Computing subgroups of bounded index in a finite group

John J. Cannon, Derek F. Holt, Michael Slattery and Allan K. Steel

Abstract

We describe a practical algorithm for computing representatives of the conjugacy classes of subgroups up to a given index in a finite group. This algorithm has been implemented in Magma, and we present some performance statistics.

1 Introduction

In this paper, we describe a method for computing representatives of the conjugacy classes of subgroups up to a given index in a finite group. Corresponding algorithms for finitely presented groups have become generally known as low index subgroups algorithms, and so we shall refer to our method as a low index subgroups algorithm for finite groups.

It has a number of important applications. For certain problems it is necessary to determine some (or all) of the conjugacy classes of subgroups of a finite group $G$ having index less than or equal to a given bound $B$, where $B$ is small compared to the order of $G$. The method we describe in this paper can often determine all such subgroups much more quickly than the time needed to enumerate all the conjugacy classes of subgroups of $G$. In particular, it is often the case that our algorithm can determine such subgroups in situations where it is not practical to enumerate all the subgroups of $G$.

One is sometimes in the position of knowing that a given group has a particular subgroup of some known index, and then the low index subgroups algorithm can enable one to find it rapidly.

For example, an important problem in inverse Galois theory is to realize each of the low degree transitive groups as the Galois group of some polynomial defined over the rational field $Q$. One technique for doing this involves constructing a given transitive group of degree $n$ as some kind of product (for example, as a wreath product), and then finding a subgroup of index $n$ that gives the original permutation action. This method is described in [12], where it is applied for $n \leq 15$, and the authors of [12] have found our low index subgroups algorithm to be an essential tool in extending their results to groups of degree $n$ where $16 \leq n \leq 23$. 
Another application occurs in the explicit calculation of the irreducible representations of a finite group over a field. An important method for producing modules involves inducing low dimensional representations of subgroups of modest index. The irreducible constituents are then computed using the ‘MeatAxe’ algorithm.

Here is an outline of the method. We first find the largest solvable normal subgroup $L$ of $G$. If $L$ is a proper subgroup of $G$ (that is, if $G$ itself is not solvable), then we find subgroups up to the required index in $G/L$. If $G/L$ is small (up to order 100000), then we do this by picking out the required subgroups from the list of all subgroups of $G/L$. Otherwise, we recursively call the maximal subgroups routine, as described in [3]. In either case, we make use of databases containing information about the subgroups of almost simple groups of small order. If $G/L$ has simple composition factors that are not in the database, then we use a slower method, which starts by solving the problem in the point stabilizer of a permutation representation of $G/L$. We are grateful to a referee for some helpful suggestions to improve this part of the algorithm; these ideas have been incorporated into Subsection 2.1.

In all cases, we find a series

$$1 = N_0 \triangleleft N_1 \triangleleft \cdots \triangleleft N_r = L \triangleleft G$$

of normal subgroups $N_i$ of $G$ with each $N_i/N_{i-1}$ an elementary abelian chief factor of $G$, and solve the problem successively in $G/N_i$ for $i = r-1, \ldots, 0$ by lifting the solution from the previous quotient. The method employed for the lifting process is the same as that used in the algorithm for computing all subgroups of a group, as described in [2] for permutation groups, and in [11] for solvable groups, but with appropriate modifications to ensure that we keep subgroups only up to the specified maximal index. Methods for solving this problem in general (finite or infinite) polycyclic groups are also described in Section 6 of [6].

Our low index subgroups algorithm has been implemented in MAGMA for finite permutation and matrix groups by the second author, and for finite solvable groups defined by a power–conjugate presentation (which we shall refer to as PC-groups from now on) by the third author.

An additional tool that we require is an algorithm for computing submodules of a $KG$-module up to a given codimension, where $K$ is a field of prime order, and $G$ is a finite group. This was devised and implemented by the fourth author, and is described in Section 3 below.

