J. Symbolic Computation (1996) **11**, 1–20

Some Algorithms for Nilpotent Permutation Groups*

EUGENE M. LUKS†, FERENC RÁKÓCZI† AND CHARLES R. B. WRIGHT‡

†*Department of Computer and Information Science, University of Oregon*
‡*Department of Mathematics, University of Oregon*

(Received)

Let G, H and E be subgroups of a finite nilpotent permutation group of degree n. We describe the theory and implementation of an algorithm to compute the normalizer $N_G(H)$ in time polynomial in n, and we give a modified algorithm to determine whether H and E are conjugate under G and, if so, to find a conjugating element of G. Other algorithms produce the intersection $G \cap H$ and the centralizer $C_G(H)$. The underlying method uses the imprimitivity structure of $\langle G, H\rangle$ and an associated canonical chief series to reduce computation to linear operations. Implementations in GAP and Magma are practical for degrees large enough to present difficulties for general-purpose methods.

1. Introduction and Related Work

The normalizer problem—given finite groups G and H, to compute the normalizer $N_G(H)$ of H in G—is of both practical and theoretical interest. In the context of permutation groups it is not known to be solvable in time polynomial in the permutation degree. Indeed, its complexity is of special interest because of its relation to the problem of testing graph isomorphism (see, e.g., Luks, 1993). Currently implemented algorithms for its solution have exponential worst case running time. The implementations appear to remain exponential even for nilpotent groups, for which the normalizer problem is known to be solvable in polynomial time (Kantor and Luks, 1990). More generally, normalizers are computable in polynomial time even for solvable groups (Luks, 1992). With this background in mind, we describe a normalizer algorithm for nilpotent subgroups of S_n that has worst case timing of $O(n^4)$. Implementations in GAP (Schönert et al., 1993) and Magma (Cannon and Playoust, 1993) show substantial performance improvements over the built-in library functions on permutation domains of moderate size.

Our point of view in this work is that if the groups under consideration are known to possess special properties, such as nilpotency, then one should hope to exploit that knowledge to devise algorithms that are faster than the generic ones that apply to less

* A portion of the work described here was presented at the 1994 ISSAC conference at Oxford and appears in the conference proceedings.
† Research supported in part by NSF grant CCR-9013410.
‡ The third author wishes to thank the Australian National University, the Deutscher Akademischer Austauschdienst and the RWTH-Aachen for support during the preparation of this article.

0747–7171/90/000000 + 00 $03.00/0

restricted classes of groups. We consider here the normalizer problem and the related conjugator problem—given groups G, H and E, determine whether there is a g in G with $g^{-1}Eg = H$, and if so find such a g—in the setting of nilpotent permutation groups.

Our aim is to take advantage of the polycyclic and hypercentral structure of nilpotent groups in the overall design of our algorithms, and to use the permutation group environment not only to multiply elements quickly but also to create a combinatoric structure forest that leads to a canonical chief series and to efficient linear calculations. See, e.g., Butler and Cannon (1993) for examples of exploitation of imprimitivity systems of p-groups to get at the group structure. Not surprisingly, our algorithms have some characteristics in common with the normalizer algorithm of Glasby and Slattery (1990) and related algorithms (cf. Celler, Neubüser and Wright, 1990) for polycyclic groups. The latter algorithms use collection methods to multiply elements. Our experimental results indicate that algorithms that can take advantage of the combination of nilpotency and permutation action can be faster than those based on either collection or permutation operations alone.

The following account starts by presenting the framework for a general normalizer algorithm and a reduction of the nilpotent problem to the computation of a subgroup of index 1 or a prime. We then discuss modifications needed to solve the conjugator problem. The next two sections develop and apply linear algebraic methods derived from a structure forest which we build from the permutation action. We briefly describe related polynomial-time algorithms for computing intersections and centralizers, and conclude with discussions of theoretical complexity and experimental results.

2. The Normalizer Algorithm

Our goal in this section will be an algorithm to compute $N_G(H)$ for subgroups G and H of a finite nilpotent group K. We can reduce immediately to the case in which K has order a power of some prime p, since K is the direct product of its Sylow subgroups, and the direct decomposition is inherited by the subgroups G, H, and $N_G(H)$. Generators for the p-Sylow groups of G, H, and K are easy to compute as powers of the generators of G, H, and K, using the Euclidean algorithm in $\mathbb{Z}$.

Thus we suppose for the rest of this section that K is a p-group. Then K has a central chief series $K = K_0 \rhd K_1 \rhd \cdots \rhd K_L = 1$, with $K_j \trianglelefteq K$ for each j and with each factor K_{j-1}/K_j of order p. In Section 5 we will explicitly construct such a series suited to our needs. For now, we suppose a series like this to be given.

Define $H_i := K_i \cap H$ for $i = 0, \ldots, L$. The overall plan of the algorithm is to compute the normalizers of certain subgroups H_iK_j of K, in a sequence starting with a subgroup that is obviously G-normal and ending with the subgroup $H_0K_L = (K_0 \cap H)K_L = H$. Figure 1 gives a preliminary version, with M an ambient normalizer which is initially G and eventually $N_G(H)$. Equations in braces $\{\ \}$ are assertions about the values of variables at those stages of the execution where the assertions appear. Here and in subsequent algorithms, **endfor** and **endif** statements are implied by the indentation. The assertions enclosed in braces in Figure 1 are immediate, since $H_L = 1$, $N_G(H_iK_i) = N_G(K_i) = G$, $H_iK_L = H_i$, $N_G(H_i) \leq N_G(H_{i+1})$, and $H_0 = H$. Hence the algorithm does produce $N_G(H)$.

Note that the overall structure of our normalizer algorithm differs from that of Glasby and Slattery (1990) only in having the inner and outer loops interchanged. At this level of

{**input:** *Subgroups G and H of a group K.*
A normal series $K = K_0 \rhd \cdots \rhd K_L = 1$ of K.
$H_i = K_i \cap H$ for $i = 0, \ldots, L$.}
{**output:** $N_G(H)$.}
begin
{*Initialize the ambient normalizer M.*}
$M := G$
{$M = N_G(H_L)$.}
for $i := L - 1$ **downto** 0 **do**
 {$M = N_G(H_{i+1})$}
 for $j := i$ **to** $L - 1$ **do**
  {$M = N_G(H_i K_j) \cap N_G(H_{i+1})$}
  $M := N_M(H_i K_{j+1})$
  {$M = N_G(H_i K_{j+1}) \cap N_G(H_{i+1})$}
 {$M = N_G(H_i)$}
{$M = N_G(H)$}
return M
end.

Figure 1. The Normalizer Algorithm.

{**input:** *Subgroups G and H of a finite p-group K.*
A chief series $K = K_0 \rhd \cdots \rhd K_L = 1$ of K.
$H_i = K_i \cap H$ for $i = 0, \ldots, L$.}
{**output:** $N_G(H)$.}
begin
$M := G$
for $i := L - 2$ **downto** 0 **and** $H_i \neq H_{i+1}$ **do**
 for $j := i + 1$ **to** $L - 1$ **do**
  $M := N_M(H_i K_{j+1})$
return M
end.

Figure 2. The Normalizer Algorithm for p-Groups.

discussion that difference is inconsequential; it becomes more meaningful as we specialize to the p-group case, and it is significant in the implementation of the resulting algorithm.

If $K = K_0 \rhd K_1 \rhd \cdots \rhd K_L = 1$ is a chief series of K, then each factor K_i/K_{i+1} is K-central of prime order, so either $H_i = H_{i+1}$, i.e., H *avoids* K_i/K_{i+1}, or $HK_i = HK_{i+1}$, i.e., H *covers* K_i/K_{i+1}. If $H_i = H_{i+1}$, then we can skip the inner loop for i. Similarly, if $HK_j = HK_{j+1}$ then since $K_i \geq K_j$ the modular law gives $H_i K_j = (K_i \cap H)K_j = K_i \cap (HK_j) = \ldots = H_i K_{j+1}$. We could skip the inside step for j in this case, but in the more detailed algorithm that we describe below we will still carry out the step for j in the covering case in order to update additional data for $j + 1$. The indices of the factors K_s/K_{s+1} that H covers and avoids can be determined initially, as a byproduct of other computations.

Since K_{L-1} is K-central, H_{L-1} must be G-normal, so we can begin the outer loop with $i = L - 2$. Moreover, $[H_i, G] \leq [K_i, G] \leq K_{i+1}$ so $N_G(H_i K_{i+1}) = G$ and we can begin the inside loop with $j = i + 1$. Figure 2 shows the resulting streamlined algorithm.

To carry out the algorithm, we must compute $N_M(H_iK_{j+1})$ assuming the following conditions:

(a) M normalizes H_{i+1};
(b) M normalizes H_iK_j;
(c) $0 \leq i < j < L$;
(d) H covers K_i/K_{i+1}, so $HK_i = HK_{i+1}$ and $[H_i : H_{i+1}] = p$.

