; #1) &,
                Length /@ First /@ dlist]][[2]];
        If[reapresult != {},
          onedims = reapresult[[1]];
```

```
          completeevals =
            Map[CompleteJMEvalLists @@ # &,
              splitList /@ devallist[[onedims]]];

          Print["Eigenvalues of 1D Subspaces: ",
            MapThread[
              List, {onedims, splitList /@ devallist[[onedims]],
                completeevals, Map[Length, completeevals, {2}]}]];,
          Null];*)

      (*Print[devallist[[onedims]]];*)
      ];
(*
Factor Construction
*)
(*
RREFDecomp[A] decomposes an m-by-n matrix A of rank r into an m-by-r matrix D \
and a k-by-n matrix C such that A = DC.
*)
RREFDecomp[(A_)?MatrixQ] :=
    Module[{B, rank, Am, An, C, D}, rank = MatrixRank[A];
      If[rank <= 0, {{{}}, {{}}},
        {Am, An} = Dimensions[A];
        B = RowReduce[Transpose[Join[Transpose[A], IdentityMatrix[Am]]]];
        C = B[[Range[1, rank], Range[1, An]]];
        D = Inverse[B[[All, Range[An + 1, An + Am]]]][[All, Range[1, rank]]];
        {C, D}]
      ];
RREFDecomp2[{{}}] = {{{}}, {{}}};
RREFDecomp2[(A_)?MatrixQ] := Module[{B, rank, Am, An, C, D},
      {Am, An} = Dimensions[A];
      If [Am < 1.5An,
        {C, D} = RREFDecomp[A],
        {D, C} = RREFDecomp[Transpose[A]];
        If[C != {{}},
          {C, B} = RREFDecomp[Transpose[C]];
          D = SmartDot[Transpose[D], B];];
        ];
      {C, D}
      ];
(*
Old FFT Factoring
*)
(*
FFTFactor takes the eigenbases and eigenvalues for the JM element and a set \
of decompression matrices and returns a new set of decompression matrices and \
the next factor in the FFT.
*)
FFTFactor[evecsarg_, evalsarg_, evecsinversesarg_, decompresslist_,
      devalsarg_] :=
    Module[{evecs = evecsarg, evals = evalsarg,
        evecsinverses = evecsinversesarg, devals = devalsarg,
        evecdecompresslist, clist, csublist, dnewlist = {}, drank, projranks,
        deadprojindices, crank, currank = 0, evallookup, C, D, i, j},

      evecdecompresslist =
        MapThread[Transpose[#1].#2 &, {evecs, evecsinverses}];
      projranks = MatrixRank /@ evecs;
      clist = {};
```

```
      MapThread[(evallookup[#1] = #2) &, {evals, Range[1, Length[evals]]}];

      For[i = 1, i <= Length[decompresslist], i++,
        csublist = {};
        drank = MatrixRank[decompresslist[[i]]];
        If[drank == 1,
          currank++;
          (*Print[
                "1D Eigenvalue at " <> ToString[currank] <> ": " <>
                  ToString[devals[[currank]]]];*)
          j = evallookup[devals[[currank]]];
          projranks[[j]]--;
          csublist = {{{1}}};
          AppendTo[dnewlist, decompresslist[[i]]];,
          For[j = 1, j <= Length[evecs] && drank > 0, j++,
              {C, D} = RREFDecomp[evecs[[j]].decompresslist[[i]]];
              crank = MatrixRank[C];
              If[crank > 0,
                AppendTo[csublist, C];
                AppendTo[dnewlist, SmartDot[evecdecompresslist[[j]], D]];

                devals[[Range[currank + 1, currank + crank]]] =
                  Table[evals[[j]], {crank}];
                drank -= crank;
                projranks[[j]] -= crank;
                currank += crank;
                (*projranks[[j]] =
                      ReplacePart[projranks, projranks[[j]] - crank, j];*),
                Null
                ];
              (* Print[{i, j}]; *)
              ];
          ];

