Chapter 18. Soluble Permutation Groups

This chapter deals with both soluble permutation groups and the special case of permutation p-groups. Both the general and special case rely on a data structure which represents both the subnormal series associated with a pc presentation and the base and strong generating sets of the subgroups in the series. The main concern is constructing the data structure from the permutation group representation - and the isomorphism between the permutation group and the group described by the pc presentation.

Combining Series Generators and Strong Generators

Let G be a finite soluble permutation group and let $B = [\beta_1, \beta_2, ..., \beta_k]$ be a base for G. A sequence $\vec{g} = [g_1, g_2, ..., g_m]$ of elements of G is called a *B-strong series-generating sequence* (B-ssgs) if

1. $\vec{g}$ is a series-generating sequence (gs) for a subnormal series

$$\{identity\} = G(m+1) \lhd G(m) \lhd \cdots \lhd G(2) \lhd G(1) = G,$$

of G, where $G(i) = \langle g_i, g_{i+1}, ..., g_m \rangle$; and

2. each set $\{g_i, g_{i+1}, ..., g_m\}$ is a strong generating set of $G(i)$ relative to B.

For example, the symmetries of the square has a base $B=[1,3]$ and a B-ssgs given by $g_1=(3,4)$, $g_2=(1,2)(3,4)$, and $g_3=(1,3)(2,4)$. The symmetric group of degree 4 has a base $B=[1,2,3]$ and a B-ssgs given by $g_1=(3,4)$, $g_2=(2,3,4)$, $g_3=(1,3)(2,4)$, and $g_4=(1,2)(3,4)$.

Given a base B, we can always find a B-ssgs. Since the group G is soluble, we can find a prime-step subnormal series

$$\{identity\} = G(m+1) \lhd G(m) \lhd \cdots \lhd G(2) \lhd G(1) = G,$$

and choose g_i to be the first permutation in $G(i) - G(i+1)$, under the lexicographical ordering on G induced from the ordering of Ω where the base points $\beta_1, \beta_2, ..., \beta_k$ are the first k points. These elements form a B-ssgs $[g_i : i = 1,2,...,m]$.

Generally, we work with a prime-step subnormal series where the index $|G(i):G(i+1)|$ is a prime p_i. A general B-ssgs is easily refined to a prime-step one by inserting appropriate powers of a generator g_i whenever the index $|G(i):G(i+1)|$ is composite.

Suppose we are working with a subgroup H of a soluble permutation group G. Suppose B is a base for G, and we know a B-ssgs $\vec{g}$ of H. A common method of constructing subgroups K is the cyclic extension method where $K = \langle H, y \rangle$ and the element $y \notin H$ normalizes H and satisfies $y^p \in H$, for some prime p. Of course, we wish to construct a B-ssgs of K. The subnormal series of K is

$$\{identity\} = H(m+1) \lhd H(m) \lhd \cdots \lhd H(2) \lhd H(1) = H \lhd K,$$

B is a base for K, and $\vec{g}$ contains the necessary strong generators of $H(i)$, so the outstanding

problem is to choose $g_0 \in K$ such that the set $\{g_0, g_1, \ldots, g_m\}$ is a strong generating set of K relative to B. To do this, we choose $g_0 \in yH$ as low in the stabiliser chain as possible. The stripping process of testing $y \in H$ will return a residue g_0 with this property. This is summarised in Algorithm 1.

Algorithm 1 : Prime-step Normalizing Generator

Input: a permutation group H with base $B=[\beta_1, \beta_2, \ldots, \beta_k]$,
strong generating set T and Schreier vectors $v^{(i)}$;
a prime p;
a permutation $y \notin H$ normalizing H such that $y^p \in H$;

Output: a base for $K = \langle H, y \rangle$ which may extend B;
an element g_0 such that $T \cup \{g_0\}$ is a strong generating set of K;

begin
$g_0 := y$; *extend_base* := true; $i := 0$;
while $i < k$ **and** *extend_base* **do**
$i := i + 1$;
if $\beta_i^{g_0} \in \Delta^{(i)}$ **then**
while $\beta_i^{g_0} \neq \beta_i$ **do**
$g_0 := g_0 \times v^{(i)}[\beta_i^{g_0}]^{-1}$;
end while;
else
extend_base := false;
end if;
end while;

if *extend_base* **then**
choose a point β_{k+1} moved by g_0;
append β_{k+1} to B;
end if;
end.

As an example of Algorithm 1, consider the symmetric group of degree 4, where $H=\langle g_2, g_3, g_4 \rangle$ is the alternating group and $y=(1,2)$; Then $K=\langle H, y \rangle$ is S_4 and the algorithm returns $g_0=(3,4)$.

The algorithm can be generalized to handle the case where y normalizes H and we do not know that the index $|K:H|$ is prime. The generalized algorithm will find a sequence of elements extending the strong generating set of H to a strong generating set of K. The details are in Algorithm 2.

Algorithm 2 : Normalizing Generator

Input: a permutation group H with base $B=[\beta_1,\beta_2,\ldots,\beta_k]$,
strong generating set T, Schreier vectors $v^{(i)}$, and basic orbits $\Delta^{(i)}$;
a permutation y normalizing H;

Output: a base for $K = \langle H,y\rangle$ which may extend B;
a sequence of elements $[g_1,g_2,\ldots,g_r]$ such that, for each j,
(a) $T \cup \{g_j,g_{j+1},\ldots,g_r\}$ is a strong generating set of $\langle H,g_j,g_{j+1},\ldots,g_r\rangle$,
(b) $g_j \notin \langle H,g_{j+1},g_{j+2},\ldots,g_r\rangle$,
(c) g_j normalizes $\langle H,g_{j+1},g_{j+2},\ldots,g_r\rangle$, and
(d) $g_j^{p_j} \in \langle H,g_{j+1},g_{j+2},\ldots,g_r\rangle$, for some prime p_j;

begin
$z := y$; $r := 0$; $i := 0$;
while $i < k$ **do**
$i := i + 1$;
if $\beta_i^z \in \Delta^{(i)}$ **then**
while $\beta_i^z \neq \beta_i$ **do** $z := z \times v^{(i)}[\beta_i^z]^{-1}$; **end while**;
else
let $p_1^{n_1} p_2^{n_2} \cdots p_t^{n_t}$ be the prime factorisation of $|\beta_i^{\langle H^{(i)},z\rangle}| / |\Delta^{(i)}|$;
for $j := 1$ **to** t **do**
for $l := 1$ **to** n_j **do**
$r := r + 1$; $g_r := z$; $z := z^{p_j}$;
end for;
end for;
end if;
end while;
while $z \neq$ *identity* **do**
$k := k + 1$;
choose a point β_k moved by z;
append β_k to B;
let $p_1^{n_1} p_2^{n_2} \cdots p_t^{n_t}$ be the prime factorisation of the orbit length $|\beta_k^{\langle z\rangle}|$;
for $j := 1$ **to** t **do**
for $l := 1$ **to** n_j **do**
$r := r + 1$; $g_r := z$; $z := z^{p_j}$;
end for;
end for;
end while;
end.