If H covers K_j/K_{j+1}, i.e., if $HK_j = HK_{j+1}$ then $H_iK_j = H_iK_{j+1}$ and no computation is needed; hence suppose also that $H_j = H_{j+1}$. It then follows from (a)—(d) that the group $V := H_iK_j/H_{i+1}K_{j+1}$ is elementary abelian of order p^2, and M acts on it as a p-subgroup of $\mathrm{GL}(2,p)$. Since M centralizes the 1-space $H_{i+1}K_j/H_{i+1}K_{j+1}$, the stabilizer of the 1-space $H_iK_{j+1}/H_{i+1}K_{j+1}$ is the subgroup of M that acts diagonally on V. The stabilizer is thus the kernel of the action, and hence has index 1 or p in M. We have proved the following.

Proposition 2.1. *Suppose that (a)—(d) hold. If H avoids K_j/K_{j+1}, then $N_M(H_iK_{j+1})$ is the kernel of the action of M on V, and hence is either M or a maximal subgroup of M.*

Indeed, we can describe the M-action explicitly. Suppose $h_{i+1} \in H_i \backslash H_{i+1}$. Relative to the basis $\{h_{i+1}H_{i+1}K_{j+1}, z_{j+1}H_{i+1}K_{j+1}\}$ for V, the action has the matrix representation

$$g \mapsto \begin{pmatrix} 1 & \theta(g) \\ 0 & 1 \end{pmatrix},$$

where $[h_{i+1}, g] \in z_{j+1}^{\theta(g)}H_{i+1}K_{j+1}$ and θ is a homomorphism of G into $\mathbb{Z}_p$.

To apply this proposition, we need elements $h_{i+1} \in H_i \backslash H_{i+1}$, which we obtain from generating sets for the groups H and K.

The group K contains elements $z_1, \dots, z_L$ with $K_{i-1} = \langle z_i, \dots, z_L \rangle$ for $i = 1, \dots, L+1$. Section 5 describes the construction of such a sequence for K in an important special case. For now, suppose that a *canonical generating sequence* (CGS) $z_1, \dots, z_L$ of this sort has been chosen for K.

If X is a subgroup of K, then $X = X \cap K_0 \trianglerighteq \cdots \trianglerighteq X \cap K_L = 1$ is a central series for X with each factor of order 1 or p. We call a sequence $x_1, \dots, x_t$ of generators for X an *induced generating system* (IGS) for X (relative to $K_0 \triangleright \cdots \triangleright K_L$) in case the subgroups $\langle x_i, \dots, x_t \rangle$ for $i = 1, \dots, t+1$ are the distinct subgroups in the chain $X \cap K_0 \trianglerighteq \cdots \trianglerighteq X \cap K_L$. If $x_1, \dots, x_t$ is an IGS for X, then $|X| = p^t$, and for each chief factor K_{j-1}/K_j covered by X there is a unique i with $K_{j-1} = \langle x_i \rangle K_j$.

An alternative type of generating sequence for X, which we call a *strong generating system* (SGS) for X relative to the CGS $z_1, \dots, z_L$, consists of elements $x_1, \dots, x_L$, with $x_i = 1$ in case $X_{i-1} = X_i$, i.e., in case X avoids K_{i-1}/K_i, and with $x_i \in X_{i-1} \backslash X_i$ and $x_i \equiv z_i \bmod K_i$ in case X covers K_{i-1}/K_i. Thus if $x_1, \dots, x_L$ is an SGS for X relative to $z_1, \dots, z_L$, then $X_i = \langle x_{i+1}, \dots, x_L \rangle$ for each i.

We will describe H by an SGS $h_1, \dots, h_L$, which we can compute initially by sifting a set of generators for H against $z_1, \dots, z_L$. In our implementations we compute $h_1, \dots, h_L$, as well as other initial SGS's and IGS's, by a modification of the organization of the Sims-Schreier methods in Knuth (1991).

We could use either an SGS or an IGS to describe the subgroup M in the normalizer algorithm. We illustrate both methods below, by using an SGS in the normalizer update algorithm of the next section and an IGS in the conjugator algorithm in Section 4.

3. Recomputing the Normalizer

The inner loop in the normalizer algorithm of Figure 2 indicates a way of updating our current normalizer, i.e., of computing $N_G(H_i)$ from $N_G(H_{i+1})$. In this section we develop a more detailed algorithm for the purpose, based on some of the ideas and facts from Section 2. In essence the plan is, as before, to go from H_iK_j to H_iK_{j+1}, taking different actions depending on whether or not these two groups are the same.

For $j = 1, \dots, L$ let $M_j = N_G(H_{i+1}) \cap N_G(H_iK_j)$. Given M_j, we want to find M_{j+1}, which we know has index 1 or p in M_j. If the index is p, then M_{j+1} covers all but one composition factor of M_j, and the idea is to locate the avoided factor and then modify M_j above it to produce M_{j+1}. To carry out this plan, we maintain two sequences: an SGS $m_1, \dots, m_L$ for M_j, and an auxiliary sequence $x_1, \dots, x_L$ of members of H_{i+1}. If $M_j > M_{j+1}$ and if m_s corresponds to a composition factor of M_j that M_{j+1} avoids, then we will replace m_s by 1 and modify $m_1, \dots, m_{s-1}$ suitably. Because $[K, K_j] \subseteq K_{j+1}$, we have $\langle m_{j+1}, \dots, m_L\rangle \subseteq M_j \cap K_j \subseteq N_{M_j}(H_iK_{j+1}) = M_{j+1}$, so in this case $s \le j$.

The main loop in Figure 3 contains the expanded algorithm for computing M_{j+1} from M_j. Since G centralizes H_i mod K_{i+1}, we have $M_i = M_{i+1}$, so we may start with $j = i+1$. Let $h = h_{i+1}$. Since $h =\in K_i$, on first entry into the main j-loop $[h, m_k] \equiv 1 \equiv x_k \bmod K_{i+1}$ for each k. To show that the algorithm produces the correct result, we assume the statements in braces at the beginning of the j-loop and verify the statements at the end of the loop by considering cases.

Suppose first that $H_j = H_{j+1}$, that $1 \le s = \max\{k\colon \phi(k) \ne 0\}$, and that $1 \le k \le s$. Then $[h, m_k] \equiv x_k z_{j+1}^{\phi(k)} \bmod K_{j+1}$ at the beginning of the loop. Proposition 1 applies, since $H_j = H_{j+1}$. In the notation of Section 2, $\theta(m_k) = \phi(k)$, so $\theta(m_k m_s^{\alpha(k)}) = \theta(m_k) + \alpha(k)\theta(m_s) = \phi(k) + \alpha(k)\phi(s) = 0$, whence $m_k m_s^{\alpha(k)} \in M_{j+1}$. Now $\theta(m_s) \ne 0$, so $m_s \notin \ker\theta = M_{j+1}$, and M_{j+1} must be a maximal subgroup of M_j. Thus the sequence $m_1 m_s^{\alpha(1)}, \dots, m_{s-1} m_s^{\alpha(s-1)}, 1, m_{s+1}, \dots, m_L$ is an SGS for M_{j+1} in this case.

We know that x_s and x_k are in H_{i+1}, which is normalized by M_j. Hence

$$(x_s m_s^{-1})^{\alpha(k)} x_k m_s^{\alpha(k)} \in H_{i+1}(m_s^{-1})^{\alpha(k)} x_k m_s^{\alpha(k)} = H_{i+1}.$$

Now to show that $[h, m_k m_s^{\alpha(k)}] \equiv (x_s m_s^{-1})^{\alpha(k)} x_k m_s^{\alpha(k)} \bmod K_{j+1}$ we may assume that $K_{j+1} = 1$, so that $z_{j+1} \in Z(K)$. Let $z := z_{j+1}$. Then $[h, m_k] = x_k z^{\phi(k)}$, so $m_k^h = m_k x_k^{-1} z^{-\phi(k)}$. It follows that

$$\begin{aligned}
(m_k m_s^{\alpha(k)})^h &= m_k x_k^{-1} z^{-\phi(k)} (m_s x_s^{-1} z^{-\phi(s)})^{\alpha(k)} \\
&= m_k x_k^{-1} (m_s x_s^{-1})^{\alpha(k)} z^{-\phi(k) - \phi(s)\alpha(k)} \\
&= m_k m_s^{\alpha(k)} m_s^{-\alpha(k)} x_k^{-1} (m_s x_s^{-1})^{\alpha(k)},
\end{aligned}$$

so $[m_k m_s^{\alpha(k)}, h] = ((x_s m_s^{-1})^{\alpha(k)} x_k m_s^{\alpha(k)})^{-1}$, as desired.

Suppose next that $H_j = H_{j+1}$ but that $\phi(k) = 0$ for every k. Then $M_j = M_{j+1}$, and already $[h, m_k] \equiv x_k \bmod K_{j+1}$ for each k.

In the final case, with $H_j \ne H_{j+1}$, $M_j = M_{j+1}$ again, and $m_1, \dots, m_L$ is still an SGS.

