

Classification of Cwatsets Through Order 23

Ben Goodwin

Dennis Lin

MS TR 00-05

December 6, 2000

Department of Mathematics
Rose-Hulman Institute of Technology
<http://www.rose-hulman.edu/Class/ma/HTML/index.html>

FAX: (812) 877-8883

Phone: (812) 877-8391

Classification of Cwatsets Through Order 23

Ben Goodwin Dennis Lin *

December 6, 2000

Abstract

A cwatset of order n can be represented by a transitive subgroup of S_n . Previous work [6] has shown that each conjugacy class of representation groups corresponds to an isomorphism class of cwatsets. We present a technique for determining whether a particular transitive subgroup of S_n can appear as the representation group for a cwatset of order n . Using this method, we provide a full classification of cwatset isomorphism classes through order 23.

1 Introduction

Definition 1 *A cwatset C is a subset of Z_2^d such that, for each element $w \in C$, $C^\sigma + w = C$ for some $\sigma \in S_d$, (see [7])*

As an example, consider the cwatset

$$F = \begin{Bmatrix} 0 & 0 & 0 \\ 1 & 1 & 0 \\ 1 & 0 & 1 \end{Bmatrix}$$

It is useful to think of F as a matrix in which the rows are formed by the elements of F . In this way, F is closed under addition, up to a reordering of the columns of the associated matrix.

Definition 2 *The set of all $(\sigma, w) \in S_d \wr C$ such that $C^\sigma + w = C$ form a group, known as the Ω -group C , or Ω_C .*

*Work supported by NSF grant 9619714

will begin with the bit 0. We also eliminate the all 0 column because it is possible to add an infinite number of them to a cwaset without changing it. This leaves us with $2^{n-1} - 1$ possible columns. Call this set \mathbf{Col}_n . For each column we assign an integer from 1 to $2^{n-1} - 1$ based on treating the column as a binary word and then converting that value to an integer. For example,

$$\begin{bmatrix} 0 \\ 1 \\ 1 \\ 0 \\ 1 \end{bmatrix} \rightarrow 01101 \rightarrow 13$$

For notational purposes, we label the column corresponding to an integer i i . For example, we would call the column above **13**. Often, we will refer to a generic column in \mathbf{Col}_n , for example \mathbf{c} :

$$\mathbf{c} = \begin{bmatrix} c_1 \\ c_2 \\ \vdots \\ c_n \end{bmatrix}$$

We will denote by \mathbf{c}^ρ the permutation of the entries in \mathbf{c} by some $\rho \in S_n$ (instead of $\mathbf{c}^{T\rho}$).

Also, we will use the notation \bar{w} to signify a word or row of C .

Definition 4 For $\mathbf{c} \in \mathbf{Col}_n$ of length n and a permutation $\rho \in S_n$ define the operation \times by:

$$\mathbf{c} \times \rho = \begin{bmatrix} c_1 \\ c_2 \\ \vdots \\ c_n \end{bmatrix}^{\rho^{-1}} + c_{1\rho} = \begin{bmatrix} c_{1\rho} \\ c_{2\rho} \\ \vdots \\ c_{n\rho} \end{bmatrix} + c_{1\rho}$$

As an example, consider the column **6** of length 4, and consider $(1, 2, 3, 4) \in S_4$. Then we have

Then, let \mathbf{c} be

$$\begin{aligned}
 \mathbf{c} &= \begin{bmatrix} c_1 \\ c_2 \\ \vdots \\ c_n \end{bmatrix} \\
 &= \begin{bmatrix} d_{1\rho^{-1}} \\ d_{2\rho^{-1}} \\ \vdots \\ d_{n\rho^{-1}} \end{bmatrix} + d_{1\rho^{-1}} \\
 &= \begin{bmatrix} d_{1\rho^{-1}} + d_{1\rho^{-1}} \\ d_{2\rho^{-1}} + d_{1\rho^{-1}} \\ \vdots \\ d_{n\rho^{-1}} + d_{1\rho^{-1}} \end{bmatrix}
 \end{aligned}$$

and thus

$$\begin{aligned}
 \mathbf{c} \times \rho &= \begin{bmatrix} c_{1\rho} \\ c_{2\rho} \\ \vdots \\ c_{n\rho} \end{bmatrix} + c_{1\rho} \\
 &= \begin{bmatrix} d_{1\rho\rho^{-1}} + d_{1\rho^{-1}} \\ d_{2\rho\rho^{-1}} + d_{1\rho^{-1}} \\ \vdots \\ d_{n\rho\rho^{-1}} + d_{1\rho^{-1}} \end{bmatrix}^{\rho^{-1}} + d_{1\rho\rho^{-1}} + d_{1\rho^{-1}} \\
 &= \begin{bmatrix} d_1 \\ d_2 \\ \vdots \\ d_n \end{bmatrix} + d_1
 \end{aligned}$$

