J. Symbolic Computation (2004) **11**, 1–000

Constructing Permutation Representations for Matrix Groups[†]

GENE COOPERMAN[‡]
LARRY FINKELSTEIN[‡]
MICHAEL TSELMAN[‡]
BRYANT YORK[§]
College of Computer Science
Northeastern University
Boston, MA 02115

(Received)

New techniques, both theoretical and practical, are presented for constructing permutation representations for computing with matrix groups defined over finite fields. The permutation representation is constructed on a conjugacy class of subgroups of prime order. We construct a base for the permutation representation, which in turn simplifies the computation of a strong generating set. In addition, we present an elementary test for checking the simplicity of the permutation image.

The theory has been successfully tested on a representation of the sporadic simple group Ly, discovered by Richard Lyons (1972). With no a priori assumptions, we find a permutation representation of degree $9,606,125$ on a conjugacy class of subgroups of order 3, find the order of the resulting permutation group, and verify simplicity. A Monte Carlo variation of the algorithm was used to achieve better space and time efficiency. The construction of the permutation representation required four CPU days on a SPARCserver 670MP with 64 Megabytes. The permutation representation was used implicitly in the sense that the group element was stored as a matrix, and its permutation action on a "point" was determined using a pre-computed data structure. Thus, additional computations required little additional space. The algorithm has also been implemented using the MasPar MP-1 SIMD parallel computer and 8 SPARC-2's running under MPI. The results of those parallel experiments are briefly reviewed.

[†] Preliminary versions of portions of this work originally appeared in (Cooperman, Finkelstein, York and Tselman, 1994) and (Cooperman, Finkelstein, and Tselman, 1995) and are copyrighted by ACM. They are included with permission from ACM.

[‡] Partially supported by NSF Grants CCR-920449 and CCR-9509783.
[§] Partially supported by ARPA Grant MDA-972-93-1-0023.

0747–7171/90/000000 + 00 $03.00/0

1. Introduction

In this paper, we present new techniques, both theoretical and practical, for computing permutation representations for matrix groups defined over finite fields. Initially, we are given a subgroup G of the general linear group $GL(n,q)$, with G specified only by a generating set S. We want to be able to analyze the structure of G or answer fundamental questions such as group membership, i.e. is a given element of $GL(n,q)$ contained in G? The primary motivation for constructing a permutation representation is that many problems seeming intractable for matrix groups have relatively efficient solutions for permutation groups. The most notable instance of this is group membership (see Babai and Szemerédi (1984), Cooperman and Finkelstein (1993)). In spite of this clear gain, practical efforts at representing matrix group problems in the permutation domain have relied on the action of the matrix groups on the set of $(q^n-1)/(q-1)$ points of projective space. This is clearly impractical for all but very small examples. To make matters worse, there are groups (for example $GL(n,q)$, except for a small set of values of n and q), where the smallest permutation representation does in fact have this degree.

We propose instead to construct a permutation representation from the conjugation action on a conjugacy class of subgroups of prime order. The motivation for this approach is twofold.

- For many important examples, a permutation representation with close to minimal degree can be constructed in this manner.
- Each subgroup in the conjugacy class has an efficiently computable signature.

The first statement is partially justified on theoretical grounds, by a result of Babai and Beals (1994), which asserts that for a simple group G there is always a conjugacy class of subgroups of prime order whose size is polynomial in the degree of the smallest permutation representation.

The technique fits well with an emerging theory of computing with matrix groups being developed by a number of researchers (Babai and Beals (1994), Holt et.al. (1994a), (1994b), Leedham-Green (1993), Luks (1992)). We expect this method to be especially useful for determining the structure of almost simple matrix groups in which the simple subgroup does not appear in its natural setting. The method also has important applications in modular representation theory (Gollan and Michler (1994)).

The theory has been successfully tested on a representation of the sporadic simple group Ly, discovered by Richard Lyons (1972). This example has independent interest. Direct application of our technique to this example yields a permutation representation of degree $9,606,125$ for the conjugation action on a conjugacy class of Z_3 subgroups. However, we are able to do more. Without any assumptions on the structure of the group, we are able to:

- find a permutation representation of degree $9,606,125$ on a conjugacy class of subgroups of order 3,
- find a base and strong generating set (which can be used to determine membership and group order),
- verify simplicity of the group.

Further, we retain an efficient invertible mapping between the matrix representation of a group element and the permutation representation. A Monte Carlo variation of the algorithm was used to achieve better space and time efficiency. The construction of the permutation representation required four CPU days on a SPARCserver 670MP with 64 Megabytes.

In (Sims, 1972), the existence and uniqueness of Ly was announced and a presentation for Ly was given, which was verified by performing a coset enumeration on a subgroup isomorphic to $G_2(5)$ with index $8,835,156$. Although one could adapt this enumeration to construct specific permutation generators of this degree, it has not yet been done. Gollan has taken the explicit permutation representation constructed here, with no assumptions about their origin, and is developing a deterministic program that will independently confirm Sims's original result (Gollan, 1995). Additionally, Soicher has used the permutation generators to compute collapsed adjacency matrices for the orbital digraphs arising from the rank 5 permutation action of Ly on its $3A$-generated cyclic subgroups (Praeger and Soicher, to appear).

Both space and time issues need to be addressed in order to develop an efficient implementation. In terms of space, the core idea is to employ both matrix and permutation representations alongside each other, in a hybrid manner. In the case of Ly, the permutation domain has size $9,606,125$ and so each permutation requires approximately 38MB of storage. In comparison, the storage for an individual matrix is approximately 12KB. In both cases, data compression can be used to realize additional savings. The data structure used to store the permutation domain requires 60MB of storage and allows efficient computation of the image of a point under an element of the group stored as a matrix. Thus we are able to use the permutation representation in an implicit manner without ever having to store the entire permutation image of any element of the group. This significantly reduces the space requirements for the computation.

In terms of time, since each point of the permutation domain is an element of a conjugacy class $\mathcal{O}$ of subgroups of prime order, the computation of the image of a *point* $\langle x\rangle \in \mathcal{O}$ under an element $g \in G$, requires two matrix multiplications to compute a generator $g^{-1}xg$ for $\langle x\rangle^g$. This would make it prohibitively expensive to both enumerate the points of the permutation domain as well as to do the numerous implicit permutation computations. For example, in the case of Ly, a naive implementation would require approximately 10^{13} arithmetic operations in $GF(5)$. Our solution is based on assigning to each matrix a signature, which is the image of a fixed vector chosen from the underlying vector space. This will save a factor of n, the matrix degree, by replacing matrix-matrix multiplication by vector-matrix multiplication. To see this, suppose there exists a vector v with the property that no non-identity element of G fixes v. Then each $g \in G$ can be uniquely identified by a signature defined to be the image of v under g.

We use randomization to gain a significant speedup in running time. This speedup is required for working with very large examples. Randomization is used in two ways. The first is to construct random elements of G in order to identify suitable elements of prime order, and hence a suitable conjugacy class. A more formal method for producing *nearly uniformly distributed* random elements has been described by Babai (1991), but its practicality has yet to be tested in an implementation.

The second use of randomization is the choice of a randomly chosen vector for computing the signature. If one has prior knowledge of the group structure, then one can give a lower bound on the probability that the signature will not mistakenly identify two distinct elements (see Lemma 3.1). When applied to Ly, this probability is at least $1 - 3.8 \times 10^{-23}$. In section 4 we discuss a verification test, which can be used to make the probability of returning an incorrect answer arbitrarily small.

In section 2, the specific construction of the permutation representation on the conjugacy class of subgroups is described. section 3 describes a number of heuristics used to accelerate the calculation. In section 5 we present a deterministic technique for constructing a base for the action of G on the conjugacy class $\mathcal{O}$ and in section 6 we give an elementary test of simplicity for G acting on $\mathcal{O}$. Both results exploit the fact that the points of the permutation domain have group theoretical significance, namely they are subgroups of the defining group. Parallel results are described in section 8.

