London Mathematical Society ISSN 1461–1570

ALGORITHMIC RECOGNITION OF GROUP ACTIONS ON ORBITALS

GRAHAM R. SHARP

Abstract

An algorithm is given that recognises (in $O(lN^2 \log N)$ time, where N is the size of the input and l the depth of a precalculated Schreier tree) when a transitive group (G, Ω) is the action on one orbit of the action of G on the set $\Gamma^{(2)}$ of ordered pairs of distinct elements of some G-set Γ (that is, Ω is isomorphic to an orbital of (G, Γ)). This may be adapted to list all essentially different such actions in $O(lN^4 \log N)$ time, where N is the sum of the sizes of the input and output. This will be a useful tool for reducing the degree of a permutation group as an aid to further study of the group.

This algorithm is then extended to provide an algorithm that will (in $O(lN^3 \log N)$ time) recognise when a transitive group is the action on one orbit of the action of G on the set $\Gamma^{\{2\}}$ of unordered pairs of distinct elements of some G-set Γ. An algorithm for finding all essentially different such actions is also provided, running in $O(lN^4 \log N)$ time. (Again, N is the sum of the input and output sizes.) It is also indicated how these results may be applied to the more general problem of recognising when an intransitive group (G, Ω) is isomorphic to $(G, \Gamma^{\{2\}})$ for some G-set Γ.

All the algorithms are practical; most have been implemented in GAP, and the code is made available with this paper. In some cases the algorithms are considerably more practical than their asymptotic analyses would suggest.

1. *Introduction*

Let (G, Γ) be a transitive permutation group. Define $\Gamma^{(2)}$ to be the set of all ordered pairs (α, β) of distinct elements of Γ, and $\Gamma^{\{2\}}$ to be the set of all unordered pairs $\{\alpha, \beta\}$ of distinct elements of Γ. Recall that the orbitals of G are the orbits of G on the set of pairs Γ^2, and that the non-trivial orbitals are those orbitals $(\alpha, \beta)^G$ where $\alpha \neq \beta$; that is, those orbitals that are contained in $\Gamma^{(2)}$.

Given an arbitrary transitive permutation group (G, Ω), we ask whether there is a transitive action of G on a set Γ such that (G, Ω) is isomorphic to the action of G on one of the orbitals of (G, Γ). In Sections 2 to 8, we describe an algorithmic method by which this question may be answered for any group (G, Ω) input to our algorithm by means of a list of generators. The principal application envisaged for this algorithm is to provide a fast method of reducing the degree of an input permutation group to a more manageable size,

The author was supported by the EPSRC.
Received 25 June 1998*; published 23 April 1999*.
1991 Mathematics Subject Classification 20B40, 68Q40

https://doi.org/10.1112/S146115700000005X Published online by Cambridge University Press

to aid further calculation and investigation of the group. Other applications are discussed below.

In Sections 9 to 12 we consider the related question of identifying whether there is an action (G, Γ) of G such that Ω is G-isomorphic to an orbit of G on $\Gamma^{\{2\}}$. The algorithm presented in the first part of this paper will form an integral part of the solution to this second problem.

Initially, we therefore seek an algorithm with the following specification.

Specification 1.1.

Input *Permutations* $g_1, \ldots, g_s \in \mathrm{Sym}(\Omega)$ *generating a transitive permutation group* G *on* Ω.

Output *An action* (G, Γ) *where* $\#\Gamma < \#\Omega$ *and an orbital* $(\alpha, \beta)^G \subseteq \Gamma^{(2)}$ *such that* (G, Ω) *is isomorphic to the action of* G *on* $(\alpha, \beta)^G$*, or the information that no such action and orbital exist.*

Here and throughout this paper, we regard two actions (G, Ω_1) and (G, Ω_2) as being isomorphic, or two sets Ω_1 and Ω_2 as being G-isomorphic, if and only if there exists a bijection $\eta : \Omega_1 \to \Omega_2$ such that $\omega\eta^g = \omega^g\eta$ for all $\omega \in \Omega_1$ and all $g \in G$.

Since a principal application of this algorithm will lie in reducing the degree of an input permutation group, we insist in Specification 1.1 that $\#\Gamma < \#\Omega$. There will be situations when one is interested in orbitals of the same size as the underlying group, and it is easy to modify the algorithm given below for the above specification, so that it can find such actions.

Another application of the algorithms in this paper is to enable one to recognise when the input action (G, Ω) (which may be intransitive) is the *whole* of $\Gamma^{(2)}$ (the set of ordered pairs of distinct elements of Γ) or $\Gamma^{\{2\}}$, rather than just one orbit. The special case where (G, Ω) is transitive and we ask whether Ω is isomorphic to $\Gamma^{\{2\}}$ for some set Γ on which G (necessarily) acts 2-homogeneously has been covered in [8].

To approach the general case we can start with one orbit of the input action, and seek actions of G on sets Γ of a particular size m (*i.e.*, where $\#\Omega = n = m(m-1)$ or $n = \binom{m}{2}$). We are therefore interested in an algorithm fulfilling Specification 1.1 where we may stipulate in advance the size of the set Γ. This is a difficult problem, and instead we formulate a problem that asks for *all* possible actions, from which we may choose the ones we want. Of course, the number of such actions may be very large, and the resulting algorithm infeasibly slow, but we provide techniques to speed up the procedure in practice, partly by removing redundancies from the output. It is, however, a theme of this paper that some of the algorithms are theoretically infeasible, but that they are in practice applicable to the vast majority of permutation groups of suitable degree; the practical upper limit on the degree of the input group is quite high, somewhere between 4000 and 10 000 on a 200 MHz Pentium.

Specification 1.2.

Input *Permutations* $g_1, \ldots, g_s \in \mathrm{Sym}(\Omega)$ *generating a transitive permutation group* G *on* Ω.

Output *The set of all (essentially different) pairs, where each pair consists of an action* (G, Γ) *and an orbital* $(\alpha, \beta)^G$ *such that* (G, Ω) *is isomorphic to the action of* G *on* $(\alpha, \beta)^G$.

https://doi.org/10.1112/S146115700000005X Published online by Cambridge University Press

By 'essentially different' we mean that if two actions (G, Γ_1) and (G, Γ_2) in the output are isomorphic by an isomorphism that identifies the corresponding orbitals, then we would like to regard them as the same. This will be formalized in Definition 3.1.

One further application of these algorithms is to find graphs (directed or undirected) where the edge set can be identified with the input set Ω in such a way that the group G embeds in the automorphism group of the graph.

In [**8**], special techniques based on the classification of finite 2-transitive groups are used to obtain a recognition algorithm that, for a very special case, runs in nearly linear time (if the orbits of a point stabilizer are known). The algorithms in the present paper are based on far more elementary mathematics, and have significantly worse asymptotic complexity. When the results of the present paper are applied to actions with a solution in the sense of [**8**], we find that the algorithms for Specifications 1.1 and 9.1, which return just one action, do not in general return a solution in the sense of [**8**]. The algorithms for Specifications 1.2 and 9.2, which return all possible actions, include in their output all 'essentially different' solutions in the sense of [**8**], and these can easily be extracted by reference to the degree of the output actions.

It would be interesting to examine the various cases of [**8**] to investigate the consequences of the definitions of equivalence of solutions contained in this paper. For example, one could ask how many different inequivalent solutions there can be in each case, and one could investigate the consequences for the algorithms in [**8**] of the various results in the present paper which use equivalence of solutions to reduce the effort involved in the search for a solution. Unfortunately, the author has not had time to undertake these investigations, so they are left as open questions for the reader to ponder.

In practice, when running algorithms from the present paper on groups that have a solution in the sense of [**8**], one finds that in most cases these groups do not have very many solutions in the sense of the present paper, and the program does not therefore take inordinately longer than the implementation of the algorithm in [**8**].

Throughout this paper, n will be the cardinality of the (finite) set Ω, and s will be the cardinality of the (finite) input set of generators used to define G.

2. *Finding solution pairs*

Let ω be a fixed element of Ω. Suppose (G, Ω) is isomorphic to the action of G on an orbital Δ of (G, Γ). Then ω corresponds to some element (α, β) of Δ, and $\Delta = (\alpha, \beta)^G$, and $G_\omega = G_\alpha \cap G_\beta$. Let $x \in G$ map α to β. Then $G_\beta = G_\alpha^x$, so in fact $G_\omega = G_\alpha \cap G_\alpha^x$. On the other hand, whenever we have a subgroup J with $G_\omega = J \cap J^x$ then we can take Γ to be the action of G by right multiplication on the set $\cos(G : J)$ of cosets of J in G, and (G, Ω) is isomorphic to the orbital $(J, Jx)^G$ (and also to the orbital $(Jx, J)^G$, which may be distinct from the first one). We have proved:

Proposition 2.1. *Fix $\omega \in \Omega$. If the action (G, Ω) is isomorphic to the action of G on an orbital Δ of (G, Γ), then there is a stabilizer $J = G_\alpha$ for some $\alpha \in \Gamma$, and an element $x \in G$ such that $J \cap J^x = G_\omega$ for some $\omega \in \Omega$. Conversely, if $J \leqslant G$ and $x \in G$ satisfy $J \cap J^x = G_\omega$ then (G, Ω) is isomorphic to the action of G on an orbital of its action on cosets of J.* □

Definition 2.2. A *solution pair* will be a pair (J, x) where $J \leqslant G$ and $x \in G$ satisfy $J \cap J^x = G_\omega$.

https://doi.org/10.1112/S146115700000005X Published online by Cambridge University Press

Our approach to Specification 1.2 will be a search over the set of pairs (J, x) where J is a subgroup of G containing G_ω, and x an element of G. We shall later show that we need not consider very many elements x. We shall find that part of the same search will suffice for Specification 1.1. The restriction on subgroups J for a particular x follows from two simple lemmas.

Lemma 2.3. *Suppose $G_\omega \leqslant J \leqslant G$ and $x \in G$, and $J \cap J^x = G_\omega$. Then $J \geqslant \langle G_\omega, G_\omega^{x^{-1}} \rangle$.*

Proof. Since $G_\omega \leqslant J^x$, we have $G_\omega^{x^{-1}} \leqslant J$. □

Lemma 2.4. *Suppose $\langle G_\omega, G_\omega^{x^{-1}} \rangle \leqslant J' \leqslant J$ and $J \cap J^x = G_\omega$. Then $J' \cap J'^x = G_\omega$.*

Proof. Certainly $J' \cap J'^x \leqslant J \cap J^x = G_\omega$. On the other hand, $J' \geqslant G_\omega^{x^{-1}}$ so $J'^x \geqslant G_\omega$ and as $J \geqslant G_\omega$ as well; the result follows. □

Therefore for a fixed value of x, we can enumerate all subgroups J with $J \cap J^x = G_\omega$ by starting with $J = \langle G_\omega, G_\omega^{x^{-1}} \rangle$, and checking whether $J \cap J^x = G_\omega$ for this J. Then recursively for any J satisfying this equation we form all proper supergroups $\bar{J}$ of J in which J is maximal, and test each of these in turn to see whether $\bar{J} \cap \bar{J}^x = G_\omega$.

The following procedure uses this technique to enumerate all solution pairs for the chosen value of x.

Algorithm 2.5.

Input Permutation group (G, Ω) as a list of generators, a point $\omega \in \Omega$ and an element $x \in G$.

Output List of all solution pairs (for the given ω) (J, x) for $J \leqslant G$ and x as it was input.

```
Function: ProcessSubgroup($J$)
    If $J \cap J^x = G_\omega$ then
        Output $(J, x)$;
        Form $\mathcal{J} := \{\bar{J} \leqslant G \mid J <_{\max} \bar{J}\}$;
        For $\bar{J} \in \mathcal{J}$ do ProcessSubgroup($\bar{J}$); End for;
    End if;
End function;
ProcessSubgroup($\langle G_\omega, G_\omega^{x^{-1}} \rangle$);
End.
```

We shall show later how this theoretical procedure can be translated into a practical algorithm.

