JOURNAL OF THE
AMERICAN MATHEMATICAL SOCIETY
Volume 3, Number 2, April 1990

EFFICIENT COMPUTATION OF THE FOURIER TRANSFORM ON FINITE GROUPS

PERSI DIACONIS AND DANIEL ROCKMORE

1. Introduction

Let G be a finite group. Let $f: G \to \mathbf{C}$ be a function. To each irreducible representation ρ of G there is associated a Fourier transform

$$\hat{f}(\rho) = \sum_{s \in G} f(s)\rho(s).$$

If ρ is given in matrix form of dimension d_ρ, direct computation of $\hat{f}(\rho)$ involves order $|G|d_\rho^2$ operations (multiplications and additions). Direct computation of the Fourier transform for all irreducible representations involves order $|G|\sum_\rho d_\rho^2 = |G|^2$ operations.

For Abelian groups, such as the integers mod n or the group of binary n-tuples, substantial speedups have been achieved. Fast Fourier transforms allow computation in order $|G|\log|G|$ operations. These ideas have been important in theoretical computer science where they have been applied to give fast algorithms for a variety of tasks ranging from multiplication of numbers to evaluation of polynomials. The algorithms offer substantial practical savings, and have revolutionized spectral analysis throughout engineering and the sciences. Aho, Hopcroft, and Ullman (1976), and Elliott and Rao (1982) contain reviews with extensive pointers to the literature.

This paper presents several algorithms that allow impressive speedups in theory and practice. The basic ideas are outlined in §2 which also shows how they reduce to the Cooley-Tukey algorithm for the integers mod n.

In brief, if a representation ρ is restricted to a subgroup, it splits up into irreducible representations. With an appropriate basis, ρ can be written in block diagonal form and the transform at ρ can be built up as a direct sum of transforms over the subgroup. This can be iterated. It yields a family of algorithms that all take fewer operations than direct computation.

Received by the editors June 2, 1988; presented to the Society at the Special Session on Combinatorics and Group Representations, Atlanta, Georgia, January 1988.

1980 *Mathematics Subject Classification* (1985 *Revision*). Primary 20C15.

The authors were supported by the National Science Foundation under Grant DMS 86-00235 (Diaconis) and a Graduate Fellowship (Rockmore).

We have recently had to compute Fourier transforms on the symmetric group S_n as part of a statistical analysis of rankings in an election. Here n is the number of candidates, and $f(\pi)$ is the number of voters choosing the rank order π. To understand the use of transforms, take $\rho(\pi)$ as the usual permutation matrix having a one in position (i, j) if $\pi(i) = j$. Then the (i, j) entry of

$$\hat{f}(\rho) = \sum_{\pi} f(\pi)\rho(\pi)$$

counts how many voters ranked candidate i in position j. These constitute a first order summary of the data. Other representations of S_n are useful in understanding higher order aspects of data such as coalitions. Irreducible representations allow a natural disentangling of levels—one would like to look at "pure" higher order effects with the lower order structure subtracted out. Diaconis (1988, 1989) presents detailed examples and a host of other applied problems where such spectral analysis is important.

As indicated above, direct computation of $\hat{f}(\rho)$ for all irreducible representations of S_n involves order $(n!)^2$ operations. This is already prohibitive for $n = 10$. Even for smaller n, Monte Carlo comparison of statistical procedures involves repeated computation of transforms. Efficient algorithms are mandatory.

Section 3 develops the basic ideas for the symmetric group. We use the number of additions and multiplications as a measure of speed. This assumes that the matrices $\rho(\pi)$ are precomputed and available. The running time of the algorithm can then be approximated as in the following theorem.

Theorem. *Let $T(n)$ be the number of operations required to compute the Fourier transforms of a function on S_n at all irreducible representations. Then*

$$T(n) \leq A(n!)^{a/2} n e^{-(a-2)c\sqrt{n/2}}$$

with A a positive computable constant, $c = .1156$, and $a > 2$ the exponent for matrix multiplication (multiplying $d \times d$ matrices takes d^a operations).

Remarks. In practice, $a = 3$. Currently the best theoretical result is $a = 2.38$ (see Coppersmith and Winograd (1987)). Taking $a = 2$, a separate argument yields $T(n) \leq n!\binom{n}{2}$ which is essentially the $|G| \log |G|$ of the Abelian case. The proof of Theorem 1 uses results of Logan-Shepp (1977) and Vershik-Kerov (1985) on the largest dimension of a representation in S_n.

Efficient implementation of the basic idea requires a careful choice of basis. The seminormal or orthogonal forms introduced by Alfred Young work perfectly. These are explained in §4.

In applications, one sometimes only needs the Fourier transform at a subset of representations. The algorithm adapts nicely to such situations, still offering large improvements over direct computation. These ideas may be useful even in the Abelian case, where there is a lot of data and one wishes to see the Fourier transform on a grid of equally spaced points. These ideas are explored in §5.

Section 6 gives several implementations of the basic algorithm on S_n in an Algol-like language. These offer trade-offs between storage, running time, ease of implementation, and start up costs—for example, representing matrices for pairwise adjacent transpositions may be easily available but the others may be built up in various ways.

The asymptotics of running times are supplemented by some exact counts of operations in §7. In S_9, a practical version of the algorithm is shown to speed things up by a factor of 100.

The final section briefly discusses applications, adaptation of some of the other ideas used to speed up the usual discrete Fourier transform, parallelizability, and some open problems. It also gives a brief indication of how the algorithms suggested here can be "run backward" to compute the inverse Fourier transform.

Our work has been developed independently of the pioneering work of T. Beth (1987), the first serious study of computational aspects of noncommutative transforms. Following work of Atkinson (who discussed product groups) Beth develops results using "Clifford theory". This analyzes representations of G using *normal* subgroups. Beth develops ideas like our equation (2.3) and Proposition 1 and shows how they yield the Cooley-Tukey algorithm when specialized to cyclic groups.

Beth's work is an important step toward developing a general algorithm. The examples here show one must go further. For simple groups like the alternating groups, or groups with a limited decomposition into normal subgroups like the symmetric group, Clifford theory is of little help.

Beth treats problems not treated here, such as efficient computation in the group algebra. He suggests novel applications of noncommutative transforms to problems of coding and computer vision.

2. The general idea

Let G be a finite group. A *representation* ρ assigns invertible matrices to group elements in such a way that $\rho(st) = \rho(s)\rho(t)$ for $s, t \in G$. Thus ρ is a homomorphism from G to $\mathrm{GL}(V)$ with V a vector space of dimension d_ρ—the *dimension* or *degree* of ρ. Serre (1977) is an accessible introduction to basic representation theory.

A representation ρ is *irreducible* if the only subspaces $W \subset V$ such that $\rho(s)W \subset W$ for all $s \in G$ are $\{0\}$ and V. The dimensions of the irreducible representations satisfy

$$\sum_\rho d_\rho^2 = |G|, \tag{2.1}$$

so the largest dimension is less than $\sqrt{|G|}$.

The *Fourier transform of a function* f *at representation* ρ is defined as

$$\hat{f}(\rho) = \sum_{s \in G} f(s)\rho(s). \tag{2.2}$$

The transforms of f at all irreducibles determine f though the inversion formula:

$$f(s) = \frac{1}{|G|} \sum_{\rho} d_\rho \operatorname{Tr}\{\rho(s^{-1})\hat{f}(\rho)\}.$$

As usual, the transform turns convolution into product:

$$\widehat{f * g}(\rho) = \hat{f}(\rho)\hat{g}(\rho) \quad \text{with } f * g(s) = \sum_{t} f(st^{-1})g(t).$$

In a variety of applications $\hat{f}(\rho)$ is required for all irreducible ρ. Suppose for now that all the representing matrices $\rho(s)$, for all s and ρ, are precomputed and stored in memory. Direct computation of $\hat{f}(\rho)$ through the definition requires order $|G|d_\rho^2$ multiplications followed by additions. We will say $|G|d_\rho^2$ operations. Summing in ρ, using (2.1), shows $|G|^2$ operations are required.

The new algorithms all use a subgroup $H \subset G$. Say $k = |G|/|H|$. Choose coset representatives $s_1, s_2, \ldots, s_k$ for H in G. The transform can be written

$$\hat{f}(\rho) = \sum_{i=1}^{k} \rho(s_i) \sum_{h \in H} f_i(h)\rho(h) \tag{2.3}$$

with $f_i(h) = f(s_i h)$.

The restriction of ρ to H is usually not irreducible. When V is a complex vector space this means that there are subspaces V_i such that

$$V = V_1 \oplus V_2 \oplus \cdots \oplus V_r$$

and $\rho(h)$ restricted to V_i is an irreducible representation of H for each i. Thus restricting ρ to H, ρ splits as $m_1\rho_1' + \cdots + m_j\rho_j'$, with ρ_i' distinct irreducible representations of H and m_i the multiplicity of ρ_i'. It follows that in a suitable basis, $\rho(h)$ can be written as the block diagonal matrix

$$\begin{pmatrix} \rho_1'(h) & & \\ & \ddots & \\ & & \rho_j'(h) \end{pmatrix}. \tag{2.4}$$

Clearly, the Fourier transform $\hat{f}_i(\rho')$ determines $\hat{f}(\rho)$ as ρ' ranges over irreducibles of H and $1 \le i \le k$. The pieces can be assembled to get the inner sum in (2.3). This is multiplied by the matrices $\rho(s_i)$ for $1 \le i \le k$, and summed to give $\hat{f}(\rho)$.

The point is, $\hat{f}_i(\rho')$ appears in several blocks as i and ρ' vary and these need only be computed once. This is the heart of the savings. Of course, the idea can be applied inductively to amplify itself. Before discussing this, let us record the recurrence implied by (2.3).

Proposition 1. *Let G be a group and H a subgroup. Let $T(G)$ and $T(H)$ be the number of operations needed to calculate the Fourier transform at all irreducible representations. Then, if bases are chosen so that (2.4) holds,*

$$T(G) \leq kT(H) + (k-1)\sum_{\rho} M(d_\rho). \tag{2.5}$$

The sum is over all irreducible representations, $M(d)$ is the number of operations required to multiply two $d \times d$ matrices, and $k = |G|/|H|$.

Remarks. (1) Taking $M(d) = d^2$ and using (2.1) gives

$$\frac{T(G)}{|G|} \leq \frac{T|H|}{|H|} + (k-1).$$

If the transforms in H are all computed directly, $T(H) = H^2$, and

$$\frac{T(G)}{|G|} \leq |H| + (k-1).$$

This is minimized (assuming the choice is possible) by choosing $|H| = \sqrt{|G|}$ which gives $2|G|^{3/2}$ as an upper bound for the running time.

As will be seen, iterative use of the identity can lead to $|G|\log|G|$ operations.

(2) In Abelian cases, and for S_n, there are representations that decompose as in (2.4) without changing bases. However, such special bases are not needed in theory or practice. A change of bases takes order $\sum_\rho M(d_\rho)$ operations. This can be done at the end, or as an intermediate step. Preliminary computations indicate that such considerations do not materially change the estimated running time.

(3) $T(G)$ counts the number of multiplications and additions. It does not count operations required to copy and keep track of the various pieces. As will be seen in the case we pursue, the bookkeeping operations can be neatly automated. At any rate, such operation counts are at best useful indicators for real running times.

