Distinguishing Permutation Isomorphism Classes of Groups

Kwok-On LAU

A Thesis
in
The Department
of
Computer Science

Presented in Partial Fulfilment of the Requirements for
the Degree of Master of Computer Science at
Concordia University
Montréal, Québec, Canada

October 1992

ISBN 0-315-84657-7

Abstract

Distinguishing Permutation Isomorphism Classes of Groups

Kwok-On LAU

The ability to distinguish permutation isomorphism classes of groups is an important step in the computation of Galois groups of polynomials over the rationals. In order to distinguish permutation isomorphism classes of groups, it is useful to have an extensive list of their invariants. These invariants include properties such as the order, imprimitivity, parity and shapes, as well as the orbit lengths of sets and sequences. Computing these characteristics can be extremely time consuming. In this thesis, a detailed description of efficient algorithms for solving the problem using the concept of expanding horizon and orbit computation is presented.

Acknowledgments

I would like to express my sincere gratitude to Dr. Clement Lam, my thesis supervisor, for his guidance and valuable insight throughout this research. His support and patience are invaluable in the preparation of this thesis.

Mr. Larry Thiel, maintainer of the ISOM package, has helped debug my programs; while Dr. Greg Butler has given me much consultation on the use of Cayley; and Dr. John McKay and Mr. Thomas Mattman have shown their patience in explaining their requirements for their Galois program. I would like to express my sincere thanks to all of them.

I would also like to thank the staff of Computer Services, the Department of Computer Science and the Centre Interuniversitaire en Calcul Mathématique Algébrique at Concordia University for their permission to use their computer facilities.

During the development of the research and the preparation of my thesis, my wife, Juliana Chan, has given me much encouragement. In addition, a number of my friends including Dr. Derek Pao and Mr. Ka-Leung Ma have given me support. In this regard, I also owe a debt of gratitude to all of them.

Contents

List of Figures

List of Tables

List of Symbols

α_i	the i-th element of a permutation
A	a sequence
$\mathcal{A}$	a sequence in the orbit on A
$C(n,r)$	the number of combinations of n objects taken r at a time
D_n	the Dihedral group of degree n and order $2n$
ϵ	the identity element of a group
g	an element of G
G	a group
G_ω	the stabilizer of ω in G
$G_{(i)}$	the pointwise stabilizer of the first i elements of a permutation
K	a generating set
n	the degree of a group
ω	an element in Ω_n
ω^G	the orbit on ω in G
Ω_n	the set $\{1,2.\ldots,n\}$
$p(n)$	the number of partitions of n
$p_m(n)$	the number of partitions of n into parts not exceeding m
$P(n,r)$	the number of permutations of n objects taken r at a time
ϕ	the encoding function for partitions in lex order
ρ	the encoding function for sequences in colex order
r	the length of a set or sequence
S_n	the symmetric group of degree n

Chapter 1

Introduction

In this chapter, an introduction to the usefulness of the research work is given. Its main use is in finding the Galois group of a polynomial. So far, there has not been much similar research done for distinguishing permutation isomorphism classes of groups.

1.1 Galois Theory and Galois Group

The aim of Galois theory is to study the solution of polynomial equations

$$f(t) = t^n + a_{n-1}t^{n-1} + \ldots + a_0 = 0$$

and in particular to distinguish those that can be solved by a 'formula' from those that cannot [24]. By a formula we mean a radical expression : anything that can be built up from the coefficients a_i by a finite number of the operations of addition, subtraction, multiplication, division, and also by taking nth roots. In modern terms, Galois's main idea is to look at the symmetries of the roots of the polynomial $f(t)$. These form a group, its Galois group, and the solution of the polynomial equation is reflected in various properties of the Galois group.

In order to apply Galois theory to specific polynomials, it is necessary to compute the corresponding Galois group. This is far from being a simple or straight-forward task.

1.2 Permutation Isomorphism

The Galois group of a polynomial can be represented as a permutation group, G, of its roots. A reordering of the roots leads to another group, H, which is permutation isomorphic to G. Thus, in order to determine the Galois groups of polynomials, it is useful to have tables of invariants for the permutation isomorphism class. These invariant can be the order, imprimitivity, parity, shapes, as well as the orbit lengths of sets and sequences.

Given a polynomial, some invariants are easy to compute but some are not. For example it is very difficult to find the order of the Galois group. On the other hand, it is easy to obtain some information about the shapes by factorizing the polynomial modulo primes that do not divide the discriminant. However, it is difficult to prove that we have generated all the possible shapes of the Galois group.

Information about orbit lengths can be obtained by constructing and factoring a resolvent polynomial using symmetric functions of the roots. For example, if $\{\alpha_i\}$ are the roots of a polynomial f, then the degrees of the factors of

$$R(t) = \prod_{i<j}(t - (\alpha_i + \alpha_j))$$

are the orbit lengths of the 2-sets under the action of the Galois group. However, this resolvent polynomial has degree $C(n,2)$, and the current state-of-the art algorithms can only factorize polynomials of degrees up to several hundreds. This puts a limit on the usefulness of the orbit length information for r sets when the value of r is large. Thus, further work still needs to be done.

In [23], every transitive permutation group of degree 3 to 7 is realized as a Galois group over the rationals. In [16], the groups of degree 8 are realized. More discussions on the advances in computational Galois theory can be found in [19].

1.3 Our Approach

J. McKay and E. Regener have studied the actions of transitive permutation groups of degree up to 11 in [18], using an adaptation of [17] to the subset indexing procedures described in [15].

It is useful to have tables of permutation isomorphism class invariants in order to identify Galois groups. We find these invariants starting from tables of the transitive permutation groups from degree 2 to 15 [**5**].

There are a number of invariants that we are interested in computing. The order is obtained by using the Schreier-Sims algorithm, imprimitivity by Atkinson's algorithm, and shapes by construrting conjugacy classes of elements using an inductive schema. These computations are done using the algebraic programming language Cayley, which is further discussed in Chapter 3. The shapes are encoded into a linear array using the number of partitions as a perfect hashing function. The parity is determined from the cycles in the generators of the group while they are being read.

Our main concern is the computation of orbit lengths. First, the group is built and sequences are generated. For r-sets, the methods of expanding horizon and registration technique are employed. For r-sequences, a theorem for computing the number of sequences in an orbit is developed. This theorem is then applied to compute the orbit length from the orbits on the elements. Here, ISOM, a package for isomorphism testing, has been used extensively. It is further discussed in Chapter 4.

A list of invariants for groups of degree 2 to 15 is obtained. This enhances the identification of permutation isomorphism classes of groups, leading to more Galois groups being determined.

1.4 Contribution

In this research work, my major contributions include:

- computation of orbit lengths of r-sets, in particular:
  - design of the algorithm
  - applying the method of expanding horizon
  - using colex order to code an r-set
  - including a consistency check
  - coding the algorithm in C language

- computation of orbit lengths of r-sequences, in particular:
  - design of the algorithm
  - developing a theorem for orbit alculation (although we later found that this theorem has been developed in somewhere else)
  - including a consistency check
  - coding the algorithm in C language
- running through all the 650 groups in the CAYLEY library

These are described in Sections 4.3 and 4.4. Other minor contributions include.

- maintaining a shell script to call CAYLEY for finding the order, imprimitivity and shapes of a group
- checking the parity of a group
- finding the smallest set of generators among the given ones
- encoding the shapes into a linear array using the number of partitions of the degree of a group
- re-formating the output from the main process so as to drive other programs

These are described in Sections 3.1, 4.1, 4.2 and 4.7.

1.5 Organization of the Thesis

The layout of the thesis is as follows. Chapter 1 introduces the research work. Chapter 2 gives the relevant definitions and mathematical preliminaries. Chapters 3 and 4 describe the methodology employed to solve the problem. Chapter 5 gives a general picture of the results. Chapter 6 analyses the time complexity and usefulness of the algorithms. It also contains a discussion of some possible further improvements and a conclusion.

Chapter 2

Basic Definitions and Mathematical Preliminaries

In this chapter, some basic concepts and definitions in group theory and number theory are presented. Based on these concepts and definitions, we develop algorithms to solve our problems. Reference to group theory can be found in **[2, 10, 14]**, while reference to number theory can be found in **[11, 20, 25]**.

A permutation is a one-to-one mapping of a set onto itself. Let

$$\Omega_4 = \{1, 2, 3, 4\}.$$

A permutation of Ω_4 can be written in either the image form (as in [1,4,3,2]) or the cycle form (as in (2,4)). When there is no ambiguity, a comma that is used to separate two numerals representing two elements of the set will be omitted. This happens when each of the numerals has only one digit. Thus, $[1, 2, 3, 4]$ will be written [1234]. However, a permutation such as $[12, 3, 4]$, which contains a numeral greater than 9, will keep the commas. The same rule applies to an r-sequence, and to an r-set like $\{13\}$. In this thesis, permutations act on the right, so the image of $\omega \in \Omega_n$ under the permutation p is denoted by ω^p. We compose permutations so that $\omega^{pq} = (\omega^p)^q$ for permutations p and q.

To illustrate the meaning of some definitions, D_4 will be used as an example. The group can be best illustrated by considering the labeling of the corners of a square as in Figure 2.1. The set of four elements, Ω_4, is chosen to be the set of corners. The permutation [2341] acting on the square is an anti-clockwise rotation that relabels

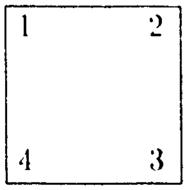

Figure 2.1: Labeling the corners of a square

the top left corner as a 2. The permutation [1432] is a reflection across the diagonal from corner 1 to corner 3.

2.1 Group and Subgroup

A group is a set G closed under an associative binary operation "$\cdot$" (product) and satisfying the following axioms :

1. There exists an identity element e such that $\forall g \in G, e \cdot g = g \cdot e = g$.
2. $\forall g \in G$, there exists an inverse g^{-1} such that $g \cdot g^{-1} = g^{-1} \cdot g = e$.

A subset H of the elements of a group G which forms a group with respect to the product as defined in G is called a subgroup of G.

2.2 Symmetric Group and Permutation Group

The symmetric group of degree n, S_n, on Ω_n is the set of all permutations on Ω_n. A subgroup of the symmetric group S_n is called a permutation group of degree n.

2.3 Generating Set

Let K be a subset of S_n. Then K is a generating set for a subgroup G of S_n if G is the smallest subgroup of S_n which contains K as a subset. The group G can be obtained from K by repeatedly applying all the permutations in K to the identity permutation e. The elements of K are called generators of G. In our example, a generating set for D_4 is

$$K = \{[1432], [2341]\}$$

shape	*permutations*
1 1 1 1	(1)(2)(3)(4)
2 1 1	(24)(1)(3), (13)(2)(4)
2 2	(13)(24), (12)(34), (14)(23)
4	(1234), (1432)

Table 2.1: Shapes of D_4

and

$$D_4 = \{e, [2341], [3412], [4123], [1432], [4321], [3214], [2143]\}.$$

2.4 Order

The number of elements of a group G is called the order of G, and is denoted by $|G|$. In our example, $|D_4| = 8$.

2.5 Shape

The shape of a permutation is a multiset of the lengths of the cycles of the permutation. Table 2.1 shows the shapes that D_4 contains.

2.6 Parity

A permutation is even if it has an even number of even length cycles, otherwise, the permutation is odd. A group is even if all its generators are even, otherwise, the group is odd. Thus, D_4 is odd.

2.7 Stabilizer

The stabilizer, G_ω, of an element $\omega \in \Omega_n$ in the group G is the set

$$G_\omega = \{g | g \in G \wedge \omega^g = \omega\}.$$

The stabilizer is a subgroup of G. In our example, the stabilizer of the point 1 in D_4 is the subgroup $\{e, [1432]\}$.

2.8 Orbit

The orbit, ω^G, on ω in G is the set

$$\omega^G = \{\mu | \mu \in \Omega_n \ \wedge \ \exists g \in G : \omega^g = \mu\}.$$

In our example, the orbit on 1 in D_4 is the set $\{1,2,3,4\}$.

2.9 Orbit on an r-sequence

An r-sequence is a sequence of length r with distinct elements taken from Ω_n. The image of an r-sequence $[\alpha_1 \dots \alpha_r]$ under a permutation g is

$$[\alpha_1 \dots \alpha_r]^g = [\alpha_1^g \dots \alpha_r^g].$$

The orbit on an r-sequence is the orbit on the r sequence under a permutation group. For example, the orbits on 2-sequences under D_4 are :

$$\{[12],[23],[34],[41],[14],[43],[32],[21]\},$$
and $$\{[13],[24],[31],[42]\}.$$

2.10 Orbit on an r-set

An r-set is a subset of size r with distinct elements taken from Ω_n. The image of an r-set $\{\alpha_1 \dots \alpha_r\}$ under a permutation g is

$$\{\alpha_1 \dots \alpha_r\}^g = \{\alpha_1^g \dots \alpha_r^g\}.$$

The orbit on an r-set is the orbit on the r set under a permutation group. For example, the orbits on 2-sets under D_4 are :

$$\{\{12\},\{23\},\{34\},\{41\}\},$$
and $$\{\{13\},\{24\}\}.$$

2.11 Imprimitivity

We say that a group G acting on Ω_n is imprimitive if Ω_n can be partitioned non-trivially into disjoint sets, $B_i, i = 1, \ldots, m$, called blocks, such that for every permutation g of G and every block B_i,

$$B_i^g = B_j, \textit{ for some } j.$$

A partition is trivial if $m = 1$ or $m = n$. If G is not imprimitive, we say that G is primitive. In our example, D_4 is imprimitive as we can have

$$\Omega_4 = \{1, 3\} \cup \{2, 4\}.$$

2.12 Transitive Group

A permutation group G on Ω_n is transitive if for all $\omega_i, \omega_j \in \Omega_n$ there is a $g \in G$ with $\omega_i^g = \omega_j$. In other words, $\omega^G = \Omega_n$ for each $\omega \in \Omega_n$. In our example, we have $1^{D_4} = 2^{D_4} = 3^{D_4} = 4^{D_4} = \Omega_4$. Therefore D_4 is transitive.

2.13 Permutation Isomorphic Groups

Two subgroups G and H of S_n are permutation isomorphic if there exists a permutation $s \in S_n$ such that

$$s^{-1}Gs = H.$$

This may be realized as a relabeling of the elements.

2.14 Lex and Colex Order

Lexicographic order (lex order for short) derives its name from the order imposed on the words in a dictionary [25].

Given two sequences of the same length

$$p = \alpha_1, \ldots, \alpha_r$$

and

$$q = \beta_1, \ldots, \beta_r.$$

If $p \neq q$, we scan from left to right until we find the first k such that $\alpha_k \neq \beta_k$. If $\alpha_k < \beta_k$, we say p is lexicographically less than q, otherwise, we say p is lexicographically greater than q. If we scan, instead, from right to left, the resulting order will be called the colex order.

A set of numbers can be viewed as a sequence if we list the elements of the set in increasing order. Thus, we also have lex and colex order for sets. As an example, the 2-sets obtained from Ω_4 in increasing lex order are $\{12\}$, $\{13\}$, $\{14\}$, $\{23\}$, $\{24\}$, $\{34\}$; while that in increasing colex order are $\{12\}$, $\{13\}$, $\{23\}$, $\{14\}$, $\{24\}$, $\{34\}$.

2.15 Partition of a Number

A partition of a number n is a representation of n as the sum of any number of positive integral parts [**11**]. Thus,

$$4 = 3 + 1 = 2 + 2 = 2 + 1 + 1 = 1 + 1 + 1 + 1$$

has 5 partitions. The order of the parts is irrelevant, so that we may, when we please, suppose the parts to be arranged in descending order of magnitude. We denote by $p(n)$ the number of partitions of n; thus $p(4) = 5$. It is convenient to define $p(0) = 1$ [**20**].

We denote by $p_m(n)$ the number of partitions of n into parts not exceeding m. The convention $p_m(0) = 1$ for all m is made in [**20**]. It is convenient to define $p_0(n) = 0$ for $n \neq 0$. In [**20**], we have the following theorems:

Theorem 2.1 *$p_m(n) = p(n)$ if $n \leq m$.*

Theorem 2.2 *$p_m(n) = p_{m-1}(n) + p_m(n - m)$ if $n \geq m > 1$.*

Table 2.2 shows the values of $p_m(n)$ for n and m running from 0 to 4.

		m				
		0	1	2	3	4
	0	1	1	1	1	1
	1	0	1	1	1	1
n	2	0	1	2	2	2
	3	0	1	2	3	3
	4	0	1	3	4	5

Table 2.2: Number of partitions

Chapter 3

Preprocessing

The generating set for each group of degree 2 to 15 is stored in the Cayley library, *trngps*. These are analyzed preliminarily as a preprocessing step. Cayley is an algebraic programming language developed at the University of Sydney in Australia [9]. It also contains many built-in functions and libraries of procedures which enable the computation of order, imprimitivity and shape. Thus, these three invariants can be computed with Cayley.

3.1 Computing Order, Imprimitivity and Shape

The order of a group may be obtained by counting the number of elements in the group. This can be done by using the method of expanding horizon [13]. However, it is a very tedious job. Instead, the Schreier-Sims algorithm [22] can be used to efficiently generate a set of strong generators relative to a base. The order of the group can then be computed as a product of the lengths of the basic orbits. For more details, see [8]. To study whether a transitive group is imprimitive, we can use Atkinson's algorithm [1]. The shapes of a group can be computed by studying the cycle types of all the elements of the group. To cut down the size of the search, we need only search one element in each conjugacy class. The conjugacy classes of elements of a permutation group can be constructed with the random algorithm in [9], or the inductive schema presented in [7].

3.2 Implementation Using Cayley

In [6], there is an introduction to using Cayley under UNIX. For a more detailed description, see [9].

Figure 3.1 shows the library for the dihedral group D_4.

```
LIBRARY t4n3;
G:perm(4); G.generators:
A = (1,2,3,4),
B = (2,4);
finish;
```

Figure 3.1: Library for D_4

The processing for a group starts with the analysis of the group by Cayley to obtain information such as the order, imprimitivity, and shape. For example, before D_4 is studied, its library is named *t4n3*, and is stored as a file with name *t4n3*. The reason for using a systematic name, instead of the group name, is to conform to the requirements of the UNIX operating system. For example, we may desire to use Σ to denote symmetric group, $+$ to denote an even group, $\ldots$, etc. However, in the UNIX system, we are not allowed to use such special characters as the filename.

This library can be called using the *library* command as *library t4n3*. This will read the library called *t4n3* and execute the Cayley commands contained in the library to define the group, say G, to be analyzed. The number of generators is printed out by *print ngenerators*(G). If x is a generator, *print* x will print x in cycle form, while *print eltseq*(x) will print x in image form. The group G is then further analyzed by calling a library procedure *gpanalyse* [5], to obtain the order, primitivity and shapes information.

For a description on the transitive permutation groups in the Cayley library, see [4, 5, 21, 22].

3.3 Output

The output from Cayley for a group G is then edited by using the stream editor *sed* in UNIX. It edits according to a script of requests [12]. Of particular interest is the repeated global substitution request which substitues all the occurrences of a specified string by a replacement string. This is very useful for processing Cayley output to extract information or reformat it.

Figure 3.2 shows an example of the edited output file for D_4 on the left column. The right column is added here for explaining the meanings associated with the data.

```
output             meaning
------             -------
2                  number of generators
2 3 4 1            1st generator in image form
1 4 3 2            2nd generator in image form
(1,2,3,4)          1st generator in cycle form
(2,4)              2nd generator in cycle form
8                  order of the group
1                  1=primitive, 0=imprimitive
3                  number of different shapes
2 1  1             1st shape
2 2                2nd shape
4                  3rd shape
```

Figure 3.2: Output produced for D_4

Chapter 4

Main Process

In the main process, the parity of a group is determined, the shapes are encoded into an array, and the group action on r-sets and r-sequences is studied.

A number of theorems have been applied to improve efficiency. These include the umber of partitions of a natural number, colex ordering, expanding horizon, and the number of sequences in an orbit.

4.1 Checking Parity and Reducing the Number of Generators

The parity of a generator p is determined while its cycle form is being read. If p has m cycles represented as

$$(\alpha_1, \ldots, \alpha_{k_1}) \ldots (\gamma_1, \ldots, \gamma_{k_m}),$$

let

$$P = \sum_{i=1}^{m}(k_i + 1) = \sum_{i=1}^{m} k_i + \sum_{i=1}^{m} 1,$$

which is equal to the degree of the group plus the number of cycles in p. Then, p is even if and only if P is even. In modulus 2,

$$P = \sum_{i=1}^{m}(k_i + 1) = \sum_{i=1}^{m}(k_i - 1).$$

Thus, the conclusion can be drawn by just counting the total number of commas in the representation of p.