2 Subgroups of bounded index in a finite group

Let $G$ be a finite group given as a permutation group or, when $G$ is solvable, as a PC-group, and suppose that we are given a positive integer $n$, and we wish to find representatives of the conjugacy classes of subgroups of $G$ having index at most $n$.

As in [2] and [3], we start by computing the largest solvable normal subgroup $L$ of $G$, and normal subgroups $N_i$ ($0 \leq i \leq r$) of $G$ as described in the introduction.
When $G/L$ is nontrivial we can use methods described in [5] or [7] to find a faithful permutation representation of $G/L$ of the same degree as that of $G$. We first solve the problem in $G/L$, which has trivial Fitting subgroup. In the solvable case, we have $L = G$, so the problem is trivial in $G/L$.

In either case, we perform a lifting process to solve the problem successively in $G/N_i$ for $i = r - 1, \ldots, 0$. When $G$ is given as a permutation group, it is not necessary to form explicit permutation representations of the quotient groups $G/N_i$; we can perform all of the necessary computations in $G$ and in the induced $G$-modules $N_{i-1}/N_i$. The methods for $G/L$ and for the lifting processes are described in the following two subsections. In the final subsection, we briefly consider the case when we have additional restrictions on the subgroups that we are seeking.

2.1 The trivial Fitting group case

In this section, we assume that the permutation group $G$ has trivial Fitting subgroup. The methods that we employ in this case are highly dependent on the availability of pre-computed stored information about the (maximal) subgroups of the simple composition factors of $G$. We shall describe our current implementation in MAGMA at this point.

There are two relevant databases in which the pre-computed information is stored. The first contains representatives of all conjugacy classes of all subgroups of all isomorphism classes of groups with trivial Fitting subgroup up to order 100000. So if $|G| \leq 100000$, then we simply set up an isomorphism between $G$ and the group in the database as described in [2], and then extract the required answer from the list of all subgroups.

The second database contains information on the maximal subgroups of all almost simple groups up to order 16000000, various individual larger groups such as $M_{24}$, the alternating and symmetric groups up to degree 1000, and all groups in various low-dimensional families of groups, currently $\text{PSL}(2, q)$ and $\text{PSL}(3, q)$ for all $q$, and $\text{PSU}(3, p)$, $\text{PSp}(4, p)$, $\text{PSL}(4, p)$, $\text{PSL}(5, p)$, for prime $p$. (New families of groups are being regularly added to this database.) As described in [3], provided that all of the nonabelian composition factors of $G$ are isomorphic to simple groups that are in this database, then we can compute representatives of the classes of maximal subgroups of $G$.

If this is the case, then we can find the maximal subgroups $H_1, \ldots, H_r$ of $G$ up to index $n$. We then consider each $H_i$ in turn. If $|G : H_i| \leq n/2$, then we apply the low index subgroups algorithm recursively to find representatives $L$ of the conjugacy classes of the subgroups of $H_i$ of index up to $n/|G : H_i|$. We then test each such $L$ for conjugacy in $G$ with all of the subgroups currently in the list $H_1, \ldots, H_r$ and, if it is not conjugate to any of them, then we append $L$ to the list. It is clear that this process will generate the required list of subgroups.

For the lifting process, described in the next subsection, we require presentations of each of the subgroups that we have found so far. Such presentations are included
in the first database for all of the subgroups of all of the groups there. Otherwise, our subgroups will arise as maximal subgroups of some overgroup, and the algorithm described in [3] includes an option to compute presentations of subgroups as they are found.

Suppose then that our group $G$ is too large to be in the first database, and that not all nonabelian composition factors of $G$ are in the second database. Let $\Omega$ be the set on which $G$ acts as a permutation group. We choose a point $1 \in \Omega$ such that $H := G_1$ is a proper subgroup of $G$, and apply the low index subgroups algorithm recursively to $H$. Of course, if the composition factors of $H$ are not in the database either, then this will lead to a further recursive call. Since these recursive calls are expensive, we wish to avoid them wherever possible, so our strategy is to choose the point $1$ such that the index $|G : H|$ is as large as possible.