For Abelian groups, the algorithm reduces to the well-known Cooley-Tukey algorithm. For example, consider the transform on Z/mnZ—the integers mod mn—and take Z/mZ as a subgroup—the integers mod m. This is contained in the larger group as $0, n, 2n, \ldots, (m-1)n$. The natural choice of coset representatives is $0, 1, 2, \ldots, n-1$. The Fourier transform at frequency j is

$$\hat{f}(j) = \sum_{k=0}^{mn-1} f(k)e^{2\pi ijk/mn} = \sum_{a=0}^{n-1}\sum_{b=0}^{m-1} f(a+bn)e^{2\pi ij(a+bn)/mn}$$
$$= \sum_{a=0}^{n-1} e^{2\pi ija/mn}\sum_{b=0}^{m-1} f_a(b)e^{2\pi ijb/m}.$$

Here, if $j = j_1 + j_2 m$, $0 \leq j_1 < m$, the inner sum is the transform of f_a at frequency $j_1 \pmod m$. These transforms are pasted together with the "twiddle factors" $e^{2\pi ija/mn}$ to determine $\hat{f}(j)$ for all $j \pmod{mn}$.

The Cooley-Tukey algorithm is just one of the ideas in the implementation of the fast Fourier transform. Some other ideas, and possible generalizations, are discussed in §8.

The algorithm may be applied recursively to any tower of subgroups

$$G = G_0 \supset G_1 \supset G_2 \supset \cdots \supset G_m \supset G_{m+1} = 1, \tag{2.6}$$

where the last step in the recursion is computed directly. It will be shown below that using as refined a tower as is available leads to the fastest algorithm.

For Abelian groups, the fundamental theorem yields a representation as a direct product of cyclic groups. There is a tower decreasing in steps of the various primes dividing $|G|$. For the group of binary n-tuples $(Z/2Z)^n$ the algorithm reduces to the standard fast Walsh transform.

For solvable groups such as p-groups and nilpotent groups, there is a tower of normal subgroups such that each quotient is a cyclic group of prime order. Thus there is a tower in steps of the various primes dividing $|G|$.

For simple groups, the situation is not as simple. Consider the alternating group A_n. For $n \geq 5$, the largest $H \subset A_n$ is A_{n-1}. Indeed, the action of A_n on A_n/H gives a permutation representation and so a homomorphism into S_d with $d = |A_n/H|$. For this to be nontrivial $d \geq n$ is required, so $|H| \leq (n-1)!$. It thus appears that the natural tower $A_n \supset A_{n-1} \supset A_{n-2} \supset \cdots$ is the most refined.

Similar towers can be derived for the other simple groups. A general group can then be decomposed, via its Jordan-Holder series, into subgroups with simple quotients. The towers for the simple quotients can be lifted to give a natural tower for G.

Of course, any practical example will be dominated by special features such as the availability of explicit representations of the various subgroups.

We conclude this section by investigating how the choice of a tower of subgroups affects the running time. This is of potential interest even in the Abelian case, where many towers are available. To simplify the discussion, assume that $d \times d$ matrices can be multiplied in d^2 operations. This is (presumably) the eventual "right answer" and, for Abelian groups $(d = 1)$, this does not impact on the conclusions.

The following result says that taking as refined a tower as possible is best.

Proposition 2. *Let G be a finite group. The tower of subgroups* (2.6) *minimizing the number of operations to compute the Fourier transform at all irreducible representations of G is such that $\sum_{i=0}^{m}[G_i: G_{i+1}]$ is minimal. Here $[G_i: G_{i+1}]$ denotes the index of G_{i+1} in G_i. This assumes $d \times d$ matrices can be multiplied in d^2 operations.*

Proof. It will be argued that $T(G)$, the number of operations based on the tower (2.6), is given by

$$T(G) = |G| \sum_{i=0}^{m} k_i, \qquad k_i = [G_i: G_{i+1}]. \tag{2.7}$$

This is argued by induction on m. From the basic identity (2.3)

$$T(G) = k_0 T(G_1) + k_0|G|.$$

Iterating, we obtain

$$\begin{aligned} T(G) &= k_0(k_1 T(G_1) + k_1|G_1|) + k_0|G| \\ &= \frac{|G|}{|G_2|} T(G_2) + |G|k_1 + |G|k_0. \end{aligned}$$

Continuing gives, for any l,

$$\begin{aligned} T(G) &= \frac{|G|}{|G_{l-1}|} T(G_{l-1}) + |G|(k_0 + \cdots + k_{l-2}) \\ &= \frac{|G|}{|G_{l-1}|} \left(\frac{|G_{l-1}|}{|G_l|} T(G_l) + k_{l-1}|G_{l-1}| \right) + |G|(k_0 + \cdots + k_{l-2}) \\ &= \frac{|G|}{|G_l|} T(G_l) + |G|(k_0 + \cdots + k_{l-1}). \end{aligned}$$

Taking $l = m$ and using $T(G_m) = |G_m|^2$ proves (2.7). □

Corollary. *If G has a tower such that each index is prime, then this tower is fastest for the algorithm.*

Remarks. The proposition assumes that the last term in the tower is computed directly, using $|G_m|^2$ operations. If G_m is a cyclic group of prime order p, the chirp-z transform can be used instead to get a final term of order $3p \log p$ as discussed in Diaconis (1980). This also applies to arbitrary solvable groups.

For practical purposes, at this writing, matrix multiplication takes order d^3 operations. Situations can be imagined where this effects the optimality. We have not pursued this carefully.

3. The symmetric group S_n

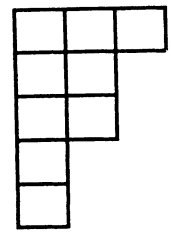
The representations of the symmetric group were determined by Frobenius and Young. We follow James (1978) and James and Kerber (1981) for notation.

There is a 1-1 correspondence between irreducible representations of S_n and partitions λ of n (denoted $\lambda \vdash n$). Here $\lambda = (\lambda_1, \ldots, \lambda_r)$ with $\lambda_1 \geq \cdots \geq \lambda_r > 0$ and $\lambda_1 + \cdots + \lambda_r = n$. Let $p(n)$ denote the number of partitions of n (so $p(5) = 7$). Then for large n

$$p(n) \sim \frac{1}{4n\sqrt{3}} e^{\pi\sqrt{2n/3}}.$$

To each partition λ of n we associate its *diagram*. Recall that this is a left-justified arrangement of empty boxes (nodes) with λ_i boxes in the ith row. For example, the diagram for $\lambda = (3, 2, 2, 1, 1)$ is:

Much of the representation theory of S_n involves the combinatorial properties of these diagrams and their generalizations.



The representation corresponding to λ is denoted ρ_λ and is of dimension d_λ. For example for $n = 5$, the seven partitions and the dimensions of their associated representations are

λ	(5)	(4, 1)	(3, 2)	(3, 1, 1)	(2, 2, 1)	(2, 1, 1, 1)	(1, 1, 1, 1, 1)
d_λ	1	4	5	6	5	4	1

The largest degree of an irreducible representation of S_n grows rapidly in n. (For $n = 10$, the maximum occurs at $\lambda = (4, 3, 2, 1)$ which has degree 768.) Since it will be needed later, we describe here the asymptotics of the maximum dimension of Logan and Shepp (1977) and Vershik and Kerov (1985).

Let P_n denote the set of partitions of n. Then the *Plancherel measure*, μ_n, is a measure defined on P_n by

$$\mu_n(\lambda) = d_\lambda / n!.$$

Use of (2.1) shows that in fact μ_n is a probability measure on P_n.

Vershik and Kerov prove:

Theorem. *There exist constants* $c_0, c_1 > 0$ *such that if* $F_n \subseteq P_n$ *is defined as*

$$F_n = \{\lambda \vdash n : \sqrt{n!}e^{-c_1\sqrt{n}} < d_\lambda < \sqrt{n!}e^{-c_0\sqrt{n}}\}, \tag{3.1}$$

then

(i) $\lim_{n\to\infty} \mu_n(F_n) = 1 - o(1)$ *and*

(ii) $\max_{\lambda\vdash n} d_\lambda < \sqrt{n!}e^{-c_0\sqrt{n}}$.

Presently it is known that one can take $c_0 = .1156$ and $c_1 = 1.238$.

The algorithms will use the nested chain of subgroups

$$S_r \supset S_{n-1} \supset \cdots \supset S_1 = \{\mathrm{id}\},$$

where $S_{n-j} = \{\pi \in S_n | \pi(n) = n, \ldots, \pi(n-j+1) = n-j+1\}$. The branching theorem (proved in James (1978), §9) determines how an irreducible representation ρ of S_n splits when restricted to S_{n-1}.

Theorem (Branching theorem). *Let* ρ *be an irreducible representation of* S_n *corresponding to the partition* λ *of* n. *Then* ρ *restricted to* S_{n-1} *splits into the direct sum of irreducible representations corresponding to the partitions* λ' *of*

$n-1$ which can be obtained by deleting a single node from the diagram of λ in all permissible ways.

Example. Let $\lambda = (3, 2, 2, 1, 1)$. Then ρ_λ restricted to S_8 splits into $\rho_{(2,2,2,1,1)}$, $\rho_{(3,2,1,1,1)}$, and $\rho_{(3,2,2,1)}$.

The splitting means that for $\pi \in S_{n-1}$, $\rho(\pi)$ can be written, perhaps after a change of basis, in block diagonal form as in (2.4). It turns out that in two well-known bases, the seminormal and orthogonal forms of Alfred Young, this block diagonal form is automatic. This result is proved in §4 which gives a description of these bases. The ideas of §2 can be iterated to prove the first result:

Proposition 3. *Let $T(n)$ denote the number of operations required to compute the Fourier transform at all irreducible representations of the symmetric group S_n. Let $M(d)$ be the number of operations required to compute the product of two $d \times d$ matrices. Then*

$$T(n) \leq n! \sum_{m=1}^{n} \frac{m-1}{m!} \sum_{\lambda \vdash m} M(d_\lambda).$$

Proof. From Proposition 1, $T(n)$ satisfies

$$T(n) \leq nT(n-1) + (n-1) \sum_{\lambda \vdash n} M(d_\lambda).$$

Now divide both sides by $n!$ and use induction. □

Corollary 1. *If $M(d) = cd^2$ for some constant c, then $T(n) \leq cn! \binom{n}{2}$.*

Proof. From (2.1), $T(n) \leq cn! \sum_{m=1}^{n} (m-1) = cn! \binom{n}{2}$. □

The bounds on the maximum degree will be combined with Proposition 3 to give a more accessible upper bound for other rates of matrix multiplication.

Theorem 1. *Let $T(n)$ be the number of operations required to compute the Fourier transform at all irreducible representations of the symmetric group S_n. If $d \times d$ matrices can be multiplied in d^a operations for $a > 2$, then*

$$\frac{T(n)}{n!} \leq (n!)^b e^{-bc\sqrt{n/2-1}} \left(n + n^{1-b} + O(n^{1-2b}) + e^{-bc} \left(\frac{n}{2} \right)^{2-bn/2} \right),$$

where $b = (a-2)/2$, $c = .2312$, and all error terms are uniform in a.

Proof. Proposition 3 yields

$$\frac{T(n)}{n!} \leq \sum_{m=1}^{n} m \sum_{\lambda \vdash m} \frac{d_\lambda^a}{m!}.$$

This may be rewritten as

$$\frac{T(n)}{n!} \le \sum_{m=1}^{n} m \cdot \sum_{\lambda \vdash m} \frac{d_\lambda^2}{m!} d_\lambda^{a-2}$$

$$\le \sum_{m=1}^{n} m \cdot \sum_{\lambda \vdash m} \frac{d_\lambda^2}{m!} (\sqrt{m!} e^{-c_0\sqrt{m}})^{a-2} \tag{3.2}$$

$$\le \sum_{m=1}^{n} m (m!)^b e^{-cb\sqrt{m}}, \tag{3.3}$$

where $c = 2c_0 = .2312$, (3.2) follows from Vershik and Kerov's estimate given in (3.1), and (3.3) follows from (2.1).