As an example of Algorithm 2, let H to be trivial subgroup, let $y=(1,2)(3,4,5,6)$ of order 4 and degree 6, and form $K = \langle y\rangle$. The first **while**-loop, which performs the stripping, does nothing in this example. The second **while**-loop could choose $\beta_1=1$, for which $|\beta_1^{\langle y\rangle}| = 2$, and $g_1=y=(1,2)(3,4,5,6)$. It would then perform a second iteration with $z=y^2=(3,5)(4,6)$, choose $\beta_2=2$ and $g_2=y^2=(3,5)(4,6)$. So, a base for K is $B=[1,2]$, and a B-ssgs is $[g_1,g_2]$.

Cyclically Extended Schreier Vectors

The advantage of a subnormal series is that elements of the group can be expressed as normal words. If we have a pcp then the collection process computes normal words, but if we are in a permutation representation how is the normal word computed? A base, strong generating set, and Schreier vectors allow us to compute a word in the strong generators for a permutation.

Lemma

Suppose $\vec{g}$ is a B-ssgs for a soluble permutation group G. Let $g \in G$ and suppose that w_1 and w_2 are words in the strong generators $\{g_1, g_2, ..., g_m\}$ for g. Then
(#occurrences of g_1 in w_1) = (#occurrences of g_1 in w_2) mod p_1.

In particular, we can calculate the exponent ε_1 of g_1 in the normal word for g from any word for g. Iterating with $g_1^{-\varepsilon_1} \times g$ will determine ε_2, and so on.

As an application of the Lemma, consider the element g=(1,2,3,4) in the symmetric group of degree 4. As $g = g_1 \times g_4 \times g_1 \times g_3 \times g_1 \times g_1 \times g_2 \times g_4 \times g_4 \times g_1$, the exponent $\varepsilon_1 = 1$. Then $g_1^{-\varepsilon_1} \times g$ = (1,2,3) satisfies the word $g_3 \times g_4 \times g_2 \times g_2 \times g_3 \times g_4$, so the exponent $\varepsilon_2 = 2$. Then $g_2^{-\varepsilon_2} \times g_1^{-\varepsilon_1} \times g$ = (1,2)(3,4) = g_4, so $\varepsilon_3 = 0$, and $\varepsilon_4 = 1$. Hence, the normal word for g is $g_1^{1} \times g_2^{2} \times g_3^{0} \times g_4^{1}$.

The next data structure to be described combines the Schreier vectors of all the subgroups in the subnormal series into just one set of Schreier vectors. Since g_i normalizes $G(i+1)$ and $g_i^{p_i} \in G(i+1)$ then

$$\{identity,\ g_i,\ g_i^2, ..., g_i^{p_i-1}\}$$

is a set of coset representatives for $G(i+1)$ in $G(i)$. Furthermore, suppose that l=*level*(g_i) is the level in the stabilizer chain which contains g_i. That is, $g_i \in G^{(l)} - G^{(l+1)}$. Define a set $U(i,l)$ of coset representatives of $G(i)^{(l+1)}$ in $G(i)^{(l)}$ recursively by

$$\{\ u \times g_i^j \ \mid\ u \in U(i+1,l),\ 0 \le j < p_i\ \},$$

and

$$U(m+1,l) = \{\ identity\ \},\ \text{for all } l.$$

We say that g_i *cyclically extends* $U(i+1,l)$ to $U(i,l)$.

A *cyclically extended Schreier vector* (cesv) is a Schreier vector $v^{(l)}$ that represents the set $U(i,l)$ for each i, $1 \le i \le m+1$.

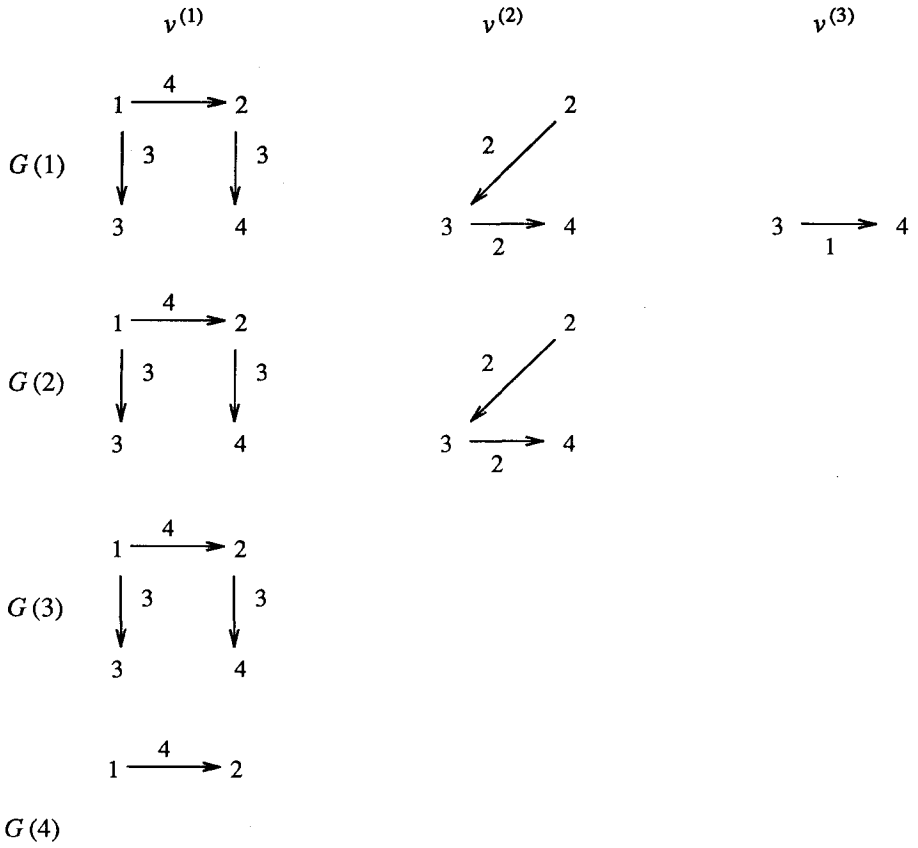
Consider again the example of S_4 with base B=[1,2,3] and B-ssgs $[g_1, g_2, g_3, g_4]$. The cyclically extended Schreier vectors are

	1	2	3	4
$v^{(1)}$		4	3	3
$v^{(2)}$			2	2
$v^{(3)}$				1

Note that we can think of the Schreier vector entries as either being the generator g_i, or just the index i of the generator in the B-ssgs. We have chosen the latter in this case, but it is often convenient to allow either notation.

By restricting to those entries which refer only to the generators $\{g_i, g_{i+1}, \ldots, g_m\}$, we have a set of Schreier vectors for the group $G(i)$. The diagrams below show the set of Schreier vectors for each group in the subnormal series of S_4 given by the B-ssgs $[g_1, g_2, g_3, g_4]$.