{**input:** *An SGS* $m_1, \ldots, m_L$ *for* $N_G(H_{i+1})$.}
{**output:** *An SGS for* $N_G(H_i)$.}

begin
{*Initialize.*}
for $k := 1$ **to** L **do**
  $x_k := 1 \in H_{i+1}$
for $j := i+1$ **to** $L-1$ **do**
  {$(m_1, \ldots, m_L)$ *is an SGS for* M_j,
   $x_1, \ldots, x_L \in H_{i+1}$, $x_k = 1$ *for* $j \leq k$,
   and $[h_{i+1}, m_k] \equiv x_k \bmod K_j$ *for* $k = 1, \ldots, L$}
  for $k := 1$ **to** j **do**
    Compute $\phi(k) \in \mathbb{Z}_p$ with
    $x_k^{-1}[h_{i+1}, m_k] \equiv z_{j+1}^{\phi(k)} \bmod K_{j+1}$
  if $H_j = H_{j+1}$ **then**
   if $\phi(k) \neq 0$ for some k **then**
    $s := \max\{k: \phi(k) \neq 0\}$
    for $k := 1$ **to** $s-1$ **do**
      Solve $\phi(s)\alpha(k) + \phi(k) = 0$ for $\alpha(k) \in \mathbb{Z}_p$
      $m_k := m_k m_s^{\alpha(k)}$
      $x_k := (x_s m_s^{-1})^{\alpha(k)} x_k m_s^{\alpha(k)}$
    $m_s := 1$
    $x_s := 1$
  else $\{H_j \neq H_{j+1}\}$
   for $k := 1$ **to** j **do**
     $x_k := x_k h_{j+1}^{\phi(k)}$
  {$(m_1, \ldots, m_L)$ *is an SGS for* M_{j+1},
   $x_1, \ldots, x_L \in H_{i+1}$, $x_k = 1$ *for* $j+1 \leq k$,
   and $[h_{i+1}, m_k] \equiv x_k \bmod K_{j+1}$ *for* $k = 1, \ldots, L$}
return $(m_1, \ldots, m_L)$
end.

Figure 3. Normalizer update from $N_G(H_{i+1})$ to $N_G(H_i)$.

Moreover, since $H_j \neq H_{j+1}$, $h_{j+1}K_{j+1} = z_{j+1}K_{j+1}$ and thus $[h, m_k] \equiv x_k h_{j+1}^{\phi(k)} \bmod K_{j+1}$, with $x_k h_{j+1}^{\phi(k)} \in H_{i+1}$ since $j \geq i+1$.

This algorithm can be speeded up somewhat by making a few small changes. If we arrange to have $x_k = 1$ and $\phi(k) = 0$ whenever $m_k = 1$, then we may ignore some cases. Specifically, let us initialize $\phi(k)$ to 0 for all k. If $m_k := m_k m_s^{\alpha(k)}$ produces $m_k = 1$, let us set $x_k := 1$ and $\phi(k) := 0$, and when we set $m_s := 1$ let us also set $\phi(s) := 0$. Then in the first k-loop we may ignore cases with $m_k = 1$, and in the other two k-loops we may ignore cases with $\phi(k) = 0$. We justify these claims as follows.

Initially, $x_k = 1$ and $\phi(k) = 0$ for every k. If m_k gets the value 1 at some stage, we explicitly set x_k and $\phi(k)$ to 1 and 0, respectively. From then on, we ignore cases in which $m_k = 1$ or $\phi(k) = 0$, so these values of x_k and $\phi(k)$ are never reset. Thus $x_k = 1$ and $\phi(k) = 0$ whenever $m_k = 1$, and in these cases, too, $[h, m_k] \equiv x_k z_{j+1}^{\phi(k)} \bmod K_{j+1}$.

4. The Conjugator Algorithm

This section describes a modification of the normalizer algorithm to test whether two subgroups, E and H, of K are G-conjugate, and if so to produce an element g of G with $E^g = H$.

For permutation groups, there is an elementary general reduction of the conjugator problem to the normalizer problem. Given generators for the subgroups G, H_0 and H_1 of Sym(Ω), let $\hat{\Omega} = \Omega \times \{0,1\}$, $\hat{G} = G \wr \mathbb{Z}_2$, and $\hat{H} = H_0 \times H_1$. Both $\hat{G}$ and $\hat{H}$ have natural interpretations as subgroups of Sym($\hat{\Omega}$):

$$\hat{G} = \{(g_0, g_1)t^e \mid g_0, g_1 \in G, e = 0 \text{ or } 1\}, \;\; \hat{H} = \{(h_0, h_1) \mid h_0 \in H_0, h_1 \in H_1\},$$

and for each $\omega \in \Omega$, $i \in \{0,1\}$, $g_0, g_1 \in G$, and $(h_0, h_1) \in \hat{H}$,

$$\begin{array}{lcl} (\omega, i)^{(g_0,g_1)} & = & (\omega^{g_i}, i) \\ (\omega, i)^t & = & (\omega, 1-i) \\ (\omega, i)^{(h_0,h_1)} & = & (\omega^{h_i}, i). \end{array}$$

If R is a set of generators for $N_{\hat{G}}(\hat{H})$, then H_0 and H_1 are G-conjugate if and only if R contains an element of the form $(g_0, g_1)t$, in which case $g_0^{-1} H_0 g_0 = H_1$ for each such element in R.

The general reduction just given is of little use in the present context, since the normalizer algorithm of Section 2 can only compute $N_{\hat{G}}(\hat{H})$ in case $\hat{G}$ is nilpotent, so only if G is a 2-group. Moreover, even in that case, doubling the degree of the permutation group potentially increases the execution time by a factor of 2^4 (see Section 8). The modified algorithm below solves the conjugator problem for p-groups in less than twice the time required by the normalizer algorithm to compute $N_G(H)$ (and typically in essentially the same time).

For the rest of this section we suppose, as in Section 2, that G and H are subgroups of the finite p-group K, with notation G_i, H_i, and K_j as before. In addition, we consider a subgroup E of K, define $E_i := E \cap K_i$ for $i = 1, \ldots, L$ and let $e_i, \ldots, e_L$ be a corresponding SGS for E. The problem is to determine, if possible, an element g of G with $E^g = H$. Figure 4 shows the general outline of an algorithm, based on the p-group normalizer algorithm of Figure 2. If E and H are G-conjugate, then they are K-conjugate as well, and hence cover and avoid the same chief factors K_{i-1}/K_i, i.e., satisfy $e_i = 1$ if and only if $h_i = 1$. Suppose that E and H pass this test. We saw in Section 2 that if the algorithm does not return *false* then $M = N_G(H)$ at the conclusion. Since $E_{L-1} = H_{L-1}$, we may begin with $i = L - 2$. If $H_i = H_{i+1}$, then also $E_i = E_{i+1}$ and no action is required. For $j = i + 1$ and $E^g_{i+1} = H_{i+1}$ we have $(E_i K_{i+1})^g = (\langle e_{i+1} \rangle K_{i+1})^g = \langle e_{i+1}[e_{i+1}, g] \rangle K_{i+1} = \langle e_{i+1} \rangle K_{i+1} = \langle h_{i+1} \rangle K_{i+1} = H_i K_{i+1}$. Thus if the algorithm does not return *false* then the statements in braces in Figure 4 are true, and $E^g = H$ at the conclusion. Note that if H (and hence also E) covers K_j/K_{j+1} in the inner loop, then H and E^{gy} both cover K_j/K_{j+1}, so $H_i K_{j+1} = H_i K_j = E_i^{gy} K_j = E_i^{gy} K_{j+1}$, and we may take $x = 1$ and leave y unchanged.

To carry out the conjugator update step, we modify the normalizer update algorithm of Figure 3 to obtain the algorithm of Figure 5. Here we have chosen to maintain an IGS for M, rather than an SGS, to illustrate the difference in details. Instead of setting $m_s := 1$, with corresponding $x_s := 1$ and $\phi(s) := 0$, we drop m_s out entirely, and shift $m_{s+1}, \ldots, m_t$ and $x_{s+1}, \ldots, x_t$ forward. For convenience, we have replaced E^g by E. Note that if $(e_1, \ldots, e_L)$ is an SGS for E, then $(e_1^g, \ldots, e_L^g)$ is also an SGS for E^g. To verify

```
{input:  Subgroups G, H and E of a finite p-group K.
         A chief series K = K_0 ▷ ··· ▷ K_L = 1 of K.
         H_i = K_i ∩ H for i = 0, ..., L.
         E_i = K_i ∩ E for i = 0, ..., L.}
{output: Either N_G(H) and an element g of G with E^g = H or
         false if no such g exists.}
begin
if H and E cover different factors in the chief series then
  return false
M := G; g := 1
{E_L^g = H_L. }
for i := L − 2 downto 0 and H_i ≠ H_{i+1} do
   {E_{i+1}^g = H_{i+1} }
   y := 1
   for j := i + 1 to L − 1 do
      {(E_i K_j)^{gy} = H_i K_j}
      if E_i^{gy} K_{j+1} is G-conjugate to H_i K_{j+1} then
         Find x ∈ M with (E_i^{gy} K_{j+1})^x = H_i K_{j+1}.
         Find N_M(H_i K_{j+1}).
         y := yx
         M := N_M(H_i K_{j+1})
      else return false
      {E_i^{gy} K_{j+1} = H_i K_{j+1}}
   g := gy
   {E_i^g = H_i }
{E^g = H}
return M and g
end.
```

Figure 4. The Conjugator Algorithm for p-Groups.

the conjugator update algorithm, we must check that the assertions in braces in Figure 5 hold on entry to the loop for $j = i+1$, that if the algorithm returns *false* then E_i and H_i are not conjugate under G, and that if the loop does not return *false* then the assertions hold at the end of the loop if they hold at the beginning. We have checked in Section 3 all of the assertions except those that relate to u or y or *false*. To verify these, we may assume, for each j, that:

- **(a)** $\langle m_1, \ldots, m_t\rangle$ normalizes H_{i+1};
- **(b)** $\langle m_1, \ldots, m_t\rangle$ normalizes $H_i K_j$;
- **(c)** $0 \le i < j < L$;
- **(d)** $HK_i = HK_{i+1}$ and $[H_i : H_{i+1}] = p$;
- **(f)** E and H cover and avoid the same chief factors of K;
- **(g)** $E_i K_j = H_i K_j$;
- **(h)** $E_{i+1} = H_{i+1}$.