Reduction of the number of generators can reduce the complexity of the program which computes the obit lengths of r-sets. In addition, this can save some storage space. In order to find the smallest subset of the given generators which can generate the same group, all the subsets of the given generators are considered. A group is generated for each subset. If the order of the group thus generated is not reduced, this is an equivalent set of generators. When the smallest of such equivalent sets is found, the rest of the gi' en generators are marked as redundant. This equivalent set thus obtained is not a minimal set of generators in the sense that the group requires this number of generators.

4.2 Partition and Shape

We let $\pi = [\alpha_1 \ldots \alpha_m]$ be a partition of n. In other words, $\alpha_1 + \ldots + \alpha_m = n$, and $\alpha_1 \geq \ldots \geq \alpha_m > 0$. Now, we can use a perfect hashing function to encode the shapes into a linear array due to the following theorem:

Theorem 4.1 *The mapping ϕ defined on a partition π by*

$$\phi(\pi) = \sum_{i=1}^{n} p_{\alpha_i - 1}(n - s_{i-1}),$$

where

$$s_i = \sum_{j=1}^{i} \alpha_j,$$

is a perfect hashing function from the partition to a ranking number.

Proof: Let us consider a position i where $1 \leq i \leq m$. The number of partitions that comes before

$$[\alpha_1 \ldots \alpha_{i-1} \alpha_i \underbrace{1 \ldots 1}_{n-s_i}]$$

down to

$$[\alpha_1 \ldots \alpha_{i-1} \underbrace{1 \ldots 1}_{n-s_{i-1}}]$$

is $p_{\alpha_i - 1}(n - s_{i-1})$. Summing over all the positions, the number of partitions ahead of $[\alpha_1 \ldots \alpha_m]$ is $\sum_{i=1}^{n} p_{\alpha_i - 1}(n - s_{i-1})$. □

index	*partition pattern*
0	[1111]
1	[211]
2	[22]
3	[31]
4	[4]

Table 4.1: Encoding partitions of the integer 4

Example: The integer 4 can be partitioned and coded as shown in Table 4.1.

The function ϕ is used as a perfect hashing function for encoding the shapes into an array. For example, the shapes of D_4 are encoded into an array as [1101]. A '1' in the array indicates that the group has a shape which has the index as its rank. The array starts with index number 1, index 0 is not shown, as every group must have the identity. In this example, the '0' in the array indicates that D_4 does not have a shape of index number 3, which corresponds to the partition [31].

4.3 Finding the Orbit Length of an r-set

4.3.1 Steps

The following steps are repeatedly used to find the orbit length of an r-set until no more r-sets can be generated:

1. Generate a new r-set such that the elements are in an increasing order.
2. If this set is in an orbit of a previously generated set, this set will be abandoned.
3. If this set is not in any previously generated orbits, the method of expanding horizon is applied to find its orbit.

4.3.2 Expanding Horizon

In the method of expanding horizon, each generator is applied to a set which is popped from the top of a stack. Any new set thus generated is pushed onto the stack.

stack	*orbit*	*popped set*	*apply*
{12}	{12}	{12}	p
{23}	{12},{23}	{12}	q
{23},{14}	{12},{14},{23}	{14}	p
{23}	{12},{14},{23}	{14}	q
{23}	{12},{14},{23}	{23}	p
{34}	{12},{14},{23},{34}	{23}	q
{34}	{12},{14},{23},{34}	{34}	p
{}	{12},{14},{23},{34}	{34}	q
{}	{12},{14},{23},{34}		

Table 4.2: Applying expanding horizon to {12}

stack	*orbit*	*popped set*	*apply*
{13}	{13}	{13}	p
{24}	{13},{24}	{13}	q
{24}	{13},{24}	{24}	p
{}	{13},{24}	{24}	q
{}	{13},{24}		

Table 4.3: Applying expanding horizon to {13}

As an example, consider D_4 with two generators $p = [2341]$ and $q = [1432]$. To find the orbit lengths of 2-sets, [12] and [13] are generated. Table 4.2 shows that the orbit length of {12} is 4, and Table 4.3 shows that the orbit length of {13} is 2.

4.3.3 Use of Colex Order

To save the time for searching whether a set is in a previously generated orbit, the registration technique is used [3]. In this technique, a boolean array is first cleared with zeros. When a set is encounted, it is mapped to an index for the array. If the boolean value for that index is zero, the set is new, and the boolean value is then set to one. If the boolean value for that index is one, the set is not new. A mapping function for colex ordering can be used to mark the sets that had been included in an orbit due to the following theorem in [25]:

		α_2			
		1	2	3	4
	1	-	1	2	4
α_1	2	-	-	3	5
	3	-	-	-	6
	4	-	-	-	-

Table 4.4: Encoding sets arranged in increasing colex order

Theorem 4.2 *The mapping ρ defined by*

$$\rho(\{\alpha_1 \ldots \alpha_r\}) = 1 + \sum_{i=1}^{r} C(\alpha_i - 1, i)$$

is the order isomorphism between the list of r-sets, which are chosen from Ω_n and arranged in increasing colex order, and

$$\{1, 2, \ldots, C(n, r)\}.$$

Example: The sets $\{\alpha_1\alpha_2\} = \{12\}, \{13\}, \{23\}, \{14\}, \{24\}, \{34\}$ can be mapped to $\{1, 2, 3, 4, 5, 6\}$ as shown in Table 4.4.

4.3.4 Sets of Special Lengths

For a transitive group of degree n, the orbit length of a 1-set is equal to n, and the orbit length of an n-set is equal to 1. The orbit length of an $(n - r)$-set is equal to that of the complementary r-set.

4.3.5 Storage

The orbit lengths are inserted into a binary search tree, and the frequency of occurrence of each length is updated. After all the orbit lengths of r-sets are found, they will be printed out in increasing order of length together with their frequencies.

4.3.6 Consistency Checking

A consistency check is that the sum of the products of the length and the frequency is $C(n,r)$, where

$$C(n,r) = \frac{n!}{(n-r)!r!}.$$

This is the total number of distinct r-sets obtainable from n distinct elements.

4.4 Finding the Orbit Length of an r-sequence

4.4.1 Steps

The following steps are repeatedly used to find the orbit length of an r-sequence until no more r-sequences can be generated:

1. Generate a new r-sequence.
2. For each element in the r-sequence, find the orbit length under the group that fixes all the preceding elements.
3. Multiply these orbit lengths together to obtain the orbit length of the r-sequence.

4.4.2 Number of Sequences in an Orbit

An efficient method for computing the orbit length of an r-sequence is based on the following theorem:

Theorem 4.3 *The orbit length of a sequence $A_r = [\alpha_1 \dots \alpha_r]$ under the action of the group G is equal to*

$$N_r = \prod_{i=1}^{r} |\alpha_i^{G_{(i-1)}}|$$

where $G_{(k)}$ denotes the subgroup of G that fixes the first k elements $\alpha_1, \dots, \alpha_k$ of A, and $G_{(0)} = G$.

Proof: We will prove this theorem by using induction on r.
Consider the extension of a sequence from $A_1 = [\alpha_1]$ to $R = [\alpha_1 \dots \alpha_r]$.
At level 1,

$$A_1 = [\alpha_1],$$

there are $N_1 = |\alpha_1^G| = |\alpha_1^{G_{(0)}}|$ distinct sequences of length 1 that are isomorphic to A under G.

Assume that, extending down the path $A_1 \rightarrow \ldots \rightarrow A_k$ where

$$A_k = [\alpha_1 \ldots \alpha_k].$$

the number of distinct sequences isomorphic to A_k is

$$N_k = |\alpha_1^{G_{(0)}}| \times \ldots \times |\alpha_k^{G_{(k-1)}}|.$$

then, for level $k+1$, A_k can be extended to

$$A_{k+1} = [\alpha_1 \ldots \alpha_k \alpha_{k+1}]$$

and

$$B = A_{k+1}^q = [\alpha_1 \ldots \alpha_k \alpha_{k+1}^q],\ q \in G_{(k)} \ \wedge \ q \neq \epsilon.$$

Let one of the N_k sequences in the orbit on A_k be

$$\mathcal{A}_k = A_k^p = [\alpha_1^p \ldots \alpha_k^p],\ p \in G \ \wedge \ p \neq \epsilon.$$

We can obtain

$$\mathcal{A}_{k+1} = A_{k+1}^p = [\alpha_1^p \ldots \alpha_k^p \alpha_{k+1}^p]$$

and

$$\mathcal{B} = B^p = [\alpha_1^p \ldots \alpha_k^p \alpha_{k+1}^{qp}].$$

Thus, $\mathcal{B} = B^p = A_{k+1}^{qp} = (\mathcal{A}_{k+1}^{p^{-1}})^{qp} = \mathcal{A}_{k+1}^{p^{-1}qp}$, showing that $\mathcal{A}_{k+1}$ and $\mathcal{B}$ are in the same orbit.

For two distinct sequences B_1 and B_2 in the orbit on A_{k+1} obtained from $q_1, q_2 \in G_{(k)}$, we have

$$\begin{aligned} & \alpha_{k+1}^{q_1} \neq \alpha_{k+1}^{q_2} \\ \Rightarrow \ & \alpha_{k+1}^{q_1 s} \neq \alpha_{k+1}^{q_2 s} \end{aligned}$$

Thus, the B's obtained from B's are distinct, so that the orbit length of $\mathcal{A}_{k+1}$ is equal to that of A_{k+1}, which is $|\alpha_{k+1}^{G_{(k)}}|$. In addition, if a sequence C is in the orbit on A_{k+1}, the parent of C must be isomorphic to A_k. Therefore, by our assumption for level k, the number of distinct sequences isomorphic to A_{k+1} is

$$\begin{aligned} N_{k+1} &= N_k \times |\alpha_{k+1}^{G_{(k)}}| \\ &= |\alpha_1^{G_{(0)}}| \times \ldots \times |\alpha_{k+1}^{G_{(k)}}|. \square \end{aligned}$$

For a transitive group G on Ω_n, $|\alpha_1^{G_{(0)}}| = n$, thus, we do not need to compute the orbit length for the first level.

4.4.3 Example

As an example, consider D_4. To find the orbit length of 2-sequences, [12] and [13] are generated. The orbit on 1 under D_4 is $\{1, 2, 3, 4\}$ with a length of 4. The orbit on 2 under a subgroup of D_4 that fixes 1 is $\{2, 4\}$ with a length of 2. The orbit on 3 under a subgroup of D_4 that fixes 1 is $\{3\}$ with a length of 1. Thus, the orbit length of [12] is $4 \times 2 = 8$, and the orbit length of [13] is $4 \times 1 = 4$. The whole picture is shown in Figure 4.1. Here, 2-sequences are shown to the right of the 1-sequences from which they extend, and the image of a sequence under a permutation is shown with an arrow pointing to its image.

4.4.4 Sequences of Special Lengths

For a transitive group of degree n, the orbit length of a 1-sequence is equal to n. The orbit length of an n-sequence is equal to the order of the group. The orbit length of an $(n-1)$-sequence is equal to that of an n-sequence obtained by appending the remaining element in Ω_n to the $(n-1)$-sequence.

4.4.5 Storage

The orbit lengths are inserted into a binary search tree, and the frequency of occurrence of each length is updated. After all the orbit lengths of r-sequences are found, they will be printed out in increasing order of length together with their frequencies.

$$\begin{tikzcd}
{[\,]} \arrow[r, dash] & {[1]} \arrow[r, dash] \arrow[rdd, dash] \arrow[ddd, "p"] & {[12]} \arrow[d, "q"] \\
 & & {[14]} \\
 & & {[13]} \\
 & {[2]} \arrow[r, dash] \arrow[rdd, dash] \arrow[ddd, "p"] & {[23]} \arrow[d, "p^{-1}qp"] \\
 & & {[21]} \\
 & & {[24]} \\
 & {[3]} \arrow[r, dash] \arrow[rdd, dash] \arrow[ddd, "p"] & {[34]} \arrow[d, "p^{-2}qp^2"] \\
 & & {[32]} \\
 & & {[31]} \\
 & {[4]} \arrow[r, dash] \arrow[rdd, dash] & {[41]} \arrow[d, "p^{-3}qp^3"] \\
 & & {[43]} \\
 & & {[42]}
\end{tikzcd}$$

Figure 4.1: Orbits on 2-sequences under D_4

4.4.6 Consistency Checking

A consistency check is that the sum of the products of the length and the frequency is $P(n,r)$, where

$$P(n,r) = n \times (n-1) \times \ldots \times (n-r+1).$$

This is the total number of distinct r-sequences obtainable from n distinct elements.

4.5 Implementation Using ISOM

ISOM is a package for isomorphism testing developed at Concordia University. It uses the symmetry group to cut down the size of a search. The symmetry group of a combinatorial object can be very large. In ISOM, a permutation group is represented in a compact form. For a detailed description of ISOM, see **[13]**.

The last appendix contains a description for the ISOM routines called in our programs. Before calling the routine *jerrum* to generate a group, we create a null group using the function *build_null_gp* to store the group, to specify the set of generators, and to specify the number of generators. The order of the group can be obtained using the routine *gporder*.

A symmetric group can be generated using the function *symmetric_gp*. The routine *find_certificate* uses this symmetric group and the group created by the routine *jerrum* to recursively generate sequences in lex order. For example, Figure 4.2 shows the sequences that will be generated by *find_certificate* when given S_4 and a null group of degree 4. This generates a list of all possible sequences from Ω_4.

[1], [12], [123], [1234], [124], [13], [134], [14],
[2], [21], [213], [2134], [214], [23], [234], [24],
[3], [31], [312], [3124], [314], [32], [324], [34],
[4], [41], [412], [4123], [413], [42], [423], [43].

Figure 4.2: All possible sequences from Ω_4

Lastly, the routine *find_orbit* enables us to find $|\alpha_i^{G_{(i-1)}}|$ as required in Theorem 4.3.

4.6 Output

Figure 4.3 shows the output file corresponding to D_4. Entry "2^{1}" indicates that there is one orbit of length 2.

```
Name = D4
  Generator    : (1,2,3,4)
                 (2,4)
  Order        : 8
  Parity       : Odd
  Imprimitive  : Yes
  Shape        : [1101]
  2-sets       : [2^{1},4^{1}]
  2-seqs       : [4^{1},8^{1}]
```

Figure 4.3: Characteristics of D_4

4.7 Re-formatted Output

Before the characteristics obtained for each group can be used to drive J. McKay's Galois program, they need to be further re-formatted and partitioned into files, depending on the degree, parity and imprimitivity of the group. At the same time, redundant generators will be thrown away to save storage space.

Figure 4.4 shows the result of re-formatting the output for D_4. This result will be written to an output file for odd imprimitive groups of degree 4.

```
{['D4', '8',
 {'(1,2,3,4)',
  '(2,4)'},
 [1, 1, 0, 1],
 [4, 8]
 [2, 4]],
```

Figure 4.4: Output for D_4

Chapter 5

Results

Upon execution of the programs described in the previous two chapters, a huge volume of output data was obtained. As it is not possible to display all the data here, only selected data are displayed in this thesis.

5.1 Distinguishing Permutation Isomorphic Classes of Groups

The invariants of each class are obtained. For groups of degree n, these invariants include the order, the imprimitivity, the parity, the shapes, and the orbit lengths of r-sets and r-sequences. For r-sets, r runs from 2 to $\lfloor \frac{n}{2} \rfloor$. For r-sequences, r runs from 2 to the minimum of $n - 2$ and 9.

In this section, we concentrate on sets of groups which are hard to distinguish. Appendix E shows the output for the groups discussed here.

5.1.1 Some Previously Indistinguishable Groups

Within the range tabulated in **[18]**, the orbit lengths and the parity of the group generally serve to determine the permutation isomorphism classes. The exceptions are now further studied and shown in Table 5.1 [1]. The group names in the Cayley library differ from those described in [4], see Table 5.1.

[1]A ? in an entry denotes that the set of groups are still indistinguishable

group name in [4]	*new group name*	*distinguishable by*
5T3, 5T5	t5n3, t5n5	3-sequences
6T9, 6T13	t6n9, t6n13	4-sequences
6T14, 6T16	t6n14, t6n16	4-sequences
8T26, 8T28, 8T30	t8n26, t8n28, t8n29	?
8T46, 8T47	t8n46, t8n47	6-sequences
9T29, 9T31	t9n30, t9n31	6-sequences
10T9, 10T10	t10n9, t10n10	?
10T11, 10T12	t10n11, t10n12	?
10T17, 10T19, 10T20	t10n18, t10n20, t10n21	?
10T36, 10T39	t10n37, t10n39	4-sequences
10T41, 10T43	t10n42, t10n43	8-sequences

Table 5.1: Groups which are previously indistinguishable

5.1.2 Groups of Degree Twelve to Fifteen

Based on only the parities and the orbit lengths of sets and sequences, all of the groups of degree 12 to 15 are distinguishable except the sets of groups shown in Table 5.2.

5.1.3 Distinguishing with Shapes

The existence of the shapes of a group can be used to distinguish some of the permutation isomorphic classes. This occurs if a class has a shape that is not in another class, while the latter also has a shape that is not in the former. Some of groups in Table 5.1 and Table 5.2 can then be distinguished as shown in Table 5.3.

5.1.4 Groups Still Indistinguishable

As concluded from the previous subsections, the groups shown in Table 5.4 are still indistinguishable using only the parity, orbit lengths and existence of the shapes.

5.2 Time Spent in Finding Orbit Lengths

Now, we consider the empirical complexity of our programs. The timing data shown in this section are expressed in milliseconds, and are obtained by running the programs

groups indistinguisable
t12n1, t12n5
t12n31, t12n40
t12n145, t12n154, t12n155
t12n152, t12n153
t12n168, t12n171, t12n172, t12n174
t12n180, t12n183
t12n196, t12n197
t12n209, t12n217
t12n210, t12n214
t12n212, t12n216
t12n221, t12n223, t12n225
t12n228, t12n229
t12n232, t12n234
t12n235, t12n237, t12n238
t12n240, t12n241
t12n248, t12n249
t12n262, t12n263, t12n267
t14n46, t14n47
t15n31, t15n32
t15n48, t15n51

Table 5.2: Groups of degree 12 to 15, indistinguishable by parity and orbit lengths

degree	*groups*
8	t8n28
10	t10n9, t10n10, t10n11, t10n12, t10n21
12	t12n155, t12n196, t12n197, t12n225, t12n232, t12n234, t12n237, t12n240, t12n241, t12n248, t12n249, t12n263
15	t15n31, t15n32, t15n48, t15n51

Table 5.3: Groups distinguishable by shapes

groups indistinguisable
t8n26, t8n29
t10n18, t10n20
t12n1, t12n5
t12n34, t12n40
t12n145, t12n154
t12n152, t12n153
t12n168, t12n171, t12n172, t12n174
t12n186, t12n183
t12n209, t12n217
t12n210, t12n214
t12n212, t12n216
t12n221, t12n223
t12n228, t12n229
t12n235, t12n238
t12n262, t12n267
t14n46, t14n47

Table 5.4: Groups still indistinguishable by parity, orbit lengths and shapes

in a DEC 5000/200 machine under ULTRIX V4.1 (Rev. 52). A theoretical analysis for the complexity of the programs is presented in the next chapter. This analysis helps to explain the patterns of the data presented in this section.

In the first two subsections, symmetric groups are chosen for studying the variations among different degrees. They are chosen as they have similar natures. They are odd and have all the shapes. Besides, they have only one orbit on each r-set and one orbit on each r-sequence, thus, complicated variations due to differences in their natures are eliminated.

In the last two subsections, all the transitive groups of degree 11 are chosen to study the variations among the groups. Degree 11 is chosen because there are only a total of 8 groups. Besides, this degree is sufficiently large so that orbits on 9-sequences will be computed. This is the maximum sequence length that our main program can handle.

	2-set	3-set	4-set	5-set	6-set	7-set
t2n1	0	-	-		-	-
t3n2	0	-	-	-		-
t4n5	0	-	-	-	-	-
t5n5	0	-	-	-	-	-
t6n16	0	0	-	-	-	-
t7n7	3	0	-	-	-	-
t8n50	0	3	7	-	-	-
t9n34	3	3	7	-	-	-
t10n45	3	7	11	15		-
t11n8	3	11	19	31	-	
t12n301	7	11	31	54	66	
t13n9	3	19	46	89	128	-
t14n63	7	23	66	144	231	277
t15n101	7	27	97	226	394	551

Table 5.5: Timing data for r-sets of symmetric groups in units of millisecond

5.2.1 Finding Orbit Lengths of r-sets of Symmetric Groups

Table 5.5 shows the time spent on finding all the orbit lengths of r sets of symmetric groups. It can be seen that, for symmetric groups, the time increases slowly with the degrees but quickly with the length, r, of the r-sets.

5.2.2 Finding Orbit Lengths of r-sequences for Symmetric Groups

We find that it only take us a few microseconds to compute the orbit length of each r-sequence under the action of a symmetric group.

5.2.3 Finding Orbit Lengths of r-sets for Groups of Degree 11

Table 5.6 shows the time spent on finding all the orbit lengths on r-sets for groups of degree 11. The time increases quickly with the lengths of the sets.

