THE UNIVERSITY OF CALGARY

On the Complexity of Computing Characters of Finite Groups

by

Charles Thomas Hepler

A THESIS
SUBMITTED TO THE FACULTY OF GRADUATE STUDIES
IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR THE
DEGREE OF MASTER OF SCIENCE

DEPARTMENT OF COMPUTER SCIENCE

CALGARY, ALBERTA
AUGUST, 1994

Abstract

This thesis examines the computational complexity of the problem of finding the characters of finite groups and some associated problems. The central focus is how the complexity changes according to how the group is specified. We examine two extremes. Considering computations from Cayley tables, when the input size is quadratic in the order of the input group, we observe that we can efficiently invert Burnside's character table algorithm to find class matrices.

We also consider computations involving the symmetric group with inputs of size polylogarithmic in the order of the input group. We show completeness and hardness results for computations of individual characters of the symmetric group. Examining the problem of decomposition of outer products of characters of the symmetric group, we show that a generalization of the problem is computationally hard. We show that lattice partitions can be enumerated efficiently.

Contents

Chapter 1

Introduction

This thesis examines the computational complexity of the problem of finding the characters of finite groups and some associated problems. The main theme is an examination of how the complexity of a problem changes according to how the group is specified.

In all cases, we will be concerned with exact computations using "symbolic" representations of the input, the output, and quantities computed along the way. Computing with symbolic representations has an advantage over fixed precision numerical computations in that, with a bit of care, we can be guaranteed to be able to determine the signs of small numbers and perform equality tests with complete reliability. Further, symbolic representations can be converted to fixed precision numbers with any desired degree of precision.

This approach has three principal disadvantages. The first is that we require symbolic representations of our input and this is unrealistic for some applications. The second is that numerical approximation algorithms may have a lower complexity than exact algorithms. For those problems where the evidence suggests that there are no efficient (polynomial time) exact algorithms,

we must resort to some kind of approximation algorithm. Even for problems with polynomial time algorithms, numerical approximation algorithms may still be sufficiently more efficient to be worthwhile. For more information on approximation algorithms in this area see [BF91]. The third disadvantage is that the manipulation of symbolic representations of numbers is not always straightforward. This is discussed in detail in [Loo83].

Numerical approximation algorithms are beyond the scope of this thesis. The choice to examine symbolic computations can be justified by the fact that some kind of analysis of the exact solution to a problem must be done before one can analyze an approximate solution.

1.1 General Relevance of the Problems

It is difficult to say exactly when group theory first came into being. Certainly some of the ideas associated with group theory, such as the investigation of symmetry, date back before recorded history. We will be mostly concerned with only one small part of the theory of groups, namely, character theory. The two most important fields contributing to the development of the theory of group characters are number theory and physics.

Regarding applications in number theory, we can, not unreasonably, say that group theory started with Évariste Galois around 1830. Certain myths surround his life. Chapter 6 of [Rot89] debunks these myths and clearly demonstrates the presence of subtleties in historical investigations. In light of this, we gloss over the rest of the history of the theory of groups. Among others, Niels Henrik Abel, Augustin-Louis Cauchy, Sir Arthur Cayley, Camille Jordan, Joseph-Louis Lagrange, Marius Sophus Lie, and Ludwig Sylow began an investigation of Galois' groups, finding new and exciting structures along

the way. Ferdinand Georg Frobenius, William Burnside, and Issai Schur are perhaps the most important names associated with the development of the representation theory of finite groups. Our work is directly dependent on the work of these three men and on the work of Alfred Young.

The theory of group characters is used to examine the structure of finite fields by considering the group constructed from a field by omitting the additive identity and considering only multiplication in the field. In particular, by evaluating the characters of such a multiplicative group, one can find the number of solutions to a wide range of equations over the field. For more information see [IR90] or [Edw77].

Character theory has important applications in (at least) two areas of physics, namely, crystallography and quantum mechanics. The power and value of character theory is demonstrated by the duplication of effort across physics and pure mathematics. It was not uncommon for a physicist to work out the structure of some group only to find that a mathematician had already done so. In [Edd56], Sir Arthur Stanley Eddington describes how this happened to him.

Both [McW63] and [Hoc66] discuss crystallographic applications. In quantum mechanics, n–fold degeneracies in the eigenvalues of the wave equation are directly related to the characters of n-dimensional representations of a group. Since the eigenvalues are directly related to observable quantities, determining group characters is very important. The symmetric group is especially important. The solutions to the wave equation for an n–particle spin system can be classified in terms of their symmetries with respect to interchanges of particles. Here, individual character values, decompositions of inner products of characters, and decompositions of outer products of characters are extremely useful. For a detailed discussion, see [Wey50]. A modern treatment

is contained in [DK85]. [Cot63] and [Ham89] are excellent introductions to group theory. They provide physical intuitions for the interpretation of group theoretical statements and contain a wealth of applications.

1.2 Summary and Readers' Guide

Chapter 2 discusses background information. We give some combinatorial definitions. Then, we give definitions and notation for the relevant aspects of algebra. We provide a brief overview of complexity theory. We review a hardness proof for a known hard problem and give definitions and citations for others. Finally, we discuss the complexity of various useful computations on groups. The reader may freely skip this chapter, returning to it only upon encountering an unfamiliar term.

Chapter 3 examines the complexity of finding complete character tables of finite groups from Cayley tables. We describe Burnside's algorithm for finding character tables from multiplication tables and note that it can be done in polynomial time. For more complete information on this topic, see [Ebe89]. We observe that all but the first step of Burnside's algorithm can be inverted efficiently. This result is not especially surprising but it is of some significance given the recent work done on computing characters from a partial tabulation of the "class matrices".

Chapter 4 looks at computing individual characters of the symmetric group. This problem has very succinct inputs and integer outputs. We examine several versions of this problem and show completeness and hardness results (depending on the formulation of the problem). These are the most significant new results in the thesis. As far as we are aware, they are the first completeness results in this area. The proof is especially satisfying since it uses only

elementary techniques.

We then turn our attention to decomposing outer products of characters. We had less success with this problem. We invent a generalization of the problem and demonstrate that it is computationally hard. Also, we show that an interesting subproblem has a polynomial time solution by framing a beautiful little theorem of Kreweras in computational terms.

Chapter 5 contains a final summary of the results and a discussion of some related problems, including computations of character tables of arbitrary finite groups from representations that are more succinct than Cayley tables.

Chapter 2

Background

This chapter includes some necessary background information. It is intended to review relevant material and to familiarize the reader with our notation. It is divided into three sections: one containing combinatorial definitions, one on algebra, and one on the theory of computing.

The first section gives definitions of permutations, partitions, and lattice partitions.

The second section discusses groups. The symmetric group S_3 is used as a running example for a brief description of Cayley tables, permutation groups, matrix representations, and characters.

The third section presents some aspects of the theory of computational complexity. We describe the classes P, NP, PP, and $\#P$ and discuss reductions, hardness, and completeness. We give definitions of two problems with known complexity: 4-PARTITION and Boolean Permanent.

The problems are used in chapter 4 to give a new classification of the computational complexity of computing characters of the symmetric group.

2.1 Three Combinatorial Definitions

Permutations are used to represent groups. Partitions are used to specify cycle structures of permutations. In particular, the conjugacy classes in the symmetric group can be encoded using partitions (see Section 2.2.2). Also, the absolutely irreducible representation classes of the symmetric group can be specified by partitions. Lattice partitions are used by the Littlewood–Richardson rule (see Section 4.2).

2.1.1 Permutations

This material is standard. For example, see [Bur55].

A *permutation* π of n objects is an invertible function from the set of objects onto itself.

$$\pi : \{a_1, a_2, \ldots, a_n\} \rightarrow \{a_1, a_2, \ldots, a_n\}$$

Since the function is invertible, it assigns a unique object to each object.

We will only be concerned with finite sets. Thus, we can specify a permutation by listing its value for each of the objects. Supposing that $\pi(a_i) = b_i$, we can write the permutation π as

$$\begin{pmatrix} a_1, a_2, \ldots, a_n \\ b_1, b_2, \ldots, b_n \end{pmatrix}$$

We call the elements of the set *points* and say that a permutation acts on that set of points.

Let us consider the images of a single symbol a as we repeatedly apply the same permutation π. Since the set of possible images is finite and π is a bijective function, there must be a smallest positive integer k such that $\pi^k(a) = a$. A *cycle* in π is a finite series of points obtained by repeatedly applying a permutation π to a single point until we return to that point. We

write a single cycle $(a_i, \pi(a_i), \pi^2(a_i), \ldots, \pi^k(a_i))$ where $\pi^{k+1}(a_i) = a_i$ and there is no $j < k$ such that $\pi^j(a_i) = a_i$. Cycles can be written starting at any point in the cycle. We will enclose cycles with parenthesis: '(' and ')'.

It is possible to represent any permutation as a list of disjoint cycles. We call this representation the *cycle form* of a permutation. The representation of a permutation as a list of disjoint cycles is unique up to the order in which the cycles are written and the starting points of the cycles. For the sake of brevity and clarity, we will usually omit cycles of length one.

The *cycle structure* of a permutation is a list of the lengths of the disjoint cycles needed to express the permutation. (The lengths of cycles of length one are always included in this list). The cycle structure for a permutation is unique up to the order in which the lengths are written.

As a conceptual simplification, we use the term *multiplication of permutations* to denote functional composition of permutations. Bearing this in mind, we read products of permutations from right to left rather than left to right. Since we will not be concerned with the nature of the symbols that are being rearranged, we can save ourselves some writing by always working with the symbols $\{1, 2, \ldots, n\}$.

Example 2.1.1: Multiplication of Permutations in Cycle Form

Consider the set of points $\Omega = \{1, 2, 3\}$. Multiplying the permutation $\alpha = (1, 3, 2)$ acting on Ω by the permutation $\gamma = (2, 3)$ gives $\gamma \circ \alpha = (1, 2)$. ■

Any permutation can be written as a product of (not necessarily disjoint) cycles of length two (called two–cycles). This representation is not unique. However, for any particular permutation, the ways of writing that permutation

as a product of two–cycles have the same parity. That is, if a permutation can be written as a product of an even number of two–cycles, then every way that that permutation can be written as a product of two–cycles uses an even number of two–cycles. We call such a permutation *even*. Similarly, *odd* permutations are those permutations that can be written as a product of an odd number of two–cycles. For example, a cycle of length k is even if and only if k is odd since

$$(\sigma_1, \sigma_2, \ldots, \sigma_k) = (\sigma_1, \sigma_2)(\sigma_2, \sigma_3)(\sigma_3, \sigma_4)\ldots(\sigma_{k-2}, \sigma_{k-1})(\sigma_{k-1}, \sigma_k).$$

It is convenient to use a function to capture the above fact. We define the function:

$$\text{sign}(\pi) = \begin{cases} 1 \text{ if } \pi \text{ is even} \\ -1 \text{ otherwise } (\pi \text{ is odd}). \end{cases}$$

2.1.2 Partitions

A *partition* of a positive integer n is a sequence λ of positive integers

$$\lambda = (\lambda_1, \lambda_2, \ldots, \lambda_k)$$

such that $\lambda_i \geq \lambda_{i+1})$ for $1 \leq i \leq k-1$ and such that $\sum_{i=1}^{k} \lambda_i = n$. If for $1 \leq j < k$, $\lambda_j \neq 0$ and $\lambda_{j+1} = 0$, or if $j = k$ and $\lambda_k \neq 0$, then j is the *length* of λ. We use the notation $\lambda \vdash n$ to say that λ is a partition of n. We will use partitions of n to specify the cycle structures of permutations acting on n points.

2.1.3 Lattice Partitions

A *lattice partition* A corresponding to a partition

$$\lambda = (\lambda_1, \ldots, \lambda_m) \vdash n$$

is a string $A = \alpha_1, \ldots, \alpha_n$ of length n of m symbols $\sigma_1, \sigma_2, \ldots, \sigma_m$ such that:

1. For $1 \leq i \leq m$, the string A has exactly λ_i occurrences of the symbol σ_i.

2. For each prefix string $A_i^{\text{pre}} = \alpha_1, \ldots, \alpha_i$, $(1 \leq i \leq n)$, for each j and k, $(1 \leq j < k \leq m)$, there are at least as many occurrences of σ_j as of σ_k.

That is, letting $\#(\sigma_i, X)$ denote the number of occurrences of σ_i in a string X,

$$\#(\sigma_j, A_i^{\text{pre}}) \geq \#(\sigma_k, A_i^{\text{pre}}) \tag{2.1}$$

for all $1 \leq i \leq n$ and $1 \leq j < k \leq m$, and $\#(\sigma_i, A) = \lambda_i$.

For clarity and to save writing, we will write σ_i as just i. With this notation, we see that '1123212' is a lattice partition of $\lambda = (3, 3, 1)$ but '1132212' is not since the prefix string '113' contains one 3 but zero 2's.

2.2 Background in Algebra

This section contains a review of definitions and cites a few useful theorems. It is not intended as an introduction to the subject. The reader not familiar with the definitions below should consult a textbook on modern algebra. [Bur55] is a good, though old, introduction to the subject. A more computationally oriented and modern introduction can be found in [Mig91]. Both [But91] and [Wie64] are good references for permutation groups.

2.2.1 Groups

The fundamental mathematical structure that we will consider is the *group*. We provide a definition for the sake of completeness and to familiarize the reader with our notation.

Definition 1 *A* group G *is a set of objects for which an associative binary operation* $*$ *is defined. The set must be closed with respect to the operation. The set must contain an identity element and each element must have an inverse.* ■

A group is a *finite group* when the set of objects is finite. A finite group is said to have *order* n when the set of objects has size n.

In general, we will use G to stand for an arbitrary finite group and I to stand for the identity element in that group. Occasionally when we are considering several groups at the same time, we will write I_G to indicate the identity element in G. Frequently, we will drop the $*$ sign and refer to the group operation as multiplication. Also, we will use the symbol G to refer to the set and let the operation be understood. We use the exponential notation g^k to indicate a product of k copies of g.

A finite group may be entirely specified by its multiplication table, with rows and columns indexed by group elements (a and b respectively) and with products ($a * b$) as table entries. The multiplication table for a group is often called its Cayley table.

Example 2.2.2: A Small Group

The set $E = \{I, \alpha, \beta, \gamma, \delta, \zeta\}$ together with the operation given by the Cayley table shown in figure 2.1 is a finite group. Later, it will be convenient to have such a table at hand. Again, for convenience later on, a table of the inverses of each element is given in Figure 2.2.

The second table shows that each element has an inverse. It is an easy matter to verify that the set E together with the operation

$g:$	I	α	β	γ	δ	ζ
$I*g$	I	α	β	γ	δ	ζ
$\alpha*g$	α	β	I	δ	ζ	γ
$\beta*g$	β	I	α	ζ	γ	δ
$\gamma*g$	γ	ζ	δ	I	β	α
$\delta*g$	δ	γ	ζ	α	I	β
$\zeta*g$	ζ	δ	γ	β	α	I

Figure 2.1: The Cayley Table for E.

$g:$	I	α	β	γ	δ	ζ
$g^{-1}:$	I	β	α	γ	δ	ζ

Figure 2.2: Inverses in E.

$*$ satisfies the other properties. ■

In the above example, it is not true that $h * g = g * h$, for each $g, h \in E$. In particular, $\alpha * \gamma = \delta \neq \zeta = \gamma * \alpha$. If we do have this additional property, called *commutativity*, then we say that the group is *Abelian.*

The *direct sum* $G = G_1 \oplus G_2$ of two groups G_1 and G_2 is the group of ordered pairs in $G_1 \times G_2$ with the group operation defined componentwise. That is, if $g_1, h_1 \in G_1$ and $g_2, h_2 \in G_2$, then $g = (g_1, g_2)$ and $h = (h_1, h_2)$ are elements of G and their product is defined to be $g * h = (g_1 * h_1, g_2 * h_2)$. It is a straightforward exercise to verify that $G_1 \oplus G_2$ is a group whenever G_1 and G_2 are groups.

A *homomorphism* from a group G_1 to another group G_2 is a function $\phi : G_1 \to G_2$ that preserves group multiplication. That is, in order for ϕ to be called a homomorphism, for all $g_1, g_2 \in G_1$, it must be the case that $\phi(g_1)\phi(g_2) = \phi(g_1 g_2)$. It follows that if $\phi : G_1 \to G_2$ is a homomorphism then $\phi(I_{G_1}) = I_{G_2}$ and $\phi(g^{-1}) = \phi(g)^{-1}$ for all $g \in G_1$. If, in addition, ϕ is one to one and onto, then ϕ is called an *isomorphism* and the groups G_1 and G_2 are said to be *isomorphic.*

A *subgroup* H of a group G is a nonempty subset of G which is still a group under the binary operation $*$ of G restricted to members of H. We write $H \leq G$ (or $G \geq H$). In case H is strictly smaller than G, we say that H is a *proper subgroup* of G and write $H < G$ (or $G > H$). In contrast to the case with rings, $H \leq G$ implies that $I_G \in H$ and is the identity element in H. Further, if $h \in H$ then the inverse of h in G, h^{-1}, is also in H and is the inverse of h in H as well.

The *trivial subgroup* of G is the set consisting only of the identity element in G.

Example 2.2.3: Subgroups of E

Our group E has four nontrivial proper subgroups. They are: $E_\alpha = \{I, \alpha, \beta\}$, $E_\gamma = \{I, \gamma\}$, $E_\delta = \{I, \delta\}$, and $E_\zeta = \{I, \zeta\}$. The group properties are easily verified. ■

The *left coset* of a subgroup H of G determined by $g \in G$ is:

$$gH = \{gh : h \in H\}. \tag{2.2}$$

The element gxg^{-1}, where $g, x \in G$, is a *conjugate* of the element x in G. We say that gxg^{-1} is the *conjugate of x with respect to g.*

$g:$	I	α	β	γ	δ	ζ
$g\alpha g^{-1}$	α	α	α	β	β	β
$g\beta g^{-1}$	β	β	β	α	α	α

Figure 2.3: α and β are conjugates in E.

Definition 2 *The* conjugacy class $C^G(x)$ *of* $x \in G$ *is the set of all conjugates of* x *in* G*:*

$$C^G(x) = \{gxg^{-1} : g \in G\}. \blacksquare$$

Example 2.2.4: Conjugacy Classes in E

E has three conjugacy classes:

$\{I\}$ is a conjugacy class since $gIg^{-1} = I$ for all $g \in E$.

$\{\alpha, \beta\}$ is a conjugacy class. The conjugates for α and β with respect to each element of E are shown in figure 2.3.

$\{\gamma, \delta, \zeta\}$ is easily seen to be a conjugacy class as well. $\blacksquare$

The set of all elements in G that commute with a particular element $x \in G$ is called the *centralizer* of x in G, is written $C_G(x)$, and is a subgroup of G. The set of all elements in G that commute with every element of G is called the *center* of G and is an Abelian subgroup of G.

A set $S \subseteq G$ is said to *generate* a finite group G if every element of G can be expressed as a product of elements of S. The set S is then called a *generating set* for G and we write $G = \langle S \rangle$.