Explicit rewriting of (3.3) gives

$$\frac{T(n)}{n!} \le (n!)^b \left[n e^{-bc\sqrt{n}} + \frac{n-1}{n^b} e^{-bc\sqrt{n-1}} + \frac{n-2}{(n(n-1))^b} e^{-bc\sqrt{n-2}} + \cdots + \frac{e^{-bc}}{(n!)^b} \right].$$

Break this sum at the term with numerator $n/2 - 1$. The first sum is bounded above by $(n!)^b$ times

$$e^{-bc\sqrt{n/2-1}} \left(n + \frac{n-1}{n^b} + \frac{n-2}{(n(n-1))^b} + \cdots + \frac{n/2-1}{(n(n-1)\cdots(n/2))^b} \right)$$

$$\le e^{-bc\sqrt{n/2-1}} \left(n + n^{1-b} + \frac{(n-1)^{1-b}}{n^b} + \cdots + \frac{(n/2)^{1-b}}{(n\cdots(n/2+1))^b} \right)$$

$$\le e^{-bc\sqrt{n/2-1}} \left(n + n^{1-b} \frac{1 - (n/2)^{-b(n/2+1)}}{1 - (n/2)^{-b}} \right).$$

Note that assuming $2 \le a \le 3$ implies $0 \le 1 - b \le 1$.

The second sum is bounded above by $(n!)^b$ times

$$e^{-bc} \frac{n}{2} \frac{n/2 \cdots}{(n/2)^{bn/2}} = e^{-bc} \left(\frac{n}{2} \right)^{2-bn/2}.$$

Collecting terms gives the stated result. □

Remark. Thus, the upper bound for $T(n)$ can be seen to be

$$T(n) \le A(n!)^{a/2} n e^{-(a-2)c'\sqrt{n/2}},$$

where A is some computable constant and $c' = .1156$. Consequently, as $a \to 2^+$, $(a-2) \to 0$, and Theorem 1 gives

$$T(n) \ll n! n^2$$

in agreement with Corollary 1. This is, roughly, $|G| \log |G|$.

4. Young's seminormal and orthogonal forms

As presented in §3, the algorithm for computing the Fourier transform depends on having, for any irreducible representation $\rho: S_n \to \mathrm{GL}(V)$, a basis for V such that when ρ is restricted from S_n to S_{n-1} the matrices $\rho(\pi)$ $(\pi \in S_{n-1})$ are block diagonal as determined by the branching theorem. For example, let ρ be the irreducible representation of S_5 corresponding to $\lambda = (3, 2)$. For $\pi \in S_5$, with $\pi(5) = 5$, we require

$$\rho(\pi) = \begin{pmatrix} B^{3,1} & 0 \\ 0 & B^{2,2} \end{pmatrix},$$

where B^{λ} is the appropriate matrix representation of π regarded as an element of S_4. Of course, if π fixes 4 and 5, then the blocks $B^{3,1}$ and $B^{2,2}$ split further into B^3, $B^{2,1}$, and $B^{2,1}$ respectively.

Two of the representations constructed by Alfred Young have this property. In what follows, we present the explicit construction of these representations. The discussion follows Kerber (1971) and James-Kerber (1981, Chapters 3 and 7).

Fix a partition $\lambda \vdash n$. Then a basis for ρ_{λ} is indexed by the "standard tableaux" of shape λ. Recall that a *tableau* of shape λ is a diagram of shape λ with numbers $\{1, \ldots, n\}$ placed in the boxes. The tableau is *standard* if the numbers increase from left to right and top to bottom. For example, the standard tableaux of shape (3,2) are:

$$\begin{array}{ccccc} 1\,3\,5 & 1\,2\,5 & 1\,3\,4 & 1\,2\,4 & 1\,2\,3 \\ 2\,4 & 3\,4 & 2\,5 & 3\,5 & 4\,5 \end{array}$$

The *last letter order* on standard tableaux is a linear order defined as follows: Consider two standard tableaux T_1 and T_2 of the same shape. If n is in a row higher up in T_1 than in T_2 declare $T_1 < T_2$ (thus $\begin{smallmatrix} 135 \\ 24 \end{smallmatrix} < \begin{smallmatrix} 134 \\ 25 \end{smallmatrix}$). If n is in the same row in both (necessarily the last entry in this row), delete n from the tableaux and consider $n - 1$, etc. The five tableaux above are shown in increasing order.

The matrices to be constructed have rows and columns indexed by standard Young tableaux in last letter order. The matrix entries are defined in terms of the *axial distance*. For this, consider a tableau T (standard or not) of shape λ. For $a \in \{1, 2, \ldots, n\}$, let $r_T(a)$ be its row position and $c_T(a)$ its column position. Given two entries a, b $\{1 \le a, b \le n\}$, define the *axial distance* from a to b as

$$d^T(a, b) = (c_T(a) - c_T(b)) + (r_T(b) - r_T(a)).$$

This is the number of moves it takes to go from a to b in T counting moves to the left or down as positive, and moves to the right or up as negative. For example, if

$$T = \begin{array}{ccc} 1 & 2 & 3 \\ 5 & 4 & \end{array}, \quad \text{then } d^T(3, 5) = 3, \quad d^T(4, 1) = 0.$$

Note that this is a signed distance and $d^T(a, b) = -d^T(b, a)$.

Let $d^i(a, b)$ denote the axial distance for the ith standard Young tableau in the last letter order. Dependence on the shape λ is suppressed in this notation.

Fix a shape λ. Given a transposition $(t, t+1) \in S_n$, define a matrix

$$\sigma(t, t+1) = ((\sigma_{ij}(t, t+1)))$$

by the following rule, with rows and columns indexed by the standard Young tableaux T_i of shape λ in the last letter order:

(a) $$\sigma_{ii} = \begin{cases} +1 & \text{if } t, t+1 \text{ are in the same row of } T_i, \\ -1 & \text{if } t, t+1 \text{ are in the same column of } T_i. \end{cases}$$

(b) If $(t, t+1)T_j = T_i$ for $i < j$ then let:

$$\begin{matrix} \sigma_{ii} & \sigma_{ij} \\ \sigma_{ji} & \sigma_{jj} \end{matrix} = \begin{matrix} -d^i(t, t+1)^{-1} & 1 - d^i(t, t+1)^2 \\ 1 & d^i(t, t+1)^{-1} \end{matrix}$$

(Here $(t, t+1)T_j$ is the tableau obtained from T_j by interchanging t and $t+1$.)

(c) $\sigma_{ij} = 0$ otherwise.

Similarly, define a matrix $\omega(t, t+1)$ by changing (b) to:

$$\begin{matrix} \omega_{ii} & \omega_{ij} \\ \omega_{ji} & \omega_{jj} \end{matrix} = \begin{matrix} -d^i(t, t+1)^{-1} & \sqrt{1 - d^i(t, t+1)^{-2}} \\ \sqrt{1 - d^i(t, t+1)^{-2}} & d^i(t, t+1)^{-1} \end{matrix}$$

Chapter 3 of James and Kerber (1981) presents a proof that these specify irreducible representations. Since the pairwise adjacent transpositions generate S_n, it is enough to specify the representations of only these elements. Let us record this as

Theorem. *The matrices σ and ω defined above generate Young's seminormal and orthogonal representations respectively. Write $\sigma(\pi)$ and $\omega(\pi)$ for these matrices at the permutation π.*

Proposition 4. *If $\pi \in S_n$ fixes n, then both $\sigma(\pi)$ and $\omega(\pi)$ decompose into blocks as described in* (2.4).

Proof. The argument exploits the compatability of the last letter order and the branching theorem. To say this clearly, consider a standard tableau T of shape $\lambda \vdash n$. Deleting the box containing n results in a standard tableau $\tilde{T}$ of shape $\tilde{\lambda} \vdash n-1$. Suppose T_1 and T_2 are distinct standard tableaux of shape λ with n in the same position. Then $\tilde{T}_1$ and $\tilde{T}_2$ are distinct and of the same shape and $T_1 < T_2$ if and only if $\tilde{T}_1 < \tilde{T}_2$. Consequently, if $T_1 < T_2 < \cdots < T_{d_\lambda}$ denotes all the standard tableaux of shape λ in the last letter order, all tableaux with n in a fixed position appear together as an interval or segment in this list. Deleting n from each tableau in such a segment will give all standard tableaux of shape $\tilde{\lambda} \vdash n-1$, in order. The number of such segments is precisely the number of ways of removing a box from the diagram of λ that leave a diagram for a partition of $n-1$.

Returning now to the proof, it is enough to prove the assertion for $\pi = (t, t+1)$, a pairwise adjacent transposition with $1 \le t \le n-2$. Fix $\lambda \vdash n$. Clearly, for any i, the axial distance $d^i(t, t+1)$ equals the axial distance between $t, t+1$ in the ith tableau with n deleted. Moreover, t and $t+1$ are in the same row or column of the ith tableau if and only if they are in the same row or column in the deleted tableau. Finally, if $(t, t+1)$ applied to the ith tableau is standard, say equal to the jth tableau, the ith and jth tableaux must have n in the same position, and $(t, t+1)$ maps the deleted tableaux to each other.

This shows that if M is either $\sigma(t, t+1)$ or $\omega(t, t+1)$, then M appears as

$$\begin{pmatrix} M_1(t, t+1) & & 0 \\ & \ddots & \\ 0 & & M_b(t, t+1) \end{pmatrix}$$

with $M_i(t, t+1)$ the representing matrix assigned by the theorem to the partition $\tilde{\lambda}$ given by deleting n from the standard tableaux in the ith segment of $T_1 < \cdots < T_{d_\lambda}$. Thus b will be the number of segments. □

Remark. There are other important bases: Young's natural form (which does not split up as above) and the Kazdahn-Lusztig form. Applications may demand yet another form. If the form is known, one can calculate for each λ a change of basis matrix H_λ. This gives $\hat{f}(\rho)$ in the required basis by conjugation which takes $2\sum M(d_\lambda)$ operations.

Thus far, the discussion has used only pairwise adjacent transpositions. For some of the algorithms which follow, representing matrices at all transpositions are needed. For other algorithms $\rho(\pi)$ is needed for all permutations π. We now discuss the issue of efficient generation of such larger sets of matrices.

Consider first the problem of building up all transpositions $\rho(ij)$ for all irreducible representations ρ. We proceed by induction, assuming we have previously computed $\{\rho(i, j)\colon 1 \le i \le j < n-1\}$. Then $\rho(j, n) = \rho(j, n-1)\rho(n-1, n)\rho(j, n-1)$. By induction we have

Proposition 5. *The number of operations needed to build up $\rho(\tau)$ for all representations ρ and all transpositions τ in S_m, $1 \le m \le n$, is*

$$\sum_{m=2}^{n} 2(m-2) \sum_{\lambda \vdash m} M(d_\lambda). \tag{4.1}$$

Remarks. (1) If $M(d) = d^2$ is assumed, (4.1) is asymptotically $2nn!$. This algorithm takes advantage of branching. If ρ is a fixed representation of S_n, and pairwise adjacent transpositions are available, $\rho(\tau)$ can be built up for all transpositions in S_n by conjugating. This takes $2\binom{n-1}{2}M(d_\rho)$ operations. Summing in ρ, this direct algorithm takes more operations than the algorithm using branching and it only computes for a fixed n, rather than for all m, $1 \le m \le n$.

(2) For Young's seminormal and orthogonal representations the representing matrices for pairwise adjacent transpositions are sparse, having at most $2d_\rho$ nonzero entries. It follows that matrices for arbitrary transpositions have at most $4^{n-2}d_\rho$ nonzero entries. Results of Vershik and Kerov, cited in §3, imply that "most" representations ρ have dimension, roughly, of order $\sqrt{n!}$. This suggests that sparse matrix techniques may be useful in getting additional speedups.