	2-set	3-set	4-set	5-set
t11n1	3	11	27	54
t11n2	3	11	31	66
t11n3	3	11	27	46
t11n4	3	7	23	39
t11n5	3	15	27	46
t11n6	3	11	27	42
t11n7	3	7	19	31
t11n8	3	11	19	31

Table 5.6: Timing data for r-sets for groups of degree 11 in units of millisecond

	2-seq	3-seq	4-seq	5-seq	6-seq	7-seq	8-seq	9-seq
t11n1	3	11	101	831	5753	32240	142252	468130
t11n2	0	7	54	468	3167	17565	76741	249964
t11n3	3	3	31	242	1581	8616	37122	119359
t11n4	3	0	23	156	996	5316	22639	71905
t11n5	0	0	7	31	203	1054	4456	14042
t11n6	0	3	0	3	23	121	492	1523
t11n7	3	0	0	3	0	3	0	3
t11n8	0	0	3	0	3	0	3	0

Table 5.7: Timing data for r-sequences for groups of degree 11 in units of millisecond

5.2.4 Finding Orbit Lengths of r-sequences for Groups of Degree 11

Table 5.7 shows the time spent on finding all the orbit lengths on r-sequences for groups of degree 11. The last two entries are for the alternating and symmetric groups. In these cases, all the r-sequences are in one orbit. Since there is no need to generate other orbits, the timing data for these two groups are substantially smaller than the others. In Table 5.8, the number of orbits on r-sequences are given. We note that the timing data seem to be linearly proportional to the number of orbits.

	2-seq	3-seq	4-seq	5-seq	6-seq	7-seq	8-seq	9-seq
t11n1	10	90	720	5040	30240	151200	604800	1814400
t11n2	5	45	360	2520	15120	75600	302400	907200
t11n3	2	18	144	1008	6048	30240	120960	362880
t11n4	1	9	72	504	3024	15120	60480	181440
t11n5	1	2	12	84	504	2520	10080	30240
t11n6	1	1	1	7	42	210	840	2520
t11n7	1	1	1	1	1	1	1	1
t11n8	1	1	1	1	1	1	1	1

Table 5.8: Number of orbits on r-sequences for groups of degree 11

5.3 Reducing the Number of Generators

Table 5.9 shows the number of redundant generators found in the Cayley library.

degree	*total no. of groups*	*no. of groups with k redundant generators*		
		$k = 1$	$k = 2$	$k = 3$
2	1	0	0	0
3	2	0	0	0
4	5	0	0	0
5	5	0	0	0
6	16	0	0	0
7	7	0	0	0
8	50	0	0	0
9	31	0	0	0
10	45	0	0	0
11	8	2	0	0
12	301	17	0	0
13	9	0	0	0
14	63	25	0	0
15	104	15	4	1

Table 5.9: Number of redundant generators found

Chapter 6

Evaluation

In this chapter, a discussion of the time complexity of the four sections of the main process is presented. This analysis helps to explain the trends in the timing data presented in the previous chapter. In addition, discussion of the limitation and capability of the programs, as well as suggestions for further improvement are also presented. Finally, a conclusion is drawn.

6.1 Complexity

The programs have different sections which vary in time complexity. The time required for initializing tables for the number of combinations, number of permutations, and number of partitions is very small. There is a binary search tree which stores the orbit lengths. However, there are usually less than four different values of orbit lengths in a group, thus, the time required for the search and insert operations on this tree is also very small.

Before describing the complexity of the main sections of the programs, we define some notation as below:

k	the number of generators
b	the number of orbits
s	the number of shapes

6.1.1 Checking Parity and Reducing the Number of Generators

Checking parity requires counting the number of commas in the generators only. This takes only $O(n)$ time.

In the creation of a group from the given generators, the routine *jerrum* of ISOM is used. To check for redundant generators, all the subsets of the given set of generators are checked using *jerrum*. The total cost here is $O(2^k) \times O(jerrum)$, where k is not greater than 5. At present, $O(jerrum)$ is $O(n^5)$.

6.1.2 Partition and Shape

Before encoding the shapes, we have to initialize a linear array to hold the information. This takes $O(p(n))$ time. The cost to encode each of the s shapes is at most $O(n)$, using the mapping ϕ. Thus, the total complexity is

$$O(p(n)) + O(ns).$$

6.1.3 Finding the Orbit Length of an r-set

To find the orbit length of an r-set, we use the procedure *extend_permutation* to generate sets. In the method of expanding horizon, we need to apply the k generators to each set in the orbit, and generate the image in $O(n)$ time. Since each of the $C(n,r)$ sets is seen once, the complexity for this part is

$$O(C(n,r) \times nk).$$

The time needed to generate new r-sets as described in Section 4.3.1 is not included.

6.1.4 Finding the Orbit Length of an r-sequence

To find the orbit length of an r-sequence, we use the procedure *extend_permutation* to generate sequences. For each of the b orbits, we need to multiply the r values of orbit lengths together. However, for transitive groups, the first orbit length is n. For the remaining orbit lengths, we use the procedure *find_orbit*, which has a complexity of $O(n^2)$. Thus, the complexity for this part is

$$O(n^2rb).$$

One can show that this complexity also bounds the cost of generating the r sequences. Hence, the total complexity is still $O(n^2rb)$. For fixed n and r, this complexity is linear in b, which explains the linear relationship between the data in Table 5.7 and Table 5.8.

6.2 Limitation and Capability

Our main program is written to run on a machine whose word length is 32 bits. Some of the computations are limited by the size of an unsigned long integer, which is about $2^{32} \approx 4.3 \times 10^9$. At present, the maximum degree of a group that the programs can handle is 15.

In fact, the programs can work for groups of any degree n, as long as the following restrictions are observed:

1. for r-set, $C(n,r)$ must be less than 2^{32}, and
2. for r-sequence, $P(n,r)$ must be less than 2^{32}.

Moreover, the array size for encoding shapes must be increased to $p(n)$. The second restriction also explains why, for r-sequences, the current program restrict r to at most 9, as $P(15, 10)$ is greater than 2^{32}.

In addition, the maximum number of generators for a group is now set to 20. This can be easily changed if the actual number of generators for a group is more than 20.

6.3 Further Improvement

Multi-length integers can be used to solve the problem of the limited size of a 32 bit integer.

Instead of computing the orbit lengths of the elements in an r sequence after the sequence has been completely generated, we can compute these lengths incrementally. This will reduce the complexity of computing the orbit lengths of r-sequences.

In the case that the orbit lengths for a group are not distinct, we can further analyze the orbits, taking each set or sequence as an element. This is particularly informative if either the order of the orbit stablilizer is smaller than the order of the group, or the orbit length is not greater than 15, the maximum degree that our programs can handle.

To better identify all permutation isomorphism classes of groups, more invariants are needed. These can include the block systems, and whether the action induced by G on an orbit is faithful.

Some computations such as finding the image of each element in a permutation, finding the orbits on individual elements, and the process using the method of expanding horizon can be done in parallel. These computations may be carried out in the synchronous mode SIMD, which can be implemented using C/Paris on the Connection Machine.

It will help maintenance of the programs if the preprocessing part using Cayley can be integrated into the main process. With this, we do not need to preprocess each file in the Cayley library to obtain an intermediate output before being processed by the main program. Instead, we can then obtain the list of invariants directly from the Cayley library just by running the main program.

6.4 Conclusion

The results obtained for groups of degree 3 to 11 agree with those obtained in [18]. This supports our confidence in the correctness of the programs involved in the research.

The method of expanding horizon, coupled with the registration technique to identify new sets, is very effective for computing the orbits on r-sets. Furthermore, the theorem on calculating the number of sequences in an orbit is very useful for computing the orbits on r-sequences. Groups, including those of degree 12 to 15, can now be studied in more detail.

The invariants obtained for each group enhance distinguishing permutation isomorphism classes of groups. More Galois groups may thus be determined for studying the solution of polynomial equations of higher degrees.

Cayley is very useful for finding conjugacy classes of permutation groups and ISOM is a powerful tool for studying the actions of transitive permutation groups of small degrees.

Bibliography

[1] Atkinson, M.D., *An algorithm for finding the blocks of a permutation group,* Math. Comp., Vol. 29, pp.911-913, 1975.

[2] Baumslag, Benjamin, *Schaum's Outline of Theory and Problems of Group Theory,* McGraw Hill, 1968.

[3] Beckenbach, Edwin F., *Applied Combinatorial Mathematics,* John Wiley and Sons, 1964.

[4] Butler, G. and McKay, J., *The transitive groups of degree up to eleven,* Comm. Alg. 11(8), 863-911 (1983).

[5] Butler, G. and McKay, J., *The transitive groups of degree up to fifteen,* (in preparation).

[6] Butler, G., *Using Cayley Under Unix - Beginner's Guide,* CICMA, Concordia University, Montreal, Canada, 1992.

[7] Butler, G., *An Inductive Schema for Computing Conjugacy Classes in Permutation Groups,* technical report 394, Basser Department of Computer Science, University of Sydney, 1990.

[8] Butler, G., *Fundamental Algorithms for Permutation Groups,* Lecture Notes in Computer Science, Vol. 559, Springer-Verlag, N.Y., 1991.

[9] Cannon, John, *A Language for Group Theory,* Department of Pure Mathematics, University of Sydney, Australia, 1987.

[10] Hall, M., Jr., *Combinatorial Theory,* Blaisdell, Waltham MA, 1967.

[11] Hardy, G.H. and Wright, E.M., *An Introduction to the Theory of Numbers*, 5th edition, pp.273-296, Oxford University Press, 1979.

[12] Kernighan, Brian W. and Pike, Rob, *The UNIX Programming Environment*, Prentice-Hall, Englewood Cliffs, N.J., 1984.

[13] Lam, Clement W.H., *Computational Combinatorics - A Maturing Experimental Approach*, course notes, Concordia University, 1990.

[14] Ledermann, W., *Introduction to Group Theory*, Longman, 1976.

[15] Lehmer, D.H., *The machine tools of combinatorics*, Applied Combinatorial Mathematics, ed. Beckenbach, E.F., pp.5-31, 1964.

[16] Mattman, Thomas, *Computation of Galois Groups over Function Fields*, Master thesis, McGill University, 1992.

[17] McKay, J. and Regener, E., *Transitivity sets, Algorithm 482*, Comm. ACM 17:8, p.470, 1974.

[18] McKay, J. and Regener, E., *Actions of permutation groups on r-sets*, Comm. Alg. 13(3), pp.619-630, 1985.

[19] McKay, J., *Advances in Computational Galois Theory*, Computers in Algebra, ed. Tangora, Marcel Dekker, Vol. 111, pp.99-101, 1988.

[20] Niven, Ivan and Zuckerman, H.S., *An Introduction to the Theory of Number*, John Wiley and Sons, N.Y., 5th edition, pp.267-288, 1980.

[21] Royle, G.F., *The transitive groups of degree twelve*, J. Symbolic Comp., Vol. 4, pp.255-268, 1987.

[22] Sims, C.C., *Computational methods in the study of permutation groups*, in John Leech, *Computational Problems in Abstract Algebra*, pp.169-183, Pergamon, Elmsford, N.Y., 1970.

[23] Soicher, L.H. and McKay, J., *Computing Galois groups over the rationals*, J. Number Theory Vol. 20 (1985), 273-281.

[24] Stewart, Ian, *Galois Theory*, Chapman and Hall, N.Y., 2nd edition, 1989.

[25] Williamson, S.G., *Combinatorics for Computer Science*, Computer Science Press, 1985.

Appendix A

Preprocessing Program

Attached is the CAYLEY program, due to G. Butler, that performs the work of preprocessing described in Chapter 3. If the program is in a file called *lib.in*, the group *t4n3* is proprocessed using the UNIX command

sed s/LL/t4n3/ lib.in | cayley > t4n3.pre

```
set workspace = 500000;
set format = true;

"the inductive schema for conjugacy classes of elements"
set libfile ='/home/faculty/butler/cayproc/rathom';
library cladt;
library clcomposite;
library clutilities;
library sylanalyse;
library isnewrat;
library addpclass;
library clprime;
library rathomcl;

"routines to analyse properties of a permutation group"
set libfile='/home/faculty/butler/batch/trngps';
library gpanalyse;

"Print the group information, especially class information
about a group"

"gpinfo is a sequence containing
  1. seq( order(G) )
  2. seq of no. blocks in each minimal block system
  3. seq( no. classes )
  4. seq(  seq of cycle shapes - sorted,
           seq of no. elements with given cycle shape,
           seq of no. classes  with given cycle shape )
  5. seq of group names as strings (indicating kernel and image)
  6. seq( G )
"
```

```
procedure printgp(  gpinfo );

   print 'SEQ ', gpinfo[1][1];
   bls =  gpinfo[2];
   blth = length( bls );
   if blth eq 0 then
      print 'SEQ 0';
   else
      print 'SEQ 1';
   end;

   clss = gpinfo[4];
   sss = clss[1]; sselts = clss[2]; sscls = clss[3];

   "convert cycle shapes for encoding"
   llsss = empty;

   print 'SEQ ', length( sss);
   for i = 1 to length( sss ) do
     oldcl = sss[ i ];
     newcl = empty;
     for j = 1 to length( oldcl ) by 2 do
        cyclth = oldcl[ j ];
        numcycs = oldcl[ j + 1 ];
        newcl = concatenate( newcl, conseq( cyclth, numcycs ) );
     end;
     llsss[ i ] = newcl;
   end;

   for i = 1 to length( sss ) do
      print llsss[i];
   end;

end; "printgp"

finish;

set  ibfile='/home/faculty/butler/cayley/caylibs/trngps';

library LL;

print 'SEQ ', ngenerators(G);
for each x in generators(G) do
   print eltseq(x);
end;
for each x in generators(G) do
   print 'CYC', x;
end;
gpanalyse (G,  'LL'; gpinfo);
printgp (gpinfo);
quit;
```

Appendix B

Editing Commands

The following *sed* editing commands are used for postprocessing the Cayley output.

```
/SEQ/ s/,//g
/SEQ/ s/(//
/SEQ/ s/)//
/SEQ/ s/SEQ//p
/CYC/ s/CYC//p
```

If the above commands are stored in the file *sed.in*, the preprocessed file *t4n3.pre* is further processed using the UNIX command

sed -n -f sed.in > t4n3.out

Appendix C

Input to Main Program

The following is the input file to the main program that is used to obtain the output for the groups discussed in Section 5.1. These are the groups which are difficult to be distinguished from one another. The first column indicates the file containing the Cayley output after preprocessing; while the second column indicates the group name associated with the data in that file. In this file, as we just use the file names as the group names, these columns are the same.

```
degree
5
t5n3  t5n3
t5n5  t5n5
degree
6
t6n9   t6n9
t6n13  t6n13
t6n14  t6n14
t6n16  t6n16
degree
8
t8n26  t8n26
t8n28  t8n28
t8n29  t8n29
t8n46  t8n46
t8n47  t8n47
degree
9
t9n30  t9n30
t9n31  t9n31
degree
10
t10n9   t10n9
t10n10  t10n10
t10n11  t10n11
t10n12  t10n12
t10n18  t10n18
```

```
t10n20  t10n20
t10n21  t10n21
t10n37  t10n37
t10n39  t10n39
t10n42  t10n42
t10n43  t10n43
degree
12
t12n1  t12n1
t12n5  t12n5
t12n34  t12n34
t12n40  t12n40
t12n145  t12n145
t12n152  t12n152
t12n153  t12n153
t12n154  t12n154
t12n155  t12n155
t12n168  t12n168
t12n171  t12n171
t12n172  t12n172
t12n174  t12n174
t12n180  t12n180
t12n183  t12n183
t12n196  t12n196
t12n197  t12n197
t12n209  t12n209
t12n210  t12n210
t12n212  t12n212
t12n214  t12n214
t12n216  t12n216
t12n217  t12n217
t12n221  t12n221
t12n223  t12n223
t12n225  t12n225
t12n228  t12n228
t12n229  t12n229
t12n232  t12n232
t12n234  t12n234
t12n235  t12n235
t12n237  t12n237
t12n238  t12n238
t12n240  t12n240
t12n241  t12n241
t12n248  t12n248
t12n249  t12n249
t12n262  t12n262
t12n263  t12n263
t12n267  t12n267
degree
14
t14n46  t14n46
t14n47  t14n47
degree
15
t15n31  t15n31
t15n32  t15n32
t15n48  t15n48
t15n51  t15n51
```

Appendix D

Main Program

The following C program performs the work described in Chapter 4. If the name of this file is *main.c*, which is compiled to *main*, and the input file to it is *in*, output file, *out*, is obtained using the UNIX command

main <in >out

In addition, files containing timing data, *time.t9* and *time.q9*, are also produced.