          (*
            cull out projection matrices that have already been completely \
projected *)
          deadprojindices = Flatten[Position[projranks, 0]];
          (*Print["Indices to be culled: " <> ToString[deadprojindices]];*)
          If[Length[deadprojindices] > 0,

            Map[(projranks[[#]] = Null; evecs[[#]] = Null;
                  evecdecompresslist[[#]] = Null; evals[[#]] = Null;) &,
              deadprojindices];

            projranks = DeleteCases[projranks, Null];
            evecs = DeleteCases[evecs, Null];
            evecdecompresslist = DeleteCases[evecdecompresslist, Null];
            evals = DeleteCases[evals, Null];


            MapThread[(evallookup[#1] = #2) &, {evals,
                Range[1, Length[evals]]}];,
            Null];

          (*For[j = 1, j <= Length[projranks], j++,
                If[projranks[[j]] == 0,
                    (*Print[j];*)
```

```
                projranks = Delete[projranks, j];
                evecs = Delete[evecs, j];
                evecdecompresslist = Delete[evecdecompresslist, j];
                evals = Delete[evals, j];
                j--;
                ];
            ];*)
        (*Print[projranks];
          Print[evecs];
          Print[evecdecompresslist];
          Print[evals];*)

        AppendTo[clist, SmartColumnAppend @@ csublist];
        ];

      dnewlist = DeleteCases[dnewlist, _?MatrixNullQ];
      dnewlist = SparseArray /@ dnewlist;
      {dnewlist, devals, MatrixDirectSumSparse[clist]}
      ];
(*
DC-Partition FFT Factoring
*)
(*
FFTFactorDC takes the eigenbases and eigenvalues for the JM element and a set \
of decompression matrices and returns a new set of decompression matrices and \
the next factor in the FFT.
** TODO: revise documentation once this routine is optimized for double coset \
partitions **
*)
FFTFactorDC[evecsarg_, evalsarg_, evecsinversesarg_, decompresslist_,
      devalsarg_, dclookuparg_] :=
    Module[{evecs = evecsarg, evals = evalsarg,
        evecsinverses = evecsinversesarg, devals = devalsarg,
        dclookup = dclookuparg, evecdecompresslist, clist, csublist,
        dnewlist = {}, drank, projranks, crank, evallookup, dcindex,
        currank = 0, CD, C, D, i, j},
      evecdecompresslist =
        MapThread[Transpose[#1].#2 &, {evecs, evecsinverses}, 2];
      projranks = Map[Length, evecs, {2}];
      clist = {};

      For[i = 1, i <= Length[decompresslist], i++,
        csublist = {};
        (*Print[i];*)
        drank = Length[Transpose[decompresslist[[i]]]];
        dcindex = dclookup[[i]];

        If[devals[[currank + 1]] != Infinity,
          currank++;

          Print["1D Eigenvalue at " <> ToString[currank] <> ": " <>
              ToString[devals[[currank]]]];
          (*j = evallookup[devals[[currank]]];
            projranks[[j]]--;*)
          csublist = {{{1}}};
          AppendTo[dnewlist, decompresslist[[i]]];,
          For[j = 1, j <= Length[evecs] && drank > 0, j++,
              CD = evecs[[j, dcindex]].decompresslist[[i]];
              (*Print[
```

```
                Dimensions[CD]];*)(*, MatrixForm[CD],
                  MatrixForm[evecs[[j, dcindex]]],
                  MatrixForm[decompresslist[[i]]]};*)
            {C, D} = RREFDecomp2[CD];

            If[C != {{}},
              crank = Length[C];
              AppendTo[csublist, C];

              AppendTo[dnewlist,
                SmartDot[evecdecompresslist[[j, dcindex]], D]];

              devals[[Range[currank + 1, currank + crank]]] =
                Table[evals[[j]], {crank}];
              currank += crank;
              drank -= crank;
              ];
            (* Print[{i, j}]; *)
            ];
          ];