Note that for $k > i$, if $u \in H_{i+1} = E_{i+1}$, $y \in N_G(H_{i+1})$, and $ue_{i+1}^{-y} h_{i+1} \in K_k$, then $E_i^y K_k = \langle e_{i+1}^y\rangle E_{i+1} K_k = \langle e_{i+1}^y\rangle H_{i+1} K_k = \langle h_{i+1}\rangle H_{i+1} K_k = H_i K_k$.

At the outset $y = 1$, and since $\{m_1, \ldots, m_t\} \subseteq N_G(H_{i+1})$, y remains in $N_G(H_{i+1})$. Suppose that $j = i+1$. Then $u = 1 \in H_{i+1}$. Since H covers K_i/K_{i+1}, so does E. Since $h_{i+1} \equiv e_{i+1} \bmod K_{i+1}$, $ue_{i+1}^{-1} h_{i+1} \in K_j$ on first entry into the loop.

{**input:** *An IGS* $(m_1, \ldots, m_t)$ *for* $N_G(H_{i+1})$, *SGS's for* H *and* E.}
{**output:** *Either an IGS for* $N_G(H_i)$ *and an element* $y \in G$ *with* $E_i^y = H_i$
or **false** *if no such* y *exists.*}
begin
$y := 1; \quad u := 1$
for $k := 1$ **to** t **do**
  $x_k := 1 \in H_{i+1}$;
for $j := i+1$ **to** $L-1$ **do**
  {$(m_1, \ldots, m_t)$ *is an IGS for* $N_G(H_{i+1}) \cap N_G(H_i K_j)$, $y \in N_G(H_{i+1})$,
  $u, x_1, \ldots, x_t \in H_{i+1}$, $ue_{i+1}^{-y} h_{i+1} \in K_j$, $E_i^y K_j = H_i K_j$, *and*
  $[h_{i+1}, m_k] \equiv x_k \bmod K_j$ *for* $k = 1, \ldots, t$}
  for $k := 1$ **to** t **do**
    Compute $\phi(k) \in \mathbb{Z}_p$ with $x_k^{-1}[h_{i+1}, m_k] \equiv z_{j+1}^{\phi(k)} \bmod K_{j+1}$.
  Compute $\lambda \in \mathbb{Z}_p$ with $ue_{i+1}^{-y} h_{i+1} \equiv z_{j+1}^{\lambda} \bmod K_{j+1}$.
  if $H_j = H_{j+1}$ **then**
    if $\phi(k) = 0$ for all k **then**
      if $\lambda \neq 0$ **then**
        return *false*
    else {$\phi(k) \neq 0$ for some k}
      $s := \max\{k : \phi(k) \neq 0\}$
      for $k := 1$ **to** $s-1$ **do**
        Solve $\phi(s)\alpha(k) + \phi(k) = 0$ for $\alpha(k) \in \mathbb{Z}_p$.
        $m_k := m_k m_s^{\alpha(k)}; \quad x_k := (x_s m_s^{-1})^{\alpha(k)} x_k m_s^{\alpha(k)}$
      Solve $\beta\phi(s) = \lambda$ for $\beta \in \mathbb{Z}_p$.
      $y := y m_s^{\beta}$
      $u := (x_s m_s^{-1})^{\beta} u m_s^{\beta}$
      $t := t - 1$
      for $k := s$ **to** t **do**
        $m_k := m_{k+1}; \quad x_k := x_{k+1}$
  else {$H_j \neq H_{j+1}$}
    for $k := 1$ **to** t **do**
      $x_k := x_k h_{j+1}^{\phi(k)}$
    $u := u h_{j+1}^{-\lambda}$
  {$(m_1, \ldots, m_t)$ *is an IGS for* $N_G(H_{i+1}) \cap N_G(H_i K_{j+1})$, $y \in N_G(H_{i+1})$,
  $u, x_1, \ldots, x_t \in H_{i+1}$, $ue_{i+1}^{-y} h_{i+1} \in K_{j+1}$, $E_i^y K_{j+1} = H_i K_{j+1}$, *and*
  $[h_{i+1}, m_k] \equiv x_k \bmod K_{j+1}$ *for* $k = 1, \ldots, t$}
return $(m_1, \ldots, m_t)$, y
end.

Figure 5. Conjugator update from $i+1$ to i.

Thus we need only check for each j that $u \in H_{i+1}$ and $ue_{i+1}^{-y} h_{i+1} \in K_{j+1}$ at the end of the loop.

Suppose that $H_j \neq H_{j+1}$, so that H_i and E_i^y cover K_j/K_{j+1}. Then y does not change, and $e_{i+1}^{-y} h_{i+1} \equiv z_{j+1}^{\lambda} u^{-1} \equiv h_{j+1}^{\lambda} u^{-1} \bmod K_{j+1}$, so $u := uh_{j+1}^{-\lambda}$ yields $ue_{i+1}^{-y} h_{i+1} \in K_{j+1}$ at the end of the loop. Since $i < j$, we have $uh_{j+1}^{-\lambda} \in H_{i+1}$ as well.

Next consider the case $H_j = H_{j+1}$. Then E and H avoid K_j/K_{j+1} and cover K_i/K_{i+1}. As we saw for the normalizer algorithm, the group $H_i K_j / H_{i+1} K_{j+1}$ is elementary abelian. Let $\bar{h} := h_{i+1} H_{i+1} K_{j+1}$, $\bar{z} := z_{j+1} H_{i+1} K_{j+1}$, and $\bar{e} := e_{i+1} H_{i+1} K_{j+1}$. Then we have $H_i K_{j+1} / H_{i+1} K_{j+1} = \langle \bar{h} \rangle$, $E_i K_{j+1} / H_{i+1} K_{j+1} = \langle \bar{e} \rangle$, and $H_{i+1} K_j / H_{i+1} K_{j+1} = \langle \bar{z} \rangle$.

If $\phi(k) = 0$ for every k, then $N_G(H_{i+1}) \cap N_G(H_i K_j)$ acts trivially on $H_i K_j / H_{i+1} K_{j+1}$, and hence normalizes both $\langle \bar{h} \rangle$ and $\langle \bar{e} \rangle$. In this case, since $ue_{i+1}^{-y} h_{i+1} \equiv z_{j+1}^{\lambda} \bmod K_{j+1}$, if $\lambda = 0$ then $ue_{i+1}^{-y}h_{i+1} \in K_{j+1}$ already, as desired. Otherwise, if $\lambda \neq 0$, which is the only case in which the loop returns *false*, $\bar{h} = \bar{z}^{\lambda}\bar{e}^{y}$ implies that $\langle \bar{h} \rangle \neq \langle \bar{e}^{y} \rangle$, so H_iK_{j+1} and $E_i^yK_{j+1}$ are not conjugate under $N_G(H_{i+1}) \cap N_G(H_iK_j)$. But if E^y and H were G-conjugate, say with $E^{yg} = H$, then we would have $H_iK_{j+1} = E_i^{yg}K_{j+1} = (E_i^yK_{j+1})^g$, but also $H_{i+1} = E_{i+1}^{yg} = H_{i+1}^{yg} = H_{i+1}^g$ and $H_iK_j = (E_i^yK_j)^g = (H_iK_j)^g$, whence $g \in N_G(H_{i+1}) \cap N_G(H_iK_j)$. Thus if the loop returns *false* then H and E are indeed not G-conjugate.