Finally, let us turn to the problem of generating $\rho(\pi)$ for all permutations π. Here we use the decomposition

$$S_n = S_{n-1} \cup (n-1, n)S_{n-1} \cup (n-2, n)S_{n-1} \cup \cdots \cup (1, n)S_{n-1}. \tag{4.2}$$

Proposition 6. *Let $B(n)$ be the number of operations needed to build up $\rho(\pi)$ for all irreducible representations ρ and permutations π in S_m for $1 \le m \le n$, given ρ at all pairwise adjacent transpositions. Then*

$$B(n) = \sum_{m=2}^{n} [(m-1)(m-1)! + (m-2)] \cdot \sum_{\lambda \vdash m} M(d_\lambda).$$

Proof. Using the decomposition (4.2), note that computing $(j, n)S_{n-1}$ includes the computation of $(j, n)(j-1, j)$. Thus, one more matrix multiplication gives $(j-1, j)(j, n)(j-1, j) = (j-1, n)$. This gives the recurrence

$$\begin{aligned} B(n) &= B(n-1) + [(n-1) \cdot (n-1)!] \sum_{\lambda \vdash m} M(d_\lambda) \\ &\quad + [n-2] \cdot \sum_{\lambda \vdash m} M(d_\lambda) \\ &= B(n-1) + [(n-1) \cdot (n-1)! + (n-2)] \sum_{\lambda \vdash m} M(d_\lambda). \quad \square \end{aligned}$$

Remarks. (1) If $M(d) = d^2$, this gives $B(n)$ asymptotic to a constant times $(n!)^2$ as a start up cost for algorithms requiring all $\rho(\pi)$.

(2) Garsia and McLarnan (1986) specify a "closed form" for $\rho(\pi)$ for all π and ρ when ρ is Young's natural representation. Unfortunately, this does not decompose in a neat way under restriction.

5. Partial transforms

The basic recurrence at (2.3) is

$$\hat{f}(\rho) = \sum_{i=1}^{k} \rho(s_i) \sum_{h \in H} f_i(h)\rho(h). \tag{5.1}$$

This can be used and iterated to allow speedups in computation for a limited number of irreducible representations. This will be illustrated first in the Abelian case and then for the symmetric group.

Example 1. *A single representation.* Consider the additive group Z/NZ with $N = 2^a$. The jth transform is

$$\hat{f}(j) = \sum_{k=0}^{N-1} f(k)e^{2\pi ijk/N}.$$

Direct computation for fixed j involves N operations. Take $H = Z/MZ$, with $M = N/2$. Coset representatives may be taken as $0, 1$. The recurrence (5.1) becomes

$$\hat{f}(j) = \sum_{k=0}^{M-1} f(2k)e^{2\pi ijk/M} + e^{2\pi ij/N} \sum_{k=0}^{M-1} f(2k+1)e^{2\pi ijk/M}$$

or, with obvious notation,

$$\hat{f}(j) = \hat{f}_0(j_0) + e^{2\pi ij/N}\hat{f}_1(j_0) \quad \text{with } j \equiv j_0 \pmod{M}. \tag{5.2}$$

This gives the following recurrence for $T_1(a)$, the number of operations needed to compute $\hat{f}(j)$:

$$T_1(a) \leq 2T_1(a-1) + 1.$$

Iterating gives

$$T_1(a) \leq 2^a + a.$$

This is a (slight) increase in running time for computing a single value. For large a's, this may fairly be thought of as equivalent to direct computation.

Example 2. *Subgroups.* A clear way to save occurs if it is desired to compute $f(j)$ with j lying in a coset of a subgroup of $Z/2^aZ$. Let us fix a subgroup $Z/2^bZ$ with $b < a$ and let j run through the set

$$\{m, m + 2^{a-b}, m + 2 \cdot 2^{a-b}, m + 3 \cdot 2^{a-b}, \ldots, m + (2^b - 1)2^{a-b}\},$$

where $0 \leq m < 2^{a-b}$ is fixed. Although there are 2^b different j's to be computed, by using (5.2) we note that only 2^{b-1} values of $j \bmod 2^{a-1}$ are needed. If $T_b(a)$ denotes the number of operations required to compute these 2^b values, (5.2) implies

$$T_b(a) = 2T_{b-1}(a-1) + 2^b.$$

Iterating b times and using direct computation for the remaining sum $(T_0(a))$ gives

$$T_b(a) = 2^a + b2^b. \tag{5.3}$$

This clearly offers a savings over either direct computation of the 2^b values (2^{a+b} operations) or complete computation using the standard FFT ($a2^a$ operations), for b large, but small compared to a.

Remarks. The algorithm could be usefully applied to a signal with lots of data, where one does not feel the need for looking at all frequencies. A route to a similar saving would be to sample the observed series at equally spaced points and compute the full discrete Fourier transform from this data. These approaches are dual in a sense.

Example 3. *General Abelian groups.* The algorithm of Example 2 makes crucial use of the commutativity of Z/NZ. The dual is considered as an Abelian group. The restriction of an irreducible representation to a subgroup is irreducible. The ideas extend in a straightforward way.

Let A be a finite Abelian group, H a subgroup, and $H^{\perp}$ the set of characters of A that restrict to the trivial character of H. This is a subgroup of the dual $\hat{A}$, isomorphic to the dual group of A/H. A coset of $H^{\perp}$ works well as a subset of representations.

If $s_1, s_2, \ldots, s_k$ are coset representatives for H in A, and χ is any character of A, the Fourier transform can be written as

$$\hat{f}(\chi) = \sum_{i=1}^{k} \chi(s_i) \hat{f}_i(\chi|_H). \tag{5.4}$$

The notation $\chi|_H$ denotes restriction. Use (5.4) for all χ in a coset of $H^{\perp}$, each such χ restricts to the same character on H. This gives

Proposition 7. *Let $T_{H^{\perp}}(A)$ be the number of iterations required to compute the Fourier transform at all characters in a coset of $H^{\perp}$ in $\hat{A}$. Then*

$$T_{H^{\perp}}(G) \leq |A| + k(k-1), \quad \text{with } k = |A/H|.$$

Remarks. (1) If k is of order $\sqrt{A}$, this allows computation in order $2|A|$ steps while direct computation takes $|A|^{3/2}$ and full use of the fast Fourier transform $|A| \log |A|$.

Of course, usually a tower of subgroups between H and A can be found to speed things up further.

(2) The identity (5.4) gives a simple argument for the Poisson summation formula for finite Abelian groups as in Good (1962). Suppose $\chi|_H$ is trivial. Summing (5.4) over $H^{\perp}$, we obtain

$$\sum_{\chi \in H^{\perp}} \hat{f}(\chi) = \sum_{i=1}^{k} f_i(\mathrm{id}) \sum_{\chi \in H^{\perp}} \chi(s_i).$$

The inner sum is $|H^{\perp}|$ for $s_i = \mathrm{id}$ and zero if $s_i \notin H$ (the sum of the characters at any nontrivial element in zero). Finally, if η is any character of A, we have the Poisson summation formula

$$\frac{|H|}{|A|} \cdot \sum_{\chi \in H^{\perp}} \hat{f}(\chi\eta) = \sum_{h \in H} f(h)\eta(h). \tag{5.5}$$

Example 4. 2^b *in a row.* With notation as in Examples 1 and 2, consider computing $\hat{f}(0), \hat{f}(1), \ldots, \hat{f}(2^b - 1)$. The idea is to use (5.2) iteratively. If $S_b(a)$ is the number of operations required to compute these 2^b values, the recurrence is

$$S_b(a) \leq 2S_b(a-1) + 2^b.$$

Use this until $S_b(b)$ is reached when the usual FFT, taking $b2^b$ operations, is used. This results in

$$S_b(a) = 2^a(b+1) - 2^b. \tag{5.6}$$

Note that this only offers a savings over the full FFT if b is a fraction of a, e.g. $b = a/2$. We have chosen 2^b values starting at 0. It is straightforward to modify this to get 2^b values centered anywhere. This allows a close look at the shape of the transform in a neighborhood of a specific frequency.

Example 5. *The symmetric group.* The usual partial order on partitions meshes with the branching theorem to allow partial computation in a useful way. Here, recursive algorithms offer a speedup, even for computation of the transform at a single representation.

For example, consider computing $\hat{f}(\rho)$ for ρ corresponding to the usual n-dimensional permutation representation. Direct computation involves $n!n$ operations—for each permutation, n entries in an $n \times n$ matrix have to be changed.

The representation ρ is the direct sum of the trivial and $n-1$ dimensional representation of S_n. This last, restricted to S_{n-1}, splits into the trivial and $n-2$ dimensional representations of S_{n-1}.

If $T(n)$ denotes the number of operations required to compute $\hat{f}(\rho)$ using recurrence, we have

$$T(n) \leq nT(n-1) + nM(n).$$

Dividing through by $n!$ and using induction gives

$$T(n) \leq n! \sum_{d=1}^{n} M(d)/(d-1)!. \tag{5.7}$$

Observe now that the sum in (5.7) is bounded for any n. Thus recursion speeds up even computation of a single transform.

To extend, sets of partitions that remain closed under the branching rule must be found. Two natural candidates may be called hooks, and partitions with l or fewer parts. To explain,

(5.8) Let H_n be the set of all hook partitions, of the form $(n-q, 1^q)$ for $0 \leq q \leq n-1$. Note that if $\lambda \in H_n$ and $\tilde{\lambda}$ is obtained from λ by deleting a box, then $\tilde{\lambda} \in H_{n-1}$.

(5.9) Let $L_n(l)$ be the set of all partitions of n of the form $(\lambda_1, \lambda_2, \ldots, \lambda_r)$ with $r \leq l$.

The main result says that all transforms at all representations in H_n, or in $L_n(l)$, with l fixed, can be computed in order $n!$ operations.

Theorem 2. *Let $T_H(n)$ be the number of operations required to compute all transforms at hook representations as defined at (5.8). Let $T_l(n)$ be the number of operations required to compute all transforms at all representations with l or fewer parts. Then, if multiplication of $d \times d$ matrices takes fewer than d^a operations $2 \le a \le 3$,*

(1) $T_H(n) \le n! e^8$;

(2) $T_l(n) \le n! c(l)$, *where for fixed l,*

$$c(l) = \sum_{m=l+1}^{\infty} \frac{1}{(m-a)!} \sum_{\substack{\lambda \vdash m \\ h(\lambda) \le l}} d_\lambda^a + (l!) < \infty.$$

Proof. For (1), there are n hook representations. The dimension of $(n-q, 1^q)$ is $\binom{n-1}{q}$.

