Algorithms for Polycyclic-by-Finite Groups[☆]

S.K. Sinanan[b,*], D.F. Holt[a]

[a] *Mathematics Institute, University of Warwick, Coventry CV4 7AL, United Kingdom*
[b] *Department of Mathematics and Statistics, University of The West Indies, Saint Augustine, Trinidad and Tobago*

Abstract

A set of fundamental algorithms for computing with polycyclic-by-finite groups is presented.

Polycyclic-by-finite groups arise naturally in a number of contexts; for example, as automorphism groups of large finite soluble groups, as quotients of finitely presented groups, and as extensions of modules by groups. No existing mode of representation is suitable for these groups, since they will typically not have a convenient faithful permutation representation.

A mixed mode is used to represent elements of such a group, utilising either a power-conjugate presentation or a polycyclic presentation for the elements of the normal subgroup, and a permutation representation for the elements of the quotient.

Keywords: polycyclic, polycyclic-by-finite, permutation, finitely presented, magma

1. Introduction

A group is *polycyclic-by-finite* if it has a normal polycyclic subgroup of finite index. That is, if it has a normal subgroup of finite index that admits a subnormal series with cyclic factors.

By a well-known theorem of P. Hall, every polycyclic-by-finite group is finitely presented — and in fact, polycyclic-by-finite groups form the largest known section-closed class of finitely presented groups. It is this fact that makes

[☆]This research was supported by the University of Warwick Postgraduate Research Scholarship, and the University of Warwick Chancellor's International Scholarship, which are, in turn, funded partly by the Engineering and Physical Sciences Research Council of the United Kingdom (EPSRC).

*Corresponding author

Email addresses: Shavak.Sinanan@sta.uwi.edu (S.K. Sinanan), D.F.Holt@warwick.ac.uk (D.F. Holt)

URL: https://sta.uwi.edu/fst/dms/Shavak_Sinanan.asp (S.K. Sinanan), http://homepages.warwick.ac.uk/~mareg/ (D.F. Holt)

polycyclic-by-finite groups natural objects of study from the algorithmic standpoint.

The algorithmic decision theory of polycyclic-by-finite groups has been investigated in the theoretical context by Baumslag et al. (1991). However, from the computational standpoint, the algorithms presented by Baumslag et al. (1991) are not applicable, nor were they intended to be. In contrast, this paper explores the computational properties of polycyclic-by-finite groups from a practical perspective, detailing algorithms which lend themselves easily to computer implementation.

Specifically, the work presented in this paper aims to:

(a) Define a computationally effective representation for polycyclic-by-finite groups.
(b) Develop a set of implementable algorithms to perform fundamental computations such as element multiplication and subgroup construction, within the class of polycyclic-by-finite groups.
(c) Use the fundamental algorithms developed to design methods that perform more advanced computations such as the construction of centralisers and conjugacy testing, within the class of polycyclic-by-finite groups.

The algorithms presented here are targeted primarily at finite non-solvable groups with a large solvable (and hence, in this case, polycyclic) normal subgroup, as such groups often do not have a convenient permutation representation. Groups of this type arise naturally in many applications, such as automorphism groups of large finite solvable groups, as quotients of finitely presented groups, and as extensions of modules by groups.

The theory developed is by no means limited to the finite case. Apart from a few natural exceptions (such as computing Sylow subgroups), all of the algorithms apply equally to infinite polycyclic-by-finite groups.

Whilst there may be many different decompositions of a given polycyclic-by-finite group as an extension of a polycyclic group by a finite group, the algorithms presented in this paper are most useful for those decompositions in which the finite quotient admits a faithful permutation representation of manageable degree ($< 10^6$). Thus, it will hereinafter be assumed that the polycyclic-by-finite groups in question can be so decomposed.

Much of what is presented in the sequel requires familiarity with the already existing efficient algorithms for computing with finite permutation groups, and with polycyclic groups. The standard references for computation with permutation groups are (Sims, 1970, 1971), and (Seress, 2003). The notation used in this paper is consistent with the latter.

Polycyclic groups form a broad class of finitely presented groups in which extensive computation is possible. In the finite case, solvability is equivalent to polycyclicity, and the literature on algorithms for computing with finite solvable groups is extensive. See (Laue et al., 1984; Mecky and Neubüser, 1989; Glasby and Slattery, 1990). Computing with infinite polycyclic groups is more cutting-edge; for excellent accounts, see (Sims, 1994, ch. 9), (Eick, 2001), and (Holt et al., 2005, ch. 8). The notation used in this paper is consistent with the latter.

Let E be a polycyclic-by-finite group, and suppose that there is a black-box representation of E on the computer. Let $N \trianglelefteq E$ be polycyclic, and assume that $G = E/N$ admits a faithful permutation representation of manageable degree. Concretely, the central goal of the theory is to set up machinery so that elements of E can be manipulated by performing operations only within N and G, without appealing to the existing representation of E.

On the other hand, although the algorithms described here are designed for situations where one is unable to perform arithmetic in E, there may nevertheless be instances in which it would be efficient to be able to do this; and similar methods to those presented here may be applied to such groups. This happens, for example, in dealing with some types of large matrix groups over finite fields as described in (Hulpke, 2013).

2. Multiplication

This section contains a detailed description of a mode by which elements of a given polycyclic-by-finite group may be represented on the computer, and a strategy for multiplication of elements represented in this manner.

Fix a transversal L of N in E with $1_E \in L$ and, for $e \in E$, denote by $\overline{e}$ the unique element of $eN \cap L$. An element $e \in E$ can of course be written uniquely as a product $e = \overline{e} \cdot n$, where $n \in N$.

Note. For ease of notation, in what follows, the element $\overline{e} \in L$ will be identified with the element eN of the quotient G; the context will be made explicit in cases where it is not clear.

Expressing group elements in this manner is a logical approach as one would naturally wish to utilise the well-developed algorithms available for the classes of permutation groups and polycyclic groups, thereby fully exploiting the structure of the group in question.

In what follows it is assumed that the normal subgroup N of E is defined by a polycyclic presentation, although the methods described can be extended to different representations of N.

2.1. Bases and Strong Generating Sets

The definitions given in (Seress, 2003, ch. 4) for a base and a strong generating set may be extended to fit the context of polycyclic-by-finite groups.

View G as a permutation group of degree d and denote by Ω the set of points on which G acts. Given an element $\overline{e}n \in E$ and a point $\omega \in \Omega$ one may unambiguously speak of the action of $\overline{e}n$ on ω, with obvious meaning, viz. $\omega^{\overline{e}n} = \omega^{\overline{e}}$; regarding $\overline{e}$ as an element of G.

The fundamental algorithms for orbit computation can be applied without modification in the case of this permutation action. Specifically, it is possible, using the methods described in (Seress, 2003, ch. 4), to compute the orbit of a point $\omega \in \Omega$ under a set $X \subseteq E$ of elements along with a Schreier vector encoding a transversal of the stabiliser $\langle X \rangle_\omega$ in $\langle X \rangle$.

A sequence $B = (\gamma_1, \ldots, \gamma_k)$ of elements belonging to Ω is called a *base* for E if every element of E that fixes B pointwise belongs to N. The sequence B defines a stabiliser subgroup chain

$$E = E^{[1]} \geqslant E^{[2]} \geqslant \cdots \geqslant E^{[k]} \geqslant E^{[k+1]} = N \tag{2.1}$$

where $E^{[i]} = E_{(\gamma_1,\ldots,\gamma_{i-1})}$ $(i > 1)$ is the pointwise stabiliser of $\{\gamma_1, \ldots, \gamma_{i-1}\}$. The base B is called *non-redundant* if $E^{[i+1]} < E^{[i]}$ for all $i = 1, \ldots, k$. The orbits ${\gamma_i}^{E^{[i]}}$ are called the *basic orbits* or *fundamental orbits* of E (relative to B). The subgroup $E^{[i]}$ is called the i-th basic stabiliser relative to B. A *strong generating set* for E relative to B is a generating set T for E with the property that

$$\langle T \cap E^{[i]} \rangle = E^{[i]} \tag{2.2}$$

for $i = 1, \ldots, k + 1$.