Historically, Butler (1976) was the first to use permutation group methods to compute with matrix groups. The methods were further developed in (Butler and Cannon, 1982). The use of hashing and signatures in computational group theory goes back at least to STACKHANDLER (Cannon et al., 1972) and GROUP (Cannon and Richardson, 1973). Signatures were used by Conway and Norton in the construction of the finite simple group, J_4 (Conway et al., 1985). The novelty of the data structure presented here is the combination of these two elements with a spanning tree to form a Schreier vector data structure that is particularly efficient in space and time for such large problems.

2. Conjugacy Class Approach

Theoretical motivation for studying the permutation action on a conjugacy class of subgroups of prime order can be justified, in part by a result of Babai and Beals (1994). For a finite group, let $\nu(G)$ be the smallest integer such that each composition factor of G has a permutation representation of degree $\leq \nu(G)$. Their result asserts that when G is simple, there is always a conjugacy class of subgroups of prime order whose size is polynomial in $\nu(G)$. Furthermore, an element of this class, necessary for the construction of the permutation representation, can be located in a random way with high probability.

Lemma 2.1. (Babai and Beals, 1994) *Let G be a non-abelian simple group and $\nu = \nu(G)$. Then there exists a conjugacy class C of G, an absolute constant c and an integer α, $1 \leq \alpha \leq \nu^c$ such that*

$$|C|/|G| \geq 1/\nu^c, \ |(x^\alpha)^{Aut(G)}| < \nu^c, \quad x \in C.$$

Remark The proof of this result depends heavily on the classification of finite simple groups (Gorenstein, 1982) in conjunction with a theorem of Landazuri and Seitz (1974). In a private communication, W.M. Kantor has indicated to the authors that one can take $c = 2$ in the above result.

For example to apply this result literally, it would be necessary to generate truly random elements, starting initially only with a generating set for the group. Techniques due to Babai for getting nearly uniformly distributed elements in this setting (Babai, 1991). would not be practical even for a relatively small example such as Lyons' group. Even with a source of random elements, the result merely ensures that one has a reasonable chance of finding a power of a random element that lies in a "small conjugacy" class. However, it could take a prohibitive amount of time before one guesses that the class is in fact too large for practical consideration. Techniques also exist for algorithmically applying Lemma 2.1 in the case where G is not necessarily simple, although they will not be discussed here.

¿From our perspective, Lemma 2.1 can be used to develop heuristics for finding elements of a small conjugacy class of subgroups, since it does not provide an efficient deterministic algorithm. Furthermore, one may as well work with a conjugacy class $\mathcal{O}$ of subgroups of prime order, since otherwise, the permutation representation will definitely be imprimitive. Clearly, the smallest conjugacy class will consist of elements of prime order. It is not hard to heuristically find all the primes that divide the order of a group. For example, by taking sufficiently many pseudorandom elements of G, one is likely to find an element whose order is a multiple of a given prime. Typically, a pseudorandom element is obtained by multiplying out the elements of a sufficiently long word in the generating set.

As one constructs the elements of a conjugacy class, one can estimate the size of the conjugacy class before the construction is completed. Assume, for the purpose of developing a heuristic, that the candidates for new elements of a conjugacy class (i.e. conjugates of a previous subgroup by a random element) are random with a uniform distribution. This is similar to the uniform hashing distribution that is often assumed in estimating the efficiency for hash functions. As one adds new elements to the conjugacy class $\mathcal{O}$, one expects the first few collisions with previously discovered elements to occur after $\sqrt{|\mathcal{O}|}$ elements have been added. This yields a rough estimate for $|\mathcal{O}|$. So, one can guess which conjugacy class is smallest without having to find more than approximately the square root of the number of its elements.

If one has specific knowledge about the conjugacy classes for which one has generators, one can find a desired conjugacy class much faster. For example, starting with generators for Ly, one can find an element of order 33 with probability 1/33 since such elements are self-centralizing. In general, the probability of finding an element of a specific conjugacy class is equal to $1/|\mathcal{C}_G(x)|$ where x is a representative element of the class. Furthermore, the 11^{th} power of such an element belongs to the conjugacy class $3A$, which is the smallest conjugacy class in Ly and has size $19,212,250$. This in turn leads to a conjugacy class of subgroups of order 3 of size $9,606,125$.

3. Construction of the Permutation Action

The algorithm is based on a data structure that serves both as a hash array and as a search tree for the conjugacy class of subgroups. We review the definition of hashing with open addressing to fix notation. Our definition covers only our own particular implementation of hashing, and makes no attempt at full generality. Let $\mathcal{O}$ be a set of objects. Let A be a *hash array* of length larger than $|\mathcal{O}|$. In practice, we use an array with $|A| = 1.6|\mathcal{O}|$. The *hash function* $h_{\alpha,\beta}$ is defined in terms of α and β. The function $\alpha\colon \mathcal{O} \to [1, |A|]$ is a *primary hashing function on* $\mathcal{O}$ if for a random element $x \in \mathcal{O}$, $\alpha(x)$ is "nearly uniformly random". A *secondary hashing function*, $\beta\colon [1, |A|] \to [1, |A|]$ is an invertible function (i.e. a permutation on $[1, |A|]$) with long cycles under iteration by β.

We now restrict $\mathcal{O}$ to be the desired conjugacy class of subgroups of G. In general, the size of $\mathcal{O}$ is not known in advance, but in many cases we will be working with a group whose structure we are attempting to verify and so we may assume that we know $|\mathcal{O}|$. This is the case, for example, in working with matrix generators for Ly.

In implementations, it is most convenient to hash on an element of the subgroup, instead of on the subgroup as a whole. Each subgroup of $\mathcal{O}$ will be represented by exactly one generating element. Since the conjugacy class $\mathcal{O}$ consists of subgroups of order p, at most $p-1$ possible generators are possible for each subgroup, and this must be considered in building the hash table.

Let x_{init} be a fixed element of G with $\langle x_{init}\rangle \in \mathcal{O}$ and let `root_index` $= \alpha(x_{init})$.

Let S generate G. The entries in the hash array A are either `NULL` or of the form $(i, g) \in [1, |A|] \times S$. For $x \in \mathcal{O}$, the *hash function* $h_{\alpha,\beta}: \mathcal{O} \to [1, |A|]$ probes the sequence $(f(x), \beta(f(x)), \beta^2(f(x)), \ldots)$ until it reaches the first index $i \in [1, |A|]$ for which either $A[i] =$ `NULL` or $A[i] = (i', g')$ *matches* x. We say that $A[i] = (i', g')$ *matches* x, with $\langle x \rangle \in \mathcal{O}$, if either $i' =$ `root_index` and $x = x_{init}$ or if $A[i']$ matches y and $y^{g'} = x$. Finally, $h_{\alpha,\beta}$ returns i and sets a condition variable to "`NULL`" or "`MATCH`".

Thus, the hash array A effectively encodes a search tree having a branching factor of at most $|S|$. If $A[i] = (i', g')$, then one can view the search tree as containing an edge from node i to node i' labeled by g'. The node i is labeled by x for which $A[i]$ matches x, and the node i' is similarly labeled. If $A[i] = (i, g)$, it is clear from the recursive definition of "match" that one can find a word w in S such that $h_{\alpha,\beta}(x_{init}^w) = i$. This provides a partial inverse for $h_{\alpha,\beta}$. With these tools, standard techniques of breadth-first search are then employed until no new elements of $\mathcal{O}$ are found. Two bit vectors, each with $|A|$ elements, are used to encode the elements of the last and next frontier set.

Thus, the space required by the algorithm is dominated by the space required for A. In the case of Lyons' group, we chose $|A| = 16,000,000$ and each entry (i, g) fits in one 32-bit word, thus requiring about 64 megabytes for the full algorithm.

The time to compute $h_{\alpha,\beta}(x_{init}^w)$ for w a word in S, dominates the overall time. Usually, $\alpha(x_{init}^w)$ returns an index satisfying "`NULL`" or "`MATCH`", and $\beta(\alpha(x_{init}^w))$ does not need to be called. So, the time is dominated by the time to first compute x_{init}^w and then to test if $\alpha(x_{init}^w)$ matches $(x_{init}^w)^j$ for some power j. Since the conjugacy class $\mathcal{O}$ consists of subgroups of order p, at most $p - 1$ possible matches need to be computed in order to test if a conjugate of x_{init} generates a subgroup in $\mathcal{O}$ which has already been identified. The time to test a match involves multiplying out a word in S whose length is bounded by $2d + 1$ where d is the depth of the search tree. This is because the word in S acts by conjugation on $x_{init} \in \mathcal{O}$ and each conjugation requires two multiplications. So, $p - 1$ words of length $2d + 1$ typically need to be multiplied out in computing $h_{\alpha,\beta}(x_{init}^w)$. In our experiment with Lyons' group, we employ several heuristics described below, so that most nodes are found at depth $d = 4$, and all nodes are found by depth $d = 5$.