The recursion (2.3) becomes

$$T_H(n) \le n T_H(n-1) + n \sum_{q=0}^{n-1} M\left[\binom{n-1}{q}\right].$$

Thus

$$\frac{T_H(n)}{n!} \le \sum_{m=1}^{n} \frac{1}{(m-1)!} \sum_{q=0}^{m-1} M\left[\binom{m-1}{q}\right].$$

Assuming $M(d) \le d^a$ for $a < 3$,

$$\sum \binom{n-1}{q}^a \le \left(\sum \binom{n-1}{q}\right)^3 = 2^{3(n-1)}.$$

Then (1) follows from

$$\sum \frac{8^n}{n!} = e^8.$$

For (2), let the height $h(\lambda)$ of the partition $\lambda = (\lambda_1, \lambda_2, \dots, \lambda_r)$ be defined as $h(\lambda) = r$. Arguing as above,

$$\frac{T_l(n)}{n!} \le \sum_{m=l}^{n} \frac{1}{(m-1)!} \sum_{\substack{\lambda \vdash m \\ h(\lambda) \le l}} M(d_\lambda) + \frac{T_l(l)}{l!}.$$

Here $T_l(l) \le (l!)^2$ (of course if l is large, this can be improved as in §3). If $M(d) \le d^a$, one must bound

$$S_l^a(n) = \sum_{\substack{\lambda \vdash m \\ h(\lambda) \le l}} d_\lambda^a.$$

Regev (1981) and Askey and Regev (1984) have worked out the appropriate asymptotics. It follows from Regev (1981), Theorems 2–10, that for a, l fixed

$$S_l^a(n) \sim \frac{l^{l^2/2}}{(2\pi)} \frac{l-1}{2} \left(\frac{1}{n}\right)^{(l-1)(l+2)/4} l^{n^a} n^{(l-1)/2} I(l, a),$$

with

$$I(l, a) = \left(\frac{1}{l}\right)^{(l-1)(al+2)/4} \frac{1}{l!}\sqrt{\frac{a}{2\pi l}}(2\pi)^{l/2}$$
$$\cdot a^{-l/2-al(l-1)/4}\Gamma\left(1+\frac{a}{2}\right)^{-l}\sum_{j=1}^{l}\Gamma\left(1+\frac{aj}{2}\right).$$

In particular, for l and a fixed, this grows like l^{na}. It follows that

$$c(l) = \sum_{m=1}^{\infty} \frac{1}{(m-1)!} S_l^a(m) < \infty. \quad \square$$

Remarks. Similar results can be derived for the set of all partitions larger than $(n-q, 1^q)$ in the majorization order (one partition is larger than another if you can get from small to large by moving up boxes one at a time). This enters naturally in processing data involving rankings of (say) 40 movies. One would like to analyze features involving the top most a and bottom most b rankings where $a+b=q$ (e.g. $a=b=5$) (see Chapter 8 of Diaconis (1988)).

6. Five algorithms for S_n

Implementing the ideas above involves tradeoffs between computation time, storage, clarity, and ease of programming. Five algorithms for S_n are presented in this section:

(1) Direct computation from the definition, assuming all matrices are available in memory.
(2) Recursive computation for each representation, assuming matrices for transpositions are available in memory.
(3) Complete use of the branching procedure, assuming all transpositions for each S_j, $2 \le j \le n$, are available in memory.
(4) Complete use of the branching procedure as in algorithm 3, but storing only partial transformations restricted to S_{n-1}.
(5) Complete use of branching using dynamic programming.

Algorithm 4 or 5 runs recursively down to S_5 with algorithm 2 used for S_5, seems like the best choice for medium size computing environments such as a Vax 370. A copy of our current implementation is available on request. The algorithms are described below in an Algol-like language with comments on their features and weaknesses.

1. Direct computation. Assume $\rho(\pi)$, for all irreducible representations ρ and all permutations π in S_n, have been generated (as in §4) and are available in memory. A function $f(\pi)$ is given. The following algorithm computes $\sum_\pi f(\pi)\rho(\pi)$ directly. (Below, we assume that we have subroutines "retrieve(matrix)" which retrieves a given matrix from memory and "store(matrix)" to place a matrix in memory. Also, "result-matrix" and $\rho(\pi)$ for any $\pi \in S_n$ are matrix variables.)

```
BEGIN
   FOR all irreducible representations ρ DO
      BEGIN
         result-matrix := 0; /* initialize final result */
         FOR π ∈ S_n DO
            BEGIN
               retrieve (ρ(π));
               result-matrix := result-matrix +f(π) * ρ(π);
            END
            store (result-matrix);
      END
END
```

This is the simplest algorithm to program and to understand. It is most expensive in start-up cost, memory, and operations.

2. Recursive computation. One implementation uses a variant of direct computation which assumes $\rho(\pi)$ is available in memory for all irreducible representations ρ and all transpositions π. The following algorithm computes $\sum f(\pi)\rho(\pi)$ recursively, using the chain of subgroups $S_n \supset S_{n-1} \supset S_{n-2} \supset \cdots$. It is presented as a "main body" that calls the subroutine "compute-transform" to perform the recursive computation. "Compute-transform" takes as parameters a degree (denoting the subgroup in the chain) and a function on S_d, $g(\pi)$, where d is the passed degree. "Partial-transform" is a matrix variable, "identity-matrix" is the obvious constant, and other variables and subroutines are as in algorithm 1. Recall that if f is a function defined on S_n, then $f_{i_n \cdots i_{k+1}}(\pi)$ for π in S_k is defined to be $f((i_n, n)\cdots(i_{k+1}, k+1)\pi)$.

```
BEGIN /* main body */
   FOR all irreducible representations ρ DO
      BEGIN
         result-matrix := 0;
         store (compute-transform (n, f));
      END
END /* main body */

compute-transform (degree, g)
BEGIN /* compute-transform */
   IF degree = 2 THEN result-matrix := g(id)* identity-matrix +g(1, 2) * ρ(1, 2);
   ELSE
      FOR i = 1 to degree DO
         BEGIN /* compute ĝ_i(ρ) */
            partial-transform := compute-transform (g_i, degree-1)
            result-matrix := result-matrix +ρ(i, degree) * partial-transform;
         END
END /* compute-transform */
```

This algorithm is used as a part of algorithms 4 and 5 to compute for $n \leq 5$. To understand it further, the number of operations can be approximated.

Proposition 8. *Let $T(n)$ be the number of operations used by algorithm 2 to compute the Fourier transforms at all irreducible representations of S_n. Then*

(1) $T(n) \ll n! \sum_\rho M(d_\rho)$.

(2) *If* $M(d) = d^a$, $T(n) \ll (n!)^{1+a/2} e^{-c(a-2)\sqrt{n}}$ *with* $c = .1156$.

Proof. Let $T_\rho(n)$ denote the number of operations used by algorithm 2 to compute the Fourier transform at any representation ρ (irreducible or not) for a function $f(\pi)$ defined on S_n. $T_\rho(n)$ satisfies the recurrence,

$$T_\rho(n) = nT_\rho(n-1) + (n-1)M(d_\rho). \tag{6.1}$$

Note that $T_\rho(n-1)$ is the number of operations needed to compute $\hat{f}_i(\rho)$.

By the branching theorem described in §3, ρ restricted to S_2 is diagonal in Young's seminormal or orthogonal forms. Hence we see that

$$T_\rho(3) = 3T_\rho(2) + 2d_\rho^2.$$

Thus, by induction, (6.1) extends to

$$\begin{aligned} T_\rho(n) &\le [(n-1) + n(n-2) + \cdots + n(n-2)(n-3)\cdots 5\cdot 3] \\ &\quad \cdot M(d_\rho) + n!\left(d_\rho + \frac{1}{3}d_\rho^2\right) \\ &\le (n!)\left[\left[\frac{1}{(n-1)!} + \cdots + \frac{1}{3!}\right]\cdot M(d_\rho) + d_\rho + \frac{1}{3}d_\rho^2\right]. \end{aligned}$$

Assuming $M(d) \ge d^\alpha$ then summing $T_\rho(n)$ over all irreducible representations ρ of S_n yields (1).

Finally, assuming $M(d) = d^a$ for $a \ge 2$, the proof of Theorem 1 estimates the sum $\sum_\rho M(d_\rho)$ by

$$\begin{aligned} \sum_\rho d_\rho^a &= (n!)\sum_\rho \frac{d_\rho^2}{n!} d_\rho^{a-2} \\ &\le (n!)\cdot(\sqrt{n!}e^{-c\sqrt{n}})^{a-2} \\ &= (n!)^{a/2} e^{-c(a-2)\sqrt{n}} \end{aligned}$$

with $c = .1156$. This completes the proof of (2). □

For any constant b, $\sqrt{n!}e^{-b\sqrt{n}}$ is unbounded as n goes to infinity. Thus, Proposition 8 implies that for $a > 2$, the running time of algorithm 2 is asymptotically greater than algorithm 1 which is $(n!)^2$. For $a = 2$, the proposition gives a running time of $(n!)^2$ for algorithm 2. Algorithm 2 requires only that $\binom{n}{2}n!$ matrix elements (the irreducible representations of the transpositions) be stored as opposed to $(n!)^2$ matrix elements for algorithm 1. Speedups for algorithm 2 can be obtained by taking more advantage of the block structure of the matrices $\rho(j, k)$ for $1 \le j, k \le n-1$.

3. Complete branching. An implementation of the ideas of §3 requires several subroutines.

1. Branch (ρ, number-of-branches, branches)—this takes a representation ρ of S_k and returns to ρ restricted for S_{k-1}, the number of branches,

in the variable number-of-branches, and the branches in an array of representations called branches.

2. Build-matrix (matrix 1, matrix 2)—inserts matrix 2 as a block in matrix 1 in some specified position.
3. Retrieve(matrix)—retrieves indicated matrix from memory.
4. Store(matrix)—stores matrix in previously specified position in memory.
5. Copy-matrix (matrix 2, matrix 1)—copies matrix 1 into matrix 2.

Below, f-transform, restricted-transform, temp-result and $\hat{f}(\rho)$ should all be read as matrix variables where it is appropriate.

The algorithm requires that all the restricted, partial transforms be stored. This is relaxed in algorithm 4 below. We stop the recurrence at some integer M and compute the transforms for S_M directly using algorithm 2. With this notation we describe a subroutine SFFT which takes as input three parameters:

n: The degree of the symmetric group.

ρ: The representation over which we compute.

f: The function whose transform we compute.

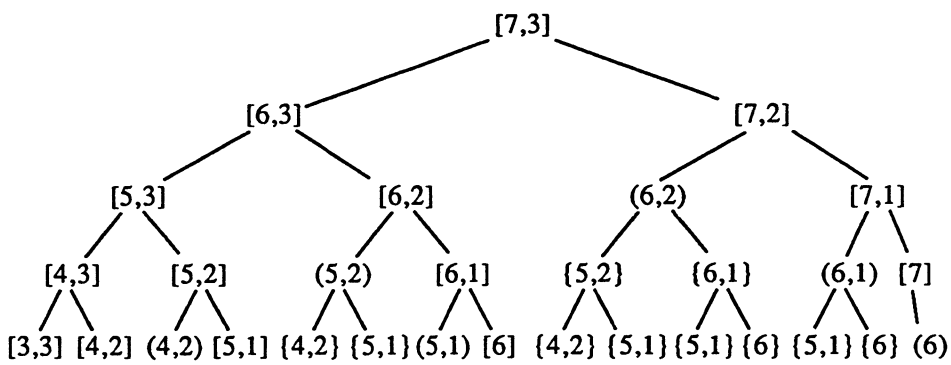
```
SFFT (n, ρ, f)
BEGIN
  IF f̂(ρ) has been computed previously THEN
    BEGIN
      retrieve (f̂(ρ));
      copy-matrix (f-transform, f̂(ρ));
    END;
  ELSE IF n = M THEN
    BEGIN
      compute f̂(ρ) á la algorithm 2;
      copy-matrix (f-transform, f̂(ρ));
    END
  ELSE
    BEGIN
      branch (ρ, number-of-branches, branches);
      FOR k = 1 to n DO
        BEGIN
          FOR j = 1 to number-of-branches DO
            BEGIN
              copy-matrix (restricted-transform, SFFT (n − 1, branches [j], f_k));
              build-matrix (temp-result, restricted-transform);
            END
          copy-matrix (f-transform, f-transform +ρ(k, n)* temp-result);
        END
      store (f-transform) /* store this as f̂(ρ) */
    END
  return (f-transform);
END
```

There is a great deal of nontrivial bookkeeping going on in algorithm 3. One must keep track of where the restricted representation blocks go, and where in memory the restricted transforms occur. In storage, there must be room to keep all of the restricted transforms: $\{\hat{f}_{i_n \cdots i_m}(\rho) | \rho$ is an irreducible representation

of S_{m-1}, $1 \le i_j \le j$, $M+1 \le m \le n\}$ as well as the transpositions. Thus, storage for $\binom{n}{2}n! + (n-M)n!$ elements is required.