        AppendTo[clist, SmartColumnAppend @@ csublist];
        ];

      dnewlist = DeleteCases[dnewlist, _?MatrixNullQ];
      dnewlist = SparseArray /@ dnewlist;
      {dnewlist, devals, MatrixDirectSumSparse[clist]}
      ];
(*
Post-Factorization Eigenvalue Manipulation
*)
ModifyFFTEigenvalueList[devallist_] :=
    Module[{devalsplitlist, devallastrow, lastletterordersort},
      (* split lists of eigenvalues into lists for left, right actions *)
      (* TODO : eliminate Drop once DC eval recovery finished *)
      (* devalsplitlist = splitList[Drop[devallist, {-1}]];*)
      devalsplitlist = splitList[devallist];

      (* compute list of eigenvalues for last right action based on \
differences of previous eigenvalues and add to master list *)
      (* devallastrow =
          Plus @@ devalsplitlist[[1]] - Plus @@ devalsplitlist[[2]];
        AppendTo[devalsplitlist[[2]], devallastrow];*)

      (* add row number,
        sorted list of eigenvalues to each entry in master list *)
      (* transpose eigenvalue list structure to create L/
            R lists for each row *)
      devalsplitlist = Transpose /@ devalsplitlist;
      devalsplitlist =
        MapThread[
          Append[#1, #2] &, {MapThread[{#1, #2, Sort[#1]} &, devalsplitlist],
            Range[1, Length[devalsplitlist[[1]]]]}];

      (* sort master list first based on sorted eval list
          (in order to sort by irreducible) *)
      devalsplitlist = Sort[devalsplitlist, OrderedQ[{#2[[3]], #1[[3]]}] &];

      (* split into groups by same irreducible *)
```

```
      devalsplitlist = Split[devalsplitlist, #1[[3]] === #2[[3]] &];

      (* sort each group into last -
          letter order (equiv. to lex. order on reversed eval list) *)
      lastletterordersort[x_] :=
        Sort[x, OrderedQ[{Reverse[Join[#2[[2]], #2[[1]]]],
                Reverse[Join[#1[[2]], #1[[1]]]]}] &];
      devalsplitlist = lastletterordersort /@ devalsplitlist;

      (* return sorted, modified master list *)
      devalsplitlist];
(*
Scaling Matrix Construction
*)
FFTScalingMatrix[n_Integer, order_Integer, fftfactors_List,
      irredrepsizes_List] :=
    Module[{matrixsize, scalingentries, remainingrows, nextrow, nextcolumn,
        currentscalingmatrix, fftvalues, snvalues, nonzerolocs, newscalings,
        genmatrices, gencolumns, i},
      matrixsize = n!;
      scalingentries = Table[1, {matrixsize}];
      remainingrows = Range[1, matrixsize];
      currentscalingmatrix = SparseDiagonalMatrix[scalingentries];

      (* make sure all entries for identity are 1 *)
      nextcolumn = RankPermutationByCoset[Range[1, n], order];
      fftvalues =
        Flatten[Dot @@ Reverse[fftfactors].ColumnSelectionMatrix[{nextcolumn},
                matrixsize]];
      nonzerolocs =
        Reap[MapThread[(If[#1 == 0, Null, Sow[#2]]; #1) &, {fftvalues,
                   Range[1, matrixsize]}]][[2]][[1]];
      nonzerolocs = Intersection[nonzerolocs, remainingrows];
      scalingentries[[nonzerolocs]] =
        MapThread[#1/#2 &, {Table[1, {Length[nonzerolocs]}],
            fftvalues[[nonzerolocs]]}];
      remainingrows = Complement[remainingrows, nonzerolocs];
      currentscalingmatrix = SparseDiagonalMatrix[scalingentries];

      (* get scalings for (i i + 1) transpositions *)
      {genmatrices, gencolumns} =
        GenSNMatrices[n, order, fftfactors, currentscalingmatrix,
          irredrepsizes];