The program is run in a DEC 5000/200 machine under ULTRIX V4.1 (Rev. 52). In this machine, ISOM of version 1.00 written in C is included by the header file *groupdcl.h*.

```
#include <stdio.h>
#include <sys/time.h>
#include <sys/resource.h>
#include "groupdcl.h"


#define max_degree 15
#define max_comb 6435 /* 15 C 7 */
#define max_part 176 /* part[15][15] */
#define max_nof_gen 20
#define MINRSEQ 2
#define MAXRSEQ 9

typedef struct nodeT {
  unsigned long size,freq;
  struct nodeT *left, *right;
} ;


short combination[max_degree+1][max_degree+1];
unsigned long permutation[max_degree+1][max_degree+1];
ptr_to_perm_gp sym_gp, group;
ptr_to_permmatrix generators;
```

```
ptr_to_permvect temppvect,poppvect,orbt;
long gp_order;
double rgp_order;
short degree,r,num_gen;
short part[max_degree+1][max_degree+1];
boolean shape[max_part];
struct nodeT *root;
short sizecount,first;
boolean seen[max_comb+1];
unsigned long set_index;
FILE *fptr,*ftime1,*ftime2;
short odd,blth,nof_shape;
struct timeval oldclock,newclock;
long cputime;
struct rusage ru;
short maxng;
double maxorder;
short non_redun[max_nof_gen+1],A[max_nof_gen+1];
ptr_to_perm_gp sym_redun,auto_redun,group_redun;
ptr_to_permmatrix gen;
short nof_1;
unsigned long total_size;


/************************** binary tree ********************************/
void init_tree(struct nodeT *node)
{
  if (node==NULL) return;
  init_tree(node->left);
  node->freq = 0;
  init_tree(node->right);
}

void print_table(struct nodeT *node)
{
  if (node==NULL) return;
  print_table(node->left);
  if ((node->freq)>0) {
    if (first==1) first = 0;
    else printf(",");
    printf("%u^{%u}", node->size, node->freq);
  }
  print_table(node->right);
  total_size += (node->size) * (node->freq);
}

void search_insert(unsigned long x, struct nodeT **node)
{
  struct nodeT *WITH;
  if (*node == NULL) {
    *node = (struct nodeT *)malloc(sizeof(struct nodeT));
    WITH = *node;
    WITH->size = x;
    WITH->freq = 1;
    WITH->left = NULL;
    WITH->right = NULL;
    return;
  }
```

```
    /* not found, insert */
    if (x < (*node)->size) {
      search_insert(x, &(*node)->left);
      return;
    }
    if (x > (*node)->size) {
      search_insert(x, &(*node)->right);
      return;
    }
    /* found */
    WITH = *node;
    (WITH->freq)++;
}

/************************* initializations *******************************/
void init_combination()
{
    short i,j;
    for (i=0; i<=max_degree; i++) {
      combination[i][1] = i;
      for (j=2; j<=i-1; j++)
        combination[i][j] = combination[i-1][j-1] + combination[i-1][j];
      combination[i][i] = 1;
      for (j=i+1; j<=max_degree; j++)
        combination[i][j] = 0;
    }
}

void init_permutation()
{
    short i,j;
    for (i=1; i<=max_degree; i++) {
      permutation[i][1] = i;
      for (j=2; j<=i; j++)
        permutation[i][j] = permutation[i][j-1] * (i-j+1);
    }
}

void init_part()
{
    short i,j;
    for (j=0; j<=max_degree; j++) part[0][j] = 1;
    for (i=1; i<=max_degree; i++) {
      part[i][0] = 0;
      for (j=1; j<i; j++) part[i][j] = part[i][j-1] + part[i-j][j];
      part[i][i] = part[i][i-1] + 1;
      for (j=i+1; j<=max_degree; j++) part[i][j] = part[i][i];
    }
}

void initialization()
{
    init_combination();
    init_permutation();
    init_part();
    root  = NULL;
    search_insert(12, &root);
```

```
  sym_gp = NULL;
  group  = NULL;
  sym_redun = NULL;
  auto_redun = NULL;
  group_redun = NULL;
}


/************************* create group ***************************/
short comma_count(char *str)
{
  short i=0,n=0;
  while (str[i]!='\0') {
    if (str[i]==',') n += 1;
    i++;
  }
  return n;
}

void process()
{
  short ii;
  group_redun = build_null_gp (group_redun, degree);
  jerrum (group_redun, gen, nof_1);
  gporder (group_redun, &rgp_order, &gp_order);
  if ( (rgp_order/maxorder>0.75) & (nof_1<maxng) ) {
    maxng    = nof_1;
    for (ii=1; ii<=num_gen; ii++) non_redun[ii] = A[ii];
  }
}

void generate(short index)
{
  short ii;
  A[index] = 1;
  nof_1++;
  for (ii=1; ii<=degree; ii++)
    gen[nof_1][ii] = generators[index][ii];
  if (nof_1<maxng) {
    if (index<num_gen) generate(index+1);
    else process();
  }

  A[index] = 0;
  nof_1--;
  if (nof_1<maxng) {
    if (index<num_gen) generate(index+1);
    else if (nof_1>0) process();
  }
}

void create_group()
{
  /*reads in the generators and call Jerrum's algorithm to
    create the group.*/
  short j,k,count;
  char genstr[60];
  fscanf(fptr, "%hd", &num_gen);
  for (j=1; j<=num_gen; j++) {
    readpvect(fptr,generators[j],degree);
    for (k=1; k<=degree; k++)
      gen[j][k] = generators[j][k];
  }
  group = build_null_gp(group, degree);
  jerrum(group, gen, num_gen);
  gporder (group, &maxorder, &gp_order);
  maxng = num_gen;
  for (j=1; j<=num_gen; j++) non_redun[j] = 1;
  if (num_gen>1)
    generate(1);

  fscanf (fptr,"%s", genstr);
  if (genstr[0]=='[') fscanf (fptr, "%s", genstr);
  odd = (odd | comma_count(genstr)%2);
  if (non_redun[1]==1)
    printf ("  Generator   : %s\n", genstr);
  else
    printf ("  Generator   : *%s\n", genstr);
  for (j=2; j<=num_gen; j++) {
    fscanf(fptr,"%s", genstr);
    if (genstr[0]=='[') fscanf(fptr, "%s", genstr);
    odd = (odd | comma_count(genstr)%2);
    if (non_redun[j]==1)
      printf("                 %s\n", genstr);
    else
      printf("                 *%s\n", genstr);
  }
} /*create_group*/

/******************************* shape ***********************************/
void find_shape()
{
  short i,n,e,next;
  fscanf(fptr, "%hd", &nof_shape);
  for (i=1; (i<part[degree][degree]); i++) shape[i] = FALSE;
  for (i=1; i<=nof_shape; i++) {
    e = 0;
    n = degree;
    while (n>0) {
      fscanf(fptr, "%hd", &next);
      e += part[n][next-1];
      n -= next;
    }
    shape[e] = 1;
  }
  printf("  Shape       : [");
  for (i=1; (i<part[degree][degree]); i++) {
    if ( (i%50==1) & (i!=1) ) printf("\n                 ");
    if (shape[i]) printf("1"); else printf("0");
  }
  printf("]\n");
}


/******************************* r-set ***********************************/
unsigned long colex_order(ptr_to_permvect alpha)
{
  short i,sum=alpha[1];
  for (i=2; i<=r; i++)
    sum += combination[alpha[i]-1][i];
  return sum;
}

void colex(long set_r, ptr_to_permvect alpha)
{
  short i=1,j;
  for (j=1; j<=degree; j++) {
    if ((unsigned)j < 32 && ((1L<<j) & set_r) != 0) {
      alpha[i] = j;
      i++;
    }
  }
}

void find_set_orbit(ptr_to_permvect choice)
{
  short j,k;
  long set_image;
  long genset_stack[max_comb+1];
  unsigned long nof_genset=1;
  unsigned long nof_image=1;

  genset_stack[1] = 0L;
  for (j=1; j<=r; j++)
    genset_stack[1] |= 1L<<choice[j];
  while (nof_genset>0) {
    colex(genset_stack[nof_genset], poppvect);
    nof_genset--;
    for (j=1; j<=num_gen; j++) {
      set_image = 0;
      for (k=1; k<=r; k++)
        set_image |= 1L<<generators[j][poppvect[k]];
      colex(set_image, temppvect);
      set_index = colex_order(temppvect);
      if (!seen[set_index]) {
        seen[set_index] = TRUE;
        nof_image++;
        nof_genset++;
        genset_stack[nof_genset] = set_image;
      }
    }
  }
  search_insert(nof_image, &root);
}

comprestype compare_set(ptr_to_permvect choice, short depth)
{
    if (depth==1) {
      return indifferent;
    } else {
      if (choice[depth] < choice[depth-1]) {
        return worse;
      } else {
        if (depth<r)
          return indifferent;
        else {
          set_index = colex_order(choice);
          if (!seen[set_index]) {
            seen[set_index] = TRUE;
            find_set_orbit(choice);
          }
          return worse;
        }
      }
    }
}   /*compare_set*/

void find_set(short rlength)
{
      short i;
      r = rlength;
      init_tree(root);
      for (i=1; i<=combination[degree][r]; i++) seen[i] = FALSE;
      find_certificate(sym_gp, group, compare_set, 0);
}


/************************** r-sequence ***************************/
void Orbit_Length(short *count, short pt, ptr_to_permvect orbt)
{
  /*calculate the length of a cycle containing the point pt. returns answer
    in count*/
  short temp;

  *count = 1;
  temp = pt;
  while (pt != orbt[temp]) {
    (*count)++;
    temp = orbt[temp];
    if (*count > degree)
      printf("problem\n");
  }
}  /*orbit_length*/

void Orbit_Size(unsigned long *size, ptr_to_permvect choice)
{
  short count, i, j;
  ptr_to_permmatrix U=NULL;

  *size = degree;
  for (i = 2; i <= r; i++) {
    for (j = 1; j <= degree; j++)
      orbt[j] = 0;
    find_orbit(group, choice[i], i - 1, orbt, U);
    Orbit_Length(&count, choice[i], orbt);
    *size *= count;
  }
} /*orbit_size*/

comprestype compare_seq(ptr_to_permvect choice, short depth)
{
  unsigned long size;

  if (depth < r)
    return indifferent;
  else {
    Orbit_Size(&size,choice);
    search_insert(size, &root);
    return worse;
  }
} /*compare*/

void find_seq(short rlength)
{
    r = rlength;
    init_tree(root);
    find_certificate(sym_gp, group, compare_seq, 0);
}


/************************** M A I N ****************************/
int main()
{
  short j,k,maxrset,maxrseq;
  char name[40],gname[40];

  initialization();
  safeopen (ftime1, "time.t9", "w");
  safeopen (ftime2, "time.q9", "w");
  fprintf (ftime1, "Group     Time(ms)\n-----     ---------");
  fprintf (ftime2, "Group     Time(ms)\n-----     ---------");
  temppvect = allocatepv(max_degree);
  poppvect = allocatepv(max_degree);
  orbt      = allocatepv(max_degree);
  generators = allocatepm(max_degree);
  gen        = allocatepm(max_degree);
  for (j=1; j<=max_nof_gen; j++) {
    generators[j] = allocatepv(max_degree),
    gen[j]        = allocatepv(max_degree);
  }
  while (scanf("%s", name)==1) {
    if (strcmp(name, "degree")==0) {
      scanf("%hd", &degree);
      maxrset = degree/2;
      if (maxrset<2) maxrset=2;
      if (degree-2>MAXRSEQ) maxrseq = MAXRSEQ;
      else                  maxrseq = degree-2;
      if (maxrseq<2) maxrseq=2;
      printf("DEGREE %d\n\n", degree);
      if (sym_gp != NULL) {
```

```
    disposepg (sym_gp);
    sym_gp = NULL;
  }
  sym_gp = symmetric_gp(sym_gp,degree);
  scanf("%s",name);
}
scanf ("%s", gname);
printf("Name = %s\n", gname);
safeopen (fptr, name, "r");
odd = 0;
create_group ();
fscanf (fptr, "%s", name);
printf ("  Order        : %s\n", name);
if (odd==0) printf ("  Parity       : Even\n");
else         printf ("  Parity       : Odd\n");
fscanf(fptr, "%hd", &blth);
printf ("  Imprimitive : ");
if (blth==0) printf ("No\n") ; else printf ("Yes\n");
find_shape();
fprintf (ftime1, "\n%-10s", gname);
fprintf (ftime2, "\n%-10s", gname);
getrusage(RUSAGE_SELF,&ru);
oldclock.tv_sec  = (ru.ru_utime).tv_sec;
oldclock.tv_usec = (ru.ru_utime).tv_usec;
for (j=2; j<=maxrset; j++) {
  find_set(j);
  first = 1;
  printf("  %d-sets       : [", j);
  total_size = 0;
  print_table(root);
  if (total_size != combination[degree][j]) {
    printf ("\ninconsistent!\n");
    exit(1);
  }
  printf ("]\n");
getrusage(RUSAGE_SELF,&ru);
newclock.tv_sec  = (ru.ru_utime).tv_sec;
newclock.tv_usec = (ru.ru_utime).tv_usec;
cputime = 1000 * (newclock.tv_sec-oldclock.tv_sec) +
       0.001 * (newclock.tv_usec-oldclock.tv_usec);
fprintf (ftime1, "%8d", cputime);
oldclock.tv_sec = newclock.tv_sec;
oldclock.tv_usec = newclock.tv_usec;
}
for (j=MINRSEQ; j<=maxrseq; j++) {
  find_seq(j);
  first = 1;
  printf("  %d-seqs       : [", j);
  total_size = 0;
  print_table(root);
  if (total_size != permutation[degree][j]) {
    printf ("\ninconsistent!\n");
    exit(1);
  }
  printf("]\n");
getrusage(RUSAGE_SELF,&ru);
```

```
    newclock.tv_sec  = (ru.ru_utime).tv_sec;
    newclock.tv_usec = (ru.ru_utime).tv_usec;
    cputime = 1000 * (newclock.tv_sec-oldclock.tv_sec) +
           0.001 * (newclock.tv_usec-oldclock.tv_usec);
    fprintf (ftime2, "%8d", cputime);
    oldclock.tv_sec = newclock.tv_sec;
    oldclock.tv_usec = newclock.tv_usec;
    }
    printf ("\n");
    fclose(fptr);
  }
  fclose (ftime1);
  fclose (ftime2);
  disposepm(generators, max_nof_gen);
  disposepm(gen, max_nof_gen);
  disposepv(temppvect);
  disposepv(poppvect);
  disposepv(orbt);
  exit(0);
}

/* End. */
```