Finally, suppose that $\phi(k) \neq 0$ for some k. Let $m := m_s$ and $x := x_s$. We must verify that

$$(xm^{-1})^{\beta}um^{\beta}e_{i+1}^{-ym^{\beta}}h_{i+1} \in K_{j+1}.$$

As in the verification of the normalizer algorithm, we may assume that $K_{j+1} = 1$. Letting $h := h_{i+1}$ and $z := z_{j+1}$, we have $m^h = mx^{-1}z^{-\phi(s)}$, so that

$$(xm^{-1})^{\beta} = (m^{-h}z^{-\phi(s)})^{\beta} = m^{-\beta h}z^{-\beta\phi(s)} = m^{-\beta h}z^{-\lambda}.$$

Hence,

$$(xm^{-1})^{\beta}um^{\beta}e_{i+1}^{-ym^{\beta}}h = m^{-\beta h}z^{-\lambda}ue_{i+1}^{-y}m^{\beta}h = m^{-\beta h}h^{-1}m^{\beta}h = 1,$$

as desired.

5. The Linear Structure

The algorithms of the preceding sections apply in the setting of an arbitrary p-group K with chief series $K = K_0 \rhd \cdots \rhd K_L = 1$. The update algorithms require the multiplication of group elements to compute products such as $(x_sm_s^{-1})^{\beta}um_s^{\beta}$. They also require finding "leading coefficients" $\phi(1), \ldots, \phi(t)$ (in the sense of Schönert et al., 1993) relative to the canonical generating sequence for K. In this section we develop a linear data structure that permits rapid computation of these coefficients in case K is a permutation group. Section 6 applies the linear results to compute leading coefficients.

We start by constructing a special normal series $K = F_0 \rhd F_1 \rhd \cdots \rhd F_t = 1$ with elementary abelian factors, and then refine this series to a chief series for K. The refinement turns out to be unique, and to be described by a sequence of K-invariant flags in the factors F_{i-1}/F_i, viewed as $\mathbb{Z}_p$-vector spaces. The matrices that describe the bases associated with the flags then provide easy computation of leading coefficients.

To explain the construction of the normal series $F_0 \rhd F_1 \rhd \cdots \rhd F_t = 1$, we use a rooted tree associated with the permutation action of K. This combinatorial structure provides a conceptual framework for the development and verification of our linear methods, but is not itself explicitly created in the implementations described in Section 9.

In general, if K is a finite group of permutations then it is possible (Luks and McKenzie, 1988, and Luks, 1986) to construct a *structure forest* for K consisting of rooted trees, one for each orbit of K, such that in each tree the children of the root correspond to maximal blocks of imprimitivity, and the subtree rooted at the child corresponding to a block is the structure tree for the restriction to that block of its setwise stabilizer. This construction can be carried out essentially as efficiently as finding imprimitivity systems (Atkinson, 1975). In case K is a p-Sylow subgroup of S_{p^t}, the repeated wreath product $K = C_p \wr C_p \wr \ldots \wr C_p$ of t groups of order p, the structure forest consists of a single full p-ary structure tree.

In this paper, G, E, and H are subgroups of a nilpotent permutation group K, so it is possible to compute a structure forest for $\langle G, H\rangle$; the general implementations of our algorithms begin by constructing such a forest for each Sylow subgroup of $\langle G, H\rangle$, using imprimitivity information about $\langle G, H\rangle$. For the following exposition we will assume that K is a p-group and that the forest consists of a single tree. The extension of the resulting linear structure to the general p-group case involves straightforward reformulation of the normal series in K; for example, one can view the disjoint trees as arranged in a vertical list, redefining "layers" in the account below accordingly. Since both the normalizer and conjugator problems for a nilpotent group reduce immediately to its Sylow subgroups, the general nilpotent case presents no special difficulties either.

Thus we let $n = p^t$ and suppose that G, H, and E (if called for) are given as subgroups of the p-Sylow subgroup K of S_n, acting as automorphisms on a full p-ary rooted tree Γ with n leaves. We choose a labeling for Γ to display the lines of our argument clearly. Label the root 0, label its children $0, \ldots p-1$, and in general give the children of the node at depth k with label s the labels $s, s+p^k, \ldots, s+(p-1)p^k$.

The nodes of Γ form layers, on each of which K acts transitively. For $r = 0, 1, \ldots, t$ let F_r be the subgroup of K fixing each of the p^r nodes at depth r. Then $K = F_0 > F_1 > \cdots > F_t = 1$, and each group F_r is normal in K. For $k = 0, \ldots, t-1$ let τ_k be the member of K that maps $xp^{k+1} + jp^k$ to $xp^{k+1} + (j+1 \bmod p)p^k$ for $0 \le j < p$ and all x. Since τ_k fixes $1, \ldots, p^k - 1$, it fixes all of the nodes at levels $0, \ldots, k$, and hence is in F_k. It permutes the children of node 0 at level k in the p-cycle $(0, p^k, \ldots, (p-1)p^k)$, permutes the subtrees rooted at those children correspondingly but otherwise leaves them unchanged, and fixes all descendents of the remaining nodes at level k. The conjugates of τ_r under K permute the children of the other nodes at depth r, so F_r/F_{r+1} is elementary abelian, generated by τ_r and its K-conjugates. Indeed, $K = \langle \tau_0, \tau_1, \ldots, \tau_{t-1}\rangle$, $F_r = \langle \tau_r, \tau_{r+1}, \ldots, \tau_{t-1}\rangle^K$ for each r, and K acts linearly on the $\mathbb{Z}_p$-vector space $V_r := F_r/F_{r+1}$, which has a basis consisting of p^r conjugates of τ_r under K (mod F_{r+1}). To refine the series $K = F_0 \rhd \cdots \rhd F_t = 1$ to a chief series for K we must find for each r a basis $b_0, \ldots, b_{p^r-1}$ for V_r such that every subspace $\langle b_s, b_{s+1}, \ldots, b_{p^r-1}\rangle$ is $\langle \tau_0, \ldots, \tau_{r-1}\rangle$-invariant. The proof of the next proposition gives an easy way to produce such bases, with an additional property that we can exploit in our algorithms.

Proposition 5.1. *For each $r = 1, \ldots, t$ there is a $p^r \times p^r$ matrix $\mathbf{B}_r$ with entries in $\mathbb{Z}_p$ and with the following properties.*

(a) The rows $\mathbf{b}_0, \ldots, \mathbf{b}_{p^r-1}$ of $\mathbf{B}_r$ form a $\mathbb{Z}_p$-basis for $V_r = \mathbb{Z}_p^{p^r}$.
(b) For $s = 0, \ldots, p^r - 1$ the subspace $V_r^{(s)}$ of V_r spanned by $\{\mathbf{b}_s, \ldots, \mathbf{b}_{p^r-1}\}$ is invariant under $\tau_0, \ldots, \tau_{r-1}$.
(c) The inner products of the rows of $\mathbf{B}_r$ satisfy

$$\mathbf{b}_i \cdot \mathbf{b}_j \equiv \begin{cases} 0 & \bmod p \text{ for } i+j \ge p^r \\ (-1)^i & \bmod p \text{ for } i+j = p^r - 1. \end{cases}$$

Moreover, the subspaces $V_r^{(s)}$ in (b) form the unique $\mathbb{Z}_p[K]$-composition series for V_r.

Proof. Note first that $\tau_0 \cdots \tau_r$ induces the p^r-cycle $c = (0, 1, \ldots, p^r - 1)$ on the nodes of Γ at depth r. One way to see this is to write the labels $0, 1, \ldots, p^r - 1$ in p-ary notation and to think of an odometer. Observe that τ_0 increases the 1's digit of a label by 1 mod p, then τ_1 does nothing unless the 1's digit is now 0, in which case it increases the p's digit by 1 mod p, etc. Let σ denote the $\mathbb{Z}_p$-linear transformation of V_r determined by the permutation action of c on the standard basis $\{\mathbf{e}_0, \ldots, \mathbf{e}_{p^r-1}\}$ of V_r. The characteristic polynomial of σ is $x^{p^r} - 1$, which is also its minimal polynomial. Let $\tau = \sigma - 1$. Then τ has minimal polynomial x^{p^r}, and the set $\{\mathbf{e}_0\tau^i : 0 \le i < p^r\}$ is a basis for V_r relative to which τ has as its matrix a single Jordan block.

For $m = 0, 1, \ldots, p^r - 1$ we have $V_r\tau^m = \langle \mathbf{e}_0\tau^i : m \le i < p^r \rangle$, and the subspaces $V_r\tau^m$ form a $\mathbb{Z}_p[\langle\sigma\rangle]$-composition series for V_r. In fact, they form the only such composition series, for if U is a $\langle\sigma\rangle$-invariant subspace of V_r with $U \subseteq V_r\tau^m$ but $U \not\subseteq V_r\tau^{m+1}$, then U contains some member of $\mathbf{e}_0\tau^m + V_r\tau^{m+1}$, hence contains each $\mathbf{e}_o\tau^k$ whenever $k \ge m$, and hence contains $V_r\tau^m$. Since V_r is $\mathbb{Z}_p[\langle\sigma\rangle]$-uniserial, it is $\mathbb{Z}_p[K]$-uniserial as well. In particular, the subspaces $V_r\tau^m$ must be τ_k-invariant for $0 \le k < r$.