We collect the subgroups of $G$ of index up to $n$ in a list $L$, which initially will just contain $G$. The recursive call to $H$ will return a set of conjugacy class representatives of subgroups $L$ of $H$ with $|H : L| \leq n$. We consider each such $L$ in turn, and find those subgroups $K$ of $G$ for which $K \cap H = L$. For each such $K$ found with $|G : K| \leq n$, we test it for conjugacy in $G$ with subgroups of $G$ that are already in $L$ and, if it is not conjugate to any of these, then we append it to $L$. We also have to compute a presentation of each such subgroup $K$, which we do using the default algorithm to find a presentation on a set of strong generators.

To find the subgroups $K$ for a fixed $L \subseteq H$, we can start with $K = L$ if $|G : K| \leq n$. Next, we find those subgroups $K$ of the form $\langle L, g \rangle$ for some $g \in G$. We shall discuss below the question of which $g \in G$ we need to try. We then consider all pairs $\{K_1, K_2\}$ of such subgroups that we have found already, and check whether $K := \langle K_1, K_2 \rangle$ is a larger subgroup with $K \cap H = L$. Typically, the majority of the subgroups $K$ that we find will have index more than $n$ in $G$, but we have to keep them in the first instance, because two of them may combine to generate a subgroup with index at most $n$. We note also that, if $K = \langle K_1, K_2 \rangle$ with $K \cap H = L$, then $|1^{K_1}|$ and $|1^{K_2}|$ both divide $|1^K|$, and $1^K$ must be a union of orbits of conjugacy classes of $H$. These conditions can often be used to rule out the pair $\{K_1, K_2\}$ in advance, and may sometimes enable us to exclude $K_1$ completely as a potential subgroup of a larger group $K$.

We often have to test a large number of subgroups $K$ for the property $K \cap H = L$, where very few of the subgroups considered actually satisfy it. We use a quick negative test, which usually reveals very quickly when $K \cap H$ is larger than $L$. To do this, we first find the orbit of 1 under $K$ and, for each $x$ in this orbit, we find a $k_x \in K$ with $1^{k_x} = x$. Now we choose a small number (10, for example) of ‘cheap’ random elements of $K$, selected as short words in the generators of $K$. For each such random element $k$, we have $k_x k^{-1} \in H$, where $1^k = x$, and if $k_x k^{-1} \notin L$, then we know that $K \cap H \neq L$. If the quick negative test fails, then we use the standard deterministic test of computing a set of Schreier generators of $K_1$ and testing each of these for membership of $L$.

Finally, we discuss for which $g \in G$ we must consider the subgroup $K = \langle L, g \rangle$. 4
First we calculate the orbits of $L$ on $\Omega \setminus \{1\}$. If $x, y$ are in the same orbit, and $1^y = y$, then we have $g = g' l$ for some $l \in L$ with $1^g = x$, and clearly $\langle L, g \rangle = \langle L, g' \rangle$. So, we find a set $X$ of orbit representatives of $L$ on $\Omega \setminus \{1\}$, and for each $x \in X$, we find a $g_x \in G$ with $1^g_x = x$. Then we only need to consider $g$ of the form $h g_x$ for $h \in H$. We still would like to reduce the number of $h \in H$ that we need to try.

If we let $h_1, \ldots, h_r$ be a right transversal of $L$ in $H$, then we only need to try $g = h_i g_x$ for $1 \leq i \leq r$. We can still do slightly better than this. For each $x \in X$, we put $J_x := H \cap g_x L g_x^{-1}$. Then for $j \in J_x$ and $h \in H$, we have $\langle L, h j g_x \rangle = \langle L, h g_x \rangle$, and so, if we consider the right action by multiplication of $H$ on the cosets $L h_1, \ldots, L h_r$, then we only need to try those $g = h_i g_x$ for which $L h_i$ is an orbit representative of $J_x$ in this action. The intersections $J_x$ can be computed quickly, since $g_x^{-1} J_x g_x$ is the stabilizer in $L$ of the point $x \in \Omega$.