Since we know that $d_1 = 0$, we have $\mathbf{c} \times \rho = \mathbf{d}$ for any $\mathbf{d} \in \text{Col}_n$. Thus, f is a surjection and therefore a bijection. ■

Proof. Let $r, s \in S_n$, let $c \in \text{Col}_n$, and let

$$\begin{aligned}
 \mathbf{d} &= \mathbf{c} \times r^{-1} \\
 &= \begin{bmatrix} c_1 \\ c_2 \\ \vdots \\ c_n \end{bmatrix} \times r^{-1} \\
 &= \begin{bmatrix} c_{1r^{-1}} \\ c_{2r^{-1}} \\ \vdots \\ c_{nr^{-1}} \end{bmatrix} + c_{1r^{-1}} \\
 &= \begin{bmatrix} d_1 \\ d_2 \\ \vdots \\ d_n \end{bmatrix}
 \end{aligned}$$

To show that $r^x s^x = (rs)^x$ we show that $c^{(r^x s^x)} = c^{(rs)^x}$. We have that

$$\begin{aligned}
 c^{r^x s^x} &= (\mathbf{c} \times r^{-1}) \times s^{-1} \\
 &= \mathbf{d} \times s^{-1} \\
 &= \begin{bmatrix} d_1 \\ d_2 \\ \vdots \\ d_n \end{bmatrix} \times s^{-1} \\
 &= \begin{bmatrix} d_{1s^{-1}} \\ d_{2s^{-1}} \\ \vdots \\ d_{ns^{-1}} \end{bmatrix} + d_{1s^{-1}}
 \end{aligned}$$

Since $r \in R \leq R_C$, there must exist (σ, b) such that

$$C^\sigma + b = C^{Tr}$$

This means that

$$\begin{bmatrix} \overline{0}^\sigma \\ \overline{w_2}^\sigma \\ \vdots \\ \overline{w_i}^\sigma \\ \vdots \\ \overline{w_n}^\sigma \end{bmatrix} + \overline{b} = \begin{bmatrix} \overline{w_1} \\ \overline{w_2} \\ \vdots \\ \overline{w_i} \\ \vdots \\ \overline{w_n} \end{bmatrix}^{Tr}$$

or

$$\begin{bmatrix} \overline{w_1}^\sigma \\ \overline{w_2}^\sigma \\ \vdots \\ \overline{w_i}^\sigma \\ \vdots \\ \overline{w_n}^\sigma \end{bmatrix} + \overline{b} = \begin{bmatrix} \overline{w_{1^{r-1}}} \\ \overline{w_{2^{r-1}}} \\ \vdots \\ \overline{w_{i^{r-1}}} \\ \vdots \\ \overline{w_{n^{r-1}}} \end{bmatrix}$$

Let $k = 1^r$. Equating across the k -th row, we have $\overline{w_k}^\sigma + b = \overline{w_{k^{r-1}}} = w_1 = 0$. Thus, we have $\overline{b} = \overline{w_k}^\sigma$. Then, we have $(C + \overline{w_k})^\sigma = C^{Tr}$ or

$$C = ((C + \overline{w_{1^r}})^\sigma)^{Tr^{-1}}.$$

Proof. For a more rigorous explanation, let $\overline{w_i} = [b_{i,1}b_{i,2} \cdots b_{i,d}]$. This leads to

Therefore, if \mathbf{c} occurs m times, $\mathbf{c} \times r$ must also occur at least m times, say m' times. But by the same argument, if $\mathbf{c} \times r$ appears in C m' times, then $(\mathbf{c} \times r) \times r^{-1} = \mathbf{c}$ appears at least m' times. Thus, $m' = m$. ■

The next theorem is the converse of the previous.