Let $\mathbf{B}_r$ be the $p^r \times p^r$ matrix whose rows $\mathbf{b}_0, \mathbf{b}_1, \ldots, \mathbf{b}_{p^r-1}$ are $\mathbf{e}_0, \mathbf{e}_0\tau, \ldots, \mathbf{e}_0\tau^{p^r-1}$, expressed relative to the standard basis. Since $\tau^k = (\sigma - 1)^k$, we have

$$\mathbf{b}_k = \mathbf{e}_0\tau^k = \sum_{j=0}^{k}(-1)^j\binom{k}{j}\mathbf{e}_j \mod p,$$

and $\mathbf{B}_r$ is a lower-triangular “alternating Pascal's triangle” matrix.

The inner products of the rows of $\mathbf{B}_r$ satisfy

$$\begin{aligned}\mathbf{b}_i \cdot \mathbf{b}_j &= \Big(\sum_{\alpha\ge 0}(-1)^\alpha\binom{i}{\alpha}\mathbf{e}_\alpha\Big)\cdot\Big(\sum_{\beta\ge 0}(-1)^\beta\binom{j}{\beta}\mathbf{e}_\beta\Big)\\ &= \sum_{\alpha\ge 0}(-1)^{\alpha+\alpha}\binom{i}{\alpha}\binom{j}{\alpha}\\ &= \binom{i+j}{i}.\end{aligned}$$

If $i + j \ge p^r$, then $\binom{i+j}{i} \equiv 0 \mod p$, and if $i + j = p^r - 1$, then $\mathbf{b}_i \cdot \mathbf{b}_j = \binom{p^r-1}{i} \equiv (-1)^i \mod p$. Thus $\mathbf{B}_r$ satisfies the assertions of the proposition. ∎

It seems to be part of the folklore that if $K = C_p \wr \cdots \wr C_p$ with r factors, then V_r is a uniserial $\mathbb{Z}_p[K]$-module. Our construction selects an especially useful basis from among the many bases that fit the unique chain of subspaces. For related results in another setting, the reader may consult Leedham-Green and Newman (1980), especially Theorem 2, and Leedham-Green, McKay and Plesken (1986), where the “alternating Pascal triangle” matrix $\mathbf{B}$ appears in Section 5.

Since $\mathbf{B}_r$ is essentially Pascal's triangle, we can easily construct $\mathbf{b}_{i+1}$ from $\mathbf{b}_i$. It is just as easy to build $\mathbf{B}_r$ from the bottom up, starting with the last row, which is $(1, 1, \ldots, 1)$. Moreover, as the next proposition shows, one can save a bit of arithmetic in computing some of the entries of $\mathbf{B}_r$ below the diagonal by simply copying already known entries. Subtraction of two entries is required only in the third case of the proposition, which occurs for $(p-1)/(p+1)$ of the pairs (i, j) with $0 \le j \le i < p^r$.

Proposition 5.2. *Let $\mathbf{b}_0, \ldots, \mathbf{b}_{p^r-1}$ be the rows of $\mathbf{B}_r$, and denote the j-th component of $\mathbf{b}_k$ by $\mathbf{b}_{k,j}$. Suppose that $0 < i < p^r$ and that p^m is the highest power of p that divides i. Then $\mathbf{b}_{i-1,0} = 1$, and if $j > 0$ then $\mathbf{b}_{i-1,j}$ is congruent mod p to*

$$\mathbf{b}_{i,j} \qquad \textit{if } p^{m+1} \textit{ divides } j,$$

$$\begin{array}{ll} -\mathbf{b}_{i-1,j-1} & \textit{if } p^m \textit{ does not divide } j, \textit{ and} \\ -\mathbf{b}_{i-1,j-1} + \mathbf{b}_{i,j} & \textit{otherwise.} \end{array}$$

PROOF. We always have $\mathbf{b}_{i-1,0} = 1$ and $\mathbf{b}_{i-1,j-1} + \mathbf{b}_{i-1,j} = \mathbf{b}_{i,j}$. Say $i = p^m\alpha$ and $j = p^s\beta$, with α and β prime to p. Suppose first that p^{m+1} divides j. Then

$$\frac{p^m\alpha}{p^{m+1}p^{s-m-1}\beta} \cdot \binom{i-1}{j-1} = \binom{i}{j} \in \mathbb{Z},$$

and since p does not divide α it follows that $\binom{i-1}{j-1} \in p\mathbb{Z}$, and $\mathbf{b}_{i-1,j-1} = 0$ as claimed.

If p^m does not divide j, then

$$\binom{i}{j} = \frac{p^m\alpha}{p^s\beta}\binom{i-1}{j-1} \in p\mathbb{Z},$$

so $\mathbf{b}_{i,j} = 0$ and $\mathbf{b}_{i-1,j} = -\mathbf{b}_{i-1,j-1}$. ∎

6. Computing Leading Coefficients and Testing Membership

Proposition 5.1(c) gives an easy method for computing the coefficients $\phi(k)$ and λ required by the normalizer and conjugator update algorithms.

In the main loops of the algorithms we are given elements $x_k^{-1}[h_{i+1}, m_k]$ in K_j and must find constants $\phi(k) \in \{0, 1, \dots, p-1\}$ such that

$$x_k^{-1}[h_{i+1}, m_k] \equiv z_{j+1}^{\phi(k)} \bmod K_{j+1}.$$

Given j, the first step is to compute r such that $F_r \geq K_j \geq K_{j+1} \geq F_{r+1}$, i.e., such that K_j and K_{j+1} correspond to K-invariant subspaces $\langle \mathbf{b}_s, \mathbf{b}_{s+1}, \dots \rangle$ and $\langle \mathbf{b}_{s+1}, \dots \rangle$ of $V_r := F_r/F_{r+1}$, with z_{j+1} corresponding to $\mathbf{b}_s$. For convenience, assume that $F_{r+1} = 1$, and let $\mathbf{v} := x_k^{-1}[h_{i+1}, m_k]$. Then $\mathbf{v} = \phi(k)\mathbf{b}_s + \mathbf{u}$ with $\mathbf{u} \in \langle \mathbf{b}_{s+1}, \dots \rangle = K_{j+1}$. By Proposition 5.1(c),

$$\mathbf{b}_{p^r-s-1} \cdot \mathbf{v} = \phi(k)\mathbf{b}_{p^r-s-1} \cdot \mathbf{b}_s \equiv (-1)^s\phi(k) \mod p,$$

so to compute $\phi(k)$ we need only take the dot product of $\mathbf{v}$ with an appropriate row of $\mathbf{B}_r$. The computation of λ in the conjugator update algorithm goes the same way; view $ue_{i+1}^y h_{i+1}$ as an element of $\langle \mathbf{b}_s, \dots \rangle$ and compute its dot product with $\mathbf{b}_{p^r-s-1}$ to get $(-1)^s\lambda$.

In our implementations of these algorithms we actually carry along the auxiliary elements $x_k^{-1}[h_{i+1}, m_k]$ and $ue_{i+1}^y h_{i+1}$, rather than x_k and u. Unless the values of these elements change in going from j to $j+1$ or the increase in j causes a level descent in the structure forest, the vectors associated with these elements do not need to be recomputed in the inner loop, thus saving a significant amount of work. Even in the case $H_j \neq H_{j+1}$, in which x_k changes, the new vector value associated with $x_k^{-1}[h_{i+1}, m_k]$ is easy to compute from the current value and the (stored) vector for h_{j+1}.

Proposition 5.1(c) also gives a test for membership in K_j, since if $\mathbf{u} \in V_r$, then

$$\mathbf{u} \in \langle \mathbf{b}_s, \mathbf{b}_{s+1}, \dots \rangle \quad \text{iff} \quad \mathbf{u} \cdot \mathbf{b}_\beta = 0 \text{ for } \beta \geq p^r - s.$$

On each pass through the main loop of the update algorithm the value of j increases by 1. Thus the test vectors for V_r run through $\mathbf{b}_{p^r-1}, \mathbf{b}_{p^r-2}, \dots, \mathbf{b}_0$. By Proposition 5.2,

```
{input: an implicit CGS z_1,...,z_L for a p-group K,
        an SGS h_1,...,h_L for its subgroup H, and
        An IGS m_1,...,m_t for its subgroup G.}
{output: An IGS for G ∩ H.}
begin
{Initialize.}
for k := 1 to t do
    x_k := 1 ∈ H,
for j := 0 to L − 1 do
    {(m_1,...,m_t) is an IGS for G ∩ (HK_j),
     x_k ∈ H and x_k m_k ∈ K_j for k = 1,...,t}
    for k := 1 to t do
        Compute φ(k) ∈ Z_p with x_k m_k ≡ z_{j+1}^{φ(k)} mod K_{j+1}
    if h_{j+1} = 1 then
      if φ(k) ≠ 0 for some k then
        s := max{k: φ(k) ≠ 0}
        for k := 1 to s − 1 do
            Solve φ(s)α(k) + φ(k) = 0 for α(k) ∈ Z_p
            m_k := m_k m_s^{α(k)}; x_k := x_s^{α(k)} x_k
        t := t − 1
        for k := s to t do
            m_k := m_{k+1}; x_k := x_{k+1}
    else {HK_j = HK_{j+1}}
      for k := 1 to t do
          x_k := h_{j+1}^{−φ(k)} x_k
return (m_1,...,m_t)
end.
```

Figure 6. Subgroup Intersection.

if space is at a premium the complete matrix **B** need not be stored in order to implement the algorithm.