See Section 4 below for some examples of the numbers of elements and subgroups involved in a specific example.

### 2.2 Lifting subgroups to the next layer

The lifting problem can be summarized as follows. We have normal subgroups $N$ and $M$ of $G$ with $N < M$ and $M/N$ an elementary abelian $p$-group for some prime $p$. We have already found class representatives of the subgroups of $G/M$ of index up to $n$, and we now wish to find the corresponding subgroups for $G/N$. This is exactly the same situation as is described in Section 4 of [2], except that there we were trying to find all subgroups of $G$.

As in [2], our strategy is to consider each of the known subgroups $S/M$ of $G/M$ in turn, and to find those subgroups $T/N$ of $G/N$ with $|G : T| \leq n$ and $T M = S$. To do this, we first find the possible intersections $T \cap M$. In [2], since we were considering all subgroups $S/M$ of $G/M$, including the trivial subgroup, we needed to consider all subgroups of $M/N$ as potential intersections, and we computed the conjugation action of $G/M$ on the set of all such subgroups. This severely limits the size of the layer $M/N$ that we can handle, and we do not wish to limit it so severely here, so we use the fact that $M/N$ is a module for $S$ over the field of order $p$, and that $(T \cap M)/N$ is a submodule of index at most $n/|G : S|$ in $M/N$. So we need to find all submodules of a module for a group algebra over a prime field up to a prescribed codimension. The solution to this problem is described in Section 3 below. Once we have done this, the remainder of the algorithm is exactly as described in Section 4 of [2].

### 2.3 Additional restrictions

Sometimes we wish to find subgroups up to a given index that satisfy some additional property. In many situations we can achieve this by choosing only subgroups with the required property at each stage during the lifting process. To date, we have implemented these ideas for PC-groups only.
For example, if we wish to find the normal subgroups of bounded index, then we can choose just the normal subgroups at each stage, and there are straightforward methods of doing this that avoid many of the complications of the general case; we omit the details. We remark also that methods for finding all normal subgroups of a finite group have been described by Hulpke [10] for the general case and by Höfling [9] for PC-groups.

Other useful additional restrictions involve the primes or primes powers that may divide either the order or the index of the subgroups that we are seeking. Some timings for examples of PC-groups using these restrictions are included in Section 4.

3 Submodules of bounded codimension

In this section, let $M$ be a $d$-dimensional $A$-module, for a matrix algebra $A$ over the finite field $F$.

In [13], an algorithm is described that uses the condensation technique to compute the complete lattice of submodules of $M$, and an implementation of this algorithm is available. It seems likely that this method could be adapted to produce a fast method of computing just those submodules of bounded dimension or codimension. Since such an adaptation is not immediately available, however, we shall instead describe an alternative and simpler method for solving this problem. In any case, the modules encountered in the low index subgroups algorithm generally have sufficiently low dimension to render the computation of their submodules of bounded index very fast, so the method described here is certainly satisfactory from that viewpoint.

First we describe an algorithm $\text{ISO\textsc{Submodules}}(M,C)$ which computes all submodules of $M$ isomorphic to an irreducible constituent $C$ of $M$. This is done by the following extension of the algorithm for isomorphism testing given in [8, Sec. 4], and we refer the reader to that source for notation. First we compute the nullspace $N_M$ of the element in $A$ which corresponds to the nullspace $N_C$ used to prove $C$ irreducible.