4. Complete branching, partial storage. Algorithm 3 requires storing every restricted transform throughout the computation. Storage restrictions may make this impossible. As an alternative, we present algorithm 4 in which only the restrictions to S_{n-1} are kept for the entire computation. Both algorithms require the storage of all irreducible representations of all transpositions. By only storing the restrictions to S_{n-1} the computation is slowed a bit, but storage requirements are reduced to manageable proportions. For example, on S_{10}, algorithm 4 requires about 12×10^6 floating point numbers be stored. Algorithm 3 requires six times this amount. Even with the storage restriction, substantial savings may be achieved.

Before sketching the general algorithm, consider the specific example of the computation of $\hat{f}(\rho)$, where ρ is the irreducible representation of S_{10} corresponding to the partition $(7, 3)$. The following "tree" represents the way in which ρ splits when restricted, the kth level showing the splitting for ρ restricted to S_{n-k}. The restrictions though S_6 are shown.



Recursing one level, $\hat{f}(\rho)$ may be rewritten explicitly as

$$\hat{f}(\rho) = \sum_{\pi \in S_{10}} f(\pi)\rho(\pi) = \sum_{i=1}^{10} \rho(i, 10) \begin{pmatrix} \hat{f}_i(\eta^3) & 0 \\ 0 & \hat{f}_i(\eta^2) \end{pmatrix},$$

where η^j denotes the irreducible representation of S_9 corresponding to the partition $(9-j, j)$. If either of the restricted transforms to S_9 had been

computed previously then they could simply be retrieved from memory and used in this computation. If not then they must now be computed.

As an illustration, assume that neither the restricted tranforms $\{\hat{f}_i(\eta^3)\}$ nor $\{\hat{f}_i(\eta^2)\}$ have already been computed. One more level of restriction allows $\hat{f}(\rho)$ to be rewritten as

$$\hat{f}(\rho) = \sum_{i_{10}=1}^{10} \sum_{i_9=1}^{9} \rho(i_{10}, 10)\rho(i_9, 9) \cdot \begin{pmatrix} \hat{f}_{i_{10}, i_9}(\eta^3) & 0 & 0 & 0 \\ 0 & \hat{f}_{i_{10}, i_9}(\eta^2) & 0 & 0 \\ 0 & 0 & \hat{f}_{i_{10}, i_9}(\eta^2) & 0 \\ 0 & 0 & 0 & \hat{f}_{i_{10}, i_9}(\eta^1) \end{pmatrix}.$$

For any fixed values of i_{10} and i_9 the restricted transform $\hat{f}_{i_{10}, i_9}(\eta^2)$ need only be computed once and then can be copied to the other block in which it occurs. To obtain greater savings further recursion is performed. As we proceed down the levels of the tree the representations have more and more restrictions in common. In general, at each level only the first occurrence of each restricted transform should be computed. Consequently, computation of the restrictions must be scheduled in such a way that all restrictions to a given subgroup are computed on the same pass. To make this point another way, consider the explicit rewriting of a given Fourier transform on S_n,

$$\hat{f}(\rho) = \sum_{i_n=1}^{n} \rho(i_n, n) \cdots \sum_{i_k=1}^{k} \rho(i_k, k)\hat{f}_{i_n \cdots i_k}(\rho|S_{k-1}).$$

Note that

$$\hat{f}_{i_n \cdots i_k}(\rho|S_{k-1}) = \bigoplus_j \hat{f}_{i_n \cdots i_k}(\rho^{(j)}),$$

where the direct sum is over all the irreducible representations, with multiplicity, that occur in ρ restricted to S_{k-1}. To take advantage of the multiplicity the algorithm must compute each of the terms in the direct sum with the indices $i_n, \ldots, i_k$ fixed as opposed to fixing the restriction and letting the subscripts vary. The recursion computes across levels in the tree, not down branches.

The preceding idea seeks to avoid unnecessary computation. The next idea seeks to avoid unnecessary copying of blocks.

Consider again the computation of the Fourier transform at the representation of S_{10} corresponding to $(7, 3)$. When restricted to S_8 two copies of the irreducible representation corresponding to $(6, 2)$ appear. Consequently, successive restrictions to smaller subgroups contain the same collections of subrepresentations (i.e. generate identical subtrees). Blindly following the above discussion, one might compute both sets of restrictions at these subgroups by computing the first occurrence and then copying to the other identical locations, only to have them overwritten as the recursion returns to S_8. Thus, in

general, the idea is that as the computation proceeds, we must make sure that upon determining that a given block of the matrix is obtained by "copying", no subsequent calculations or copying are performed within that block.

In the tree for S_{10} and the partition $(7, 3)$ the partitions enclosed by square brackets are representations at which Fourier transforms must be computed (the first occurrences). Those within parentheses will have the restricted transforms copied into their corresponding positions in the matrix. Lastly, those inside curly brackets will be ignored entirely. These two ideas are at the heart of the present algorithm. Rockmore (1988) gives a more thorough description of an actual implementation of this algorithm.

The algorithm which follows is in two parts, a "main" body and the subroutine "Compute-Restriction". Compute-Restriction assumes the existence of a subroutine "copy-first-occurrences" which takes as input the partially formed block matrix and moves data as indicated above. Compute-Restriction is called with the following parameters:

k: Indicates computation of the restriction to S_k.

f: Indicates computation of $\hat{f}(\rho|S_k)$.

result: A matrix variable returning the block matrix $\hat{f}(\rho|S_k)$. (Note that this matrix may only be partially filled.)

Other referenced subroutines and matrix variables are as in algorithm 3.

```
main body:
BEGIN
  FOR ρ = first representation of S_n to the last DO
    BEGIN
      branch (ρ, number-of-branches, branches);
      IF all the branches have been computed previously THEN
        FOR i = 1 to n DO
          BEGIN
            FOR j = 1 to number-of-branches DO
              BEGIN
                retrieve (f̂_i (branch [j]);
                build-matrix (temp, f̂_i (branch [j]);
              END
            f-transform= f-transform +ρ(i, n)* temp;
          END
    ELSE
      BEGIN
        FOR i = 1 to n DO
          BEGIN
            Compute-Restriction (n − 1, f_i, temp);
            FOR j = 1 to number-of-branches DO
              BEGIN
                IF branch [j] has been computed THEN
                  BEGIN
                    retrieve (temp);
                    build-matrix (restriction, temp);
                  END
                ELSE
                  copy to storage the block in temp equal to this restriction for later
                  use;
```

```
                    END
                f-transform= f-transform + restriction * ρ(i, n);
            END
        END
    END
END

Compute-Restriction (k, f, result)

BEGIN

                    IF k = M THEN compute this restriction directly (i.e. those rep-
                                  resentations which are "first" occurrences should
                                  be computed and copied to appropriate "later" oc-
                                  currences of the same representation—see preced-
                                  ing discussion)
  ELSE
    BEGIN
      FOR i = 1 to k DO
        BEGIN
          Compute-Restriction (k − 1, restriction, f_i);
          FOR η = first representation in the restriction to S_k TO the last DO
            IF we are to compute f̂(η) THEN
              BEGIN
                temp = the block in restriction corresponding to the restriction of
                       η;
                ηth block of result = ηth block +η(i, k) * temp;
              END
        END
      copy-first-occurrences (result);
    END
END
```

5. Dynamic programming. Michelle Wachs has suggested what she calls a "dynamic programming" algorithm to reorganize the computation of algorithm 3. This allows a dramatic reduction in the amount of required memory. We state it here with her permission.

The main idea is that computation of Fourier transforms on S_m for any m requires only that the restricted transforms on S_{m-1} be computed. Thus, suppose that we were computing $\hat{f}(\rho)$ for all irreducible representations ρ of S_n. Fix a positive integer $M < n$. An efficient reorganization of the computation of algorithm 4 would be to compute first all restricted Fourier transforms at all the irreducible representations of S_M. By piecing these together in the appropriate manner, the restrictions to S_{M+1} may be computed. This can be iterated until $\hat{f}(\rho)$ has been computed for all ρ.

Note that after computing the restricted transforms at any S_m, the restricted transforms at S_{m-1} play no further role in the computation. Thus, they may be removed from storage. Consequently, only $2n!$ floating point numbers need be stored (in addition to the transpositions).

We note that this idea adapts nicely for computing the Fourier transform at any single representation. Specifically, suppose we were computing $\hat{f}(\rho)$

for ρ an irreducible representation of S_n. Using the branching theorem we may then determine all irreducible representations which occur in $(\rho|S_M)$ and then compute all restricted transforms at these representations. As above we may then successively build the restricted transforms at the subgroups S_m for $M \leq m \leq n$, finishing with the computation of $\hat{f}(\rho)$. As illustration refer back to the computation of the Fourier transform at the irreducible representation of S_{10} corresponding to (7.3). This is shown for algorithm 4 in §6.

Consider the tree generated for the splitting of the representation corresponding to this partition. Here, we let $M = 6$. Then the algorithm proceeds by first computing the restricted transforms at the representations corresponding to the partitions (3, 3), (4, 2), (5, 1), and (6) of 6 for the functions

$$\{f_{i_{10},\ldots,i_7} | 1 \leq i_j \leq j\}.$$

These restricted transforms may now be used to build the restricted transforms on S_7 at the partitions (4, 3), (5, 2), (6, 1) and (7). This is iterated until the full Fourier transform has been computed.

In the algorithm which follows let I_m be the set of words

$$I_m = \{i_n, i_{n-1}, \ldots, i_{m+1} | 1 \leq i_j \leq j\}.$$

Thus, for $\alpha \in I_m$ let $f_\alpha = f_{i_n,\ldots,i_{m+1}}$. Let "store-matrix" and "retrieve-matrix" be as before, and let the new subroutine "free-storage" free up the appropriate memory locations (i.e. it should free those positions storing the restricted transforms at S_{m-1} after the restricted transforms at S_m have been computed). Other variables and subroutines are as explained in the previous algorithms.

Wachs' dynamic programming approach:

```
BEGIN
  FOR α ∈ I_M DO
    FOR all irreducible representations ρ of S_M DO
      BEGIN
        compute f̂_α(ρ) directly (á la algorithm 1);
        store-matrix (f̂_α(ρ));
      END
  FOR m = M + 1 TO n DO
    BEGIN
      FOR all representations ρ of S_m DO
        BEGIN
          branch (ρ, number-of-branches, branches)
          FOR ρ ∈ I_m DO
            BEGIN
              FOR i = 1 TO m DO
                BEGIN
                  FOR j = 1 TO number-of-branches DO
                    BEGIN
                      retrieve-matrix (f̂_{α,i} (branches [j]));
                      build-matrix (temp, f̂_{α,i} (branches [j]));
                    END
                  f̂_α(ρ) = f̂_α(ρ) + ρ(i, m) * temp;
                END
```

```
                END
            END
        free-storage;
    END
END
```

7. Some numerical comparisons

The asymptotics for the algorithms on S_n give one an indication of the running time. As a complement to these, we have done some exact counts of the number of additions and multiplications required for the matrix multiplications involved. This assumes that a matrix multiplication routine is available which allows $d \times d$ matrices to be multiplied in d^a operations. It neglects other costs such as bookkeeping and matrix additions.

The data is presented as three graphs for $a = 3, 2.5, 2$ (see Figures 1–3). On the x-axis n runs from 3 to 9. The y-axis shows the base 10 log of the ratio of the running time of the direct algorithm (algorithm 1) over the running time of algorithm 3 (full branching, full memory shown dotted) and algorithm 4 (full branching, limited memory, shown solid).