Cyclically Extended Schreier Vectors of S_4



The cyclically extended Schreier vectors can be calculated directly from the definition, by working up the subnormal series, as done in Algorithm 3. The cesv's can be used, as in Algorithm 4, to compute a normal word of an element g of the group G.

Algorithm 3 : Construct Cyclically Extended Schreier Vectors

Input : a base $B = [\beta_1, \beta_2, ..., \beta_k]$ for a group G;
a B-ssgs $[g_1, g_2, ..., g_m]$ for G;

Output : cyclically extended Schreier vectors $v^{(i)}$, i=1,2,...,k of G;

begin

for $l := 1$ **to** k **do**
initialize $v^{(l)}$ to all zero;
end for;

for $i := m$ **downto** 1 **do**
$l :=$ level(g_i); $b := |G(i) : G(i+1)|$;
for each point α in orbit of $v^{(l)}$ **do**
$\gamma := \alpha$;
for $j := 1$ **to** $b-1$ **do**
$\gamma := \gamma^{g_i}$;
$v^{(l)}[\gamma] := g_i$; (* or we could store just i *)
end for;
end for;
end for;

end.

Algorithm 4 : Normal Word

Input : a soluble permutation group G;
a base $B = [\beta_1, \beta_2, ..., \beta_k]$ for a group G;
a B-ssgs $[g_1, g_2, ..., g_m]$ for G;
cyclically extended Schreier vectors $v^{(i)}$ of G;
an element g of G;

Output : the exponents $[\varepsilon_1, \varepsilon_2, \cdots, \varepsilon_m]$ of the normal word of g;

begin

for $i := 1$ **to** m **do**
$l :=$ level(g_i);
(* strip g to level l *)
$h := g$;
for $j := 1$ **to** $l-1$ **do**
$\alpha := \beta_j{}^h$;
while $\alpha \neq \beta_j$ **do**
$h := h \times v^{(j)}[\alpha]^{-1}$;(* entry of cesv as a generator *)
$\alpha := \beta_j{}^h$;
end while ;
end for;
(* determine exponent ε_i *)
$\varepsilon_i := 0$; $\gamma := \beta_l{}^h$;
while $v^{(l)}[\gamma] = i$ **do** (* entry of cesv as an index *)
$\varepsilon_i := \varepsilon_i + 1$; $\gamma := \gamma^{g_i^{-1}}$;
end while;
(* residue is now in $G(i+1)$ *)
$g := g_i{}^{-\varepsilon_i} \times g$;
end for;

end.

There are occasions where we require a subnormal series to satisfy additional properties. The subnormal series will be obtained by modifying an existing B-ssgs in an iterative way. The simplest modification is where a B-ssgs $\vec{g} = [g_1, g_2, ..., g_m]$ is changed to $\bar{g} = [g_1, g_2, ..., g_{l-1}, h, g, g_{l+2}, g_{l+3}, ..., g_m]$. Of course, the elements g and h are chosen so that $\bar{g}$ is a B-ssgs. In the simplest case we may also choose them to lie in $G(l)$. Let $\bar{G}(i)$ denote the i-th term of the subnormal series defined by $\bar{g}$. Then $G(i) = \bar{G}(i)$, for all i except $i = l+1$. Hence, when we modify the cesv's to correspond to the new B-ssgs, there is very little to do. The details are in Algorithm 5.

Algorithm 5 : Changing a B-ssgs

Input : a soluble permutation group G with a base $B = [\beta_1, \beta_2, ..., \beta_k]$;
a B-ssgs $\vec{g} = [g_1, g_2, ..., g_m]$ and cesv's $v^{(i)}$ of G;
an integer l, and elements g,h in $G(l)$ such that
$G(l) = \langle g,h,g_{l+2}, g_{l+3}, ..., g_m \rangle$
and $[g_1, g_2, ..., g_{l-1}, h, g, g_{l+2}, g_{l+3}, ..., g_m]$ is a B-ssgs of G;
Output : cyclically extended Schreier vectors relative to new B-ssgs;

```
begin
  i := level(g_l);  i_1 := level(g_{l+1});
  if i = i_1 then
    b := |Ḡ(l+1) : G(l+2)|;  (* for new group Ḡ(l+1) *)
    (* Run through Schreier vector noting effect of g, the new g_{l+1} *)
    for each α in Ω do
      if v^(i)[α] ≥ l+2 then
        for j := 1 to b−1 do v^(i)[α^{g^j}] := '*'; end for;
      end if;
    end for;
    (* Run through noting effect of h, the new g_l, and correcting entries *)
    for each α in Ω do
      if v^(i)[α] = l+1 or l then v^(i)[α] := l; end if;
      if v^(i)[α] = '*' then v^(i)[α] := l+1; end if;
    end for;
  else (* One of g or h belongs to level i, and the other belongs to level i_1 *)
    if level(h) = i_1 then (* orbits don't change, just labels *)
      for each α in Ω do
        if v^(i_1)[α] = l+1 then v^(i_1)[α] := l; end if;
        if v^(i)[α] = l then v^(i)[α] := l+1; end if;
      end for;
    end if;
  end if;
end.
```

For example, the symmetries of the square has a base B=[1,3] and a B-ssgs $\vec{g}$ given by g_1=(3,4), g_2=(1,2)(3,4), and g_3=(1,3)(2,4). The cyclically extended Schreier vectors are

	1	2	3	4
$v^{(1)}$		2	3	2
$v^{(2)}$				1

Changing the B-ssgs to $\vec{g}$ = [(3,4), (1,3)(2,4), (1,4)(2,3)] gives the cesv's

	1	2	3	4
$v^{(1)}$		2	2	3
$v^{(2)}$				1

Changing the original B-ssgs $\vec{g}$ to $\bar{g}$ = [(3,4), (1,3)(2,4), (1,2)(3,4)] gives the cesv's

	1	2	3	4
$v^{(1)}$		3	2	2
$v^{(2)}$				1

Changing this B-ssgs to $\bar{g}$ = [(1,3)(2,4), (3,4), (1,2)(3,4)] gives the cesv's

	1	2	3	4
$v^{(1)}$		3	1	1
$v^{(2)}$				2

Using Homomorphisms

This section considers the interaction between homomorphisms and a gs or B-ssgs of a soluble group G. The first result follows directly from the fact that the kernel of a homomorphism is a normal subgroup.

Lemma

Let $f : G \longrightarrow H$ be a homomorphism. If $[g_1, g_2, \ldots, g_t]$ is a series-generating sequence for $ker(f)$, and $[f(h_1), f(h_2), \ldots, f(h_r)]$ is a series-generating sequence for $im(f)$, then $[h_1, h_2, \ldots, h_r, g_1, g_2, \ldots, g_t]$ is a series-generating sequence for G.

If both gs's define a prime-step series then so does the resulting gs $[h_1, h_2, \ldots, h_r, g_1, g_2, \ldots, g_t]$. A B-ssgs can be constructed from this sequence by repeated calls to Algorithm 1. This construction is presented in Algorithm 6.