      (* TODO : remove debugging Print statements *)
      For[i = 1, i <= Length[genmatrices], i++,
        fftvalues =
          Flatten[currentscalingmatrix.(Dot @@
                    Reverse[
                      fftfactors].ColumnSelectionMatrix[{gencolumns[[i]]},
                    matrixsize])];
        snvalues = Flatten[genmatrices[[i]]];
        nonzerolocs =
          Reap[MapThread[(If[#1 == 0, Null, Sow[#2]]; #1) &, {fftvalues,
                    Range[1, matrixsize]}]][[2]][[1]];
        nonzerolocs = Intersection[nonzerolocs, remainingrows];
        newscalings =
          MapThread[#1/#2 &, {snvalues[[nonzerolocs]],
              fftvalues[[nonzerolocs]]}];
```

```
        scalingentries[[nonzerolocs]] = newscalings;
        remainingrows = Complement[remainingrows, nonzerolocs];
        Print[
          "Number of rows remaining to scale: " <>
            ToString[Length [remainingrows]]];
        (* Print["Rows remaining to scale:" <> ToString[remainingrows]]; *)
];

      (* compute scalings for remaining entries *)
      While[Length[remainingrows] > 0,
        nextrow = First[remainingrows];

        (* select next row from FFT matrix *)
        rowvalues =
          Flatten[Transpose[
                ColumnSelectionMatrix[{nextrow}, matrixsize]].Dot @@
                Reverse[fftfactors]];
        nextcolumn = 1;
        While[rowvalues[[nextcolumn]] == 0, nextcolumn++];
        Print["Next column to scale: " <> ToString[nextcolumn]];
        fftvalues =
          Flatten[currentscalingmatrix.(Dot @@
                    Reverse[fftfactors].ColumnSelectionMatrix[{nextcolumn},
                    matrixsize])];
        snvalues =
          Flatten[SNMatricesFromGens[n, order, nextcolumn, genmatrices]];
        nonzerolocs =
          Reap[MapThread[(If[#1 == 0, Null, Sow[#2]]; #1) &, {fftvalues,
                    Range[1, matrixsize]}]][[2]][[1]];
        nonzerolocs = Intersection[nonzerolocs, remainingrows];
        newscalings =
          MapThread[#1/#2 &, {snvalues[[nonzerolocs]],
              fftvalues[[nonzerolocs]]}];
        scalingentries[[nonzerolocs]] = newscalings;
        remainingrows = Complement[remainingrows, nonzerolocs];
        Print[
          "Number of rows remaining to scale: " <>
            ToString[Length [remainingrows]]];
        (* Print["Rows remaining to scale:" <> ToString[remainingrows]]; *)
        ];

      SparseDiagonalMatrix[scalingentries]
      ];
(*
GenSNMatrices creates the lists of Young's seminormal repn matrices for the \
generating tranpositons (i i+1) from the FFT matrix factors and the degrees \
of the irreducibles.
*)
GenSNMatrices[n_Integer, order_Integer, fftfactors_, currentscalingmatrix_,
      irredrepsizes_] :=
    Module[{genindices, gencolumns, genmatrices, snmatrices, matrixsize},
      matrixsize = n!;

      genindices =
        Map[RankPermutationByCoset[#, order] &,
          Table[Transposition[n, i, i + 1], {i, 1, n - 1}]];
      gencolumns =
        Transpose[
          currentscalingmatrix.(Dot @@
```