Suppose $N_M$ has dimension $k$ and basis $n_1, \ldots, n_k$, and let $c$ be a non-zero vector in $N_C$. We then calculate the submodule $L$ spanned by $(c, n_1, \ldots, n_k)$ in the direct sum $C \oplus M^k$, by ‘spinning’ the vector $c$ in $C$ and performing the parallel operations on each of the $n_i$ in $M$. Each time we encounter a vector in $L$ of the form $(0, r_1, \ldots, r_k)$, we store the relation $\sum_{i=1}^{k} \alpha_i r_i = 0$, for the unknowns $\alpha_1, \ldots, \alpha_k$. When all such relations are collected, we solve this system of linear equations for the $\alpha_i$, and for each such solution, the map $v \mapsto \sum_{i=1}^{k} \alpha_i s_i$, where $(v, s_1, \ldots, s_k) \in L$, gives a homomorphism from $C$ into $M$ (which is well-defined because of the above relations). We compute a basis $H$ for all these homomorphisms, and then collect the elements of $H$ into equivalence classes, where two homomorphisms $h_1$ and $h_2$ are equivalent if $h_1 = e \cdot h_2$ for any $e$ in the centralizing field of $C$ (which is easily computed via [8, Sec. 3]). We return the set of the images of a set of class representatives, thus yielding the submodules of $M$ isomorphic to $C$, without repetition.
We next give an algorithm \textsc{BoundedDimSubmodules}(M,D) which returns the set $S$ of all submodules of $M$ whose dimension is at most a given bound $D$.

Set $S = \{\text{the zero submodule of } M\}$ and set $\text{Processed} = \emptyset$.

While $S \setminus \text{Processed} \neq \emptyset$ do the following:

1. Choose $S \in S \setminus \text{Processed}$ and insert $S$ into $\text{Processed}$.
2. Let $Q = M/S$ with $f : M \to Q$ the natural epimorphism.
3. For each constituent $C$ of $M$ do:
   (i) If $\text{Dim}(S) + \text{Dim}(C) > D$, then skip this $C$.
   (ii) For each $T \in \text{ISOSUBMODULES}(M,C)$, insert $U = f^{-1}(T)$ into $S$.
      (Note that $U$ is a submodule of $M$, containing $S$, with $U/S \cong C$.)

Return $S$.

To see correctness, note that for any submodule $S$ already constructed, all submodules $T_M$ of $M$ containing $S$, having dimension at most $B$, and such that $T_M/S$ is isomorphic to an irreducible constituent $C$ of $M$, will be included in $S$, and by induction, all submodules of $M$ with dimension at most $D$ will be included in $S$.

The implementation (which only takes 30 lines in the MAGMA language) includes the following extra optimisations. For each submodule $S$, it is trivial to keep track of its constituents (with corresponding multiplicities). Thus in Step 3, if the multiplicity of $C$ in $S$ is already the full multiplicity of $C$ in $M$, then we can also skip $C$ for this $S$, since there can be no submodules of $Q = M/S$ isomorphic to $C$. Also, we need not compute the full quotient $M/S$ from scratch each time, because when we form a new submodule $U$, the quotient $M/U$ is isomorphic to $Q/T$, which can be stored. The set $S$ of submodules is efficiently represented by a hash table.

Finally, to compute all submodules of $M$ with maximal codimension $B$ we simply apply \textsc{BoundedDimSubmodules} to the dual $M'$, with the maximal dimension bound $D$ taken to be $B$, and then take the duals of these submodules of $M'$.

Note that these algorithms do not assume that $M$ is an $FG$-module, but may be an $A$-module for any matrix algebra $A$ over a finite field. Also, the dimension and codimension bounds are of course optional; ignoring these, we have given a complete algorithm to construct all submodules of $M$, although the alternative method described in [13] may perform better on some types of examples, particularly those of large dimension in which there is a large number of submodules.

4 Timings

The three tables give run-times for the algorithms as invoked by the MAGMA functions \textsc{LowIndexSubgroups} for permutation groups (Table 1) and \textsc{Subgroups} for PC-groups (Tables 2 and 3). These were all run on 750 MHz SunBlade 1000 with 4 GB of RAM.
In the first table, the notation used for group structure is that of the ATLAS [4]. Note that, at the time that these examples were run, the Higman-Sims group HS and \( L_4(3) \) were in the almost simple groups database, but \( L_4(5) \) was not. The example \( L_5(5) \) required two recursive calls, the first to its point stabilizer, which has structure \( 5^4.4.L_4(5).4 \), and the second to the point stabilizer of its radical quotient \( L_4(5).4 \).