Algorithm 6 : B-ssgs using Homomorphism

Input: soluble permutation groups G and H;
a base B for G;
a homomorphism $f : G \longrightarrow H$;
a prime-step gs $[g_1, g_2, \ldots, g_t]$ of $ker(f)$;
a prime-step gs $[f(h_1), f(h_2), \ldots, f(h_r)]$ of $im(f)$;
Output: a B-ssgs of G;
begin
$K := < identity >$;
for $i := t$ **downto** 1 **do**
$K := < K, g_i >$ using Algorithm 1;
end for;
(* now have B-ssgs of ker(f) *)

for $i := r$ **downto** 1 **do**
$K := < K, h_i >$ using Algorithm 1;
end for;
(* now have B-ssgs of G *)
end.

The transitive constituent homomorphism $f:G \longrightarrow G|_\Delta$, where $\Delta \subseteq \Omega$ is invariant under the soluble permutation group G, interacts nicely with B-ssgs's of the image and kernel. The following theorem gives the details.

Theorem

Let G be a soluble permutation group. Let Δ be a proper subset of Ω which is invariant under G. Let $f: G \longrightarrow G|_\Delta$ be the natural constituent homomorphism, and let $\bar{f}: \Delta \longrightarrow \{1,2,...,|\Delta|\}$ be the associated relabelling of points of Δ. Suppose $A = [\bar{f}(\alpha_1),\bar{f}(\alpha_2),...,\bar{f}(\alpha_k)]$ is a base for $im(f)$ and $[f(h_1),f(h_2),...,f(h_r)]$ is a prime-step A-strong series-generating sequence of $im(f)$. Further suppose that $B = [\beta_1,\beta_2,\ldots,\beta_l]$ is a base for $ker(f)$ chosen from $\Omega - \Delta$ and that $[g_1,g_2,\ldots,g_t]$ is a prime-step B-strong series-generating sequence of $ker(f)$. Let $AB = [\alpha_1,\alpha_2,\cdots,\alpha_k,\beta_1,\cdots,\beta_l]$. Then AB is a base for G and $[h_1,h_2,...,h_r,g_1,g_2,...,g_t]$ is a prime-step AB-strong series-generating sequence of G.

The blocks homomorphism has similar nice properties when G is a p-group and the block system is maximal. In this situation the image is a cyclic group of order p permuting the p blocks of the system as a p-cycle. Hence, the preimage of this p-cycle extends a B-ssgs of $ker(f)$ to a strong generating set of G (and hence to a B-ssgs of G).

Theorem

Let G be a transitive permutation p-group. Let π be a system of imprimitivity of G with p blocks. Let $f: G \longrightarrow G|_\pi$ be the natural blocks homomorphism. Let $f(g)$ be a generator of $im(f)$. Suppose $B = [\beta_1,\beta_2,\ldots,\beta_k]$ is a base for $ker(f)$ and that $[g_1,g_2,\ldots,g_r]$ is a B-strong series-generating sequence of $ker(f)$. Then B is a base for G, and $[g,g_1,g_2,...,g_r]$ is a B-strong series-generating sequence of G.

The only primitive permutation p-group is the cyclic group of order p acting regularly on p points.

The result about blocks homomorphisms does not extend nicely to soluble groups. We can still apply the Lemma to construct a gs, and then use Algorithm 6 to establish the strong generating property.

Computing with the Isomorphism

A B-ssgs $\vec{g} = [g_1, g_2, ..., g_m]$ defines an isomorphism f between the soluble permutation group G and a group H defined by the pc presentation

$$< a_1,a_2,...,a_m \mid a_i^{p_i} = \text{normal word of } g_i^{p_i},\ 1 \le i \le m,$$
$$a_j a_i = \text{normal word of } g_j g_i,\ 1 \le i < j \le m >,$$

where the normal words are written in terms of the generator symbols a_i rather than the B-ssgs generators g_i. The isomorphism is induced from its action on the generators, viz

$$f: G \longrightarrow H$$
$$g_i \longmapsto a_i$$

The computational tasks with the isomorphism are performed as follows:

1. Computing the image of the group: The image is H.

2. Computing the kernel: The kernel is the trivial subgroup of G.

3. Constructing the image of an element: Let $g \in G$. Form the normal word w of g using the B-ssgs and the cyclically extended Schreier vectors. Rewrite the word w in terms of the a_i to give the image $f(g) \in H$.

4. Constructing the preimage of an element: Let $h \in H$ be given by a normal word w in the a_i. Evaluate this word w using the permutations g_i to form $f^{-1}(h)$.

5. Constructing the image of a subgroup: Let $L \leq G$ be a subgroup generated by the set T of elements. Form $f(T) = \{ f(t) \mid t \in T \}$ and use non-commutative Gaussian elimination to form a canonical generating sequence (cgs) of the subgroup $f(L) = < f(T) >$ of H.

6. Constructing the preimage of a subgroup: Let $L \leq H$ be given by a cgs $[c_1, c_2, ..., c_r]$. A B-ssgs for $K = f^{-1}(L)$ is formed by

$K := <$ *identity* $>$;

for $i := r$ **downto** 1 **do**
  $K := < K, f^{-1}(c_i) >$ using Algorithm 1;
end for;

As an example of 6, consider the symmetric group of degree 4 with $B = [1,2,3]$ and $\vec{g} = [g_1, g_2, g_3, g_4]$. The subgroup $L = < a_1a_3, a_2a_4 >$ is isomorphic to S_3 and has cgs $[a_1a_3, a_2a_4]$. The preimages of these elements are $g_1g_3 = (1,2)$ and $g_2g_4 = (1,3,2)$. A B-ssgs of $K = f^{-1}(L)$ is [(2,3), (1,3,2)] because Algorithm 1 strips $g_1g_3 = (1,2)$ by g_2g_4 to give (2,3) ($= g_1g_2$).

Subnormal Series: p-Groups

Given a permutation p-group G, we can directly apply the results on homomorphisms to determine a base B and a B-ssgs of G describing a prime-step subnormal series of G.

Algorithm 7 : B-ssgs of a p-group

Input : a permutation p-group G with a base and strong generating set;
Output : a base B for G and a prime-step B-ssgs $\vec{g}$ for G;

```
procedure BSSGS( G : group; var B : base; var g⃗ : sequence of elements );
begin
 if G = <identity> then
     B := empty; g⃗ := empty;
 else if G has order p then
     let g be a generator of G;
     B[1] := a point moved by g; g⃗[1] := g;
 else if G is transitive then
     find a maximal system π of imprimitivity;
     let f be the blocks homomorphism G --> G|π;
     BSSGS (ker(f), B, g⃗);
     g⃗ := [f^-1(generator of im(f))] cat g⃗;
 else
     find a non-trivial orbit Δ of G;
     let f be the constituent homomorphism G --> G|Δ;
     let f be the associated relabelling Δ --> {1,2,...,Δ};
     BSSGS (ker(f), B, g⃗); BSSGS (im(f), A, h⃗);
     B := f^-1(A) cat B; g⃗ := f^-1(h⃗) cat g⃗;
 end if;
end (* BSSGS *);

begin
  B := empty; g⃗ := empty; BSSGS( G, B, g⃗ );
end.
```

As an example consider the symmetries G of the square generated by a=(1,4,3,2) and b=(2,4). A maximal system of imprimitivity is π = {1,3|2,4}. The image of the homomorphism f_π is generated by $f_\pi(a)$, and the kernel K is $< a^2, b >$. The kernel K has orbits Δ = {1,3} and {2,4}. The image of the homomorphism f_Δ is $<f_\Delta(a^2)>$, while the kernel is $<b>$. Hence, the group G has a base B=[1,2] and a B-ssgs $[a,a^2,b]$.