Theorem 9 Consider a cwatset C of order n and degree d , and consider a permutation $r \in S_n$. If, for any column \mathbf{c} that appears in C m times, $\mathbf{c} \times r$ also appears in C m times, then $r \in R_C$.

Proof. Denote by $\mathbf{c}_1, \mathbf{c}_2, \dots, \mathbf{c}_d$ the columns of C , and denote by $\overline{w_1}, \overline{w_2}, \dots, \overline{w_n}$ the words of C . By the definition of the \times operation, we have:

$$\left[\begin{array}{cccc} \mathbf{c}_1 & \mathbf{c}_2 & \dots & \mathbf{c}_d \end{array} \right]^{Tr^{-1}} + \overline{w_{1r}} = \left[\begin{array}{cccc} \mathbf{c}_1 \times r & \mathbf{c}_2 \times r & \dots & \mathbf{c}_d \times r \end{array} \right]$$

We know that for each \mathbf{c} that appears in C m times, $\mathbf{c} \times r$ also appears in C m times. So it is clear that there exists some $\sigma \in S_d$ that maps the set of m $\mathbf{c} \times r$ columns to the set of m \mathbf{c} columns for all $\mathbf{c} \in C$. (If C contains repeated columns, there will exist more than one such permutation.) So we have:

$$\begin{aligned} C^\sigma + w_{1r} &= \left[\begin{array}{cccc} \mathbf{c}_1 & \mathbf{c}_2 & \dots & \mathbf{c}_d \end{array} \right]^\sigma + w_{1r} \\ &= \left[\begin{array}{cccc} \mathbf{c}_1 \times r & \mathbf{c}_2 \times r & \dots & \mathbf{c}_d \times r \end{array} \right] + w_{1r} \\ &= \left[\begin{array}{cccc} \mathbf{c}_1 & \mathbf{c}_2 & \dots & \mathbf{c}_d \end{array} \right]^{Tr^{-1}} \end{aligned}$$

The last step follows from the definition of \times as explained above. By this, we know that $(\sigma, w_{1r}) \in \Omega_C$, and it induces $r^{-1} \in R_C$. Since R_C is a group, $r \in R_C$. ■

Proof. Let \bar{w} be word i of C or the i^{th} row of C . Since R is transitive, there exists $r \in R$ such that $i = 1^r$. Thus, $\bar{w} = \overline{w_{1^r}}$.

Then, we have

$$\begin{aligned} \mathbf{C} &= [\text{orb}_{R^X}(\mathbf{c}_1), \text{orb}_{R^X}(\mathbf{c}_2), \dots, \text{orb}_{R^X}(\mathbf{c}_n)] \\ &= [\text{orb}_{R^X}(\mathbf{c}_1) \times r, \text{orb}_{R^X}(\mathbf{c}_2) \times r, \dots, \text{orb}_{R^X}(\mathbf{c}_n) \times r]^\pi \\ &= ([\text{orb}_{R^X}(\mathbf{c}_1), \text{orb}_{R^X}(\mathbf{c}_2), \dots, \text{orb}_{R^X}(\mathbf{c}_n)] \times r)^\pi \end{aligned}$$

for some $\pi \in S_d$, where d is the number of columns in C . This is because the action of $\times r$ at most rearranges the elements of an orbit of R^X . Then, by the definition of \times , we have:

$$\mathbf{C} = \left([\text{orb}_{R^X}(\mathbf{c}_1), \text{orb}_{R^X}(\mathbf{c}_2), \dots, \text{orb}_{R^X}(\mathbf{c}_n)]^{T_{r^{-1}}} + \overline{w_{1^r}} \right)^\pi$$

or

$$\mathbf{C}^{\pi^{-1}} = \mathbf{C}^{T_{r^{-1}}} + \overline{w_{1^r}}$$

To get this back to the standard form of the definition, let $\sigma = \pi^{-1}$. Also note that if we convert C or $C^{T_{r^{-1}}}$ to a multi-set, we get C because the action of $T_{r^{-1}}$ only changes the order of the words in the set. That said, we have

$$C^\sigma = C + \bar{w}$$

for all $\bar{w} \in C$. ■

This theorem gives us a some insight in a result proven earlier.