3. *Equivalence of solution pairs*

As we saw earlier, every action of G with an orbital action isomorphic to (G, Ω) has a point stabilizer J for which there exists $x \in G$ with $J \cap J^x = G_\omega$, and for a given value of x we can find all subgroups J fulfilling this condition. We now consider which values of x we need to look at to make sure we find all the actions in which we are interested, and identify the corresponding orbitals. We introduce a notion of equivalence of solution pairs to formalize the 'essentially the same' aspect of Specification 1.2; it will also reduce the number of elements x that need to be considered for Specification 1.1.

https://doi.org/10.1112/S146115700000005X Published online by Cambridge University Press

Definition 3.1. Two solution pairs (J_1, x_1) and (J_2, x_2) are called *equivalent* if there exists $g_0 \in G$ such that $J_2 = J_1^{g_0}$ and $x_2 \in J_2 x_1^{g_0} J_2$.

A simple calculation shows that the relation of Definition 3.1 is an equivalence relation on the set of solution pairs.

Proposition 3.2. *Two solution pairs (J_1, x_1) and (J_2, x_2) are equivalent if and only if there exists a G-isomorphism $\theta : \cos(G : J_1) \to \cos(G : J_2)$ such that*

$$((J_1, J_1x_1)^G)\theta = (J_2, J_2x_2)^G.$$

Proof. Suppose θ is a G-isomorphism as described. Then the trivial coset J_1 maps under θ to a coset of J_2, say J_2g. As θ is a G-isomorphism, the stabilizers of these cosets are the same, so $J_1 = J_2^g$. As θ maps $(J_1, J_1x_1)^G$ to $(J_2, J_2x_2)^G$, it must map J_1x_1 to J_2x_2hg for some $h \in J_2$. But since θ is a G-isomorphism, it must also map J_1x_1 to $(J_1\theta)x_1$, which equals J_2gx_1. Thus $J_2x_2hg = J_2gx_1$ and so $x_2 \in J_2x_1^{g^{-1}}h^{-1}$, which is as required (with $g_0 = g^{-1}$ in Definition 3.1).

Conversely, suppose $g_0 \in G$ is such that $J_2 = J_1^{g_0}$ and $x_2 \in J_2x_1^{g_0}J_2$. Define $\theta : \cos(G : J_1) \to \cos(G : J_2)$ by $(J_1g)\theta = J_2g_0^{-1}g$ for all $g \in G$. It is easily checked that this is a well-defined G-isomorphism with the required property. □

Remark 3.3. Since any orbital $(J, Jx)^G$ is isomorphic to the reverse orbital $(Jx, J)^G$ or $(J, Jx^{-1})^G$, in a natural way, Definition 3.1 could be extended to reflect this. We could do this by replacing the condition '$x_2 \in J_2x_1^{g_0}J_2$' in that definition by 'either $x_2 \in J_2x_1^{g_0}J_2$ or $x_2 \in J_2(x_1^{-1})^{g_0}J_2$'. Most of the following results based on the above definition could be altered to take account of the changed definition, but the alterations would be quite complicated. It turns out not to be as easy to test for this sort of equivalence and, as we only ever expect to gain a small factor with this change, it was left out of the final algorithm.

We now prove an important characterization of equivalence of solution pairs, and deduce as a corollary that if we have two solution pairs that are equivalent, then any further solution found by starting from one of them is equivalent to one found by starting from the other.

Theorem 3.4. *Let (J_1, x_1) be a solution pair and let $J_2 \leqslant G$ and $x_2 \in G$. Then (J_2, x_2) is a solution pair equivalent to (J_1, x_1) if and only if there exists $g \in N_G(G_\omega)$ such that $J_2 = J_1^g$ and $x_2 \in J_2x_1^g$.*

Proof. Suppose first that $J_2 \cap J_2^{x_2} = G_\omega$ and that there exists $g_0 \in G$ such that $J_2 = J_1^{g_0}$ and $x_2 \in J_2x_1^{g_0}J_2$. Then there exists $z \in J_2$ such that $x_2 \in J_2x_1^{g_0}z$, and since $J_2 \cap J_2^{x_2} = G_\omega$, we get that

$$J_2 \cap J_2^{x_1^{g_0}} = G_\omega^{z^{-1}}.$$

It follows that

$$J_1^{g_0} \cap J_1^{x_1g_0} = G_\omega^{z^{-1}}$$

since $J_2 = J_1^{g_0}$, and so

$$J_1 \cap J_1^{x_1} = G_\omega^{z^{-1}g_0^{-1}}.$$

However $J_1 \cap J_1^{x_1} = G_\omega$ by hypothesis, so $z^{-1}g_0^{-1}$ normalizes G_ω. We can therefore take $g = g_0z$, which normalizes G_ω, and also $J_1^g = J_1^{g_0z} = J_2^z = J_2$ and $J_2x_1^g = J_2x_1^{g_0z} = J_2x_1^{g_0}z$, which contains x_2.

https://doi.org/10.1112/S146115700000005X Published online by Cambridge University Press

Conversely, suppose that there exists g normalizing G_ω such that $J_2 = J_1^g$ and $x_2 \in J_2 x_1^g$. If we can show that (J_2, x_2) is a solution pair, then it will immediately follow that it is equivalent to (J_1, x_1) by taking $g_0 = g$ in the definition of equivalence.

So it suffices to prove that $J_2 \cap J_2^{x_2} = G_\omega$. We have

$$J_1 \cap J_1^{x_1} = G_\omega^{g^{-1}}$$

since (J_1, x_1) is a solution pair and g normalizes G_ω. Now

$$J_2 \cap J_2^{x_1^g} = G_\omega$$

and so

$$J_2 \cap J_2^{x_2} = G_\omega$$

since $x_2 \in J_2 x_1^g$. □

Corollary 3.5. *Suppose (J_1, x_1) and (J_2, x_2) are equivalent solution pairs, and that $(\bar{J}_1, x_1)$ is also a solution pair, where $\bar{J}_1 \geqslant J_1$. Then there exists $\bar{J}_2 \geqslant J_2$ such that $(\bar{J}_2, x_2)$ is a solution pair that is equivalent to $(\bar{J}_1, x_1)$.*

Proof. By Theorem 3.4 there exists $g \in N_G(G_\omega)$ such that $J_2 = J_1^g$ and $x_2 \in J_2 x_1^g$. Let $\bar{J}_2 = \bar{J}_1^g$. Then $\bar{J}_2$ contains J_2 and so the result now follows from a second application of Theorem 3.4. □

This means that not only can we safely discard one of a pair of equivalent solutions in the output, we actually need to process only one of any pair of equivalent solutions when looking for solutions containing ones that have already been found. This is very important, and will reduce the workload of the algorithm considerably (even though it will probably not reduce its overall worst-case complexity).

There now follow two results, which between them characterize the equivalence of solutions in a more practical way.

Proposition 3.6. *Let (J, x_1) be a solution pair. If $x_2 \in J x_1 G_\omega$ then (J, x_2) is a solution pair and is equivalent to (J, x_1).*

Proof. There exists $g \in G_\omega$ such that $x_2 \in J x_1 g$. Then $J^g = J$ as $G_\omega \leqslant J$ and $J x_1^g = J x_1 g$, and so the result follows by Theorem 3.4. □

It is natural to define a set of pairs whose elements correspond to collections of equivalent solution pairs that are closely related as in Proposition 3.6:

$$\mathcal{X} = \{(J, JxG_\omega) \mid (J, x) \text{ is a solution pair}\}.$$

Each element (J, JxG_ω) of $\mathcal{X}$ satisfies the condition that for every $x' \in JxG_\omega$, the pair (J, x') is a solution pair equivalent to (J, x).

Proposition 3.7. *The normalizer $N = N_G(G_\omega)$ acts on $\mathcal{X}$ by conjugation:*

$$(J, JxG_\omega)^g = (J^g, J^g x^g G_\omega) \textit{ for all } g \in N \textit{ and all } (J, JxG_\omega) \in \mathcal{X}.$$

The kernel of this action contains G_ω and so the action induces an action of the quotient N/G_ω.

The orbits of this action are the equivalence classes under the equivalence relation of Definition 3.1, in the sense that solution pairs (J_1, x_1) and (J_2, x_2) are equivalent if and only if the orbit of N on $\mathcal{X}$ that contains $(J_1, J_1 x_1 G_\omega)$ also contains $(J_2, J_2 x_2 G_\omega)$.

https://doi.org/10.1112/S146115700000005X Published online by Cambridge University Press

Proof. Observe that the set $J^g x^g G_\omega$ is simply the image of the set JxG_ω under the action of conjugation of elements of G by g, since g normalizes G_ω, and it follows that the image $J^g x^g G_\omega$ is not dependent on the choice of x. The image $(J^g,\ J^g x^g G_\omega)$ lies in $\mathcal{X}$ whenever $(J,\ JxG_\omega)$ does, by Theorem 3.4. The mapping defined above is an action since it is derived from the action of N on G by conjugation.

If $g \in G_\omega$, then $J^g = J$ as $G_\omega \leqslant J$, and $J^g x^g G_\omega = g^{-1}JxG_\omega g = JxG_\omega$, for all solution pairs (J, x), so G_ω lies in the kernel of the action of N on $\mathcal{X}$. Finally the characterization of orbits in terms of equivalence is a consequence of Theorem 3.4 and Definition 3.1. □

Corollary 3.8. *Suppose there exists $g \in N$ such that $x_2 \in G_\omega x_1^g G_\omega$ for two elements x_1, x_2 of G. If one of $(\langle G_\omega, G_\omega^{x_1^{-1}} \rangle, x_1)$ and $(\langle G_\omega, G_\omega^{x_2^{-1}} \rangle, x_2)$ is a solution pair, then the other is as well, and they are equivalent.*

Proof. Let $J_i = \langle G_\omega, G_\omega^{x_i^{-1}} \rangle$ for $i = 1, 2$. If $x_2 \in G_\omega x_1^g G_\omega$, then $x_1 \in G_\omega x_2^{g^{-1}} G_\omega$ and vice-versa, so without loss we may assume that $(J_1,\ J_1 x_1 G_\omega)$ lies in $\mathcal{X}$. We have $x_2^{-1} \in G_\omega g^{-1} x_1^{-1} gh$ for some $h \in G_\omega$, so $G_\omega^{x_2^{-1}} = G_\omega^{x_1^{-1}gh}$. Then $J_2 = \langle G_\omega^{h^{-1}g^{-1}}, G_\omega^{x_1^{-1}} \rangle^{gh} = J_1^{gh}$ but as $h \in J_1^g$, we get $J_2 = J_1^g$. By Proposition 3.7, therefore, $\mathcal{X}$ contains the pair $(J_2,\ J_2 x_1^g G_\omega)$ and since $G_\omega x_1^g G_\omega \subseteq J_2 x_1^g G_\omega$, it follows that (J_2, x_2) is a solution pair, and that it is equivalent to (J_1, x_1). □

This corollary is even more helpful than previous results on equivalence; rather than merely enabling us to discard solutions after considering them, it significantly reduces the number of elements x that we need to consider to start with.

4. *High-level algorithm*

We present a high-level version of the algorithm for Specification 1.2 (an algorithm for Specification 1.1 can be easily deduced). Discussion of how this procedure is to be converted into a practical procedure with reasonable asymptotic complexity is deferred until the next section.

The algorithm is in two parts, a main procedure (Algorithm 4.1 on page 9) and a subroutine, called Add (Algorithm 4.2 on page 9). Instead of a recursive approach (as in Algorithm 2.5), we store solutions awaiting processing in a list of pairs L. The purpose of the subroutine Add is to update L by adding newly discovered solutions, and to find and filter out solutions that are equivalent to previously discovered solutions. Much of the remainder of the algorithm is similar to the corresponding parts of Algorithm 2.5.