Subnormal Series: Soluble Groups

A primitive soluble permutation group G has several very specific properties:

i. the degree of G is a prime power p^s;

ii. G has a subgroup N such that

   a. N acts regularly on Ω;

   b. N has order p^s;

   c. N is elementary abelian;

d. N is normal in G;

e. the point stabiliser G_1 is isomorphic to the quotient G/N;

f. every element $g \in G$ can be uniquely expressed as $g = a \times g'$, where $a \in N$ and $g' \in G_1$;

The subgroup N is called an *earns* of G - for elementary abelian regular normal subgroup.

For example, the symmetric group S_4 of degree 4 has the Klein four-group <(1,2)(3,4),(1,3)(2,4)> as its earns.

Given a nonredundant generating set $\{a_1, a_2, \ldots, a_s\}$ of N, we can easily construct a B-ssgs of N by taking B=[β] for any point $\beta \in \Omega$, and taking the generators in any order whatsoever.

Given any element $a \in N$, we can construct N as

$$< a^g \mid g \in G >,$$

the normal closure of $<a>$ in G. However, it is not always easy to locate an initial element $a \in N$.

The subgroup N can also be characterised as $O_p(G)$, the intersection of all Sylow p-subgroups of G. So if S is a Sylow p-subgroup of G, then N is constructed by Algorithm 8. However, this approach can be expensive.

Algorithm 8 : EARNS as $O_p(G)$

```
Input: a primitive soluble permutation group G of degree p^s;
       a Sylow p-subgroup S of G;
Output: the elementary abelian regular normal subgroup (earns) N of G;
begin
  N := S;  notnormal := true;
  while notnormal do
    notnormal := false;
    for each generator g of G do
      if N^g ≠ N then
        notnormal := true; N := N ∩ N^g; break;
      end if;
    end for;
  end while;
end.
```

Having constructed N and a base B = [1] for N and a B-ssgs $\vec{a}$ of N, we can recursively compute a base A and an A-ssgs $\vec{g}$ of the point stabiliser G_1. The concatenation of these is a base $BA = B$ *cat* A and a BA-ssgs $\vec{g}$ *cat* $\vec{a}$ of G. The details are given in Algorithm 9. For example, let G be the symmetric group of degree 4. Then N is the Klein four-group $< g_3, g_4 >$. The point stabiliser has a base A = [2,3] and A-ssgs $[g_1, g_2]$, giving a base B = [1,2,3] for G and B-ssgs $[g_1, g_2, g_3, g_4]$.

Algorithm 9 : B-ssgs of a Soluble Group

Input : a soluble permutation group G with a base and strong generating set;

Output : a base B for G and a prime-step B-ssgs $\vec{g}$ for G;

procedure BSSGS(G : group; **var** B : base; **var** $\vec{g}$: sequence of elements);

begin
 if $G = \langle identity \rangle$ **then**
  $B :=$ empty; $\vec{g} :=$ empty;
 else if G is intransitive **then**
  find a non-trivial orbit Δ of G;
  let f be the constituent homomorphism $G \longrightarrow G^{\Delta}$;
  let f be the associated relabelling $\Delta \longrightarrow \{1,2,...,\Delta\}$;
  BSSGS $(ker(f), B, \vec{g})$; BSSGS $(im(f), A, \vec{h})$;
  $B := f^{-1}(A)$ **cat** B; $\vec{g} := f^{-1}(\vec{h})$ **cat** $\vec{g}$;
 else if G is imprimitive **then**
  find a system π of imprimitivity;
  let f be the blocks homomorphism $G \longrightarrow G^{\pi}$;
  BSSGS $(ker(f), B, \vec{g})$; BSSGS $(im(f), A, \vec{h})$;
  $K := ker(f)$;
  for $i :=$ length($\vec{h}$) **downto** 1 **do**
   $K := \langle K, h_i \rangle$ using Algorithm 1;
  end for;
 else
  compute earns N and a [1]-ssgs $\vec{a}$ of N;
  BSSGS $(G_1, B, \vec{h})$;
  $B :=$ [1] **cat** B; $\vec{g} := \vec{h}$ **cat** $\vec{a}$;
 end if;
end (* BSSGS *);

begin
 $B :=$ empty; $\vec{g} :=$ empty; BSSGS($G, B, \vec{g}$);
end.

An alternative approach, which avoids costly constructions of the earns as $O_p(G)$, handles the costly cases by constructing a normal subgroup M of G of prime index in G. The computation of M uses homomorphisms and the structure of primitive soluble permutation groups. The computation is described in Algorithm 10, and its use in computing a B-ssgs is presented in Algorithm 11.

Algorithm 10 : Prime Index

Input : a nontrivial soluble permutation group G
with a base and strong generating set;

Output : a prime p;
a normal subgroup M of G of index p;
an element $g \in G-M$;

procedure prime_index(G : group) : prime, subgroup, element;

begin
if G is intransitive **then**
find a non-trivial orbit Δ of G;
let f be the constituent homomorphism $G \longrightarrow G|_{\Delta}$;
$p, M, g := prime\_index(f(G))$;
return $p, f^{-1}(M), f^{-1}(g)$;
else if G is imprimitive **then**
find a maximal system π of imprimitivity;
let f be the blocks homomorphism $G \longrightarrow G|_{\pi}$;
$p, M, g := prime\_index(f(G))$;
return $p, f^{-1}(M), f^{-1}(g)$;
else
$H := G_1$;
if $H = \langle identity \rangle$ **then** (* G has prime order *)
let g generate G;
return $|G|, \langle identity \rangle, g$;
else if H has prime order **then** (* special case where N is easy to find *)
let h generate H;
find a generator g of G such that $a := [g,h] \neq identity$;
$N := \langle a, a^h, a^{h^2}, \ldots \rangle$;
return $|H|, N, h$;
else
$p, L, h := prime\_index(H)$;
$M :=$ normal closure of L in G;
return p, M, h;
end if;
end if;
end (* prime index *);

The normal closure computation in Algorithm 10 is particularly easy. The normal closure M is $N\,L = \{\, n \times l \mid n \in N,\ l \in L \,\}$, where N is the earns, so the point stabiliser of M is L, and the order of M is $|L| \times |\Omega|$. Hence, it suffices to add conjugates x^g, where x is a generator of L and g is a generator of G to the strong generating set of L until the group is transitive. There is no need to perform a Schreier-Sims method as we already have a strong generating set of the point stabiliser.