Element	I	α	β	γ	δ	ζ
Expression	α^3	α	α^2	γ	$\alpha\gamma$	$\alpha^2\gamma$

Figure 2.4: E is generated by $\{\alpha, \gamma\}$.

Example 2.2.5: A Generating Set for E

The set $S = \{\alpha, \gamma\}$ generates E, since each element of E can be written in terms of elements of the set, as is shown in Figure 2.4.

These expressions are not unique. ■

2.2.2 Permutation Groups

A set of permutations acting on a set Ω of size n generates a group where the group multiplication operation is defined to be permutation multiplication. Such a group is called a *permutation group* and is said to be of *degree n*. Every finite group is isomorphic to a permutation group. Proof of this statement may be found in any standard text (for example, see [Bur55]).

Example 2.2.6: A Permutation Group

Let the permutations α and γ act on the set $\Omega = \{1, 2, 3\}$. The permutations: $\alpha = (1, 3, 2)$ and $\gamma = (2, 3)$ generate the permutation group $\{I_\Omega, (1, 2, 3), (1, 3, 2), (1, 2), (1, 3), (2, 3)\}$. We have seen that $\gamma \circ \alpha = (1, 2)$. Also, $\gamma \circ \alpha \circ \alpha = (1, 3)$, $\alpha \circ \alpha = (1, 2, 3)$, and $\gamma \circ \gamma = I_\Omega$, so all of the listed permutations can be generated from

Element of E	Permutation
I	$(1)(2)(3)$
α	$(1,3,2)$
β	$(1,2,3)$
γ	$(2,3)$
δ	$(1,2)$
ζ	$(1,3)$

Figure 2.5: An Isomorphism Between E and S_3

α and γ. Since there are no other permutations on Ω, this is the group generated by α and γ, as claimed. ■

Definition 3 *The* symmetric group S_n *is the permutation group containing all permutations of* n *objects.* ■

Example 2.2.7: The Symmetric Group S_3

The example group E is isomorphic to S_3. An isomorphism is shown in figure 2.5. Recalling that products of permutations are read from right to left, it is an easy matter to verify that the group operation is preserved. ■

The symmetric group S_n has a very simple generating set. Let S_n act on $\Omega = \{1,2,\ldots,n\}$. The set $\{(1,2),(1,2,\ldots,n)\}$ (written in cycle form) generates S_n.

The definitions given in the previous section can be carried over to permutation groups.

The definition of direct sum can be conveniently reformulated for permutation groups as follows. The direct sum G of two permutation groups G_1 acting on Ω_1 and G_2 acting on Ω_2, where Ω_1 and Ω_2 are disjoint, can be found by constructing all permutations π acting on $\Omega_1 \cup \Omega_2$ such that $\pi(\Omega_1) = \Omega_1$, $\pi(\Omega_2) = \Omega_2$ and such that the restriction of π to Ω_1 or Ω_2 is a member of G_1 or G_2, respectively.

Next, consider the conjugacy classes of S_3 and observe the relationship between conjugacy classes and cycle structures.

Example 2.2.8: The Conjugacy Classes of S_3

From example 2.2.1, we have that the conjugacy classes in the group E are: $\{e\}$, $\{\alpha, \beta\}$, and $\{\gamma, \delta, \zeta\}$. Using the isomorphism shown in figure 2.5, we see that these translate into the sets $\{(1)(2)(3)\}$, $\{(1,2,3),(1,3,2)\}$ and $\{(1)(2,3),(1,3)(2),(1,2)(3)\}$, having elements whose cycle structures are $(1,1,1)$, (3), and $(2,1)$ respectively. ■

This is not a coincidence. In fact, the conjugacy classes of the symmetric group are characterized by their cycle structures. Any two elements of the symmetric group with the same cycle structure are conjugate and any two conjugates have the same cycle structure (see, for example, [CR62]). Thus, we can specify a conjugacy class in the symmetric group by giving a partition which specifies a cycle structure. In general, all elements of a single conjugacy class in a permutation group have the same cycle structures although two elements of a permutation group may have the same cycle structure without being conjugate. For example, in the group $G = \langle (1,2),(3,4) \rangle$ acting on $\Omega = \{1,2,3,4\}$, the elements $(1,2)$ and $(3,4)$ have the same cycle structure,

$\lambda = (2,1,1)$, but are not conjugate.

2.2.3 Representations and Characters

There is much that can be said about representations of groups. We merely touch on a few of the theorems that are most useful to us. There are many texts on the subject and a large proportion of the introductory group theory texts contain several chapters on representation theory. The reader is referred to [Keo75], [FH62], and [Ser77] for general treatments of the theory of matrix representations. For the representation theory of the symmetric group, see [JK81] and [dBR61]. [Led87] is a good introduction to character theory.

Definition 4 *A* representation T *of a group* G *is a homomorphism* $T : G \rightarrow H$. *Since* T *is a homomorphism, there must be a binary operation defined on* H *such that*

$$T(x)T(y) = T(xy) \in H$$

for all $x, y \in G$. ■

Representations are most useful when they are homomorphisms from an abstract group to a less abstract structure. This allows one to investigate an abstract group by examining a more easily understandable structure. In addition, using such concrete representations, one can specify a group considerably more succinctly than would otherwise be possible.

The isomorphism between our example group E and the symmetric group S_3 is a representation. The group E is abstract. The group S_3 is a set of relatively less abstract objects, namely permutations. This type of representation gave us a straightforward characterization of the conjugacy classes of the symmetric group. Whenever the codomain of a representation is a set of

permutations, we call the representation a *permutation representation.* Also, whenever a representation is injective, we say that the representation is *faithful.*

Let K be a field. Let $GL(n, K)$ be the group of invertible $n \times n$ matrices over K.

Definition 5 *A* matrix representation *of dimension (or degree) n over K is a representation $T : G \to GL(n, K)$ of G.* ■

Example 2.2.9: Matrix Representations of S_3

We give three representations over $\mathbb{C}$ of S_3. The first representation is the *trivial representation.* All of the elements of S_3 are taken to the 1 by 1 identity matrix. For technical reasons (see [dBR61]), we call this representation $A_{(3)}$ and define it as follows: $A_{(3)}(\pi) \stackrel{def}{=} [1]$.

The second representation that we will consider is called the *alternating representation.* The elements of the group are taken either to the 1 by 1 identity matrix or to the matrix $[-1]$ depending on whether the permutations are even or odd. We call this representation $A_{(1,1,1)}$ and define it as follows:

$$A_{(1,1,1)}(\pi) \stackrel{def}{=} \begin{cases} [1] \text{ if } \pi \text{ is even} \\ [-1] \text{ otherwise.} \end{cases}$$

The third representation is more interesting. We call it $A_{(2,1)}$ and define it with the table shown in figure 2.6. ■

$\pi \in S_3$	$A_{(2,1)}(\pi)$
$(1)(2)(3)$	$\begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}$
$(1,2,3)$	$\begin{pmatrix} -\frac{1}{2} & \frac{\sqrt{3}}{2} \\ -\frac{\sqrt{3}}{2} & -\frac{1}{2} \end{pmatrix}$
$(1,3,2)$	$\begin{pmatrix} -\frac{1}{2} & -\frac{\sqrt{3}}{2} \\ \frac{\sqrt{3}}{2} & -\frac{1}{2} \end{pmatrix}$
$(1)(2,3)$	$\begin{pmatrix} -\frac{1}{2} & \frac{\sqrt{3}}{2} \\ \frac{\sqrt{3}}{2} & \frac{1}{2} \end{pmatrix}$
$(1,3)(2)$	$\begin{pmatrix} -\frac{1}{2} & -\frac{\sqrt{3}}{2} \\ -\frac{\sqrt{3}}{2} & \frac{1}{2} \end{pmatrix}$
$(1,2)(3)$	$\begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix}$

Figure 2.6: A Matrix Representation of S_3

The *tensor product* $C = A \otimes B$ of two square matrices A and B, with dimensions m and n respectively, is obtained by replacing each entry in A with the product of that entry and the matrix B to get the $mn \times mn$ matrix

$$\begin{aligned} C &= A \otimes B \\ &= \begin{pmatrix} a_{11}B & a_{12}B & \cdots & a_{1m}B \\ a_{21}B & a_{22}B & \cdots & a_{2m}B \\ \vdots & \vdots & \ddots & \vdots \\ a_{m1}B & a_{m2}B & \cdots & a_{mm}B \end{pmatrix} \end{aligned}$$

where A has $(i,j)^{\text{th}}$ entry a_{ij} for $1 \leq i, j \leq m$.

The *tensor product* of two matrix representations A_1 of degree m and A_2 of degree n of a group G is the matrix representation of degree mn of G in which each element $x \in G$ is represented by the tensor product $A_1(x) \otimes A_2(x)$. Since for $m \times m$ matrices A and C and $n \times n$ matrices B and D, $(A \otimes B)(C \otimes D) = (AC) \otimes (BD)$, $A = A_1 \otimes A_2$ is a indeed matrix representation.

A representation T_H of a subgroup $H \leq G$ *induces* a representation $T \uparrow G$ of G. Since we do not make explicit use of the construction, it is not included here. For more information on induced representations see [Ser77] or [FH62].

Suppose $A(x)$ is a representation of G over K and T is a nonsingular matrix (of the same degree) with coefficients in K. Then $B(x) = T^{-1}A(x)T$ is also a representation of G. We say that A and B are *equivalent* over K and write $A \sim B$.

A matrix representation $A(x)$ is *reducible* over K if there exists a non-singular matrix T over K such that

$$B(x) = TA(x)T^{-1} = \begin{pmatrix} C(x) & 0 \\ E(x) & D(x) \end{pmatrix}, \text{ for all } x \in G.$$

In the above, $C(x)$ and $D(x)$ are both matrix representations of G over K.

Theorem 1 (Maschke) *Let G be a finite group of order g, and let K be a field whose characteristic is either zero or has no common factors with g. Suppose $A(x)$ is a matrix representation of G over K such that:*

$$A(x) \sim \begin{pmatrix} C(x) & 0 \\ E(x) & D(x) \end{pmatrix}.$$

Then

$$A(x) \sim \begin{pmatrix} C(x) & 0 \\ 0 & D(x) \end{pmatrix}. \blacksquare$$

A proof is given in [CR62].

The fields we are concerned with are the complex field and finite extensions of the rationals. All of these have characteristic zero, so that Maschke's theorem (theorem 1) is applicable.

A representation is said to be *irreducible* when it is not reducible. If A is a representation of a group G over a field K, and K is a subfield of a field L, then A can also be considered as a matrix representation over L. A representation over a subfield of the complex numbers $\mathbb{C}$ is said to be *absolutely irreducible* when it is an irreducible representation over $\mathbb{C}$.

Equivalent representations are said to belong to the same *representation class*. For any finite group, there are as many absolutely irreducible representation classes as there are conjugacy classes (see [CR62]). This is an upper bound on the number of irreducible representation classes over smaller fields. When the upper bound is met, the field is called a *splitting field* for the group.

The above implies that, since there is a one to one correspondence between partitions of n and the conjugacy classes in S_n, there must also be a one to one correspondence between the absolutely irreducible representation classes of S_n and the partitions of n. There is a natural correspondence between the

partitions of n and the absolutely irreducible representations of S_n which is part of the "special" representation theory of the symmetric group. The reader is referred to [dBR61] for more information.

The representations given for S_3 in the example above are absolutely irreducible representations. The names we gave the representations reflect the natural correspondence between absolutely irreducible representations of S_n and partitions.

Definition 6 *The* character *of a group G with respect to a representation A of dimension n is the function*

$$\varphi(x) = Trace(A(x)) = \sum_{i=1}^{n} a_{i,i}(x). \blacksquare$$

The character has two important properties:

1 Equivalent representations have the same character.

2 If g and h are conjugates in G then $\phi(g) = \phi(h)$ for any character ϕ.

Thus, it makes sense to write characters both as functions of the elements of a group and as functions of the conjugacy classes of a group.

We say that a character is *(absolutely) irreducible* if it is the character of an (absolutely) irreducible representation. When a character is not irreducible, we say that it is a *compound* character. Compound characters can be expressed as linear combinations of irreducible characters. The values of the absolutely irreducible characters for a group with m conjugacy classes can be tabulated in an $m \times m$ table. Unless otherwise specified, when we talk about the characters of a group, we mean the absolutely irreducible characters.

Conjugacy Class	$(1,1,1)$	$(2,1)$	(3)
Character in $A_{(3)}$	1	1	1
Character in $A_{(2,1)}$	2	0	-1
Character in $A_{(1,1,1)}$	1	-1	1

Figure 2.7: The Character Table for S_3

Example 2.2.10: The Character Table for S_3

We can read the characters directly from the absolutely irreducible representations of S_3 given in the example above. The results are shown in the figure 2.7. ■

2.2.4 Character Relations

A more detailed examination of group characters yields some elegant relations among the characters of any group. Aside from being pretty, they can be used to generate the character tables of some very small groups. For example, they are used to compute the character tables of the groups S_3, A_4, and S_4 (the symmetric group on three points, and the alternating and symmetric groups on four points) in [CR62]. We will have to use some additional facts in order to compute character tables for larger groups but these relations will be of use nonetheless.

Let G be a finite group with n elements and k conjugacy classes C_1, C_2, ..., C_k. Let h_i be the number of elements of the conjugacy class C_i and let $\chi^{(1)}, \ldots, \chi^{(k)}$ be the distinct absolutely irreducible characters of G. We denote the dimension of an absolutely irreducible matrix representation of G

with character $\chi^{(i)}$ by z_i. Finally, we use the symbol * on subscripts to refer to the conjugacy class containing the inverses of a given conjugacy class. That is, $C_{i^*} = \{g^{-1} : g \in C_i\}$. We may now write down the *orthogonality relations for group characters*

$$\sum_{g \in G} \chi^{(i)}(hg)\chi^{(j)}(g^{-1}) = \frac{\chi^{(i)}(h) \cdot n}{z_i} \cdot \delta_{ij} \tag{2.3}$$

$$\sum_{g \in G} \chi^{(i)}(g)\chi^{(j)}(g^{-1}) = n \cdot \delta_{ij} \tag{2.4}$$

$$\sum_{l=1}^{k} h_l \chi_l^{(i)} \chi_{l^*}^{(j)} = n \cdot \delta_{ij} \tag{2.5}$$

$$\sum_{l=1}^{k} \chi_i^{(l)} \chi_{j^*}^{(l)} = \frac{n}{h_i} \cdot \delta_{ij} \tag{2.6}$$

for $h \in G, 1 \leq i, j \leq k$ and for δ_{ij} the Kronecker delta.

For proofs of these relations see [CR62] or [Led87].

2.3 Background in Complexity Theory

We quickly and informally describe some aspects of abstract complexity theory and then present a few known hard problems. Those not already confident with this material are encouraged to consult [GJ79] and [HU79]. Further information is contained in the first three of the chapters in the [vL90], namely [vEB90], [Joh90], and [Sei90]. These articles are extremely useful, in part, because of their extensive bibliographies.

We show that Garey and Johnson's proof of the NP–hardness of the decision problem **4–PARTITION** can be adapted to prove $\#P$–hardness for the corresponding enumeration problem. While this is neither surprising nor difficult, we are unaware of the result appearing elsewhere.

For the most part, we follow the notation of [HU79]. Deviations from this notation are noted.

2.3.1 Complexity Classes, Hardness, and Completeness

Informally, the complexity class P is the set of all decision problems with deterministic polynomial time solutions. The set P has been defined to formally capture the notion of the set of tractable problems. The class NP is the set of all decision problems whose positive instances can be verified in polynomial time and clearly contains P. It is unknown whether the classes P and NP are equal but it is widely conjectured that they are not.

The class PP is also important. It may be roughly defined as the set of all decision problems with probabilistic polynomial time solutions. The only bound on the probability of error is that it must be strictly less than $\frac{1}{2}$. NP is contained in PP and it is widely conjectured that the containment is strict. In summary, we have

$$P \subseteq NP \subseteq PP$$

Clearly, $P \neq PP$ is a weaker assumption than $P \neq NP$.

Definitions of P, NP, and PP can be found in [Joh90] and [HU79]. For more detailed information on the relationships between P, NP, and PP, see [Joh90].

We say that a decision problem A is *many-one reducible* to another decision problem B if there exists a function M which maps instances of A to instances of B such that, for a an instance of A, $M(a)$ is a positive instance of B if and only if a is a positive instance of A. Other kinds of reducibilities, such as oracle reducibility, appear in the literature. We will be concerned only with many-one reducibility and thus will omit the modifier 'many-one' in further

discussions. If the function M can be implemented in polynomial time, we say that M is a *polynomial time reduction from A to B* so that A is polynomial time reducible to B. The classes P, NP, and PP have the important property that they are closed with respect to polynomial time reducibility.

Nondeterministic and probabilistic machines can be thought of as having a computation tree rather than a computation path. That is, at any given stage in a computation, nondeterminism arises when there are more than one possible next steps. Thus, rather than proceeding through a series of machine configurations, making a single path, a nondeterministic computation by a given machine on a given input is described by a set of computation paths. Since these paths all start out the same, it is more compact and more illuminating to consider this set as a tree. By allowing the machines to consider the best of these paths or all of the paths at once, we (likely) add power to the machine.

The term *hard* is applied to a problem, a complexity class, and a type of reducibility when it has been shown that all problems in the complexity class can be reduced to the problem using the specified type of reducibility. Together with the fact that P is closed with respect to polynomial time reducibility, this implies that if a problem known to be hard for NP or for PP is in P, then $P = NP$ or $P = PP$ respectively.

We say that a problem is *NP–hard* when it is hard for NP with respect to polynomial time reducibility. Similarly, we say that a problem is *PP–hard* when it is hard for PP with respect to polynomial time reducibility. If, in addition to being hard for a complexity class, a problem is a member of that class, we say that it is *complete* for that class. For NP and PP, this is abbreviated to *NP–complete* and *PP–complete* respectively.

Since we do not believe that $P = NP$, classifying a problem as NP–hard

or NP–complete is highly indicative that the problem is intractable. Since PP contains NP, showing PP–hardness or PP–completeness for a problem is even stronger evidence for the intractability of a problem.

We have used polynomial–time many–one reductions to define NP and PP–hardness. Some sources define NP–hardness with respect to a stricter form of reducibility, "log–space reducibility". This distinction is not important for our results.

The class $\#P$ is the set of all enumeration problems that can be solved in polynomial time by a counting Turing machine. A counting Turing machine is conceptually very similar to a probabilistic Turing machine or a nondeterministic Turing machine. The significant difference is that rather than returning a 'yes' or a 'no' based on the existence of an accepting computation (as for a nondeterministic Turing machine) or a 'yes' or a 'no' based on the ratio of accepting computations to all computations (as for a probabilistic Turing machine), a counting Turing machine returns the number of accepting computations. Since $\#P$ contains enumeration problems rather than decision problems, it includes the class FP of all enumeration problems that are computable by a deterministic Turing machine in polynomial time. It is widely believed, but unproved, that FP is a proper subset of $\#P$ — and it has been shown that $FP = \#P$ would imply $P = NP$.