3.1. Signatures of Group Elements

As described above, the algorithm requires approximately $|\mathcal{O}| 2d(p-1)$ matrix multiplications plus additional time for hash collisions. For example, in the case of Lyons' group, $|\mathcal{O}| = 9,606,125$, $d = 4$ and $p = 3$ and so the algorithm would require approximately 154,000,000 matrix multiplications in $GL(111, 5)$. This would be unacceptably slow. The key to making the algorithm fast is to avoid the time for matrix multiplication.

Let $V = V(n, q)$ be the underlying vector space, and suppose we can find $v \in V$ which satisfies the condition

$$v^g = v,\ g \in G \iff g = 1. \tag{$*$}$$

Then testing equality of two elements of G reduces to testing if their images of v are the same.

Assume v_{init} satisfies $(*)$. Given the generator $x' = x_{init}^w$ for a subgroup in $\mathcal{O}$, where w is a word in S, one can then use the image $v_{init}^{w^{-1} x_{init} w}$ of v_{init} under x' as input to $h_{\alpha,\beta}$ instead of the matrix x_{init}^w. This results in a cost of $2|w| + 1$ vector-matrix multiplications instead of $2|w|+1$ matrix multiplications, saving a factor of roughly n in time, for $n = 111$.

The next result gives some indication as to how likely it is that we can find a vector which satisfies $(*)$.

LEMMA 3.1. *Let m be the maximum value which occurs as the dimension of the fixed point subspace of any non-identity element of $G \leq GL(n,q)$. Then the probability that a randomly chosen vector satisfies $(*)$ is at least $1 - |G|/q^{n-m}$.*

PROOF. At most $(|G|-1)(q^m-1)$ non-zero vectors of $V(n,q)$ will be fixed by a non-identity element of G. Thus, the probability that a randomly chosen vector is not fixed by any non-identity element of G is at least $1 - q^m|G|/q^n$. □

Remark One can achieve finer estimates by also considering the size of the conjugacy class corresponding to m and the second largest dimension, m_2, of a fixed point subspace. With the use of a character table, one can do still better. It is also clear that the use of k independent initial random vectors increases the probability to at least $1-(|G|/q^{n-m})^k$.

Applying Lemma 3.1 to Lyons' group, a randomly chosen vector $v \in V$ can be shown to satisfy $(*)$ with probability at least $1 - 3.8 \times 10^{-23}$. To determine this, we computed the dimension of the fixed point subspace for each conjugacy class of elements of prime order, since this dimension is maximized for such conjugacy classes. For Lyons' group, $m = 55$ (corresponding to the class of involutions) and $|G| = 5.2 \times 10^{16}$.

3.2. SOFTWARE OPTIMIZATIONS

The first version of the program was tested on Lyons' group. At that time, it was estimated that it would require more than a month of CPU time to complete. Hence, a series of software optimizations were successively applied to reduce the computation time to the four CPU days currently observed. We carry along and further refine the lower bound of $|\mathcal{O}|2d(p-1)$ vector-matrix multiplications required from section 3.1 in order to illustrate how each successive optimization is expected to lower the CPU time. The actual number of vector-matrix multiplications required was usually larger by some proportional factor.

3.2.1. FASTER VECTOR-MATRIX MULTIPLICATION

A lookup table of all linear combinations of each set of four adjacent rows under $GF(5)$ is kept for each matrix. This achieves a computation time for a vector-matrix multiplication in $GL(111,5)$ of 900 microseconds on a SPARCserver 670MP. Such a lookup table was first used by Arlazarov et. al. (1970) (see also Aho et al., 1974), and was popularized by Parker (1984) in his software for the meataxe algorithm.

3.2.2. SHALLOW SEARCH TREES

We add 50 redundant generators, chosen at random, to the original generating set of size 2 and then add in all inverses. This decreases the depth, d, of the search tree resulting in shorter words. If the number of generators $|S|$ is 2, then $d \geq \log_2|\mathcal{O}|/(\log_2|S|) > 30$. If $|S| = 104$, then $d \geq \log_2|\mathcal{O}|/(\log_2|S|) > 3.5$. Experimentally, the value of d is observed to be close to the lower bound. Towards the end of the computation, when most of the elements of $\mathcal{O}$ have been discovered, we revert to the original, smaller generating set for finding the last ones.

3.2.3. Fast Completion of Subgroup Orbits

Suppose we discover a subgroup Z of G with the property that Z acts semi-regularly on $\mathcal{O}$. This means that each orbit of Z on $\mathcal{O}$ has size $|Z|$. Then the existence of Z can substantially speed up the discovery of elements of $\mathcal{O}$ through the following observation. Each time we discover a new element $\langle y \rangle$ of $\mathcal{O}$, we generate the complete set of elements in the orbit of $\langle y \rangle^Z$. We can be assured that each element $\langle y' \rangle$ in this orbit is not yet in the hash table. Thus, for each such $\langle y' \rangle$, we need only probe for the next `NULL` slot. If $A[\alpha(\beta^j(y'))]$ is not `NULL` for some j, then we can skip the test for a match, since $A[\alpha(\beta^j(y'))]$ cannot match y'. If $\ell = |Z|$, then this reduces the approximate lower bound on the time to $\Theta(|\mathcal{O}|2d(p-1)/\ell) + |\mathcal{O}|2d(\ell-1)/\ell))$ vector-matrix multiplications. The term $\Theta(2d(p-1))$ is approximately the time to discover and hash the first point of a new orbit. The term $\Theta(2d(\ell-1))$ is approximately the time to hash the remaining $\ell - 1$ points of a new orbit. Furthermore, since all orbits are of length ℓ, each expansion of an orbit will lead to ℓ new elements. Thus $d \geq \log_2(|\mathcal{O}|)/(\log_2(|S|\ell))$. In the case of Lyons' group, with $|S| = 104$ and $\ell = 67$, this leads to the lower bound $d > 1.8$. If $|Z|$ is sufficiently small, then, we can precompute the elements in advance.

In application of this technique to Lyons' group, experimentally, we observe that 6,968 (i.e. $|S|\ell$) new elements are found after applying generating set S to the elements of the orbit of the initial element, and 9,536,646 new elements are found during the next stage of expanding orbits and applying generating set S, thus discovering most of the elements by the depth $d = 4$.

The difficulty in applying this technique when Z does not act semiregularly on $\mathcal{O}$ is that one no longer has a guarantee that repeats won't occur while entering the elements of $\langle y \rangle^Z$ in the table. There are several ways around this obstruction. If Z has prime order p, and $\langle y \rangle$ is a new point, then $\langle y \rangle^Z$ has order 1 or p. Assuming that Z does not fix a large subset of $\mathcal{O}$, we can first check that $\langle y \rangle$ is not fixed by Z and then proceed as above.

More generally, we can use a suggestion of Gene Luks. Given $\langle y \rangle^Z$, one can pre-compute a subset $Z' \subset Z$, such that for each $g \in Z$ there is a unique $h \in Z'$ with $\langle y \rangle^g = \langle y \rangle^h$. For example, one could use a second hash table proportional in size to $|Z|$ to record if an element of $\langle y \rangle^Z$ was previously seen. Only elements of $\langle y \rangle^Z$ not previously seen are entered into the main table.

The main difficulty in applying this technique is finding subgroups that are known to be proper, aside from cyclic subgroups. One obvious strategy is to look at subgroups generated by a random pair of involutions. For example, suppose an element of prime order p is self-centralizing, and is inverted by an involution. Then it will usually be easy to find an element of order p as a product of two involutions. In Lyons' group, the probability that a random pair of involutions generates an element of order 67 is approximately 0.09. Thus these two involutions will generate a dihedral group of order $2 \cdot 67$.