Thus, for $a = 3$ and $n = 9$, the direct algorithm used 13,168,189,400 operations, algorithm 3 used 243,802,167 operations, and algorithm 4 used 1,610,184,048 operations. The ratios Direct/3 and Direct/4 were 540 and 82 respectively. This leads to logs of 2.7 and 1.9 respectively. It follows that algorithm 3 speeds up direct computation by a factor of $10^{2.7}$ and algorithm 4 speeds up direct computation by a factor of $10^{1.9}$ when $a = 3$, $n = 9$.

For $n = 9$, these numerical results show a speed-up of about 100-fold for the implemented algorithm, and of about 1000-fold for the algorithm using full branching and memory. The two graphs appear to be separating exponentially.

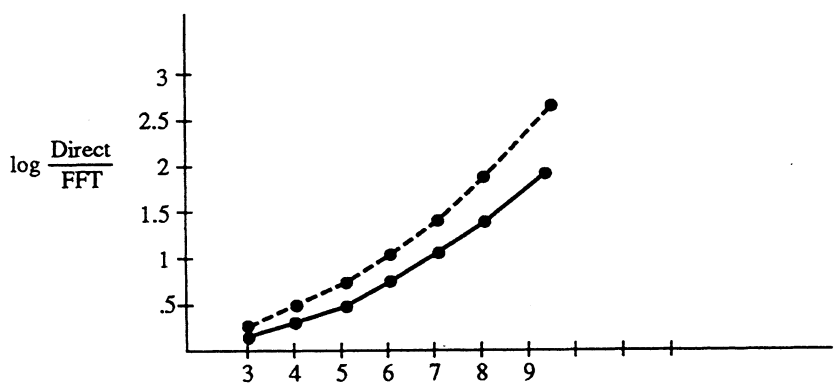


FIGURE 1. Running time for algorithm 3 (dashed) and algorithm 4 (solid) versus direct, $a = 3$.

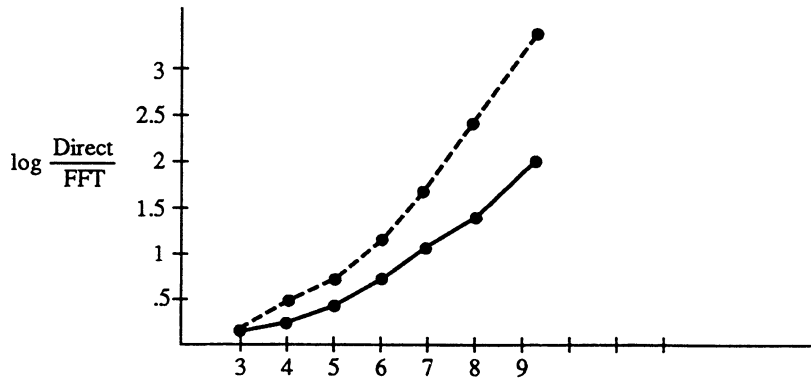


FIGURE 2. Running time for algorithm 3 (dashed) and algorithm 4 (solid) versus direct, $a = 2.5$.

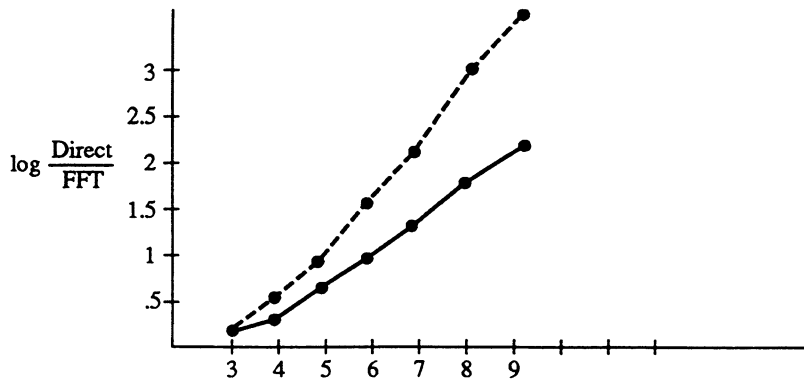


FIGURE 3. Running time for algorithm 3 (dashed) and algorithm 4 (solid) versus direct, $a = 2$.

We remind the reader that the storage requirements for algorithm 3 become prohibitive for large n.

Remark. We believe the counts reported here offer reasonable comparisons of actual running times. Neglected operations should only make a small difference. For example, the number of operations used in adding the matrices $f(\pi)\rho(\pi)$ has been neglected above. It can be incorporated into the recurrences, but makes no material difference. For instance, in algorithm 3, the recurrence changes to

$$T(n) = nT(n-1) + (n-1)\sum_{\lambda \vdash n}(d_\lambda^a + d_\lambda^2).$$

This changes things by a negligible amount if $a > 2$, and at worst by a factor of 2 for $a = 2$.

8. Final remarks

A. Applications. The Discrete Fourier Transform (DFT) on $Z/2^aZ$ was originally introduced by Gauss to compute the cosine transform of a numerical function. Goldstine (1977) contains the early history. There have been numerous reinventions; the most spectacular being those of Good (1958) building on work of Yates in statistics, and Cooley-Tukey (1965) who were interested in signal processing applications.

The DFT has seen other applications: It is used to compute correlations in time series applications, to multiply and evaluate polynomials, and as the base of the modern fast ways of multiplying integers.

Our interest in noncommutative transforms originates in applied statistics problems of analyzing rankings. As explained in Diaconis (1989), a similar spectral analysis may be based on the automorphism group of any designed experiment.

Non-Abelian transforms can be useful in theoretical investigation as well: In investigating rates of convergence of Markov chains to their stationary distributions, knowledge of the eigenvalues is useful. Diaconis and Shahshahani (1981) show that for chains arising as random walks on finite groups, the eigenvalues of the transition matrix are precisely the eigenvalues of the Fourier transforms of the underlying probability measure. Sometimes these eigenvalues have simple structure which can be proved, once guessed at. Numerical computation for small examples is one mainstay. This was a crucial step in the work reported in Diaconis and Shahshahani (1981).

Eigenvalues are also useful in investigating the existence of codes (see Rothaus-Thompson (1966) or Chihara (1987)). They are useful in investigating invertibility of Radon transforms and elsewhere as explained in Diaconis and Graham (1985). Edelman and White (1988) present an energetic data analysis of the eigenvalues of noncommutative transforms for such a problem.

To be honest, we hope that the availability of fast new transforms will suggest novel applications.

B. Parallelizability. Parallel computing environments are beginning to become available. It appears that a naive computational approach yields an algorithm that runs in $O(\log|G|)$ operations. We describe such an algorithm as well as a potentially more practical variation of it which adapts ideas used in the algorithms presented in §6.

The parallel processor we have in mind may be called a fine-grained SIMD machine such as the Connection Machine TM system (see Hillis (1985)). Here "fine-grained" means that there are many processors (perhaps $4k$ or even $128k$) linked together. Each processor has some local memory and usually has limited computational capacity. A SIMD (simultaneous instruction multiple data) machine allows a single instruction to be sent simultaneously to all processors. The instruction is executed or ignored depending on an internal switch.

Direct computation yields a dramatic speed-up under strong assumptions. Let G be a finite group. Suppose that all the irreducible representations can be stored with one matrix entry per processor. Thus, a given processor contains $\rho_{ij}(s)$. As ρ, i, j, s vary this entails $|G|^2$ processors. Given a real-valued function $f(s)$, suppose that this is stored one element per processor as well. We note that these values need not be stored in processors distinct from those containing the matrix elements.

The algorithm proceeds in two steps:

1. Simultaneously send each element $f(s)$ to all processors containing $\rho_{ij}(s)$ (ρ, i, j vary over all possible values), multiply these numbers together, and store them at the receiving processor.
2. Simultaneously sum over s the values $\rho_{ij}(s)$.

In parallel, each of these operations takes $O(\log|G|)$ operations.

Note that any matrix representation may be used. The only potential problem here is in the large storage requirements of this algorithm. Loading a new function $f(s)$ into the machine can be done in just a few seconds (again we have a Connection Machine environment in mind). Putting the data into any particular configuration may be done with a single machine instruction.

We can relax these storage requirements by adapting the basic idea of §2. Recalling the notation there, we fix a subgroup $H \subset G$ and coset representatives $s_1, \ldots, s_k$. Assume that the irreducible representations for H are stored one element per processor. Furthermore, we also assume that all the irreducible representations of $s_2, \ldots, s_k$ (s_1 may be taken to be the identity) are stored as well. (If storage capacity is being pressed, we might only assume that at any given time we only keep the function f_i and each $\rho(s)$ for all irreducibles ρ and some fixed i.) This requires at most $k|G| + |H|^2$ processors. In the algorithm "sketch" that follows, f-transform is a parallel matrix variable with segments indexed by the representations of G that will hold the Fourier transforms.

BEGIN

Simultaneously for all irreducible representations ρ of G set the matrices f-transform(ρ) to 0.

FOR $i = 1$ to k DO

BEGIN

1. Compute $\hat{f}_i(\eta)$ for all irreducible representations η of H in parallel as indicated above (i.e. directly).
2. Simultaneously build the matrices $\hat{f}_i(\rho|H)$ for all irreducible representations ρ of G. (Recall that in a suitable basis, these will be block diagonal with the blocks having been computed in step 1.)
3. Simultaneously, for all irreducible representations ρ of G, compute and add to f-transform(ρ) the matrix products $\rho(s_i) * \hat{f}_i(\rho|H)$.

END

END

Matrix multiplication of n by n matrices may be done in $O(\log n)$ time in parallel. Thus, the running time of step 3 is dictated by the size of the largest representation. This is at most $\sqrt{|G|}$. Data movement may also be performed in log time (that is, any single piece of data may be sent to n processors in

$\log n$ operations). So, in building the block matrices we may assume $O(\log|G|)$ operations. Hence, the above algorithm gives on the order of

$$(|G|/|H|)\log(|G|) + \log(|H|) + \tfrac{1}{2}\log(|G|)$$

operations. This is small when $|H|$ is large. The storage grows as $|G|^2/|H|+|H|$. This allows some tradeoff.

We recognize the limitations in the suggestions made here but feel they may be useful as a start.

C. Other ideas. The Cooley-Tukey ideas are not the only novelties in computing transforms. The chirp-z and primitive root transforms allow computation of all transforms on Z/pZ in order $p\log p$ operations for p a prime. Diaconis (1980) contains a brief description and references to the literature. These ideas all extend the function to a larger group where the transform is easy to compute and the answer is recoverable from the extended computation. The Cooley-Tukey algorithm is given a thought provoking reinterpretation as polynomial evaluation in Aho, Hopcroft, and Ullman (1976). It is not clear how these ideas extended to non-Abelian cases.

Winograd (1978), and Auslander-Tolimieri (1979) have had other different ideas to speed up conventional FFT's. These too seem fair game for extension.

D. Inverse transforms. This paper has focussed on computing the Fourier transforms $\hat{f}(\rho)$. While this is all that is needed for the statistical applications, it is clearly desirable to have efficient algorithms for computing the inverse transform. This takes as input a collection of matrices $\hat{f}(\rho)$ and returns the function values $f(s)$ for every $s \in G$. Direct computation using the Fourier inversion theorem requires $|G|^2$ operations.

For Abelian groups fast algorithms for performing Fourier inversion already exist. These all take advantage of the fact that in this case the irreducible representations of G form a group $\widehat{G}$ (the dual group) isomorphic to G. Fourier inversion then amounts to computing Fourier transforms on $\widehat{G}$—so the same ideas apply. For non-Abelian groups, the dual has no group structure to exploit. New ideas must be developed.

Recently, one of us has shown that the FFT algorithm of this paper essentially can be run backwards to achieve the same savings. A full account of this appears in Rockmore (1990). Here is a sketch of the main idea.