Since many–one reductions apply to decision problems, we need another kind of reduction in order to prove results about enumeration problems. We say that a function M from instances of an enumeration problem A to instances of an enumeration problem B is a *polynomial time parsimonious reduction* from A to B if M is computable in deterministic polynomial time and there is a function $f : \mathbb{N} \rightarrow \mathbb{N}$ that is computable by a deterministic Turing machine using time polynomial in the length of its input and in the length of the instance

a of A such that, for any instance a of A, if $b = M(a)$ is the corresponding instance of B and m is the output of B on instance b, then $f(m)$ is the output of A on instance a.

An enumeration problem A is *hard* for $\#P$, or "$\#P$–hard", if there is a polynomial time parsimonious reduction from every enumeration problem in $\#P$ to A, and A is *$\#P$–complete* if A is $\#P$–hard and belongs to $\#P$.

Our definition of parsimonious polynomial time reduction is weaker (that is, less restrictive) than that found in the literature which requires that the output for the original instance a of A and for the derived instance b of B be identical. The reduction we call a "polynomial time parsimonious reduction" is frequently called a "polynomial time *weakly* parsimonious reduction." Since the relation "polynomial time weakly parsimonious reducibility" is a transitive relation on enumeration problems and since FP is closed with respect to weakly parsimonious reductions, membership of a $\#P$–hard or $\#P$–complete problem in FP implies that $FP = \#P$. Thus, $\#P$–hardness or completeness is still very good evidence for the intractability of a problem even with the weaker notion of reduction. Since we only use weakly parsimonious reductions, we omit the modifier "weakly" in all that follows.

Some problems involve numerical inputs. Normally, we assume that inputs are encoded efficiently. That is, numbers are represented using a place value system. When the structure of a problem is such that the problem remains complete or hard even when the numbers are represented in a tally system, we say those problems are *strongly* complete or hard. This is conventionally said of NP hard problems. We will also use this terminology for PP and $\#P$ hard problems.

2.3.2 Computing with Algebraic Numbers

We will be concerned with computations involving the complex numbers. Since most complex numbers do not have finite representations, some comment is required.

Whenever we are attempting to compute a value in $\mathbb{C}$, we will always be concerned with problems with both finite specifications and unique solutions. Thus, the numbers are finitely represented by the specification of the problem. However, such a representation is of no use to us. It would be highly desirable if we could efficiently perform operations such as multiplication, addition, and zero testing on the representations of the numbers.

In the cases that we are concerned with, this can be done by working with finite algebraic extensions of the rationals. First, we observe that such fields are subfields of the complex numbers. Furthermore, each can be obtained by adjoining a single algebraic number (say, α) to $\mathbb{Q}$. The generator α can be represented by its minimal polynomial over $\mathbb{Q}$ and by numerical approximation (to distinguish it from the other roots of this polynomial). This information identifies the field $\mathbb{Q}[\alpha]$. Second, any element β of a field $\mathbb{Q}[\alpha]$ can be represented by a polynomial $f \in \mathbb{Q}[x]$ with rational coefficients — namely, the polynomial f (with degree less than that of the minimal polynomial of α) such that $\beta = f(\alpha)$. Arithmetic operations over the field can be implemented in terms of operations on the polynomials used to represent elements of the field. For a detailed discussion see [Loo83].

2.3.3 Problems

The problems described below are used to prove hardness or efficiency results later in the thesis.

2.3.3.1 4–PARTITION

Garey and Johnson [GJ79] show that the problem 4–PARTITION is strongly NP–complete. Their transformation is parsimonious and so we immediately have $\#P$–completeness for the corresponding enumeration problem and PP–completeness for the threshold problem. We follow their notation for 4–PARTITION, use their transformation, and extend their proof of correctness to show that the transformation is parsimonious.

The NP–completeness proof in Gary and Johnson proceeds by reducing 3–Dimensional Matching to 4–PARTITION. We give a definition of 3–Dimensional Matching below:

Decision Problem 1: 3DM

3–Dimensional Matching

Input:

An integer q represented in unary and

a set $M \subseteq W \times X \times Y$, where W, X and Y are disjoint sets, each with q elements.

Question:

Does M contain a *matching*, that is, a subset $M' \subseteq M$ such that $|M'| = q$ and no two elements of M' agree in any coordinate? ■

The problem **3DM** is shown to be NP–complete in Gary and Johnson. Also, it is shown that the corresponding enumeration problem **#3DM** and the corresponding threshold problem **T–3DM** are $\#P$ and PP–complete in [Sim77] and [Gal74]. Although not included in the literature's definition of the problem, our inclusion of q represented in unary does not affect the cited

results. In order for there to be a matching, M must contain at least q elements and thus, inclusion of q represented in unary does not cause a significant increase of the size of the input for the hard instances of the problem. We include q in the input to simplify statements made later.

We give a definition of 4–PARTITION below:

Decision Problem 2: 4–PARTITION

Strongly NP–complete problem

Input:

m: an integer represented in unary,

A: a finite set with $4m$ elements,

B: a positive integer bound represented in unary,

s: a function from A to the positive integers such that if

$a \in A$ then $B/5 < s(a) < B/3$ and such that

$\sum_{a \in A} s(a) = mB$.

Question:

Is there a valid *4–partition* of A? That is, can A be

partitioned into m disjoint sets $S_1, S_2, \ldots, S_m$

such that for $1 \leq i \leq m$: $\sum_{a \in S_i} s(a) = B$? ■

Again, including m represented in unary in the input does not change the complexity of the problem since A has more than m elements. The fact that B can be represented in unary without affecting the NP–completeness of the problem is shown in [GJ79]. The demonstration of this fact is a significant portion of the proof that **4–PARTITION** is strongly NP–complete.

In order to prove that **4–PARTITION** is strongly NP–complete, Gary and Johnson give a transformation from **3DM** to **4–PARTITION** (see pages 97 to 99 of [GJ79]). They prove that the transformation can be done in polyno-

mial time and that it yields an instance of **4–PARTITION** which has element sizes that are bounded by a polynomial in the size of the original instance of **3DM**. We describe the relevant aspects of their proof in order to show that their transformation is parsimonious, and provide a simple example.

The transformation takes an instance $W = \{w_1, w_2, \ldots, w_q\}$, $X = \{x_1, x_2, \ldots, x_q\}$, $Y = \{y_1, y_2, \ldots, y_q\}$ and $M \subseteq W \times X \times Y$ of **3DM** to an instance (A, B, s) of **4–PARTITION** with $4|M|$ elements. The set A contains one member for each element of each of the triples in M. These are indexed by their membership in W, X, or Y and by their position within whichever set they belong to. Thus, the elements of the set A are denoted $w_i[l]$, $x_j[l]$, and $y_k[l]$ where i, j, and k range from 1 to q and for each particular i, j, or k, the variable l ranges from 1 to the number $N(z)$ of times that the element z of W, X, or Y is contained in a triple in M. Thus, by construction, there are exactly $|M|$ elements of A with the form $w_i[l]$ (with $1 \leq i \leq q$ and $1 \leq l \leq N(w_i)$), $|M|$ with the form $x_j[l]$ (with $1 \leq j \leq q$ and $1 \leq l \leq N(x_j)$), and $|M|$ with the form $y_k[l]$ (with $1 \leq k \leq q$ and $1 \leq l \leq N(y_k)$). Finally, the set A includes another $|M|$ elements — denoted $u_1, u_2, \ldots, u_{|M|}$.

The elements $w_i[1]$, $x_j[1]$, and $y_k[1]$ are called *actual elements* where i, j, and k have the same ranges as before. All of the other elements of A except $u_1, u_2, \ldots, u_{|M|}$ are called *dummy elements.*

Gary and Johnson's construction includes formulas (on page 97) defining the sizes for the elements of A. The sizes of the elements depend on (and are computable deterministically in polynomial time from) the indices of the corresponding elements in the set W, X, or Y and on which set they belong to. The actual elements all have different sizes. Each of the dummy elements for a particular element of W, X, or Y has the same size, and the size is different for each different element of W, X, or Y. Furthermore, none of the sizes of the

actual elements is the same as any of the sizes of any of the dummy elements.

Gary and Johnson give the following construction of a 4-partition from a matching. Suppose that $M' \subseteq M$ is a matching. The corresponding 4-partition is made up of $|M|$ 4-sets, each containing a u_l, a $w_i[\cdot]$, an $x_j[\cdot]$, and a $y_k[\cdot]$, where $(w_i, x_j, y_k) = m_l \in M$. If $1 \leq l \leq q$ and $m_l \in M'$, we group u_l with the actual elements $w_i[1]$, $x_j[1]$, and $y_k[1]$. If $m_l \in M - M'$, we group u_l with dummy elements corresponding to w_i, x_j, and y_k. Gary and Johnson show that for every matching $M' \subseteq M$, the above construction gives a valid 4–partition of A. They also give a construction for a matching $M' \subseteq M$ from a valid 4–partition of A, establishing that 4–partitions corresponding to the same matching $M' \subseteq M$ can only differ by having dummy elements for the same element of $W \cup X \cup Y$ exchanged.

We illustrate the transformation below.

Example 2.3.11: Transforming an instance of 3DM to an instance of 4–PARTITION

Let $q = 2$, $W = \{w_1, w_2\}$, $X = \{x_1, x_2\}$, $Y = \{y_1, y_2\}$, and $M = \{(w_1, x_1, y_1), (w_1, x_1, y_2), (w_2, x_2, y_1)\}$ be an instance of **3DM**.

We construct an instance (m, A, B, s) of **4–PARTITION** as follows. First, we count the number of occurrences of the elements of W, X, and Y in the ordered triples of M, summarized below:

$$N(w_1) = 2 \quad N(w_2) = 1$$
$$N(x_1) = 2 \quad N(x_2) = 1$$
$$N(y_1) = 2 \quad N(y_2) = 1$$

Now, letting $r = 32q = 64$ and $m = 12$, we define the size function s (and at the same time actual and dummy elements of the set A)

for each of the elements of $W \cup X \cup Y$ as follows:

$$s(w_1[1]) = 10r^4 + 1r + 1 = 167772225$$
$$s(w_2[1]) = 10r^4 + 2r + 1 = 167772289$$
$$s(w_1[2]) = 11r^4 + 1r + 1 = 184549441$$
$$s(x_1[1]) = 10r^4 + 1r^2 + 2 = 167776258$$
$$s(x_2[1]) = 10r^4 + 2r^2 + 2 = 167780354$$
$$s(x_1[2]) = 11r^4 + 1r^2 + 2 = 184553474$$
$$s(y_1[1]) = 10r^4 + 1r^3 + 4 = 168034308$$
$$s(y_2[1]) = 10r^4 + 2r^3 + 4 = 168296452$$
$$s(y_1[2]) = 8r^4 + 1r^3 + 4 = 134479876$$

For each of the triples of M, we define the size function (and the rest of the elements of A) as follows:

$$s(u_1) = 10r^4 - 1r^3 - 1r^2 - 1r^1 + 8 = 167505864$$
$$s(u_2) = 10r^4 - 2r^3 - 1r^2 - 1r^1 + 8 = 167243720$$
$$s(u_3) = 10r^4 - 1r^3 - 2r^2 - 2r^1 + 8 = 167501704$$

Finally, we set $B = 40r^4 + 15 = 671088655$ and the transformation is complete.

We observe that the only matching in M is $M' = \{(w_1, x_1, y_2), (w_2, x_2, y_1)\}$ and the only valid 4–partition of (m, A, B, s) is:

$$S_1 = \{u_1, w_1[2], x_1[2], y_1[2]\}$$
$$S_2 = \{u_2, w_1[1], x_1[1], y_2[1]\}$$
$$S_3 = \{u_3, w_2[1], x_2[1], y_1[1]\}$$

up to interchange of the indices of the sets S_i. ■

We call two valid 4–partitions of A *equivalent* if one can be obtained from the other by interchange of elements with the same size. This defines an equivalence relation on the 4–partitions of A.

Recall that, by the definition of s, two elements of A have the same size if and only if they are both dummy elements for the same element in $W \cup X \cup Y$. Thus, Gary and Johnson's construction specifies a bijection between the equivalence classes of valid 4–partitions of A and the matchings for the instance of **3DM**.

We now show that these equivalence classes of 4–partitions all have the same size and that this size is easy to compute. Consider the 4–partitions in the equivalence class corresponding to some matching $M' \subseteq M$. The number of ways that the $N(z) - 1$ dummy elements corresponding to a member z in $W \cup X \cup Y$ can be arranged in 4–sets corresponding to elements of $M \setminus M'$ with z as an entry is $(N(z) - 1)!$. The dummy elements corresponding to different elements of $W \cup X \cup Y$ can be placed independently. Thus the number of ways that we can place all of the dummy elements is exactly

$$\prod_{1 \le i \le q} \prod_{1 \le j \le q} \prod_{1 \le k \le q} ((N(w_i) - 1)!(N(x_j) - 1)!(N(y_k) - 1)!). \tag{2.7}$$

Since the sizes of the actual elements are all distinct, this is the size of the equivalence class of 4–partitions of A corresponding to a matching M'. This size does not depend on the matching that is chosen so the total number of 4–partitions in the constructed instance is

$$K \cdot \prod_{1 \le i \le q} \prod_{1 \le j \le q} \prod_{1 \le k \le q} ((N(w_i) - 1)!(N(x_j) - 1)!(N(y_k) - 1)!) \tag{2.8}$$

where K is the number of 3–dimensional matchings M.

The values $N(z)$ can be determined in polynomial time from the description of M. Furthermore, this description has length at least linear in $\sum_{i=1}^{q}(N(w_i) + N(x_i) + N(y_i))$, so the values $(N(z) - 1)!$ can be computed efficiently as well. We can find the product given in equation (2.7) in polynomial time and then, in time polynomial in the size of M, find K using the product just computed,

equation (2.8), and the number of valid 4–partitions in the constructed instance of **4–PARTITION**.

We now give explicit definitions of the threshold and enumeration problems associated with **4–PARTITION** so that we may summarize our results in a single theorem.

Decision Problem 3: T–4–PARTITION

Threshold 4–PARTITION

Input:

m: an integer represented in unary,

A: a finite set with $4m$ elements,

B: a positive integer bound represented in unary,

s: a function from A to the positive integers such that if

$a \in A$ then $B/5 < s(a) < B/3$ and such that

$\sum_{a \in A} s(a) = mB$,

x: a threshold value represented in binary.

Question:

Do there exist strictly more than x valid 4–partitions of A? ■

Number Problem 4: #4–PARTITION

4-PARTITION Enumeration

Input:

m: an integer represented in unary,

A: a finite set with $4m$ elements,

B: a positive integer bound represented in unary,

s: a function from A to the positive integers such that if

$a \in A$ then $B/5 < s(a) < B/3$ and such that

$\sum_{a \in A} s(a) = mB$.

Output:

The number of valid 4–partitions of A. ■

Note that the values of the function s can be represented in unary with no significant increase in the input size since m and B are both represented in unary.

Theorem 2 *The decision problem* ***T–4–PARTITION*** *is PP–complete. The enumeration problem* ***#4–PARTITION*** *is #P–complete.* ■

Proof: It is easy to adapt Gary and Johnson's proof of membership of **4–PARTITION** in NP in order to show that **T–4–PARTITION** belongs to PP and **#4–PARTITION** belongs to $\#P$. We have demonstrated that Gary and Johnson's transformation (establishing NP–hardness of **4–PARTITION**) is parsimonious; this implies PP–hardness of **T–4–PARTITION** and $\#P$–hardness of **#4–PARTITION**. ■

2.3.3.2 Boolean Permanent

The problem **BOOLEAN PERMANENT** is shown to be $\#P$–complete in [Val79]. We formally define the problem below.

Number Problem 5: BOOLEAN PERMANENT

$\#P$–complete problem

Input:

An $n \times n$ matrix M of 0's and 1's.

Output:

The value of the permanent of M, given by:

$$\mathrm{Perm}(M) = \sum_{\sigma \in S_n} \prod_{i=1}^{n} M_{i,\sigma(i)}$$

where, as shown, the sum runs over all $n!$ permutations σ of the n integers $\{1, 2, \ldots, n\}$. ■

Chapter 3

Characters of Finite Groups

There are several sensible ways that a finite group may be specified. The least succinct is to give the complete multiplication table, or Cayley table, for the group. In this chapter, we consider the computational complexity of finding the absolutely irreducible character table for a finite group that is represented by a Cayley table.

Burnside's algorithm is one known method (of several) for computation of a character table from a Cayley table. It is efficient — see [Ebe89] for an analysis. We summarize the algorithm and its analysis. Then, we show that a significant part of Burnside's algorithm can be inverted efficiently using a well known theorem — see [CR81]. In particular, we show that the structure constants used by the algorithm can be found efficiently from an absolutely irreducible character table. This is motivated by some work by Schneider (see for example [Sch90]) on finding characters from incomplete sets of structure constants.

3.1 Computing Characters from Cayley Tables

The problem of finding the character table of a finite group given its Cayley table has a long history and has been the subject of extensive work. For numerous examples, see the surveys [Fel78] and [Neu83].

There has been considerable recent interest in Dixon's modification to Burnside's algorithm ([Dix67]). In particular, Schneider ([Sch90]) has explored the removal of wasted computation in Dixon's algorithm caused by redundancies in the structure constants. In the next section, we demonstrate that the structure constants are entirely recoverable from the character table. Although this is only a minor extension of the work of Burnside, Dixon, and Schneider, it is important in that it shows that the k^3 structure constants contain exactly the same amount of information as the $k \times k$ absolutely irreducible character table. Further, it shows that these pieces of information are equivalent in terms of polynomial time computations.

In order to prove the result in the next section, we now describe Burnside's algorithm. The algorithm is derived in, among other places, [CR62]. Although the details of the correctness proof are interesting, they are rather long and widely available. Thus, a proof is not included here. The algorithm (and two modifications) are analyzed in [Ebe89].

To begin with, let us formally define the problem under consideration.

Problem 6: χ from $\times$

Computation of Character Table from Cayley Table

Input:

A multiplication table for a finite group $G = \{g_1, g_2, \ldots, g_n\}$.

Output:

The character table for G over $\mathbb{C}$. ■

Intuitively, Burnside's algorithm works as follows. First, we compute a set of values expressing connections between conjugacy classes — the *class matrices* of the group. It can be shown using the character orthogonality relations given in section 2.2.4 that the components of the common eigenvectors of the class matrices are directly related to the characters. After computing these common eigenvectors, only a minor amount of rearranging and arithmetic remains in order to obtain the characters.

It is worth noting that Burnside's algorithm and Dixon's modification can be used to compute character tables over some fields other than the complex numbers. In order to simplify matters, we will only be concerned with characters over $\mathbb{C}$.

We now review our notation and state the algorithm formally. Let G be a finite group with order n. As in section 2.2.4, we denote the k conjugacy classes of G by $C_1, \ldots C_k$, with a convention that $C_1 = \{I_G\}$. Aside from this convention, ordering of the conjugacy classes is arbitrary. We use the superscript * on the indices of the conjugacy classes to refer to the conjugacy class containing the inverses of a given conjugacy class. That is, $C_{i^*} = \{g^{-1} : g \in C_i\}$.