Each entry of L is a pair (J, X) such that the solution pairs to be processed are (J, x) for each $x \in X$. They are stored in L in increasing order of the size of J, and each subgroup J will be the subject of at most one entry in L (although that entry will move within L, eventually working its way to the head of the list). We always process the first entry in L, thus ensuring that the smallest outstanding subgroup is processed. This means that no subgroup will ever need to be processed twice, since whenever a new solution that will need processing is produced, the subgroup in that solution is always strictly larger than the subgroup currently being processed.

As in Algorithm 2.5, processing a set of solutions (J, X) involves finding all minimal overgroups $\bar{J}$ of J (*i.e.*, all subgroups $\bar{J}$ that have J as a maximal subgroup) and for each such $\bar{J}$ and each $x \in X$, checking whether $\bar{J} \cap \bar{J}^x = G_\omega$. For each $\bar{J}$ for which there exists x satisfying this condition, the pair $(\bar{J}, \bar{X})$ is added to L in an appropriate place according to the size of $\bar{J}$. Here $\bar{X}$ is the subset of X of elements x for which $\bar{J} \cap \bar{J}^x = G_\omega$.

The purpose of the subroutine Add is to ensure that only one representative from each equivalence class (under the relation of Definition 3.1) is actually added to L. It also ensures that only one subgroup from each N-conjugacy class of subgroups is used, thus reducing the number of subgroups J for which the set of minimal overgroups must be calculated.

Those sets $(\bar{J}, \bar{X})$ of solutions which, according to the description in the previous paragraph, are to be added to L are in fact passed to this subroutine. For each solution pair $(\bar{J}, x)$, one of three things can happen.

(i) It can be entered as a new entry into L.

(ii) It can be conjugated to give an equivalent solution $(\bar{J}', x')$ where $\bar{J}'$ already appears in L, though no solution equivalent to $(\bar{J}', x')$ yet appears in that entry (or anywhere else); then x' is added to the second component of the current entry containing $\bar{J}'$.

(iii) It may be decided that $(\bar{J}, x)$ is equivalent to a solution already in L, and so it is not added.

The first few lines of Algorithm 4.1 use the results of Corollary 3.8 to restrict the number of x with which to start, and then initiate the process by forming and testing the subgroups $\langle G_\omega, G_\omega^{x^{-1}} \rangle$, and adding those that pass the test to L. Again the procedure Add is used for this because Corollary 3.8 does not pick up every equivalence between solutions of this form; also, we need to check for conjugacy between the subgroups of non-equivalent solutions.

The subroutine Add (Algorithm 4.2) takes a pair (J, X) to add to L, as described above. It starts by identifying those entries (J', X') in L for which J' has the same size as J, and then decides which, if any, has J' conjugate to J by an element of the normalizer N. Note that by construction of L, there will be at most one such entry. The entry is $L[i]$; if none were found then a new entry in L is created with J as subgroup, immediately after all the other entries in L whose subgroups have the same size as J. Although the entry is created with an empty second component, if $X \neq \emptyset$ (as is always the case when Add is called) then this second component can be guaranteed not to be empty at the end of the subroutine.

The second part of the subroutine handles the elements of X. We know that any solution in L that is equivalent to a solution in (J, X) must be in the entry $L[i] = (J', X')$. Let $\mathcal{Y}$ denote the subset of $\mathcal{X}$ consisting of pairs whose first component is J'. Then the stabilizer $N_{J'}$ of N/G_ω in its action on $\mathcal{X}$ acts on $\mathcal{Y}$. Each orbit of N/G_ω on $\mathcal{X}$ that contains a pair (J', x) containing J' obviously meets $\mathcal{Y}$, and the intersection of such an orbit with $\mathcal{Y}$ is precisely the orbit of (J', x) under $N_{J'}$.

The set $\mathcal{X}_1$ is then the set of equivalence classes of the relation on G that relates x and y if and only if $(J', J'xG_\omega)$ and $(J', J'yG_\omega)$ are in the same $N_{J'}$-orbit. Thus, solutions (J', x) and (J', y) are equivalent if and only if x and y are in the same member set of $\mathcal{X}_1$. The set T will be the union of those sets in $\mathcal{X}_1$ that contain elements of X'. As X' is enlarged, T is expanded correspondingly.

If (J, x) is equivalent to (J', x') (for $x \in X, x' \in X'$) then so is (J^g, x^g) for any $g \in N$, in particular for the $g \in N$ chosen in the algorithm, for which $J^g = J'$. But (J', x^g) is equivalent to some (J', x') if and only if $x^g \in T$, by Proposition 3.7. Therefore, for each $x \in X$, we test whether $x^g \in T$; if not, then (J', x^g) is an equivalent solution to (J, x) that is not equivalent to any solution currently in L, so x^g is added to the second component of $L[i]$ and T is updated accordingly.

https://doi.org/10.1112/S146115700000005X Published online by Cambridge University Press

Algorithm 4.1.

```
Form $N := N_G(G_\omega)$;
Form $\mathcal{D}$, the set of double cosets of $G_\omega$ in $G$;
Let $\mathcal{R}$ be a subset of $\mathcal{D}$ containing one double coset from each orbit of $N$ in its action on $\mathcal{D}$ by conjugation;
Let $X$ be a subset of $G$ containing one element from each double coset in $\mathcal{R}$;
Let $L := [\,]$;
For $x \in X$ do
    Let $J := \langle G_\omega, G_\omega^{x^{-1}} \rangle$;
    If $J \cap J^x = G_\omega$ then $\mathsf{Add}(J, \{x\})$; end if;
End for;
While $L \neq [\,]$ do
    Let $(J, X) := L[1]$; remove $L[1]$ from $L$;
    Output $(J, X)$;
    Form $\mathcal{J} := \{\bar{J} \leqslant G \mid J <_{\max} \bar{J}\}$;
    For $\bar{J} \in \mathcal{J}$ do
        Let $\bar{X} := \{x \in X \mid \bar{J} \cap \bar{J}^x = G_\omega\}$;
        If $\bar{X} \neq \emptyset$ then $\mathsf{Add}(\bar{J}, \bar{X})$; end if;
    End for;
End while;
End.
```

Algorithm 4.2.

```
Subroutine $\mathsf{Add}(J, X)$
    Set $i_0$ to be the smallest positive integer such that $i_0 > \#L$ or the first component of $L[i_0]$ has the same size as $J$; set $i_1$ to be the smallest positive integer such that $i_1 > \#L$ or the first component of $L[i_1]$ has size strictly greater than that of $J$;
    Find the first value of $i$ with $i_0 \leqslant i < i_1$ for which there exists $g \in N/G_\omega$ such that $J'^g = J$, where $(J', X') = L[i]$; if none exists then set $i := i_1$;
    If $i = i_1$ then insert a new entry $(J, \emptyset)$ into the list $L$ with index $i$, shifting all entries beyond that point up by one, so $L[i+1]$ is the old $L[i]$, etc.;
    Let $(J', X') := L[i]$;
    Let $N_{J'} := \{g \in N/G_\omega \mid J'^g = J'\}$;
    Form $\mathcal{D} := \{J'xG_\omega \mid x \in G\}$;
    Form $Z$, whose elements are the orbits of $N_{J'}$ in its action by conjugation on $\mathcal{D}$;
    Form $\mathcal{X}_1 := \{\bigcup \mathcal{A} \mid \mathcal{A} \in Z\}$;
    Let $T := \bigcup\{D \in \mathcal{X}_1 \mid X' \cap D \neq \emptyset\}$;
    Fix $g \in N$ such that $J^g = J'$;
    For $x \in X$ do
        If $x^g \notin T$ then
            Add $x^g$ to the second component of $L[i]$;
            Join the element of $\mathcal{X}_1$ containing $x^g$ to $T$;
        End if;
    End for;
End subroutine.
```

https://doi.org/10.1112/S146115700000005X Published online by Cambridge University Press

By this construction, we ensure that no two elements of the set of solutions $\{(J, x) \mid (\exists i, X)((x \in X) \wedge (L[i] = (J, X)))\}$ are equivalent, and that no two subgroups from the set $\{J \mid (\exists i, X)(L[i] = (J, X))\}$ are conjugate under the action of N.

5. *Lower-level algorithm*

In this section we introduce computationally efficient ways of representing and manipulating the objects required for the above algorithm.

For $\alpha \in \Omega$, let $r(\alpha)$ denote a representative of α, that is, $r(\alpha)$ is an element of G mapping ω to α. We will assume that the $r(\alpha)$ are stored as words in the elements of a generating set in a Schreier tree of depth l. (A Schreier tree, sometimes called a Schreier vector, is an efficient means of storing a transversal for a point stabilizer in a permutation group without storing the permutations explicitly. See [4] for example. If the maximum depth of the tree is l then the image of any $\beta \in \Omega$ under any $r(\alpha)$ can be calculated in $O(l)$ time, and the whole permutation $r(\alpha)$ in $O(nl)$ time.)

There is a one-to-one correspondence between subgroups of G containing G_ω and blocks of imprimitivity of (G, Ω) containing ω. Instead of the subgroups J we store the corresponding block ω^J. Accordingly, a lot of calculations with blocks will be necessary. We will often work with a small subset S of Ω, and will want to find the smallest block containing S and ω. This is ω^H where $H = \langle G_\omega, \{r(\alpha) \mid \alpha \in S\}\rangle$.

It is easier to calculate the orbits of G_ω (which can be done in $O(sn^2 \log n)$ time and $O(ns)$ space using the Schreier generators, or much more quickly and with high probability by taking a small random subset of the Schreier generators) than it is to find a small generating set for G_ω. Furthermore, we will need to know the orbits of G_ω anyway when we use the algorithm of Schönert and Seress [6] for calculating minimal blocks in a group action. We will therefore assume that we know the orbits of G_ω, but not necessarily a generating set for G_ω. This will save a significant amount of space (and some time), as by using a Schreier tree we can avoid having to explicitly store any permutation apart from the original generators of the group.

The following lemma and proposition show that the usual algorithm for calculating orbits given the generators of a group can be adapted to cope with this situation. Note that we do not assume that the orbit to be calculated is a block.

Lemma 5.1. *Let $H = \langle H_1, H_2, \ldots, H_t\rangle$. Then ω^H is the unique minimal (with respect to inclusion) subset of Ω that contains ω and is a union of H_i-orbits for $i = 1, \ldots, t$.*

Proof. The proof is straightforward, and is omitted here. □

The following proposition will only be applied either with $J = G_\omega$ or as part of a sequence of applications that start with $J = G_\omega$ and gradually extend J (see Proposition 5.3).

Proposition 5.2. *Suppose we are given the orbits of a subgroup J of G in the form of linked lists, together with an array indexed by Ω, mapping each element of Ω to the first element of the list containing that element. Suppose also that a transversal $\{r(\alpha) \mid \alpha \in \Omega\}$ is given, in the form of a Schreier tree of depth l. Let T, S be subsets of Ω, and ω_0 a given element of Ω. The following algorithm will calculate ω_0^H where H is the subgroup of Ω generated by the groups $J^{r(\alpha)}$ for each $\alpha \in T$ together with the elements $r(\alpha)$ for each $\alpha \in S$. (Note that if J contains G_ω and either $\omega \in T$ or $S = \emptyset$, then H is independent of the choice of transversal.) The running time is $O((\#T + \#S)l\#\omega_0^H)$, and the space used is $O(n\#T)$.*

https://doi.org/10.1112/S146115700000005X Published online by Cambridge University Press

```
 1. Set $\Delta := \emptyset$, $Q := \{\omega_0\}$;
 2. While $Q \neq \emptyset$ do
 3.     Choose $\xi \in Q$; add $\xi$ to $\Delta$ and remove $\xi$ from $Q$;
 4.     For $\alpha \in T$ do
 5.         If $\xi^{J^{r(\alpha)}} \not\subseteq \Delta \cup Q$ then
 6.             Set $Q := Q \cup (\xi^{J^{r(\alpha)}}) \setminus \Delta)$;
 7.         End if;
 8.     End for;
 9.     For $\alpha \in S$ do
10.         If $\xi^{r(\alpha)} \notin \Delta \cup Q$ then set $Q := Q \cup \{\xi^{r(\alpha)}\}$; end if;
11.     End for;
12. End while;
13. Output $\Delta$;
14. End.
```

Proof. It is easy to see that after execution, Δ contains ω_0, is a union of $J^{r(\alpha)}$-orbits for each $\alpha \in T$ and a union of $\langle r(\alpha)\rangle$-orbits for each $\alpha \in S$, and that it is the smallest such set, so correctness follows from Lemma 5.1.