```
                Reverse[fftfactors].ColumnSelectionMatrix[genindices,
                matrixsize])];
      genmatrices = (VectorToBlocks[#1, irredrepsizes] &) /@ gencolumns;
      (* Debug : print matrices we generate correct forms for
            Print[Map[MatrixForm, genmatrices, {2}]];*)
      snmatrices = Map[IrredSNMatrix, genmatrices, {2}];
      (* Debug : print correct seminormal matrices for transpositions
            Print[Map[MatrixForm, snmatrices, {2}]];*)
      {snmatrices, genindices}
      ];
(*
IrredSNMatrix computes the Young's seminormal matrix representation of a \
generator of S_n from its diagonal (which is assumed to be correct).
*)
IrredSNMatrix[matrix_] :=
    Module[{nonzerolist, diagelems, snentries, dim},
      dim = Dimensions[matrix][[1]];
      diagelems = (matrix[[#1, #1]] &) /@ Range[1, dim];
      nonzerolist =
        Reap[MapThread[
                If[#1 == 0, Null, Sow[#2]] &, {matrix,
                  Table[{i, j}, {i, 1, dim}, {j, 1, dim}]}, 2]][[2]][[1]];
      snentries =
        Map[If[#1[[1]] != #1[[2]], 1 + diagelems[[#1[[1]]]],
              diagelems[[#1[[1]]]]] &, nonzerolist, {1}];
      SparseArray[nonzerolist -> snentries, {dim, dim}]
      ];
(*
SNMatricesFromGens computes the Young's seminormal matrix representation for \
an element of S_n from the seminormal matrices of the generators.
*)
SNMatricesFromGens[n_Integer, order_Integer, rank_Integer, genmatrices_] :=
    Module[{perm, rcrank, decomp, snmatrices},
      (* convert rank to perm under specified order,
        then to rank under right coset order *)
      perm = UnrankPermutationByCoset[rank, n, order];
      rcrank = RankPermutationByCoset[perm, 1];
      (* TODO :
          Explain decomposition algorithm in greater detail wrt new right \
coset order, including - sign on Floor *)
      decomp = ({Mod[-Floor[(rcrank - 1)/#1!], #1 + 1], #1 + 1} &) /@
          Range[1, n - 1];
      decomp = DeleteCases[decomp, {0, _}];
      (* start with identity matrix *)
      snmatrices = MapThread[Dot, {genmatrices[[1]], genmatrices[[1]]}];
      If[Length[decomp] > 0,
        snmatrices =
          MapThread[Dot,
            Map[SNTransposMatrices[#1[[1]], #1[[2]], genmatrices] &, decomp,
              1]], snmatrices =
          MapThread[Dot, {genmatrices[[1]], genmatrices[[1]]}]];
      snmatrices];
(*
SNTransposMatrices computes the Young's seminormal matrix representation for \
a transposition of S_n from the seminormal matrices of the generators.
*)
SNTransposMatrices[i_Integer, j_Integer, genmatrices_] /; i < j :=
    Module[{indices},
      indices = Join[Range[i, j - 1]];
```

```
      indices = Join[indices, Reverse[Delete[indices, -1]]];
      MapThread[Dot, (genmatrices[[#1]] &) /@ indices]
      ];
(*
FFT Evaluation
*)
(*
Convolution Operations
*)
VectorToBlocks[(v_)?VectorQ, (blocksizes_)?(VectorQ[#1, IntegerQ] &)] /;
      Length[v] == Plus @@ (#1^2 &) /@ blocksizes :=
    Module[{vblocks, blocksquares},
      blocksquares = (Plus @@
                Function[{x}, x^2] /@ blocksizes[[Range[1, #1]]] &) /@
          Range[1, Length[blocksizes]];
      vblocks =
        MapThread[
          v[[Range[#1 + 1, #2]]] &, {Join[{0},
              blocksquares[[Range[1, Length[blocksquares] - 1]]]],
            blocksquares}];
      vblocks = MapThread[Partition[#1, #2] &, {vblocks, blocksizes}]];

ConvolveInFourier[(x_)?VectorQ, (y_)?
        VectorQ, (blocksizes_)?(VectorQ[#1, IntegerQ] &),
      changeofbasis : {_?MatrixQ ..}] :=
    Module[{xhat, yhat, product},
      xhat = Fold[#1.Transpose[#2] &, {x}, changeofbasis][[1]];
      yhat = Fold[#1.Transpose[#2] &, {y}, changeofbasis][[1]];
      product =
        MapThread[
          Dot, {VectorToBlocks[xhat, blocksizes],
            VectorToBlocks[yhat, blocksizes]}]];