One has

$$|G| = \prod_{i=1}^{k} [E^{[i]}/N : E^{[i+1]}/N].$$

By the Third Isomorphism Theorem and the Orbit–Stabiliser Theorem, one obtains $[E^{[i]}/N : E^{[i+1]}/N] = |\gamma_i^{E^{[i]}}| \leqslant d$ for $i = 1, \ldots, k$. Moreover, if B is non-redundant, then $[E^{[i]}/N : E^{[i+1]}/N] \geqslant 2$ for each i. These inequalities, combined with the expression for $|G|$ above, yield

$$\frac{\log_2 |G|}{\log_2 d} \leqslant |B| \leqslant \log_2 |G|. \tag{2.3}$$

2.2. The Normal Form

A base and strong generating set data structure for the permutation group G combined with a power-conjugate or polycyclic presentation for N automatically induce a normal form for elements of E. The existence of a normal form for group elements allows one to devise a multiplication strategy which is suitable for computer implementation.

Let $S = \{\overline{x_1}, \ldots, \overline{x_m}\}$ be a strong generating set relative to a base $B = (\beta_1, \ldots, \beta_l)$ for G, and assume that S is non-redundant. Denote by S^{-1} the set $\{\overline{x_1^{-1}}, \ldots, \overline{x_m^{-1}}\}$. For technical reasons, the transversal L is constructed so that, for each $\overline{x} \in S$ which is not self-inverse in G, one has $\overline{x^{-1}} = \overline{x}^{-1}$.

For each i, denote the i-th basic stabiliser relative to B by $G^{[i]}$, and denote the i-th basic orbit relative to B by $\Delta_i = \{\delta_{i,1}, \ldots, \delta_{i,d_i}\}$, where $\delta_{i,1} = \beta_i$. Let U_i be a right transversal of $G^{[i+1]}$ in $G^{[i]}$; then each element of G can be represented uniquely as a product of transversal elements $\overline{u_l} \cdot \overline{u_{l-1}} \cdots \overline{u_1}$ where $\overline{u_i} \in U_i$.

Additionally, let $S_i = S \cap G^{[i]} = \{\overline{x_{i,1}}, \ldots, \overline{x_{i,s_i}}\}$ and denote by S_i^{-1} the set $\{\overline{x_{i,1}^{-1}}, \ldots, \overline{x_{i,s_i}^{-1}}\}$. In what follows, one assumes that there is a data structure that encodes elements of the basic stabilisers as words over $S \cup S^{-1}$ (such as

a *Schreier vector* — see (Seress, 2003)), and that this data structure remains unchanged throughout the operation of the method.

An element $e \in E$ may be expressed as $\overline{u_l} \cdot \overline{u_{l-1}} \cdots \overline{u_1} \cdot n_1$ where $\overline{u_i} \in U_i$. Consider multiplying the elements e and $\overline{x} \cdot n_2$ (where $\overline{x} \in S \cup S^{-1}$ and $n_2 \in N$):

$$\overline{u_l} \cdot \overline{u_{l-1}} \cdots \overline{u_1} \cdot n_1 \cdot \overline{x} \cdot n_2 = \overline{u_l} \cdot \overline{u_{l-1}} \cdots \overline{u_1} \cdot \overline{x} \cdot n_1^{\overline{x}} \cdot n_2.$$

In this setup, the multiplication algorithm should possess the following functionality:

(i) Conjugate elements of N by elements $\overline{x} \in S \cup S^{-1}$.
(ii) Rewrite an expression of the form $\overline{u_l} \cdot \overline{u_{l-1}} \cdots \overline{u_1} \cdot \overline{x}$ as $\overline{u'_l} \cdot \overline{u'_{l-1}} \cdots \overline{u'_1} \cdot n$ where $u'_i \in U_i$ and $n \in N$.

Utilising the existing representation of N, items (i) and (ii) are sufficient to formulate a strategy by which arbitrary elements of E may be multiplied without appealing to the existing representation of E. The details of how this is accomplished are given in Subsections 2.3–2.4.

2.3. The Shifting Method

Fix i and j, and let $\overline{u} \in U_i$ be the permutation taking β_i to $\delta_{i,j} \in \Delta_i$. Take $\overline{x} \in S_i$, and let $\overline{h}$, $\overline{h'}$ be the permutations in U_i which map β_i to $\beta_i^{\overline{u} \cdot \overline{x}} = \delta_{i,j}^{\overline{x}}$, $\beta_i^{\overline{u} \cdot \overline{x^{-1}}} = \delta_{i,j}^{\overline{x^{-1}}}$ respectively. Then, in the group E, one has

$$\overline{u} \cdot \overline{x} = \overline{y_1}\,\overline{y_2} \cdots \overline{y_k} \cdot \overline{h} \cdot n, \tag{2.4a}$$

$$\overline{u} \cdot \overline{x^{-1}} = \overline{z_1}\,\overline{z_2} \cdots \overline{z_{k'}} \cdot \overline{h'} \cdot n', \tag{2.4b}$$

for some $n, n' \in N$, and where $\overline{y_1}, \ldots, \overline{y_k}, \overline{z_1}, \ldots, \overline{z_{k'}} \in S_{i+1} \cup S_{i+1}^{-1}$.

The elements n and n' are called the *heads* of $\overline{u}$ relative to $\overline{x}$ and $\overline{x^{-1}}$ respectively while the words $\overline{y_1}\,\overline{y_2} \cdots \overline{y_k}$ and $\overline{z_1}\,\overline{z_2} \cdots \overline{z_{k'}}$ are called the *tails* of $\overline{u}$ relative to $\overline{x}$ and $\overline{x^{-1}}$ respectively. Equations (2.4) are called the *shift equations*. Note that there are $O(d|B||S|)$ shift equations.

The shift equations suggest a scheme by which elements of E may be multiplied. Assume that the tails can be computed consistently for each pair $\overline{u}, \overline{x}$, and $\overline{u}, \overline{x^{-1}}$ of the shift equations. Furthermore, assume that the conjugates of elements of N by elements of $S \cup S^{-1}$ can be calculated without appealing to the existing representation of E.

Let $e_1 = \overline{e_1} n_1$ and $e_2 = \overline{e_2} n_2$ be two elements of E in normal form. Using the methods available for permutation groups, the algorithm begins by writing $\overline{e_1}$ as $\overline{u_l}\,\overline{u_{l-1}} \cdots \overline{u_1}$, where $\overline{u_i} \in U_i$ for each i, and expresses $\overline{e_2}$ as a word over $\overline{S} \cup \overline{S^{-1}}$, say $w = \overline{q_1}\,\overline{q_2} \cdots \overline{q_\sigma}$ where $\overline{q_j} \in S \cup S^{-1}$ for each j. (Note that the existing methods available for permutation groups allow one to do this consistently.)

The algorithm then proceeds to rearrange the terms of the product by conjugating the element n_1 by the word w, as illustrated in Equation (2.5):

$$\begin{aligned} e_1 \cdot e_2 &= (\overline{u_l}\,\overline{u_{l-1}} \cdots \overline{u_1}) \cdot n_1 \cdot (\overline{q_1}\,\overline{q_2} \cdots \overline{q_\sigma}) \cdot n_2 \\ &= \overline{u_l}\,\overline{u_{l-1}} \cdots \overline{u_2}\ \underbrace{\overline{u_1} \cdot \overline{q_1}}\ \overline{q_2} \cdots \overline{q_\sigma} \cdot n_1^w n_2. \end{aligned} \tag{2.5}$$

The under-bracketed segment of Equation (2.5) can be recognised as the left-hand side of a shift equation, say $\overline{u_1} \cdot \overline{q_1} = \overline{y_1}\,\overline{y_2} \cdots \overline{y_\kappa} \cdot \overline{h_1} \cdot n_3$ for some $n_3 \in N$, $\overline{h_1} \in U_1$, and where $\overline{y_j} \in S_2 \cup S_2^{-1}$ for each j. Via direct substitution, this equation can be used to *shift* the transversal element $\overline{q_1}$ past $\overline{u_1}$. The algorithm executes this shifting procedure as illustrated in Equation (2.6).

$$\begin{aligned} e_1 \cdot e_2 &= \overline{u_l}\,\overline{u_{l-1}} \cdots \overline{u_2}\ \underbrace{\overline{u_1} \cdot \overline{q_1}}_{\text{replace by } \overline{y_1}\,\overline{y_2} \cdots \overline{y_\kappa} \cdot \overline{h_1} \cdot n_3}\ \overline{q_2} \cdots \overline{q_\sigma} \cdot n_1^w n_2 \\ &= \overline{u_l}\,\overline{u_{l-1}} \cdots \overline{u_2} \cdot \overline{y_1}\,\overline{y_2} \cdots \overline{y_\kappa} \cdot \underline{\overline{h_1} \cdot \overline{q_2}}\ \overline{q_3} \cdots \overline{q_\sigma} \cdot n_3^{\overline{q_1}^{-1} w} n_1^w n_2 \end{aligned} \tag{2.6}$$

The under-bracketed segment of the product in Equation (2.6) can again be recognised as the left-hand side of a shift equation. The algorithm repeats the procedure above to shift $\overline{q_2}$ past $\overline{h_1}$.