Algorithm 11 : B-ssgs of a Soluble Group using Prime Index

Input : a soluble permutation group G with a base and strong generating set;
Output : a base B for G and a prime-step B-ssgs $\vec{g}$ for G;

```
procedure BSSGS( G : group; var B : base; var g⃗ : sequence of elements );
begin
 if G = <identity> then
     B := empty; g⃗ := empty;
 else if G is intransitive then
     find a non-trivial orbit Δ of G;
     let f be the constituent homomorphism G ⟶ G|Δ;
     let f̄ be the associated relabelling Δ ⟶ {1,2,...,Δ};
     BSSGS (ker(f), B, g⃗); BSSGS (im(f), A, h⃗);
     B := f̄⁻¹(A) cat B; g⃗ := f⁻¹(h⃗) cat g⃗;
 else if G is imprimitive then
     find a system π of imprimitivity;
     let f be the blocks homomorphism G ⟶ G|π;
     BSSGS (ker(f), B, g⃗); BSSGS (im(f), A, h⃗);
     K := ker(f);
     for i := length( h⃗ ) downto 1 do
       K := <K, h_i > using Algorithm 1;
     end for;
 else
     H := G_1;
     if H = <identity> then
       N := G;
       B := [1]; g⃗ := nonredundant generators of N in any order;
     else if H has prime order then (* special case where N is easy to find *)
       let h generate H; let β be a point moved by h;
       find a generator g of G such that a := [g,h] ≠ identity;
       N := < a, a^h, a^(h^2), ... >;
       a⃗ := nonredundant generators of N in any order;
       B := [ 1, β ]; g⃗ := [ h ] cat a⃗;
     else
       p, L, h := prime_index( H ); M := normal closure of L in G;
       BSSGS ( M, B, g⃗ );
       M := < M, h > using Algorithm 1;
     end if;
 end if;
end (* BSSGS *);

begin
  B := empty; g⃗ := empty; BSSGS( G, B, g⃗ );
end.
```

For example, let G be the symmetric group of degree 4. Algorithm 11 will construct $H = G_1$ isomorphic to S_3, $L = \langle g_2 \rangle$ of index 2 in H, and $h = (3,4)$. When calculating the normal closure, the conjugate ${g_2}^{g_3} = (1,2,4)$ is added to the strong generating set, giving $M = \langle (2,3,4),(1,2,4) \rangle$ isomorphic to the alternating group A_4.

In the recursive call to *BSSGS* with A_4, the algorithm executes the special case. It finds the commutator $a = [(1,2,4),(2,3,4)] = (1,4)(2,3)$ and constructs the Klein four-group $\langle (1,4)(2,3),(1,2)(3,4) \rangle$. Thus it returns a base $B=[1,2]$ and a B-ssgs [(2,3,4), (1,4)(2,3), (1,2)(3,4)].

On return from the recursive call, the call to Algorithm 1 with $h=(3,4)$ does not alter h, but it does extend the base B to [1,2,3]. The resulting B-ssgs is [(3,4), (2,3,4), (1,4)(2,3), (1,2)(3,4)].

Subnormal Series: Soluble Schreier Method

One can work directly from a generating set S of a soluble permutation group G to construct a B-ssgs. The algorithm comes straight from

a. the definition of a soluble group in that G has a proper normal subgroup N such that G/N is abelian (and N is soluble); and

b. the use of Algorithm 2 to construct a B-ssgs of the extensions of N formed during the execution of the algorithm. The conditions of Algorithm 2 are fulfilled because N is normal in G.

The algorithm constructs a subgroup M such that $N \leq M \leq G$. Eventually $M = G$. Given an element $y \in G-N$, the subgroup M is the normal closure of $\langle N,y \rangle$ in G. During the course of this computation N may be updated, and at the conclusion of the computation M is normal in G.

The computation of M forms the normal closure (as usual) by considering the set U of conjugates of y under generators of G (and their iterated products). The properties of N are maintained so that $N \lhd M$ and M/N is abelian, by ensuring that all commutators $[u,v]$ with $u,v \in U$ are in N. If a commutator $w = [u,v]$ does not lie in N, then we recursively compute the normal closure of $\langle N,w \rangle$ in G as the updated value of N. The details are presented as Algorithm 12.

Algorithm 12 : Soluble Schreier Method

Input: a soluble permutation group G given by generators S;
Output: a base B for G and a B-ssgs $\vec{g}$ of G;
begin
  $N :=$ < *identity* >; $B :=$ empty; $\vec{g} :=$ empty;
  for each s in S **do**
    sol_normal_closure(G, N, s);
  end for;
  (* N is G *)
end.

```
procedure sol_normal_closure( G : group; var N : group; y : element of G );

(* Compute M, the normal closure of <N,y> in G where N ◁ G.
  We always have M/N is abelian during the course of the procedure.
  At the end of the procedure, M is returned as N. *)
begin
  M := N;  U := {};  Z := {y};
  while Z is not empty do
    choose z ∈ Z;  Z := Z - {z};
    if z ∉ M then
      T := U;  done := false;
      while T is not empty do
        choose u ∈ T;  T := T - {u};  w := [u,z];
        if w ∉ N then
          sol_normal_closure( G, N, w );
          V := {};  M := N;
          for each v in U do
            if v ∉ M then
              M := <M,v> using Algorithm 2; V := V ∪ {v};
            else
              T := T ∪ {v};
            end if;
          end for;
          U := V;
          if z ∈ M then
            done := true; break out of while T not empty loop;
          end if;
        end if; (* w ∉ N *)
      end while; (* T not empty *)
      if not done then
        M := <M,z> using Algorithm 2; U := U ∪ {z};
        Z := Z ∪ { s^-1 × z × s : s is a generator of G };
      end if; (* not done *)
    end if; (* z ∉ M *)
  end while; (* Z not empty *)
  N := M;
end (* sol_normal_closure *);
```

For example, consider the group $G = \langle(1,2),(1,2,3,4)\rangle$, the symmetric group of degree 4. The first call to *sol_normal_closure* computes the normal closure of $\langle(1,2)\rangle$. Initially, N and M are both trivial, and $y = (1,2)$, the first generator. The first iteration of the outer **while**-loop has $z=(1,2)$ and U empty, so M is updated by Algorithm 2 to $\langle(1,2)\rangle$, and the elements $z^{(1,2)}$ = *identity* and $z^{(1,2,3,4)} = (2,3)$ are added to Z. Let us ignore the identity element as it is in M, so the next iteration of the outer **while**-loop has $z=(2,3)$ and $U=\{(1,2)\}$. The commutator $w = [\,(1,2), (2,3)\,] = (1,2,3)$, which is not in N, so we recursively compute N = normal closure of $\langle(1,2,3)\rangle$ in G. When we return from the recursive call, N is the alternating group of degree 4, and M is updated to G. Hence, $z \in M$ and the algorithm is finished.