In the case of \( L_4(5) \), the point stabilizer \( H \) is an extension of an elementary abelian group of order \( 5^3 \) by \( L_3(5) \). Using the database to find maximal subgroups of \( L_3(5) \) and computing maximal subgroups repeatedly, the program found 85 classes of subgroups \( L_3(5) \) of index up to 5000. This took about 4 seconds. Lifting these subgroups through the layer of order \( 5^3 \) took a further 14 seconds and resulted in a total of 133 classes of subgroups of index up to 5000 in \( H \). The remainder of the time (that is, about 152 seconds) went into extending these subgroups to subgroups of \( G \). For the vast majority of the subgroups \( L \) of \( H \), the number of \( K \) found was very small. For example, there was a subgroup \( L \) of index 620 in \( H \) for which 12 elements \( g \) needed to be considered as candidates for \( K = \langle L, g \rangle \), and only one such yielded \( K \) with \( K \cap H = L \), and that had index 16120, which is too large. The effort was dominated by about six bad cases. For example, there was a subgroup \( L \) of index 3100 in \( H \), for which 285 elements \( g \) had to be tried, resulting in a total of about 330 subgroups \( K \) with \( K \cap H = L \), but all of them had index larger than 5000.

It is a frustrating feature of the algorithm that it does not seem to be possible to avoid the consideration of large numbers of subgroups having index that is much too large, and which do not ultimately yield any results! But the time taken to compute all subgroups of the first four groups in Table 1 was about 5 times longer than the times for those of bounded index and, for the remaining examples, it was not possible to find all subgroups within a reasonable amount of time.

For the PC-group algorithm, we give sample timings using some of the groups in S. Glasby’s library of solvable groups (‘solgps’). In particular, \( G_5 \) denotes a group of order \( 2^4 \cdot 3^3 \cdot 5^9 \) which is \( \text{GL}(2,3).3^2.5^9 \). The group \( G_{10} \) is the Borel subgroup of \( \text{GL}(4,8) \) of order \( 2^{18} \cdot 7^4 \) and \( G_8 \) is \( F_3 : \text{Aut}(F) \) where \( F = \text{GF}(3^{10}) \). This group has order \( 2^4 \cdot 5 \cdot 11^2 \cdot 61 \cdot 3^{10} \). The group \( B_{26} \) is the Burnside group with 2 generators and exponent 6 of order \( 2^{28} \cdot 3^{25} \). Table 2 gives run times for subgroups of index less than a specified limit or dividing a specified value. Table 3 gives times for computing the normal subgroups of index less than a specified limit.