Remark 2.1. Each time a shift is made, the word over $S \cup S^{-1}$ immediately to the left of the sequence of normal subgroup elements in the expression for the product is reduced by exactly one symbol.

The algorithm continues iterating through the word w, shifting at every step, arriving at an expression of the form $\overline{u_l}\,\overline{u_{l-1}} \cdots \overline{u_2} \cdot \overline{y_1}\,\overline{y_2} \cdots \overline{y_\zeta} \cdot \overline{u_1'} \cdot n'$ for the product, where $\overline{u_1'} \in U_1$, $n' \in N$.

At this stage, the algorithm has decreased the sequence of transversal elements $\overline{u_i}$ on the right-hand side of Equation (2.5) by one term, thus reducing the problem to a smaller case. The algorithm restarts its inner loop to process the word $\overline{y_1}\,\overline{y_2} \cdots \overline{y_\zeta}$ by, as before, recognising $\overline{u_2} \cdot \overline{y_1}$ as the left-hand side of a shift equation.

Remark 2.2. Each time a word over $S \cup S^{-1}$ is processed fully as described above, the leftmost sequence of transversal elements $\overline{u_i}$ in the expression for the product is decreased by exactly one term.

Remark 2.3. Whenever a shift is made at level i (with respect to the base and strong generating set hierarchy), the elements of $S \cup S^{-1}$ that are placed to the left of the new element, $\overline{h_i}$ (where $h_i \in U_i$), belong to $S_{i+1} \cup S_{i+1}^{-1}$. In particular, since $S_{l+1} = \varnothing$, no non-trivial elements of $S \cup S^{-1}$ are placed to the left of an element of U_l.

The word-processing procedure is repeated for each $\overline{u_i}$, after which a series of conjugations of elements of the normal subgroup is performed. This yields an expression of the form $\overline{u_l'}\,\overline{u_{l-1}'} \cdots \overline{u_1'} \cdot n''$ for the product, where $n'' \in N$, and $\overline{u_i'} \in U_i$ for each i.

Remark 2.4. The shift equations (2.4), together with the conjugation equations $n\overline{x} = \overline{x} n^{\overline{x}}$ for $\overline{x} \in S \cup S^{-1}$ and generators n of N, and the relations in the polycyclic presentation of N together constitute a confluent terminating rewriting system for the group E.

2.4. The Multiplication Algorithm

The multiplication algorithm relies directly on a precomputed set of data to facilitate its execution. More specifically, the following data is computed (using the existing representation of E) on initialisation and held in memory.

(i) The tail of each shift equation. This computation takes place entirely within the permutation group. Assuming (henceforth) that a Schreier vector data structure is used, the storage required is $O(|S|(d|B|)^2)$, where d is the degree of the permutation group G.

(ii) The head of each shift equation. This computation uses the existing representation of E and requires $O(dr|B||S|)$ storage, where r is the length of the polycyclic sequence used to define N.

(iii) For each polycyclic generator a of N, and each $\overline{y} \in S \cup S^{-1}$, the conjugate $a^{\overline{y}}$. The memory requirement here is $O(r^2|S|)$.

The multiplication method is presented in Algorithm 1. The notation employed thus far is used in the pseudocode, and the method assumes that the tails of the shift equations are stored as arrays of strong generators. The method also assumes the existence of a function LENGTH which, when supplied with an array (or a sequence), returns the number of non-null entries of that array (or sequence).

Theorem 2.5. *The multiplication algorithm terminates with the correct value for the desired product.*

Proof. Termination is guaranteed by Remarks 2.1–2.3, while Equations (2.5) and (2.6) imply correctness. □

The running time of MULTIPLY is determined by the number of times that Lines 13–19 are executed. To estimate this number, an upper bound for LENGTH(*rightword*) will be established (for each of the l iterations of the outer loop).

The rate of growth of *rightword* is investigated as follows. Initially, *rightword* has length at most $d|B|$. Each shift made in the inner loop of the algorithm appends a tail of length at most $d|B|$ to the variable *leftword*. This operation is performed in Line 15. Since a shift is made for each element of *rightword*, it follows that, at the end of the first iteration of the outer loop, *leftword* has length $\leqslant (d|B|)^2$; and so, in the second iteration of the outer loop, *rightword* has length $\leqslant (d|B|)^2$. Repeating this argument shows that *rightword* has length $\leqslant (d|B|)^i$ in the i-th iteration of the outer loop. Thus, in a run of MULTIPLY, Lines 13–19 are executed $\leqslant \sum_{i=1}^{|B|} (d|B|)^i \in O\left((d|B|)^{|B|}\right)$ times.

The most expensive computations within Lines 13–19 occur in Line 16 and in Line 18. Line 16 consists of an application of a straightforward iterative procedure to compute the conjugate of an element of N by an element of $S \cup S^{-1}$ using the precomputed data set. Therefore, in a single run, MULTIPLY performs $O\left((d|B|)^{|B|}\right)$ such conjugations. Line 18 is a standard Schreier vector

Algorithm 1 Element multiplication

function MULTIPLY($\overline{e_1}n_1$, $\overline{e_2}n_2$)
  if $\overline{e_2}$ is trivial **then**
    return $\overline{e_1}n_1n_2$
  Write $\overline{e_1}$ in normal form $\overline{u_l}\,\overline{u_{l-1}}\cdots\overline{u_1}$ where $\overline{u_i} \in U_i$ for each i
  Write $\overline{e_2}$ as a word $w = \overline{q_1}\,\overline{q_2}\cdots\overline{q_t}$ where $\overline{q_i} \in S \cup S^{-1}$ for each i
  rightword $\leftarrow [\overline{q_1}, \overline{q_2}, \ldots, \overline{q_t}]$, *leftword* $\leftarrow []$
  $n'_1 \leftarrow n_1$
  for $i \leftarrow 2$ **to** l **do**
    $n'_i \leftarrow 1_N$
  for $i \leftarrow 1$ **to** l **do**
    $\overline{u'_i} \leftarrow \overline{u_i}$
    for $j \leftarrow 1$ **to** LENGTH(*rightword*) **do**
      Retrieve from memory the tail array, *tailword*, and head, n_σ, of
        the shift equation corresponding to the pair u'_i, *rightword*$[j]$
      leftword $\leftarrow$ *leftword* cat *tailword*
      Find the conjugate n_γ of n'_i by *rightword*$[j]$ ▷ Use the data set
      $n'_i \leftarrow n_\sigma \cdot n_\gamma$
      Find $\overline{h} \in U_i$ which maps $\beta_i^{u'_i}$ to the image of $\beta_i^{u'_i}$ under *rightword*$[j]$
      $\overline{u'_i} \leftarrow \overline{h}$
    rightword $\leftarrow$ *leftword*
    leftword $\leftarrow []$
  $\overline{e_*} \leftarrow 1_G$, $n_* \leftarrow 1_N$, $i \leftarrow l$
  while $i > 0$ **do**
    $n_* \leftarrow n_*^{\overline{u'_i}} \cdot n'_i$, $\overline{e_*} \leftarrow \overline{e_*} \cdot \overline{u'_i}$ ▷ Use the data set
    $i \leftarrow i - 1$
  $n_* \leftarrow n_* \cdot n_2$
  return $\overline{e_*}n_*$ ▷ Regard $\overline{e_*}$ now as an element of L

computation in G which requires $O(d)$ multiplications in the finite quotient G; for a proof of this, (see Seress, 2003, ch. 4). Thus, a run of MULTIPLY requires $O\left(d(d|B|)^{|B|}\right)$ multiplications in G.