Definition 15 [5] *A perfect cwat-multi-set of order n is a cwat-multi-set for which each column is of weight k or $n - k$ for some k .*

Perfect cwat-multi-sets are useful because we know that every cwatset is a concatenation of cwat-multi-sets [5]. However, it is possible to arbitrarily permute each cwat-multi-set before concatenating, and not every concatenation is possible. The following shows how cwat-multi-sets may be concatenated to form a cwatset.

Corollary 16 *For any transitive group $R \leq S_n$ and any column \mathbf{c} of length n , the columns in $\text{orb}_{R^X}(\mathbf{c})$ form a perfect cwat-multi-set.*

$$\begin{aligned}
(2^{(n-j)})^{r^{x-1}} &= 2^{(n-j)} \times r \\
&= \begin{bmatrix} 0 \\ \vdots \\ 0 \\ \vdots \\ 1 \\ \vdots \\ 0 \end{bmatrix}^{r^{-1}} + c_{1r} \\
&= \begin{bmatrix} 0 \\ \vdots \\ 1 \\ \vdots \\ 0 \\ \vdots \\ 0 \end{bmatrix} + 0 \\
&= 2^{(n-i)}
\end{aligned}$$

When we permute the column $2^{(n-i)}$ by r^{-1} , we know that the j -th bit, which must be a 1, is mapped to the i -th position. It is unimportant how r^{-1} permutes the other bits, for they are each 0. For a similar reason, we know that $c_{1r} = 0$, because we know r maps i to j , so it must map 1 to a different bit, and every other bit is 0.

Now consider the case where r maps i to j , but $i = 1$. We want to show that r^x maps $(2^{(n-1)} - 1)$ to $2^{(n-j)}$. As before, we have:

Corollary 18 *The mapping χ is an injective homomorphism.*

Proof. If $r^\chi = id_{R^\chi}$, then r^χ maps each of the columns in W to itself. From what we have just shown, it should be clear that this implies r maps i to itself for all i . Thus, $r = id_R$, and the kernel of χ is trivial. This result also allows us to construct the inverse of χ . If, for some $r^\chi \in R^\chi \leq S_{Col_n}$, r^χ maps column $2^{(n-i)}$ to column $2^{(n-j)}$, then $r^{\chi\chi^{-1}} = r$ maps i to j , etc. ■

3.2 Specific Subgroups

Theorem 19 *The pyramid cwatset of order n (the cwatset formed by the columns $\{2^0, 2^1, \dots, 2^{(n-2)}, (2^{(n-1)} - 1)\}$) has S_n as its representation group.*

Proof. Proof of this follows directly from Theorems 9, 14, and 17. ■

Thus, we know that we can always produce a cwatset of degree n (with n columns) such that it has with S_n as its representation group. However, in some cases, there exist such cwatsets with smaller degree. For instance,

$$C = \left\{ \begin{array}{ccc} 0 & 0 & 0 \\ 0 & 1 & 1 \\ 1 & 0 & 1 \\ 1 & 1 & 0 \end{array} \right\}$$

has S_4 as its R -group.

Theorem 20 *For any $n > 0$, let C be a formed by the columns $\{1, 2, \dots, 2^{n-1} - 1\}$. Then, C is a cwatset, and $R_C = S_n$.*

Proof. Proof of this result also follows from Theorems 9 and 14.

Theorem 21 *The cwatset C of order n containing all columns of weight 2 and weight $n - 2$ with the property that all ones in each of these columns are vertically adjacent has the dihedral group (D_n) as its representation group.*

Proof. The following is an example of such a construction:

$$C = \left\{ \begin{array}{cccccc} 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 1 \\ 1 & 1 & 0 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 & 0 & 0 \end{array} \right\}$$

$$\begin{aligned}
\mathbf{c} \times r s &= \left(\begin{array}{c} c_1 \\ \vdots \\ c_i \\ \vdots \\ c_j \\ \vdots \\ c_n \end{array} \right) \times s \times r \\
&= \begin{array}{c} c_1 \\ \vdots \\ c_j \\ \vdots \\ c_i \\ \vdots \\ c_n \end{array} \times r \\
&= \begin{array}{c} c_1 \\ \vdots \\ c_i \\ \vdots \\ c_j \\ \vdots \\ c_n \end{array} \times r \\
&= \mathbf{c} \times r
\end{aligned}$$