The recursive call initially has N and M trivial, and $y = (1,2,3)$. The first iteration of the outer **while**-loop updates M to be the cyclic group $\langle(1,2,3)\rangle$ and $U = \{(1,2,3)\}$, and $Z = \{(2,3,4)\}$ (ignoring (1,3,2), the conjugate of (1,2,3) by (1,2), because it is in M). The next iteration of the outer **while**-loop checks the commutator $w = [\ (1,2,3),\ (2,3,4)\ ] = (1,4)(2,3)$. The commutator is not in N, so *sol_normal_closure* is called recursively to form the normal closure of $\langle(1,4)(2,3)\rangle$ in G. When the call returns, N is the Klein four-group $\langle(1,4)(2,3),(1,3)(2,4)\rangle$, and M is updated to be the alternating group of degree 4. The element $z = (2,3,4)$ is in M, so the algorithm is finished.

The recursive call for $y=(1,4)(2,3)$ first creates the cyclic group $\langle(1,4)(2,3)\rangle$ and sets $U = \{(1,4)(2,3)\}$ and $Z = \{(1,3)(2,4),(1,2)(3,4)\}$. The commutator of $u=(1,4)(2,3)$ and $z=(1,3)(2,4)$ is trivial, so Algorithm 2 updates M to $\langle(1,4)(2,3),(1,3)(2,4)\rangle$, the Klein four-group, $U = \{$ (1,4)(2,3), (1,3)(2,4) $\}$, and $Z = \{$ (1,2)(3,4), (1,4)(2,3), (1,3)(2,4) $\}$. All the elements of Z are in M, so the algorithm quickly terminates.

The depth of recursion of Algorithm 12 is bounded by the length of the derived series of G. Bounds (in terms of the degree $|\Omega|$) are known on the length of the derived series of a soluble permutation group of degree $|\Omega|$. This information can be used to generalize the algorithm to apply even when we are not certain that the group is soluble. If the group is soluble, then the algorithm will terminate normally. If the group is not soluble then the bound on the depth of recursion will be exceeded.

Conditioned PC Presentation of a p-Group

The subnormal series associated with a conditioned pcp of a p-group must be prime-step and central. We have seen how to achieve the first criterion. The second criterion requires that

$$j > i \text{ implies } [g_j,g_i] \in G(j+1).$$

So we ensure the criterion is satisfied by repeatedly testing if the commutator $[g_j,g_i] \in G(j+1)$ and whenever the commutator is not in the $j+1$-st term of the series, then the series is modified by placing $[g_j,g_i]$ as the new g_j (and eliminating some now redundant generator). This process, described in Algorithm 13, converges to a central series.

A group of order p^2 is abelian, so we always have $[g_j,g_{j-1}] \in G(j+1)$.

Algorithm 13 : Central Series of a p-group

```
Input: a permutation p-group G with a base B;
    a prime-step B-ssgs [g_1, g_2, ..., g_m] of G;
Output: a prime-step B-ssgs [g_1, g_2, ..., g_m] of G with a central series;
begin
  for j := m downto 3 do
    for i := j-2 downto 1 do
      while not [g_j,g_i] ∈ <g_{j+1}, g_{j+2}, ..., g_m> do
        g := [ g_j, g_i ]; k := li( g ); (* the leading index of g as a normal word *);
        delete g_k;      insert g between g_{j+1} and g_j;
      end while;
    end for;
  end for;
end.
```

To preserve the B-strong property, the insertion of the commutator is done by repeatedly transposing elements (beginning at level k) until g is in the correct position. The B-ssgs change algorithm is used for each transposition.

As an example of Algorithm 13 consider the symmetries G of the square which has a base B=[1,3] and a B-ssgs $\vec{g}$ given by g_1=(3,4), g_2=(1,2)(3,4), and g_3=(1,3)(2,4). The commutator $[g_3,g_1]$ is g_2, so we change the B-ssgs to $[g_1,g_3,g_2]$. Now the commutator is the identity, and the subnormal series is central.

Conditioned PC Presentation of a Soluble Group

A conditioned pc presentation of a soluble group requires that the series associated with the pcp refines a normal series with elementary abelian factors. The easiest way to achieve this is for the normal series to be a refinement of the derived series. The factors of the derived series are abelian, and the abelian factors can be refined to elementary abelian ones (as in Algorithm 14).

Algorithm 14 : Refining an Abelian Factor

Input: a group G with a normal subgroup N such that G/N is abelian;
a prime-step B-ssgs $\vec{g} = [g_1, g_2, ..., g_m]$ which extends N to G;

Output: a refinement of G/N into elementary abelian factors
and the corresponding B-ssgs;

```
begin
  while N ≠ G do
    choose a prime p dividing the order of G/N;
    M := N;
    for each g in g⃗ but not in N do
      if g^p ∈ N do
        M := <M,g> using Algorithm 1;
      end if;
    end for;
    N := M;  (* the next term in the series with e.a. factors *)
  end while;
end.
```

The soluble Schreier method can be modified to keep track of the derived series and return a B-ssgs with an associated subnormal series that refines the derived series. The modifications are presented in Algorithm 15.

Algorithm 15 : Modified Soluble Schreier Method

Input: a soluble permutation group G given by generators S;
Output: a base B for G and a B-ssgs $\vec{g}$ of G such that
the series refines the derived series of G;

```
begin
  N := < identity >;  B := empty;  g⃗ := empty;
  for each s in S do
    derived_normal_closure( G, N, s );
  end for;
  (* N is G *)
end.

procedure derived_normal_closure( G : group; var N : group; y : element of G );

(* Compute M, the normal closure of <N,y> in G where N ◁ G.
  We always have M/N is abelian during the course of the procedure.
  Assume we know the derived series of N and a corresponding base
  and B-ssgs. We maintain D(M) = D(N).
  At the end of the procedure, M is returned as N. *)

begin
  M := N;  U := generators extending D(N) to N;  Z := {y};
  while Z is not empty do
    choose z ∈ Z;  Z := Z - {z}; T := U;
      while T is not empty do
        choose u ∈ T;  T := T - {u};  w := [u,z];
        if w ∉ D(N) then
          derived_normal_closure( G, D(N), w ); M := D(N);
          V := {};
          for each v in U do
            if v ∉ M then
              M := <M,v> using Algorithm 2; V := V ∪ {v};
            end if;
          end for;
          U := V; (* generators extending D(N) =D(M) to M *)
        end if; (* w ∉ N *)
      end while; (* T not empty *)
      if not z ∈ M then
        M := <M,z> using Algorithm 2; U := U ∪ {z};
        Z := Z ∪ { s^-1 × z × s : s is a generator of G };
      end if;
  end while; (* Z not empty *)
  N := M;
end (* derived_normal_closure *);
```

Summary

This chapter has introduced the data structures of B-strong series-generating sequence and cyclically extended Schreier vectors as a representation of a soluble permutation group. Algorithms to construct the data structure, compute normal words, and compute with the isomorphism between the permutation group and the group defined by the pc presentation have been presented.