Let G be a group and $H \subseteq G$ a subgroup and suppose that $\{s_1(=1), s_2, \ldots, s_k\}$ is a complete set of coset representatives for G/H. The idea is to find a way to efficiently recover the restricted transforms $\hat{f}_i(\eta)$ (where η runs over all irreducible representations of H) from the original set of Fourier transforms $\hat{f}(\rho)$. This will reduce the problem to performing Fourier inversion for these k functions on H. This idea can be made to work using the techniques of *induced representations.*

Specifically, if $H \subseteq G$ is a subgroup, then representations of G may be constructed from representations of H by a process known as *induction.* If η is a representation of H, then η gives rise to a representation of G, denoted

$(\eta \uparrow G)$ (read η induced to G) of degree $d_\eta \cdot |G|/|H|$ (see Coleman (1966) for a friendly and constructive treatment of induced representations). In general, $(\eta \uparrow G)$ is reducible. Let

$$(\eta \uparrow G) = \rho_1 \oplus \cdots \oplus \rho_m$$

be the direct sum decomposition of $(\eta \uparrow G)$ into irreducible representations of G. In the language of matrices this means that in some basis $(\eta \uparrow G)(s)$ is block diagonal for every $s \in G$ with blocks given by the matrices $\rho_i(s)$. Consequently, with respect to such a basis the Fourier transform $\hat{f}(\eta \uparrow G)$ is block diagonal with the transforms $\hat{f}(\rho_i)$ on the diagonal.

The main fact which is now employed is that there is another more "natural" basis for the induced representation $(\eta \uparrow G)$ with respect to which the Fourier transform $\hat{f}(\eta \uparrow G)$ has a more useful form. With respect to this natural basis the Fourier transform $\hat{f}(\eta \uparrow G)$ may be viewed as a $k \times k$ block matrix in which the first column is given by the blocks $\hat{f}_i(\eta)$.

Now the "algorithm" is clear. Given the original Fourier transforms $\hat{f}(\rho)$ for all irreducible representations ρ of G the block diagonal form of $\hat{f}(\eta \uparrow G)$ may be constructed immediately. An appropriate change of basis allows the recovery of the restricted transforms $\{\hat{f}_i(\eta)\}$. These changes of basis matrices are readily constructed by a neat application of Frobenius reciprocity. This is done for all irreducible representations η of H. The problem is now reduced to performing Fourier inversion for the functions $\{f_i\}$. This may of course now be iterated through subgroups of H to amplify the savings.

Open problems. In the course of doing this work, several theoretical problems have surfaced:

1. Let G be a group, and let $S \in G$ be a set of generators. Given $\rho(s)$, for each irreducible representation ρ and each s in S, how should $\rho(t)$ be built up, for all $t \in G$, with the fewest number of operations? If a directed graph is formed with elements of G as vertices and an edge from t to t' if $t' = st$ for some $s \in S$, the problem reduces to finding an intelligable spanning tree and traversing it by multiplying by the appropriate s for each edge.

This is reminiscent of addition chains (Knuth (1981), p. 444) which efficiently generate x^n given x. In the present application, the efficient generation of subsets $\rho(t)$, $t \in T$, is also of interest, with T being the set of coset representations for a subgroup.

2. Let d_λ be the dimension of the irreducible representation of S_n associated to the partition λ of n. Find the behavior of

$$\sum_{\lambda \vdash n} d_\lambda^a.$$

For $a = 1$, this sum counts the number of involutions; its behavior is well understood (Knuth (1975), p. 48). For $a = 2$ it is $n!$. The behavior at $a = 3$ would be useful for the present investigation.

3. We do not know how to take advantage of special symmetry. For example, if $G = (Z/2Z)^n$ and f is invariant under permutation of coordinates, then f is determined by $f(0), f(1), \ldots, f(n)$, where $f(i) \stackrel{d}{=} f(x)$; here x has i ones. The appropriate transform involves Krawtchouk polynomials

$$\hat{f}(k) = \sum_{i=0}^{n} f(i) p_i(k)$$

(see e.g. Stanton (1984)).

If $p_i(k)$ are available, direct computation of $\hat{f}(k)$ for all k takes n^2 operations. Orszag (1986) suggests algorithms that work in order $n(\log n)^2$ operations for similar problems. We do not see how to adapt his ideas.

Similar questions arise for any of the spherical function transforms associated to finite groups (see Stanton (1984)).

Note added in proof. Clausen (1989,a,b) has begun to develop a similar theory in different language. Babai and Royani (1989) discuss exact computation of representations. Babai (1986) gives more refined towers for S_n. Driscoll and Healy have made a breakthrough in working with compact groups.

Acknowledgment

The authors would like to thank Michelle Wachs for a careful reading and her helpful suggestions, especially her ideas for the "dynamic programming" algorithm in §6.

References

A. Aho, J. Hopcroft, and J. Ullman (1976), *The design and analysis of computer algorithm*, Addison-Wesley, Reading, Mass.

R. Askey and A. Regev (1984), *Maximal degrees for Young diagrams in a strip*, European J. Combin. **5**, 189–191.

M. D. Atkinson (1977), *The complexity of group algebra computations*, Theoret. Comput. Sci. **5**, 205–209.

L. Auslander and P. Tolimieri (1979), *Is computing with finite Fourier transforms pure or applied mathematics?*, Bull. Amer. Math. Soc. (N.S.) **1**, 847–897.

L. Babai (1986), *On the length of subgroup chains in the symmetric group*, Comm. Alg. **14**, 1729–1736.

L. Babai and L. Rónyai (1989), *Computing irreducible representations of finite groups*, Technical report, computer science dept. University of Chicago.

T. Beth (1984), *Verfahren der Schnellen Fourier-Transform*, Teubner, Stuttgart.

_____(1987), *On the computational complexity of the Fourier transform on finite groups*, J. Theoret. Compt. Sci. **51**, 331–356.

L. Chihara (1987), *On the zeroes of the Askey-Wilson polynomials, with application to coding theory*, SIAM J. Math. Anal. **18**, 183–207.

M. Clausen (1989a), *Fast Fourier transforms for meta-abelian groups*, SIAM J. Comput. **18**, 584–593.

_____(1989b), *Fast generalized Fourier transforms*, J. Theoret. Compt. Sti. **67**, 55–63.

A. J. Coleman (1966), *Induced representations with applications to S_n and GL_n*, Queen's Papers in Pure and Appl. Math., No. 4, Queen's Univ., Kingston, Ontario.

J. W. Cooley and J. W. Tukey (1965), *An algorithm for the machine calculation of complex Fourier series*, Math. Comp. **19**, 297–301.

D. Coppersmith and S. Winograd (1987), *Matrix multiplication via arithmetic progressions*, Proc. 19th ACM Sympos. on the Theory of Computing (STOC), pp. 1–6.

P. Diaconis (1980), *Average running time of the fast Fourier transform*, J. Algorithms **1**, 187–208.

_____(1989), *A generalization of spectral analysis with application to ranked data*, Ann. Statist. **17**, 349–379.

_____(1988), *Group representations in probability and statistics*, Institute of Mathematical Statistics, Hayward, CA.

P. Diaconis and R. L. Graham (1985), *The Radon transform on Z_2^k*, Pacific J. Math. **118**, 323–346.

P. Diaconis and M. Shahshahani (1981), *Generating a random permutation with random transposition*, Zeit. Wahr. Verw. Gebiete **57**, 159–179.

J. Driscoll and D. Healy (1989), *A symptotically fast algorithms for spherical and related transforms*, Technical report, Dept. of Math. and computer Sci., Dartmouth College.

P. Edelman and D. White (1988), *Codes, transforms, and the spectrum of the symmetric group*, Paper 231, presented at AMS Atlanta meetings.

D. Elliott and K. Rao (1982), *Fast transforms*, Academic Press, Orlando, Fla.

A. Garsia and T. McLarnan (1986), *Relations between Young's natural and the Kazhdan-Lusztig representations of S_n*, Technical report, Department of Mathematics, Univ. of California, San Diego.

H. Goldstine (1977), *A history of numerical analysis from the 16th through the 19th century*, Springer, New York.

I. J. Good (1958), *The interaction algorithm and practical Fourier series*, J. Roy. Statist. Soc. Ser. B **20**, 361–372.

_____(1962), *Analogues of Poisson's summation formula*, Amer. Math. Monthly **69**, 259–266.

W. D. Hillis (1985), *The connection machine*, MIT Press, Cambridge, Mass.

G. D. James (1978), *The representation theory of the symmetric groups*, Lecture Notes in Math., vol. 682, Springer-Verlag, Berlin.

G. D. James and A. Kerber (1981), *The representation theory of the symmetric group*, Addison-Wesley, Reading, Mass.

A. Kerber (1971), *Representations of permutation groups* 1, Lecture Notes in Math., vol. 240, Springer-Verlag, Berlin.

D. Knuth (1981), *The art of computer programming.* II, 2nd ed., Addison-Wesley, Reading, Mass.

_____(1975), *The art of computer programming.* III, Addison-Wesley, Reading, Mass.

B. Logan and L. Shepp (1977), *A variational problem for random Young tableaux*, Adv. in Math. **26**, 206–222.

D. Rockmore (1990), *Fast Fourier analysis for Abelian group extensions*, Adv. in Appl. Math. (to appear).

S. Orszag (1986), *Fast eigenfunction transforms*, in Science and Computers. A Volume Dedicated to Nicholas Metropolis (Gian-Carlo Rota, ed.), Adv. Math. Suppl. Stud., vol. 10, Academic Press, Orlando, Fla., pp. 23–30.

A. Regev (1981), *Asymptotic values for degrees associated with strips of Young diagrams*, Adv. in Math. **41**, 115–136.

D. Rockmore (1988), *Efficient computation of Fourier transforms on the symmetric group*, Proc. 1989 Conf. on Computers and Mathematics E. Kaltafen, S. Walt (eds) Springer, New York.

_____(1989), *Fast Fourier inversion for finite groups technical report, Dept. of Math. Columbia Univ.*

O. Rothaus and J. G. Thompson (1966), *A combinatorial problem in the symmetric group*, Pacific J. Math. **18**, 175–178.

J. P. Serre (1977), *Linear representations of finite groups*, Springer-Verlag, New York.

D. Stanton (1984), *Orthogonal polynomials and Chevalley groups*, in Special Functions: Group Theoretical Aspects and Applications (R. Askey et al., eds.), Dordrecht, Boston, pp. 87–92.

A. M. Vershik and S. V. Kerov (1981), *Asymptotic theory of characters of the symmetric group*, Functional Anal. Appl. **15**, 246–255.

——(1985), *Asymptotics of the largest and typical dimensions of the irreducible representations of a symmetric group*, Functional Anal. Appl. **19**, 21–31.

S. Winograd (1978), *On computing the discrete Fourier transform*, Math. Comp. **32**, 175–199.

ABSTRACT. Let G be a finite group, $f: G \to \mathbf{C}$ a function, and ρ an irreducible representation of G. The Fourier transform is defined as $\hat{f}(\rho) = \sum_{s \in G} f(s)\rho(s)$. Direct computation for all irreducible representations involves order $|G|^2$ operations. We derive fast algorithms and develop them for the symmetric group S_n. There, $(n!)^2$ is reduced to $n(n!)^{a/2}$, where a is the constant for matrix multiplication (2.38 as of this writing). Variations of the algorithm allow efficient computation for "small" representations. A practical version of the algorithm is given on S_n. Numerical evidence is presented to show a speedup by a factor of 100 for $n = 9$.

DEPARTMENT OF MATHEMATICS, HARVARD UNIVERSITY, CAMBRIDGE, MASSACHUSETTS 02138

DEPARTMENT OF MATHEMATICS, COLUMBIA UNIVERSITY, NEW YORK, NEW YORK, 10027