Since r is odd, $r(i, j) \in A_n$. Above, we assume that $i, j \neq 1$. In other words, (i, j) maps 1 to itself, so we add $c_1 = 0$ in performing the ' \times ' operation. But if we assume that either i or j equals 1, then $c_i = c_j = 0$, and the proof follows identically. ■

database, we need only compute the χ images of each transitive subgroup and then the column orbits induced by the χ -image. For a given set of orbits, we simply take the largest group whose χ -image induces those orbits; the largest will be a representation group, and all others will not.

This transitive subgroup database is included in the software package GAP. Appendix A contains and describes the GAP code used to perform this algorithm. One obstacle to overcome is the fact that the database contains only one representative of each conjugacy class. So, we can have the case where elements of class A are subgroups of class B , but the representative in the database from A is not a subgroup of the representative for B . To account for this, we examine the weights of the columns in the orbits under the χ -groups. It is clear that conjugation of R -groups preserves conjugation of R^x groups, and thus preserves orbit structure on the set of columns. But in fact conjugation also preserves the weights of the columns in their orbits. To see this, consider the following:

Definition 24 Let $R \in S_n$ and let $c \in \text{Col}_n$. Then, define the “weight structure” of $\text{orb}_{R^x}(c)$ to be a multi-set given by

$$ws(\text{orb}_{R^x}(c)) = \{\text{weight of } d \mid d \in \text{orb}_{R^x}(c)\}$$

Theorem 25 Let $R \leq S_n$ be a transitive group. Let $\{c_1, c_2, \dots, c_n\}$ be given such that

$$\text{Col}_n = \bigcup \{\text{orb}_{R^x}(c_1), \text{orb}_{R^x}(c_2), \dots, \text{orb}_{R^x}(c_n)\}$$

and $\text{orb}_{R^x}(c_i) \neq \text{orb}_{R^x}(c_j)$ for $i \neq j$.

Let $\alpha \in S_n$, and let $S = \alpha R \alpha^{-1}$. Then,

$$\text{Col}_n = \bigcup \{\text{orb}_{S^x}(c_1 \times \alpha), \text{orb}_{S^x}(c_2 \times \alpha), \dots, \text{orb}_{S^x}(c_n \times \alpha)\}$$

and $\text{orb}_{S^x}(c_i \times \alpha) \neq \text{orb}_{S^x}(c_j \times \alpha)$ for $i \neq j$.

Furthermore, $ws(\text{orb}_{R^x}(c_i)) = ws(\text{orb}_{S^x}(c_i \times \alpha))$ for all $i \leq k$.

Proof. The first part is a fairly straightforward application of the properties of conjugation. Let $d_i \in \text{orb}_{R^x}(c_i)$. Then, $d_i = c_i^{r^x}$ for some $r \in R$. By the

4.3 Search for Conjugate Subgroups

However, after all the preliminary checks, there still are cases where we need to know if H is conjugate to a subgroup of G . To actually perform this search, we advantage of the following fact.

Fact 26 *Let $H, G \leq S_n$, and suppose that there exists $\alpha \in S_n$ such that $\alpha H \alpha^{-1} \leq G$. Then, for all $g \in G$, $(\alpha g) H (\alpha g)^{-1} \leq G$.*

This is a very useful statement. Suppose that $h \in H$. Then, if $\alpha H \alpha^{-1} \leq G$, then $\alpha h \alpha^{-1} = g \in G$. We can search for α by examining every g to which h could be conjugated. But by this result, we can conjugate every element of G by every element of G , which will create a partition of G , and then simply examine one representative from each class of this partition.

To summarize, the following describes an efficient way for checking if H is conjugate to a subgroup of G .