The time bound is obtained by taking care in the implementation. We use an array indexed by Ω, each of whose entries indicates whether the corresponding element of Ω is a member of Δ or Q, so membership of $\Delta \cup Q$ can be tested in constant time; for each element of T we also maintain an array of flags indexed by the heads of the linked lists representing the J-orbits; the flag corresponding to an element $\alpha \in T$ and a J-orbit A will be set when it has been shown that the whole of $A^{r(\alpha)}$ is contained in $Q \cup \Delta$, that is when lines 5–7 have been executed for this value of α and an ξ such that $\xi^{r(\alpha)^{-1}} \in A$. This will enable us to decide in line 5 whether the image under $r(\alpha)$ of the J-orbit containing $\xi^{r(\alpha)^{-1}}$ has previously been added to Q; using the data structure described this test can be done in constant time once we know $\xi^{r(\alpha)}$. Line 5 is executed $\#T\#\omega_0^H$ times, and a Schreier tree calculation is needed each time to find $x^{r(\alpha)}$. Thus all these checks can be made in total time $O(l\#T\#\omega_0^H)$.

It follows that we only need to do an element-by-element check in lines 5 and 6 once for each pair α, A where $\alpha \in T$ and A is a J-orbit. Using the linked list structure, the number of individual element checks made in lines 5 and 6 during the whole execution of the algorithm is therefore $\#T\#\omega_0^H$, and as each check requires a Schreier tree calculation, these also take $O(l\#T\#\omega_0^H)$ time. It is straightforward to see that the same data structures allow for the execution of lines 9–11 in $O(l\#S\#\omega_0^H)$ time, and it is clear that all the other lines execute within the time bound stated in the proposition. The space bound is clear from the use of the data structures described. □

Proposition 5.3. *If $G_\omega \leqslant \bar{J} \leqslant G$ and the block $\omega^{\bar{J}}$ is known, then given the orbits of G_ω on Ω and a Schreier tree of depth l, we can calculate a subset of $\omega^{\bar{J}}$ of size at most $\log \#\omega^{\bar{J}}$ that generates $\omega^{\bar{J}}$ as a block containing ω, in $O(l\#\omega^{\bar{J}} \log \#\omega^{\bar{J}})$ time.*

Proof. We will obtain a chain of subgroups J_i with $G_\omega = J_0 < J_1 < \cdots < J_r = \bar{J}$ as follows. We will have $J_0 = G_\omega$ and $J_{i+1} = \langle J_i, r(\alpha_i)\rangle$ where $\alpha_i \in \omega^{J} \setminus \omega^{J_i}$, as long as this is possible. Each α_i will be determined by calculating the orbits of J_i on ω^{J} and then choosing a suitable point.

The procedure is to use Proposition 5.2 repeatedly. For each value of i, starting with $i = 0$, several calls will be made, with $T = \{\omega\}$, $J = J_i$ and $S = \{\alpha_i\}$, using various

https://doi.org/10.1112/S146115700000005X Published online by Cambridge University Press

elements as ω_0 sufficient to find all the orbits of J_i on $\omega^{\bar{J}}$. Thus we find the orbits of J_{i+1} on $\omega^{\bar{J}}$; when the orbits of J_{i+1} are known we can then choose α_{i+1} and (in $O(\#\omega^{\bar{J}})$ time) process the orbits of J_{i+1} to get into a suitable form to be used as input on the next iteration. □

As mentioned earlier, there is a one-to-one correspondence between subgroups of G containing G_ω and blocks of imprimitivity of (G, Ω) containing ω. Instead of the subgroups J we store the corresponding block ω^J. The manipulations we require with these subgroups are: conjugation by elements x, where the resulting subgroup also contains G_ω, testing whether the intersection $J \cap J^x$ is G_ω, testing the equality of two such subgroups, and forming the set of minimal overgroups for a subgroup J.

Lemma 5.4. *Let $S \subseteq \Omega$ and let B be the smallest block containing $\{\omega\} \cup S$. Let $g \in G$. Then $\omega^{G^g_{\{B\}}}$ is the smallest set containing ω which is a union of G^g_ω-orbits and which is closed under the action of the set $\{r(\alpha)^g \mid \alpha \in S\}$.*

Proof. We have $G^g_{\{B\}} = \langle G^g_\omega, \{r(\alpha)^g \mid \alpha \in S\}\rangle$, and the result follows by Lemma 5.1. □

A suitable set S can be calculated and stored with each block; indeed, the block can be calculated from S, the orbits of G_ω and a Schreier tree to obtain the representatives $r(\alpha)$. This idea of calculating blocks from the orbits of G_ω and the representatives of one or more points which, together with G_ω, generate the stabilizer of the block, is taken from Beals [2].

Observe that we may apply Proposition 5.2 to calculate the smallest block B containing ω and a given set S in time $O(l\#B\#S)$, where l is the depth of our Schreier tree, and a one-off cost of $O(n)$ for setting up the data structure, assuming that we know the orbits of G_ω.

In general, of course, the set S corresponding to the subgroup J needs have size no more than $\log |J : G_\omega|$, and since all our blocks are to be built up by repeated extensions of smaller blocks with each enlargement being minimal, we will have no difficulty in maintaining this condition.

Lemma 5.5. *Suppose $J \geqslant \langle G_\omega, G_\omega^{x^{-1}}\rangle$. Then $J \cap J^x = G_\omega$ if and only if $\omega^J \cap \omega^{J^x} = \{\omega\}$.*

Proof. The initial hypothesis gives $J \cap J^x \geqslant G_\omega$, and the result is now immediate. □

Since the algorithm arranges that all subgroups J for which we want to know whether $J \cap J^x = G_\omega$ contain $\langle G_\omega, G_\omega^{x^{-1}}\rangle$, this lemma enables us to perform this test efficiently.

The real advantage of the use of blocks to represent subgroups is the ease with which we can find the blocks $\omega^{\bar{J}}$ corresponding to the minimal subgroups $\bar{J}$ containing the subgroup J corresponding to a given block ω^J. The blocks we require correspond to the minimal blocks of the action of G on the block system of translates of the block ω^J, and these can be found by the extension of an algorithm by Schönert and Seress [6]. Their paper gives a deterministic algorithm, running in $O(ln \log n)$ time (and which runs quickly in practice), which, given a Schreier tree of depth l and the orbits of G_ω, will return all the minimal blocks of the input action (of degree n).

We will be calling it with various actions of G on the sets of translates of different blocks B of (G, Ω), so we will need to calculate the action corresponding to the known block B, taking $O(sn)$ time, and then convert our Schreier tree for Ω to one for the induced action, and find the suborbits of the induced action. As we can easily work out the orbits of G_ω

on the translates of B from knowledge of its orbits on Ω, and we know a set S such that $G_{\{B\}} = \langle G_\omega, \{r(\alpha) \mid \alpha \in S\}\rangle$, we can then apply Proposition 5.2 to obtain the suborbits in only $O(ln\#S)$ time, and using a small S of size at most $\log \#B$, this works out as $O(ln \log n)$ time. As conversion of the Schreier tree is straightforward to do in $O(n)$ time, we can set up and make each call to the Schönert–Seress routine in $O(n(s + l \log n))$ time.

Observe that finding a small generating set for each $\omega^{\bar{J}}$ as a block containing ω is easy, given that we know one for ω^J: simply extend the generating set for ω^J by the addition of one point from $\omega^{\bar{J}} \setminus \omega^J$; this works because of the minimality of $\bar{J}$ as an overgroup of J.

Each element $G_\omega g$ of N/G_ω can of course be represented by the point ω^g, and the Schreier tree can be used to calculate the images α^g and $\alpha^{g^{-1}}$ as required. We can also represent the group element x in the solution pair (J, x) by the point ω^x, since whenever (J, x) is a solution pair, (J, x') is an equivalent one, for any x' in $G_\omega x$, by Proposition 3.6.

6. *Implementation and analysis*

We consider the implementation of each line in Algorithms 4.1 and 4.2 in terms of these representations of our data. We assume that we have a Schreier tree of depth l available, and that we know the orbits of G_ω on Ω.

Finding N and N/G_ω represented by the set of points ω^N is straightforward because $\omega^N = \{\alpha \in \Omega \mid \alpha^{G_\omega} = \{\alpha\}\}$, so we take the set of points that are singleton G_ω-orbits; this requires $O(n)$ time.

The set $\mathcal{R}$ will be represented by the set $\{\omega^\Delta \mid \Delta \in \mathcal{R}\}$; that is, take the image of ω under each of the sets in $\mathcal{R}$ (all the sets are unions of cosets of G_ω). Formation of this representation involves starting with the orbits of G_ω and calculating the orbits of N/G_ω in its action on this set of orbits. This is a straightforward orbit calculation, except that the action involved is a little different: for $G_\omega g \in N/G_\omega$ and a G_ω-orbit α^{G_ω} then the image required is the G_ω-orbit containing $\omega^{g^{-1}r(\alpha)g}$; since g is represented by ω^g, this means that three Schreier tree calculations will be necessary to form each image. We also note that we need a generating set for N/G_ω, and that if we use the whole of the group then the calculation will require $O(ln|N : G_\omega|)$ time, whereas if we pre-calculate a subset of ω^N that generates it as a block containing ω and has size $\log |N : G_\omega|$, we can do the whole thing, including the precalculation, in $O(ln \log |N : G_\omega|)$ time. (See Proposition 5.3 for the precalculation.) The set X will appear out of our construction of $\mathcal{R}$, but of course is stored as a set of points (from Ω), not as a set of group elements.

Note that these calculations assume that we represent the orbits of G_ω not just as a set of sets but also as a look-up table, indexed by the elements of Ω, with each entry containing some reference or pointer to the relevant orbit.

Formation of $\langle G_\omega, G_\omega^{x^{-1}}\rangle$ requires a straightforward application of Proposition 5.2. The calculation of the block $B = \omega^{\langle G_\omega, G_\omega^{x^{-1}}\rangle}$ therefore requires $O(ln)$ time.

Recall that we need to know not only the block B, but also a subset S of B of size at most $\log \#B$ that generates B as a block containing ω. This can be calculated either by modifying the process for calculating B, or afterwards, as in Proposition 5.3, in time $O(l\#B \log \#B)$.

By Lemmas 5.4 and 5.5, conjugating and testing whether $\bar{J} \cap \bar{J}^x = G_\omega$ can be done by forming $\omega^{\bar{J}^x}$ and testing whether $\omega^{\bar{J}} \cap \omega^{\bar{J}^x} = \{\omega\}$. Forming $\omega^{\bar{J}^x}$ from $\omega^{\bar{J}}$ requires $O(l|\bar{J} : G_\omega| \log |\bar{J} : G_\omega|)$ time, by Proposition 5.2. Testing whether $\omega^{\bar{J}} \cap \omega^{\bar{J}^x} = \{\omega\}$ can also be done in $O(|\bar{J} : G_\omega| \log |\bar{J} : G_\omega|)$ time.

This means that the first part of Algorithm 4.1, up to the start of the main loop, can be executed in $O(ln^2 \log n)$ time, plus $O(n)$ calls to Add, since $\#X$ is $O(n)$. In particular, this is sufficient to answer the question posed in Specification 1.1.