In computing with matrix generators for a group G that we believe is isomorphic to Ly, we use known properties of Ly to help guide the computation, but make no explicit assumptions on G. For example, in Ly, there are 3 conjugacy classes of elements of order 67 and each such element is self centralizing (Conway et.al., 1985). This implies that the probability of a randomly chosen element of Ly lying in a specific conjugacy class of elements of order 67 is 1/67. Hence, the probability that a randomly chosen element of Ly has order 67 is 3/67. In particular, this leads us to believe that for G, we will find an element z of order 67 rather easily by using the heuristic method described in section 2. To show that z acts fixed-point freely on $\mathcal{O}$, we first compute $\dim(\mathcal{C}_V(x)) = 21$ for $\langle x \rangle \in \mathcal{O}$ and $\dim(\mathcal{C}_V(z)) = 3$, where $\mathcal{C}_V(x)$ is the subspace of V fixed by x. The following result then shows that z acts fixed point freely on $\mathcal{O}$.

Lemma 3.2. *Let $z \in G \leq GL(n,q)$ be an element of prime order r acting on a conjugacy class $\mathcal{O}$ of G of subgroups of prime order p, with $p \neq r$. Let $\langle x \rangle \in \mathcal{O}$. Set $d_1 = \dim(\mathcal{C}_V(x))$ and $d_2 = \dim(\mathcal{C}_V(z))$. If r does not divide $|GL(d_2,q)|$ and $d_1 < d_2$, then $\langle z \rangle$ acts fixed-point freely on $\mathcal{O}$.*

Proof. It suffices to show that z does not fix any points of $\mathcal{O}$. If z in fact has a fixed point on $\mathcal{O}$, then z normalizes $\langle y \rangle \in \mathcal{O}$ and hence must leave invariant the d_2-dimensional subspace $\mathcal{C}_V(y)$. But z cannot act non-trivially on $\mathcal{C}_V(y)$ since r does not divide $|GL(d_2,q)|$. Thus, $d_2 \geq \dim(\mathcal{C}_V(z)) = d_1$ contradicting the hypotheses. □

Remark Let $g \in GL(n,q)$ have order r, relatively prime to q. Let $m = \dim(\mathcal{C}_V(g))$. Then m can be quickly computed by a randomized algorithm based on the following observation. If v is chosen at random from V according to the uniform distribution, then $\sum_{i=1}^{r} v^g$ satisfies the uniform distribution in the fixed point subspace of g. (The last statement is not true in general unless r is relatively prime to q.) To construct a basis for the fixed point space of g, initialize $B \leftarrow \emptyset$. Execute a loop in which the basic step is to choose a random $v \in V$ and test if $u = \sum_{i=1}^{r} v^g \in \langle B \rangle$. If the test fails, add u to B. After t consecutive successes, the probability is at least $1 - 1/q^t$ that B is a basis for the fixed point subspace.

3.2.4. Elimination of Spurious Matches through Check Bits

We maintain an array of check bits for each entry in our hash table. This is used to efficiently recognize hash collisions. When $h_{\alpha,\beta}(x) = i$, we store four check bits in $B[i]$ derived from the computer encoding of x. Then whenever we must check if $A[i]$ matches y, with $\langle y \rangle \in \mathcal{O}$, we first check if $B[i]$ equals the four check bits for y. If not, we can quickly eliminate the possibility of a match. For each x, we normally have to check up to $p-1$ words corresponding to $\langle x_{init} \rangle^x$. This is reduced to checking only $(p-1)/16$ words on average. Frequently, one will find that none of the check bits match. In this case, we can conclude that there was a hash collision, and immediately do a second hash. It is especially important in the latter phase, when few new orbits are discovered.

3.2.5. Pre-computation of Common Subwords

Some vector-matrix multiplications can be saved through pre-computation (matrix multiplication) of subwords. This is especially valuable in conjunction with the optimization on subgroup orbits in section 3.2.3. The subword corresponding to the initial element of the orbit can be pre-computed at the first encounter, saving computations during the rest of that orbit.

3.2.6. Caching Common Prefixes

The image of v_{init} under prefixes of words in S can be cached, instead of repeatedly computed.

4. Reliability of Using Signatures

In general, it is hard to give an a priori estimate that a randomly chosen vector satisfies property $(*)$ of section 3.1. Under certain conditions, it is possible to give a lower bound on the probability that a randomly chosen vector satisfies $(*)$. In this case, we also have a lower bound on the reliability that our enumeration of $\mathcal{O}$ will be correct and we can proceed as in section 3. See for example, Lemma 3.1. Otherwise, there is no guarantee that a v satisfying $(*)$ can even be found. Errors can occur as follows:

$$v^y = v^z, \text{ but } \langle y \rangle \neq \langle z \rangle, \langle y \rangle, \langle z \rangle \in \mathcal{O}$$

Note that $v \in \mathcal{C}_V(yz^{-1})$.

The following randomized method, which is simple to implement, will detect an error in our enumeration with provable reliability. Choose two random vectors, v and u. The signature is computed using v and errors are detected with u as follows: If for some $j,\ 1 \leq j \leq p-1$

$$v^y = (v^z)^j, \text{ but } u^y \neq (u^z)^k, 1 \leq k \leq p-1, \langle v \rangle, \langle z \rangle \in \mathcal{O}$$

then v is a bad choice and the computation stops.

Lemma 4.1. *Let α be the probability that the construction returns an incorrect answer that is not detected. Then $\alpha \leq 1/q$.*

Proof. To see this, assume the procedure completes and

$$\mathcal{O} = \mathcal{D}_1 \cup\ \mathcal{D}_2$$

where $\mathcal{D}_1$ is the subset of $\mathcal{O}$ actually determined. Note that $\mathcal{D}_1$ and $\mathcal{D}_2$ are determined solely by the choice of v and are not influenced by the choice of u. If $\langle z \rangle \in \mathcal{D}_2$, then u is in the fixed point subspace

$$yz^{-k}, \ \langle y \rangle \in \mathcal{D}_1, \quad 1 \leq k \leq p-1.$$

An error will be reported unless u is in the intersection of all subspaces formed in this way. This will happen with probability at most $1/q$. □

Let $m = \max_{1 \neq g \in G}(\dim(\mathcal{C}_V(G)))$. Then it is easy to modify the argument in the previous result to get an improved bound of

$$\alpha \leq 1/q^{n-m}.$$

In the case where no additional information is known, the test can be modified to improve the reliability by choosing ℓ random vectors $u_1, u_2, \ldots, u_\ell$ to check the computation. Thus, if g_1 and g_2 have the same signature, then we conclude $g_1 = g_2$ if

$$u_i^{g_1} = u_i^{g_2}, \ 1 \leq i \leq \ell.$$

As before, it follows easily that $\alpha \leq 1/q^\ell$.

5. Verification of a Base for a Conjugation Action

Having computed a permutation representation one would like to construct a base and strong generating set. This section describes how to use the special data structure previously constructed in order to efficiently find a permutation base for the conjugation action. This method depends on computations done within the matrix domain, and in practice is sublinear in the the size of the permutation domain. This is made precise in the body of this section.

Once the base has been constructed, there are numerous strategies available for computing a strong generating set for $G^{\mathcal{O}}$. One strategy is to use a heuristic procedure to obtain a candidate strong generating set $\mathcal{S}$ and then use either the Monte Carlo strong generating test described in (Cooperman and Finkelstein, 1993)) or the deterministic Brownie-Sims *Verify* routine, originally implemented in CAYLEY (Cannon and Richardson, 1973) and now in MAGMA. The knowledge of a small base in advance speeds up both procedures.

The first result gives a description of the kernel of $G^{\mathcal{O}}$.