1. Conjugate every element of G by every element of G . This uses the `ConjugacyClasses` command in GAP or Magma.
2. Find all the different cycle structures of elements of H . To do this, we used another call to `ConjugacyClasses`. At this point, we can stop if there is a cycle structure present in H that is not found in G .
3. For each cycle structure in H , compute the size of its centralizer. Multiply that by the number conjugacy class representatives in G that has the same cycle structure. This value represents the cost of checking that particular cycle structure. Find the cycle structure with the minimal cost.
4. Find an element $h \in H$ with the optimal cycle structure. Compute its centralizer C .
5. We know that we cannot conjugate h to an element with a different cycle structure. Thus, for every conjugacy class representative $g \in G$ that has the same cycle structure as h , find $x \in S_n$ so that $x^{-1} h x = g$.
6. At this point, the right coset Cx is the set of all permutations that will conjugate h to g . Check to see if any of its elements will conjugate H to a subgroup of G .

Order (n)	Transitive Subgroups of S_n	R -Groups	Cyclic Cwatset R -Groups
1	1	1	1
2	1	1	1
3	2	1	1
4	5	3	2
5	5	2	2
6	16	5	4
7	7	6	6
8	50	32	12
9	34	20	7
10	45	18	10
11	8	7	7
12	301	199	59
13	9	8	8
14	63	40	32
15	104	52	30
16	1954	1599	198
17	10	9	9
18	983	615	150
19	8	7	7
20	1117	644	206
21	164	109	76
22	59	41	31
23	7	6	6

5.1 Conjectures

The column entitled “Cyclic Cwatset R -Groups” gives the number of R -groups of cyclic cwatssets that appear for a given order.

Definition 27 *A cwatset C is cyclic if, for all $w \in C$, $w = b^{\sigma^k} + b^{\sigma^{k-1}} + \dots + b$ for some k . In this case, we say $C = \langle (\sigma, b) \rangle$.*

It was shown in [6] that a cwatset of order n is cyclic if and only if its R -group contains an element of order n , providing a simple test for cyclic cwatssets R -groups in our list of all R -groups.

The data suggest several interesting observations. Notice that every cwatset of prime order in our data is cyclic. This was shown to be true for any

$$\begin{aligned}
& \{15, 30, 60, 120\} \\
& \{23, 29, 46, 58, 71, 92, 113, 116\} \\
& \{27, 39, 54, 57, 78, 99, 108, 114\} \\
& \{43, 53, 77, 83, 86, 89, 101, 106\} \\
& \{45, 75, 90, 105\} \\
& \{51, 102\} \\
& \{85\}
\end{aligned}$$

Any group cwatset with \mathbb{Z}_8 as its representation group will contain some collection of these orbits. But, interestingly, almost none of these orbits forms a group in its own right: only $\{51, 102\}$ and $\{85\}$ do. (It has also been conjectured that every cwatset with all half-weight columns will form a group. From this it is clear that this is also untrue.) None of the other orbits can be a part of any group cwatset because they require a permutation of the columns to achieve closure. Note that by Theorem 14, conjugating \mathbb{Z}_8 by some α means we simply perform the ' $\times \alpha$ ' operation on each of these orbits, which clearly will not turn any nongroup orbit into a group orbit. We can test all combinations of the orbits $\{51, 102\}$ and $\{85\}$ to confirm that none of them produce a cwatset with \mathbb{Z}_8 as its representation group. ■

The isotropy subgroup of a cwatset C is defined $I_C = \{(\sigma, \mathbf{0}) \in \Omega_C\}$. It was shown in [6] that I_C is trivial if, and only if, C contains no repeated columns. It would be nice if every cwatset isomorphism class contained a representative with trivial isotropy. As early as order 4, though, we can find a counterexample.

Counter Example 32 *The cwatset*

$$C = \left\{ \begin{array}{ccc} 0 & 0 & 0 \\ 1 & 1 & 0 \\ 0 & 0 & 1 \\ 1 & 1 & 1 \end{array} \right\}$$

is isomorphic to no cwatset with trivial isotropy.

Proof. There are only 7 possible columns for an order 4 cwatset, so there are only 2^7 potential cwatsets with no repeated columns. A check of these affirms that none has R_C as its representation group. ■

Conjecture 33 *The Ω -group of any cwatset of order n whose representation group is C_n will have a nontrivial isotropy subgroup.*