Appendix E

Output for Difficult Groups

The following shows the output obtained, as described in the previous appendix, for the groups discussed in Section 5.1. These are the groups which are difficult to be distinguished from one another.

```
DEGREE 5

Name = t5n3
  Generator    : (2,5,3,4)
                 (1,4,2,3)
  Order        : 20
  Parity       : Odd
  Imprimitive  : No
  Shape        : [010011]
  2-sets       : [10^{1}]
  2-seqs       : [20^{1}]
  3-seqs       : [20^{3}]

Name = t5n5
  Generator    : (1,2,3,4,5)
                 (1,2)
  Order        : 120
  Parity       : Odd
  Imprimitive  : No
  Shape        : [111111]
  2-sets       : [10^{1}]
  2-seqs       : [20^{1}]
  3-seqs       : [60^{1}]

DEGREE 6

Name = t6n9
  Generator    : (1,6,3,4,2,5)
                 (2,3)(4,5)
  Order        : 36
  Parity       : Odd
  Imprimitive  : Yes
  Shape        : [0111010001]
  2-sets       : [6^{1},9^{1}]
  3-sets       : [2^{1},18^{1}]
  2-seqs       : [12^{1},18^{1}]
  3-seqs       : [12^{1},36^{3}]
  4-seqs       : [36^{10}]

Name = t6n13
  Generator    : (1,2,3)(5,6)
                 (1,4)(2,5,3,6)
  Order        : 72
  Parity       : Odd
  Imprimitive  : Yes
  Shape        : [1111110101]
  2-sets       : [6^{1},9^{1}]
  3-sets       : [2^{1},18^{1}]
  2-seqs       : [12^{1},18^{1}]
  3-seqs       : [12^{1},36^{3}]
  4-seqs       : [36^{4},72^{3}]

Name = t6n14
  Generator    : (2,5,4,6,3)
                 (1,4,2,3,6,5)
  Order        : 120
  Parity       : Odd
  Imprimitive  : No
  Shape        : [0110011011]
  2-sets       : [15^{1}]
  3-sets       : [20^{1}]
  2-seqs       : [30^{1}]
  3-seqs       : [120^{1}]
  4-seqs       : [120^{3}]

Name = t6n16
  Generator    : (1,2,3,4,5,6)
                 (1,2)
  Order        : 720
  Parity       : Odd
  Imprimitive  : No
  Shape        : [1111111111]
  2-sets       : [15^{1}]
  3-sets       : [20^{1}]
  2-seqs       : [30^{1}]
  3-seqs       : [120^{1}]
  4-seqs       : [360^{1}]

DEGREE 8

Name = t8n26
  Generator    : (1,2)(3,4)(5,7,6,8)
                 (1,5,3,8,2,6,4,7)
                 (1,6,3,8,2,5,4,7)
  Order        : 64
  Parity       : Odd
  Imprimitive  : Yes
  Shape        : [0111000001010100000001]
  2-sets       : [4^{1},8^{1},16^{1}]
  3-sets       : [8^{1},16^{1},32^{1}]
  4-sets       : [2^{1},4^{1},16^{2},32^{1}]
  2-seqs       : [8^{1},16^{1},32^{1}]
  3-seqs       : [16^{3},32^{3},64^{3}]
  4-seqs       : [16^{3},32^{3},64^{24}]
  5-seqs       : [64^{105}]
  6-seqs       : [64^{315}]

Name = t8n28
  Generator    : (1,8)(2,7)(3,5,4,6)
                 (1,6,4,7,2,5,3,8)
  Order        : 64
  Parity       : Odd
  Imprimitive  : Yes
  Shape        : [010100000011010000001]
  2-sets       : [4^{1},8^{1},16^{1}]
  3-sets       : [8^{1},16^{1},32^{1}]
  4-sets       : [2^{1},4^{1},16^{2},32^{1}]
  2-seqs       : [8^{1},16^{1},32^{1}]
  3-seqs       : [16^{3},32^{3},64^{3}]
  4-seqs       : [16^{3},32^{3},64^{24}]
  5-seqs       : [64^{105}]
  6-seqs       : [64^{315}]

Name = t8n29
  Generator    : (1,2)(5,7)(6,8)
                 (1,7)(2,8)(3,6,4,5)
  Order        : 64
  Parity       : Odd
  Imprimitive  : Yes
  Shape        : [011100000101010000000]
  2-sets       : [4^{1},8^{1},16^{1}]
  3-sets       : [8^{1},16^{1},32^{1}]
  4-sets       : [2^{1},4^{1},16^{2},32^{1}]
  2-seqs       : [8^{1},16^{1},32^{1}]
  3-seqs       : [16^{3},32^{3},64^{3}]
  4-seqs       : [16^{3},32^{3},64^{24}]
  5-seqs       : [64^{105}]
  6-seqs       : [64^{315}]

Name = t8n46
  Generator    : (1,4,2,3)(6,7)
                 (1,7,2,8,4,6)(3,5)
  Order        : 576
  Parity       : Even
  Imprimitive  : Yes
  Shape        : [010110110010010000100]
  2-sets       : [12^{1},16^{1}]
  3-sets       : [8^{1},48^{1}]
  4-sets       : [2^{1},32^{1},36^{1}]
  2-seqs       : [24^{1},32^{1}]
  3-seqs       : [48^{1},96^{3}]
  4-seqs       : [48^{1},192^{4},288^{3}]
  5-seqs       : [192^{5},576^{10}]
  6-seqs       : [576^{35}]

Name = t8n47
  Generator    : (1,7)(2,5)(3,6)(4,8)
                 (1,5,3,7,2,8,4,6)
  Order       : 1152
  Parity      : Odd
  Imprimitive : Yes
  Shape       : [111111110111110000101]
  2-sets      : [12^{1},16^{1}]
  3-sets      : [8^{1},48^{1}]
  4-sets      : [2^{1},32^{1},36^{1}]
  2-seqs      : [24^{1},32^{1}]
  3-seqs      : [48^{1},96^{3}]
  4-seqs      : [48^{1},192^{4},288^{3}]
  5-seqs      : [192^{5},576^{10}]
  6-seqs      : [576^{15},1152^{10}]

DEGREE 9

Name = t9n30
  Generator   : (1,4,2,5)(3,6)(7,8)
                (1,7,5,2,8,4,3,9,6)
  Order       : 648
  Parity      : Odd
  Imprimitive : Yes
  Shape       : [011010111010010000000001010001]
  2-sets      : [9^{1},27^{1}]
  3-sets      : [3^{1},27^{1},54^{1}]
  4-sets      : [18^{1},27^{1},81^{1}]
  2-seqs      : [18^{1},54^{1}]
  3-seqs      : [18^{1},108^{3},162^{1}]
  4-seqs      : [108^{4},216^{3},324^{6}]
  5-seqs      : [216^{10},324^{10},648^{15}]
  6-seqs      : [216^{10},648^{90}]
  7-seqs      : [648^{280}]

Name = t9n31
  Generator   : (1,7,3,8)(2,9)(4,5,6)
                (1,4,9,2,5,7)(3,6,8)
  Order       : 1296
  Parity      : Odd
  Imprimitive : Yes
  Shape       : [111111111110110100000001110001]
  2-sets      : [9^{1},27^{1}]
  3-sets      : [3^{1},27^{1},54^{1}]
  4-sets      : [18^{1},27^{1},81^{1}]
  2-seqs      : [18^{1},54^{1}]
  3-seqs      : [18^{1},108^{3},162^{1}]
  4-seqs      : [108^{4},216^{3},324^{6}]
  5-seqs      : [216^{10},324^{10},648^{15}]
  6-seqs      : [216^{10},648^{60},1296^{15}]
  7-seqs      : [648^{70},1296^{105}]

DEGREE 10

Name = t10n9
  Generator   : (1,6,5,10,4,9,3,8,2,7)
                (1,4)(2,3)(6,8)(9,10)
  Order       : 100
  Parity      : Odd
  Imprimitive : Yes
  Shape         : [0001100000000000000000001000001000000000001]
  2-sets        : [10^{2},25^{1}]
  3-sets        : [10^{2},50^{2}]
  4-sets        : [10^{1},25^{2},50^{3}]
  5-sets        : [2^{1},50^{5}]
  2-seqs        : [20^{2},50^{1}]
  3-seqs        : [20^{6},100^{6}]
  4-seqs        : [20^{12},100^{48}]
  5-seqs        : [20^{12},100^{300}]
  6-seqs        : [100^{1512}]
  7-seqs        : [100^{6048}]
  8-seqs        : [100^{18144}]

Name = t10n10
  Generator     : (1,6,4,10)(2,9,3,7)(5,8)
                  (1,2)(3,5)(7,10)(8,9)
  Order         : 100
  Parity        : Odd
  Imprimitiv-   : Yes
  Shape         : [0001000000000000000000110000010000000000000]
  2-sets        : [10^{2},25^{1}]
  3-sets        : [10^{2},50^{2}]
  4-sets        : [10^{1},25^{2},50^{3}]
  5-sets        : [2^{1},50^{5}]
  2-seqs        : [20^{2},50^{1}]
  3-seqs        : [20^{6},100^{6}]
  4-seqs        : [20^{12},100^{48}]
  5-seqs        : [20^{12},100^{300}]
  6-seqs        : [100^{1512}]
  7-seqs        : [100^{6048}]
  8-seqs        : [100^{18144}]

Name = t10n11
  Generator     : (1,2,4,7,9)(3,6,8,10,5)
                  (1,3)(2,5)(4,6)(7,8)(9,10)
  Order         : 120
  Parity        : Odd
  Imprimitive   : Yes
  Shape         : [000110000100000000000010000001001000000000]
  2-sets        : [5^{1},20^{2}]
  3-sets        : [20^{1},40^{1},60^{1}]
  4-sets        : [10^{2},30^{1},40^{1},60^{2}]
  5-sets        : [2^{1},10^{1},20^{1},40^{1},60^{1},120^{1}]
  2-seqs        : [10^{1},40^{2}]
  3-seqs        : [40^{6},120^{4}]
  4-seqs        : [40^{6},120^{40}]
  5-seqs        : [120^{252}]
  6-seqs        : [120^{1260}]
  7-seqs        : [120^{5040}]
  8-seqs        : [120^{15120}]

Name = t10n12
  Generator     : (1,3,5,9,7)(2,4,6,10,8)
                  (1,6)(2,5)(3,10)(4,9)(7,8)
  Order         : 120
  Parity        : Odd
  Imprimitive : Yes
  Shape       : [00011000010000000000000000001001000000001]
  2-sets      : [5^{1},20^{2}]
  3-sets      : [20^{1},40^{1},60^{1}]
  4-sets      : [10^{2},30^{1},40^{1},60^{2}]
  5-sets      : [2^{1},10^{1},20^{1},40^{1},60^{1},120^{1}]
  2-seqs      : [10^{1},40^{2}]
  3-seqs      : [40^{6},120^{4}]
  4-seqs      : [40^{6},120^{40}]
  5-seqs      : [120^{252}]
  6-seqs      : [120^{1260}]
  7-seqs      : [120^{5040}]
  8-seqs      : [120^{15120}]

Name = t10n18
  Generator   : (1,5,2,3)(6,10,7,8)
                (1,6,2,8)(3,10,5,9)(4,7)
  Order       : 200
  Parity      : Odd
  Imprimitive : Yes
  Shape       : [00011000000000000000111000001000000000001]
  2-sets      : [20^{1},25^{1}]
  3-sets      : [20^{1},100^{1}]
  4-sets      : [10^{1},50^{2},100^{1}]
  5-sets      : [2^{1},50^{1},100^{2}]
  2-seqs      : [40^{1},50^{1}]
  3-seqs      : [40^{3},200^{3}]
  4-seqs      : [40^{6},200^{24}]
  5-seqs      : [40^{6},200^{150}]
  6-seqs      : [200^{756}]
  7-seqs      : [200^{3024}]
  8-seqs      : [200^{9072}]

Name = t10n20
  Generator   : (1,2,4,3)(6,9,8,10)
                (1,6,2,8,3,10,4,7,5,9)
  Order       : 200
  Parity      : Odd
  Imprimitive : Yes
  Shape       : [00011000000000000000101000001000000000001]
  2-sets      : [20^{1},25^{1}]
  3-sets      : [20^{1},100^{1}]
  4-sets      : [10^{1},50^{2},100^{1}]
  5-sets      : [2^{1},50^{1},100^{2}]
  2-seqs      : [40^{1},50^{1}]
  3-seqs      : [40^{3},200^{3}]
  4-seqs      : [40^{6},200^{24}]
  5-seqs      : [40^{6},200^{150}]
  6-seqs      : [200^{756}]
  7-seqs      : [200^{3024}]
  8-seqs      : [200^{9072}]

Name = t10n21
  Generator   : (1,6)(2,8,5,9)(3,10,4,7)
                (1,4,5,2)(6,9,8,10)
  Order       : 200
  Parity       : Odd
  Imprimitive  : Yes
  Shape        : [00010000000000000000111000001000000000000]
  2-sets       : [20^{1},25^{1}]
  3-sets       : [20^{1},100^{1}]
  4-sets       : [10^{1},50^{2},100^{1}]
  5-sets       : [2^{1},50^{1},100^{2}]
  2-seqs       : [40^{1},50^{1}]
  3-seqs       : [40^{3},200^{3}]
  4-seqs       : [40^{6},200^{24}]
  5-seqs       : [40^{6},200^{150}]
  6-seqs       : [200^{756}]
  7-seqs       : [200^{3024}]
  8-seqs       : [200^{9072}]

Name = t10n37
  Generator    : (1,4,10,2,3,9)(7,8)
                 (1,3,8,10,5,2,4,7,9,6)
  Order        : 1920
  Parity       : Odd
  Imprimitive  : Yes
  Shape        : [11111000011100011000110000000111100000001]
  2-sets       : [5^{1},40^{1}]
  3-sets       : [40^{1},80^{1}]
  4-sets       : [10^{1},80^{1},120^{1}]
  5-sets       : [32^{1},60^{1},160^{1}]
  2-seqs       : [10^{1},80^{1}]
  3-seqs       : [80^{3},480^{1}]
  4-seqs       : [80^{3},480^{6},960^{2}]
  5-seqs       : [480^{15},960^{20},1920^{2}]
  6-seqs       : [480^{15},960^{90},1920^{30}]
  7-seqs       : [960^{210},1920^{210}]
  8-seqs       : [960^{210},1920^{840}]

Name = t10n39
  Generator    : (1,4,8,5,2,3,7,6)
                 (1,4,7,9,6,2,3,8,10,5)
  Order        : 3840
  Parity       : Odd
  Imprimitive  : Yes
  Shape        : [11111000011101111001110000001111010001101]
  2-sets       : [5^{1},40^{1}]
  3-sets       : [40^{1},80^{1}]
  4-sets       : [10^{1},80^{1},120^{1}]
  5-sets       : [32^{1},60^{1},160^{1}]
  2-seqs       : [10^{1},80^{1}]
  3-seqs       : [80^{3},480^{1}]
  4-seqs       : [80^{3},480^{6},1920^{1}]
  5-seqs       : [480^{15},1920^{10},3840^{1}]
  6-seqs       : [480^{15},1920^{45},3840^{15}]
  7-seqs       : [1920^{105},3840^{105}]
  8-seqs       : [1920^{105},3840^{420}]

Name = t10n42
  Generator    : (1,8)(2,6,3,7)(4,10,5,9)
                 (1,2,5,3)(6,7,10)(8,9)
  Order        : 14400
  Parity       : Odd
  Imprimitive  : Yes
  Shape        : [0101110101010010001011101100100100000001]
  2-sets       : [20^{1},25^{1}]
  3-sets       : [20^{1},100^{1}]
  4-sets       : [10^{1},100^{2}]
  5-sets       : [2^{1},50^{1},200^{1}]
  2-seqs       : [40^{1},50^{1}]
  3-seqs       : [120^{1},200^{3}]
  4-seqs       : [240^{1},600^{4},800^{3}]
  5-seqs       : [240^{1},1200^{5},2400^{10}]
  6-seqs       : [1200^{6},4800^{15},7200^{10}]
  7-seqs       : [4800^{21},14400^{35}]
  8-seqs       : [14400^{126}]

Name = t10n43
  Generator    : (1,2,3,5,4)(6,7)(8,10,9)
                 (1,6)(2,10)(3,9,4,7)(5,8)
  Order        : 28800
  Parity       : Odd
  Imprimitive  : Yes
  Shape        : [11111111111101111110111111111001010000101]
  2-sets       : [20^{1},25^{1}]
  3-sets       : [20^{1},100^{1}]
  4-sets       : [10^{1},100^{2}]
  5-sets       : [2^{1},50^{1},200^{1}]
  2-seqs       : [40^{1},50^{1}]
  3-seqs       : [120^{1},200^{3}]
  4-seqs       : [240^{1},600^{4},800^{3}]
  5-seqs       : [240^{1},1200^{5},2400^{10}]
  6-seqs       : [1200^{6},4800^{15},7200^{10}]
  7-seqs       : [4800^{21},14400^{35}]
  8-seqs       : [14400^{56},28800^{35}]

DEGREE 12

Name = t12n1
  Generator    : (1,2,3,4,5,6,7,8,9,10,11,12)
  Order        : 12
  Parity       : Odd
  Imprimitive  : Yes
  Shape        : [000001000000000000010000000000000000100000000000000000
                  0000001000000000000000000001]
  2-sets       : [6^{1},12^{5}]
  3-sets       : [4^{1},12^{18}]
  4-sets       : [3^{1},6^{2},12^{40}]
  5-sets       : [12^{66}]
  6-sets       : [2^{1},4^{1},6^{3},12^{75}]
  2-seqs       : [12^{11}]
  3-seqs       : [12^{110}]
  4-seqs       : [12^{990}]
  5-seqs       : [12^{7920}]
  6-seqs       : [12^{55440}]
  7-seqs       : [12^{332640}]
  8-seqs       : [12^{1663200}]
  9-seqs       : [12^{6652800}]

Name = t12n5
  Generator    : (1,2,3,4,5,6)(7,12,11,10,9,8)
                 (1,7,4,10)(2,8,5,11)(3,9,6,12)
  Order        : 12
  Parity       : Odd
  Imprimitive  : Yes
  Shape        : [00000100000000000100000000000000000010000000000000000
                  0000001000000000000000000000]
  2-sets       : [6^{1},12^{5}]
  3-sets       : [4^{1},12^{18}]
  4-sets       : [3^{3},6^{1},12^{40}]
  5-sets       : [12^{66}]
  6-sets       : [2^{1},4^{1},6^{3},12^{75}]
  2-seqs       : [12^{11}]
  3-seqs       : [12^{110}]
  4-seqs       : [12^{990}]
  5-seqs       : [12^{7920}]
  6-seqs       : [12^{55440}]
  7-seqs       : [12^{332640}]
  8-seqs       : [12^{1663200}]
  9-seqs       : [12^{6652800}]

Name = t12n34
  Generator    : (1,2,4,7,11,10)(3,6,9,5,8,12)
                 (1,3)(2,5)(4,8)(6,10)(7,12)(9,11)
  Order        : 72
  Parity       : Even
  Imprimitive  : Yes
  Shape        : [00010100000010000010000000000000001000000000000000001
                  0000001000000000000000000000]
  2-sets       : [6^{1},12^{2},18^{2}]
  3-sets       : [4^{1},12^{1},24^{1},36^{3},72^{1}]
  4-sets       : [6^{1},9^{1},12^{4},18^{2},36^{5},72^{3}]
  5-sets       : [12^{4},24^{1},36^{6},72^{7}]
  6-sets       : [2^{3},12^{2},18^{3},24^{2},36^{10},72^{6}]
  2-seqs       : [12^{1},24^{2},36^{2}]
  3-seqs       : [24^{10},36^{6},72^{12}]
  4-seqs       : [24^{30},36^{6},72^{152}]
  5-seqs       : [24^{60},72^{1300}]
  6-seqs       : [24^{60},72^{9220}]
  7-seqs       : [72^{55440}]
  8-seqs       : [72^{277200}]
  9-seqs       : [72^{1108800}]

Name = t12n40
  Generator    : (1,11)(2,12)(3,9,5,7)(4,10,6,8)
                 (1,2)(3,4)(5,6)(7,12,9,8,11,10)
  Order        : 72
  Parity       : Even
  Imprimitive  : Yes
  Shape        : [00010100000010000010000000000000001000000000000000001
                  0000001000000000000000000000]
  2-sets       : [6^{1},12^{2},18^{2}]
  3-sets       : [4^{1},12^{1},24^{1},36^{3},72^{1}]
  4-sets       : [6^{1},9^{1},12^{4},18^{2},36^{5},72^{3}]
```

```
    5-sets       :  [12^{4},24^{1},36^{6},72^{7}]
    6-sets       :  [2^{3},12^{2},18^{3},24^{2},36^{10},72^{6}]
    2-seqs       :  [12^{1},24^{2},36^{2}]
    3-seqs       :  [24^{10},36^{6},72^{12}]
    4-seqs       :  [24^{30},36^{6},72^{152}]
    5-seqs       :  [24^{60},72^{1300}]
    6-seqs       :  [24^{60},72^{9220}]
    7-seqs       :  [72^{55440}]
    8-seqs       :  [72^{277200}]
    9-seqs       :  [72^{1108800}]

Name = t12n145
    Generator    :  (1,6)(2,5)(3,8,4,7)(9,11)(10,12)
                    (1,6,11,4,8,10)(2,5,12,3,7,9)
    Order        :  384
    Parity       :  Odd
    Imprimitive  :  Yes
    Shape        :  [0101110000000000001001010000001101000000000000000000
                     0000001000000000000000000000]
    2-sets       :  [6^{1},12^{1},24^{2}]
    3-sets       :  [12^{1},16^{1},24^{2},48^{1},96^{1}]
    4-sets       :  [3^{1},6^{2},24^{2},48^{5},96^{2}]
    5-sets       :  [12^{2},24^{4},48^{2},96^{4},192^{1}]
    6-sets       :  [2^{1},6^{1},12^{1},24^{7},48^{4},64^{1},96^{1},
                    192^{2}]
    2-seqs       :  [12^{1},24^{1},48^{2}]
    3-seqs       :  [24^{3},48^{6},96^{10}]
    4-seqs       :  [24^{3},48^{6},96^{60},192^{30}]
    5-seqs       :  [96^{150},192^{300},384^{60}]
    6-seqs       :  [96^{150},192^{1350},384^{1020}]
    7-seqs       :  [192^{3150},384^{8820}]
    8-seqs       :  [192^{3150},384^{50400}]
    9-seqs       :  [384^{207900}]

Name = t12n152
    Generator    :  (1,11,5,4,10,7,2,12,6,3,9,8)
                    (1,10,2,9)(3,11,4,12)(5,7)(6,8)
    Order        :  384
    Parity       .  Odd
    Imprimitive  :  Yes
    Shape        :  [0101110000000000001001000000011101000000000000000000
                     0000001000000000000000000001]
    2-sets       :  [6^{1},12^{1},48^{1}]
    3-sets       :  [12^{1},48^{1},64^{1},96^{1}]
    4-sets       :  [3^{1},12^{1},24^{2},48^{1},96^{2},192^{1}]
    5-sets       :  [24^{1},48^{2},96^{3},192^{2}]
    6-sets       :  [8^{1},12^{1},16^{1},24^{1},48^{6},192^{1},384^{1}]
    2-seqs       :  [12^{1},24^{1},96^{1}]
    3-seqs       :  [24^{3},96^{3},192^{3},384^{1}]
    4-seqs       :  [24^{3},96^{3},192^{24},384^{18}]
    5-seqs       :  [192^{105},384^{195}]
    6-seqs       :  [192^{315},384^{1575}]
    7-seqs       :  [192^{630},384^{10080}]
    8-seqs       :  [192^{630},384^{51660}]
    9-seqs       :  [384^{207900}]
```

```
Name = t12n153
  Generator    : (1,8,10,3,6,11,2,7,9,4,5,12)
                 (1,12,2,11)(3,9,4,10)
  Order        : 384
  Parity       : Odd
  Imprimitive  : Yes
  Shape        : [010101000000000000100101000001110100000000000000000
                  00000010000000000000000001]
  2-sets       : [6^{1},12^{1},48^{1}]
  3-sets       : [12^{1},48^{1},64^{1},96^{1}]
  4-sets       : [3^{1},12^{1},24^{2},48^{1},96^{2},192^{1}]
  5-sets       : [24^{1},48^{2},96^{3},192^{2}]
  6-sets       : [8^{1},12^{1},16^{1},24^{1},48^{6},192^{1},384^{1}]
  2-seqs       : [12^{1},24^{1},96^{1}]
  3-seqs       : [24^{3},96^{3},192^{3},384^{1}]
  4-seqs       : [24^{3},96^{3},192^{24},384^{18}]
  5-seqs       : [192^{105},384^{195}]
  6-seqs       : [192^{315},384^{1575}]
  7-seqs       : [192^{630},384^{10080}]
  8-seqs       : [192^{630},384^{51660}]
  9-seqs       : [384^{207900}]

Name = t12n154
  Generator    : (1,9)(2,10)(3,11,4,12)(5,8,6,7)
                 *(1,3)(2,4)(5,7)(6,8)(9,11,10,12)
                 (1,5,12,4,7,9,2,6,11,3,8,10)
  Order        : 384
  Parity       : Odd
  Imprimitive  : Yes
  Shape        : [010111000000000000100101000000110100000000000000000
                  00000010000000000000000001]
  2-sets       : [6^{1},12^{1},24^{2}]
  3-sets       : [12^{1},16^{1},24^{2},48^{1},96^{1}]
  4-sets       : [3^{1},6^{2},24^{2},48^{5},96^{2}]
  5-sets       : [12^{2},24^{4},48^{2},96^{4},192^{1}]
  6-sets       : [2^{1},6^{1},12^{1},24^{7},48^{4},64^{1},96^{1},
                 192^{2}]
  2-seqs       : [12^{1},24^{1},48^{2}]
  3-seqs       : [24^{3},48^{6},96^{10}]