Let h_i be the number of elements of the conjugacy class C_i and let $\chi^{(1)}, \ldots, \chi^{(k)}$ be the distinct absolutely irreducible characters of G. We use subscripts to denote the value of the character for members of a particular conjugacy class. That is, $\chi_j^{(i)}$ is the value of the i^{th} irreducible character at an element of the conjugacy class C_j. In a fashion similar to our notation for conjugacy classes, we adopt the convention that $\chi^{(1)}$ be the character of the trivial representation

(so $\chi_j^{(1)} = 1$ for $1 \leq j \leq k$) and that the order of the other representations is arbitrary. We denote the dimension of an absolutely irreducible matrix representation of G with character $\chi^{(i)}$ by z_i. Note that $z_i = \chi_1^{(i)}$ since $I_G \in C_1$.

For $1 \leq r, s, t \leq k$, the *structure constant* c_{rst} is the number of solutions (x, y) to the equation $xy = z$ with $x \in C_r, y \in C_s$, for some fixed $z \in C_t$. The number of solutions is easily shown to be independent of the particular $z \in C_t$ that is picked. Define V_s to be the matrix whose $(r,t)^{\text{th}}$ entry is c_{rst}. The matrices $V_1, V_2, \ldots V_k$ are called the *class matrices.*

It can be shown, using the character orthogonality relations (given in section 2.2.4), that if

$$\omega_i^{(j)} = \frac{h_i \chi_i^{(j)}}{z_j} \text{ for } 1 \leq i, j \leq k$$

then the $\omega_i^{(j)}$'s are both eigenvalues and components of the eigenvectors of the class matrices. In particular,

$$V_i \cdot \begin{bmatrix} \omega_1^{(j)} \\ \omega_2^{(j)} \\ \vdots \\ \omega_k^{(j)} \end{bmatrix} = \omega_i^{(j)} \cdot \begin{bmatrix} \omega_1^{(j)} \\ \omega_2^{(j)} \\ \vdots \\ \omega_k^{(j)} \end{bmatrix} \text{ for } 1 \leq i, j \leq k.$$

It can be shown using linear independence of the characters $\chi^{(1)}, \chi^{(2)}, \ldots, \chi^{(k)}$ that these relations uniquely determine the values $\omega_j^{(i)}$. The character values $\chi_j^{(i)}$ can then be recovered from $\omega_j^{(i)}$ using the orthogonality relations and the fact that $\chi_i^{(1)} = 1$ for $1 \leq i \leq k$. This method for computing character tables is stated in more detail below.

Algorithm 1: Burnside's Character Algorithm

Input:

A multiplication table for a finite group $G = \{g_1, g_2, \ldots, g_n\}$

Output:

The character table for G over $\mathbb{C}$.

Step 1:

Identify a representative x of each conjugacy class C_i in G and find the size of the conjugacy class containing that element. Call the sizes of the k conjugacy classes $h_1, \ldots h_k$. The order is not important other than that $C_1 = \{1\}$.

Step 2:

For each triple (r, s, t) where $1 \leq r, s, t \leq k$, count the number c_{rst} of solutions of $xy = z$ such that $x \in C_r$, $y \in C_s$ for any fixed $z \in C_t$.

Step 3:

For each i where $1 \leq i \leq k$, find the index i^* of the conjugacy class C_{i^*} containing the inverses of the elements of the class C_i.

Step 4:

For each s where $1 \leq s \leq k$, let $V_s \in M_{k\times k}(\mathbb{C})$ be the class matrix given by $(V_s)_{rt} = c_{rst}$ for $1 \leq r, t \leq k$.

Find the eigenvalues and bases for the eigenspaces of each of the matrices V_s. Find bases

$$w_1 = \begin{bmatrix} \omega_1^{(1)} \\ \omega_2^{(1)} \\ \vdots \\ \omega_k^{(1)} \end{bmatrix}, w_2 = \begin{bmatrix} \omega_1^{(2)} \\ \omega_2^{(2)} \\ \vdots \\ \omega_k^{(2)} \end{bmatrix}, \cdots, w_k = \begin{bmatrix} \omega_1^{(k)} \\ \omega_2^{(k)} \\ \vdots \\ \omega_k^{(k)} \end{bmatrix}$$

for the intersections of the eigenspaces, such that these are common eigenvectors of $V_1, \ldots, V_k$, span $M_{k\times 1}(\mathbb{C})$, and each has first component 1.

Step 5:

For $1 \leq i \leq k$, compute the integer

$$z_i = \sqrt{\frac{n}{\sum_{l=1}^{k} \frac{1}{h_l} \omega_l^{(i)} \omega_{l^*}^{(i)}}}$$

Step 6:

For each pair (r, s) such that $1 \leq r, s \leq k$, the $(r, s)^{\text{th}}$ entry in the character table of G is given by:

$$\chi_s^{(r)} = \frac{z_r \cdot \omega_s^{(r)}}{h_s}$$

where $\omega_s^{(r)}$ is the r^{th} component of the vector ω_s.

Step 7:

Output the values $\chi_s^{(r)}$ for $1 \leq r, s \leq k$. ■

Since G is finite, the elements of its character table are algebraic numbers in $\mathbb{Q}[\eta]$ where η is a k^{th} primitive root of unity and k divides the order of G. See section 2.3.2 for a discussion of the representation of these numbers.

The fact that this algorithm can be implemented in polynomial time was used in [Ebe89] to prove the following theorem.

Theorem 3 *χ from $\times \in FP$.* ■

3.2 Inverting Part of Burnside's Algorithm

In this section, we observe that most of Burnside's algorithm (described in section 3.1) is invertible. More specifically, the "structure constants" found in step 2 of Burnside's algorithm can be found efficiently from a character table.

As before, we restrict ourselves to talking about the character table of a finite group G over $\mathbb{C}$. We carry over the notation of the last section. That is, we write G for a finite group of order n. The k conjugacy classes in G are written C_i, where i ranges from 1 to k, and their sizes are written h_i. Once again, $C_1 = \{I_G\}$. By i^* we mean the index of the class C_{i^*} containing the inverses of the elements of the class C_i. The value z_i is the dimension of the i^{th} irreducible representation. Finally, $\chi_j^{(i)}$ is value of the character of the representation class i at the conjugacy class C_j. Columns in character tables correspond to conjugacy classes in the group and rows correspond to the equivalence classes of absolutely irreducible representations.

We now define a new problem.

Problem 7: c_{rst} **from** χ

Structure Constants from Character Table

Input:

An absolutely irreducible character table $\chi_\nu^{(\mu)}$ of a finite group G.

Output:

A "table" of structure constants c_{rst} for the group G where

c_{rst} is the number of solutions to $xy = z$ for any fixed z in the conjugacy class with index t (denoted C_t) and x and y are in C_r and C_s respectively. ■

This problem can be solved in polynomial time by using a few identities to find the size n of the group, the sizes h_i of the conjugacy classes, the values

i^* of the indexes of the conjugacy classes C_{i^*} containing the inverses of the members of the class C_i, and the values z_i of the dimensions of the absolutely irreducible representations of G, and then using the formula

$$c_{rst} = \frac{h_r h_s}{n} \sum_{m=1}^{k} \frac{\chi_r^{(m)} \chi_s^{(m)} \chi_{t^*}^{(m)}}{z_m} \tag{3.1}$$

proved on page 216 of [CR81] to invert the last stages of Burnside's algorithm.

We begin by observing that we can locate the column in χ corresponding to the class consisting of only the identity element, since it will be the only column with only positive integers as entries: this column must have positive integer entries since any matrix representation with dimension z_i must represent the identity element of G by the $z_i \times z_i$ identity matrix. Consider the fourth orthogonality relation (equation (2.6))

$$\sum_{l=1}^{k} \chi_i^{(l)} \chi_{j^*}^{(l)} = \frac{n}{h_i} \cdot \delta_{ij}$$

If there is a second column with index j^* in the character table whose entries are all positive then the value of the left hand side of the orthogonality relation is positive when i and j^* are the indexes of these columns. Since C_i is the conjugacy class containing the identity element, $i^* = i$. So, if i and j^* are distinct then i and j are distinct. Thus, the right hand side of the relation is zero when $i \neq j$. This is a contradiction, so there are no other columns whose entries are all positive. Therefore, since the "identity" column is easily locatable, we can assume without loss of generality that it is the first column χ_1 in the character table.

Since degree z_i of the i^{th} irreducible character equals $\chi_1^{(i)}$, for $1 \leq i \leq k$, we can now directly read these degrees from the character table.

We can now use the identity $n = \sum_{l=1}^{k} {z_l}^2$ to find n. (The identity is simply equation (2.6) with $i = j = 1$.)

We use the fourth orthogonality relation (equation (2.6), again), to find the values j^* by computing the sum $\sum_{l=1}^{k} \chi_i^{(l)} \chi_{j^*}^{(l)}$ for pairs of rows i and j^* and observing that $j = i$ if and only if the sum is non-zero. At the same time, we can find the values of the h_i's using n and the non-zero sums above.

We now have all of the values on the right hand side of equation (3.1) and thus can use it to evaluate the structure constants c_{rst}.

The above is summarized in the algorithm below.

Algorithm 2: Inversion of Burnside's Character Algorithm

Input:

A character table $\chi_j^{(i)}$ $(1 \leq i, j \leq k)$ for a finite group G.

Output:

The structure constants c_{rst} $(1 \leq r, s, t \leq k)$ for G.

Step 1:

Identify the column in the character table corresponding to the conjugacy class containing the identity element in the group by finding a column with only positive integer entries. Call this column χ_1.

Step 2:

Read the dimensions of the irreducible representation classes z_i from the column located in step 1: $z_i = \chi_1^{(i)}$ for $1 \leq i \leq k$.

Step 3:

Compute the size of the group, $n = \sum_{i=1}^{k} {z_i}^2$.

Step 4:

For each $1 \leq i \leq k$, find the values i^* and h_i by computing sums

$$X_{i,j} = \sum_{l=1}^{k} \chi_i^{(l)} \chi_j^{(l)} \text{ for } 1 \leq j \leq k$$

and setting $i^* = j$ and $h_i = n/X_{i,j}$ for the unique index j such that the sum is nonzero.

Step 5:

Output the values c_{rst} found using equation (3.1):

$$c_{rst} = \frac{h_r h_s}{n} \sum_{m=1}^{k} \frac{\chi_r^{(m)} \chi_s^{(m)} \chi_{t^*}^{(m)}}{z_m}$$

for $1 \leq r, s, t \leq k$. ■

Theorem 4 *c_{rst} from $\chi \in FP$.* ■

Proof: Each of the steps of the above algorithm can be accomplished in polynomial time. Steps 1 and 2 involve simply searching the input and copying part of it. Step 3 is a sum over k values and step 4 involves at most $O(k^2)$ sums over k values. Step 5 involves k^3 summations, each of which can be done with $k-1$ additions, $2(k-1)$ multiplications, and $k-1$ divisions. Thus, we need no more than $O(k^4)$ field operations for the entire algorithm. ■

The inversion process cannot proceed any further. There are finite groups which are non-isomorphic and have the same character table. For example, the fourth dihedral group D_4 and the quaternion group Q of order eight are non-isomorphic and have the same character table.

It is interesting to consider the inversion problem for special classes of groups. There are no known examples of non-isomorphic simple non-abelian groups with the same character table (see [CR81]). It seems possible that Cayley tables could be found from character tables of these groups or some large subclass of these groups. This takes us well beyond the scope of this thesis. We end the discussion on this topic by noting that since the size of

the multiplication table can be superpolynomial in the size of the character table, the standard definitions of efficiency are not directly applicable to the complete inversion problem.

Some comments on the analysis remain. We have counted arithmetic operations rather than Boolean operations. Since our inputs are algebraic numbers, it is not immediately clear that only a polynomial number of Boolean operations are needed. This subject is beyond the scope of this thesis. For now, it will suffice to say that the proof that Burnside's algorithm could be implemented in polynomial time explicitly counted the number of Boolean operations required (see [Ebe89]). Since we are dealing with the same algebraic numbers, our proof carries over. For more information on the complexity of arithmetic for algebraic numbers see [Loo83].

The application of the theorem to Schneider's strategy for computing characters from an incomplete set of structure constants [Sch90] allows the derivation of a lower bound on the number of structure constants that must be used. That is, since the entire set of structure constants can be recovered from the character table, one can only find the character table if one has enough information to construct all of the structure constants. However, it is still not clear how exactly the structure constants depend on one another.

As well, the inversion algorithms provides an efficient reduction from the problem "given a specification of a finite group G (in some form), find the structure constants for G" to the problem "given (the same) specification of G, find the absolutely irreducible character table of G". That is, finding all of the structure constants of a group does not require substantially more resources than finding the group's character table.

Chapter 4

Characters of the Symmetric Group

In this chapter, we examine two problems in the character theory of the symmetric group. The hardness and completeness results (for computing individual characters of the symmetric group and for a generalization of computing coefficients in the decomposition of the outer product of characters of the symmetric group) in this chapter are new. The character and decomposition algorithms are standard parts of the literature but the analysis is new. The algorithm for counting lattice partitions is a straightforward application of Kreweras' Theorem. We are unaware of any previous publication of this algorithm but it seems likely that the algorithm has been known for some time.

For general groups, there is no known way to sensibly and succinctly specify a particular class of absolutely irreducible representations and thus we cannot formulate a good version of the problem of computing individual entries in the character table of a general group. However, for the symmetric groups, we can

sensibly and succinctly specify both conjugacy classes and classes of absolutely irreducible representations. Thus, we can formulate computational problems about individual entries in the character table of the symmetric group.

These problems are quite old. Frobenius gave a formula for the characters and subsequent researchers have used the formula and related results to produce correct algorithms for the problem. Littlewood and Richardson gave a rule for finding the coefficients in the decomposition of the outer product of characters of symmetric groups.

We formulate two numerical versions of the character problem. The first is simply the problem of finding the character of a representation at a conjugacy class. For technical reasons connected with giving a good classification of the complexity of the problem, we need to be able to work with positive numbers. Thus, we define a second version of this problem where we find the sum of the character and a sufficiently large number. We define a decision problem by adding a threshold value to the above and asking if the character is larger than the threshold.

We use a known algorithm to show that the second version of the numerical problem is in $\#P$ and that the decision problem is in PP. We then show that the counting problems are hard for $\#P$ and the decision problem is hard for PP.

We have less success with the outer product problem. Again, we show membership in $\#P$ and PP for number and decision versions of the problem. We were unable to show hardness results for the problem. We do show that a generalization of computing outer products is hard and we identify an interesting class of easy instances of the problem.

4.1 Characters of the Symmetric Group

A variant of the Murnaghan–Nakayama rule for computing characters (following [Keo75]) is presented. There is a strong connection between the Frobenius formula (see, for example, [Ham89]) and the Murnaghan–Nakayama rule. When the algorithm is recursive, it is called the Murnaghan–Nakayama rule. Otherwise, it is an application of the Frobenius formula.

Correctness proofs of various formulations of the algorithm are contained in [Ham89], [dBR61], [Ker91], [JK81], and [Sag91]. Using the correctness of this algorithm, we establish reductions from appropriate formulations of the hard problem **4–PARTITION** (see section 2.3.3) to the problem of finding individual entries in the character table of the symmetric group. Finally, we use the algorithm to show that the character problems can be solved within certain resource constraints. These resource bounds, together with the hardness results, imply completeness results.

The reader may recall (from Definition 3) that the symmetric group S_n is the group of all permutations of n objects and has size $n!$. The conjugacy classes of S_n are directly identifiable with the partitions of n (partitions are described in section 2.2.2). The classes of irreducible representations of S_n (described in section 2.2.3) have a natural one–to–one correspondence with the partitions of n. This is a consequence of the special representation theory of the symmetric group. The description of the correspondence can be found in [Ker91], [JK81], and [Sag91].

We will write $\lambda = [\lambda_1, \lambda_2, \ldots, \lambda_k]$ for a partition of n specifying a class of irreducible representations of S_n and $\mu = [\mu_1, \mu_2, \ldots, \mu_l]$ for a partition of n specifying a conjugacy class in S_n. Further, we adopt as a convention that the entries in the partitions are given in non-increasing order and only positive

entries are included in the specification of a partition.

We now have the capability to ask: 'for the symmetric group S_n, what is the value of the character of an irreducible representation given by λ at a conjugacy class μ?' Throughout this chapter, we will denote this value by $\chi^{\lambda}(\mu)$.

We formally define several problems related to the above question. The first problem is a restatement of the question above. The inputs for each of these problems are essentially the same.

Number Problem 8: #CSG

Individual Character of the Symmetric Group

Input:

n: expressed in unary,

λ: a partition of n specifying a class of equivalent irreducible representations of S_n,

μ: a partition of n specifying a conjugacy class in S_n.

Output:

The value of the character of an irreducible representation in λ at the conjugacy class μ: $\chi^{\lambda}(\mu)$. ■

Since the value $\chi^{\lambda}(\mu)$ can be negative and we will be concerned with computations on a counting Turing machine, we give a definition of the same problem offset so that all values are positive and thus not trivially uncomputable in this model.

Number Problem 9: #CSG+

Sum of an Individual Character of the Symmetric Group and n^n

Input:

n: expressed in unary,

λ: a partition of n specifying a class of equivalent irreducible representations of S_n,

μ: a partition of n specifying a conjugacy class in S_n.

Output:

The value of sum of character of an irreducible representation in λ at the conjugacy class μ and n^n: $\chi^\lambda(\mu) + n^n$ ■

This variation of the problem allows us to give a good characterization of the complexity of the problem.

Decision Problem 10: TCSG

Threshold for Individual Characters of the Symmetric Group

Input:

n: expressed in unary,

λ: a partition of n specifying a class of equivalent irreducible representations of S_n,

μ: a partition of n specifying a conjugacy class in S_n.

x: an integer threshold value expressed in binary.

Question:

Is the value of the character of an irreducible representation in λ at the conjugacy class μ greater than or equal to x?

That is, is $\chi^\lambda(\mu) \geq x$? ■

All of the above problems include an input of n in unary. This means that the way that we express the numbers in the partitions is inconsequential. We

may express them in binary or unary without a significant change in the input size. Furthermore, our hardness results hold with this padding of the input and our membership results are not affected. Thus, the padding allows us to state the results in the strongest possible manner.

4.1.1 A Graphical Version of the Murnaghan–Nakayama Rule

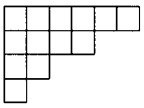
The graphical version of the algorithm inspired the polynomial transformation given in section 4.1.3. Also, it is useful for visualizing the steps of the hardness proof.

The *extended diagram* for a partition $\lambda = [\lambda_1, \lambda_2, \ldots, \lambda_k]$ can be formally defined as the set of points $\{(i,j) \in \mathbb{Z}^2$ such that $1 \leq j \leq \lambda_i + k - i\}$.