References

- [1] Daniel K. Biss. On the Symmetry Groups of Hypergraphs of Perfect Cwatsets. *Ars Combinatoria* 56 (2000), pp. 271-288.
- [2] Greg Butler. The transitive groups of degree fourteen and fifteen. *Journal of Symbolic Computation* 16:413-422, Nov. 1993.
- [3] Greg Butler and John McKay. The transitive groups of degree up to 11. *Comm. Algebra*, 11:863-911, 1983.
- [4] Alexander Hulpke. *Lonstruktion transitiver permutationsgruppen*. Dissertation, Rheinisch Westfälische Technixche Hochschule, Aachen, Germany.
- [5] Julie Kerr. Hypergraph representation and orders of cwatsets. Technical Report MS TR 93-02, Rose-Hulman, 1993
- [6] Carolyn M. Girod, Matthew Lepinsky, Joseph R. Mileti, and Jennifer R. Paulhus. Cwatset isomorphism and its consequences. Technical Report MS TR 00-01, Rose-Hulman, 2000.
- [7] Gary J. Sherman and Martin Wattenberg. Introducing...Cwatsets! *Mathematics Magazine*, 67:109-117, April 1994.

```

}

sub unlock {
    flock(LOCK,LOCK_UN);
    #print "Released lock!\n"
}

#This is so named because this program was orgininally
#designed for Magma. We switched to GAP when we discovered
#a bug in ConjugacyClasses.
sub createMagmaFile{
    my ($num) = @_ ;

    open (TEMPLATE, "template") or die "Can't open template $!";
    open (FILE, ">in.$num") or die "Can't open output $!";

    while(<TEMPLATE>)
    {
s/\$num/$num/g;
print FILE ;
    }
    close TEMPLATE;
    close FILE;
}

$stopDir = "/afs/rose-hulman.edu/users/reu00/lindj2/stop/";

#Touch a .go file... Stop if this file ever gets deleted
open (GO, ">" . $ENV{"HOST"} . ".go");
print GO ($ENV{"HOST"}, "\n$$\n", scalar(localtime()),"\n");
close (GO);

for (my ($num) = 1; $num <= $targetNum ; $num++)
{
    # Lock to make sure that no one else is trying to create
    # an "in" file
    open(LOCK, ">lock-file") or die "Can't open lock file: $!";
    lock();
}

```

```

    $w = 'w';
    print $w;
    print STAT $w;
    close (STAT);
}

unlink($ENV{"HOST"} . ".go") or die "$!";

```

A.2 Determining the Weight Structure

The first task we had was to determine the weight structure of the orbits of the χ -group of every transitive group of order 22. It was tempting to try to store the χ -groups and its orbits for future reference, but the space requirements were too great.

For this case, we had the following template file in the gap22/chi/ subdirectory:

```

myDeg:=22;
Read("../..gapChi");
go($num);

```

We then started the search perl script by executing

```
nice -n 19 ../..searchgap 59
```

in the gap22/chi/ subdirectory. The nice is not necessary, but prevents the computer from taking up too many resources. Notice that this file referred to gapChi, which does most of the work. The listing of that files is as follows:

```

gapChiID="$Id: final.tex,v 1.1 2000/11/13 16:43:46 lindj Exp lindj $";
#Modified so that everything is based off of myDeg (instead of
#trying to generate a function to handle a particular degree...)

```

```
#Note: that myDeg must be set before Read'ing this file
```

```

num2list := function (n)
  return List([1..myDeg], i -> RemInt(QuoInt(n, 2^(myDeg-i)), 2));
end;

```

```
list2num := function(list)
```

A.3 Finding unique Weight Structures

The computation took us about a day running on about ten Sun Ultra 10 workstations. The output of the previous command was 59 out files. Each one assigned one element of an array called `weights`. To glue them together, we executed

```
cat chi/out.* > weights
```

in the `gap20/` subdirectory.

We then had a single machine sort through all the weight structures to find distinct examples. It also identified which groups were associated with a particular weight structure. To do this, we executed

```
gap -q < ../gapMakeMatch
```

in the `gap20/` subdirectory.