  4-seqs       : [24^{3},48^{6},96^{60},192^{30}]
  5-seqs       : [96^{150},192^{300},384^{60}]
  6-seqs       : [96^{150},192^{1350},384^{1020}]
  7-seqs       : [192^{3150},384^{8820}]
  8-seqs       : [192^{3150},384^{50400}]
  9-seqs       : [384^{207900}]

Name = t12n155
  Generator    : (1,9,7,3,11,5,2,10,8,4,12,6)
                 (1,9)(2,10)(3,12)(4,11)(5,7,6,8)
  Order        : 384
  Parity       : Odd
  Imprimitive  : Yes
  Shape        : [010101000000000000100011000001010100000000000000000
                  00000010000000000000000001]
  2-sets       : [6^{1},12^{1},24^{2}]
  3-sets       : [12^{1},16^{1},24^{2},48^{1},96^{1}]
  4-sets       : [3^{1},6^{2},24^{2},48^{5},96^{2}]
  5-sets      : [12^{2},24^{4},48^{2},96^{4},192^{1}]
  6-sets      : [2^{1},6^{1},12^{1},24^{7},48^{4},64^{1},96^{1},
                192^{2}]
  2-seqs      : [12^{1},24^{1},48^{2}]
  3-seqs      : [24^{3},48^{6},96^{10}]
  4-seqs      : [24^{3},48^{6},96^{60},192^{30}]
  5-seqs      : [96^{150},192^{300},384^{60}]
  6-seqs      : [96^{150},192^{1350},384^{1020}]
  7-seqs      : [192^{3150},384^{8820}]
  8-seqs      : [192^{3150},384^{50400}]
  9-seqs      : [384^{207900}]

Name = t12n168
  Generator   : (1,2)(4,6)(7,9)(10,11)
                (1,10,2,11,3,12)(4,8)(5,9)(6,7)
                (1,7,3,9,2,8)(4,10)(5,11)(6,12)
  Order       : 648
  Parity      : Even
  Imprimitive : Yes
  Shape       : [00010110000100010100000000000000000000000000000001
                 000000100000000000000000000]
  2-sets      : [12^{1},18^{3}]
  3-sets      : [4^{1},36^{3},108^{1}]
  4-sets      : [12^{3},18^{3},81^{1},108^{3}]
  5-sets      : [12^{3},36^{3},108^{3},324^{1}]
  6-sets      : [2^{3},36^{6},108^{2},162^{3}]
  2-seqs      : [24^{1},36^{3}]
  3-seqs      : [24^{1},72^{9},108^{6}]
  4-seqs      : [72^{30},216^{36},324^{6}]
  5-seqs      : [72^{60},216^{240},648^{60}]
  6-seqs      : [72^{60},216^{1080},648^{660}]
  7-seqs      : [216^{3360},648^{5040}]
  8-seqs      : [216^{6720},648^{28560}]
  9-seqs      : [216^{6720},648^{120960}]

Name = t12n171
  Generator   : (1,11,3,12)(2,10)(4,8,5,7)(6,9)
                (1,8)(2,9,3,7)(4,12,5,10)(6,11)
  Order       : 648
  Parity      : Even
  Imprimitive : Yes
  Shape       : [00010110000100010100000000000010000000000000000001
                 000000100000000000000000000]
  2-sets      : [12^{1},18^{3}]
  3-sets      : [4^{1},36^{3},108^{1}]
  4-sets      : [12^{3},18^{3},81^{1},108^{3}]
  5-sets      : [12^{3},36^{3},108^{3},324^{1}]
  6-sets      : [2^{3},36^{6},108^{2},162^{3}]
  2-seqs      : [24^{1},36^{3}]
  3-seqs      : [24^{1},72^{9},108^{6}]
  4-seqs      : [72^{30},216^{36},324^{6}]
  5-seqs      : [72^{60},216^{240},648^{60}]
  6-seqs      : [72^{60},216^{1080},648^{660}]
  7-seqs      : [216^{3360},648^{5040}]
  8-seqs      : [216^{6720},648^{28560}]
  9-seqs      : [216^{6720},648^{120960}]
```

```
Name = t12n172
  Generator    : (1,10,3,12)(2,11)(4,7,6,9)(5,8)
                 (1,4,2,5,3,6)(7,10,8,12,9,11)
  Order        : 648
  Parity       : Even
  Imprimitive  : Yes
  Shape        : [000101100001000101000000000000000010000000000000000001
                  000000100000000000000000000000]
  2-sets       : [12^{1},18^{3}]
  3-sets       : [4^{1},36^{3},108^{1}]
  4-sets       : [12^{3},18^{3},81^{1},108^{3}]
  5-sets       : [12^{3},36^{3},108^{3},324^{1}]
  6-sets       : [2^{3},36^{6},108^{2},162^{3}]
  2-seqs       : [24^{1},36^{3}]
  3-seqs       : [24^{1},72^{9},108^{6}]
  4-seqs       : [72^{30},216^{36},324^{6}]
  5-seqs       : [72^{60},216^{240},648^{60}]
  6-seqs       : [72^{60},216^{1080},648^{660}]
  7-seqs       : [216^{3360},648^{5040}]
  8-seqs       : [216^{6720},648^{28560}]
  9-seqs       : [216^{6720},648^{120960}]

Name = t12n174
  Generator    : (1,5)(2,4,3,6)(7,12)(8,11,9,10)
                 *(1,6,2,5)(3,4)(7,10)(8,12,9,11)
                 (1,9,3,8)(2,7)(4,10,6,11)(5,12)
  Order        : 648
  Parity       : Even
  Imprimitive  : Yes
  Shape        : [000100100001000101000000000000000010000000000000000000
                  000000000000000000000000000000]
  2-sets       : [12^{1},18^{3}]
  3-sets       : [4^{1},36^{3},108^{1}]
  4-sets       : [12^{3},18^{3},81^{1},108^{3}]
  5-sets       : [12^{3},36^{3},108^{3},324^{1}]
  6-sets       : [2^{3},36^{6},108^{2},162^{3}]
  2-seqs       : [24^{1},36^{3}]
  3-seqs       : [24^{1},72^{9},108^{6}]
  4-seqs       : [72^{30},216^{36},324^{6}]
  5-seqs       : [72^{60},216^{240},648^{60}]
  6-seqs       : [72^{60},216^{1080},648^{660}]
  7-seqs       : [216^{3360},648^{5040}]
  8-seqs       : [216^{6720},648^{28560}]
  9-seqs       : [216^{6720},648^{120960}]

Name = t12n180
  Generator    : (1,12,9,6,3,2,11,10,5,4)(7,8)
                 (1,8,3,12)(2,7,4,11)(5,10)(6,9)
  Order        : 720
  Parity       : Even
  Imprimitive  : Yes
  Shape        : [000101000000100000010000000000000010000000000000100001
                  000000100000000000000000000100]
  2-sets       : [6^{1},30^{2}]
  3-sets       : [40^{1},60^{1},120^{1}]
  4-sets       : [15^{1},30^{1},90^{1},120^{3}]
  5-sets       : [12^{1},60^{1},120^{3},360^{1}]
  6-sets       : [2^{1},12^{1},20^{2},30^{1},60^{1},180^{3},240^{1}]
  2-seqs       : [12^{1},60^{2}]
  3-seqs       : [60^{6},240^{4}]
  4-seqs       : [60^{6},240^{24},720^{8}]
  5-seqs       : [240^{60},720^{112}]
  6-seqs       : [240^{60},720^{904}]
  7-seqs       : [720^{5544}]
  8-seqs       : [720^{27720}]
  9-seqs       : [720^{110880}]

Name = t12n183
  Generator    : (1,2)(3,5)(4,6,8,10,11,12)(7,9)
                 (1,3)(2,4)(5,7)(6,8)(9,10)(11,12)
  Order        : 720
  Parity       : Even
  Imprimitive  : Yes
  Shape        : [0001010000010000010000000000000100000000000000100001
                  0000001000000000C000000000]
  2-sets       : [6^{1},30^{2}]
  3-sets       : [40^{1},60^{1},120^{1}]
  4-sets       : [15^{1},30^{1},90^{1},120^{3}]
  5-sets       : [12^{1},60^{1},120^{3},360^{1}]
  6-sets       : [2^{1},12^{1},20^{2},30^{1},60^{1},180^{3},240^{1}]
  2-seqs       : [12^{1},60^{2}]
  3-seqs       : [60^{6},240^{4}]
  4-seqs       : [60^{6},240^{24},720^{8}]
  5-seqs       : [240^{60},720^{112}]
  6-seqs       : [240^{60},720^{904}]
  7-seqs       : [720^{5544}]
  8-seqs       : [720^{27720}]
  9-seqs       : [720^{110880}]

Name = t12n196
  Generator    : (1,7,6,9,4,12)(2,8,5,10,3,11)
                 (1,12,4,8,6,9,2,11,3,7,5,10)
  Order        : 1152
  Parity       : Odd
  Imprimitive  : Yes
  Shape        : [0101110000010100010010100000011010000000000000000101
                  0000001000000000000000000001]
  2-sets       : [6^{1},24^{1},36^{1}]
  3-sets       : [16^{1},24^{1},36^{1},144^{1}]
  4-sets       : [6^{1},9^{1},24^{1},72^{1},96^{1},144^{2}]
  5-sets       : [12^{1},36^{1},48^{1},72^{1},144^{1},192^{1},288^{1}]
  6-sets       : [2^{1},18^{1},64^{1},72^{5},192^{1},288^{1}]
  2-seqs       : [12^{1},48^{1},72^{1}]
  3-seqs       : [48^{3},72^{3},96^{1},288^{3}]
  4-seqs       : [48^{3},72^{3},96^{6},288^{18},576^{10}]
  5-seqs       : [96^{15},288^{45},576^{100},1152^{20}]
  6-seqs       : [96^{15},288^{45},576^{450},1152^{340}]
  7-seqs       : [576^{1050},1152^{2940}]
  8-seqs       : [576^{1050},1152^{16800}]
  9-seqs       : [1152^{69300}]

Name = t12n197
```

```
  Generator    : (1,3)(2,4)(5,6)(7,9,8,10)
                 *(1,2)(3,6)(4,5)(7,8)(9,11)(10,12)
                 (1,7,6,12,3,9,2,8,5,11,4,10)
  Order        : 1152
  Parity       : Odd
  Imprimitive  : Yes
  Shape        : [01010100000101000100011000001010100000000000000000101
                  0000001000000000000000000001]
  2-sets       : [6^{1},24^{1},36^{1}]
  3-sets       : [16^{1},24^{1},36^{1},144^{1}]
  4-sets       : [6^{1},9^{1},24^{1},72^{1},96^{1},144^{2}]
  5-sets       : [12^{1},36^{1},48^{1},72^{1},144^{1},192^{1},288^{1}]
  6-sets       : [2^{1},18^{1},64^{1},72^{5},192^{1},288^{1}]
  2-seqs       : [12^{1},48^{1},72^{1}]
  3-seqs       : [48^{3},72^{3},96^{1},288^{3}]
  4-seqs       : [48^{3},72^{3},96^{6},288^{18},576^{10}]
  5-seqs       : [96^{15},288^{45},576^{100},1152^{20}]
  6-seqs       : [96^{15},288^{45},576^{450},1152^{340}]
  7-seqs       : [576^{1050},1152^{2940}]
  8-seqs       : [576^{1050},1152^{16800}]
  9-seqs       : [1152^{69300}]

Name = t12n209
  Generator    : (1,10,4,9)(2,11,5,7)(3,12,6,8)
                 (1,10,2,12)(3,11)(4,9)(5,8,6,7)
  Order        : 1296
  Parity       : Odd
  Imprimitive  : Yes
  Shape        : [00111110010100110100000000000000101000000000000001011
                  101000100000000000000000001]
  2-sets       : [12^{1},18^{1},36^{1}]
  3-sets       : [4^{1},36^{1},72^{1},108^{1}]
  4-sets       : [12^{1},18^{1},24^{1},36^{1},81^{1},108^{1},216^{1}]
  5-sets       : [12^{1},24^{1},36^{1},72^{1},108^{1},216^{1},324^{1}]
  6-sets       : [2^{1},4^{1},72^{3},108^{2},162^{1},324^{1}]
  2-seqs       : [24^{1},36^{1},72^{1}]
  3-seqs       : [24^{1},72^{3},144^{3},216^{3}]
  4-seqs       : [72^{10},144^{10},432^{18},648^{3}]
  5-seqs       : [72^{20},144^{20},432^{120},1296^{30}]
  6-seqs       : [72^{20},144^{20},432^{540},1296^{330}]
  7-seqs       : [432^{1680},1296^{2520}]
  8-seqs       : [432^{3360},1296^{14280}]
  9-seqs       : [432^{3360},1296^{60480}]

Name = t12n210
  Generator    : (1,7,2,9)(3,8)(4,11,6,12)(5,10)
                 (1,3)(5,6)(7,9,8)(10,12,11)
                 (1,4)(2,5)(3,6)(7,10,9,12,8,11)
  Order        : 1296
  Parity       : Even
  Imprimitive  : Yes
  Shape        : [0101011010010101010000000000000010000000000000000001
                  0000001000000000000000000000]
  2-sets       : [12^{1},18^{3}]
  3-sets       : [4^{1},36^{3},108^{1}]
  4-sets       : [12^{3},18^{3},81^{1},108^{3}]
  5-sets       : [12^{3},36^{3},108^{3},324^{1}]
  6-sets       : [2^{3},36^{6},108^{2},162^{3}]
  2-seqs       : [24^{1},36^{3}]
  3-seqs       : [24^{1},72^{9},108^{6}]
  4-seqs       : [72^{18},144^{6},216^{36},324^{6}]
  5-seqs       : [72^{20},144^{20},216^{120},432^{60},648^{60}]
  6-seqs       : [72^{20},144^{20},216^{240},432^{420},648^{300},
                 1296^{180}]
  7-seqs       : [216^{280},432^{1540},648^{840},1296^{2100}]
  8-seqs       : [432^{3360},648^{1120},1296^{13720}]
  9-seqs       : [432^{3360},1296^{60480}]

Name = t12n212
  Generator    : (1,12,3,10)(2,11)(4,8,5,7)(6,9)
                 (1,10,5,8)(2,11,6,7,3,12,4,9)
  Order        : 1296
  Parity       : Even
  Imprimitive  : Yes
  Shape        : [000100100C110001010000000000000001000000000000000C0100
                  0100000000000000000010000000]
  2-sets       : [12^{1},18^{1},36^{1}]
  3-sets       : [4^{1},36^{1},72^{1},108^{1}]
  4-sets       : [12^{1},18^{1},24^{1},36^{1},81^{1},108^{1},216^{1}]
  5-sets       : [12^{1},24^{1},36^{1},72^{1},108^{1},216^{1},324^{1}]
  6-sets       : [2^{1},4^{1},72^{3},108^{2},162^{1},324^{1}]
  2-seqs       : [24^{1},36^{1},72^{1}]
  3-seqs       : [24^{1},72^{3},144^{3},216^{3}]
  4-seqs       : [72^{4},144^{13},432^{18},648^{3}]
  5-seqs       : [144^{30},432^{120},1296^{30}]
  6-seqs       : [144^{30},432^{540},1296^{330}]
  7-seqs       : [432^{1680},1296^{2520}]
  8-seqs       : [432^{3360},1296^{14280}]
  9-seqs       : [432^{3360},1296^{60480}]

Name = t12n214
  Generator    : (1,5)(2,6,3,4)(7,11,8,10)(9,12)
                 (1,7,2,9,3,8)(4,11,5,12,6,10)
                 (1,3)(4,6)(7,9,8)(10,12,11)
  Order        : 1296
  Parity       : Even
  Imprimitive  : Yes
  Shape        : [0101011010010101010000000000000100000000000000000001
                  0000001000000000000000000000]
  2-sets       : [12^{1},18^{3}]
  3-sets       : [4^{1},36^{3},108^{1}]
  4-sets       : [12^{3},18^{3},81^{1},108^{3}]
  5-sets       : [12^{3},36^{3},108^{3},324^{1}]
  6-sets       : [2^{3},36^{6},108^{2},162^{3}]
  2-seqs       : [24^{1},36^{3}]
  3-seqs       : [24^{1},72^{9},108^{6}]
  4-seqs       : [72^{18},144^{6},216^{36},324^{6}]
  5-seqs       : [72^{20},144^{20},216^{120},432^{60},648^{60}]
  6-seqs       : [72^{20},144^{20},216^{240},432^{420},648^{300},
                 1296^{180}]
  7-seqs       : [216^{280},432^{1540},648^{840},1296^{2100}]
  8-seqs       : [432^{3360},648^{1120},1296^{13720}]
  9-seqs       : [432^{3360},1296^{60480}]
```

```
Name = t12n216
  Generator    : (1,5,9,12,2,6,7,10)(3,4,8,11)
                 (1,9,3,7,2,8)(4,5)(10,11,12)
  Order        : 1296
  Parity       : Even
  Imprimitive  : Yes
  Shape        : [000101100011000101000000000000001000000000000000101
                  010000100000000000010000000]
  2-sets       : [12^{1},18^{1},36^{1}]
  3-sets       : [4^{1},36^{1},72^{1},108^{1}]
  4-sets       : [12^{1},18^{1},24^{1},36^{1},81^{1},108^{1},216^{1}]
  5-sets       : [12^{1},24^{1},36^{1},72^{1},108^{1},216^{1},324^{1}]
  6-sets       : [2^{1},4^{1},72^{3},108^{2},162^{1},324^{1}]
  2-seqs       : [24^{1},36^{1},72^{1}]
  3-seqs       : [24^{1},72^{3},144^{3},216^{3}]
  4-seqs       : [72^{4},144^{13},432^{18},648^{3}]
  5-seqs       : [144^{30},432^{120},1296^{30}]
  6-seqs       : [144^{30},432^{540},1296^{330}]
  7-seqs       : [432^{1680},1296^{2520}]
  8-seqs       : [432^{3360},1296^{14280}]
  9-seqs       : [432^{3360},1296^{60480}]

Name = t12n217
  Generator    : (1,7,6,11,3,9,5,10,2,8,4,12)
                 (1,4,3,5,2,6)(7,12)(8,11)(9,10)
                 (1,6)(2,4)(3,5)(7,9,8)
  Order        : 1296
  Parity       : Odd
  Imprimitive  : Yes
  Shape        : [001111100101001101000000000000001000000000000001011
                  101000100000000000000000001]
  2-sets       : [12^{1},18^{1},36^{1}]
  3-sets       : [4^{1},36^{1},72^{1},108^{1}]
  4-sets       : [12^{1},18^{1},24^{1},36^{1},81^{1},108^{1},216^{1}]
  5-sets       : [12^{1},24^{1},36^{1},72^{1},108^{1},216^{1},324^{1}]
  6-sets       : [2^{1},4^{1},72^{3},108^{2},162^{1},324^{1}]
  2-seqs       : [24^{1},36^{1},72^{1}]
  3-seqs       : [24^{1},72^{3},144^{3},216^{3}]
  4-seqs       : [72^{10},144^{10},432^{18},648^{3}]
  5-seqs       : [72^{20},144^{20},432^{120},1296^{30}]
  6-seqs       : [72^{20},144^{20},432^{540},1296^{330}]
  7-seqs       : [432^{1680},1296^{2520}]
  8-seqs       : [432^{3360},1296^{14280}]
  9-seqs       : [432^{3360},1296^{60480}]

Name = t12n221
  Generator    : (1,10)(2,9)(3,11,4,12)(7,8)
                 (1,7,4,5)(2,8,3,6)(9,10)
  Order        : 1536
  Parity       : Odd
  Imprimitive  : Yes
  Shape        : [011111000000000000110111000001110100000000000000000
                  000000100000001010100000001]
  2-sets       : [6^{1},12^{1},48^{1}]
  3-sets       : [12^{1},48^{1},64^{1},96^{1}]
  4-sets       : [3^{1},12^{1},48^{2},96^{2},192^{1}]
  5-sets       : [24^{1},48^{2},96^{1},192^{3}]
  6-sets       : [8^{1},12^{1},24^{1},48^{3},64^{1},96^{1},192^{1},
                 384^{1}]
  2-seqs       : [12^{1},24^{1},96^{1}]
  3-seqs       : [24^{3},96^{3},192^{3},384^{1}]
  4-seqs       : [24^{3},96^{3},192^{18},384^{9},768^{6}]
  5-seqs       : [192^{45},384^{45},768^{60},1536^{15}]
  6-seqs       : [192^{45},384^{150},768^{270},1536^{255}]
  7-seqs       : [384^{315},768^{630},1536^{2205}]
  8-seqs       : [384^{315},768^{630},1536^{12600}]
  9-seqs       : [1536^{51975}]

Name = t12n223
  Generator    : (1,12)(2,11)(3,10,4,9)(5,6)(7,8)
                 (1,11,7,3,9,5)(2,12,8,4,10,6)
                 *(1,3,2,4)(5,9)(6,10)(7,11)(8,12)
  Order        : 1536
  Parity       : Odd
  Imprimitive  : Yes
  Shape        : [0101110000000000001011110000011101000000000000000000
                  000000100000000101000000000]
  2-sets       : [6^{1},12^{1},48^{1}]
  3-sets       : [12^{1},48^{1},64^{1},96^{1}]
  4-sets       : [3^{1},12^{1},48^{2},96^{2},192^{1}]
  5-sets       : [24^{1},48^{2},96^{1},192^{3}]
  6-sets       : [8^{1},12^{1},24^{1},48^{3},64^{1},96^{1},192^{1},
                 384^{1}]
  2-seqs       : [12^{1},24^{1},96^{1}]
  3-seqs       : [24^{3},96^{3},192^{3},384^{1}]
  4-seqs       : [24^{3},96^{3},192^{18},384^{9},768^{6}]
  5-seqs       : [192^{45},384^{45},768^{60},1536^{15}]
  6-seqs       : [192^{45},384^{150},768^{270},1536^{255}]
  7-seqs       : [384^{315},768^{630},1536^{2205}]
  8-seqs       : [384^{315},768^{630},1536^{12600}]
  9-seqs       : [1536^{51975}]

Name = t12n225
  Generator    : (1,6,4,8,2,5,3,7)(9,11)(10,12)
                 (1,2)(3,4)(5,12,8,10)(6,11,7,9)
  Order        : 1536
  Parity       : Odd
  Imprimitive  : Yes
  Shape        : [0111110000000000001101110000011101000000000000000000
                  000000100000000110000000001]
  2-sets       : [6^{1},12^{1},48^{1}]
  3-sets       : [12^{1},48^{1},64^{1},96^{1}]
  4-sets       : [3^{1},12^{1},48^{2},96^{2},192^{1}]
  5-sets       : [24^{1},48^{2},96^{1},192^{3}]
  6-sets       : [8^{1},12^{1},24^{1},48^{3},64^{1},96^{1},192^{1},
                 384^{1}]
  2-seqs       : [12^{1},24^{1},96^{1}]
  3-seqs       : [24^{3},96^{3},192^{3},384^{1}]
  4-seqs       : [24^{3},96^{3},192^{18},384^{9},768^{6}]
  5-seqs       : [192^{45},384^{45},768^{60},1536^{15}]
  6-seqs       : [192^{45},384^{150},768^{270},1536^{255}]
  7-seqs       : [384^{315},768^{630},1536^{2205}]
  8-seqs       : [384^{315},768^{630},1536^{12600}]
  9-seqs       : [1536^{51975}]

Name = t12n228
  Generator    : (1,5,10,2,7,12,3,6,9)(4,8,11)
                 (1,10,6,2,12,8,4,11,7)(3,9,5)
  Order        : 1728
  Parity       : Even
  Imprimitive  : Yes
  Shape        : [01010100000101010000000000000000000000000000000000
                  00000000000000000000010000]
  2-sets       : [18^{1},48^{1}]
  3-sets       : [12^{1},64^{1},72^{2}]
  4-sets       : [3^{1},48^{2},108^{1},288^{1}]
  5-sets       : [12^{2},72^{2},192^{1},432^{1}]
  6-sets       : [18^{2},48^{2},216^{1},288^{2}]
  2-seqs       : [36^{1},48^{2}]
  3-seqs       : [36^{2},144^{6},192^{2}]
  4-seqs       : [36^{2},144^{16},432^{6},576^{12}]
  5-seqs       : [144^{20},432^{40},576^{40},1728^{30}]
  6-seqs       : [432^{140},576^{60},1728^{330}]
  7-seqs       : [432^{280},1728^{2240}]
  8-seqs       : [432^{280},1728^{11480}]
  9-seqs       : [1728^{46200}]

Name = t12n229
  Generator    : (1,11,7,2,9,5)(3,12,8,4,10,6)
                 (1,2,3)(6,8,7)(9,12)(10,11)
  Order        : 1728
  Parity       : Even
  Imprimitive  : Yes
  Shape        : [01010100000101010100000000000000000000000000000000
                  00000010000000000000000000]
  2-sets       : [18^{1},48^{1}]
  3-sets       : [12^{1},64^{1},72^{2}]
  4-sets       : [3^{1},48^{2},108^{1},288^{1}]
  5-sets       : [12^{2},72^{2},192^{1},432^{1}]
  6-sets       : [18^{2},48^{2},72^{3},288^{2}]
  2-seqs       : [36^{1},48^{2}]
  3-seqs       : [36^{2},144^{6},192^{2}]
  4-seqs       : [36^{2},144^{16},432^{6},576^{12}]
  5-seqs       : [144^{20},432^{40},576^{40},1728^{30}]
  6-seqs       : [432^{140},576^{60},1728^{330}]
  7-seqs       : [432^{280},1728^{2240}]
  8-seqs       : [432^{280},1728^{11480}]
  9-seqs       : [1728^{46200}]

Name = t12n232
  Generator    : (1,11,3,12)(2,10)(4,7,6,8)(5,9)
                 (1,4,9)(2,6,7,3,5,8)(10,12)
  Order        : 1944
  Parity       : Even