7. Intersection and Centralizer

The overall outline of the normalizer update algorithm can be modified to yield algorithms for computing $G \cap H$ and $C_G(h)$ for h in H. Figures 6 and 7 illustrate such algorithms. Again, the linear structure for $\langle G, H\rangle$ can be used to compute the necessary leading coefficients quickly. The resulting algorithms have one less level of nested looping than the full normalizer and conjugator algorithms, so they have correspondingly faster running times, once an SGS for G has been set up (and, if necessary, one for H as well).

The element centralizer algorithm yields a set centralizer algorithm: starting with $M := G$, go through the elements of the set X one by one, for each x in X replacing M by $C_M(x)$ until finally $M := C_G(X)$. Element Centralizer produces the SGS's for the replacement groups. If H is a subgroup of K and X is a set of generators for H (perhaps an SGS), the resulting algorithm computes the subgroup centralizer $C_G(H)$.

If a linear structure already set up for another purpose has produced output in the form of an SGS, that output can be used as input to these algorithms without additional setup costs. We note also that neither of these algorithms is tied to the permutation group context. Both are valid in a more general nilpotent setting; they simply require a structure that corresponds to a chief series for $\langle G, H\rangle$, together with some method

{**input:** *An implicit CGS* $z_1, \ldots, z_L$ *for the parent* p*-group* K,
an SGS $m_1, \ldots, m_L$ *for the subgroup* G, *and*
an element h *of* K.}
{**output:** *An SGS for* $C_G(h)$.}

begin
if $h \in K_{L-1}$ **then return** $(m_1, \ldots, m_L)$
for $i := 1$ **to** $L-1$ **do**
  {$(m_1, \ldots, m_L)$ *is an SGS for* $C_G(hK_i/K_i)$}
  for $k := 1$ **to** i **and** $m_k \neq 1$ **do**
    Compute $\phi(k) \in \mathbb{Z}_p$ with $[h, m_k] \equiv z_{i+1}^{\phi(k)} \bmod K_{i+1}$
  if $\phi(k) \neq 0$ for some k **then**
    $s := \max\{k: \phi(k) \neq 0\}$
    for $k := 1$ **to** $s-1$ **do**
      Solve $\alpha(k)\phi(s) + \phi(k) = 0$ for $\alpha(k) \in \mathbb{Z}_p$
      $m_k := m_k m_s^{\alpha(k)}$
    $m_s := 1$
return $(m_1, \ldots, m_L)$
end.

Figure 7. Element Centralizer.

for computing leading coefficients. In particular, the chief series could be taken to be a refinement of the lower p-central series, as in a special AG-presentation of the type considered in Eick (1993).

We omit the verifications of these algorithms, which are similar to those for the update algorithms above.

The subgroup intersection algorithm is very like a noncommutative version of the well-known Zassenhaus sum-intersection algorithm for vector spaces. In our implementation we carry along auxiliary elements $x_k m_k$ rather than x_k, perhaps saving some time over straightforward implementations of the Zassenhaus algorithm, which must simultaneously compute $\langle G, H \rangle$ as well as $G \cap H$.

Presenting the centralizer algorithm in terms of an SGS for $C_G(h)$ illustrates one way in which group-theoretic knowledge can save time. The range for k can be restricted to $\{1, \ldots, i\}$, and could be further restricted if one had additional information about the location of h in a central series for K, as one might in the special AG-presentation setting.

8. Complexity

We use n, the degree of the permutation group K, as a measure of input size for our algorithms. If $n = p^t$, then the composition length of a p-Sylow subgroup of S_n is $(n-1)/(p-1)$, which for fixed p is roughly proportional to n. At the other extreme, a cyclic group of order p^t has minimum degree $n = p^t$ and composition length $\log_p n$. In any case, a p-subgroup of S_n has composition length less than n.

In addition to bookkeeping operations and arithmetic mod p, our algorithms require multiplying, taking inverses and taking small powers (g^e with $0 < e < p$) of permutations. The first two of these types of operations can clearly be carried out in $O(n)$ time. The powers can be computed cycle by cycle. The image under g^e of one element ω in a nontrivial cycle Γ of g can be found in time $O(e)$, which is at most linear in the length of Γ (a power of p). By using $(\omega^g)^{g^e} = (\omega^{g^e})^g$, the images under g^e of the other elements of Γ can be successively computed, each in constant time. Hence the restriction of g^e to Γ can be computed in time linear in the length of Γ, so g^e itself is computable in $O(n)$ time.

Computation of $N_G(H)$ with the normalizer algorithm breaks into two parts: setting up the linear structure, and executing the algorithm. The setup phase consists of preparing the linear structure for $K = \langle G, H\rangle$, and computing generating sequences to describe the chief series for G and H. Preparing the linear structure includes determining the maximal block decompositions that give the parameters for the structure forest, and computing the necessary matrices $\mathbf{B}_r$. These steps entail just $O(n^3)$ group operations, and hence $O(n^4)$ time. Implementation details may be found in GAP and Magma programs available from the authors. Rákóczi (1995) has given a theoretical account of the construction of the structure forest, as well as fast recognition algorithms for permutation p-groups and nilpotent groups. In practice, much of the setup time goes into building the generating sequences, which we can do by a variation of the Sims-Schreier procedure, as organized by Knuth (1991). In the worst case, each sequence requires time $O(n^4)$—less than in Knuth's analysis since we have fewer than $2n/p$ subgroups in the chain, each with p cosets, whereas in Knuth's situation each of these quantities is n in the worst case.

The normalizer and conjugator algorithms themselves consist essentially of two nested loops, each of length at most n, within which are two loops of length at most n. The bodies of the innermost loops are each made up of a small number of group operations, perhaps combined with computation of the dot product mod p of two sequences of length at most n/p. Thus the normalizer and conjugator algorithms each require just $O(n^4)$ time.

As we noted in Section 7, the intersection and element centralizer algorithms have one less level of looping than the normalizer algorithm. They require just $O(n^3)$ time after the setup phase. The subgroup centralizer algorithm can compute $C_G(H)$ in $O(n^4)$ time, since one can compute an SGS of length at most n for H during the setup and then use it as a set of generators for H. In practice, of course, H might well be given by a much smaller set of generators.

9. Implementation and Experiments

We have written implementations of the algorithms described above in GAP (Schönert, et al., 1993) and in Magma (Cannon and Playoust, 1993). In addition, we have written programs to construct the linear structure and the corresponding input composition series required by the algorithms.

Running times for our implementations of the normalizer algorithm reflect the structure of the algorithm. When n is small and G and H have very small composition lengths, time for the setup phase is a substantial fraction of the total. The time to compute $N_G(H)$ itself with our methods increases roughly in proportion to the composition length of H, with successive passes through the j-loop generally taking longer and longer, influenced, however, by reductions in the length of the ambient normalizer M. Running time is loosely coupled to the composition length of $N_G(H)$, as well as to the length of G.

Our algorithms are based on composition series, so composition lengths are reflected in running times. Algorithms whose fundamental structures are different, such as the backtrack algorithms of Leon (1991), may be expected to show markedly different behavior from ours; in particular, their running times may be influenced by factors such as the number of generators for G or for H. The timings we give below, which compare our normalizer implementations with the generic permutation group normalizer functions in GAP and Magma, appear to exhibit such differences, and should be taken only as rough indicators, not as refined comparisons. They do show, however, that our methods are practical for groups of degree large enough to cause difficulty for the generic programs.

In tests of our algorithms using earlier versions of GAP our methods were often as much as several hundred times faster than the generic functions, even for comparatively small degrees, where we might expect setup times to put them at a disadvantage. Subsequent improvements in the GAP library functions have raised the threshhold degrees at which we can expect our programs to significantly outperform the built-in functions.

In addition to the generic Normalizer function for permutation groups, GAP offers the possibility of converting a polycyclic permutation group to a group with an Ag presentation, to which the Ag group Normalizer function (based upon the method of Glasby and Slattery, 1990) can be applied. To compare our methods with conversion to the Ag setting, we first applied the GAP function AgGroup to the group $\langle G, H\rangle$, obtaining the embedding of G and H in the resulting Ag group by a specially adapted variant of the PreImage function. After determining $N_G(H)$ as an Ag group, we used the Image function to lift the answer back to a permutation group.

Table 1 shows the results of some typical experiments with GAP, and compares our normalizer implementation with the two alternative methods. The timings were obtained on a 486/DX 50Mhz PC running FreeBSD with an initial allocation of 8 MB of memory to GAP. In some instances, GAP increased the memory allocation during the course of the experiment. The tests were run with GAP Version 3.5 (unreleased) which at the time of the experiments incorporated some of Leon's ideas in its generic permutation Normalizer function and also had an implementation by Theissen of an improved AgGroup function for permutation groups.