Lemma 5.1. *If K is the kernel of $G^{\mathcal{O}}$, then either $K = \mathcal{C}_G(C)$, or $\mathcal{C}_G(C)$ is properly contained in K and $\langle C\rangle$ is an elementary abelian subgroup.*

Proof. By definition, $K = \{g \in G\colon \langle x\rangle^g = \langle x\rangle, \forall\langle x\rangle \in \mathcal{O}\}$. Clearly $\mathcal{C}_G(C) \subseteq K$. Therefore, if K commutes with C, then $K = \mathcal{C}_G(C)$. We may assume for the remainder of the proof, that $\mathcal{C}_G(C) \neq K$. By construction, K normalizes $\langle x\rangle$ for each $\langle x\rangle \in \mathcal{O}$. Further, x normalizes K, since $K \triangleleft G$. So,

$$[K, \langle x\rangle] \subseteq \langle x\rangle \cap K = 1 \text{ or } \langle x\rangle \;\; \forall\langle x\rangle \in \mathcal{O},$$

since x has prime order. But $K \neq \mathcal{C}_G(C)$ implies that $[K, \langle x\rangle] \neq 1$ for at least some $\langle x\rangle \in \mathcal{O}$. In this case, $[K, \langle x\rangle] = \langle x\rangle$. Therefore $x \in K$, which implies that $C \subseteq K$, since C is a conjugacy class of G and $K \triangleleft G$. It now follows that $\langle C\rangle$ is elementary abelian and $K/\mathcal{C}_G(C)$ acts as a non-trivial group of scalar matrices on $\langle C\rangle$. $\square$

The elements of the conjugacy class C generate an elementary abelian group precisely when C is in the kernel of $G^{\mathcal{O}}$. Testing if C is in the kernel is easy since this condition is true if a single element of C acts trivially on $\mathcal{O}$. In the case where $\langle C\rangle$ is elementary abelian, it is more appropriate to consider the linear action of G on $\langle C\rangle$ rather than the permutation action and therefore we will assume the following for the remainder of this section.

Assumption 5.2. *$\langle C\rangle$ is non-abelian.*

As a consequence of this assumption, we then have from Lemma 5.1 that $G/\mathcal{C}_G(C)$ acts faithfully on $\mathcal{O}$.

We now consider the task of finding a base for G acting on $\mathcal{O}$. The construction of a base requires two steps. The first is to find a set $B \subseteq C$ with the property that $\mathcal{C}_G(B) = \mathcal{C}_G(C)$. Note that B can be characterized as a base for the conjugation action of G on C. If $\mathcal{O}_B = \{\langle b\rangle : b \in B\}$, then $\mathcal{O}_B$ is "close" to being a base for $G^{\mathcal{O}}$. The second step is to augment $\mathcal{O}_B$ to a base and this is described in Lemma 5.9. We present a deterministic algorithm that will either construct the set B or else construct a non-trivial block of imprimitivity for G acting on $\mathcal{O}$. This latter case leads to a reduction in the degree of the permutation representation and is of independent interest.

The proper setting for constructing B is either the natural embedding of G in the full matrix ring $\mathrm{Mat}(n,q)$ or else the homomorphic image of G as a subgroup of $\mathrm{Sym}(\mathcal{O})$. Because the arguments are similar, we will adopt notation that emphasizes the general nature of the proof. We define two algebraic objects H and $\bar{G} = G/K$. Either $H = \mathrm{Mat}(n,q)$ and K=1, so that $G = \bar{G}$, or else $H = \mathrm{Sym}(\mathcal{O})$ and $K = \mathcal{C}_G(C)$. Note that G acts by conjugation as a group of automorphisms of $\mathrm{Mat}(n,q)$ and thus familiar group theory definitions such as $\mathcal{C}_H(B)$, $B \subseteq C$ and $\mathcal{N}_G(M)$ for M a G-invariant subalgebra of $\mathrm{Mat}(n,q)$ make sense. Also, we will assume in the case $H = \mathrm{Mat}(n,q)$, that whenever we conjugate by an element $h \in H$, that h is a unit in H, i.e. that $h \in GL(n,q)$.

Lemma 5.3. *If $\bar{B}^h$ commutes with $\mathcal{C}_H(\bar{B})$ for $\bar{B} \subseteq \bar{G}, h \in H$, then h normalizes $\mathcal{C}_H(\bar{B})$.*

Proof. Note that $\bar{B}^h$ commutes with $\mathcal{C}_H(\bar{B})$ implies that $\mathcal{C}_H(\bar{B}) \subseteq \mathcal{C}_H(\bar{B}^h) = \mathcal{C}_H(\bar{B})^h$. But $|\mathcal{C}_H(\bar{B})| = |\mathcal{C}_H(\bar{B})^h|$ and so $\mathcal{C}_H(\bar{B}) = \mathcal{C}_H(\bar{B})^h$. $\square$

Theorem 5.4. *Assume a generating set S for G and an element $b \in C$ are known. One can find a set, $B \subseteq C$, of at most $\log_2 |G|$ elements satisfying $\mathcal{C}_{\bar{G}}(\bar{B}) = \mathcal{C}_{\bar{G}}(\bar{C})$. Let t be an upper bound on the time to compute $\mathcal{C}_H(\bar{B}')$ for $\bar{B}' \subseteq B$, and let k be an upper bound on the time to compute if an element of H commutes with $\mathcal{C}_H(\bar{B}')$. Then the time to compute B is $O(|B||S|k + |B|t)$.*

Proof. The set B is found through a transitive construction. The element $b \in C$ is added to B and $\mathcal{C}_H(\bar{B})$ is computed. For each $g \in S$, one tests if $\bar{b}^{\bar{g}}$ does not commute with $\mathcal{C}_H(\bar{B})$. If it does not commute, $\bar{b}^{\bar{g}}$ is added to B, and $\mathcal{C}_H(\bar{B})$ is recomputed. One repeats the construction for each new element that has been added to C, until all pairs $(b,g) \in B \times S$ have been tested. At this stage, $\mathcal{C}_{\bar{G}}(\bar{B}) = \mathcal{C}_{\bar{G}}(\bar{C})$. Since $\langle B \rangle \subseteq G$ and the elements of B form a non-redundant generating set for $\langle B \rangle$, $|B| \leq \log_2 |G|$.

The proof divides into two cases. In the first case, assume that $\mathcal{N}_{\bar{G}}(\mathcal{C}_H(\bar{B}))$ is proper in $\bar{G}$. Then there exists a $g \in S$, for which $\bar{g} \notin \mathcal{N}_{\bar{G}}(\mathcal{C}_H(\bar{B}))$. By Lemma 5.3, $\bar{B}^{\bar{g}}$ and $\mathcal{C}_H(\bar{B})$ do not commute. So, there is a $b \in B$ such that $\bar{b}^{\bar{g}}$ does not commute with $\mathcal{C}_H(\bar{B})$, and our algorithm finds a new pair (b,g) and augments B.

In the second case, $\mathcal{N}_{\bar{G}}(\mathcal{C}_H(\bar{B})) = \bar{G}$. This implies that for all $\bar{g} \in \bar{G}$, $\mathcal{C}_H(\bar{B}^{\bar{g}}) = \mathcal{C}_H(\bar{B})^{\bar{g}} = \mathcal{C}_H(\bar{B})$. Hence $\bar{B}^{\bar{g}}$ commutes with $\mathcal{C}_H(\bar{B})$. But $\bar{C} = \bar{b}^{\bar{G}}$, and so $\bar{C}$ commutes with $\mathcal{C}_H(\bar{B})$. In particular, $\mathcal{C}_{\bar{G}}(\bar{B}) \subseteq \mathcal{C}_{\bar{G}}(\bar{C})$. Since the reverse inclusion is clearly true, we may infer that $\mathcal{C}_{\bar{G}}(\bar{B}) = \mathcal{C}_{\bar{G}}(\bar{C})$ as claimed.

Given the defined parameters k and t, the timing follows directly from the algorithm. $\square$

When $H = \mathrm{Sym}(\mathcal{O})$, so that $\bar{G} = G/\mathcal{C}_G(C)$, it may be the case that $\mathcal{C}_{\bar{G}}(\bar{C})$ is non-trivial. The next results shows that when this happens, $G^{\mathcal{O}}$ is imprimitive and this leads to a permutation representation of smaller degree.