ConvolveInStandard[(x_)?VectorQ, (y_)?
        VectorQ, (blocksizes_)?(VectorQ[#1, IntegerQ] &),
      changeofbasis : {_?MatrixQ ..}, inverses : {_?MatrixQ ..}] :=
    Module[{xhat, yhat, product},
      xhat = Fold[#1.Transpose[#2] &, {x}, changeofbasis][[1]];
      yhat = Fold[#1.Transpose[#2] &, {y}, changeofbasis][[1]];
      product =
        MapThread[
          Dot, {VectorToBlocks[xhat, blocksizes],
            VectorToBlocks[yhat, blocksizes]}];
      product =
        Fold[#1.Transpose[#2] &, {Flatten[product]},
            Reverse[inverses]][[1]]];

ConvolveByPermutations[(x_)?VectorQ, (y_)?VectorQ, n_Integer, order_Integer] :=
     Module[{xf, yf, products}, xf = PadRight[x, n!][[Range[1, n!]]];
      yf = PadRight[y, n!][[Range[1, n!]]];
      products =
        Sort[Flatten[
            Outer[{#1[[1]]*#2[[1]],
                  RankPermutationByCoset[
                    Permute[UnrankPermutationByCoset[#1[[2]], n, order],
                      UnrankPermutationByCoset[#2[[2]], n, order]], order]} &,
              MapThread[{#1, #2} &, {xf, Range[1, n!]}],
              MapThread[{#1, #2} &, {yf, Range[1, n!]}], 1], 1],
          OrderedQ[{#1[[2]], #2[[2]]}] &];
```