Theorem 6.1. *Specification 1.1 can be implemented in $O(ln^3 \log n)$ time and $O(sn)$ space, given a Schreier tree of depth l and the orbits of G_ω.*

Proof. It suffices to run Algorithm 4.1 until the first call to Add in which $J > G_\omega$. Since we stop as soon as a solution (J, x) is found with $J > G_\omega$, we only need to implement Add, to handle calls made with $J = G_\omega$. This means that no equivalence checking is needed within Add as equivalent solutions with $J = G_\omega$ will arise from the same N-conjugacy class of double cosets of G_ω and so will be eliminated in the early lines of Algorithm 4.1. Therefore Add can be implemented in constant time. We may need one iteration of the main loop, and therefore one call to the Schönert–Seress routine. We may then have to test whether $\bar{J} \cap \bar{J}^x = G_\omega$ a further $O(n^2)$ times, as there may be up to n values of x and up to $n - 1$ subgroups $\bar{J}$.

The spatial bound is easily seen from the discussion preceding the theorem; the only point worth commenting on is that L requires only $O(n)$ storage space as it has only one entry, which stores a block of size 1 and $O(n)$ values of x (which are of course stored as points ω^x in Ω). □

Peter Neumann has suggested an improvement to this algorithm, which is the subject of Section 8.

Leaving aside the subroutine Add for the moment, the only remaining question about the implementation and complexity of Algorithm 4.1 is how many iterations of the main loop there will be. This is bounded by the number of N-conjugacy classes of subgroups containing G_ω, but this can easily be seen to grow exponentially as a function of n for certain classes of groups (*e.g.*, elementary abelian groups acting regularly). In what follows the number of iterations will be denoted by K; observe that this is bounded linearly by the size of the output. Thus our complexity analyses will be in terms of the sizes of both the input and the output, rather than (as is more usual) just the input. Secondly, observe that in most practical cases K is very small, although experiments have been undertaken with groups with large K, with successful results. Even though a K^2 term appears in the final analysis, this does not appear to be a serious restriction on the practicality of the algorithm.

There are K calls to the Schönert–Seress algorithm, and at most $n + nK$ calls to Add, where the n term arises from the calls before the main loop and the nK from the calls in the main loop, since there are at most n minimal blocks in any action of degree no more than n.

We turn now to the implementation of the subroutine Add, Algorithm 4.2. If size information is calculated and stored with each block as it is constructed, finding i_0 and i_1 requires only $O(\log K)$ time. To find i we need to calculate a lot of conjugates of blocks; however in this case the conjugating elements g normalize G_ω and so the conjugate block $\omega^{J'^g}$ is generated as a block containing ω by the set $\{\omega^{r(\alpha)^g} \mid \alpha \in S\}$ where S generates J' as a block containing ω. Therefore we can test whether $\omega^{J'^g} = \omega^J$, where $g \in N$, by taking a subset S of $\omega^{J'}$ that generates it as a block containing ω, and forming the conjugate set $\{\omega^{g^{-1}r(\alpha)g} \mid \alpha \in S\}$. This set will be a subset of ω^J if and only if $\omega^{J'^g} = \omega^J$, since the two blocks are known to have the same size. Therefore finding i can be done in $O((i_1 - i_0)l|N : G_\omega| \log |J : G_\omega|)$ time if an array indexed by the points of Ω and indicating membership of ω^J is prepared and used. In the theoretical analysis $i_1 - i_0$ will be taken as $O(K)$, but note that it is normally going to be a small fraction of K.

Insertion of a new entry into L requires $O(K)$ time, but note that this happens on at most K of the calls to Add. If L were stored as a linked list, this would take constant time, but finding i_0 and i_1 would be $O(K)$ instead of $O(\log K)$, and this happens every time Add is called. In practice, we could implement L by a technique such as hashing, or as an array of linked lists indexed by the divisors of n, but K is anticipated to be sufficiently small for these techniques to be unnecessary. Anyway, we cannot eliminate the K^2 term arising from the search for i.

Forming $N_{J'}$ is a similar process to that of finding i, and so can be done in $O(l|N : G_\omega| \log |J : G_\omega|)$ time.

Formation of $\mathcal{X}_1$ is similar to that of $\mathcal{R}$ in Algorithm 4.1, except that here we start with the block system of translates of $\omega^{J'}$, then form the orbits of G_ω on this system, and then the orbits of $N_{J'}$. Forming the block system can be done in $O(sn)$ time; although we will need this system many times, we do not store it between calls to Add as this would unacceptably increase the storage requirement of the algorithm. Forming the orbits of G_ω on the block system requires $O(n)$ time, as we know the orbits of G_ω on Ω. As before, the orbits of $N_{J'}$ can now be formed in $O(ln \log \#N_{J'})$ time.

Naturally, T is stored as a subset of Ω rather than a set of elements of G. Using a suitable look-up table constructed whilst $\mathcal{X}_1$ was being formed, we can both form T and extend it as necessary in a total time of no more than $O(n)$. In doing this we will provide a look-up table indexed by the elements of Ω to determine membership of T.

The set X certainly has size at most $O(n)$ (usually much less), and so the loop executes $O(n)$ times. Each execution of the body of the loop (apart from the extensions of T) takes no more than $O(l)$ time for testing the condition in the 'If' statement, plus a constant time for the storage of x^g. Therefore the total time taken to build T and execute the loop is at worst $O(n(s+l \log \#N_{J'}))$, so the total time to execute Add is at worst $O(n(s+l \log \#N_{J'})+Kl|N : G_\omega| \log |J : G_\omega|)$, which is $O(n(s + Kl \log n))$.

We conclude this section by giving the overall complexity of the algorithm in terms of n, s and K.

The first tranche of calls to Add require no more than $O(n^2(s + Kl \log n))$ time (the K here could be taken as $O(n)$ but K can on the whole be expected to be significantly smaller than n) as there are $O(n)$ of them.

The calls to the Schönert–Seress routine require at most $O(Kln \log n)$ time as there are K of them. The calls to Add made from within the main routine require no more than $O(Kn^2(s + Kl \log n))$ time. The remaining parts of the main loop require a total time of at most $O(Kln^3 \log n)$, since in each iteration of the main loop there are up to $O(n^2)$ tests of the form 'is $J' \cap J'^x = G_\omega$?'.

It is easy to see from the preceding discussion that the spatial requirement of the algorithm is dominated by the size of the input, which is $O(sn)$, and the space needed to store L, which is $O(Kn)$, as each entry can take up to $O(n)$ space.

We have proved:

Theorem 6.2. *Algorithms 4.1 and 4.2 can be implemented in $O(Kn^2(s + (K + n)l \log n))$ time, and $O((s + K)n)$ space, provided that the G_ω orbits and a Schreier tree of depth l are known in advance.* □

We can calculate the G_ω-orbits in $O(sln^2)$ time, using the Schreier generators for G_ω, stored as words in the original generators. In practice, a much faster method would be used to approximate the orbits, or we would find the orbits in the process of finding a whole strong generating set if that were needed for other purposes.

https://doi.org/10.1112/S146115700000005X Published online by Cambridge University Press

Forming a Schreier tree of depth $O(n)$ takes just $O(sn)$ time, but more complicated techniques such as cube-doubling (see [1], for example) enable one to construct one of depth $l \leqslant 2\log \#G$ in time $O(n \log \#G(s + \log \#G))$, which for families of so-called 'small base groups' is better asymptotically. In practice, the standard $O(sn)$ algorithm is used with a breadth-first search since the possibility of the worst case arising is small, and the depth is on average much better than $O(n)$.

In practice. The algorithm has been implemented in GAP [7]. Results to date are pleasing: groups of degree up to 4000 are handled efficiently on a P200MMX with 32MB RAM running Linux; the time taken varies considerably, but all the groups tried to date are handled within a quarter of an hour of CPU time (after the calculation of the suborbits), and many much faster. The implementation is to be found in Appendix A.

A comparison was made between two programs implementing different versions of the algorithm. One implemented the full strength of the definition of equivalence, guaranteeing that no two solution pairs that it output were equivalent. The other was a simpler algorithm whose only check to reduce the number of equivalent solutions produced was that it started off with only one value of x from each double coset of G_ω; thereafter, the number of solutions was allowed to grow freely. Not only were there dramatically fewer solutions produced by the more complicated algorithm (so it in fact produced significantly more information, as the complete sets of solutions can be constructed quickly from representatives of each equivalence class of solutions), but in virtually every case where there were solutions to find it was faster by a factor of 2 or more (often very much more than that: in at least one case there was a speed-up by a factor of about 70), even when there were very few solutions for either algorithm to find.

7. *Self-paired and non-self-paired orbitals*

Orbitals fall into two classes: those that contain (β, α) for each pair (α, β) in the orbital, and those that, for each pair (α, β) in the orbital, do not contain the reversed pair (β, α). The former are termed *self-paired*, the latter *non-self-paired*.

Proposition 7.1. *Suppose (J, x) is a solution pair. The corresponding orbital $(J, Jx)^G$ is self-paired if and only if (J, x) is equivalent to a solution pair (J, y) where $y^2 \in G_\omega$. If this is the case then $y \in N_G(G_\omega)$, and, additionally, y can be chosen in Jx.*

Proof. We have $(Jx, J) \in (J, Jx)^G$, so there is $y \in G$ mapping J to Jx and Jx to J by right multiplication. In particular, $Jy = Jx$, so $y \in Jx$; considering the stabilizers of the action on cosets of J, we get that $J^y = J^x$ and $J^{xy} = J$. Thus $G_\omega^y = (J \cap J^x)^y = J \cap J^x = G_\omega$. Also $Jy^2 = J$ and $Jxy^2 = Jx$ so $y^2 \in J \cap J^x = G_\omega$, and (J, y) is a solution pair since $J^y = J^x$; it is equivalent to (J, x) since $y \in Jx$.

Conversely, if (J, y) is a solution where $y^2 \in G_\omega$ then the orbital $(J, Jy)^G$ must be self-paired, since $(J, Jy)^y = (Jy, J)$. Equivalence of solutions respects whether the corresponding orbital is self-paired, as is easily seen from Proposition 3.2. □

Corollary 7.2. *If (J, x) and (J, y) are equivalent solution pairs with $y^2 \in G_\omega$ (so the corresponding orbitals are self-paired) and $G_\omega \leqslant J' \leqslant J$, then (J', y) is a solution pair corresponding to a self-paired orbital; if $\bar{J} \geqslant J$ and one of $(\bar{J}, x)$ and $(\bar{J}, y)$ is a solution pair, then so is the other one; they are equivalent and correspond to a self-paired orbital.*

https://doi.org/10.1112/S146115700000005X Published online by Cambridge University Press

Proof. The first part is immediate from Lemma 2.4 and Proposition 7.1 (using the fact that y normalizes G_ω); the second is immediate from Corollary 3.5 and Proposition 7.1. □

This means that if we only wish to recognise self-paired orbitals, we need only look at pairs (J, x) for which $x \in N_G(G_\omega)$ and $x^2 \in G_\omega$. No solution pair arising 'above' such a pair will ever be non-self-paired, and every self-paired solution arises in this way.

On the other hand, if we only wish to recognise non-self-paired orbitals, we need only consider solution pairs that are non-self-paired, as exploring the possible pairs above a self-paired solution can only ever yield self-paired solutions. We will, however, expect to generate self-paired solutions as we proceed, but they can safely be discarded without losing any non-self-paired solutions. However, to do this we need to be able to identify quickly whether a solution is self-paired.

Lemma 7.3. *Suppose (J, y) is a solution pair with $y^2 \in G_\omega$, so by Proposition 7.1, $y \in N_G(G_\omega)$. If $y' \in G_\omega y$, then $y'^2 \in G_\omega$ and $y' \in N_G(G_\omega)$.*

This means that to decide whether a solution (J, x) is self-paired, we need only look at representatives of the elements of $\omega^{Jx \cap N_G(G_\omega)}$; if any of them have squares in G_ω then (J, x) is self-paired, otherwise it is not self-paired.