In light of the exponential growth of the words through which MULTIPLY must iterate, small-base representations for the quotient group are desirable for satisfactory performance of the multiplication algorithm.

2.5. Applications

There are several immediate applications of the multiplication procedure.

(i) Given $g \in G$, compute the representative of g in L in normal form. The way that this is done depends on the data structure used to represent the transversals in G; if a Schreier vector is used, then the representative is found by multiplying through the sequence in $S \cup S^{-1}$ encoding g.

(ii) Given $e = \overline{e}n_1$ written in normal form, compute e^{-1} in normal form. To do this, first use the method in (i) to compute the representative of $\overline{e}^{-1}$ (regarded as an element of G) in L, say $\overline{e^{-1}}n_2$. Then $\overline{e}n_1 \cdot \overline{e^{-1}}n_2 = n_3$ for some $n_3 \in N$ which can be computed using MULTIPLY. Therefore, one has $e^{-1} = (\overline{e^{-1}}n_2) \cdot n_3^{-1}$. Assuming that Schreier vectors are used to encode transversal elements, finding the inverse of an element in E requires $O(d)$ multiplications.

(iii) Compute the order of $e = \overline{e}n$. First utilise the existing methods for permutation groups to compute the order of $\overline{e}$ regarded as an element of G. Then use MULTIPLY to raise $\overline{e}n$ to this power. This yields an element of N whose order can be computed using the existing methods for polycyclic groups; the order of $\overline{e}n$ is of course the product of the two numbers.

3. Transfer to the Category of Polycyclic Groups

Outlined in this brief section is a method that computes a consistent power-conjugate (or polycyclic) presentation for a polycyclic-by-finite group E that is in fact polycyclic. For the sake of clarity, it will be assumed in the following discussion that E is finite, and a power-conjugate presentation for E will be constructed; a similar procedure may be employed in the infinite case.

Continuing with the notation from previous sections, if E is polycyclic, then the quotient $G = E/N$ is polycyclic and one may use the already available methods for permutation groups to compute a power-conjugate presentation for G (see Seress, 2003, ch. 7). A consistent polycyclic or power-conjugate presentation for E may be found by "glueing together" the presentations for G and N as described below.

Let N and G have consistent power-conjugate presentations

$$\langle a_1, \ldots, a_r \mid a_j^{p_j} = w_{j,j} \quad \text{for } 1 \leqslant j \leqslant r, \quad a_j^{a_i} = w_{i,j} \quad \text{for } 1 \leqslant i < j \leqslant r \rangle,$$

and

$$\langle \overline{b_1}, \ldots, \overline{b_s} \mid \overline{b_j}^{q_j} = v_{j,j} \quad \text{for } 1 \leqslant j \leqslant s, \quad \overline{b_j}^{\overline{b_i}} = v_{i,j} \quad \text{for } 1 \leqslant i < j \leqslant s \rangle,$$

respectively, where

(i) p_j, q_j are the least primes such that $a_j^{p_j} \in \langle a_{j+1}, \ldots, a_r \rangle$ for $j < r$ and $\overline{b_j}^{q_j} \in \langle \overline{b_{j+1}}, \ldots, \overline{b_s} \rangle$ for $j < s$, and $a_r^{p_r}$, $\overline{b_s}^{q_s}$ are the identity elements in N, G respectively, and

(ii) $w_{i,j}$, $v_{i,j}$ are words in the generator sets $\{a_{i+1}, \ldots, a_r\}$, $\{\overline{b_{i+1}}, \ldots, \overline{b_s}\}$ respectively.

A power-conjugate presentation for E can be constructed on the set of generators $\{\overline{b_1}, \ldots, \overline{b_s}, a_1, \ldots, a_r\}$ as follows. First, regard the elements $\overline{b_j}$ as elements of E (the normal form of each of these elements may be obtained using the procedure outlined in Subsection 2.5); one has $\overline{b_j}^{q_j} = v_{j,j} n'$ for some $n' \in N$. The power-conjugate presentation for N may be used to express n' as a word in the required form over its polycyclic generators. Power relations with left-hand-side $a_j^{p_j}$ remain unchanged in the new presentation.

Conjugate relations are derived in a similar manner; the multiplication algorithm is partly utilised to conjugate the elements a_j according to the hierarchy induced by G and N. The presentation so obtained is a consistent power-conjugate presentation for E.

4. The Extended Schreier–Sims Algorithm

The class of polycyclic-by-finite groups is of course closed under the formation of subgroups, and so, it is theoretically possible to represent subgroups using the scheme described in Section 2.

Using the notation of 2, the problem of representing subgroups may be formulated more concretely as follows. Given a set of generators, written in normal form relative to G and N, of a subgroup $H \leqslant E$ it is required to compute the following information relating to H:

(i) The normal polycyclic subgroup $N_H = H \cap N$ of H.

(ii) A base B_H and strong generating set T relative to B_H, for the quotient $G_H = H/N_H \cong HN/N$.

(iii) Elements of E representing images of T in a transversal L_H of N_H in H.

(iv) Conjugates of each polycyclic generator of N_H by the images of T in L_H.

(v) Data corresponding to Equations (2.4) in the context of H.

Items (iv) and (v) are calculated in a manner similar to that of the parent group.

The difficulty lies in computing (i) and (iii). The approach taken is to extend the well-known Schreier–Sims method described in (Seress, 2003, ch. 4), to efficiently compute the segments of data described in (i), (ii), and (iii) simultaneously. Given a set of elements of E that generate a subgroup $H \leqslant E$ the extended Schreier–Sims algorithm aims to compute the intersection $H \cap N$, together with a base for the quotient HN/N and a set of elements of E (or more precisely HN) whose images in HN/N define a strong generating set relative to the computed base.

In the language of the definitions made in Subsection 2.1, the chief objective of the Extended Schreier–Sims method may be stated more simply: given a set of generators of a subgroup $H \leqslant E$ the method attempts to compute a base and a strong generating set for H.

4.1. The Sifting Procedure

Before turning to the problem of how a base and a strong generating set for a subgroup of a polycyclic-by-finite group are computed, it will be assumed for the moment that these are given, and a version of the *sifting* (see Seress, 2003, ch. 4) procedure for polycyclic-by-finite groups will be introduced.

Let S_H be a strong generating set for the subgroup H of E, with associated base $B_H = (\gamma_1, \ldots, \gamma_k)$. Assume that the basic orbits $\Theta_i = {\gamma_i}^{H^{[i]}}$ have been calculated and that Schreier vectors encoding right transversals R_i of $H^{[i+1]}$ in $H^{[i]}$ exist for each i, with elements of R_i represented as products over $S_H \cap H^{[i]}$.

The sifting algorithm operates as follows. Given $\overline{e}n \in E$ and $j \in \{1, \ldots, k\}$, the algorithm attempts to find the coset representative $\overline{y_1}n_1' \in R_j$ such that $\gamma_j^{\overline{e}n} = \gamma_j^{\overline{y_1}n_1'}$, and (if $j < k$) computes $\overline{e_2}n_2 = \overline{e}n \cdot (\overline{y_1}n_1')^{-1} \in H^{[j+1]}$ if a coset representative is found. If no such coset representative exists, then $\overline{e}n$ takes γ_j out of the orbit Θ_j, and the algorithm breaks and returns the *siftee* $\overline{e}n$ along with an integer indicating the point at which the break occurred (in this case j). Otherwise, the algorithm continues and attempts find $\overline{y_2}n_2' \in R_{j+1}$ such that $\gamma_{j+1}^{\overline{e_2}n_2} = \gamma_2^{\overline{y_2}n_2'}$ and then computes $\overline{e_3}n_3 = \overline{e_2}n_2 \cdot (\overline{y_2}n_2')^{-1}$ if possible. The algorithm attempts to perform $k - j + 1$ iterations of this type, immediately breaking if, at any stage, the base point is taken out of orbit, in which case the siftee and an integer indicating the stage is returned. If the algorithm is able to perform all iterations successfully, then it returns the siftee and the integer $k + 1$.