We draw the diagrams using the same indexing conventions as are used with matrices. The first coordinate i designates the row and increases from top to bottom. The second coordinate is for columns and increases from left to right. Thus, the extended diagram corresponding to $[3,2,1,1]$ is the set of points:

$$\{(1,1),(1,2),(1,3),(1,4),(1,5),(1,6),(2,1),$$
$$(2,2),(2,3),(2,4),(3,1),(3,2),(4,1)\}$$

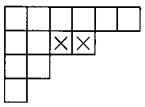
and is drawn:



The first operation on extended diagrams that we will consider is the removal of a cycle. We remove a cycle of length l from some row m of an extended diagram by colouring in the rightmost boxes of the row rather than

erasing the boxes from the row. The advantage of this is that we can see what operation we are performing on a single diagram.

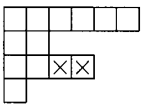
Thus, the extended diagram for $[3,2,1,1]$ with a 2-cycle removed from the second row is:



(4.1)

Further cycles can be removed from that row by colouring the appropriate number of the rightmost uncoloured boxes. Whenever we remove a cycle from an extended diagram, all of the boxes that we colour for that cycle must be removed from only one row.

The second operation is row transposition. This can be done to extended diagrams that have had cycles removed. Transposing rows i_1 and i_2 can be thought of as a simple exercise with scissors. Simply cut both rows from the diagram and replace the rows in the diagram in opposite order. In terms of the formal definition, this means replacing all occurrences of i_1 with i_2 and i_2 with i_1 in the first (row) position of the elements of the extended diagram. Thus, transposing the second and third rows of the diagram shown in 4.1 results in:

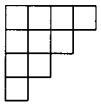


(4.2)

A sequence of transpositions can be viewed as a permutation of the rows of a diagram. This allows us to talk about the sign of a sequence of transpositions.

Bearing in mind that colouring boxes is shorthand for erasing boxes, we say that two extended diagrams are equivalent if their uncoloured boxes are in the same positions. From this point of view, the diagrams shown in 4.1 and 4.2 are equivalent.

There is one special diagram that we need. The *k-staircase* is the diagram consisting of the points $(i, j) \in \mathbb{Z}^2$ such that $1 \leq i \leq k$ and $1 \leq j \leq k - i$. The 5-staircase looks like this.



The bottom row of the staircase is empty. That is, the widths of the rows are $(4, 3, 2, 1, 0)$.

Now that we have established the notation, we proceed with the algorithm.

Algorithm 3: The Graphical Murnaghan-Nakayama Rule

Input:

n: expressed in unary,

λ: a partition of n specifying a class of equivalent irreducible representations of S_n,

μ: a partition of n specifying a conjugacy class in S_n.

Output:

The value of the character of an irreducible representation in λ at the conjugacy class μ:

$$\chi^{\lambda}(\mu).$$

Step 1:

$counter \leftarrow 0$

for $i \leftarrow 1$ **to** m **do**

$z_i \leftarrow 1$

endfor

$j \leftarrow m$

Step 2:

while $(j \neq 0)$ **do**

continue $\leftarrow$ **True**

$h \leftarrow$ extended diagram(λ)

$i \leftarrow 1$

while ((*continue*) **and** $(i \leq m)$) **do**

remove μ_i from row z_i of the extended diagram h

(to get a new h)

if ((row i of h is negative) **or**

(h has two equal rows) **then**

continue $\leftarrow$ **False**

$j \leftarrow i$

endif

increment i

endwhile

if (*continue*) **then**

if (h is an even row permutation of the k-staircase) **then**

increment *counter*

else

decrement *counter*

endif

$j \leftarrow m$

endif

increment z_j

while $((z_j > k)$ **and** $(j \geq 1))$ **do**

decrement j

endwhile

increment z_j

for $i \leftarrow j+1$ **to** m **do**

$z_i \leftarrow 1$

endfor

endwhile

Step 3:

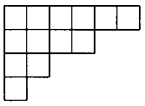
Output *counter* ■

Note that we can rearrange the initial order of the μ_i's without affecting correctness although this can affect the efficiency of the algorithm.

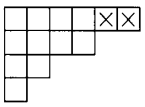
We now present an example of using the graphical Murnaghan-Nakayama rule.

Example 4.1.12: Graphical Evaluation of $\chi^{[3,2,1,1]}([2,2,2,1])$

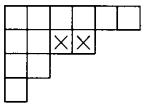
We let $\lambda = [3,2,1,1]$ and $\mu = [2,2,2,1]$. We have already determined h from λ to be the diagram



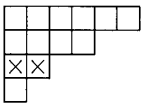
We start with $z_1 = 1$, $z_2 = 1$, $z_3 = 1$, and $z_4 = 1$. Thus, we remove the first cycle of μ from the first row of h to get



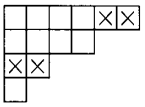
Since the first and second rows in the diagram are now the same size, we continue with $z_1 = 2$, $z_2 = 1$, $z_3 = 1$, and $z_4 = 1$. Removing the first cycle of μ from the second row of h gives



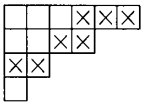
This time, the second and third rows in the diagram are the same size. We continue with $z_1 = 3$, $z_2 = 1$, $z_3 = 1$, and $z_4 = 1$. Removing the first cycle of μ from the third row of h gives



This diagram does not violate the constraints given in step 2 of the algorithm, so, since $z_2 = 1$, we remove the second cycle of μ from the first row of the diagram to get



Again the first and second rows are the same. This leads to considering $z_1 = 3$, $z_2 = 2$, $z_3 = 1$, and $z_4 = 1$. Removing the constituents of μ from h according to these z's gives the diagram



The permutation taking the diagram to a staircase is odd so we decrement the counter to -1.

Having seen the details of each step, we proceed through the rest of the execution of the algorithm more quickly.

Continuing with $z_1 = 3$, $z_2 = 2$, $z_3 = 1$, and $z_4 = 2$ gives a diagram in which the second and fourth rows have the same length. Thus, without changing the counter, we continue with $z_1 = 3$, $z_2 = 2$, $z_3 = 1$, and $z_4 = 3$. The third row in the diagram for this case has a negative width so we continue with $z_1 = 3$, $z_2 = 2$, $z_3 = 1$, and $z_4 = 4$. In this case, the third and fourth rows are the same so we jump to $z_1 = 3$, $z_2 = 2$, $z_3 = 2$, and $z_4 = 1$. After removing the third cycle of μ from h, we see that the second and third rows have the same size. Thus, we continue with $z_1 = 3$, $z_2 = 2$, $z_3 = 3$, and $z_4 = 1$. Removing the third cycle of μ from the third row yields a row with negative size and so we continue with $z_1 = 3$, $z_2 = 2$, $z_3 = 4$, and $z_4 = 1$. Again we get a negative row upon removing the third cycle so we set $z_1 = 3$, $z_2 = 3$, $z_3 = 1$, and $z_4 = 1$. This yields a negative row upon removing the second cycle, as does $z_1 = 3$, $z_2 = 4$, $z_3 = 1$, and $z_4 = 1$. This takes us to $z_1 = 4$, $z_2 = 1$, $z_3 = 1$, and $z_4 = 1$ which gives a negative fourth row on removing the first cycle of μ. At this point, the algorithm terminates, returning a value of -1 since we have not incremented the counter and we have decremented it only once. ■

4.1.2 A Concise Version of the Murnaghan-Nakayama Rule

We translate the graphical algorithm into a form more amenable to symbolic manipulation.

We use the following notation. The extended diagram for λ is denoted by $h = [h_1, h_2, \ldots, h_k]$ where the components h_i, called the *principal hooks*, are given by:

$$h_i = \lambda_i + k - i \quad (1 \leq i \leq k) \tag{4.3}$$

This is equivalent to the definition of extended diagram earlier but is more succinct. The extended diagram h completely determines the class of irreducible representations λ and each can easily be found from the other in deterministic polynomial time. A *hook structure* is an extended diagram that may have had cycles removed from it.

We use $\biguplus$ to denote multiset union (adding multiplicities). We use angle brackets $\langle\rangle$ to denote multisets. We define the action of a cycle μ_i on a hook structure $h = [h_1, h_2, \ldots h_k]$ to be the multiset

$$\mu_i\langle[h_1, h_2, \ldots, h_k]\rangle = \biguplus_{m=1}^{k} \langle[h_1, h_2, \ldots, h_m - \mu_i, \ldots, h_k]\rangle$$

and the action of a cycle on a multiset of hook structures to be the multiset union of the action of the cycle on each of the members of the multiset.

$$\mu_i\langle h^a, \ldots, h^b\rangle = \mu_i\langle h^a\rangle \uplus \cdots \uplus \mu_i\langle h^b\rangle$$

Note that we are using superscripts to differentiate between hook structures, and not to indicate exponentiation.

The value of a hook structure $h = [h_1, h_2, \ldots, h_k]$ is defined to be

$$|h| = \begin{cases} 0 & \text{if } \exists\, i \text{ such that } h_i < 0 \\ 0 & \text{if } \exists\, i, j\ (i \neq j) \text{ such that } h_i = h_j \\ 1 & \text{if an even permutation sorts } h_1, h_2, \ldots, h_k \text{ into descending order} \\ -1 & \text{if an odd permutaion sorts } h_1, h_2, \ldots, h_k \text{ into descending order} \end{cases}$$

The value of a multiset of hook structures is the sum of the values of the individual hook structures in the multiset.

$$||\langle h^1, h^2, \ldots, h^l \rangle|| = \sum_{i=1}^{l} |h^i|.$$

With this notation in place, we can see that the value of the character $\chi^\lambda(\mu)$ given by the graphical algorithm is the value of the multiset formed by allowing each of the cycles of μ to act on the extended diagram h formed from λ.

The algorithm is stated below.

Algorithm 4: The Concise Murnaghan–Nakayama Rule

Input:

n: expressed in unary,

λ: a partition of n specifying a class of equivalent irreducible representations of S_n,

μ: a partition of n specifying a conjugacy class in S_n.

Output:

The value of the character of an irreducible representation in λ at the conjugacy class μ: $\chi^\lambda(\mu)$.

Step 1:

Determine the extended diagram h for λ

Step 2:

Evaluate $\chi^\lambda(\mu) = ||\mu_1(\mu_2 \ldots (\mu_m h) \ldots))||$

and return the value. ■

Equivalent versions of the above algorithm are shown to be correct in [Ham89], [dBR61], [Ker91], [JK81], and [Sag91]. The notation and exact formulation of the algorithm is different in each of the sources. With the

appropriate translation and possibly some rearrangement of the order of the steps, each of them yields the following theorem.

Theorem 5 *The Concise Murnaghan–Nakayama Rule on input n, λ, and μ returns the value of the character of an absolutely irreducible representation of S_n, in λ at μ. That is, the Concise Murnaghan–Nakayama Rule correctly finds $\chi^{\lambda}(\mu)$.* ■

4.1.3 A Polynomial Time Transformation

Consider an instance (A, B, s) of 4–PARTITION (see Section 2.3.3) where $A = \{a_1, a_2, \ldots a_{4m}\}$, $s : A \to \mathbb{Z}$, and $B \in \mathbb{Z}$ is polynomially bounded in $m = |A|$. Without loss of generality, suppose

$$s(a_1) \geq s(a_2) \geq \ldots \geq s(a_{4m}).$$

We construct an instance (n, λ, μ) of #CSG from (A, B, s). Let

$$\begin{aligned} n &= m^2 \cdot B, \\ \lambda &= [(m \cdot B)^m], \\ \text{and } \mu &= [m \cdot s(a_1), \ldots, m \cdot s(a_{4m})]. \end{aligned} \tag{4.4}$$

This transformation can be done in polynomial time whenever B is bounded by a specific polynomial $p(m)$.

Intuitively, we use the rows of the extended diagram for λ to hold 4–sets and the partition μ to encode the sizes in the 4–PARTITION instance. The sizes are scaled up so that any valid 4–set S_i^4 (with $\sum_{a \in S_i^4} s(a) = B$) will fit into a row. Furthermore, with the row i filled by a valid 4-set, there will be $m - i$ boxes left unfilled in that row. Thus, if all rows are filled with valid 4-sets, then we are left with a staircase which is counted as one. Any other

way to fill the diagram with the cycles of μ will result in at least one row taking elements (in the instance of 4–PARTITION) whose sizes sum to more than B and thus, whose sizes (in the instance of #CSG) sum to more than $m \cdot (B+1)$. Since none of the rows in the diagram for λ are that big, the resulting hook structure has a negative component. Thus, the resulting hook structure cannot be a permuted staircase and so it is not counted.

Lemma 1 *If there are N valid 4–partitions of (A, B, s) then the value of the character $\chi^{\lambda}(\mu)$ of S_n at λ and μ given by equation (4.4) is $N \cdot m!$* ■

Proof: The extended diagram h resulting from λ is given below.

$$h = [mB + m - 1, mB + m - 2, \ldots, mB]$$

That is, the components h_i of h are given by

$$h_i = mB + m - i \qquad (1 \leq i \leq m).$$

For any subset $S \subseteq A$ let $D(S)$ be defined by

$$D(S) = B - \sum_{a \in S} s(a)$$

If we remove only the cycles μ_j such that $a_j \in S \subseteq A$ from the i^{th} row of h we get a resulting extended diagram where the value of the i^{th} component, which we denote by $H(S, i)$, is

$$\begin{aligned} H(S,i) &= h_i - \textstyle\sum_{a \in S} m s(a) \\ &= mB + m - i + m(B - \textstyle\sum_{a \in S} s(a) - B) \\ &= mB + m - i + mD(S) - mB \\ &= m - i + mD(S). \end{aligned}$$

If $D(S) = 0$, then $H(S,i) = m - i$. If $D(S) < 0$, then

$$\begin{aligned} H(S,i) &= m - i + mD(S) \\ &\leq m - i - m < 0. \end{aligned}$$

Thus, if $S_1, S_2, \ldots, S_m$ is a valid 4–partition of A then $D(S_i) = 0$ for $1 \leq i \leq m$ and so we can remove the μ_j's corresponding to the S_i's from h, by removing μ_j from h_i whenever $\mu_j \in S_i$, to get $[H(S,1), \ldots, H(S,m)] = [m-1, m-2, \ldots, 0]$, which contributes one to the value of the character.

Now, from an expression $S_1, S_2, \ldots, S_m$ for a valid 4–partition, any permutation π of m objects applied to the subscript in the expression gives another expression $S_{\pi(1)}, S_{\pi(2)}, \ldots, S_{\pi(m)}$ for the same 4–partition. Since there are $m!$ such expressions for each such 4–partition and each of these expressions contributes one to the value of the character, if there are N valid 4–partitions of A, we get a contribution of $N \cdot m!$ to the character.

We now prove that there are no further contributions to the character. Given any sequence $z = (z_1, \ldots z_k) \in \{1, 2, \ldots m\}^k$, we can reverse the process described above by putting a_i into S_{z_i} to get a partition $S_1, S_2, \ldots, S_m$ of A. If the resulting partition is a valid partition then we have a contribution of one as described above. Now, consider the case where the resulting partition is not a valid partition of A. In such a partition, there must be an S_i such that the sum of the sizes of the elements is greater than B. If not, the partition would be valid. Let S_i be such a set and consider what happens to the i^{th} row when we remove the μ_j's from h following z. Since

$$\sum_{a \in S_i} s(a) > B$$

we have $D(S_i) < 0$. This, in turn, implies that $H(S,i) < 0$ and so the contribution to the value of the character is zero. ■

4.1.4 A Probabilistic Polynomial Time Version of the Murnaghan–Nakayama Rule

Recall that a probabilistic Turing machine is a non–deterministic Turing machine that accepts a string if strictly more than half of the possible computation paths on that string are accepting computations. For a more detailed description see [Joh90].

Informally, we use the phrase 'generate n accepting computations' to indicate a process where we allow the computation tree to branch $\lceil \log_2 n \rceil$ times by writing $\lceil \log_2 n \rceil$ 0's or 1's to the tape. For all of the sequences of 0's and 1's that correspond to a number less than n, the machine jumps immediately to an accepting state. For those greater than or equal to n, the machine writes another 0 or 1 and rejects if it wrote a 0 and accepts if it wrote a 1. After this process, there will be n more accepting computations than rejecting computations and, since the balance of the number of accepting computations versus the number of rejecting computations is all that matters for overall acceptance in this model of computation, we can ignore the matching acceptances and rejections produced for numbers greater than or equal to n. We use the phrase 'generate n accepting computations and continue' to indicate that the process should be prefaced by a single branching of the computation tree where one branch generates n accepting computations and the other continues. Similar definitions apply to generating rejections. 'Nondeterministically generate' means produce a branch in the computation tree for each of the things specified.

We describe a probabilistic polynomial time version of the Murnaghan-Nakayama rule.

Let $h = [h_1, \ldots, h_k]$ be an extended diagram. Let $\mu = [\mu_1, \ldots, \mu_l]$ be a

partition (specifying a conjugacy class) and let $z \in \{1, 2, \ldots, k\}^l$. We define the symbol $(h - \mu)_z$ as the value of the extended diagram resulting from removing μ from h according to z. That is, let

$$h^{(0)} = h = [h_1, \ldots, h_k],$$

and

$$h^{(j)} = [h_1^{(j-1)}, \ldots, h_{z_j}^{(j-1)} - \mu_j, \ldots, h_k^{(j-1)}].$$

The value of $(h - \mu)_z$ is the value of the resulting extended diagram $h^{(l)}$.

Algorithm 5: PP Algorithm for TCSG

Input:

n: expressed in unary,

λ: a partition of n specifying a class of equivalent irreducible representations of S_n,

μ: a partition of n specifying a conjugacy class in S_n.

x: an integer threshold value expressed in binary.

Question:

Is the value of the character of an irreducible representation in λ at the conjugacy class μ greater than or equal to x?
That is, is $\chi^{\lambda}(\mu) \geq x$?

Step 1:

if $(x > 0)$ **then**

generate $2x - 1$ rejecting computations and continue

else

generate $2|x| + 1$ accepting computations and continue

endif

Step 2:

Non-deterministically generate a $z \in \{1, 2, \ldots, k\}^l$

(ie. for each z do):

Step 2.1:

if $(h - \mu)_z = 0$ **then**

generate an accepting computation

and a rejecting computation

else if $((h - \mu)_z = 1)$ **then**

generate two accepting computations

else

$\{(h - \mu)_z = -1\}$

generate two rejecting computations

endif ■

We show below that correctness of this algorithm is implied by that of the Concise Murnaghan-Nakayama rule.

Let $P^{\lambda,\mu}(y)$ be the number of ways that μ can be removed from the principle hook structure h for representation class λ so that the resulting hook structure h' evaluates to y. That is, $P^{\lambda,\mu}(y) = |\{z : (h - \mu)_z = y\}|$.