8. *Improved algorithm for Specification 1.1*

The algorithm in this section was suggested by Peter Neumann as an improvement to the algorithm described in Theorem 6.1. Its asymptotic complexity is better than that of Theorem 6.1 by a factor of n (the improvement is from $O(ln^3 \log n)$ to $O(ln^2 \log n)$), but its practicality has not been tested.

Theorem 8.1. *Algorithm 8.2 on page 18 fulfils Specification 1.1 and can be implemented to run in $O(ln^2 \log n)$ time and $O(sn)$ space.*

Proof. We first show correctness. The first eleven lines of Algorithm 8.2 are essentially the same as the first 9 lines of Algorithm 4.1. As the reader will recall from the proof of Theorem 6.1, and the discussion preceding it, this part of the algorithm finds a solution (J, x) where x does not normalize G_ω, if one exists. If no such solution exists, then, as in the earlier algorithm, we may restrict our search to pairs (J, x) where J contains G_ω as a maximal subgroup and $x \in X \cap N$ (note that $N = N_G(G_\omega)$ from the first line of the algorithm). The algorithm for Theorem 6.1 checked every one of these pairs; in the present case we analyse the situation a little more closely and eliminate some of the search.

If there is a solution with $J \leqslant N$, then a minimal such solution (J, x) with $J > G_\omega$ will correspond under the natural map to a subgroup of N/G_ω of prime order, which is not normalized by x; moreover, any non-normal subgroup of N/G_ω of prime order must yield a solution in this way, as its intersection with a non-trivial conjugate must be trivial.

If there is no solution (other than G_ω) with $J \leqslant N$, then by Lemma 2.4, any solution (J, x) must have $J \cap N = G_\omega$. If J is minimal with respect to properly containing G_ω, then $J = \langle G_\omega, G_\omega^{y^{-1}} \rangle$ for any $y \in J \setminus G_\omega$, since such y cannot normalize G_ω. So we can restrict our search to groups J of this form; observe that if two elements y are in the same coset of N then they generate the same subgroup $\langle G_\omega, G_\omega^{y^{-1}} \rangle$. Thus the outer loop in the last section of the algorithm (lines 18 to 26) loops over sufficiently many subgroups J to be sure of finding a solution if one exists. The inner loop checks sufficiently many possibilities for x for each subgroup J.

Algorithm 8.2.

```
Form $N := N_G(G_\omega)$;
Form $\mathcal{D}$, the set of double cosets of $G_\omega$ in $G$;
Let $\mathcal{R}$ be a subset of $\mathcal{D}$ containing one double coset from each orbit of $N$ in its action
        on $\mathcal{D}$ by conjugation;
Let $X$ be a subset of $G$ containing one element from each double coset in $\mathcal{R}$;
For $x \in X \setminus N$ do
    Let $J := \langle G_\omega, G_\omega^{x^{-1}} \rangle$;
    If $J \cap J^x = G_\omega$ then
        Output $(J, x)$;
        Exit;
    End if;
End for;
If $N/G_\omega$ contains a non-normal subgroup $Y$ of prime order then
    Let $J$ be the inverse image of $Y$ in $N$;
    Choose $x \in N$ such that $Y^x \neq Y$ in $N/G_\omega$;
    Output $(J, x)$;
    Exit;
Else
    For $y$ in a set of coset representatives for $N$ in $G$ do
        Let $J := \langle G_\omega, G_\omega^{y^{-1}} \rangle$;
        For $x \in X \cap N$ do
            If $J \cap J^x = G_\omega$ then
                Output $(J, x)$;
                Exit;
            End if;
        End for;
    End for;
End if;
Output FALSE;
End.
```

We now consider the complexity of the algorithm. It was shown earlier that the first loop runs in $O(ln^2 \log n)$ time. The group $Q = N/G_\omega$ has size $O(n)$, and so we can find a generating set (stored as elements of the block ω^N) of size at most $\log n$ in $O(ln \log n)$ time (Proposition 5.3). We can find the prime subgroups of Q in $O(ln \log n)$ time (for example using the Schönert–Seress method).

We claim that testing the prime subgroups for normality in Q can be done in $O(ln \log n)$ time. We do this by conjugating a generator of each subgroup by each member of a generating set for Q in turn, and deciding whether the conjugate lies in the subgroup that we started with. The conjugation can be done in $O(ln \log n)$ time as there are $O(n)$ subgroups and so $O(n \log n)$ conjugates to form, each one taking $O(l)$ time (elements of Q being represented as elements of the block ω^N, so group operations with them require a Schreier tree). The membership testing can be done in $O(n \log n)$ time. This is because for a subgroup of size k, the membership testing takes at most $Ck \log n$ time (faster if a binary search is used) for some constant C, independent of the choice of subgroup, as there are at most $\log n$ tests, each taking a time Ck. As prime subgroups of Q intersect trivially, the sum of the sizes k of all the subgroups in question is at most $2n$, so summing over all prime subgroups gives a bound of $Cn\log n$, that is, $O(n\log n)$.

Finally if there are no non-normal prime subgroups of Q, we pass into the nested loops at the end. Each iteration of the inner loop takes $O(ln\log n)$ time. There are $|G:N|$ elements y to consider, and for each one, at most $|N:G_\omega|$ elements x, so the total number of iterations of the inner loop is at most $|G:G_\omega|$. Thus the nested loops run in $O(ln^2\log n)$ time.

The bound on the space used comes from the size of the input and the data structures used; the only tricky issue is the generating set for Q, but this is stored as a subset of the block ω^N, so in fact causes no problems. □

9. *Actions on unordered pairs*

In the remaining sections we shall consider the problem: is there an algorithm which, given a permutation group (G,Ω) as input can tell if there exists an action of G on a set Γ such that Ω is G-isomorphic to an orbit of G on $\Gamma^{\{2\}}$ (the set of unordered pairs of distinct elements of Γ)? Thus, if we replace $\Gamma^{\{2\}}$ by $\Gamma^{(2)}$ in that last sentence, we get the problem of recognition of actions on orbitals that we have up to now been considering. Formally we have two more specifications:

Specification 9.1.

Input *Permutations* $g_1,\dots g_s\in\mathrm{Sym}(\Omega)$ *generating a transitive permutation group G on* Ω.

Output *An action* (G,Γ) *with* $\#\Gamma\leqslant\#\Omega$ *and an orbit* $\{\alpha,\beta\}^G\subseteq\Gamma^{\{2\}}$ (*with* $\alpha\neq\beta$) *in the action of G on unordered pairs of elements of* Γ, *such that* (G,Ω) *is isomorphic to the action of G on* $\{\alpha,\beta\}^G$, *or the information that no such action and orbit exist.*

Specification 9.2.

Input *Permutations* $g_1,\dots g_s\in\mathrm{Sym}(\Omega)$ *generating a transitive permutation group G on* Ω.

Output *The set of all* (*essentially different*) *pairs, where each pair consists of an action* (G,Γ) *with* $\#\Gamma\leqslant\#\Omega$ *and an orbit* $\{\alpha,\beta\}^G$ (*with* $\alpha\neq\beta$) *of G on unordered pairs of elements of* Γ *such that* (G,Ω) *is isomorphic to the action of G on* $(\alpha,\beta)^G$.

In Definition 10.3, we define a formal notion of equivalence of solutions, which enables us to give a precise meaning to the notion of ‘essentially different’ in this specification.

We rule out the pathological case where $\#\Gamma>\#\Omega$, as to recognise this requires different techniques, and it is not thought likely to be of interest. Note that in this case $\#\Gamma=2\#\Omega$ and Ω corresponds to a block system in (G,Γ) with blocks of size 2. The action induced on this block system may not be faithful, and so there may be larger groups than G that have systems of blocks of size 2 that are isomorphic to Ω. However, given (G,Ω) we can find every action (G,Γ) with a system of blocks of size 2 that is isomorphic to Ω, simply by finding all the subgroups of G_ω of index 2. The action on the cosets of any such subgroup clearly has such a block system, and all actions with such block systems arise this way.

Lemma 9.3. *Suppose* (G,Γ) *is a transitive permutation group, and that Ω is an orbit in the action of G on* $\Gamma^{\{2\}}$, *such that* $\#\Omega\geqslant\#\Gamma$. *Then*

https://doi.org/10.1112/S146115700000005X Published online by Cambridge University Press

(i) *for any* $\alpha \in \Gamma$*, there exist* $\beta, \gamma \in \Gamma$ *such that* α, β, γ *are all distinct and such that* $\{\alpha, \beta\}$ *and* $\{\alpha, \gamma\}$ *both lie in* Ω*; and*

(ii) G *acts faithfully on* Ω*.*

Proof. By transitivity, it suffices to prove the first part for just one $\alpha \in \Gamma$. But if each element of Γ appears in no more than one of the pairs in Ω then $\#\Omega \leqslant \#\Gamma/2$, so there must be at least one element α appearing in more than one pair in Ω.

The second part now follows, for suppose $g \in G$ fixes every pair in Ω. Then for every $\alpha \in \Gamma$, there are at least two distinct pairs $\{\alpha, \beta\}$ and $\{\alpha, \gamma\}$ fixed by g, and so g must fix α. The result follows, since G acts faithfully on Γ. □

10. *Solutions and equivalence for the unordered pair problem*

Fix $\omega \in \Omega$.

Definition 10.1. A solution pair for the unordered pair problem for (G, Ω) is a pair (J, x) where $J \leqslant G$ and $x \in G$ such that the setwise stabilizer (in the action of G by right multiplication) of the pair $\{J, Jx\}$ of cosets of J is G_ω.

Proposition 10.2. *Let* (J, x) *be a solution to the unordered pair problem for* (G, Ω)*. Let* H *be the stabilizer in G of the ordered pair* (J, Jx)*, so* $H = J \cap J^x$*. Then either* $H = G_\omega$ *or* $|G_\omega : H| = 2$ *and for all* $y \in G_\omega \setminus H$*, we have* $(J, Jx)^y = (Jx, J)$*.*

Proof. Certainly $H \trianglelefteq G_\omega$ and G_ω/H embeds in the symmetric group on two points. The result follows. □

Solutions where $|G_\omega : H| = 1$ will be said to be of *index 1 type*; similarly those where $|G_\omega : H| = 2$ will be said to be of *index 2 type*.

Definition 10.3. Suppose (J_1, x_1) and (J_2, x_2) are solution pairs for the unordered pair problem for (G, Ω). We say they are equivalent precisely when there exists $g_0 \in G$ such that $J_2 = J_1^{g_0}$ and x_2 lies in either $J_2x_1^{g_0}J_2$ or $J_2(x_1^{-1})^{g_0}J_2$.

Proposition 10.4. *Solution pairs* (J_1, x_1) *and* (J_2, x_2) *for the unordered pair problem for* (G, Ω) *are equivalent if and only if there exists a* G*-isomorphism* $\theta : \cos(G : J_1) \to \cos(G : J_2)$ *such that* $(\{J_1, J_1x_1\}^G)\theta = \{J_2, J_2x_2\}^G$*. If they are equivalent then they have the same index type (as defined above), and if they are equivalent and of index 2 type then there exists* $g_0 \in G$ *such that* $J_2 = J_1^{g_0}$ *and* $x_2 \in J_2x_1^{g_0}J_2$*.*

Proof. Suppose first that θ is as described. Then θ maps some coset J_1g of J_1 to J_2, and as θ is a G-isomorphism, we get $J_1^g = J_2$. Now the other condition on θ gives us that

$$\{J_1, J_1x_1\}\theta = \{J_2g', J_2x_2g'\}$$

for some $g' \in G$, and now either (i) $J_1\theta = J_2g'$ and $J_1x_1\theta = J_2x_2g'$, so $g' \in J_2g^{-1}$ and $J_2g'x_1 = J_2x_2g'$ whence $x_2 \in J_2g^{-1}x_1gJ_2$, or (ii) $J_1\theta = J_2x_2g'$ and $J_1x_1\theta = J_2g'$, so $g' \in J_2g^{-1}x_1$ and $J_2g^{-1} = J_2x_2g'$, whence $x_2 \in J_2g^{-1}x_1^{-1}gJ_2$.