Remark 4.1. The element $\overline{e}n \in E$ belongs to H if and only if $k-j+1$ iterations are performed by the sifting procedure, and the siftee returned belongs to $H \cap N$. Thus, the sifting algorithm has reduced membership testing in H to membership testing in $H \cap N$, which can be accomplished using the available methods for polycyclic groups (see Holt et al., 2005, sec. 8.3).

The sifting procedure is presented in Algorithm 2. The function operates as described above, taking as input an element $\overline{e}n$ of a polycyclic-by-finite group E, a base B_H for a subgroup $H \leqslant E$, the sequence $\Theta = (\Theta_1, \ldots, \Theta_k)$ of basic orbits, a sequence $R = (R_1, \ldots, R_k)$ where R_i is a transversal of $H^{[i+1]}$ in $H^{[i]}$, together with an integer j.

Algorithm 2 Sifting

```
1: function SIFT(ēn, B_H, Θ, R, j)
2:     k ← LENGTH(B_H)
3:     ē_s n_s ← ēn
4:     for i ← j to k do                    ▷ ē_s n_s fixes base points γ_1, ..., γ_{i-1}
5:         ω ← γ_i^{ē_s n_s}
6:         if ω ∉ Θ_i then
7:             return ē_s n_s, i
8:         Find the coset representative ȳ_i n'_i ∈ R_i such that γ_i^{ȳ_i n'_i} = ω
9:         ē_s n_s ← ē_s n_s · (ȳ_i n'_i)^{-1}
10:    return ē_s n_s, k + 1
```

Assuming that the transversals are encoded as Schreier vectors, SIFT requires $O(dk)$ multiplications in E, where k is the length of the input base.

4.2. Extending the Schreier–Sims Algorithm

The operation of the extended Schreier–Sims algorithm is based on the following lemma, whose proof is a straightforward induction on the base length.

Lemma 4.2. *Let E be a polycyclic-by-finite group, $H \leqslant E$ and let $N \lhd E$ be polycyclic of finite index in E. Assume that there is a known permutation representation for the quotient $G = E/N$, and that the elements of E are expressed in normal form relative to G and N. Denote the set on which G acts by Ω and let $\{\gamma_1, \ldots, \gamma_k\} \subseteq \Omega$. For each i in $\{1, \ldots, k+1\}$, let $S_{H,i} \subseteq H_{(\gamma_1,\ldots,\gamma_{i-1})}$ such that $\langle S_{H,i}\rangle \geqslant \langle S_{H,i+1}\rangle$ holds for $i \leqslant k$. If $H = \langle S_{H,1}\rangle$, $S_{H,k+1} \subseteq H \cap N$, and*

$$\langle S_{H,i}\rangle_{\gamma_i} = \langle S_{H,i+1}\rangle \tag{4.1}$$

holds for each i, then $B_H = (\gamma_1, \ldots, \gamma_k)$ is a base for H and $S_H = \bigcup_{i=1}^{k+1} S_{H,i}$ is a strong generating set for H relative to B_H. □

Given a set X of generators for a subgroup $H \leqslant E$ the extended Schreier–Sims algorithm constructs a base and strong generating set in the following way. A data structure containing a list $B_H = (\gamma_1, \ldots, \gamma_k)$ of already known elements of a non-redundant base is maintained, along with an approximation $S_{H,i}$ for a generator set of the stabiliser $H_{(\gamma_1,\ldots,\gamma_{i-1})}$ for each $i \in \{1, \ldots, k+1\}$. Throughout execution, the $S_{H,i}$ satisfy the property that, for all i, $\langle S_{H,i}\rangle \geqslant \langle S_{H,i+1}\rangle$, and $S_{H,k+1} \subseteq H \cap N$. The data structure is said to be *up-to-date below level j* if Equation (4.1) holds for each i in the range $j < i \leqslant k$.

In the case where the data structure is up-to-date below level j, a transversal R_j of $\langle S_{H,j}\rangle_{\gamma_j}$ in $\langle S_{H,j}\rangle$ is computed. Then a check is made to determine whether Equation (4.1) is satisfied for $i = j$. This can be done by sifting the Schreier generators (see Seress, 2003, ch. 4) obtained from R_j and $S_{H,j}$ in the group $\langle S_{H,j+1}\rangle$. By Remark 4.1, membership testing is possible in the group, $\langle S_{H,j+1}\rangle$, since Lemma 4.2 implies that $\bigcup_{i=j+1}^{k+1} S_{H,i}$ is a strong generating set

for $\langle S_{H,j+1} \rangle$ relative to the base $(\gamma_{j+1}, \ldots, \gamma_k)$. If all Schreier generators are in $\langle S_{H,j+1} \rangle$ then the data structure is up-to-date below level $j-1$. Otherwise there exists a non-trivial siftee $\overline{e_s} n_s$ which takes some base point, say γ_t, out of orbit. If $t = k+1$ and $\overline{e_s} \neq 1_G$, then a new base point γ_{k+1} is appended to B_H from $\mathrm{supp}(\overline{e_s} n_s)$ and the set $S_{H,k+2}$ is initialised with the contents of $S_{H,k+1}$. The siftee $\overline{e_s} n_s$ is added to the sets $S_{H,j+1}, \ldots, S_{H,t}$, and the data structure is now up-to-date below level $\min(t, |B_H|)$.

The algorithm initialises B_H to contain a sequence of points $\gamma_1, \ldots, \gamma_k$ in Ω such that each point is moved by at least one generator in X with non-trivial first component, and sets $S_{H,i}$ to $X \cap \langle X \rangle_{(\gamma_1,\ldots,\gamma_{i-1})}$ for $i = 1, \ldots, k+1$. At that moment, the data structure is up-to-date below level k; the algorithm terminates when the data structure becomes up-to-date above level 0.

A simplified version of the extended Schreier–Sims method is presented in Algorithm 3. The function takes as input a set X of elements of a polycyclic-by-finite group E and operates as described above to compute a base and strong generating set for $H = \langle X \rangle$.

The notation employed thus far is used in the pseudocode, and SuperSchreierSims also assumes the existence of a procedure Append which, when supplied with an array (or sequence) and an element, appends the element to the end of the array (or sequence).

The normal subgroup $H \cap N$ of H can be computed easily from the strong generating set S_H returned by SuperSchreierSims as it is generated by the elements in S_H with trivial first component.

Theorem 4.3. *The extended Schreier–Sims method terminates, returning a correct base and strong generating set.*

Proof. This is immediate from Lemma 4.2 and the discussion following it. □

The dominant computations in SuperSchreierSims are the multiplications in E using the normal form for elements, and these multiplications occur in Lines 18, 20, 21, and 25. Lines 18, 20, and 21 each require $O(d)$ multiplications; and, with an application of (2.3), one sees that Line 25 requires $O(d \log |G|)$ multiplications.

In practice, when a new loop is commenced in Line 16 only Schreier generators which have not been previously checked are constructed (Lines 18, 20, and 21) and sifted (Line 25). Therefore, the lines in question are executed exactly once for each Schreier generator.

After initialisation in Lines 3–11, each R_i has size at most d, and each $S_{H,i}$ has size at most $|X|$. By (2.3), the number of Schreier generators before any sift operations are performed is $O(d|X| \log |G|)$.