This referred to the `gapMakeMatch`, which is listed as follows:

```
#To use: gap -q < ../gapMakeMatch
weights=[];
Read("weights");
ws:=AsSet(weights);;
Size(weights);
Size(ws);
matches:=List(ws, i ->
  Filtered([1..Size(weights)], j -> weights[j] = i)
)
;;
AppendTo("matches.all", "matches:=", matches, ";\n");
Print(Size(matches));
Exec("mkdir matches");
for i in [1..Size(matches)] do
  PrintTo(Concatenation("matches/match.", String(i)),
    "myMatch:=", matches[i], ";\n");
od;
```

A.4 Sorting Through the Weight Structures

The last command created the subdirectory `gap22/matches/` and put 41 files in it name `match.1` through `match.41`. Each of those files represents a particular weight structure and stores a list of all the transitive groups which correspond to that structure.

```

    ans := ans * Factorial(deg - tot) ;
    return ans;
end;

```

```

multiplicity := function(list, obj)
    local i, ans;
    ans := 0;
    for i in list do
        if i = obj then
            ans := ans + 1;
        fi;
    od;
    return ans;
end;

```

```

isConjSub:=function(G1,G2)

```

```

#1) Find the element with the smallest centralizer in G1. Call it
#   src.
#2) Find all the elements in G2 that have the same cycle structure
#   as src.
#3) Reduce the list in (2) to one representative from each
#   conjugacy class under G2.
#4) Try all the choices

```

```

    local deg, Sn, class1,class2, struct1, struct2, costs, cost, minpos, cyc,
           src, reps, target, conj, cur, cent, i;

```

```

    deg := LargestMovedPoint(G2);
    Sn:=SymmetricGroup(deg);

```

```

    class1 := List(ConjugacyClasses(G1), i -> Representative(i));
    class2 := List(ConjugacyClasses(G2), i -> Representative(i));

```

```

    struct1:=AsSet(List(class1, i -> CycleStructurePerm(i)));
    struct2:=List(class2, i -> CycleStructurePerm(i));

```

```

findR:=function(deg,matches)
  local groups, eliminated, big, little, isConj, okRgroups;

  Print ("Working on: ", matches, "\n");
  if Size(matches) = 1 then
    Print ("That was quick :-)\n");
    return [TransitiveGroup(deg,matches[1])];
  fi;

  groups := List(matches, i -> TransitiveGroup(deg,i));

  # Get the lower orders first
  SortParallel(groups, matches,
    function(x, y) return Size(y) < Size(x); end);
  Print ("Sorted: ", matches, "\n");

  eliminated := [];
  # Make a subgroups pass
  for big in [1..(Size(groups)-1)] do

    for little in [(big+1)..Size(groups)] do

      if IsSubset(groups[big], groups[little]) then
        Print (little, " is a subset of ", big, "\n");
        AddSet(eliminated, little);
      fi;
    od;
  od;

  # Make the long and complicated pass
  for big in [1..(Size(groups)-1)] do
    # No continue :- (
    if not big in eliminated then

      for little in [(big+1)..Size(groups)] do
        if not little in eliminated then
          Print ("Comparing: ",big, " ", little, "\n");
        fi;
      od;
    fi;
  od;

```

```

AppendTo (outFile, "rGrp",deg,"[" ,num,"]:=[\n");
AppendTo (outFile, "Group(", GeneratorsOfGroup(ans[1]), ")");
for i in [2..Size(ans)] do
  AppendTo (outFile, ",\nGroup(", GeneratorsOfGroup(ans[i]), ")");
od;
AppendTo (outFile, "\n];\n");

return true;
end;

```

A.5 Putting it all together

If we had to use older methods, the process probably would have run for days without completing. However, method 9 is efficient enough that, running on about ten machines, we finished checking in under five minutes. The output of the command was 41 out files, which contained all the transitive groups which were not eliminated.

To glue them together, we ran

```
cat method9/out.* > final9
```

in the gap22/ subdirectory.

Finally, to produce the output files we ran

```
cat ../gapMakeR 22 | gap -q
```

in the gap22/ subdirectory. This executes a perl script that just output a list of GAP commands based on the order. The script is as follows:

```
#!/usr/local/bin/perl -w

($deg) = @ARGV;
die "Useage: $0 degree\n" unless $deg;

print <<END;
#To use: $0 $deg | gap -q
rGrp$deg:=[];
Read("final9");
s:=AsSet(Concatenation(rGrp$deg));;
Print("Number of rGroups: ",Size(s),"\n");
PrintTo("rGroup$deg.gap","rGrp$deg:=",s,";\n");
id:=Filtered([1..NrTransitiveGroups($deg)],i -> TransitiveGroup($deg,i) in s);

```