Conversely, if $J_1^{g_0} = J_2$ and $x_2 \in J_2x_1^{g_0}J_2 \cup J_2(x_1^{-1})^{g_0}J_2$ then the map $\theta : J_1x \mapsto J_2g_0^{-1}x$ is a G-isomorphism, which is readily seen to have the desired property.

Now suppose that θ exists as described above. Then the stabilizer of the ordered pair (J_1, J_1x_1) is a conjugate of that of the ordered pair (J_2, J_2x_2), and so they have the same size. Finally suppose these stabilizers have index 2 in G_ω, so there is $y \in G_\omega \setminus (J_2 \cap J_2^{x_2})$. If case (ii) above arises then in fact we have $J_1\theta = J_2yg'$ and $J_1x_1\theta = J_2x_2yg'$ since $J_2y = J_2x_2$ and $J_2x_2y = J_2$, and then $yg' \in J_2g^{-1}$ and $J_2yg'x_1 = J_2x_2yg'$, whence $x_2 \in J_2g^{-1}x_1gJ_2$ as required. □

Note that the stabilizers of the ordered pairs (J_1, J_1x_1) and (J_2, J_2x_2) may be different even if the solutions are equivalent as unordered pair solutions.

The index 1 case is essentially a search for non-self-paired solutions to the ordered pair problem for (G, Ω), except that each solution (J, x) is now considered equivalent to the reverse solution, (J, x^{-1}). Recall that such a definition of equivalence was the subject of Remark 3.3. As checking for this extended form of equivalence is difficult (although analogues do exist for most of the results on equivalence), the simplest way to check for this case is to run a search for non-self-paired orbitals as described in the preceding section, and throw away half the solutions.

We now prove a version of Theorem 3.4 for the unordered pair case, index 2 type.

Theorem 10.5. *Let (J_1, x_1) be an unordered pair solution of index 2 type, let $J_2 \leqslant G$ and $x_2 \in G$ and let $H_i = J_i \cap J_i^{x_i}$ for $i = 1, 2$. Then (J_2, x_2) is an unordered pair solution that is equivalent to (J_1, x_1) if and only if there exists $g \in N_G(G_\omega)$ such that $J_2 = J_1^g$ and $x_2 \in J_2x_1^g$; furthermore if that is the case then $H_2 = H_1^g$.*

Proof. Suppose (J_2, x_2) is an unordered pair solution, equivalent to (J_1, x_1), so $J_2 = J_1^{g_0}$ and $x_2 \in J_2x_1^{g_0}J_2$ for some $g_0 \in G$. Choose $z \in J_2$ such that $x_2 \in J_2x_1^{g_0}z$. Then G_ω is the stabilizer of $\{J_2, J_2x_2\}$, and so of $\{J_2, J_2x_1^{g_0}z\}$ and of $\{J_2, J_2x_1^{g_0}\}^z$. The map θ : $\cos(G : J_1) \to \cos(G : J_2)$ given by $\theta : J_1x \mapsto J_2g_0^{-1}x$ is a G-isomorphism, so G_ω is also the stabilizer of $\{J_1g_0, J_1x_1g_0\}^z$. Thus G_ω is is the stabilizer of both $\{J_1, J_1x_1\}$ and $\{J_1, J_1x_1\}^{g_0z}$, and so is normalized by g where $g = g_0z$. Furthermore, $J_1^g = J_2^z = J_2$ and $J_2x_1^g = J_2z^{-1}x_1^{g_0}z$ which contains x_2 as $z^{-1} \in J_2$. Finally, $H_2 = J_2 \cap J_2^{x_2} = J_2 \cap J_2^{x_1^g} = J_1^g \cap J_1^{x_1g} = H_1^g$.

Conversely suppose $J_2 = J_1^g$ and $x_2 \in J_2x_1^g$, where g normalizes G_ω. It is clear from Definition 10.3 that if (J_2, x_2) is an unordered pair solution then it is equivalent to (J_1, x_1); we show that it is such a solution. Now G_ω is the stabilizer of $\{J_1, J_1x_1\}$ and so of $\{J_1g, J_1x_1g\}$. As before the map $J_1x \mapsto J_2g^{-1}x$ is a G-isomorphism, so G_ω is also the stabilizer of $\{J_2, J_2x_1^g\}$, and as $x_2 \in J_2x_1^g$, it follows that G_ω is the stabilizer of $\{J_2, J_2x_2\}$, as required. □

Corollary 10.6. *If H_1 and H_2 are subgroups of G_ω of index 2 and $H_1^g = H_2$ where $g \in N_G(G_\omega)$, then to every solution (J_1, x_1) with $J_1 \cap J_1^{x_1} = H_1$ there is an equivalent one (J_2, x_2) with $J_2 \cap J_2^{x_2} = H_2$.*

Proof. By Theorem 10.5, we can take $J_2 = J_1^g$ and $x_2 = x_1^g$. □

Proposition 10.7. *Fix $H \leqslant G_\omega$ of index 2. There is a one-to-one correspondence between equivalence classes of solutions (J, x) of the unordered pair problem where $J \cap J^x = H$ and the equivalence classes of solutions (J, x) of the ordered pair problem for the action of G on cosets of H (with ω taken as the trivial coset H), with the added restriction that $x \in G_\omega \setminus H$.*

https://doi.org/10.1112/S146115700000005X Published online by Cambridge University Press

Proof. The correspondence is the obvious one: a solution (J, x) of the ordered pair problem for $\cos(G : H)$ with $x \in G_\omega \setminus H$ has $J \cap J^x = H$ and the stabilizer of $\{J, Jx\}$ is $\langle H, x\rangle = G_\omega$, so (J, x) is a solution of the unordered pair problem with $J \cap J^x = H$. To see that the correspondence between equivalence classes goes the other way, note that if (J, x) is a solution of the unordered pair problem with $J \cap J^x = H$ and $y \in G_\omega \setminus H$ then $(Jy, Jxy) = (Jx, J)$ so $y \in Jx$ and thus (J, y) is equivalent (as unordered solution) to (J, x). Finally note that the correspondence respects equivalence of solutions: this is immediate from Definitions 3.1 and 10.3 and Proposition 10.4. □

It follows that a suitable algorithm to handle the index 2 case is to find the subgroups of G_ω of index 2 and then, for one subgroup H from each $N_G(G_\omega)$-conjugacy class of such subgroups, form the action on cosets of H and run the algorithm described earlier, restricting to the special case where we only consider one value of x, taken from $G_\omega \setminus H$. (By Proposition 3.6, we need consider only one element x from $G_\omega \setminus H$.)

Observe that the algorithm of Section 4 sometimes conjugates a solution by an element of $N_G(H)$ to reduce the number of calls to the Schönert–Seress routine, and (since the conjugating element may not normalize G_ω in the present situation) this can lead to a value of x not from $G_\omega \setminus H$ appearing. This is not a problem as it can be proved that the algorithm is still restricted to ordered pair solutions (J, x) that are equivalent to a solution (J', x') where $x' \in G_\omega \setminus H$, so all that is necessary is to conjugate the solution back to one where x lies in $G_\omega \setminus H$ at the end of the algorithm. Alternatively, one can restrict to conjugating by elements of $N_G(G_\omega) \cap N_G(H)$, but this may lead to extra calls to the Schönert–Seress routine.

11. *Subgroups of index* 2 *in* G_ω

It remains to find the subgroups H that should be tested in this way from amongst the (possibly exponentially many) subgroups of G_ω of index 2.

Assume $\omega = \{\alpha, \beta\}$. Under the condition that $\#\Omega \geqslant \#\Gamma$ that we imposed earlier, we know (by Lemma 9.3) that there is a point $\omega' = \{\alpha, \gamma\} \in \Omega$. Then $G_{\omega,\omega'} = G_{\alpha,\beta,\gamma} \leqslant G_{\alpha,\beta}$. We cannot easily identify a suitable point ω', but nevertheless we see that $G_{\alpha,\beta}$ contains $G_{\omega,\omega'}$ for at least one $\omega' \in \Omega$.

Proposition 11.1. *Suppose G_ω has k orbits on Ω, and $\#\Omega = n$. Then there are at most $n - k$ subgroups H of index 2 in G_ω for which there exists $\omega' \in \Omega$ such that $H \geqslant G_{\omega,\omega'}$.*

Proof. Let $\omega' \in \Omega$. Let A be the intersection of the subgroups of G_ω of index 2 that contain $G_{\omega,\omega'}$. Then $A \trianglelefteq G_\omega$, and the quotient is an elementary abelian 2-group, of size at most $|G_\omega : G_{\omega,\omega'}|$. The number of subgroups of index 2 in a finite elementary abelian 2-group is one less than the size of the group. Also note that the subgroups of index 2 in G_ω containing $G_{\omega''}$ for any ω'' in the same G_ω-orbit as ω' are the same as those containing $G_{\omega'}$, since they are all normal in G_ω. Therefore, for each orbit X of G_ω on Ω, there are at most $\#X - 1$ subgroups of index 2 in G_ω containing the point-wise stabilizer of ω and any point in X. The result follows by summing over all G_ω-orbits. □

Lemma 11.2. *Suppose $\omega'' = \omega'^g$ for some $g \in N = N_G(G_\omega)$. If $G_{\omega,\omega''} \leqslant H < G_\omega$ for some subgroup H of index 2 in G_ω, then $G_{\omega,\omega'} \leqslant H^{g^{-1}} < G_\omega$ and $H^{g^{-1}}$ also has index 2 in G_ω.*

Proof. We have $G_{\omega,\omega''} = G_\omega \cap G^g_{\omega'}$, and since g normalizes G_ω this equals $G^g_\omega \cap G^g_{\omega'}$, and so

$$G_{\omega,\omega''} = G^g_{\omega,\omega'}.$$

The result follows. □

By Corollary 10.6, this means that once we have considered all subgroups of index 2 in G_ω that contain $G_{\omega,\omega'}$, we do not need to consider the subgroups of index 2 that contain $G_{\omega,\omega''}$ for any ω'' in the same N-orbit as ω', as the only solutions they can yield will be equivalent to ones we have already found.

The subgroup A of G_ω in the proof of Proposition 11.1 is the product of G^2_ω (the group generated by the squares of all elements of G_ω, which is also the intersection of the index 2 subgroups of G_ω) and $G_{\omega,\omega'}$.

Lemma 11.3. *The subgroup $A = G_{\omega,\omega'}G^2_\omega$ is equal to B, the normal closure in G_ω of $\langle G_{\omega,\omega'}, \{r(\alpha)^2 \mid \alpha \in \Delta\}\rangle$, where Δ is the G_ω-orbit containing ω', and $r(\alpha)$ is an element of G_ω mapping ω' to α.*

Proof. Clearly $G_{\omega,\omega'} \leqslant B \leqslant A$. Suppose for a contradiction that $B < A$. It follows that there is $g \in G_\omega$ such that $g^2 \notin B$. We can write $g = hr(\alpha) = r(\alpha)h'$ for some $r(\alpha)$ and some $h, h' \in B$ (by normality of B), so $g^2 = hr(\alpha)^2h'$ lies in B if and only if $r(\alpha)^2$ does; this is a contradiction, since B contains $r(\alpha)^2$, and we conclude that $B = A$. □

We construct not A but the block (in the action of G_ω) ω'^A. We first construct the block ω'^H where $H = \langle G_{\omega,\omega'}, \{r(\alpha)^2 \mid \alpha \in \Delta\}\rangle$ using standard techniques, then close it under conjugation by the elements of a transversal for it in G_ω. We do this by constructing the block system of translates of ω'^H under the action of G_ω, and looking for a block in this block system that is not fixed by the action of H; if none exists then H is normal; otherwise, we say $(\omega'^H)^g$ is not fixed by H. That means that $H^{g^{-1}}$ does not fix ω'^H, and if $h \in H$ does not fix $(\omega'^H)^g$ then $ghg^{-1} \in H^G \setminus H$. We find such an element, add it to H, recalculate ω'^H and repeat the whole process until all orbits of H in its action on the translates of ω'^H by elements of G_ω are singletons. Of course, H can only be so extended $\log \#\Delta$ times.