Observe that although the sets R_i and $S_{H,i}$ change during the operation of the algorithm, they are always only augmented, and therefore, elements of R_i and $S_{H,i}$ are be combined to form a Schreier generator only once. Fix i and set $K = \langle S_{H,i} \rangle$; every time the set $S_{H,i}$ is augmented, the group $KN/N \leqslant G$ increases. Therefore, the set $S_{H,i}$ can change at most $\log_2 |G|$ times during the algorithm. Combining this with the bound $|R_i| \leqslant d$ and (2.3), it follows that the

Algorithm 3 The extended Schreier–Sims algorithm

function SUPERSCHREIERSIMS(X)
  $B_H \leftarrow ()$, $k \leftarrow 0$
  for $\overline{e}n \in X$ **do**
    if $\overline{e} \neq 1_G$ **and** $\gamma_i^{\overline{e}n} = \gamma_i \ \forall\ i \in \{1, \ldots, k\}$ **then** ▷ $\overline{e}n$ fixes all points in B_H
      Find $\gamma_{k+1} \in \Omega$ with $\gamma_{k+1}^{\overline{e}n} \neq \gamma_{k+1}$
      APPEND(B_H, γ_{k+1})
      $k \leftarrow k + 1$
  for $i \leftarrow 1$ **to** k **do**
    $S_{H,i} \leftarrow \{\overline{e}n \in X \mid \gamma_j^{\overline{e}n} = \gamma_j \ \forall\ j \in \{1, \ldots, i-1\}\}$
    $H^{[i]} \leftarrow \langle S_{H,i} \rangle$, $\Theta_i \leftarrow \gamma_i^{H^{[i]}}$,
    Compute a stabiliser transversal R_i in $H^{[i]}$ corresponding to Θ_i
  $\Theta \leftarrow (\Theta_1, \ldots, \Theta_k)$, $R \leftarrow (R_1, \ldots, R_k)$
  $S_{H,k+1} \leftarrow \{\overline{e}n \in X \mid \overline{e} = 1_G\}$
  $N_H \leftarrow \langle S_{H,k+1} \rangle$ ▷ This subgroup is generated in N
  $i \leftarrow k$
  while $i \geqslant 1$ **do**
    for $\theta \in \Theta_i$ **do**
      Find the coset representative $\overline{e}n \in R_i$ such that $\gamma_i^{\overline{e}n} = \theta$
      for $\overline{x}n_x \in S_{H,i}$ **do**
        Find the coset representative $\overline{e'}n' \in R_i$ such that $\gamma_i^{\overline{e'}n'} = \theta^{\overline{x}n_x}$
        $\overline{e}n \leftarrow \overline{e}n \cdot \overline{x}n_x \cdot (\overline{e'}n')^{-1}$
        if $\overline{e}n = 1$ **then**
          continue $\overline{x}n_x$
        uptodate $\leftarrow$ **true**
        $\overline{e}n$, $j \leftarrow$ SIFT($\overline{e}n$, B_H, Θ, R, $i+1$)
        if $\overline{e} \neq 1_G$ **then**
          uptodate $\leftarrow$ **false**
          if $j > k$ **then**
            Find $\gamma_{k+1} \in \Omega$ with $\gamma_{k+1}^{\overline{e}n} \neq \gamma_{k+1}$
            APPEND(B_H, γ_{k+1}) ▷ Extend base
            $k \leftarrow k + 1$
            $S_{H,k+1} \leftarrow S_{H,k}$ ▷ Maintain inclusion
        else if $n \notin N_H$ **then**
          uptodate $\leftarrow$ **false**
          $S_{H,k} \leftarrow S_{H,k} \cup \{\overline{e}n\}$, $N_H \leftarrow \langle N_H, n \rangle$
          $j \leftarrow k$
        if not *uptodate* **then**
          for $t \leftarrow i + 1$ **to** j **do**
            $S_{H,t} \leftarrow S_{H,t} \cup \{\overline{e}n\}$, $H^{[t]} \leftarrow \langle S_{H,t} \rangle$, $\Theta_t \leftarrow \gamma_t^{H^{[t]}}$,
            Compute a stabiliser transversal R_t in $H^{[t]}$
            corresponding to Θ_t

42: $\Theta \leftarrow (\Theta_1, \ldots, \Theta_k)$, $R \leftarrow (R_1, \ldots, R_k)$
43: $i \leftarrow j + 1$
44: **break** θ
45: $i \leftarrow i - 1$
46: **return** B_H, $\bigcup_{i=1}^{k+1} S_{H,i}$

total number of Schreier generators throughout the operation of the algorithm is $O(d|X|\log|G| + d\log^2|G|)$.

It follows from the discussion above that the modified Schreier–Sims method requires $O((d\log|G|)^2(|X| + \log|G|))$ multiplications in E.

4.3. Modifications

In this subsection, two useful modifications of the extended Schreier–Sims procedure are briefly discussed.

The first modification concerns the manner in which $H \cap N$ is computed. Consider Lines 17–21 of SuperSchreierSims. If $\overline{x} = 1_G$, then $\theta^{\overline{x}n_x} = \theta$ whence $\overline{e'}n' = \overline{e}n$ and the multiplication of elements in Line 21 is reduced to the conjugation $\overline{e}n \cdot n_x \cdot (\overline{e}n)^{-1}$ of a generator in N by an element of H. The algorithm proceeds to sift this conjugate down to level $k + 1$, adding it to the generator set of $H \cap N$.

One may modify SuperSchreierSims so that strong generators with trivial first component are not added at every level. Instead, generators of this type are kept in a separate set and a normal closure computation is performed after the structure becomes up-to-date below level 0. This normal closure is of course the subgroup $H \cap N$.

Eliminating strong generators with trivial first component from the data structure has the obvious advantage of speeding up both the orbit computations and the membership checks at each level. From a theoretical perspective, the time complexity of this version of the extended Schreier–Sims method is no different from the first version. However, in practice, methods used to compute normal closures often terminate rapidly, and so this version may offer a slight advantage in efficiency.

The second modification has to do with the computation of the base of the subgroup. Both versions of the extended Schreier–Sims method attempt to compute a new base. Situations may arise where one wishes to use the base of the parent group in the representation of the subgroup. Only a minor adjustment is needed: initialise B_H to B at the beginning of the algorithm.

As indicated in the discussion immediately following Theorem 2.5, smaller bases are preferable in relation to the performance of Multiply. Taking this into consideration, it may be prudent in most instances to opt for a possible reduction in base size, unless the application specifically requires that the original base be retained.

4.4. Applications

Following from the discussions above, one is now able to compute the data which enables subgroups of a given polycyclic-by-finite group to be represented as in Section 2. Using this data, one may write simple iterative procedures which construct derived and lower central series. For a review of how these standard functions are implemented, see Holt et al. (2005, Sec. 3.3).

In the subsections below, two straightforward yet useful applications of the methods of this section are given. The methods presented heavily exploit the polycyclic-by-finite structure of the group in question. The natural map $E \to G$ is denoted by ρ.

4.4.1. The Solvable Radical

The *solvable radical* of E, denoted by $\mathbf{O}_\infty(E)$, is defined as the largest solvable normal subgroup of E. It follows from the definition of the solvable radical that $N \leqslant \mathbf{O}_\infty(E)$.

Thus, finding $\mathbf{O}_\infty(E)$ amounts to computing $\mathbf{O}_\infty(G)$, and then finding the preimage under ρ of $\mathbf{O}_\infty(G)$. One may utilise the existing methods for permutation groups (see Seress, 2003, ch. 6) to execute the former, while the latter task is accomplished using the methods of this section.

4.4.2. Sylow Subgroups

In this subsection, it is assumed that E is finite. Let p be a prime dividing $|E|$ and let H_p be a Sylow p-subgroup of G. Observe that a Sylow p-subgroup of the preimage $\overline{H_p} = \rho^{-1}(H_p)$ is a Sylow p-subgroup of E.

Thus, one may compute a Sylow p-subgroup of E as follows. First, find a Sylow p-subgroup H_p of G. This is done using the available methods for permutation groups (see Holt et al., 2005, ch. 4). Then, using a method similar to that of Subsection 4.4.1, generate $\overline{H_p} = \rho^{-1}(H_p)$ as a subgroup of E. This group is finite and solvable, and so, one may produce a consistent power-conjugate presentation defining it using the method of Subsection 3. The existing methods for computing Sylow subgroups in finite solvable groups represented by power-conjugate presentations are now applicable (see Seress, 2003, ch. 7). Finally, the isomorphism from the subgroup defined by the power-conjugate presentation can be used to map a generating set for a Sylow p-subgroup back into E.