  Imprimitive  : Yes
  Shape        : [00010010000100010100000000000010000000000000000000
                  01000000000000000001010000]
  2-sets       : [12^{1},54^{1}]
  3-sets       : [4^{1},108^{2}]
  4-sets       : [36^{1},54^{1},81^{1},324^{1}]
  5-sets       : [36^{1},108^{1},324^{2}]
  6-sets       : [6^{1},108^{4},486^{1}]
  2-seqs       : [24^{1},108^{1}]
  3-seqs       : [24^{1},216^{3},324^{2}]
  4-seqs       : [216^{10},648^{12},972^{2}]
  5-seqs       : [216^{20},648^{80},1944^{20}]
  6-seqs       : [216^{20},648^{360},1944^{220}]
  7-seqs       : [648^{1120},1944^{1680}]
  8-seqs       : [648^{2240},1944^{9520}]
  9-seqs       : [648^{2240},1944^{40320}]

Name = t12n234
  Generator    : (1,8,2,7,3,9)(4,11,6,12,5,10)
                 (1,10,6)(2,11,4)(3,12,5)(7,8,9)
  Order        : 1944
  Parity       : Even
  Imprimitive  : Yes
  Shape        : [0001011000010001010000000000000000000000000000001
                  01000010000000000001010000]
  2-sets       : [12^{1},54^{1}]
  3-sets       : [4^{1},108^{2}]
  4-sets       : [36^{1},54^{1},81^{1},324^{1}]
  5-sets       : [36^{1},108^{1},324^{2}]
  6-sets       : [6^{1},108^{4},486^{1}]
  2-seqs       : [24^{1},108^{1}]
  3-seqs       : [24^{1},216^{3},324^{2}]
  4-seqs       : [216^{10},648^{12},972^{2}]
  5-seqs       : [216^{20},648^{80},1944^{20}]
  6-seqs       : [216^{20},648^{360},1944^{220}]
  7-seqs       : [648^{1120},1944^{1680}]
  8-seqs       : [648^{2240},1944^{9520}]
  9-seqs       : [648^{2240},1944^{40320}]

Name = t12n235
  Generator    : (1,8,5,11,2,7,6,12)(3,10)(4,9)
                 (1,5,3,2,6,4)(7,9)(8,10)
                 (1,10)(2,9)(3,8,6,12,4,7,5,11)
  Order        : 2304
  Parity       : Odd
  Imprimitive  : Yes
  Shape        : [0111110000010110011011100010101010000000000000000111
                  00001010000000000100000000]
  2-sets       : [6^{1},24^{1},36^{1}]
  3-sets       : [16^{1},24^{1},36^{1},144^{1}]
  4-sets       : [6^{1},9^{1},24^{1},72^{1},96^{1},144^{2}]
  5-sets       : [12^{1},36^{1},48^{1},72^{1},144^{1},192^{1},288^{1}]
  6-sets       : [2^{1},18^{1},64^{1},72^{3},144^{1},192^{1},288^{1}]
  2-seqs       : [12^{1},48^{1},72^{1}]
  3-seqs       : [48^{3},72^{3},96^{1},288^{3}]
  4-seqs       : [48^{3},72^{3},96^{6},288^{18},576^{4},1152^{3}]
  5-seqs       : [96^{15},288^{45},576^{40},1152^{30},2304^{10}]
  6-seqs       : [96^{15},288^{45},576^{180},1152^{135},2304^{170}]
  7-seqs       : [576^{420},1152^{315},2304^{1470}]
  8-seqs       : [576^{420},1152^{315},2304^{8400}]
  9-seqs       : [2304^{34650}]
```

```
Name = t12n237
  Generator    : (1,12,5,7)(2,11,6,8)(3,10,4,9)
                 (1,9,4,7,6,11,2,10,3,8,5,12)
  Order        : 2304
  Parity       : Odd
  Imprimitive  : Yes
  Shape        : [010101000001010001010110000110101000000000000000000101
                  0001001000000000001000000001]
  2-sets       : [6^{1},24^{1},36^{1}]
  3-sets       : [16^{1},24^{1},36^{1},144^{1}]
  4-sets       : [6^{1},9^{1},24^{1},72^{1},96^{1},144^{2}]
  5-sets       : [12^{1},36^{1},48^{1},72^{1},144^{1},192^{1},288^{1}]
  6-sets       : [2^{1},18^{1},64^{1},72^{3},144^{1},192^{1},288^{1}]
  2-seqs       : [12^{1},48^{1},72^{1}]
  3-seqs       : [48^{3},72^{3},96^{1},288^{3}]
  4-seqs       : [48^{3},72^{3},96^{6},288^{18},576^{4},1152^{3}]
  5-seqs       : [96^{15},288^{45},576^{40},1152^{30},2304^{10}]
  6-seqs       : [96^{15},288^{45},576^{180},1152^{135},2304^{170}]
  7-seqs       : [576^{420},1152^{315},2304^{1470}]
  8-seqs       : [576^{420},1152^{315},2304^{8400}]
  9-seqs       : [2304^{34650}]

Name = t12n238
  Generator    : (1,9,4,12,2,10,3,11)(5,8,6,7)
                 (1,7,5,10,4,11,2,8,6,9,3,12)
  Order        : 2304
  Parity       : Odd
  Imprimitive  : Yes
  Shape        : [011111000001011001101110001010101000000000000000000111
                  0000101000000000000010000001]
  2-sets       : [6^{1},24^{1},36^{1}]
  3-sets       : [16^{1},24^{1},36^{1},144^{1}]
  4-sets       : [6^{1},9^{1},24^{1},72^{1},96^{1},144^{2}]
  5-sets       : [12^{1},36^{1},48^{1},72^{1},144^{1},192^{1},288^{1}]
  6-sets       : [2^{1},18^{1},64^{1},72^{3},144^{1},192^{1},288^{1}]
  2-seqs       : [12^{1},48^{1},72^{1}]
  3-seqs       : [48^{3},72^{3},96^{1},288^{3}]
  4-seqs       : [48^{3},72^{3},96^{6},288^{18},576^{4},1152^{3}]
  5-seqs       : [96^{15},288^{45},576^{40},1152^{30},2304^{10}]
  6-seqs       : [96^{15},288^{45},576^{180},1152^{135},2304^{170}]
  7-seqs       : [576^{420},1152^{315},2304^{1470}]
  8-seqs       : [576^{420},1152^{315},2304^{8400}]
  9-seqs       : [2304^{34650}]

Name = t12n240
  Generator    : (1,12,3,7,5,10,2,11,4,8,6,9)
                 (1,8,6,10,4,12)(2,7,5,9,3,11)
                 (1,7,2,8)(3,11)(4,12)(5,10)(6,9)
  Order        : 2304
  Parity       : Odd
  Imprimitive  : Yes
  Shape        : [111111000001111001001110000011101000000000000000001111
                  0010001000000000000000000001]
  2-sets       : [6^{1},24^{1},36^{1}]
  3-sets       : [16^{1},24^{1},36^{1},144^{1}]
  4-sets       : [6^{1},9^{1},24^{1},72^{1},96^{1},144^{2}]
  5-sets       : [12^{1},36^{1},48^{1},72^{1},144^{1},192^{1},288^{1}]
  6-sets       : [2^{1},18^{1},64^{1},72^{5},192^{1},288^{1}]
  2-seqs       : [12^{1},48^{1},72^{1}]
  3-seqs       : [48^{3},72^{3},96^{1},288^{3}]
  4-seqs       : [48^{3},72^{3},96^{6},288^{18},576^{10}]
  5-seqs       : [96^{15},288^{45},576^{100},1152^{20}]
  6-seqs       : [96^{15},288^{45},576^{450},1152^{300},2304^{20}]
  7-seqs       : [576^{1050},1152^{2100},2304^{420}]
  8-seqs       : [576^{1050},1152^{8400},2304^{4200}]
  9-seqs       : [1152^{18900},2304^{25200}]

Name = t12n241
  Generator    : (3,4)(7,10,11)(8,9,12)
                 (1,11,5,7)(2,12,6,8)(3,10)(4,9)
  Order        : 2304
  Parity       : Odd
  Imprimitive  : Yes
  Shape        : [111111000001111001001110000011101000000000000001111
                  0010001000000000001010000000]
  2-sets       : [6^{1},24^{1},36^{1}]
  3-sets       : [16^{1},24^{1},36^{1},144^{1}]
  4-sets       : [6^{1},9^{1},24^{1},72^{1},96^{1},144^{2}]
  5-sets       : [12^{1},36^{1},48^{1},72^{1},144^{1},192^{1},288^{1}]
  6-sets       : [2^{1},18^{1},64^{1},72^{3},144^{1},192^{1},288^{1}]
  2-seqs       : [12^{1},48^{1},72^{1}]
  3-seqs       : [48^{3},72^{3},96^{1},288^{3}]
  4-seqs       : [48^{3},72^{3},96^{6},288^{18}.576^{10}]
  5-seqs       : [96^{15},288^{45},576^{100},1152^{20}]
  6-seqs       : [96^{15},288^{45},576^{450},1152^{300},2304^{20}]
  7-seqs       : [576^{1050},1152^{2100},2304^{420}]
  8-seqs       : [576^{1050},1152^{8400},2304^{4200}]
  9-seqs       : [1152^{18900},2304^{25200}]

Name = t12n248
  Generator    : (1,10,6,7)(2,11,4,8)(3,12,5,9)
                 (1,8)(2,7)(3,9)(4,11,5,10,6,12)
                 (1,4,2,5,3,6)(7,11,9,12,8,10)
  Order        : 2592
  Parity       : Odd
  Imprimitive  : Yes
  Shape        : [011111101101011101000000000000001010000000000000001011
                  101000100000000000000000001]
  2-sets       : [12^{1},18^{1},36^{1}]
  3-sets       : [4^{1},36^{1},72^{1},108^{1}]
  4-sets       : [12^{1},18^{1},24^{1},36^{1},81^{1},108^{1},216^{1}]
  5-sets       : [12^{1},24^{1},36^{1},72^{1},108^{1},216^{1},324^{1}]
  6-sets       : [2^{1},4^{1},72^{3},108^{2},162^{1},324^{1}]
  2-seqs       : [24^{1},36^{1},72^{1}]
  3-seqs       : [24^{1},72^{3},144^{3},216^{3}]
  4-seqs       : [72^{10},144^{4},288^{3},432^{18},648^{3}]
  5-seqs       : [72^{20},288^{10},432^{60},864^{30},1296^{30}]
  6-seqs       : [72^{20},288^{10},432^{120},864^{210},1296^{150},
                 2592^{90}]
  7-seqs       : [432^{140},864^{770},1296^{420},2592^{1050}]
  8-seqs       : [864^{1680},1296^{560},2592^{6860}]
  9-seqs       : [864^{1680},2592^{30240}]
```

```
Name = t12n249
  Generator    : (1,6,2,5,3,4)(7,11,8,10,9,12)
                 (1,8,10,5,2,7,12,4)(3,9,11,6)
  Order        : 2592
  Parity       : Even
  Imprimitive  : Yes
  Shape        : [0101011010010101010101001001001000000000000000000001
                  0000001000000000000010000000]
  2-sets       : [12^{1},18^{1},36^{1}]
  3-sets       : [4^{1},36^{1},72^{1},108^{1}]
  4-sets       : [12^{1},18^{1},24^{1},36^{1},81^{1},108^{1},216^{1}]
  5-sets       : [12^{1},24^{1},36^{1},72^{1},108^{1},216^{1},324^{1}]
  6-sets       : [2^{1},4^{1},72^{3},108^{2},162^{1},324^{1}]
  2-seqs       : [24^{1},36^{1},72^{1}]
  3-seqs       : [24^{1},72^{3},144^{3},216^{3}]
  4-seqs       : [72^{10},144^{4},288^{3},432^{18},648^{3}]
  5-seqs       : [72^{20},288^{10},432^{60},864^{30},1296^{30}]
  6-seqs       : [72^{20},288^{10},432^{120},864^{210},1296^{150},
                 2592^{90}]
  7-seqs       : [432^{140},864^{770},1296^{420},2592^{1050}]
  8-seqs       : [864^{1680},1296^{560},2592^{6860}]
  9-seqs       : [864^{1680},2592^{30240}]

Name = t12n262
  Generator    : (1,7,5,11,3,9,6,12)(2,8,4,10)
                 (2,3)(7,10,8,11)(9,12)
  Order        : 5184
  Parity       : Odd
  Imprimitive  : Yes
  Shape        : [0111111011010111010010100100001000000000000000001011
                  1010101000000000000010000000]
  2-sets       : [12^{1},18^{1},36^{1}]
  3-sets       : [4^{1},36^{1},72^{1},108^{1}]
  4-sets       : [12^{1},18^{1},24^{1},36^{1},81^{1},108^{1},216^{1}]
  5-sets       : [12^{1},24^{1},36^{1},72^{1},108^{1},216^{1},324^{1}]
  6-sets       : [2^{1},4^{1},72^{3},108^{2},162^{1},324^{1}]
  2-seqs       : [24^{1},36^{1},72^{1}]
  3-seqs       : [24^{1},72^{3},144^{3},216^{3}]
  4-seqs       : [72^{4},144^{7},288^{3},432^{18},648^{3}]
  5-seqs       : [144^{10},288^{10},432^{30},864^{45},1296^{30}]
  6-seqs       : [144^{10},288^{10},864^{180},1296^{60},1728^{45},
                 2592^{135}]
  7-seqs       : [864^{210},1728^{315},2592^{630},5184^{315}]
  8-seqs       : [1728^{840},2592^{840},5184^{3150}]
  9-seqs       : [1728^{840},5184^{15120}]

Name = t12n263
  Generator    : (1,10)(2,12,3,11)(4,9)(5,8)(6,7)
                 (1,8,6,11,2,7,5,10,3,9,4,12)
  Order        : 5184
  Parity       : Odd
  Imprimitive  : Yes
  Shape        : [0101011010110101010101101001001010000000000000000101
                  0101010100000000000000000001]
  2-sets       : [12^{1},18^{1},36^{1}]
  3-sets       : [4^{1},36^{1},72^{1},108^{1}]
  4-sets       : [12^{1},18^{1},24^{1},36^{1},81^{1},108^{1},216^{1}]
  5-sets       : [12^{1},24^{1},36^{1},72^{1},108^{1},216^{1},324^{1}]
  6-sets       : [2^{1},4^{1},72^{3},108^{2},162^{1},324^{1}]
  2-seqs       : [24^{1},36^{1},72^{1}]
  3-seqs       : [24^{1},72^{3},144^{3},216^{3}]
  4-seqs       : [72^{4},144^{7},288^{3},432^{18},648^{3}]
  5-seqs       : [144^{10},288^{10},432^{30},864^{45},1296^{30}]
  6-seqs       : [144^{10},288^{10},864^{180},1296^{60},1728^{45},
                 2592^{135}]
  7-seqs       : [864^{210},1728^{315},2592^{630},5184^{315}]
  8-seqs       : [1728^{840},2592^{840},5184^{3150}]
  9-seqs       : [1728^{840},5184^{15120}]

Name = t12n267
  Generator    : (1,7,12,4,3,9,10,5,2,3,11,6)
                 (1,11)(2,12)(3,10)(5,6)(7,8)
                 (1,6,3,4)(2,5)(7,10,8,12)(9,11)
  Order        : 5184
  Parity       : Odd
  Imprimitive  : Yes
  Shape        : [0111111011010111010010000100001010000000000000001011
                  101000100000000000000000001]
  2-sets       : [12^{1},18^{1},36^{1}]
  3-sets       : [4^{1},36^{1},72^{1},108^{1}]
  4-sets       : [12^{1},18^{1},24^{1},36^{1},81^{1},108^{1},216^{1}]
  5-sets       : [12^{1},24^{1},36^{1},72^{1},108^{1},216^{1},324^{1}]
  6-sets       : [2^{1},4^{1},72^{3},108^{2},162^{1},324^{1}]
  2-seqs       : [24^{1},36^{1},72^{1}]
  3-seqs       : [24^{1},72^{3},144^{3},216^{3}]
  4-seqs       : [72^{4},144^{7},288^{3},432^{18},648^{3}]
  5-seqs       : [144^{10},288^{10},432^{30},864^{45},1296^{30}]
  6-seqs       : [144^{10},288^{10},864^{180},1296^{60},1728^{45},
                 2592^{135}]
  7-seqs       : [864^{210},1728^{315},2592^{630},5184^{315}]
  8-seqs       : [1728^{840},2592^{840},5184^{3150}]
  9-seqs       : [1728^{840},5184^{15120}]

DEGREE 14

Name = t14n46
  Generator    : (1,2,5,9,12,8,4)(3,6,10,14,13,11,7)
                 (1,3)(2,6)(4,7)(5,10)(8,11)(9,13)(12,14)
  Order        : 5040
  Parity       : Odd
  Imprimitive  : Yes
  Shape        : [0001001000000010001000100000000000000000011000000000
                  000000000000001000000000100000000000100001000000000000
                  0001000000000000000000000001000000000]
  2-sets       : [7^{1},42^{2}]
  3-sets       : [70^{1},84^{1},210^{1}]
  4-sets       : [21^{1},70^{1},210^{3},280^{1}]
  5-sets       : [42^{1},210^{2},280^{1},420^{1},840^{1}]
  6-sets       : [14^{1},35^{1},84^{1},140^{1},210^{2},420^{2},630^{1},
                 840^{1}]
  7-sets       : [2^{1},14^{1},42^{1},70^{1},84^{1},280^{1},420^{2},
                 840^{1},1260^{1}]
  2-seqs       : [14^{1},84^{2}]
  3-seqs       : [84^{6},420^{4}]
  4-seqs       : [84^{6},420^{24},1680^{8}]
  5-seqs       : [420^{60},1680^{80},5040^{16}]
  6-seqs       : [420^{60},1680^{360},5040^{304}]
  7-seqs       : [1680^{840},5040^{3152}]
  8-seqs       : [1680^{840},5040^{23744}]
  9-seqs       : [5040^{144144}]

Name = t14n47
  Generator    : (1,2)(3,4)(5,6)(7,8)(9,10)(11,12)(13,14)
                 (1,3,5,7,9,11,13)(2,4,6,8,10,12,14)
                 (3,5,7)(4,6,8)
  Order        : 5040
  Parity       : Odd
  Imprimitive  : Yes
  Shape        : [00010010000001000100010000000000000000001000000000
                  00000000000000100000000010000000000000100000000000
                  00010000000000000000000001000000001]
  2-sets       : [7^{1},42^{2}]
  3-sets       : [70^{1},84^{1},210^{1}]
  4-sets       : [21^{1},70^{1},210^{3},280^{1}]
  5-sets       : [42^{1},210^{2},280^{1},420^{1},840^{1}]
  6-sets       : [14^{1},35^{1},84^{1},140^{1},210^{2},420^{2},630^{1},
                 840^{1}]
  7-sets       : [2^{1},14^{1},42^{1},70^{1},84^{1},280^{1},420^{2},
                 840^{1},1260^{1}]
  2-seqs       : [14^{1},84^{2}]
  3-seqs       : [84^{6},420^{4}]
  4-seqs       : [84^{6},420^{24},1680^{8}]
  5-seqs       : [420^{60},1680^{80},5040^{16}]
  6-seqs       : [420^{60},1680^{360},5040^{304}]
  7-seqs       : [1680^{840},5040^{3152}]
  8-seqs       : [1680^{840},5040^{23744}]
  9-seqs       : [5040^{144144}]

DEGREE 15

Name = t15n31
  Generator    : (1,2,3,4,5)
                 (1,6,11)(2,7,12)(3,8,13)(4,9,14)(5,10,15)
                 (2,5)(3,4)(6,11)(7,15)(8,14)(9,13)(10,12)
  Order        : 750
  Parity       : Odd
  Imprimitive  : Yes
  Shape        : [00000010000000000000000001000000000000000000000000
                  00010000000000000000000000100000100000000000000000
                  00000000000000000000000000000000000000000000000000
                  00000000100000000000000001]
  2-sets       : [15^{2},75^{1}]
  3-sets       : [15^{2},125^{1},150^{2}]
  4-sets       : [15^{1},75^{2},150^{3},375^{2}]
  5-sets       : [3^{1},150^{5},375^{4},750^{1}]
  6-sets       : [30^{1},75^{2},125^{2},150^{3},375^{3},750^{4}]
  7-sets       : [30^{2},75^{1},150^{2},375^{6},750^{5}]
  2-seqs       : [30^{2},150^{1}]
  3-seqs       : [30^{6},150^{12},750^{1}]
  4-seqs       : [30^{12},150^{96},750^{24}]
  5-seqs      : [30^{12},150^{600},750^{360}]
  6-seqs      : [150^{3024},750^{4200}]
  7-seqs      : [150^{12096},750^{40824}]
  8-seqs      : [150^{36288},750^{338688}]
  9-seqs      : [150^{72576},750^{2407104}]

Name = t15n32
  Generator   : (1,2,3,4,5)
                (1,6,11)(2,7,12)(3,8,13)(4,9,14)(5,10,15)
                (6,11)(7,12)(8,13)(9,14)(10,15)
  Order       : 750
  Parity      : Odd
  Imprimitive : Yes
  Shape       : [00001000000000000000000001000000000000000000000000
                 00010000100000000000000000100000100000000000000000
                 00000000000000000000000000000000000000000000000000
                 0000001000001000000000001]
  2-sets      : [15^{2},75^{1}]
  3-sets      : [15^{2},125^{1},150^{2}]
  4-sets      : [15^{1},75^{2},150^{3},375^{2}]
  5-sets      : [3^{1},150^{5},375^{4},750^{1}]
  6-sets      : [30^{1},75^{2},125^{2},150^{3},375^{3},750^{4}]
  7-sets      : [30^{2},75^{1},150^{2},375^{6},750^{5}]
  2-seqs      : [15^{4},150^{1}]
  3-seqs      : [15^{12},150^{12},750^{1}]
  4-seqs      : [15^{24},150^{96},750^{24}]
  5-seqs      : [15^{24},150^{600},750^{360}]
  6-seqs      : [150^{3024},750^{4200}]
  7-seqs      : [150^{12096},750^{40824}]
  8-seqs      : [150^{36288},750^{338688}]
  9-seqs      : [150^{72576},750^{2407104}]

Name = t15n48
  Generator   : (1,2,3,4,5)
                (1,6,11)(2,7,12)(3,8,13)(4,9,14)(5,10,15)
                (6,11)(7,15,10,12)(8,14,9,13)
                *(2,5)(3,4)(7,10)(8,9)
                *(7,10)(8,9)(12,15)(13,14)
  Order       : 3000
  Parity      : Odd
  Imprimitive : Yes
  Shape       : [00010010000000000000000001000000000000000001000000
                 00010001000000000000000001100000100000000000000000
                 00000000000000000000000000000000000000000000000000
                 0000000010000000000000001]
  2-sets      : [15^{2},75^{1}]
  3-sets      : [15^{2},125^{1},150^{2}]
  4-sets      : [15^{1},75^{2},150^{3},375^{2}]
  5-sets      : [3^{1},150^{5},375^{4},750^{1}]
  6-sets      : [30^{1},75^{2},125^{2},150^{3},375^{3},750^{4}]
  7-sets      : [30^{2},75^{1},150^{2},375^{6},750^{5}]
  2-seqs      : [30^{2},150^{1}]
  3-seqs      : [30^{6},300^{6},750^{1}]
  4-seqs      : [30^{12},300^{24},600^{12},1500^{12}]
  5-seqs      : [30^{12},300^{60},600^{120},1500^{60},3000^{60}]
  6-seqs      : [300^{72},600^{720},1500^{180},3000^{960}]
  7-seqs      : [600^{3024},1500^{252},3000^{10080}]
```

```
    8-seqs      :  [600^{9072},3000^{84672}]
    9-seqs      :  [600^{18144},3000^{601776}]

Name = t15n51
    Generator   :  (1,2,3,4,5)
                   (1,6,11)(2,7,12)(3,8,13)(4,9,14)(5,10,15)
                   (6,11)(7,12)(8,13)(9,14)(10,15)
                   (2,5)(3,4)(7,10)(8,9)
                   *(7,10)(8,9)(12,15)(13,14)
    Order       :  3000
    Parity      :  Odd
    Imprimitive :  Yes
    Shape       :  [00011000000000000000000001000000000000000000010000
                    00010001100000000000000000100000100000000000000000
                    00000000000000000000000000000000000000000000000000
                    0000001000001000000000001]
    2-sets      :  [15^{2},75^{1}]
    3-sets      :  [15^{2},125^{1},150^{2}]
    4-sets      :  [15^{1},75^{2},150^{3},375^{2}]
    5-sets      :  [3^{1},150^{5},375^{4},750^{1}]
    6-sets      :  [30^{1},75^{2},125^{2},150^{3},375^{3},750^{4}]
    7-sets      :  [30^{2},75^{1},150^{2},375^{6},750^{5}]
    2-seqs      :  [30^{2},150^{1}]
    3-seqs      :  [30^{6},300^{6},750^{1}]
    4-seqs      :  [30^{12},300^{24},600^{12},1500^{12}]
    5-seqs      :  [30^{12},300^{60},600^{120},1500^{60},3000^{60}]
    6-seqs      :  [300^{72},600^{720},1500^{180},3000^{960}]
    7-seqs      :  [600^{3024},1500^{252},3000^{10080}]
    8-seqs      :  [600^{9072},3000^{84672}]
    9-seqs      :  [600^{18144},3000^{601776}]
```