```
      products =
        Partition[products, n!]; (Plus @@ #1 &) /@ (First /@ #1 &) /@
          products];
(*
Operation Counting
*)
RowAdditions[(x_)?VectorQ] := Length[DeleteCases[Normal[x], 0]] - 1;
RowMultiplications[(x_)?VectorQ] := Length[DeleteCases[Normal[x], 0 | 1 | \
-1]];
RowCombinedOps[(x_)?VectorQ] := Max[RowAdditions[x], RowMultiplications[x]];
MatrixAdditions[(A_)?MatrixQ] := Plus @@ RowAdditions /@ A;
MatrixMultiplications[(A_)?MatrixQ] := Plus @@ RowMultiplications /@ A;
MatrixCombinedOps[(A_)?MatrixQ] := Plus @@ RowCombinedOps /@ A;
(*
End of Package
*)
EndPackage[]
```

Bibliography

[1] Adkins, W. A. and Weintraub, S. H. (1992). *Algebra: An Approach via Module Theory*. Number 136 in Graduate Texts in Mathematics. Springer, New York.

[2] Brown, H., Hjelmeland, L., and Masinter, L. (1972). Constructive graph labeling using double cosets. *Discrete Mathematics*, 7:1–30.

[3] Chirikjian, G. S. and Kyatkin, A. B. (2001). *Engineering Applications of Noncommutative Harmonic Analysis: With Emphasis on Rotation and Motion Groups*. CRC Press, Boca Raton, FL.

[4] Clausen, M. (1989). Fast generalized Fourier transforms. *Theoretical Computer Science*, 67:55–63.

[5] Clausen, M. (2001). Elements of a general algebraic theory of standard tableaux. In Betten, A., Kohnert, A., Laue, R., and Wassermann, A., editors, *Algebraic Combinatorics and Applications*, pages 67–78, Berlin. Springer-Verlag.

[6] Clausen, M. and Baum, U. (1993a). *Fast Fourier Transforms*. BI-Wissenschaftsverlag, Mannheim, Germany.

[7] Clausen, M. and Baum, U. (1993b). Fast Fourier transforms for symmetric groups: Theory and implementation. *Mathematics of Computation*, 61(204):833–847.

[8] Diaconis, P. (1989). A generalization of spectral analysis with application to ranked data. *The Annals of Statistics*, 17(3):949–979.

[9] Driscoll, J. R. and Healy, Jr., D. M. (1994). Computing Fourier transforms and convolutions on the 2-sphere. *Advances in Applied Mathematics*, 15:202–250.

[10] Drozd, Y. A. and Kirichenko, V. V. (1994). *Finite Dimensional Algebras*. Springer-Verlag, New York.

[11] Dummit, D. and Foote, R. (1999). *Abstract Algebra*. John Wiley and Sons, New York.

[12] Eisberg, R. and Resnick, R. (1985). *Quantum Physics of Atoms, Molecules, Solids, Nuclei, and Particles*. John Wiley and Sons, New York.

[13] Gross, K. I. (1978). On the evolution of noncommutative harmonic analysis. *Am. Math. Monthly*, 85:525–548.

[14] James, G. and Kerber, A. (1981). *The Representation Theory of the Symmetric Group*. Addison-Wesley, Reading, MA.

[15] Jucys, A. A. (1966). On the Young operators of symmetric groups. *Lithuanian Physics Journal*, 6:163–180.

[16] Lafferty, J. D. and Rockmore, D. N. (1997). Spectral techniques for expander codes. In *ACM Symposium on Theory of Computing (STOC '97)*.

[17] Logan, J. D. (1998). *Applied Partial Differential Equations*. Undergraduate Texts in Mathematics. Springer-Verlag, New York.

[18] Maslen, D. K. (1998). The efficient computation of Fourier transforms on the symmetric group. *Mathematics of Computation*, 67(223):1121–1147.

[19] Maslen, D. K., Orrison, M. E., and Rockmore, D. N. (2004). Computing isotypic projections with the Lanczos iteration. *SIAM J. Matrix Anal. Appl.*, 25(3):784–803.

[20] Maslen, D. K. and Rockmore, D. N. (1997). Generalized FFTs - a survey of some recent results. *DIMACS Series in Discrete Mathematics and Computer Science*, 28:183–237.

[21] Murphy, G. E. (1981). A new construction of Young's seminormal representation of the symmetric groups. *Journal of Algebra*, 69:287–297.

[22] Murphy, G. E. (1983). The idempotents of the symmetric group and Nakayama's conjecture. *Journal of Algebra*, 81:58–265.

[23] Okounkov, A. and Vershik, A. (1996). A new approach to representation theory of symmetric groups. *Selecta Mathematica*, 2(4):581–605.

[24] Pemmaraju, S. and Skiena, S. (2003). *Computational Discrete Mathematics: Combinatorics and Graph Theory with* Mathematica. Cambridge University Press, Cambridge.

[25] Püschel, M., Moura, J. M. F., Johnson, J., Padua, D., Veloso, M., Singer, B. W., Xiong, J., Franchetti, F., Gačić, A., Voronenko, Y., Chen, K., Johnson, R. W., and Rizzolo, N. (2005). SPIRAL: Code generation for DSP transforms. *Proceedings of the IEEE, special issue on "Program Generation, Optimization, and Adaptation"*, 93(2). to appear.

[26] Ram, A. (1997). Seminormal representations of Weyl groups and Iwahori-Hecke algebras. *Proc. London Math. Soc.*, 73:99–133.

[27] Rockmore, D. N. (2003). Recent progress and applications in group FFTs. NATO Advanced Study Institute on Computational Noncommutative Algebra and Applications.

[28] Sagan, B. E. (2001). *The Symmetric Group: Representations, Combinatorial Algorithms, and Symmetric Functions*. Number 203 in Graduate Texts in Mathematics. Springer, New York.

[29] Serre, J.-P. (1977). *Linear Representations of Finite Groups*. Number 42 in Graduate Texts in Mathematics. Springer, New York.

[30] Sloane, N. J. A. (1999). Sequence A000041 in *The Online Encyclopedia of Integer Sequences*. Found at `http://www.research.att.com/~njas/sequences/`.

[31] Townsend, J. S. (2000). *A Modern Approach to Quantum Mechanics*. University Science Books, Sausolito, CA.