The major limiting factors are the actual number of subgroups satisfying the specified limits and the size of chief factors within the group. A large chief factor, located close to the top of the group, can cause the algorithm to attempt to enumerate all submodules (or a large portion) of a large module. For instance, there are \( 317,886,556 \) subspaces of codimension two in a vector space of order \( 5^8 \), thus any computation which involves all of these subspaces (which are submodules for a trivial action) is unlikely to complete. One of the uses of the index dividing condition is to avoid large chief factors if possible. It should also be noted that the position of
<table>
<thead>
<tr>
<th>$G$</th>
<th>Degree</th>
<th>Order</th>
<th>$n$</th>
<th>Subgrp classes</th>
<th>Time (secs)</th>
</tr>
</thead>
<tbody>
<tr>
<td>HS</td>
<td>100</td>
<td>$4.4 \times 10^4$</td>
<td>5000</td>
<td>12</td>
<td>2.4</td>
</tr>
<tr>
<td>HS</td>
<td>100</td>
<td>$4.4 \times 10^7$</td>
<td>20000</td>
<td>22</td>
<td>3.4</td>
</tr>
<tr>
<td>$L_4(3)$</td>
<td>40</td>
<td>$6.1 \times 10^6$</td>
<td>5000</td>
<td>99</td>
<td>9.0</td>
</tr>
<tr>
<td>$L_4(5)$</td>
<td>156</td>
<td>$7.2 \times 10^9$</td>
<td>5000</td>
<td>12</td>
<td>170.0</td>
</tr>
<tr>
<td>$L_5(5)$</td>
<td>781</td>
<td>$5.7 \times 10^{16}$</td>
<td>1000</td>
<td>3</td>
<td>392.7</td>
</tr>
<tr>
<td>$L_5(5)$</td>
<td>781</td>
<td>$5.7 \times 10^{16}$</td>
<td>5000</td>
<td>7</td>
<td>2788.0</td>
</tr>
<tr>
<td>$C_2 \wr L_3(31)$</td>
<td>64</td>
<td>$6.4 \times 10^{13}$</td>
<td>500</td>
<td>27</td>
<td>99.8</td>
</tr>
<tr>
<td>$C_2 \wr L_3(31)$</td>
<td>64</td>
<td>$6.4 \times 10^{13}$</td>
<td>1000</td>
<td>49</td>
<td>114.1</td>
</tr>
<tr>
<td>$S_3 \wr L_3(31)$</td>
<td>96</td>
<td>$1.1 \times 10^{29}$</td>
<td>500</td>
<td>36</td>
<td>2297.7</td>
</tr>
<tr>
<td>$C_2 \wr Sz(8)$</td>
<td>130</td>
<td>$1.1 \times 10^{24}$</td>
<td>500</td>
<td>8</td>
<td>40.9</td>
</tr>
<tr>
<td>$C_2 \wr Sz(8)$</td>
<td>130</td>
<td>$1.1 \times 10^{24}$</td>
<td>1000</td>
<td>17</td>
<td>91.7</td>
</tr>
<tr>
<td>$S_3 \wr Sz(8)$</td>
<td>195</td>
<td>$1.1 \times 10^{55}$</td>
<td>500</td>
<td>12</td>
<td>1574.1</td>
</tr>
</tbody>
</table>

Table 1: Timings for permutation groups

<table>
<thead>
<tr>
<th>$G$</th>
<th>Order</th>
<th>$\leq n$</th>
<th>dividing $n$</th>
<th>Subgrp classes</th>
<th>Time (secs)</th>
</tr>
</thead>
<tbody>
<tr>
<td>G5</td>
<td>$8.4 \times 10^8$</td>
<td>100</td>
<td>2160</td>
<td>71</td>
<td>2.3</td>
</tr>
<tr>
<td>G5</td>
<td>$8.4 \times 10^8$</td>
<td>2160</td>
<td></td>
<td>4325</td>
<td>192</td>
</tr>
<tr>
<td>G5</td>
<td>$8.4 \times 10^8$</td>
<td>2160</td>
<td>2160</td>
<td>3071</td>
<td>172</td>
</tr>
<tr>
<td>G10</td>
<td>$6.3 \times 10^8$</td>
<td>$2^9$</td>
<td></td>
<td>22</td>
<td>2.4</td>
</tr>
<tr>
<td>G10</td>
<td>$6.3 \times 10^8$</td>
<td>$2^{18}$</td>
<td></td>
<td>40</td>
<td>2.8</td>
</tr>
<tr>
<td>G8</td>
<td>$3.5 \times 10^{10}$</td>
<td>80000</td>
<td></td>
<td>116</td>
<td>7.4</td>
</tr>
<tr>
<td>B26</td>
<td>$2.3 \times 10^{20}$</td>
<td>50</td>
<td></td>
<td>88</td>
<td>1060</td>
</tr>
</tbody>
</table>

Table 2: Timings for all subgroups of PC-groups

<table>
<thead>
<tr>
<th>$G$</th>
<th>Order</th>
<th>$\leq n$</th>
<th>Normal subgrps</th>
<th>Time (secs)</th>
</tr>
</thead>
<tbody>
<tr>
<td>G5</td>
<td>$8.4 \times 10^8$</td>
<td>100</td>
<td>8</td>
<td>0.2</td>
</tr>
<tr>
<td>G8</td>
<td>$3.5 \times 10^{10}$</td>
<td>80000</td>
<td>37</td>
<td>1.1</td>
</tr>
<tr>
<td>B26</td>
<td>$2.3 \times 10^{20}$</td>
<td>50</td>
<td>88</td>
<td>543</td>
</tr>
</tbody>
</table>