If we know a generating set for $G_{\omega,\omega'}$ in its action on Δ of size $O(\#\Delta)$ (for example through a Sims-style base and strong generating set calculation, and a base-change if necessary) then this process runs in time $O(l(\#\Delta)^2 \log \#\Delta)$ where l is the depth of a Schreier tree, since it can all be done within the action induced by G_ω on Δ. Therefore the total time needed to find ω'^A for each G_ω-orbit Δ (or all those for which it is needed; see Lemma 11.2) is at most $O(ln^2 \log n)$. By [3], the base-change can be done deterministically in $O(n^2)$ time and $O(n^2)$ space (note that we only need a cyclic base-change). Alternatively, if we are working with so-called small-base groups (groups with a base of size $O(\log^c n)$ for some constant c, so $\log \#G$ is $O(\log^{c+1} n)$), the base-change can be executed in deterministic nearly linear time and space using a result stated in [5]; and by [1] a strong generating set of size $O(\log^2 \#G)$ can be found in Monte-Carlo nearly linear time, so the whole procedure runs in nearly linear time and space.

12. *Algorithmic results*

Algorithm 12.1 is designed to test whether (G, Ω) is isomorphic to an orbit on unordered pairs of an action (G, Γ) of G.

https://doi.org/10.1112/S146115700000005X Published online by Cambridge University Press

Algorithm 12.1.

```
1.  Use an adaptation of Algorithms 4.1 and 4.2 to decide whether there is a solution pair
        $(J, x)$ corresponding to a non-self-paired orbital; if so the orbit $\{J, Jx\}^G$ is as
        required;
2.  Set $\mathcal{B} = \emptyset$;
3.  For each $N$-orbit $\Delta$ (where $N = N_G(G_\omega)$) with $\#\Delta > 1$ do
4.      Choose $\omega' \in \Delta$;
5.      Form $\omega'^A$, where $A = G_\omega^2 G_{\omega,\omega'}$; this is a block in the action of $G_\omega$ on $\Delta$;
6.      Add to the set $\mathcal{B}$ those blocks in the action of $G_\omega$ on translates of $\omega'^A$ whose
            stabilizer has index 2 in $G_\omega$;
7.  End for;
8.  Form the blocks in $\mathcal{B}$ into equivalence classes corresponding to the action of $N$ by
        conjugation on the stabilizers of the blocks (note that this will identify any
        pairs of blocks in $\mathcal{B}$ that have the same stabilizer);
9.  For each of these equivalence classes choose a representative $B$ from the class and
        do
10.     Form the action of $G$ on cosets of the stabilizer $H$ of $B$;
11.     Let $x \in G_\omega \setminus H$;
12.     Use Algorithms 4.1 and 4.2 on the action of $G$ on cosets of $H$, restricting the
            search to just this one value of $x$; if $(J, x)$ is a solution pair then the orbit
            $\{J, Jx\}^G$ is as required;
13. End for;
14. End.
```

Correctness follows immediately from the discussion above (using the correspondence between blocks of imprimitivity containing a point α and subgroups containing G_α). Notice that the purpose of forming equivalence classes of the blocks in $\mathcal{B}$ is to ensure that none of the solutions produced are equivalent, and also to reduce the number of calls to the ordered pair algorithm; at the same time we eliminate duplicates, where different blocks in $\mathcal{B}$ have the same stabilizer. This step can be implemented by finding generators for the stabilizer of each block in $\mathcal{B}$, and conjugating them as appropriate; membership testing in the stabilizers can of course be done by observing whether the element leaves the corresponding block invariant. Asymptotically this step is quite expensive: there are $O(n)$ elements of $\mathcal{B}$, each with $O(n)$ generators (depending on how many generators we have for G_ω (from which Schreier generators can be constructed) or for the various $G_{\omega,\omega'}$, which can be extended). There are $O(n)$ conjugates of each stabilizer, and each one has to be checked against $O(n)$ other blocks. Thus the complexity of this stage is $O(n^4)$. However the stage can be omitted, or replaced with a cheaper stage to simply remove duplicate subgroups; this will certainly be possible if only one solution is required, or if it is not important that all solutions produced are pairwise inequivalent. In practice, however, there may well be use for this step as the cost of processing each subgroup of index 2 is high.

Finding subgroups of index 2 from the action on translates of ω'^A should present no difficulty since this action is the regular action of an elementary abelian 2-group, and it is merely a matter of finding a basis and writing down generators for each block of index 2 in this action in terms of the basis. If we can show that this can be done in time $O(k \log k)$

where k is the number of blocks found, then doing this for all the ω'^A previously calculated will take $O(n \log n)$ time.

A basis is simply a generating set without redundancies and has size $\log k$, so finding one in $O(k \log k)$ time presents no problems. To write down generators for each subgroup of index 2 is best done by recursion over the length of the basis. If the $2^r - 1$ subgroups of index 2 of the elementary abelian 2-group H generated by basis elements $b_1, \ldots, b_r$ are $H_1, \ldots, H_{2^r-1}$ then the $2^{r+1} - 1$ subgroups of index 2 of the elementary abelian 2-group $\bar{H}$ generated by basis elements $b_1, \ldots, b_r, b_{r+1}$ are the group $\langle b_1, \ldots, b_r \rangle$, and for each H_i, the groups $\langle H_i, b_{r+1} \rangle$ and $\langle H_i, x_i b_{r+1} \rangle$ where x_i is an element of $H \setminus H_i$. The recursive algorithm should store a suitable x_i with the generators for each H_i. The ith recursive step takes time proportional to $i(2^i - 1)$ since it involves writing down i generators for each of $2^i - 1$ subgroups. Thus the whole process takes $O(k \log k)$ time.

Finding the action on cosets of a subgroup H of index 2 can be done efficiently by forming an action on the Cartesian product $\Omega \times \{0, 1\}$; if $g \in G$ and $(\alpha, z) \in \Omega \times \{0, 1\}$ then $(\alpha, z)^g = (\alpha^g, z')$ where $z' = z$ if $B^{r(\alpha) g r(\alpha^g)^{-1}} = B$ and $z' = 1 - z$ otherwise. Here B is the block (under the action of G_ω) whose stabilizer is the desired subgroup H of index 2. This will take $O(sln)$ time for each action formed.

It remains to consider the complexity of each call to the ordered pair algorithm. We consider the version of this algorithm in which the normalizer N of the stabilizer H of one point in the action on which the procedure is called is replaced by its intersection with the normalizer of G_ω; by Theorem 10.5 this leads to the correct notion of equivalence (given that H is fixed when we call this algorithm). In this version, if a solution (J, x) has $x \in G_\omega \setminus H$, then no conjugate (J^g, x^g) by an element $g \in N$ of this solution has $x^g \notin G_\omega \setminus H$, as that coset is fixed under conjugation by N. That means that every entry into L has the same value of x, and there is no need at all to build the set T in Algorithm 4.2. Thus the complexity of the call to Algorithm 4.1 is reduced to $O(n^2 K_1^2 l \log n)$, where K_1 is the length of the output of that call to Algorithm 4.1. The time needed for all the calls to Algorithm 4.1 in the ‘index 2’ part of the algorithm is therefore at most $O(n^2 K^2 l \log n)$, where K is now the size of the output of Algorithm 12.1.

The spatial requirements of this process are dominated by the input and the requirements of the ordered pair algorithm, except possibly for the demands of whatever algorithm is used to perform the base-change. To enable us to give a precise result, we shall assume that the cyclic base-change algorithm of [**3**] is used, with a space requirement of $O(n^2)$.

Theorem 12.2. *Specification 9.2 can be implemented by an algorithm that runs in time at most $O(n^4 + Kn^2(s + (K + n)l \log n))$ time, if a base and strong generating set and a Schreier tree of depth l are known in advance. Here K is the number of distinct actions returned, and s is the original number of generators given for (G, Ω). The algorithm uses $O(n(n + s + K))$ space.*

Proof. By the preceding discussion and Theorem 6.2. □

We turn now to the situation of Specification 9.1, where only one solution is required.

Theorem 12.3. *Specification 9.1 can be implemented by an algorithm that runs in time at most $O(ln^3 \log n)$ time and $O(n(n + s))$ space, if a base and strong generating set and a Schreier tree of depth l are known in advance.*

Proof. There are $O(n)$ calls to the algorithm of Theorem 6.1, corresponding to the $O(n)$ relevant subgroups of G_ω of index 2. However each call requires only $O(ln^2 \log n)$ time

https://doi.org/10.1112/S146115700000005X Published online by Cambridge University Press

as there is only one value of x to consider. The first call, where a non-self-paired solution is sought, also takes only $O(ln^2 \log n)$ time, as there is no need to enter the main loop of Algorithm 4.1 to find a smallest non-self-paired solution. □

In practice. This algorithm has also been implemented, and again, practical results are encouraging, handling groups of a similar size as the implementation of the algorithm for recognising actions on orbitals, and in similar lengths of time. The GAP implementations are available in Appendix A.

Acknowledgements. I am grateful to my supervisor, Peter Neumann, for suggesting that I study algorithms in this area, for much helpful advice whilst I was working on this paper, and for suggesting the algorithm of Section 8 whilst reading an early draft. I am grateful to Robert Beals for a helpful discussion, out of which many of the ideas in this paper developed.

My doctoral research was supported by the EPSRC.

Appendix A. GAP *code for implementing the algorithm*

This appendix is available to subscribers to the journal at:
http://www.lms.ac.uk/jcm/2/lms98006/appendixa/.

References

1. L. Babai, G. Cooperman, L. Finkelstein and Á. Seress, ‘Nearly linear time algorithms for permutation groups with a small base’, Proceedings 1991 ACM International Symposium on Symbolic and Algebraic Computation (1991) 200–209. 16, 23

2. R. Beals, ‘Computing blocks of imprimitivity for small-base groups in nearly linear time’, *Groups and Computation,* DIMACS Ser. Discrete Math. Theoret. Comput. Sci. 11 (ed. L. Finkelstein and W. M. Kantor, Amer. Math. Soc., Providence, RI, 1993) 17–26. 12

3. Cynthia A. Brown, Larry Finkelstein and Paul W. Purdom, ‘A new base change algorithm for permutation groups’, *SIAM J. Comput.* 18 (1989) 1037–1047. 23, 25

4. Gregory Butler, *Fundamental algorithms for permutation groups*, Lecture Notes in Computer Science 559 (Springer-Verlag, 1991). 10

5. Gene Cooperman and Larry Finkelstein, ‘Random algorithms for permutation groups’, *CWI Quarterly* 5 (1992) 107–125. 23

6. Martin Schönert and Ákos Seress, ‘Finding blocks of imprimitivity in small-base groups in nearly linear time’, Proceedings 1994 ACM-SIGSAM International Symposium on Symbolic and Algebraic Computation (1994) 154–157. 10, 12

7. Martin Schönert *et al.*, GAP — *Groups, algorithms and programming* (Lehrstuhl D für Mathematik, Rheinisch Westfälische Technische Hochschule, Aachen, Germany, 1994). 16

8. Graham Sharp, ‘Algorithmic recognition of actions of 2-homogeneous groups on pairs’, *LMS J. Comput. Math.* 1 (1998) 109–147.
http://www.lms.ac.uk/jcm/1/lms97008/. 2, 3, 3, 3, 3, 3, 3, 3, 3

https://doi.org/10.1112/S146115700000005X Published online by Cambridge University Press

Graham R. Sharp GRSharp@smithgroup.co.uk

The Queen's College
Oxford
OX1 4AW

Current address:
31 Oaklands
Haslemere
GU27 3RD

https://doi.org/10.1112/S146115700000005X Published online by Cambridge University Press