Corollary 5.5. *If $H = \mathrm{Sym}(\mathcal{O})$ and $\mathcal{C}_H(\bar{C})$ is non-trivial, then the set of orbits of $\mathcal{C}_H(\bar{C})$ forms a proper block system for $\bar{G}$.*

Proof. Since $\bar{G}$ acts transitively on $\mathcal{O}$ and $\mathcal{C}_H(\bar{C})$ is normalized by $\bar{G}$, the set of orbits of $\mathcal{C}_H(\bar{C})$ forms a block system for $\bar{G}$. In order to show that the block system is proper, it suffices to show that $\mathcal{C}_H(\bar{C})$ acts intransitively on $\mathcal{O}$. By Assumption 5.2, $\langle C\rangle$ is non-abelian and so $\langle\bar{C}\rangle$ is a non-trivial subgroup of $\bar{G}$ by Lemma 5.1. Since C is a conjugacy class of G, $C\cap\mathcal{C}_G(C)=\emptyset$. This implies that each $x\in C$ acts non-trivially on $\mathcal{O}$ but does have fixed points, namely $\langle x\rangle$. In particular, each element of $\bar{C}$ has fixed points on $\mathcal{O}$. It then follows that $\mathcal{C}_H(\bar{C})$ acts intransitively as required. □

In the case where $H=\mathrm{Sym}(\mathcal{O})$ and $\mathcal{C}_H(\bar{C})$ is non-trivial, then the previous result shows how to construct a non-trivial system of imprimitivity for $\bar{G}$ acting on $\mathcal{O}$. In this case, we have a smaller degree permutation representation for G and work with that instead of $\mathcal{O}$. Thus for the remainder of the section, we assume that the following assumption holds.

Assumption 5.6. *If $H=\mathrm{Sym}(\mathcal{O})$, then $\mathcal{C}_H(\bar{C})=1$.*

Under Assumption 5.6, if $H=\mathrm{Sym}(\mathcal{O})$ and B is chosen by the algorithm in Theorem 5.4, then $\bar{B}$ satisfies $\mathcal{C}_H(\bar{B})=\mathcal{C}_H(\bar{C})=1$. Since $B\subseteq C$, $\mathcal{C}_G(C)\subseteq\mathcal{C}_G(B)$, and this implies that $\mathcal{C}_G(B)=\mathcal{C}_G(C)$. We can then extract from Theorem 5.4 the following criteria.

Corollary 5.7. *B is a base for G^C if $[\bar{B}^{\bar{S}},\mathcal{C}_H(\bar{B})]=1$.*

The next lemma sometimes provides a faster way to verify that B is a base for G^C. The proof is clear.

Lemma 5.8. *If $\langle B\rangle$ is transitive in its conjugate action on C, then B is a base.*

The previous results allow us to to obtain a base B for the conjugation action of G on C. By Lemma 5.1 and Assumption 5.2, $G/\mathcal{C}_G(C)$ has a faithful, transitive representation on $\mathcal{O}$. The following result is the start point for extending $\mathcal{O}_B=\{\langle b\rangle : b\in B\}$ to a base for $G^{\mathcal{O}}$.

Lemma 5.9. *Assume B is a base for G^C. Let $\hat{\Gamma}=\cup_{\langle b\rangle\in\mathcal{O}_B}\langle b\rangle^{\langle B\rangle}$ and let L be the point stabilizer $G_{\mathcal{O}_B}$ of $\mathcal{O}_B$. Then $L/\mathcal{C}_G(C)$ acts faithfully on $\hat{\Gamma}$ and the action of an element of L on $\hat{\Gamma}$ is uniquely determined from its action on each $\langle b\rangle\in\mathcal{O}_B$. In particular, this shows that $|L/\mathcal{C}_G(C)|\leq(p-1)^{|\mathcal{O}_B|}$ and $L/\mathcal{C}_G(C)$ is abelian.*

Proof. To prove that $L/\mathcal{C}_G(C)$ acts faithfully on $\hat{\Gamma}$, it suffices to show that $G_{\hat{\Gamma}}=\mathcal{C}_G(C)$. Let $K=G_{\hat{\Gamma}}$. Since B leaves $\hat{\Gamma}$ invariant, it follows that K is normalized by $\langle B\rangle$. On the other hand, K normalizes each subgroup of $\hat{B}$, and thus

$$[K,\langle b\rangle]\subseteq K\cap\langle b\rangle,\ \forall\langle b\rangle\in\mathcal{O}_B.$$

Since $\langle b\rangle\cong Z_p$ for some prime p, either K commutes with B, or else there exists $\langle b\rangle\in\mathcal{O}_B$ with $[K,\langle b\rangle]=\langle b\rangle$. If the latter occurs, then $b\in K$, and so b normalizes, and hence centralizes each $\langle b'\rangle\in\mathcal{O}_B$. This implies that $b\in\mathcal{C}_G(B)=\mathcal{C}_G(C)$, which contradicts Assumption 5.2. Thus K commutes with B and so $K=\mathcal{C}_G(C)$ as required.

Now, let $h\in L$. The action of h on $\langle b\rangle\in\mathcal{O}_B$ is determined by a rule that specifies the value of $b^h \in \langle b\rangle$, for each $\langle b\rangle \in \mathcal{O}_B$. We must show that this rule uniquely determines the action of h on $\hat{\Gamma}$. Let $\langle y\rangle$ be a point of $\hat{\Gamma}$. Then there exists $b_0 \in B$ and $b = b_1 b_2 \cdots b_k \in \langle B\rangle$ with each $b_i \in B$, such that $\langle y\rangle = \langle b_0\rangle^b$. But then $\langle y^h\rangle = \langle {b_0}^{bh}\rangle = \langle {b_0}^{hb'}\rangle = \langle {b_0}^{b'}\rangle$ where $b' = {b_1}^h {b_2}^h \cdots {b_k}^h$.

Since each element of $L/\mathcal{C}_G(C)$ is uniquely determined by its action on the points of $\mathcal{O}_B$, it follows that $|L/\mathcal{C}_G(C)| \le (p-1)^{|\mathcal{O}_B|}$. To show that $L/\mathcal{C}_G(C)$ is abelian, it suffices to prove that the commutator $[x,y]$ of arbitrary elements $x, y \in L$ acts trivially on each $\langle b\rangle \in \mathcal{O}_B$. But this follows from the fact that x and y normalize $\langle b\rangle$ and $Aut(\langle b\rangle)$ is abelian. □

Lemma 5.9 and the next lemma give bounds on the base size and support of $G^{\mathcal{O}}$.

Lemma 5.10. (from Babai, Cooperman, Finkelstein and Seress (1991), Lemma 5.1) *Let $G \subseteq \mathrm{Sym}(\Omega)$ act transitively on Ω. Let b be the size of a minimal base for G and m the minimal size of support for a non-identity element of G. Then $bm \ge |\Omega|$.*

Corollary 5.11. *G has a base of size at most $|\mathcal{O}_B|(1+\log_2(p-1))$. Consequently, the minimal support of an element of G acting on $\mathcal{O}$ is at least $|\mathcal{O}|/|\mathcal{O}_B|(1+\log_2(p-1))$. In particular, if $p=3$, then G has a base of size at most $2|\mathcal{O}_B|$ and the minimal support of an element is at least $|\mathcal{O}|/(2|\mathcal{O}_B|)$.*

Lemma 5.9 and Corollary 5.11 give us the tools to extend the base $\mathcal{O}_B$ into a base for the action of G on $\mathcal{O}$. Lemma 5.9 provides $(p-1)^{|\mathcal{O}_B|}$ candidates for elements of L. Let $\mathcal{H}$ be the set of $(p-1)^{|\mathcal{O}_B|}-1$ rules that specify the possible action of a non-identity element of L on the $|\mathcal{O}_B|$ subgroups $\langle b\rangle \in \mathcal{O}_B$. For each $h \in \mathcal{H}$, the rules prescribed by h allows one to compute $\langle y\rangle^h$ without knowing in advance that there exists a corresponding element of L (or even of $GL(n,q)$).

One can extend $\mathcal{O}_B$ to a base $\mathcal{O}_{B'}$ by repeatedly adding randomly chosen $\langle y\rangle \in \hat{\Gamma}$ to $\mathcal{O}_{B'}$ and removing from consideration all $h \in \mathcal{H}$ such that $\langle y\rangle^h \ne \langle y\rangle$. Eventually, all $h \in \mathcal{H}$ are removed from consideration. The only uncertainty is how long the procedure takes, but the answer returned is always correct.