Each column in Table 1 describes an experiment for one choice of G and H. We use the notation $\ell(X)$ to denote the composition length of the p-group X; thus $\ell(X) = \log_p |X|$. Running times are given in seconds of cpu-time as reported by the GAP function Runtime, rounded to the nearest second. The row labeled SETUP shows the time to construct the composition series for G and for H with our methods. The LINEAR row gives the total time for our method, so the difference between these two rows gives the time for our normalizer program alone. We have presented the Ag group data slightly differently; AG gives the total time for the over-and-back process, while PURE indicates the time for the Ag group Normalizer computation alone. Table 1 shows the Ag group time to be dominated by the conversion process, but even the PURE figures are commonly as high as the total LINEAR times. Finally, the PERM row gives times for GAP's generic permutation group Normalizer function.

The groups for these tests were generated in several ways as subgroups of a random Sylow p-subgroup of S_n. In the first five cases, we forced $H \leq G$; indeed G is the full Sylow group in the fourth and fifth examples. Otherwise, the test groups were generated by choosing small numbers of random generators in the Sylow group, or by intersecting two subgroups generated in that way in order to get larger numbers of generators while maintaining reasonable size. The results shown are typical, especially for our methods, of those produced in a number of trials. The backtrack-based permutation group Normalizer function can take times differing by factors of as much as a hundred for groups that appear essentially similar, so timing results for that method show considerable variance.

Because the Magma function Normalizer required H to be a subgroup of G, we limited Magma tests to that setting, rather than computing $G \cap N_{\langle G,H\rangle}(H)$ more generally. Like

Table 1. GAP

Example	1	2	3	4	5	6	7	8
p	2	2	2	2	2	2	3	5
n	100	100	100	100	100	150	150	200
$\ell(G)$	71	79	79	97	97	136	62	41
$\ell(H)$	47	5	52	4	84	135	54	33
$\ell(N_G(H))$	56	17	65	32	91	133	55	33
SETUP	40	68	79	9	84	858	67	19
LINEAR	127	79	228	19	410	3864	240	72
PURE	64	483	150	36	157	11946	155	168
AG	330	1702	661	603	780	69325	1979	956
PERM	11370	412	340	1763	332	2658	1944	>57300

Table 2. $p = 2$ MAGMA Smaller Degrees

Example	1	2	3	4	5	6	7	8	9
n	24	24	24	32	32	50	50	50	50
$\ell(G)$	15	16	17	27	27	36	36	47	47
$\ell(H)$	13	14	6	17	18	24	27	34	39
$\ell(N_G(H))$	14	15	11	19	19	26	29	40	45
SETUP	4	3	3	19	18	36	38	11	16
LINEAR	16	15	8	41	45	125	142	45	57
PERM	0.2	0.3	15	2723	1	>7000	4166	>1050	1

GAP, Magma affords the opportunity to convert solvable permutation groups to groups with polycyclic presentations. In view of our experience with GAP and our other Magma results, we have not carried out tests in that direction with Magma.

The results of our experiments using Magma V1.01 on a Sparc Station ELC with 24 MB of RAM are consistent with those we obtained with GAP. Because of differences in the computer environments in which they were run, however, the timings should not be viewed as providing a meaningful comparison between the GAP and Magma generic normalizer functions. Moreover, as Tables 2 and 3 suggest, there was great variation among the Magma times for groups that appeared outwardly similar. Changes incorporated in the Magma algorithms since these tests were run would presumably now give still different numbers, and increase the speeds of both the generic normalizer function and our own implementation.

Table 3. $p = 2$ MAGMA Larger Degrees

Example	1	2	3	4	5	6	7	8
n	100	100	100	100	100	100	100	100
$\ell(G)$	47	47	48	48	71	71	79	79
$\ell(H)$	4	36	11	22	4	43	5	52
$\ell(N_G(H))$	44	40	41	30	19	56	17	65
SETUP	108	161	47	49	125	171	460	637
LINEAR	159	693	137	234	219	828	571	1527
PERM	7617	>52000	>5300	>3600	>7500	>4900	>19000	>4800

Row headings in the Magma tables have similar meanings to those in Table 1. Here the row headed PERM gives times for the Magma Normalizer function. Where PERM times are given as "$> x$", program execution was halted after x seconds. The tables describe experiments for $p = 2$. Tests with $p = 3$ produced similar results. Examples 6, 7 and 8 in Table 3 are the same as Examples 1, 2 and 3 in Table 1.

The Magma tests for $n = 100$ were designed to note the effect on running times of the numbers of generators of G and of H. Groups with large numbers of generators were produced by intersecting groups generated by small numbers of random elements. We also ran tests for $p = 2$ and $n = 100$ with G cyclic. Though our program took less than 10 seconds in such cases, the Magma Normalizer function was typically substantially faster yet.

The GAP and Magma implementations of our conjugator algorithm exhibit running times consistent with those for the normalizer algorithm. Setup times for conjugator reflect the need to compute the additional generating sequence for E, while execution times for the algorithm itself are only slightly longer than those for normalizer.

We have also run tests of our intersection and centralizer implementations to compare them with built-in GAP and Magma functions based on the methods of Leon (1991). As expected, setup time dominates overall running time for our intersection and element centralizer algorithms. Comparisons with backtrack-based programs are difficult. In one instance, conjugating a 2-group G of degree 50 by a random permutation in S_{50} turned an example for which our implementation found $C_G(G)$ 100 times faster than the GAP built-in function into an example for which our program was 10 times slower. In the range of degrees we considered, the times for our intersection and centralizer algorithms were typically greater than times for the corresponding built-in permutation group functions in Magma.

Other polynomial time approaches to finding centralizers and intersections in nilpotent groups are known (see, e.g., Luks, 1982 and 1993). Although the methods we describe here give asymptotically fast algorithms, at least two of the authors conjecture that further work will yield centralizer and intersection methods with even faster implementations.

References

Atkinson, M.D. (1975). An algorithm for finding the blocks of a permutation group. *Math. Comp.* **29**, 911–913.

Butler, G., Cannon, J.J. (1993). On Holt's algorithm. *J. Symb. Comp.* **15**, 229–233.

Butler, G. (1983). Computing normalizers in permutation groups. *J. Algorithms* **4**, 163–175.

Cannon, J., Playoust, C. (1993). *An Introduction to MAGMA*. School of Mathematics and Statistics, University of Sydney.

Celler, F., Neubüser, J., Wright, C.R.B. (1990). Some remarks on the computation of complements and normalizers in soluble groups. *Acta Applic. Math.* **21**, 57–76.

Eick, B. (1993). *Spezielle PAG-Systeme im Computeralgebrasystem GAP.* Diplomarbeit, Rheinisch-Westfälische Technische Hochschule, Aachen, Germany.

Glasby, S.P., Slattery, M.C. (1990). Computing intersections and normalizers in soluble groups. *J. Symb. Comp.* **9**, 637–651.

Holt, D.F. (1991). *The computation of normalizers in permutation groups,* J. Symb. Comp, 12 (1991), 498–516.

Kantor, W.M., Luks, E.M. (1990). Computing in quotient groups. *Proc. 22*$^{\text{nd}}$ *ACM Symposium on Theory of Computing*, 524–533.

Knuth, D.E. (1991). Efficient representation of perm groups. *Combinatorica* **11**, 33–43.

Leedham-Green, C.R., McKay, S., Plesken, W. (1986). Space groups and groups of prime-power order, V. A bound to the dimension of space groups with fixed coclass. *Proc. London Math. Soc. (3)* **52**, 73–94.

Leedham-Green, C.R., Newman, M.F. (1980). Space groups and groups of prime-power order I. *Archiv d. Math.* **35**, 193–202.

Leon, J.S. (1991). Permutation group algorithms based on partitions I: Theory and algorithms. *J. Symb. Comp.* **12**, 533–583.

Luks, E.M. (1982). Isomorphism of graphs of bounded valence can be tested in polynomial time. *J. Comp. Syst. Sci.* **25**, 42–65.

Luks, E.M. (1986). Parallel algorithms for permutation groups and graph isomorphism, *Proc. 27*th *IEEE Symp. on the Foundations of Comp. Sci.*, 292–302.

Luks, E.M. (1992). Computing in solvable matrix groups. *Proc. 33*rd *IEEE Symp. on the Foundations of Comp. Sci.*, 111–120.

Luks, E.M. (1993). Permutation groups and polynomial-time computation. *Groups and Computation, DIMACS Series in Discrete Mathematics and Theoretical Computer Science* **11**, Amer. Math. Soc., ed. L. Finkelstein and W. Kantor, 139–175.

Luks, E.M., McKenzie, P. (1988). Parallel computation in solvable permutation groups. *J. Comp. Syst. Sci.* **37**, 39–62.

Rákóczi, F. (1995). Fast recognition of the nilpotency of permutation groups. *Proc. 1995 International Symp. on Symbolic and Algebraic Computation*, 265–269.

Schönert, M., et al., (1993). *GAP, Groups, Algorithms and Programming*. Lehrstuhl D für Mathematik, Rheinisch-Westfälische Technische Hochschule, Aachen, Germany, 3rd edition.