By similar methods, it is also possible to design an algorithm which, when given two Sylow p-subgroups of E, returns an element which conjugates one Sylow subgroup to the other. The first step is to find, using the available functions for permutation groups (see Holt et al., 2005, ch. 4), an element $g \in G$ which conjugates $\rho(P_1)$ to $\rho(P_2)$ in G. Using the data set, one then computes an element $\bar{e}n \in E$ corresponding to g. It follows then that $P_1^{\bar{e}n}$ is a Sylow p-subgroup of the group $\rho^{-1}(\rho(P_2)) = P_2N$, which contains P_2 as a Sylow p-subgroup.

The group P_2N is finite and solvable, and so may be represented by a consistent power-conjugate presentation. Using the available methods for finite

solvable groups (see Seress, 2003, ch. 7), one may find a conjugating element $y \in P_2 N$ such that $P_1^{\bar{e}n \cdot y} = P_2$, as required.

5. Centralisers and Conjugacy

The material presented in this section utilises the machinery developed thus far to perform some advanced structural calculations within polycyclic-by-finite groups. Specifically, the problem of computing centralisers and testing element conjugacy is addressed.

5.1. The Centre

The approach to computing the centre of a given polycyclic-by-finite group E is as follows:

1. Find a normal abelian subgroup A of E that contains $\mathbf{Z}(E)$.
2. For each member x of a generating set X for E, compute the matrix M_x that specifies the conjugation action of that generator on elements of A (written as vectors).
3. Construct a matrix whose null space is equal to the subspace corresponding to $\mathbf{Z}(E)$ in A.
4. Map a set of vectors generating $\mathbf{Z}(E)$ in A back into E.

The following lemma furnishes one with a feasible choice for the normal abelian subgroup; its proof is straightforward.

Lemma 5.1. *Let G be a group with solvable radical S. Then $\mathbf{Z}(G) \trianglelefteqslant \mathbf{Z}(S) \trianglelefteqslant G$.* □

Let S be the solvable radical of E. Then by Lemma 5.1, $\mathbf{Z}(E)$ is contained in the normal abelian subgroup $\mathbf{Z}(S)$.

The solvable radical S can be computed as a subgroup of E using the method described in Subsection 4.4.1, after which it is rewritten as a polycyclic group by the procedure given in Subsection 3. The available functions for polycyclic groups may then be utilised to compute the centre $\mathbf{Z}(S)$ of S as an abelian group; see (Holt et al., 2005, ch. 8) for the finite case and (Eick, 2001) for the infinite case.

Write $\mathbf{Z}(S)$ as an abelian group A with invariant factor decomposition $\mathbb{Z}_{m_1} \times \cdots \times \mathbb{Z}_{m_t} \times \mathbb{Z}^k$ where $m_i \mid m_{i+1}$ for each i. Writing the elements of A as row vectors, let $\mathbf{a}_1, \ldots, \mathbf{a}_{t+k}$ be the standard basis for A relative to its invariant factor decomposition. Let $x \in X$. Regarding matrices as acting on the right, the i-th row of the matrix M_x over $\mathbb{Z}$, defining conjugation in A by x is given by $\mathbf{a}_i x$ expressed as a row vector, where E acts on A by conjugation.

The element $\mathbf{a}_i x$ is found by first mapping $\mathbf{a}_i$ into the group E, and then using the multiplication algorithm to perform the conjugation. The result is mapped back into A and written as a row vector.

An element $\mathbf{z} \in A$ lies in $\mathbf{Z}(E)$ if and only if, for each $x \in X$, $\mathbf{z}M_x = \mathbf{z}$. A matrix whose null space corresponds to the set of all such $\mathbf{z}$ is constructed as follows. Let D be the $t \times (t+k)$ matrix over $\mathbb{Z}$,

$$\begin{pmatrix} m_1 & 0 & \cdots & 0 & 0 & \cdots & 0 \\ 0 & m_2 & \cdots & 0 & 0 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots & 0 & \cdots & 0 \\ 0 & 0 & \cdots & m_t & 0 & \cdots & 0 \end{pmatrix}.$$

Let $X = \{x_1, \ldots, x_q\}$ and let $T_x = M_x - I$ for each $x \in X$, where I is the identity $(t+k) \times (t+k)$ matrix. Form the following matrix Q over $\mathbb{Z}$

$$\left(\begin{array}{c|c|c|c} T_{x_1} & T_{x_2} & \cdots & T_{x_q} \\ \hline D & & & \\ & D & & \\ & & \ddots & \\ & & & D \end{array}\right),$$

where blank spaces indicate zero entries.

Proposition 5.2. *If a vector $\mathbf{z} \in A$ lies in $\mathbf{Z}(E)$, then there exists a vector $(\mathbf{z}' \mid \mathbf{v}) \in \mathbb{Z}^{t(q+1)+k}$ in the null space of Q, where $\mathbf{z}'$ denotes the vector $\mathbf{z}$ viewed as a vector in $\mathbb{Z}^{t+k}$. Conversely, the first $t+k$ entries of any vector in the null space of Q define a vector belonging to $\mathbf{Z}(E)$.*

Proof. Suppose that $\mathbf{z} \in A$ lies in $\mathbf{Z}(E)$. Then for each $x_i \in X$, $\mathbf{z}T_{x_i} = \mathbf{0}$, equality being in the abelian group $A \cong \mathbb{Z}_{m_1} \times \cdots \times \mathbb{Z}_{m_t} \times \mathbb{Z}^k$. Denote by $\mathbf{z}'$ the vector $\mathbf{z}$ viewed as a vector in $\mathbb{Z}^{t+k}$, $\mathbf{z}T_{x_i} = \mathbf{0}$ is equivalent to $\mathbf{z}'T_{x_i} = -\mathbf{v}_i D$ for some $\mathbf{v}_i \in \mathbb{Z}^t$. Hence, the vector $(\mathbf{z}' \mid \mathbf{v}_1 \mid \cdots \mid \mathbf{v}_q)$ belongs to $\ker(Q)$.

Conversely, let $\mathbf{v} \in \ker(Q)$ and partition $\mathbf{v}$ into segments, the first of length $t+k$, and the rest each of length t: $(\mathbf{y} \mid \mathbf{v}_1 \mid \cdots \mid \mathbf{v}_q)$. Then $\mathbf{y}T_{x_i} = -\mathbf{v}_i D$ for each i, which, as above, is equivalent to $\mathbf{z}T_{x_i} = \mathbf{0}$ in A, where $\mathbf{z}$ denotes the vector $\mathbf{y}$ viewed as a vector in $\mathbb{Z}_{m_1} \times \cdots \times \mathbb{Z}_{m_t} \times \mathbb{Z}^k$; whence $\mathbf{z}$ lies in $\mathbf{Z}(E)$. □

Thus, the image of $\mathbf{Z}(E)$ in A, can be recovered from a generating set of $\ker(Q)$. Mapping this image back into E completes the computation.

Assuming that the normal abelian subgroup containing the centre is given (or has been computed), the running time of the rest of the computations outlined is dominated by the calculation of the null space. The fastest known algorithms to perform such computations use p-adic expansions, and were introduced by Dixon (1982). They require $O(c^4(\log_2 r)^2)$ time for an $r \times c$ matrix. For recent developments in this area, see (Haramoto and Matsumoto, 2009). It is worth noting here that, if $k = 0$ and $m_i = m$ for each i and some $m \in \mathbb{N}$, then the computation can be performed entirely in $\mathbb{Z}_m$, offering a significant improvement in efficiency. For a practical account of computations of this type, the reader is referred to (Holt et al., 2005, ch. 7).

5.2. Centralisers and Conjugacy

In this section the problems of centraliser computation and element conjugacy testing are briefly discussed. In what follows it is assumed that E is finite.