Corollary 5.11 shows that for a fixed element of $\mathcal{H}$, the probability that a random $\langle y\rangle \in \hat{\Gamma}$ is moved by h is at least $1/|\mathcal{O}_B|(1+\log_2(p-1))$. Thus with constant probability, after we have examined $|\mathcal{O}_B|(1+\log_2(p-1))$ randomly chosen points, we will find a $\langle y\rangle$ moved by h. In particular, with constant probability, the procedure described above will terminate after we have examined $2^{|\mathcal{O}_B|}|\mathcal{O}_B|(1+\log_2(p-1))$ randomly chosen elements of $\mathcal{O}$. This will work fairly well when $|\mathcal{O}_B|$ is small. Further work will be required to make it efficient when $|\mathcal{O}_B|$ is "large".

6. Verification of Simplicity

We continue with the notation of section 5. Thus $C = \{x^g : g \in G\}$ and $\mathcal{O} = \{\langle x^g\rangle : g \in G\}$. In this section, we present an elementary test for simplicity of the permutation representation of G on $\mathcal{O}$. We operate under Assumption 5.2, so that C is not in the kernel of the permutation representation. Thus, by Lemma 5.1, $\bar{G} = G/\mathcal{C}_G(C)$ acts faithfully on $\mathcal{O}$. We will assume that we know a base and strong generating set for G, and hence the order of G. We will also assume that G is primitive. Otherwise, G has a permutation representation of smaller degree and we use that one instead.

Observe that $\{\langle xz\rangle \in \mathcal{O} : z \in \mathcal{C}_G(C)\}$ is a block of $\mathcal{O}$ containing $\langle x\rangle$. By our assumption of primitivity, this block must be trivial. Hence if $[x,g] \in \mathcal{C}_G(C)$, then $[x,g] = 1$. In particular, for each $x \in C$, $\mathcal{C}_{\bar{G}}(\bar{x}) = \bar{\mathcal{C}}_G(x)$. As a consequence of this fact, we may assume that $\mathcal{C}_G(C) = 1$ and that $\bar{G} = G$.

Proposition 6.1. *If K is a regular normal subgroup of G, then K is an elementary abelian r-subgroup for some prime r.*

Proof. The stabilizer in G of the point $\langle x\rangle \in \mathcal{O}$ is $\mathcal{N}_G(\langle x\rangle)$. Since K is a regular normal subgroup of G, it follows that $\mathcal{N}_K(\langle x\rangle) = 1$. In particular, $\mathcal{C}_K(x) = 1$. An equivalent formulation is that K admits a fixed point free automorphism x of order p. By a theorem of Thompson, (Huppert (1967), Haupsatz 8.14) K is nilpotent. But $G^{\mathcal{O}}$ is primitive, and so K must be an elementary abelian r subgroup for some prime r. □

Proposition 6.2. *Let $N = \langle C\rangle$. Then one of the following two cases occurs:*

1. *G has a regular normal subgroup K, and $N = \langle x\rangle K$.*
2. *G has a unique minimal normal subgroup H and $N = \langle x\rangle H$.*

Proof. By the previous result, if K is a regular normal subgroup of G, then K is an elementary abelian r subgroup for some prime r. But then $G = KN_G(\langle x\rangle)$ and (1) follows. To prove (2), we may assume that G does not have a regular normal subgroup. In this case, G has a unique minimal normal subgroup H, which must be transitive. Thus if $g \in G$, then $g = g_1 h$ where $g_1 \in N_G(\langle x\rangle)$ and $h \in H$. Thus

$$(\langle x\rangle)^{g_1 h} = (\langle x\rangle)^h \subseteq \langle x\rangle H,$$

which implies that $N \subseteq \langle x\rangle H$. On the other hand, since N contains both $\langle x\rangle$ and H, equality follows. □

It is fairly straightforward to apply this proposition to test the simplicity of G. First compute $N = \langle C\rangle$. If $|N| < |G|$, then G is not simple. So, assume that $G = N$. Case (1) can only arise if $|G|$ involves exactly two distinct primes. But then G is solvable by Burnside's theorem. Thus if $|G|$ involves more than 2 primes, then we can assume that $N = H\langle x\rangle$ where H is the unique minimal normal subgroup of G.

At this point, simplicity can be verified by showing that $x \in H$, or equivalently that $x \in [N,N]$. An alternative way to verify that $x \in H$ is to check if x is conjugate in G to one of its distinct powers.

7. Lyons' Group Construction

The previous methods were used to construct Lyons' group. Initially, we were given two matrices, $\alpha, \beta \in GL(111,5)$, where $|\alpha| = 2$ and $|\beta| = 5$. These matrices were developed by Robert Wilson and provided to us by Klaus Lux. We describe only how to construct a permutation representation for G acting by conjugation on a conjugacy class $\mathcal{O}$ of subgroups of order 3. See (Cooperman et al., 1995) for a description of how to identify the structure of the resulting permutation representation $G^{\mathcal{O}}$ (G acting on $\mathcal{O}$).

The computation of $G^{\mathcal{O}}$ used the signature described in section 3.1. The additional calculation of a test vector as in section 4 was omitted. Since $G \cong Ly$ was already known in advance, a randomly chosen vector had very high probability of satisfying property $(*)$ from section 3.1, and a verification of a base and strong generating set could later be carried out. An element z of order 67 could be constructed using the method suggested in section 3.2.3, although a heuristic search of pseudo-random elements sufficed. The construction of $\mathcal{O}$ was then carried out in 4-1/2 days on a SPARC-2 using AKCL Common LISP with matrix-vector multiplications written directly in C.

Once $\mathcal{O}$ had been enumerated, the representation $G^{\mathcal{O}}$ was determined by the representation of each element of a generating set for G. Since the storage of permutation generators required considerable disk space, tricks were used to reduce the storage. The maximal subgroup structure for Lyons' group (Conway et.al., 1985) told us that if $x, z \in G$ have orders 2 and 67, respectively and if x does not normalize $\langle z \rangle$, then $G = \langle x, z \rangle$. The generator z of order 67 required no storage at all. Since each orbit of $\langle z \rangle$ had been expanded as soon as an initial point in the orbit were discovered, the points were enumerated a cycle at a time, and so z could be implicitly specified in cycle notation as $(1\ 2\ \cdots 67)(68\ 69 \cdots 138) \cdots$. So, only the remaining generator, the involution x, needed to be stored. Assuming one word (four bytes) of storage per element, this would require 38.4 MB of storage. This was reduced almost by half by maintaining a bit vector for the 9,606,125 "points", such that for each 2-orbit one of the bits of the bit vector was set, and the other was not set. This yielded the orbit representatives, and an accompanying vector of 9,606,125/2 words specifying the "other" point of each 2-orbit, listed in the same order as the orbit representatives of the bit vector. Finally, recognizing that only three bytes were needed instead of four for a number less than 9,606,125 reduced the space savings by another 1/4 to 14.4 Mb, and data compression techniques could have lower the storage further.

8. Parallel Implementation

Finally, there is a recent parallel version of the Lyons construction based on the techniques described here. It is reported on in (Tselman, 1996) and briefly summarized here. Two implementations were done: in the C language on the MasPar MP-1, a SIMD (single instruction, multiple data); and in LISP on a cluster of SPARC-2 workstations using LISP/MPI, a package described in (Cooperman, 1995). LISP/MPI is a package that provides a simple, task-oriented model of parallelism to the LISP programmer. It sits on top of the message passing library, MPI (MPI Forum, 1994), thus inheriting the portability of MPI. The two parallel implementations were tested on Ly, HN, J_3 and $G_2(5)$ with the machine configurations: a MasPar MP-1, 8 SPARC-2's under MPI, 4 SPARC-2's under MPI, and a sequential SPARC-2 implementation.

The MasPar implementation split the parts of each matrix among the individual processors in order to execute each matrix vector multiplication faster. The rest of the computation was done sequentially on the front-end, but since the time for the matrix vector multiplication dominated, a net speedup was observed. Parallelizing only the matrix-vector routines also had the advantage that the human cost of modifying the sequential software was quite minimal. This implementation ran in about one day, about four times faster than a sequential SPARC-2 implementation. The MasPar computation used 112 clusters, where each cluster contained 16 4-bit processor elements. Communication is faster within a cluster and somewhat slower between clusters.