Appendix F

Timing Data for r-sets

The following shows the timing data for r-sets, in the file *time.19*, obtained by running the main program with *m* as the input file. The columns under the heading "Time" corresponds to timing data for 2-sets, 3-sets, ...and so on.

```
Group      Time(ms)
-----      ---------
t5n3              0
t5n5              0
t6n9              3       0
t6n13             0       3
t6n14             0       0
t6n16             0       3
t8n26             3       3       7
t8n28             3       3       7
t8n29             3       3       7
t8n46             3       3       3
t8n47             3       3       3
t9n30             0       7       7
t9n31             0       7       7
t10n9             3       7      15      23
t10n10            3       7      15      23
t10n11            3       7      15      27
t10n12            3       7      15      27
t10n18            0       7      15      23
t10n20            3       7      15      19
t10n21            3       7      15      19
t10n37            3       7      15      19
t10n39            0       7      15      19
t10n42            0       7      15      19
t10n43            3       7      11      19
t12n1             3      11      42      93     167
t12n5             3      19      50     113     195
t12n34            3      15      39      82     132
t12n40            3      15      42      82     128
t12n145           7      11      39      74     109
t12n152           3      15      35      66      89
t12n153           3      15      35      62      93
t12n154           7      19      50      93     128
```

t12n155	3	15	35	74	105	
t12n168	3	19	46	85	117	
t12n171	3	15	35	66	93	
t12n172	7	11	39	66	89	
t12n174	7	19	46	85	117	
t12n180	3	15	35	66	93	
t12n183	3	11	35	62	85	
t12n196	3	15	35	62	82	
t12n197	7	19	46	82	109	
t12n209	7	11	35	66	85	
t12n210	3	19	46	85	117	
t12n212	3	15	35	62	85	
t12n214	7	19	46	85	117	
t12n216	7	11	39	62	82	
t12n217	3	19	46	85	109	
t12n221	7	11	35	62	85	
t12n223	3	19	46	82	113	
t12n225	3	15	35	62	85	
t12n228	3	15	35	62	82	
t12n229	3	15	35	62	82	
t12n232	3	15	31	62	82	
t12n234	3	15	35	58	82	
t12n235	3	19	46	82	109	
t12n237	3	15	35	62	82	
t12n238	3	15	35	62	82	
t12n240	3	19	46	82	113	
t12n241	7	11	39	62	82	
t12n248	3	19	46	82	109	
t12n249	3	15	35	62	82	
t12n262	3	15	35	62	82	
t12n263	3	15	35	62	82	
t12n267	7	19	46	82	105	
t14n46	3	27	70	156	246	320
t14n47	3	35	97	210	359	472
t15n31	7	42	140	335	632	929
t15n32	7	42	140	339	628	929
t15n48	15	62	207	503	917	1300
t15n51	15	62	207	499	917	1304

Appendix G

Timing Data for r-sequences

The following shows the timing data for r-sequences, in the file *time.q9*, obtained by running the main program with *in* as the input file. The columns under the heading "Time" corresponds to timing data for 2-sequences, 3-sequences, ... and so on.

```
Group     Time(ms)
-----     ---------
t5n3             0       0
t5n5             0       0
t6n9             0       3       0
t6n13            0       0       3
t6n14            0       3       0
t6n16            0       0       0
t8n26            0       3       7      19      62
t8n28            0       0       7      19      62
t8n29            0       3       3      23      62
t8n46            0       3       3       3      11
t8n47            0       3       0       7       7
t9n30            0       3       3      11      35      89
t9n31            0       3       3      11      35      62
t10n9            3       0      15      74     371    1585    5124
t10n10           0       3      15      74     367    1574    5089
t10n11           0       3      11      58     316    1351    4370
t10n12           0       3      11      62     316    1347    4351
t10n18           0       3       7      42     207     886    2835
t10n20           0       0      11      42     207     878    2827
t10n21           0       3       7      42     207     886    2835
t10n37           0       3       3      15      54     175     453
t10n39           0       3       3      11      35      97     242
t10n42           0       3       3       7      15      31      62
t10n43           3       0       3      11      15      31      46
t12n1            0      11     140    1328   10776   73065  405017 1773851
t12n5            0      15     140    1328   10800   73210  404970 1773999
t12n34           0       7      42     332    2410   15545   84018  361027
t12n40           3       3      46     332    2410   15565   84033  361004
t12n145          0       7      27     160     843    4226   19502   78096
t12n152          0       3      19     105     691    4136   21182   88365
t12n153          0       3      19     105     695    4124   21166   88275
t12n154          3       3      31     156     847    4245   19623   78541
```

t12n155	3	3	27	160	851	4253	19654	78690
t12n168	0	7	19	109	581	2886	12768	48434
t12n171	0	7	19	109	581	2874	12749	48305
t12n172	3	3	23	109	581	2882	12756	48356
t12n174	3	3	19	109	581	2882	12768	48391
t12n180	0	7	15	70	378	2195	11346	47610
t12n183	3	3	11	66	378	2191	11358	47680
t12n196	3	3	15	66	332	1628	7452	29689
t12n197	3	3	15	66	335	1632	7495	29869
t12n209	0	3	15	70	339	1644	7214	27209
t12n210	0	7	19	89	410	1765	7046	25377
t12n212	0	3	15	66	332	1640	7210	27138
t12n214	0	3	23	89	410	1769	7054	25428
t12n216	3	3	15	62	335	1636	7183	27103
t12n217	0	3	15	70	339	1640	7187	27138
t12n221	3	3	15	66	304	1390	6136	24068
t12n223	0	3	15	66	308	1406	6175	24240
t12n225	0	3	15	66	300	1398	6144	24111
t12n228	0	3	15	50	214	1011	4792	19475
t12n229	3	3	15	50	214	1015	4796	19490
t12n232	0	3	11	46	261	1257	5530	20779
t12n234	0	3	11	50	257	1261	5542	20834
t12n235	0	3	15	54	230	968	4113	15940
t12n237	3	3	11	54	226	960	4077	15791
t12n238	3	3	11	54	230	960	4093	15842
t12n240	0	3	15	66	316	1456	5905	20213
t12n241	3	3	15	66	320	1460	5937	20315
t12n248	0	3	15	58	238	1011	4007	14448
t12n249	3	3	11	58	242	1007	4007	14436
t12n262	0	3	15	50	187	667	2292	7659
t12n263	3	3	11	50	191	671	2304	7706
t12n267	3	3	15	46	191	667	2296	7683
t14n46	0	7	19	85	414	2327	14319	85939
t14n47	3	3	23	89	425	2425	14854	89131
t15n31	3	7	50	414	3355	26588	199909	1389231
t15n32	3	7	58	441	3488	27197	203576	1410905
t15n48	3	3	31	160	1050	7573	55320	380850
t15n51	0	7	31	160	1054	7573	55328	380877

Appendix H

Re-formatting the Output

The following C program re-formats the output obtained from the main process. If this output file is *out*, and this program is in the file *format.c*, which is compiled to *format*, the UNIX command

format <out

produces outputs that is similar to the one shown in Figure 4.4.

```
#include <stdio.h>

short j,k,maxrset,maxrseq;
char ch,str[132],sdum[2],ldum[30];
short out_index;
FILE *fptr,*out[4];
char *fname[4];
short first_group[4],first_size;
short degree,ig;
char generator[30][52];
char parity[5],imprimitive[4];
char gname[40],order[20];

void get_generator()
{
  short cmp_res;
  scanf ("%s%s%s", ldum, sdum, &generator[0][0]);
  ig = 1;
  scanf ("%s", &generator[ig][0]);
  cmp_res = strcmp (&generator[ig][0], "Order");
  while (cmp_res != 0) {
    ig++;
    scanf ("%s", &generator[ig][0]);
    cmp_res = strcmp (&generator[ig][0], "Order");
  }
}

void print_generator()
```

```
{
  short fg=1,i;
  for (i=0; i<ig; i++)
    if (generator[i][0]!='*') {
      if (fg==1) {
fg = 0;
fprintf (fptr, " {");
      } else
fprintf (fptr, ",\n  ");
      fprintf (fptr, "'%s'", &generator[i][0]);
    }
  fprintf (fptr, "},\n");
}

void get_print_shape()
{
  short i,count,finish;
  scanf ("%s%s%s", ldum, sdum, str);
  fprintf (fptr, " [%c", str[1]);
  i = 2;
  count = 2;
  finish = 0;
  while (finish == 0) {
    while ( (str[i]!=']') & (str[i]!='\0') ) {
      if (count%20==1) fprintf (fptr, ",\n  %c", str[i]);
      else         fprintf (fptr, ", %c", str[i]);
      i++;
      count++;
    }
    if (str[i]==']') finish = 1;
    else {
      scanf ("%s", str);
      i = 0;
    }
  }
 fprintf (fptr, "],\n");
}

void get_print_size()
{
  long size,freq;
  char c1,c2,c3,c4;
  short i,ii=0;
  if (first_size==1) {
    first_size = 0;
    fprintf (fptr, " [");
  } else {
    fprintf (fptr, ",\n [");
  }
  scanf ("%s%s%c%c", ldum, sdum, &c1, &c2);
  do {
    scanf ("%d%c%c%d%c%c", &size, &c1,&c2, &freq, &c3,&c4);
    for (i=1; i<=freq; i++) {
      ii++;
      if (ii!=1) {
        if (ii%10==1) fprintf (fptr, ",\n  ");
        else         fprintf (fptr, ", ");
        }
      fprintf (fptr, "%d", size);
    }
  }
  while (c4!=']');
  fprintf (fptr, "]");
}

void terminate_file()
{
  if (first_group[0] != 1) fprintf (out[0], "],");
  fclose  (out[0]);

  if (first_group[1] != 1) fprintf (out[1], "]");
  fprintf (out[1], "}],");
  fclose  (out[1]);

  if (first_group[2] != 1) fprintf (out[2], "],");
  fclose  (out[2]);

  if (first_group[3] != 1) fprintf (out[3], "]");
  fprintf (out[3], "},");
  fclose  (out[3]);
}

void start_file()
{
  short j;
  for (j=0; j<4; j++) {
    if (degree<10)
      fname[j][0] = '0' + degree;
    else
      fname[j][0] = 'a' + degree - 10 ;
    out[j] = fopen (fname[j], "w");
    first_group[j] = 1;
  }
  fprintf (out[0], "{");
  fprintf (out[2], "[{");
}


/************************** M A I N ****************************/
main()
{
  fname[0] = "0oi";
  fname[1] = "0op";
  fname[2] = "0ei";
  fname[3] = "0ep";
  while (scanf("%s",str)!=EOF) {
    if (strcmp(str,"DEGREE")==0) {
      if (degree>0) terminate_file();
      scanf("%hd",&degree);
      maxrset - degree/2;
      if (maxrset < 2) maxrset = 2;
      maxrseq = degree - 2;
      if (maxrseq<2) maxrseq=2;
```

```
        if (maxrseq>9) maxrseq=9;
        scanf("%s",str);
        start_file();
      }
      scanf ("%s%s", sdum, gname);
      get_generator();
      scanf ("%s%s", sdum, order);
      scanf ("%s%s%s", ldum, sdum, parity);
      scanf ("%s%s%s", ldum, sdum, imprimitive);
      if ( strcmp (parity,"Odd")==0 ) {
        if ( strcmp (imprimitive, "Yes") == 0 ) out_index = 0;
        else out_index = 1;
      } else {
        if ( strcmp (imprimitive, "Yes") == 0 ) out_index = 2;
        else out_index = 3;
      }
      fptr = out[out_index];
      if (first_group[out_index] == 1)
        first_group[out_index] = 0;
      else
        fprintf (fptr, "],\n\n");
      fprintf (fptr, "['%s', '%s',\n", gname, order);
      print_generator();
      get_print_shape();
      first_size = 1;
      for (j=2; j<=maxrset; j++) get_print_size();
      get_print_size();
      for (j=3; j<=maxrseq; j++) scanf ("%s%s%s", ldum, sdum, str);
    }
    terminate_file();
  }

  /* End. */
```

Appendix I

ISOM rountines

Here is a part of the declaration file *groupdcl.h* showing the ISOM routines that are used in the research work.

```
/* The data structures are designed to allow the library routines to
/* operate independently of the degree of the permutation group, while using
/* the minimum required space.  This is done by using pointers to reference
/* all data structures which depend on the degree, and doing
/* all the allocations according to the size of the permutation group.

typedef enum {worse, indifferent, better, isom_exit} comprestype;
typedef short permval;
typedef permval *ptr_to_permvect;
typedef struct { /* declare a labelled branching */
  short permsize; /* actual size of the permutation. */
  gp_status_type gp_status;
  /* = is_dir_prod if built as symmetric or direct product group.
  /* = other_gp if not.
  /*  Invalid otherwise. Allows optimisation in
  /*  handling of the group in find_certificate */
  ptr_to_permvect base;    /* relative to base */
  ptr_to_permvect rank;    /* index relative to Omega,
                           /* value relative to base. */
  ptr_to_permvect orbits;    /* relative to Omega */
    /* Contains most recently manipulated orbit(s). Set by various
    /* routines in various ways.  Careful!! Reflects top level
    /* orbits of dir_prod_gps. */
  ptr_to_permvect t1; /* Strictly local temporary, of correct size.
    /* any procedure is free to use it, but it is not preserved accross
    /* a call.  Included to avoid the overhead of allocating and releasing
    /* temporaries, which cannot be simply declared. */
  point_type *point; /* perms: relative to Omega */
} perm_gp;  /* end declaration of a labelled branching */
typedef ptr_to_permvect *ptr_to_permmatrix;
typedef perm_gp *ptr_to_perm_gp;
typedef int base_index;  /* relative to base */
typedef int point_index; /* relative to Omega */

extern ptr_to_permvect Allocatepv(short size);
  /* allocate a permvect of the user required size. */

extern void disposepv(ptr_to_permvect pv);
  /* Dispose pv */

extern ptr_to_permmatrix allocatepm(short size);
  /* Allocate a permmatrix of the user required size. The first size
  /* elements will be used, and are initialized to NULL. */

extern void disposepm(ptr_to_permmatrix pm,
                      short size);
  /* Dispose pm, and its first size elements. */

extern ptr_to_perm_gp build_null_gp(ptr_to_perm_gp pg,
                                    short size);
  /* Build a trivial branching of the given size in pg.
  /* Return (resized) pg. The base will be 1..size. */

extern ptr_to_perm_gp symmetric_gp(ptr_to_perm_gp pg,
                                   short size);
  /* Build branching for the symmetric group of given size in pg.
  /* Return (resized) pg. Base will be 1..size. */

extern void disposepg(ptr_to_perm_gp pg);
  /* Dispose pg after disposing its components. */

extern void gporder(ptr_to_perm_gp branch,
                    double * rorder,
                    long * iorder);
  /* Calculate the group order of the group in branch, both
  /* as a real number (rorder) and as an integer number (iorder).
  /* For large groups, a machine integer may not be big enough
  /* to hold the order.  If this is the case, iorder
  /* is set to  -1. */

extern void find_certificate(ptr_to_perm_gp symgroup,
                             ptr_to_perm_gp autogroup,
                             comprestype (*test_part_perm)(),
                             short stats);
  /* Procedure to find the certificate of a combinatorical object A,
  /* under the action of the symmetry group symgroup.
  /* On entry, the automorphism group autogroup is either
  /* NULL or points to a perm_gp describing (part of) the group
  /* stabilizing the object.  On return, autogroup (if not NULL on entry)
  /* will point to the augmented automorphism group.
  /* The function test_part_perm compares the temporary maximum
  /* canonical form Atmax of A with A**choice[1..depth].
  /* It returns 'worse' if  A**choice[1..depth]   <  atmax
  /*            better                            >
  /*            isom_exit  to abandon the search for a certificate.
  /*            indifferent              otherwise.
  /* However, if depth=permsize, then returns indifferent
  /* iff A**choice[1..permsize] = Atmax.
  /* If depth=permsize and better is returned, ptmax will be updated.
  /* The parameter stats controls the printing of statistics.  When stats
  /*   = 0 - no statistics will be printed
  /*   = 1 - print time and number of accepted choices.
  /*   = 2 - print time and number of accepted choices at each level. */

extern void find_orbit(ptr_to_perm_gp pg,
                       point_index p,
                       base_index level,
                       ptr_to_permvect orbit,
                       ptr_to_permmatrix u);
  /* Find the orbit of 'p' in 'pg' fixing the first 'level' basepoints.
  /* Link the points of the orbit into a circular list in 'orbit'.
  /* If u <> NULL, compute the perm which maps p to each point of the
  /* orbit and store it into the corr. element of u. The identity
  /* element is not created for p. If u = NULL, only the orbit is
  /* computed. On entry, orbit[i] <= 0 for each i which may possibly
  /* be in the orbit. This allows several orbits to be built up in
  /* a single vector. Orbits are not sorted. */

extern void jerrum(ptr_to_perm_gp pg,
                   ptr_to_permmatrix genlist,
                   short nngens);
  /* Augment the (possibly trivial) labeled branching 'pg' to reflect the
  /* new set of 'nngens' generators in 'genlist'. */
```