Now from the Concise Murnaghan-Nakayama rule, we see that

$$\chi^{\lambda}(\mu) = P^{\lambda,\mu}(1) - P^{\lambda,\mu}(-1). \tag{4.5}$$

So $\chi^{\lambda}(\mu) \geq x$ if and only if

$$P^{\lambda,\mu}(1) - P^{\lambda,\mu}(-1) \geq x;$$

that is,

$$P^{\lambda,\mu}(1) - P^{\lambda,\mu}(-1) - x \geq 0.$$

We now examine the possible outputs from the algorithm in two cases.
Case 1: If $x > 0$, the algorithm gives:

From step 1: $2x - 1$ rejections and

From step 2:

$P^{\lambda,\mu}(0)$ acceptances,

$P^{\lambda,\mu}(0)$ rejections,

$2P^{\lambda,\mu}(1)$ acceptances, and

$2P^{\lambda,\mu}(-1)$ rejections

so the total number of accepting computations is $P^{\lambda,\mu}(0) + 2P^{\lambda,\mu}(1)$ and the total number of rejecting computations is $2x - 1 + P^{\lambda,\mu}(0) + 2P^{\lambda,\mu}(-1)$.

The algorithm accepts the input if and only if the number of acceptances is larger than the number of rejections. If $x > 0$, this is true if and only if

$$P^{\lambda,\mu}(0) + 2P^{\lambda,\mu}(1) > 2x - 1 + P^{\lambda,\mu}(0) + 2P^{\lambda,\mu}(-1)$$

Consolidating terms involving $P^{\lambda,\mu}$ yields

$$2P^{\lambda,\mu}(1) - 2P^{\lambda,\mu}(-1) > 2x - 1$$

and from equation (4.5), we have

$$2\chi^{\lambda}(\mu) > 2x - 1.$$

Since $\chi^{\lambda}(\mu)$ and x are integers, this is equivalent to the condition

$$2\chi^{\lambda}(\mu) \geq 2x$$

or

$$\chi^{\lambda}(\mu) \geq x.$$

Similarly, the algorithm rejects the input if and only if the number of rejections is at least as big as the number of acceptances. Again, when $x > 0$ we get

$$P^{\lambda,\mu}(0) + 2P^{\lambda,\mu}(1) < 2x - 1 + P^{\lambda,\mu}(0) + 2P^{\lambda,\mu}(-1)$$

and again consolidating terms and applying equation (4.5) gives

$$\chi^{\lambda}(\mu) < x.$$

Thus, when $x > 0$ the algorithm accepts if $\chi^{\lambda}(\mu) \geq x$ and it rejects otherwise.
Case 2: If $x \leq 0$, then by a similar calculation to the above, the number of accepting computation paths is

$$2|x| + 1 + P^{\lambda,\mu}(0) + 2P^{\lambda,\mu}(1)$$

and the number of rejecting computation paths is

$$P^{\lambda,\mu}(0) + 2P^{\lambda,\mu}(-1).$$

Now if $\chi^{\lambda}(\mu) \geq x$, then

$$2\chi^{\lambda}(\mu) \geq 2x = -2|x|.$$

Applying equation (4.5) and subtracting 1 from the right hand side to get an inequality, we have

$$2P^{\lambda,\mu}(1) - 2P^{\lambda,\mu}(-1) > -2|x| - 1.$$

Adding $P^{\lambda,\mu}(0)$ to both sides and rearranging terms gives

$$2P^{\lambda,\mu}(1) + P^{\lambda,\mu}(0) + 2|x| + 1 > 2P^{\lambda,\mu}(-1) + P^{\lambda,\mu}(0)$$

which implies that the number of accepting computation paths is larger than the number of rejecting computation paths. Similarly, if $\chi^{\lambda}(\mu) < x$, then

$$2\chi^{\lambda}(\mu) < 2x - 1.$$

Again adding $P^{\lambda,\mu}(0)$ to both sides and rearranging terms gives

$$2P^{\lambda,\mu}(1) + P^{\lambda,\mu}(0) + 2|x| + 1 < 2P^{\lambda,\mu}(-1) + P^{\lambda,\mu}(0)$$

which implies the number of accepting computation paths is smaller than the number of rejection computation paths.

Lemma 2 *TCSG* $\in$ *PP.* ■

Proof: As we have just seen, the algorithm is correct. Furthermore, each branch of the algorithm is of polynomial length. The paths terminating in step 1 are of length $O(\log(x))$ which is clearly bounded by a polynomial in the input size. The paths terminating in step 2 are of length $0(\log(n) \cdot k \cdot l)$ which is polynomial in the input size even when the input is not written in unary. ■

4.1.5 The Murnaghan-Nakayama Rule on a Counting Turing Machine

Recall that a counting Turing machine (CTM) is structurally the same as a probabilistic machine except that the value returned by a CTM is the number of accepting computations, rather than just a 'yes' or 'no' depending on whether there are more accepting computations than rejecting computations.

The class #P is the set of all problems solvable by a CTM in polynomial time. Since we do not care at all about the number of rejecting computations, when we say 'generate n accepting computations' we mean generate exactly n accepting computations. This is easily done by nondeterministically generating $\lceil \log_2 n \rceil$ 0's and 1's and accepting only if the resulting number (in binary) is less than n. For a more detailed discussion of counting Turing machines see [Joh90].

Since a CTM cannot return a negative number, instead of directly evaluating the character, we consider the problem of evaluation of the sum of the value of a character and an easily computable large positive number. This allows us to give an exact classification of the problem. From this, we can easily compute the desired character value and thus we have a good classification of computing the character as well. We choose the large number to be n^n since it is both easily computable and is always at least as large as the absolute value of the character of S_n.

We now present a CTM version of the Murnaghan-Nakayama Rule.

Algorithm 6: CTM Version of the Murnaghan-Nakayama Rule

Input:

n: expressed in unary,

λ: a partition of n specifying a class of equivalent irreducible representations of S_n,

μ: a partition of n specifying a conjugacy class in S_n.

Output:

The value of the character of an irreducible representation in λ at the conjugacy class μ, plus n^n:

$$\chi^{\lambda}(\mu) + n^n.$$

Step 1:

Step 1.1:

Find k^l and n^n.

Step 1.2:

Generate $n^n - k^l$ accepting computations and continue.

Step 2:

for each $z \in \{1, \ldots, k\}^l$ **do**

if $(h - \mu)_z = 1$ **then**

generate two accepting computations

else if $((h - \mu)_z = -1)$ **then**

reject

else

$\{(h - \mu)_z = 0\}$

accept

endif

endfor ■

The number of accepting computations is

$$\begin{aligned} 2P^{\lambda,\mu}(1) + P^{\lambda,\mu}(0) + n^n + k^l &= P^{\lambda,\mu}(1) + P^{\lambda,\mu}(0) + P^{\lambda,\mu}(-1) \\ &+ P^{\lambda,\mu}(1) - P^{\lambda,\mu}(-1) + n^n - k^l \end{aligned}$$

and, since

$$P^{\lambda,\mu}(1) + P^{\lambda,\mu}(0) + P^{\lambda,\mu}(-1) = k^l$$

and

$$\chi_{\mu}^{\lambda} = P^{\lambda,\mu}(1) - P^{\lambda,\mu}(-1),$$

the number of accepting computations is

$$k^l + \chi^{\lambda}(\mu) + n^n - k^l = \chi^{\lambda}(\mu) + n^n$$

We observe that, since $k, l \leq n$, we have that $n^n \geq k^l$. Thus step 1 above can always be done since the value $n^n - k^l$ is always non-negative.

Lemma 3 *$\#CSG+ \in \#P$.* ■

Proof: Counting operations on hook structures as unit cost, each branch of the algorithm takes $O(n \cdot k \cdot l)$ time and the n^n branches can be generated in $O(n \log(n))$ time. Thus the total running time is $O(n \log(n) + nkl)$. ■

4.1.6 Completeness Theorems

We are now in a position to prove the main theorems of this section.

Theorem 6 ***TCSG** is PP–complete.* ■

Proof: We recall from section 2.3.3 that the problem **T–4–PARTITION** is PP–complete. The transformation given in section 4.1.3 was shown to be parsimonious so **TCSG** is PP–hard. Combining this with lemma 2 immediately implies the result. ■

Theorem 7 ***#CSG** is #P–hard.* ■

Proof: We recall for section 2.3.3 that the problem **#–4–PARTITION** is #P–complete. The transformation given in section 4.1.3 was shown to be parsimonious so **#CSG** is #P–hard. ■

Theorem 8 *#CSG+ is #P-complete.* ■

Proof: Theorem 7 and the fact that n^n is easily computable (in binary from unary input n) implies #P–hardness. Lemma 3 shows membership in #P. Combining these implies the result. ■

Theorem 8 shows that **#CSG** fails to be #P–complete only by virtue of having some negative answers.

4.2 Outer Products, Schur Functions, and The Littlewood–Richardson Rule

In this section, we describe problem of decomposition of outer products of characters of the symmetric group. There is a well known connection between this problem and computing coefficients of Schur polynomials. Namely, both are solved by the Littlewood–Richardson rule. We analyze the Littlewood–Richardson rule as it stands and use it to define several related problems to get a better picture of the complexity of the above problems. Schur polynomials and outer products of characters are discussed at length in [Mac79], [Sag91], [JK81] and [Ker91].

Other work has been done in this problem. The Littlewood–Richardson rule is modified to produce another combinatorial algorithm in [RW84]. [Eğe82] documents an implementation of the Littlewood–Richardson rule with pruning. [ER85] contains a table of Littlewood–Richardson coefficients for two special cases up to $n = 30$.

4.2.1 Outer Products of Characters of the Symmetric Group

Given two matrix representations over the same field, T_1 of S_n and T_2 of S_m of dimensions $\hat{n}$ and $\hat{m}$ respectively, we can construct the tensor product of these representations $T_1 \otimes T_2$ of $S_n \oplus S_m$ for any $g_1 \in S_n$ and any $g_2 \in S_m$. The dimension of this representation is $\hat{n} \times \hat{m}$.

Recalling the definition of the direct sum of groups, we see that $S_n \oplus S_m$ is a subgroup of S_{n+m}. The matrix representation $T_1 \otimes T_2$ of $S_n \oplus S_m$ can be extended to the whole of S_{n+m} — in particular, to the induced representation (defined, for example in [Ser77] and [FH62] and mentioned in Section 2.2.3). The resulting matrix representation $T' = (T_1 \otimes T_2) \uparrow S_{n+m}$ is called the *outer product* of T_1 and T_2. We shorten the notation to $T' = T_1 \diamond T_2$ so that we can more easily generalize the above notation to characters.

This gives us a well defined operation on the characters of the symmetric groups since if T_1 and T_1' are similar representations and T_2 and T_2' are similar representations then $T_1 \diamond T_2$ and $T_1' \diamond T_2'$ are similar. Using n and m and the superscripts (1) and (2) rather than subscripts, we write $\chi^{(1)} \diamond \chi^{(2)}$ for the character of the (reducible) representation $T' = T_1 \diamond T_2$ of S_{n+m}. Since T' may be reducible, we can decompose T' into its irreducible constituents. Thus we write $\chi^{(1)} \diamond \chi^{(2)} = \sum_{\lambda \vdash (n+m)} c_\lambda \chi^\lambda$ where the c_λ's are to be determined.

We now have the notation necessary to define the problem of computing the coefficients of the irreducible constituents in the outer product.

Number Problem 11: DecOutSym

Decomposition of Outer Products of the Symmetric Group

Input:

Integers $n, m > 0$ (expressed in unary).

Partitions μ, γ, and λ of n, m and $n+m$ respectively,

specifying absolutely irreducible characters of S_n, S_m,

and S_{n+m} respectively.

Output:

The coefficient c_λ in the decomposition

$\chi^{(\mu)} \diamond \chi^{(\gamma)} = \sum_{\phi \vdash n+m} c_\phi \chi^{(\phi)}$. ■

4.2.2 The Littlewood–Richardson Rule

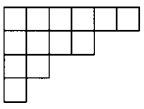
The *standard diagram* for a partition

$$\lambda = (\lambda_1, \lambda_2, \ldots, \lambda_l)$$

is the set of points

$$\{(x, y) : 1 \leq x \leq \lambda_y\}.$$

The standard diagram for $(6, 4, 2, 1)$ is shown below.



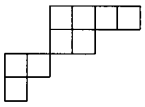
(4.6)

For visually obvious reasons, we will often call the points boxes.

Given two partitions $\lambda \vdash n+m$ and $\gamma \vdash m$, we make the diagram $\lambda \backslash \gamma$ by erasing the standard diagram for γ from the upper left corner of the standard diagram for λ. Thus, to make $(6, 4, 2, 1) \backslash (2, 2)$ we remove the diagram for $(2, 2)$

from the diagram for $(6, 4, 2, 1)$ shown in (4.6) above to get the diagram shown

below.



This operation is only well defined, if for all applicable i, $\lambda_i \geq \gamma_i$.

The Littlewood–Richardson rule can now be stated. Given three partitions $\lambda \vdash n+m$, $\mu \vdash n$ and $\gamma \vdash m$, we make a diagram of $\lambda \backslash \gamma$ and count the ways that we can fill the boxes with symbols directly identified with positive integers according to the following rules.

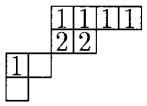
1. One and only one integer is written in each box.
2. If $\mu = (\mu_1, \ldots, \mu_k)$, then exactly μ_i boxes contain i.
3. The symbols are entered into the boxes in numerical order. That is, we start by adding 1's and continues such that all symbols i are entered before we add any of symbol $i+1$.
4. No symbol is added directly to the right of an empty box.
5. No symbol is added directly below an empty box.
6. No symbol is added to a row that is above a row already containing that symbol.
7. No two boxes in the same column contain the same integer.
8. The sequence of integers obtained by reading each row from *right to left* and reading the rows from top to bottom is a lattice partition (see section 2.1.3).

If it is impossible to make the diagram $\lambda\backslash\gamma$ then there are no ways to fill the diagram.

Example 4.2.13: Using the Littlewood–Richardson Rule

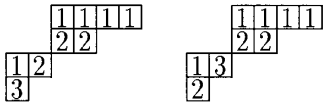
Let us consider the partitions $\lambda = (6,4,2,1)$ and $\gamma = (2,2)$ as above, and let $\mu = (5,3,1)$. We must fill the diagram $\lambda\backslash\gamma$ with five 1's, three 2's and one 3. We observe that the first row of $\lambda\backslash\gamma$ must be filled entirely with 1's, otherwise we would violate either condition 4 or condition 8. The remaining 1 cannot be placed in the second row lest we violate condition 7. It can be placed in the first box of the third row, but not in the second box of the third row by condition 4, and not in the fourth row by condition 5.

The second row must be completely filled with 2's for the same reasons that forced us to fill the first row with 1's. At this point, we have the diagram:



and we have one 2 and one 3 left to place.

The remaining 2 can be placed in either of the remaining open boxes. The 3 must be placed in the other box. Both alternatives are shown below.



Thus, there are two ways that the diagram $\lambda\backslash\gamma$ can be filled in accordance with μ. ■

Since we have not been able to determine the complexity of the the problems solved by the Littlewood–Richardson rule, we do not present a more formal version of the rule.

The fact that the Littlewood–Richardson Rule solves **DecOutSym** is proved, among other places, in [Sag91].

4.2.3 Analysis of the Littlewood–Richardson Rule

There are a number of pruning techniques which allow one to avoid filling the boxes in all possible ways and then checking conditions 1 through 8. However, since the result of the algorithm can be superpolynomial in the input size (see section 4.2.5) and since the algorithm generates every valid placement of integers in $\lambda\backslash\gamma$, so that its running time is at least linear in the value it returns as output, even with perfect pruning, the algorithm still has superpolynomial time complexity. Still, the above algorithm does allow us to observe the following.

Theorem 9 ***DecOutSym*** $\in \#P$. ■

Proof: Each of conditions 1 through 8 above can easily be checked by a Turing machine in polynomial time. Thus, a counting Turing machine, which generates all functions from the set of allowable symbols to the set of boxes in the standard diagram and then accepts only if the placement of symbols satisfies the conditions, solves **DecOutSym**. Further, each branch takes only polynomial time. ■

4.2.4 The Complexity of Associated Problems

The description of the Littlewood–Richardson Rule is only dependent on λ and γ being partitions for determining the diagram $\lambda\backslash\gamma$. The conditions given above can be used on a diagram even if it is not obtainable as a difference of standard diagrams. We define a problem based on the above.

Number Problem 12: L–R/GenDiag

Littlewood–Richardson Problem on Generalized Diagrams

Input:

Two vectors: $X_m, Y_m, \in \mathbb{N}^m$ with components

x_i, y_i such that $x_i \leq y_i$ for $1 \leq i \leq m$.

These specify the left and right boundaries of the diagram.

One vector: $Z_n \in (\mathbb{Z}^+)^n$ with components z_j such

that $z_j \geq z_{j+1}$ for $1 \leq j \leq n-1$ and such that

$\sum_{i=1}^{m}(y_i - x_i) = \sum_{j=1}^{n} z_j$.

This specifies the number of each symbol used to fill the diagram.

Output:

The number of ways that the diagram specified by X_m and Y_m

can be filled with symbols from Z satisfying conditions 1

through 8 in the Littlewood–Richardson Rule. ■

If the components of X_m and Y_m, when listed by their order in the vectors, are in descending order, then the diagram is a difference of standard diagrams. In this case, X_m and Y_m correspond to γ and λ (respectively) in the definition of **DecSymOut**. Z corresponds to μ.

Unfortunately, this new problem is an extreme generalization of the Littlewood–Richardson problem so we are unable to draw any strong conclusions about the

Littlewood–Richardson problem from an analysis of the new problem. However, we are able to give a precise classification of its complexity. The result indicates that if there is a polynomial time algorithm solving the Littlewood–Richardson problem, then the algorithm must make use of the fact that the input diagram is a difference of standard diagrams (unless FP = #P).

The following lemma follows immediately from Theorem 9 since there is no mention of the dropped input requirements in the proof.

Lemma 4 ***L–R/GenDiag*** $\in \#P$. ■

Now, we give a transformation from **Boolean Permanent** (see section 2.3.3.2) to **L–R/GenDiag**.

Given an instance of **Boolean Permanent** $B = [b_{ij}]$, an $n \times n$ boolean matrix, we construct an instance (X, Y, Z) of **L–R/GenDiag** by defining a set of components and then saying how these components are to be combined to produce an instance of **L–R/GenDiag**.

We construct an *initialization component* using the first $n+1$ rows of the diagram. For $0 \leq j \leq n$, we denote the left and right boundaries of row j by x_j^{init} and y_j^{init} respectively. The values are as follows.

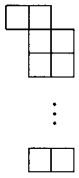
$$x_0^{\text{init}} \leftarrow (n+1)^2$$

$$y_0^{\text{init}} \leftarrow (n+1)^2 + 2$$

$$x_j^{\text{init}} \leftarrow (n+1)^2 + 1 \quad (1 \leq j \leq n)$$

$$y_j^{\text{init}} \leftarrow (n+1)^2 + 3 \quad (1 \leq j \leq n).$$

This gives the following shape.