Table 3: Timings for normal subgroups of PC-groups
Table 4: Timings for bounded-codimension submodules

<table>
<thead>
<tr>
<th>Field Size</th>
<th>Dimension Bound</th>
<th>Codim Bound</th>
<th>Num Submods</th>
<th>Time (secs)</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>5</td>
<td>1</td>
<td>8</td>
<td>0.019</td>
</tr>
<tr>
<td></td>
<td></td>
<td>2</td>
<td>24</td>
<td>0.06</td>
</tr>
<tr>
<td></td>
<td></td>
<td>3</td>
<td>40</td>
<td>0.189</td>
</tr>
<tr>
<td>3</td>
<td>10</td>
<td>1</td>
<td>41</td>
<td>0.90</td>
</tr>
<tr>
<td></td>
<td></td>
<td>2</td>
<td>626</td>
<td>1.45</td>
</tr>
<tr>
<td></td>
<td></td>
<td>3</td>
<td>4386</td>
<td>33.041</td>
</tr>
<tr>
<td>2</td>
<td>26</td>
<td>1</td>
<td>1</td>
<td>0.060</td>
</tr>
<tr>
<td></td>
<td></td>
<td>2</td>
<td>5</td>
<td>0.079</td>
</tr>
<tr>
<td></td>
<td></td>
<td>3</td>
<td>5</td>
<td>0.090</td>
</tr>
<tr>
<td></td>
<td></td>
<td>4</td>
<td>11</td>
<td>0.130</td>
</tr>
<tr>
<td></td>
<td></td>
<td>6</td>
<td>36</td>
<td>0.280</td>
</tr>
</tbody>
</table>

large factors in the chief series can also have a significant effect on the algorithm’s performance. For instance, the timings given in the table for G5 rely on the largest chief factor (5^8) appearing at the bottom of the chief series. If it appears as the second to last term, the computations do not complete.

To illustrate the submodule algorithm, let $G$ be the two-generator Burnside group $B(2,6)$ of exponent 6 and order $2^{28}3^{25}$. A normal series for a soluble group $G$ having the property that the quotient group of any two successive terms of the series is elementary abelian, is called an elementary abelian series. In the shortest such series for $G$, the elementary abelian sections have orders $2^2$, $3^5$, $3^{10}$, $3^{10}$, $2^{26}$, where the top section is trivial and the two of order $3^{10}$ are isomorphic as $G$-modules. For each of the three non-trivial non-isomorphic abelian sections in this series, we have constructed the corresponding $G$-module and then computed the bounded-codimension submodules for various bounds. The running time clearly depends strongly on the number of submodules which need to be constructed. Of course, the computation of the bounded-codimension submodules will in practice be negligible in the whole subgroups algorithm, unless the input group has large abelian sections.

References


ATLAS of Finite Groups, Oxford.


Computation, III (Columbus, OH, 1999), 139–153, Ohio State Univ. Math. Res.
Inst. Publ., 8, de Gruyter, Berlin.


Math. Soc. 57, 1–16.


ternational Symposium on Symbolic and Algebraic Computation (Rostock),


degree up to 15. J. Symbolic Comput. 30, 675–716.


Addresses:

School of Mathematics and Statistics
University of Sydney
NSW 2006
Australia
e-mail: john@maths.usyd.edu.au

Mathematics Institute
University of Warwick
Coventry CV4 7AL
Great Britain
e-mail: dfh@maths.warwick.ac.uk

Department of Mathematics, Statistics, and Computer Science
Marquette University
Milwaukee WI 53201-1881
USA
e-mail: mikes@mscs.mu.edu

School of Mathematics and Statistics
University of Sydney
NSW 2006
Australia
e-mail: allan@maths.usyd.edu.au