Exercises

(1/Easy) Consider the group G of order 2^7 and degree 8 generated by

g_7=(2,6)(4,5),
g_6=(4,5)(7,8),
g_5=(1,3)(2,6)(4,5)(7,8),
g_4=(4,7)(5,8),
g_3=(1,2)(3,6)(4,7)(5,8),
g_2=(1,7)(2,4)(3,8)(5,6),
g_1=(7,8).

The group has a base B=[1,2,4,7] and the sequence $[g_1,g_2,...,g_7]$ is a B-ssgs.

(a) Construct the cyclically extended Schreier vectors of G.

(b) Form the normal word for y=(1,4,6,8,3,5,2,7).

(c) Let y=(1,7,6,5,3,8,2,4), and let $H = G(2) = \langle g_2,g_3,...,g_7 \rangle$. The square of y lies in H. Use Algorithm 1 to construct a B-ssgs of $\langle H,y \rangle$.

(d) Let y=(1,7,6,5,3,8,2,4), and let $H = G(5) = \langle g_5,g_6,g_7 \rangle$. Use Algorithm 2 to construct a B-ssgs of $\langle H,y \rangle$.

(2/Moderate) For the group G of Exercise 1 with the given base B and B-ssgs, perform a sequence of changes of B-ssgs (by transposing adjacent members of the B-ssgs) until the final B-ssgs is $[g_1,g_4,g_6,g_7,g_2,g_3,g_5]$.

(3/Easy) Let $G = \langle a$=(1,2,3,4)(5,6,7,8), b=(1,5,3,7)(2,8,4,6) $\rangle$ of degree 8 and order 8. Execute Algorithm 7 to construct a base B and B-ssgs of G. Is the associated pc presentation conditioned?

(4. Moderate) For the group G of Exercise 1, perform Algorithm 7 and construct a base B and a B-ssgs of G.

Then perform Algorithm 13 (with appropriate uses of Algorithm 5) to construct a B-ssgs that gives a conditioned pc presentation of G (regarded as a p-group).

(5/Moderate) Let G be the soluble group of degree 6 and order 72 generated by

a=(1,2,3),
b=(2,3),
c=(1,4)(2,6,3,5).

(a) Execute Algorithm 12 to construct a base B=[1,2,4,5] and B-ssgs of G.

(b) Execute Algorithm 15 and Algorithm 14 to construct a B-ssgs of G such that the associated pc presentation is conditioned.

(c) Execute Algorithm 9 with G as the input group.

(d) Execute Algorithm 10 to find a normal subgroup M of G of prime index.

(6/Difficult) A complete labelled branching combines a set of Schreier vectors for a stabilizer chain into a single data structure, while a cesv combines the Schreier vectors of a subnormal series for a fixed level of their stabilizer chains. Can the two data structures be reconciled into one for a soluble permutation group?

Bibliographical Remarks

The first success in converting between a permutation representation and a pc presentation was by D.F. Holt in 1982 as part of his work on Schur multipliers: D.F. Holt, *"A computer program for the calculation of the Schur multiplier of a permutation group"*, **Computational Group Theory,** M.D. Atkinson (ed.), Academic Press, Academic Press, 1984, pp. 307-319. Given a B-ssgs of a Sylow p-subgroup, his algorithm produced a conditioned pc presentation by making the subnormal series prime-step and central. He also realised that cyclically extended Schreier vectors could represent all groups in the series and produce a normal word for a permutation. A proof of the algorithm is given in G. Butler, *"A proof of Holt's algorithm"*, J. Symbolic Comp. **5** (1988) 275-283. The final step for permutation p-groups of constructing a B-ssgs from an arbitrary base and strong generating set was done by G. Butler and J.J. Cannon, *"On Holt's algorithm"*, J. Symbolic Comp. to appear.

The search for generalizations of these techniques to soluble groups began in 1985 with G. Butler, *"Data structures and algorithms for cyclically extended Schreier vectors"*, Congressus Numerantum **52** (1986) 63-78, which started with a derived series. The derived series was (is) too expensive to compute. That cost altered with the soluble Schreier method of Sims in 1987 published as C.C. Sims, *"Computing the order of a solvable permutation group"*, J. Symbolic Comp. **9** (1990) 699-705. Sims' algorithm computes a B-ssgs, and can be modified so that the corresponding pc presentation of the soluble group is conditioned. The modifications were done by G. Butler, *"Computing a conditioned pc presentation of a soluble permutation group"*, TR 392, Basser Department of Computer Science, University of Sydney, 1990, 5 pages.

Also in 1985, W.M. Kantor, *"Sylow's theorem in polynomial time"*, J. Comput. System Sci. **30** (1985) 359-394, in the appendix, presents an algorithm for finding a normal subgroup of prime index in a soluble permutation group. The algorithm uses homomorphisms to reduce to the primitive case where the structure of the group is known. His treatment of the primitive case completes Algorithm 11 for constructing a B-ssgs in a way which is analogous to Algorithm 7 for permutation p-groups. The implementation details (mainly to avoid calls to Schreier-Sims method) are presented in G. Butler, *"Implementing some algorithms of Kantor"*, 1991.

The structure of a primitive soluble group is discussed in Satz 3.2 of B. Huppert, **Endliche Gruppen I**, Springer-Verlag, Berlin, 1967, p.159. Bounds on the order of such groups are given by L. Palfy, *"A polynomial bound on the order of primitive soluble groups"*, J. Algebra **77** (1982) 127-137 and T.R. Wolf, *"Solvable and nilpotent subgroups of $GL(n,q^m)$"*, Canadian

J. Math. **34** (1982) 1097-1111. J.D. Dixon, "*The solvable length of a solvable linear group*", Math. Zeitschrift **107** (1968) 151-158, bounds the derived length of a soluble permutation group of degree $|\Omega|$ by $2.5 \log_3(|\Omega|)$.

The computation of $O_p(G)$ is presented in G. Butler and J.J. Cannon, "*Computing with permutation and matrix groups III: Sylow subgroups*", J. Symbolic Comp. **8** (1989) 241-252, while P.M. Neumann, "*Some algorithms for computing with permutation groups*", **Groups - St Andrews 1985**, E.F. Robertson and C.M. Campbell (eds), London Mathematics Society Lecture Notes **121**, Cambridge University Press, Cambridge, 1986, pp. 59-92, discusses an alternative approach to constructing an elementary abelian regular normal subgroup of a primitive group.

The series-generating sequences are called "smooth generating sequences" by Z. Galil, C.M. Hoffman, E.M. Luks, C.P. Schnorr, and A. Weber, "*An $O(n^3 \log n)$ deterministic and an $O(n^3)$ Las Vegas isomorphism test for trivalent graphs*", Journal of the ACM **34**, 3 (1987) 513-531, who use them to describe 2-groups given by permutations.