The normal subgroup $N \trianglelefteq E$ is solvable and finite, and hence possesses a normal elementary abelian series: $N = N_1 \trianglerighteq \cdots \trianglerighteq N_t \trianglerighteq N_{t+1} = 1$. Let $e \in E$, and denote by $\mathbf{C}_{E/N_i}(e)$ the centraliser of eN_i in E/N_i. The algorithm begins by computing a set of generators (in normal form) for the preimage of $\mathbf{C}_{E/N}(e)$ in E. This is accomplished by utilising the procedure for preimage calculation described in Subsection 2.5 along with the already available methods for permutation groups described in (Seress, 2003) and (Holt et al., 2005, ch. 4).

The algorithm continues by lifting through the normal elementary abelian series of N, successively computing preimages in E of the centralisers $\mathbf{C}_{E/N_i}(e)$, as described in (Mecky and Neubüser, 1989) and (Holt et al., 2005, sec. 8.8). More concretely, assume that a set of generators for the preimage, C, of $\mathbf{C}_{E/N_i}(e)$ in E have been computed. The group C acts on the elementary abelian layer N_i/N_{i+1}, and the lifting step is essentially reduced to a vector stabiliser computation. Testing for element conjugacy is done in much the same way, except that the algorithm computes both the orbit and the stabiliser at each stage.

The method described here is similar in theory to the method which involves a single orbit-stabiliser computation, but the induction method computes relatively small orbits of vectors instead of one relatively large orbit of elements in a polycyclic-by-finite group. Thus, the induction method is usually more efficient than the single orbit-stabiliser application.

6. Implementation and Examples

The suite of algorithms described in the paper have been implemented using the Magma Computational Algebra System. Table 1 gives running times for a sample of test groups.

The format of Table 1 is as follows. The first column gives the Atlas notation of the group. The largest prime q dividing the order of the group (when the group is finite) is given in the second column. The remaining columns give the running times (in milliseconds, averaged over 100 runs, randomised where applicable) to perform the indicated computation.

The first four examples in Table 1 are extensions of modules by their groups; these groups are constructed with the help of the natively implemented cohomology functions in Magma. The last two examples in Table 1 are quotients of finitely presented groups, each of which map onto A_5.

The first entry of Table 1 is a non-split extension of the irreducible module $(\mathbb{Z}/11\mathbb{Z})^3$ by $\mathrm{PSL}(2, 11)$. It has relatively small order (878460) and has minimal degree of permutation representation 132.

The second and third entries of Table 1 are non-split extensions of permutation modules by their groups: $\mathbb{Z}^7$ by $\mathrm{PSL}(2, 7) \cong \mathrm{PSL}(3, 2)$, and $\mathbb{Z}^{10}$ by $A_6 \cong \mathrm{PSL}(2, 9)$.

The fourth entry of Table 1 is a non-split extension of the irreducible module $(\mathbb{Z}/2\mathbb{Z})^{10}$ by M_{12}. It has order $2^{16} \cdot 3^3 \cdot 5 \cdot 11$ and minimal degree of permutation representation 264.

The fifth entry of Table 1 is a non-split extension of an extra special group of order 2^9 by $\mathrm{O}_8^+(2)$. It has order $2^{21} \cdot 3^5 \cdot 5^2 \cdot 7$, and the authors have obtained a permutation representation of this group having degree 34560.

The last two entries of Table 1 are finitely presented groups, each of which map onto A_5. The penultimate entry is a quotient of the Heineken group,

$$H = \langle a, b, c \mid [a, [a, b]] = c, [b, [b, c]] = a, [c, [c, a]] = b \rangle.$$

The quotient has order $2^{26} \cdot 3 \cdot 5$, and the authors conjecture that the minimal degree of a permutation representation of this group is 138240.

The final entry is a quotient of the group having presentation

$$\langle a, b, c \mid a^2, b^3, (ab)^5, abcb^{-1}ac^{-1}b^{-1}c^{-1}bc^{-1} \rangle.$$

The quotient has order $2^2 \cdot 3 \cdot 5 \cdot 13^{55}$. The minimal degree of a permutation representation of this group is unknown, but the authors believe that it is too large to be computationally useful.

Benchmark running times are not available, as there is no existing standard method in Magma to perform most of the computations listed, and, in the case of the last two groups of Table 1, it is not possible to natively construct the extension.

The tests were performed on a system with the following specifications: 16GB memory, 2.90GHz quad-core processor, Linux kernel version 3.14.0, Magma V2.21-7.

Table 1: Sample running times

Group Information		Average running time (milliseconds)				
Atlas Notation	q	$x \cdot y$	x^{-1}	Syl_2	Syl_q	$\mathbf{Z}$
$11^3{\cdot}\mathrm{L}_2(11)$	11	1.1	2.3	278.8	567.2	12.1
$\mathbb{Z}^7{\cdot}\mathrm{L}_3(2)$	-	0.8	1.5	-	-	32.6
$\mathbb{Z}^{10}{\cdot}\mathrm{A}_6$	-	1.8	2.4	-	-	53.9
$2^{10}{\cdot}\mathrm{M}_{12}$	11	28.4	33.4	48868.7	33120.1	148.5
$2^{1+8}{\cdot}\mathrm{O}_8^+(2)$	7	164.8	115.5	66722.7	56815.8	24.8
$2^{24} \cdot \mathrm{A}_5$	5	0.7	1.1	367.4	284.7	51.0
$13^{55} \cdot \mathrm{A}_5$	13	0.9	1.6	626.3	121.2	1519.9

7. Conclusion

The running times of Table 1 are encouraging. Since the computations described involve large numbers of executions of individual group operations, such as mulitplication and conjugation, it is possible that the Magma interpreter is causing a significant overhead, and that running times could be significantly improved by a native implementation.

One expects that, after optimised implementation, the data structure and algorithms described in this paper would become a suitable option for computation with polycyclic-by-finite groups for which there is no available permutation or matrix representation, but whose finite quotient is relatively small and easily represented as a permutation group.

8. Acknowledgements

The authors would like to thank Colva Roney-Dougal for her helpful suggestions. We would also like to recognise David Howden whose work on automorphism groups has helped in the construction of some of the motivating examples.

9. Bibliography

Baumslag, G., Cannonito, F. B., Robinson, D. J. S., Segal, D., 1991. The Algorithmic Theory of Polycyclic-by-Finite Groups. Journal of Algebra 142, 118–149.

Dixon, J. D., 1982. Exact solution of linear equations using p-adic expansion. Numerische Mathematik 40, 137–141.

Eick, B., 2001. Algorithms for Polycyclic Groups. Habilitationsschrift, Universität Kassel.

Glasby, S. P., Slattery, M. C., 1990. Computing intersections and normalizers in soluble groups. J. Symbolic Comput. 9, 637–51.

Haramoto, H., Matsumoto, M., 2009. A p-adic algorithm for computing the inverse of integer matrices. Journal of Computational and Applied Mathematics 225, 320–322.

Holt, D. F., Eick, B., O'Brien, E. A., 2005. Handbook of Computational Group Theory. Discrete Mathematics and its Applications. Chapman & Hall/CRC.

Hulpke, A., 2013. Computing conjugacy classes of elements in matrix groups. Journal of Algebra 387, 268–286.

Laue, R., Neubüser, J., Schoenwaelder, U., 1984. Algorithms for finite soluble groups and the SOGOS system. In: Atkinson, M. D. (Ed.), Computational Group Theory. Academic Press, pp. 105–35.

Mecky, M., Neubüser, J., 1989. Some remarks on the computation of conjugacy classes of soluble groups. Bull. Australian Math. Soc. 40, 281–92.

Seress, Á., 2003. Permutation Group Algorithms. Vol. 152 of Cambridge Tracts in Mathematics. Cambridge University Press.

Sims, C. C., 1970. Computational methods in the study of permutation groups. In: Leech, J. (Ed.), Computational Problems in Abstract Algebra. Pergamon Press, Oxford, pp. 169–183.

Sims, C. C., 1971. Computation with permutation groups. In: Proc. Second Symposium on Symbolic and Algebraic Manipulation. ACM Press, New York, pp. 23–28.

Sims, C. C., 1994. Computation with finitely presented groups. Vol. 48 of Encyclopedia of Mathematics and its Applications. Cambridge University Press.