We construct components for each row of the matrix in three parts. We call the parts the *upper triangular control component*, the *selection component*, and the *lower triangular control component*. The two triangular components are each made using $n+1$ rows in the diagram. For now, we will denote the left and right boundaries of the j^{th} row of the component for the i^{th} row of B with superscript ∇i (for the "upper triangular component") or Δi (for the "lower triangular component") and subscript j. The selection component for each row of the matrix takes only one row in the diagram. The left and right boundaries for the selection component for the i^{th} row of the matrix are denoted by x_i^{sel} and y_i^{sel}, respectively.

For $1 \leq i \leq n$ and $0 \leq j \leq n$, the values of $x_j^{\nabla i}$,$y_j^{\nabla i}$ and $x_j^{\Delta i}$ are assigned as follows.

$$x_j^{\nabla i} \leftarrow i(n+1)$$

$$y_j^{\nabla i} \leftarrow (i+1)(n+1) - 1 - \sum_{k=1}^{j} b_{ik}$$

and

$$x_j^{\Delta i} \leftarrow y_j^{\nabla i}.$$

For $1 \leq i \leq n$ and $0 \leq j \leq n-1$,

$$y_j^{\Delta i} \leftarrow (i+1)(n+1)$$

and for $1 \leq i \leq n$,

$$y_n^{\Delta i} \leftarrow (i+1)(n+1) - 1$$

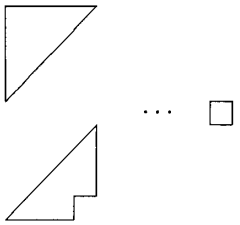
For $1 \leq i \leq n$,

$$x_i^{\text{sel}} \leftarrow (n+1)^2$$

and

$$y_i^{\text{sel}} \leftarrow (n+1)^2 + 1$$

For each row of the matrix, we place the upper triangular component above the selection component and the selection component above the lower triangular component. This gives a shape of:



We now explicitly construct the vectors $X = (x_1, \ldots x_{2n^2+4n+1})$ and $Y = (y_1, \ldots y_{2n^2+4n+1})$ from the components made so far. Both X and Y are constructed in the same manner. The initialization component comprises the first part of the vector:

$$x_i \leftarrow x_{i-1}^{\text{init}} \text{ for } (1 \leq i \leq n+1)$$

$$y_i \leftarrow y_{i-1}^{\text{init}} \text{ for } (1 \leq i \leq n+1).$$

Then, we add the upper triangular component, the selection component, and the lower triangular component for each row i, consecutively:

$$x_{i(2n+3)+j-n-1} \leftarrow x_j^{\nabla i} \text{ for } (1 \leq i \leq n \text{ and } 0 \leq j \leq n),$$

$$y_{i(2n+3)+j-n-1} \leftarrow y_j^{\nabla i} \text{ for } (1 \leq i \leq n \text{ and } 0 \leq j \leq n),$$

$$x_{i(2n+3)} \leftarrow x_i^{\text{sel}} \text{ for } (1 \leq i \leq n),$$

$$y_{i(2n+3)} \leftarrow y_i^{\text{sel}} \text{ for } (1 \leq i \leq n),$$

and

$$x_{i(2n+3)+j+1} \leftarrow x_j^{\Delta i} \text{ for } (1 \leq i \leq n \text{ and } 0 \leq j \leq n),$$

$$y_{i(2n+3)+j+1} \leftarrow y_j^{\Delta i} \text{ for } (1 \leq i \leq n \text{ and } 0 \leq j \leq n).$$

Finally, we make the vector Z. Z is of dimension $n+1$ and each component is $n(n+1)+2$.

For example, the transformation can be applied to the boolean matrix

$$B = \begin{pmatrix} 1 & 0 & 1 \\ 0 & 1 & 0 \\ 1 & 1 & 1 \end{pmatrix}.$$

to get the diagram shown in figure 4.1.

We now describe how to fill figure 4.1 following the Littlewood-Richardson rule (see section 4.2.2).

Following condition 3, we begin by inserting 1's into the boxes. There are boxes in fifteen columns (columns 4 through 18). By condition 7, each of the fourteen 1's must be placed in a different column. We cannot place a 1 in column 18 because condition 5 forbids placing it to the right of an empty box. Therefore, one 1 must be placed in each of columns 4–17. By condition 8, a 1 must be placed in column 17 of the top row; by conditions 3 and 4, this must occur after a 1 has been placed in column 16 of this row. Now, the remaining twelve 1's must be placed in columns 4–15 — and, therefore, in rows 5–31. Again, by conditions 4, 5, and 8, all three boxes in row 5 (columns 5–7) must be filled with 1's. Remaining 1's must be placed in columns 8–15 and, therefore, rows 10–31. Continuing to use conditions 4, 5, and 8 in this manner, one can argue that there is only one valid placement of the fifteen 1's in this diagram.

We can proceed more quickly if we place symbols in any convenient order and verify that we could have followed the order constraints to obtain the same placements when we are done.

Rows 1–4 include eight boxes, exactly two of which contain 1's. Condition 8 can be used to conclude that the remaining six boxes must be filled by exactly two 2's, two 3's, and two 4's. Similarly, the sixteen boxes in rows 5–13 must be filled by exactly four 1's, four 2's, four 3's, and four 4's. The sixteen boxes in rows 14–22 and the sixteen boxes in rows 23–31, must then each be filled by exactly four 1's, four 2's, four 3's, and four 4's as well.

Condition 8 now forces two 2's, 3's, and 4's to be placed into rows 2–4 in the positions shown in the diagram. We also have no choice (by condition 5) in the placement of entries in the remaining three boxes in column 4. Condition 8 (and, at the end, condition 7) can then be used to determine the placement of the remaining entries in column 5.

Only three entries – in row 9 and column 16, row 11 and column 7, and row 12 and column 17, remain to be filled. As in the diagram, let a denote the number assigned to row 9 and column 16. Since there is already a 1 in column 16, $a \neq 1$. As well, $a \neq 3$, since this would violate condition 8 – so $a \in \{2, 4\}$. Denote the entries in column 17 and rows 11 and 12 by c and d respectively. Since rows 5–13 must include exactly four 1's, 2's, 3's, and 4's, $\{a, c, d\} = \{2, 3, 4\}$. If $a = 2$ then, by conditions 3 and 5, $c = 3$ and $d = 4$; otherwise $a = 4$ and, by the same conditions, $c = 2$ and $d = 3$.

Now consider rows 14–22. Again, condition 5 determines the place of entries in column 8; conditions 7 and 8 then determine the placement of entries in rows 14, 15, 16, and 17. The remaining entries (in rows 21 and 22) of column 10 are then fixed by conditions 5 and 7. Once again, since all 1's have already been placed, the entry in row 18 and column 16 must either be 2, 3,

or 4. However, (since the entries in rows 1–17 are now fixed) condition 8 is violated if this entry is either 2 or 4 – so there is a 3 in this position. Now, only a 2 and a 4 remain to fill the remaining boxes in column 7; condition 5 forces us to fill the boxes as shown in the diagram.

Now, only rows 23–31 remain. Conditions 4, 5, and 8 force the placement of entries in the upper triangular component in rows 23–25. Conditions 5 and 7 then fix the entries for the remaining boxes in columns 12–14. The entry b in row 26 and column 16 must be whichever of $\{2,4\}$ is not equal to the entry a above it in the same column, in order to satisfy condition 7. Now, only entries e and f in rows 28 and 29 and column 15 must be determined. Since $\{b,e,f\} = \{2,3,4\}$ and the integer e must be less than the integer f, e and f are determined by the choice of a (since b is).

Thus, there are two ways that we can fill the diagram. All of the boxes that are filled with numbers are forced by the conditions. Those with letters can be filled either with $a = 2$, $c = 3$, $d = 4$, $b = 4$, $e = 2$, and $f = 3$ or with $a = 4$, $c = 2$, $d = 3$, $b = 2$, $e = 3$, and $f = 4$. This is to be expected since the permanent of B is two.

We will now show that the transformation works. That is, the result of applying the Littlewood–Richardson rule to the instance of **L–R/GenDiag** given by the transformation acting on a Boolean matrix B is the permanent of the matrix B.

We can think of the permanent of a Boolean matrix as the number of paths through the matrix, starting on the top row and proceeding row by row to the bottom row, such that we use exactly one element from each row and exactly one element from each column, under the constraint that we use only elements whose value is 1.

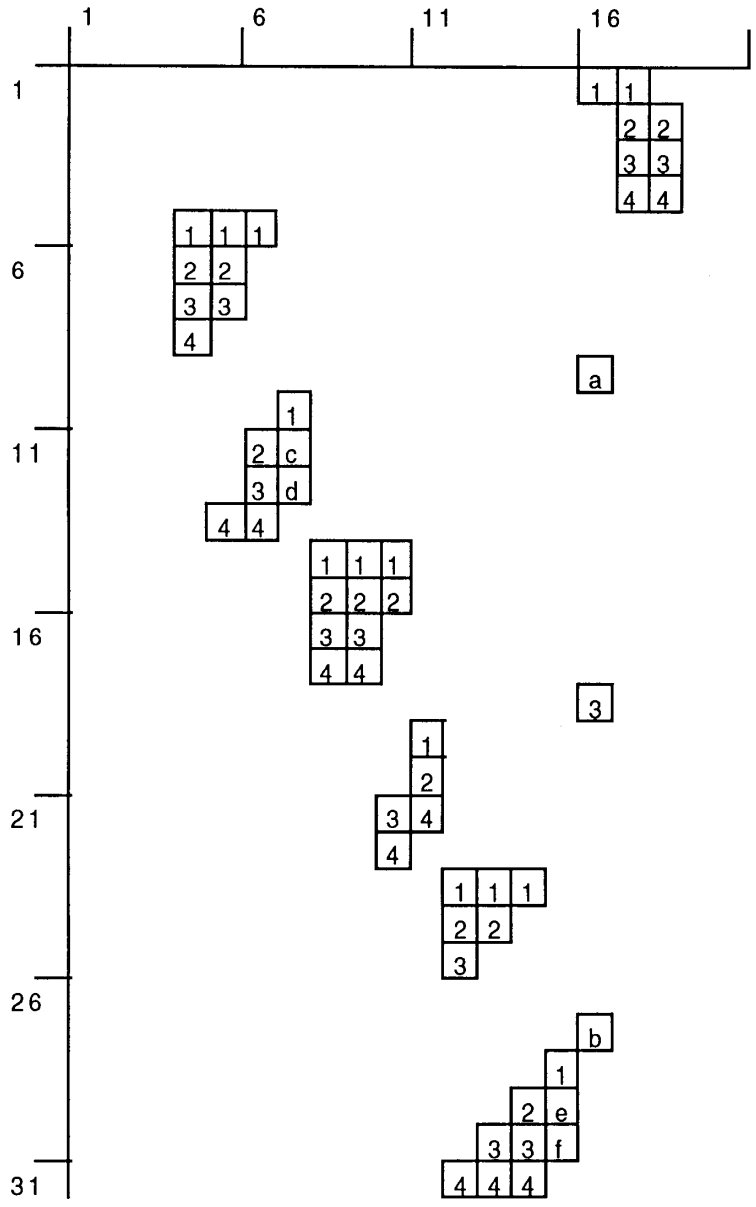


Figure 4.1: The diagram for B.

For each such path through the Boolean matrix B, there is one way that we can fill the diagram (X, Y) with symbols given by Z according to the Littlewood–Richardson rule. Further, there are no other ways that we can fill the diagram according to the Littlewood–Richardson rule.

Specifically, suppose that $\hat{B} = (b_{1,i_1}, \ldots, b_{n,i_n})$ is a path through B contributing to the permanent of B. That is, all $b_{j,i_j} = 1$ and for each $j \neq k$, $i_j \neq i_k$. Then we can fill the diagram (X, Y) as follows: for $1 \leq i \leq n+1$, row i of the initialization component is filled with symbol $i+1$ (see rows 1–4 in Figure 4.1). Each row i of each of the upper triangular components is filled with symbol i (as is the case for the components in rows 5–8, 14–17, and 23–25 in the example). The j^{th} selection component is filled with symbol i_j+1 (specified in the definition of $\hat{B}$ above). Note the entries in column 16 in Figure 4.1. All but the rightmost column of the lower triangular components are filled in same way that the upper triangular components are filled. That is, the i^{th} row of each component is filled with the symbol $i+1$ except possibly for the box in the rightmost column. The rightmost column of each lower j^{th} triangular component is filled, in numerical order from top to bottom, with symbols 1 through $n+1$ leaving out symbol $i_j + 1$. Thus, Figure 4.1 corresponds to the "path" b_{11}, b_{22}, b_{33} through the matrix B if $a = 2$, and corresponds to the "path" b_{13}, b_{22}, b_{31} when $a = 4$.

For the sake of comprehensibility of the description, the above does not make explicit use of the order constraints (conditions 3, 4, 5, and 6 in section 4.2.2). By examining the results of the above description, we can see that it satisfies all of the constraints. We give demonstrations below.

1. This condition is trivially satisfied.

2. Two copies of each symbol are used by the initialization component. Each group of upper triangular, lower triangular, and selection components contains $n+1$ copies of each symbol. Summing, we get $n(n+1)+2$ copies of each symbol, which is what is required by Z.

3. This condition can be trivially satisfied.

4. This condition is satisfied in the initialization component, the upper triangular components, and the selection components since the rows in these components are each filled with one symbol. Now consider a single row in a lower triangular component. Each box except for the one in the last column is filled with the same symbol. The symbol in the last column is either the same or larger than the one in the rest of the row. Since the above is true for each row in each lower triangular component and values in boxes increase as one moves down a column within any of these components, condition 4 can be satisfied in conjunction with condition 3.

5. This is clearly satisfied for the selection components since there are no boxes directly above any of them. Each column in other components is filled from top to bottom with symbols in increasing order and thus this condition can be satisfied in conjunction with condition 3.

6. This condition can be trivially satisfied.

7. Examining each column and referring to the construction and to the proof for condition 5, we see that the only place where this could be violated is in the selection components, which are all contained in a single column. By the construction from the path $\hat{B}$, it immediately follows that the constraint is satisfied there as well.

8. This is clearly satisfied by the placement of values in the initialization component. Also, the prefix string associated with the initialization component contains an equal number of each of the symbols, so the entire string will be a lattice partition if and only if removing the prefix string leaves us with a lattice partition. Each group of contiguous upper triangular component, selection component, and lower triangular component has the same property, so we need only show that the inverse row word for each one of the groups is a lattice partition.

   Consider the i^{th} such group. Clearly, the prefix string corresponding to the upper triangular component causes no trouble. The symbol in the selection component can follow this prefix string since if there is a 1 in the boolean matrix in the $(i,j)^{\text{th}}$ position then there is one more box in the j^{th} row of the upper triangular component than in the $(j+1)^{\text{th}}$ row. Similarly, there can be no problems with the lower triangular component.

Only a few additional comments need be made to show that there are no other ways to fill the diagram in accordance with the rules (given in section 4.2.2). Since there can be no duplicates in any column (condition 7), the initialization component must be filled as described above or the corresponding inverse row word would not be a lattice partition (condition 8). Because of the insertion order constraints (conditions 3, 4, 5, and 6), the first upper triangular component must be filled as given above or we would violate either the column constraint (condition 7) or the lattice partition constraint (condition 8). The selection component for the first row must be given a symbol corresponding to a 1 in the first row of B or we violate the lattice partition constraint (condition 8), because of the construction of the upper triangular component. For the same reasons guiding the filling of the upper triangular

component, the lower triangular component must be filled as described.

Again, since we are forced to fill each group of upper triangular, selection and lower triangular components with the same number of copies of each symbol, the same reasoning can be applied to the each of the components in turn. Finally, all of the selection components are in the same column so they must contain different symbols. They are not vertically adjacent to one another, so the symbols do not need to be in numerical order.

Thus, the result of applying the modified Littlewood-Richardson rule to the constructed instance of **L–R/GenDiag** is the same as evaluating the permanent of the original matrix. Since the transformation can easily be accomplished in polynomial time, we have demonstrated the following lemma.

Lemma 5 ***L–R/GenDiag*** *is #P–hard.* ■

By combining lemmas 4 and 5, we have the following result.

Theorem 10 ***L–R/GenDiag*** *is #P–complete.* ■

4.2.5 Counting Lattice Partitions

An important nontrivial restriction of the Littlewood-Richardson problem can be solved in polynomial time. We restrict the input so that the difference of diagrams has at most one box per row and at most one box per column. In this case, the only constraints that have any effect are conditions 1, 2, and 8. Thus, the problem reduces to counting the lattice partitions corresponding to the partition μ in the input of **DecSymOut**.

We do not use the full power of Theorem 11 (Kreweras' theorem, which follows) in dealing with this case. The theorem gives a formula for counting lattice paths with any start point and any end point. The lattice paths that

we consider always start at the origin. By considering other start points, our method can be extended to give efficient solutions to larger subproblems of **DecSymOut**. However, it appears that these subproblems are still extreme restrictions of **DecSymOut**.

The problem of counting lattice paths has important applications in statistics. Since this is far beyond the scope of this thesis we provide only two pointers to other work in this area: [Rap87] and [Nar79].

Now, we formally define the problem under consideration.

Number Problem 13: CLP

Count Lattice Partitions

Input:

$\lambda \vdash n$

n: given in unary.

Output:

The number of lattice partitions corresponding to λ. ■

Kreweras theorem (theorem 11 below) is used to count paths in a lattice. Before we prove our result, we describe a way of envisaging a lattice partition (see section 2.1.3) in terms of the type of lattice used by Kreweras. Let $a = (a_1, a_2, \ldots, a_m)$ be a point in $\mathbb{Z}^m$ such that $a_i \geq a_{i+1} \geq 0$ for $1 \leq i \leq m-1$ and let $b = (0, 0, \ldots, 0) \in \mathbb{Z}^m$. Geometrically, we think of a lattice path from b to a as a (finite) sequence $s^{(0)}, \ldots, s^{(k)}$ of points in $\mathbb{Z}^m$. For convenience of notation, let us write $b = s^{(0)}$ and $a = s^{(k)}$. Now, we can imagine a point moving from b and ending at a, following the sequence given by the s's.

If $s^{(0)}, s^{(1)}, \ldots, s^{(k)}$ describes a path which involves only moves towards a by a distance of 1 in only one dimension at a time and, further, never crosses above any of a set of diagonal hyperplanes passing through the origin (with

lower indexed coordinates taking priority in measuring height), then the s's describe a lattice partition. The diagonal hyperplanes that we consider are diagonal on only two coordinates. That is, the $(i,j)^{\text{th}}$ diagonal hyperplane is completely specified by $x_i = x_j$.