The second parallel implementation used LISP/MPI and required major modifications of the initial sequential design. It also required a new heuristic described in (Tselman, 1996) in order to avoid the expense of resolving hash collisions among distributed parallel processors. This heuristic does not come with a guarantee of correctness, but in the experiments, the heuristic never failed. The advantage of this implementation is that roughly linear speedup with the number of processors was observed. Thus, n SPARC-2 processors resulted in almost a factor of n speedup over the sequential implementation. In particular, with 8 SPARC-2 processors, an elapsed time of 12 hours was observed for the Lyons group.

Acknowledgment The authors gratefully acknowledge discussions with the following people: Alexander Hulpke, Bill Kantor, Wolfgang Lempken, Steve Linton, Gene Luks, Klaus Lux, Gerhard Michler, Reiner Staszewski and Michael Weller. The authors especially acknowledge that the matrix generators for Lyons' group, for which these techniques were tested, were developed by Robert Wilson. We also wish to recognize the suggestion of Gene Luks on how to modify fast completion of subgroup orbits, when the subgroup does not act semiregularly. Finally, we thank Bill Kantor for originally suggesting the "testbed" of Lyons' groups.

References

Aho, A., Hopcroft, J., and Ullman, J. (1974). "The Design and Analysis of Computer Algorithms", Addison-Wesley, p. 245.

Arlazarov, V.L., Dinic, E.A., Kronrod, M.A. and Faradzev, I.A., (1970). "On Economical Construction of the Transitive Closure of a Directed Graph", Dokl. Nauk SSSR **194**, pp. 487–488. English translation in Soviet Math. Dokl. **11**:5, pp. 1209–1210.

Babai, L., and Szemerédi,E., (1984). "On the Complexity of Matrix Group Problems I," *Proc. 25th IEEE FOCS*, Palm Beach, FL, pp. 229–240.

Babai, L., (1991). "Local Expansion of Vertex-Transitive Graphs and Random Generation in Finite Groups", *Proc. 23rd ACM STOC*.

Babai, L. and Beals, R., (1993). "Las Vegas Algorithms for Matrix Groups,", *Proc. 24th IEEE FOCS*, pp. 427–436.

Babai, L., Cooperman, G., Finkelstein, L., and Seress Á. (1991). "Nearly Linear Time Algorithms for Permutation Groups with a Small Base", *Proc. of the 1991 International Symposium on Symbolic and Algebraic Computation* (ISSAC '91), Bonn, pp. 200–209.

Beals, R., (1993). "Computing blocks of imprimitivity for small-base groups in nearly linear time," *DIMACS Series in Discrete Mathematics and Theoretical Computer Science* **11**, pp. 17–26, L. Finkelstein and W.M. Kantor (eds.), AMS, Providence, RI. **11**, 1993, pp. 17–26, L. Finkelstein and W.M. Kantor (eds.), AMS, Providence, RI.

Butler, G. (1976). "The Schreier Algorithm for Matrix Groups", *Proc. ACM Symp. Symbolic and Algebraic Computation* (SYMSAC '76) pp. 167–170, ACM Press, New York.

Butler, G. and Cannon, J. (1982). "Computing in Permutation and Matrix Groups I: Normal Closure, Commutator Subgroups, Series", Math. Comp. **39**, pp. 663–670.

Butler, G. and Cannon, J. (1989). "Cayley, version 4: the user language", *Proceedings of 1988 International Symposium on Symbolic and Algebraic Computation*, Lecture Notes in Computer Science **358**, Springer-Verlag, Berlin, pp. 456–466.

Cannon, J., Gallagher, R., and McAllister, K., (1972). "STACKHANDLER: A Language Extension for Low Level Set Processing. Programming and Implementation Manual", Technical Report 5, Dept. of Pure Mathematics, U. of Sydney.

Cannon, J. and Richardson, J., (1973). "The GROUP System for Investigating the Structure of Groups. User and maintenance manual", Technical Report 8, Dept. of Pure Mathematics, U. of Sydney.

Conway, J. H., Curtis, R. T., Norton, S. P., Parker, R. A., and Wilson, R. A., (1985). *Atlas of finite groups*, Clarendon Press, Oxford.

Cooperman, G. (1995). "STAR/MPI: Binding a Parallel Library to Interactive Symbolic Algebra Systems", *Proc. of International Symposium on Symbolic and Algebraic Computation (ISSAC '95)*, ACM Press, pp. 126–132.

Cooperman, G. and Finkelstein, L., (1993). "Combinatorial Tools for Computational Group Theory", *DIMACS Series in Discrete Mathematics and Theoretical Computer Science* **11**, pp. 53–86, L. Finkelstein and W.M. Kantor (eds.), AMS, Providence, RI.

Cooperman, G., Finkelstein, L., York, B. and Tselman, M., (1994). "Constructing Permutation Representations for Large Matrix Groups", *Proc. of International Symposium on Symbolic and Algebraic Computation (ISSAC '94)*, ACM Press, pp.134–138.

Cooperman, G., Finkelstein, L. and Tselman, M., (1995). "Computing with Matrix Groups using Permutation Representations", *Proc. of International Symposium on Symbolic and Algebraic Computation (ISSAC '95)*, ACM Press, pp. 259-264.

Gollan, H.W. (1995). manuscript.

Gollan, H.W. and Michler, G.O. (1994). "Construction of a 45694-dimensional simple module of Lyons' sporadic group over $GF(2)$.", technical report, Institut für Experimentelle Mathematik, Universität GH Essen, Germany.

Gorenstein, D., (1982). *The Classification of Finite Simple Groups*, Vol. 1, Plenum Press, New York.

Holt, D., Leedham-Green, C., O'Brien, E. and Rees, S., (1994a). "Computing Matrix Group Decompositions with Respect to a Normal Subgroup", *J. Algebra*

Holt, D., Leedham-Green, C., O'Brien, E. and Rees, S., (1994b). "Primitivity Testing for Matrix Groups", *J. Algebra*.

Holt, D. and Rees, S., (1994). "Testing modules for irreducibility", *J. Australian Mathematics Society Ser. A* **57**.

Huppert, B., (1967). "Endliche Gruppen I", Springer Verlag, New York.

Landazuri, V. and Seitz, G.M., (1974). "On the Minimal Degrees of Projective Representations of the Finite Chevalley Groups", *J. of Algebra* **32**, pp. 418-443.

Leedham-Green, C., (1993). Lecture at the MAGMA Conference, London, England, August, 1993.

Luks, E. M., (1992). "Computing in Solvable Matrix Groups", Proc. 33rd IEEE FOCS, pp. 111–120.

Lyons, R., (1972). "Evidence for a New Finite Simple Group", Journal of Algebra, **20**, 540–569.

Message Passing Interface Forum (author) (1994). "MPI: A Message-Passing Interface Standard", *International Journal of Supercomputing Applications* **8**, Number 3/4.

Parker, R., (1984). 'The computer calculation of modular characters. (The Meat-Axe)', in M. Atkinson (ed.), *Computational Group Theory*, Academic Press, London, 267-74.

Praeger, C.E. and Soicher, L.H. *Low Rank Representations and Graphs for Sporadic Groups*, Australian Math. Soc. Lecture Notes, Cambridge University Press, to appear.

C.C. Sims, "Computation with Permutation Groups", in *Proc. Second Symposium on Symbolic and Algebraic Manipulation*, edited by S.R. Petrick, ACM Press, New York, 1971, pp. 23–28.

Sims, C.C., (1973). "The Existence and Uniqueness of Lyons Group", Proceeding of the 1972 Gainesville Conference, North-Holland Mathematics Studies **7**, North-Holland and American Elsevier, pp 138–141.

Tselman, M. (1996), "Computing permutation representations for matrix groups in a distributed environment", *Proc. of DIMACS Workshop on Groups and Computation II* (1995), *DIMACS Series in Discrete Mathematics and Theoretical Computer Science*, L. Finkelstein and W.M. Kantor (eds.), AMS, Providence, RI., to appear.