More formally, let $a = (a_1, \ldots, a_m)$ and $b = (0, \ldots, 0)$ as above, let $n = \sum_{i=1}^{m} a_i$, and let $S = (s^{(0)}, \ldots, s^{(n)})$. For $1 \leq i \leq m$ let e_i denote the i^{th} unit vector, whose j^{th} component is δ_{ij}, for $1 \leq j \leq m$. If for each i, $0 \leq i \leq n$,

$$s^{(i+1)} = s^{(i)} + f(i+1)$$

where f is a function from the first n positive integers to the set of unit vectors in $\mathbb{Z}^m$

$$f : \{1 \ldots n\} \rightarrow \{e_1, e_2, e_3, \ldots, e_m\}$$

and if

$$s_j^{(i)} \geq s_k^{(i')} \quad (0 \leq i' \leq i) \tag{4.7}$$

for $1 \leq i \leq n$ and $1 \leq j < k \leq m$, then we say that the sequence S is a *valid lattice path.*

Since $a_i \geq a_{i+1}$ for $1 \leq i \leq m-1$, we can think of a as a partition. We now construct a one-to-one mapping between the valid lattice paths from $(0, \ldots, 0)$ to $a = (a_1, \ldots, a_m)$ and the lattice partitions of a. Let $A = (A_1, A_2, \ldots, A_n) \in \{1, 2, 3, \ldots m\}^n$ be a lattice partition of a. Now, let

$$s^{(i)} = \sum_{j=1}^{i} e_{A_j} \quad (1 \leq i \leq n).$$

For $1 \leq i \leq n$, let $f(i) = e_{A_j}$. In this light, equation 2.1 and equation 4.7 express the same condition using different notation. Thus, for any $a \in \mathbb{Z}^m$ we have a bijective mapping between lattice paths and lattice partitions and so we know that the number of valid lattice paths from the origin to a is equal to the number of lattice partitions of a.

Kreweras' theorem gives an expression for counting lattice paths allowing duplicate points and jumps in the path. We give the theorem without proof below. Among other places, the theorem is proved in [Nar79].

Theorem 11 (Kreweras 1965) *Let* $0 \leq a_1 \leq \ldots \leq a_n$ *and* $0 \leq b_1 \leq \ldots \leq b_n$ *be two sets of integers satisfying* $b_i \leq a_i$ $(1 \leq i \leq n)$. *Let* $s^{(j)} = (s_1^{(j)}, \ldots, s_n^{(j)})$, $j = 1, 2, \ldots, r$, *be a set of vectors satisfying the inequalities*

$$0 \leq s_1^{(j)} \leq \ldots \leq s_n^{(j)} \quad (1 \leq j \leq r) \tag{4.8}$$

and

$$b_i \leq s_i^{(j)} \leq s_i^{(j+1)} \leq a_i \quad (1 \leq j < r, 1 \leq i \leq n). \tag{4.9}$$

Let $|(b, a; r)|$ *denote the number of* $n \times r$ *matrices* $[s_i^{(j)}]$ *satisfying equations 4.8 and 4.9. For* $r \geq 1$

$$|(b, a; r)| = det\, c_{ij}^{(r)}$$

where

$$c_{ij}^{(r)} = \begin{pmatrix} a_i - b_j + r \\ r + j - i \end{pmatrix} \quad (1 \leq i, j \leq n)$$

and, as usual, if $y < z$ *or* $z < 0$ *then*

$$\begin{pmatrix} y \\ z \end{pmatrix} = 0. \blacksquare$$

By applying the law of inclusion–exclusion, we obtain the following algorithm for **CLP**.

Algorithm 7: Count Lattice Partitions

Input:

$\lambda \vdash n$

n: given in unary.

Output:

The number of lattice partitions corresponding to λ.

Step 1:

Let l be the length of λ.

for $s \leftarrow 1$ **to** $n+1$ **do**

Let $C^{(s)}$ be the $l \times l$ matrix

whose $(i,j)^{\text{th}}$ entry is

$$\begin{pmatrix} \lambda_i + s \\ s + j - i \end{pmatrix}$$

$D^{(s)} \leftarrow \det(C^{(s)})$

endfor

Step 2:

Let $L(1) = D^{(1)}$.

for $s \leftarrow 2$ **to** $n+1$ **do**

Evaluate

$$L(s) = D^{(s)} - \sum_{t=1}^{s-1} \left(L(t) \cdot \begin{pmatrix} s-1 \\ s-t \end{pmatrix} \right)$$

Step 3:

Output $L(n+1)$ ■

We observe that the algorithm is correct. Let $L(s)$ denote the number of paths of length s from the origin to λ where there are no duplicate points. Since $D^{(s)}$ includes duplicate points, in order to find $D^{(s)}$ in terms of L, we count the shorter paths and then account for duplicate points. Suppose there are $L(t)$ distinct paths to λ of length $t < s$. For each of these paths, we can construct some number $Q(s,t)$ of paths of length s by duplicating points in

the path. Now, $s-t$ points must be added to the path and they can be added to t locations thus, so we have

$$Q(s,t) = \begin{pmatrix} t+s-t-1 \\ s-t \end{pmatrix} = \begin{pmatrix} s-1 \\ s-t \end{pmatrix}$$

Then $L(1) = |(0,\lambda;1)| = D^{(1)}$ and for $s > 1$

$$D^{(s)} = |(0,\lambda;s)| = L(s) + \sum_{t=1}^{s-1} L(t) \cdot \begin{pmatrix} s-1 \\ s-t \end{pmatrix}.$$

Solving this for $L(n+1)$ gives the algorithm.

Since the arguments for the binomial coefficients involve values given in unary and determinants can be evaluated in polynomial time (see [AVAU74]), the entire algorithm runs in polynomial time. Thus, we have the following theorem.

Theorem 12 *Count Lattice Partitions* $\in$ *FP.* ■

The number of lattice partitions can be superpolynomial in the input. Consider the number of lattice partitions that can be made corresponding to the partition $\lambda = (2m,m) \vdash 3m$. Clearly, this is at least

$$\begin{pmatrix} 2m \\ m \end{pmatrix}$$

since if we put the first m copies of the first symbol at the beginning of the string, we are free to arrange the remaining m symbols any way we like. This number is superpolynomial in m and thus superpolynomial in $3m$.

Now, let $\lambda = (3m, 3m-1, \ldots, 1)$, let $\mu = (2m,m)$, and let $\gamma = (3m-1, 3m-2, \ldots, 1)$; then the coefficient c_λ of $\chi^{(\lambda)}$ in the decomposition

$$\chi^{(\mu)} \diamond \chi^{(\gamma)} = \sum_{\phi \vdash n+m} c_\phi \chi^{(\phi)}$$

is the number of lattice partitions corresponding to $(2m, m) \vdash 3m$. Thus, **DecSymOut** has instances with solutions that are superpolynomial in the input size.

This means that a counting algorithm (which computes a value by incrementing a counter and thus requires time at least linear in the value it returns) cannot solve **DecSymOut** in polynomial time. If a polynomial time algorithm exists, it must do more than just count. Thus, at the very least, major modifications to the Littlewood–Richardson rule will be required in order to find an efficient algorithm for **DecSymOut**.

Chapter 5

Conclusions and Additional Problems

5.1 Summary of Results

We have examined the computational complexity of finding the characters of finite groups. It was known that the problem can be solved efficiently by Burnside's algorithm when the group is given by its complete multiplication table. The first step of Burnside's algorithm is the computation of "structure constants". Recent work to improve the algorithm has involved reduction of the number of these constants that are computed. We have shown how to efficiently compute a complete set of these "structure constants" from a character table.

Considering the other end of the spectrum of representation sizes, we have shown that finding individual entries in the character table of the symmetric group is computationally hard (under standard complexity theoretic assumptions).

We had limited success in classifying the problem of decomposing outer products of characters of the symmetric group. We defined a generalization of this problem and showed that it was computationally hard (under standard complexity theoretic assumptions). We gave an efficient solution to an important subproblem, namely counting lattice partitions.

5.2 Related Problems

5.2.1 Succinct Specifications of Groups

Giving the multiplication table for a group is not a space efficient method for specifying a group. In particular, a Cayley table requires size quadratic in the order of the group. There are methods for specifying groups where, for many groups, the space required is polylogarithmic in the order of the group. The character table may have size superpolynomial in the input size if a concise specification of the input group is given. In general, there is no natural way to 'index' into the character table as we do with the symmetric group. Thus, the complexity theoretic question becomes, can one compute the character table of a group in time polynomial in the maximum of the input size and the output size?

The problem of finding character tables from such succinct specifications of groups is presently unclassified with regards to its computational complexity. Considerable work has been done on computations with permutation groups (see [But91] for a good introduction). However, even the apparently very special case of finding characters for p–groups has not been analyzed (see [Con90a] and [Sla86]). The state of affairs is similar for matrix groups and finitely presented groups. Groups specified by permutations can be efficiently converted

to finitely presented groups or to matrix groups. Thus, finding the character tables of matrix groups or of finitely presented groups is at least as (computationally) hard as the corresponding problem for permutation groups.

5.2.2 Characters Over Other Fields

We have only considered finding character tables over $\mathbb{C}$. All of the problems asking for complete character tables can be generalized so that a specification for a field K is included in the input and then the question becomes 'what is the character table over K for the group?' There has been extensive work on algorithms in this area. For example, see [Con90b].

5.2.3 Decomposition of Inner Products of Characters of the Symmetric Group

Let T_1 and T_2 be absolutely irreducible matrix representations of S_n. Consider the tensor product $T = T_1 \otimes T_2$ (see section 2.2.3). T is also a representation of S_n but it is not generally irreducible. Like all representations of finite groups over fields of characteristic zero, it is similar to a direct sum of irreducible representations.

Define the inner product of the characters χ^1 and χ^2 of the representations T_1 and T_2 to be the character of the representation $T = T_1 \otimes T_2$. We denote this character by $\phi = \chi^1 \times \chi^2$. The characters of tensor products of similar representations are the same. That is, if $T_1 \sim T'_1$ and $T_2 \sim T'_2$ then $T_1 \otimes T_2 \sim T'_1 \otimes T'_2$. So, the characters of $T_1 \otimes T_2$ and $T'_1 \otimes T'_2$ are the same.

Since $\phi = \chi^1 \times \chi^2$ is a well defined character of S_n, it is expressible as a linear combination of the irreducible characters of S_n. We recall that for the symmetric group, we can succinctly specify irreducible representation classes

using partitions. Thus, we may ask, 'given partitions λ_1, λ_2, $\mu \vdash n$, what is the coefficient c_μ of χ^μ in the decomposition

$$\chi^{\lambda_1} \times \chi^{\lambda_2} = \sum_{\phi \vdash n} c_\phi \chi^\phi ?'$$

It is known (see [Ebe89]) that the coefficients c_μ can be found from such inputs using polynomial space. At present, nothing more is known about the complexity of this problem. It seems quite plausible given the hardness of computing individual characters that this problem is hard as well.

Bibliography

[AVAU74] John E. Hopcroft Alfred V. Aho and Jeffrey D. Ullman. *The Design and Analysis of Algorithms.* Addison–Wesley Publishing Company, Don Mills, Ontario, 1974.

[BF91] Lásló Babai and K. Freidl. Approximate representation theory of finite groups. In *Proceedings of the* 32^{nd} *Annual IEEE FOCS*, pages 733–742, 1991.

[Bur55] W. Burnside. *Theory of Groups of Finite Order.* Dover, New York, second edition, 1955. This is a reprint of the original second edition published in 1911 by Cambridge University Press.

[But91] Gregory Butler. *Fundamental Algorithms for Permutation Groups*, volume 559 of *Lecture Notes in Computer Science.* Springer–Verlag, New York, 1991.

[Con90a] S. B. Conlon. Calculating characters of p–groups. *Journal of Symbolic Computation*, 9:535–550, 1990.

[Con90b] S. B. Conlon. Computing modular and projective character degrees of soluble groups. *Journal of Symbolic Computation*, 9:551–570, 1990.

[Cot63] F. Albert Cotton. *Chemical Applications of Group Theory.* John Wiley and Sons, Inc., New York, 1963.

[CR62] Charles W. Curtis and Irving Reiner. *Representation Theory of Finite Groups and Associative Algebras.* John Wiley and Sons, New York, 1962.

[CR81] Charles W. Curtis and Irving Reiner. *Methods of Representation Theory with applications to finite groups and orders*, volume 1 of *Pure and Applied Mathematics.* John Wiley and Sons, New York, 1981.

[dBR61] G. de B. Robinson. *Representation Theory of the Symmetric Group.* University of Toronto Press, Toronto, 1961.

[Dix67] J. D. Dixon. High speed computations of group characters. *Num. Math.*, 10:446–450, 1967.

[DK85] Wlodzisław Duch and Jacek Karwowski. Symmetric group approach to configuration interaction methods. *Computer Physics Reports*, 2(3):95–170, January/February 1985.

[Ebe89] Wayne Eberly. *Computations for Algebras and Group Representations.* PhD thesis, University of Toronto, 1989. Technical Report: 225/89.

[Edd56] Sir Arthur Stanley Eddington. The theory of groups. In James R. Newman, editor, *The World of Mathematics*, volume 3, pages 1558–1573. New York, Simon and Schuster, 1956.

[Edw77] Harold M. Edwards. *Fermat's Last Theorem: A Genetic Introduction to Algebraic Number Theory.* Springer–Verlag, New York, 1977.

[Eğe82] Ömer Eğecioğlu. Computation of outer products of schur functions. *Computer Physics Communications*, 28:183–187, 1982.

[ER85] Ömer Eğecioğlu and J. B. Remmel. Symmetric and antisymmetric outer plethysms of schur functions. *Atomic and Nuclear Data Tables*, 32:157–196, 1985.

[Fel78] V. Felsch. A bibliography on the use of computers in group theory and related topics: algorithms, implementations, and applications. *SIGSAM Bulletin*, 12:23–86, 1978.

[FH62] William Fulton and Joe Harris. *Representation Theory: A First Course.* Springer-Verlag, New York, 1962.

[Gal74] Z. Galil. On some direct encodings of nondeterministc Turing machines operating in polynomial time into P–complete problems. *SIGACT News*, 6:1:19–24, 1974.

[GJ79] Micheal R. Garey and David S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness.* W. H. Freeman and Company, New York, 1979.

[Ham89] Morton Hamermesh. *Group Theory and Its Application to Physical Problems.* Dover Publications Inc., New York, 1989. This is a reprint of the second (corrected) printing (1964) of the work first published by Addison-Wesley Publishing Compuan, Inc., Reading, Massachusetts, 1962.

[Hoc66] Robin M. Hochstrasser. *Molecular Aspects of Symmetry.* W. A. Benjamin, Inc., New York, 1966.

[HU79] John E. Hopcroft and Jeffery D. Ullman. *Introduction to Automata Theory, Languages, and Computation.* Addison-Wesley Publishing Company, Don Mills, Ontario, 1979.

[IR90] Kenneth Ireland and Michael Rosen. *A Classical Introduction to Number Theory.* Springer–Verlag, New York, second edition, 1990.

[JK81] Gordon James and Adalbert Kerber. *The Representation Theory of the Symmetric Group*, volume 16 of *Encycolpedia of Mathematics and Its Applications.* Addison-Wesley Publishing Company, Reading, Massechusetts, 1981.

[Joh90] D. S. Johnson. A catalog of complexity classes. In J. van Leeuwen, editor, *Handbook of Theoretical Computer Science*, volume A: Algorithms and Complexity, pages 67–162. The MIT Press, Cambridge, Massachusetts, 1990.

[Keo75] R. Keown. *An Introduction to Group Representation Theory*, volume 116 of *Mathematics in Science and Engineering.* Academic Press, New York, 1975.

[Ker91] Adalbert Kerber. *Algebraic Combinatorics Via Finite Group Actions.* Wissenschaftsverlag, Mannheim/Wein/Zurich, 1991.

[Led87] Walter Ledermann. *Introduction to Group Characters.* Cambridge University Press, New York, second edition, 1987.

[Loo83] R. Loos. Computing in algebraic extensions. In B. Buchberger, G. E. Collins, and R. Loos, editors, *Computer Algebra. Symbolic*

and Algebraic Computation., pages 173–187. Springer, New York, 1983. (2nd Edition).

[Mac79] I. G. MacDonald. *Symmetric Functions and Hall Polynomials.* Claredon Press, Oxford, 1979.

[McW63] R. McWeeny. *Symmetry: An Introduction to Group Theory and Its Applications*, volume 3 of *The International Encycolpedia of Physical Chemistry and Chemical Physics.* The Macmillan Company, New York, 1963.

[Mig91] Maurice Mignotte. *Mathematics for Computer Algebra.* Springer-Verlag, New York, 1991.

[Nar79] T. V. Narayana. *Lattice path combinatorics with statistical applications*, volume 23 of *Mathematical Expositions.* University of Toronto Press, Toronto, 1979.

[Neu83] J. Neubüser. Computing with groups and their character tables. In B. Buchberger, G. E. Collins, and R. Loos, editors, *Computer Algebra. Symbolic and Algebraic Computation.*, pages 45–56. Springer, New York, 1983. (2nd Edition).

[Rap87] D. C. Rapaport. Algorithms for lattice statistics. *Computer Physics Reports*, 5(6):268–349, November 1987.

[Rot89] Tony Rothman. *Science à la Mode: Physical Fashions and Fictions.* Princton University Press, Princeton, New Jersey, 1989.

[RW84] J. B. Remmel and R. Whitney. Multiplying schur functions. *Journal of Algorithms*, 5:471–487, 1984.

[Sag91] Bruce E. Sagan. *The Symmetric Group: Representations, Combinatorial Algorithms, and Symmetric Functions.* Wadsworth and Brooks/Cole Advanced Books and Software, Pacific Grove, California, 1991.

[Sch90] Gerhard H. A. Schneider. Dixon's character table algorithm revisited. *Journal of Symbolic Computation*, 9:601–606, 1990.

[Sei90] J. I. Seiferas. Machine independent complexity theory. In J. van Leeuwen, editor, *Handbook of Theoretical Computer Science*, volume A: Algorithms and Complexity, pages 163–186. The MIT Press, Cambridge, Massachusetts, 1990.

[Ser77] J. P. Serre. *Linear Representations of Finite Groups.* Springer-Verlag, New York, 1977.

[Sim77] Janos Simon. On the difference between the one and the many (preliminary version). In *Automata, Languages, and Programming*, volume 52 of *Lecture Notes in Computer Science*, pages 480–491. Springer, Berlin, 1977.

[Sla86] M. C. Slattery. Computing character degrees in p–groups. *Journal of Symbolic Computation*, 2:51–58, 1986.

[Val79] L. G. Valiant. The complexity of computing the permanent. *Theoret. Comput. Sci.*, 8:189–201, 1979.

[vEB90] P. van Emde Boas. Machine models and simulations. In J. van Leeuwen, editor, *Handbook of Theoretical Computer Science*, volume A: Algorithms and Complexity, pages 1–66. The MIT Press, Cambridge, Massachusetts, 1990.

[vL90] J. van Leeuwen, editor. *Handbook of Theoretical Computer Science*, volume A: Algorithms and Complexity. The MIT Press, Cambridge, Massachusetts, 1990.

[Wey50] Hermann Weyl. *The Theory of Groups and Quantum Mechanics.* Dover Publications, Inc, New York, 1950. This is a reprint of the original English translation published in 1931 by Methuen and Company, Ltd.

[Wie64] Helmut Wielandt. *Finite Permutation Groups.* Academic Press, New York, 1964. Translated by R. Bercov.