ON THE DIMENSION OF MATRIX REPRESENTATIONS OF FINITELY GENERATED TORSION FREE NILPOTENT GROUPS

MAGGIE HABEEB AND DELARAM KAHROBAEI

Abstract. It is well known that any polycyclic group, and hence any finitely generated nilpotent group, can be embedded into $GL_n(\mathbb{Z})$ for an appropriate $n \in \mathbb{N}$; that is, each element in the group has a unique matrix representation. An algorithm to determine this embedding was presented in [6]. In this paper, we determine the complexity of the crux of the algorithm and the dimension of the matrices produced as well as provide a modification of the algorithm presented in [6].

1. Background Information

In this section we will review basic facts about polycyclic and nilpotent groups. We refer the reader to [3] and [2] for more information on these groups.

1.1. Polycyclic Groups.

Definition 1. [3] *A group is called polycyclic if it admits a finite subnormal series*

$$G = G_1 \triangleright G_2 \triangleright G_3 \triangleright \cdots \triangleright G_{n+1} = 1$$

where each G_i/G_{i+1} is cyclic.

The number of infinite factors in the polycyclic series is called the *Hirsch length*, and is independent of the polycyclic series chosen. Since each factor is cyclic there exists an $x_i \in G$ such that $\langle x_i G_{i+1}\rangle = G_i/G_{i+1}$. We call the sequence $X = [x_1, x_2, \cdots, x_n]$ a polycyclic sequence for G. The sequence of relative orders of X is the sequence $R(X) = (r_1, \cdots, r_n)$ where $r_i = [G_i : G_{i+1}]$. We denote the set of indices in which r_i is finite by $I(X)$.

Polycyclic groups have finite presentation

$$\left\langle a_1, \cdots, a_n; a_i^{a_j} = w_{ij}, a_i^{a_j^{-1}} = v_{ij}, a_k^{r_k} = u_{kk} \text{ for } k \in I, 1 \leq j < i \leq n \right\rangle$$

Date: August 16, 2021.

Research of M.H. was supported partially by the NSF-LSAMP fellowship.

Research D.K. was partially supported by the Office of Naval Research grant N000141210758, PSC-CUNY grant from the CUNY research foundation, as well as the City Tech foundation.

where $r_i \in \mathbb{N} \cup \infty$, $r_i < \infty$ if $i \in I \subseteq \{1, 2, \cdots, n\}$ and w_{ij}, v_{ij}, u_{jj} are words in the generators $a_{j+1}, \cdots, a_n$. The relations $a_i^{a_j} = w_{ij}, a_i^{a_j^{-1}} = v_{ij}$ are called conjugacy relations, while the relations $a_k^{r_k} = u_{kk}$ are called power relations. If $r_i = [G_i : G_{i+1}]$ for each $i \in \{1, 2, \cdots, n\}$ then this presentation is called a consistent polycyclic presentation. Every polycyclic group admits a consistent polycyclic presentation, which results in the following normal form.

Definition 2. [3] *Let $X = [x_1, \cdots, x_n]$ be a polycyclic sequence for G and $R(X) = (r_1, \cdots, r_n)$ be its sequence of relative orders. Then every $g \in G$ can be written uniquely in the form $g = x_1^{e_1} \cdots x_n^{e_n}$ with $e_i \in \mathbb{Z}$ and $0 \leq e_i < r_i$. This expression is called the normal form of G with respect to X.*

The normal form for an element in a group given by a consistent polycyclic presentation can be determined via the *collection algorithm*. Thus, the collection algorithm gives a solution to the word problem for a group G given by a consistent polycyclic presentation. The collection algorithm works by iteratively applying the power and conjugacy relations of the presentation until the normal form for a word is obtained. The nature of the power and conjugacy relations ensure that the collection algorithm terminates (see [3]). The collection algorithm is currently the best known way to obtain the normal form of an element in a polycyclic group, although the worst case time complexity is unknown.

By using different representations of elements in a polycyclic group one can solve some group theoretic problems such as the conjugacy search problem. This can be done by using a different representation of group elements. For example to find a different representation of group elements one can use the well known fact, due to L. Auslander (see [4]), that every polycyclic group is linear; that is, every polycyclic group G can be embedded into $GL_n(\mathbb{Z})$ for an appropriate n.

1.2. **Nilpotent Groups.**

Definition 3. [2] *A group, G, is called nilpotent of class $c \geq 1$ if*

$$[y_1, y_2, \cdots, y_{c+1}] = 1 \text{ for any } y_1, y_2, \cdots, y_{c+1} \in G,$$

where $[y_1, y_2, y_3] = [[y_1, y_2], y_3]$. Equivalently we may define a nilpotent group as follows. We define a lower central series of G inductively: let $\gamma_1(G) = G$, $\gamma_2(G) = [G, G]$, and $\gamma_k(G) = [\gamma_{k-1}(G), G]$. If $\gamma_k(G) = \{e\}$ for some k, then G is nilpotent.

Finitely generated nilpotent groups are in fact polycyclic. Let G be a finitely generated torsion-free nilpotent group. Since G is torsion-free and finitely generated nilpotent, there exists a central series

$$G = G_1 \triangleright G_2 \triangleright G_3 \cdots \triangleright G_{n+1} = \{1\}$$

such that for each $1 \leq r \leq n$ the factor G_r/G_{r+1} is infinite cyclic generated by $x_r G_{r+1}$ for some $x_r \in G_r$. Given such $x_1, \cdots, x_n$ each element $x \in G$ has unique normal form

$$x = x_1^{e_1} \cdots x_n^{e_n},$$

where $e_i \in \mathbb{Z}$. The n-tuple $(e_1, \cdots, e_n)$ is called the vector of exponents of x.

As mentioned above all finitely generated nilpotent groups are polycyclic, and hence admit a polycyclic presentation. In particular every finitely generated nilpotent group admits a presentation with conjugacy relations of the form:

$$x_i^{x_j} = x_i x_{i+1}^{b_{i,j,i+1}} \cdots x_n^{b_{i,j,n}} \text{ for } 1 \leq j < i \leq n,$$

$$x_i^{x_j^{-1}} = x_i x_{i+1}^{c_{i,j,i+1}} \cdots x_n^{c_{i,j,n}} \text{ for } 1 \leq j < i \leq n,$$

where the $c_{i,j,k}, b_{i,j,k} \in \mathbb{Z}$. A polycyclic presentation with these conjugacy relations is called a *nilpotent presentation*. Since G is torsion free, there exists a consistent nilpotent presentation with each $r_i = \infty$. Hence, for any finitely generated torsion-free nilpotent group we may find a consistent nilpotent presentation of the form:

$$\left\langle x_1, \cdots, x_n; x_i^{x_j} = x_i x_{i+1}^{b_{i,j,i+1}} \cdots x_n^{b_{i,j,n}}, x_i^{x_j^{-1}} = x_i x_{i+1}^{c_{i,j,i+1}} \cdots x_n^{c_{i,j,n}} \text{ for } 1 \leq j < i \leq n \right\rangle.$$

Let G be a finitely generated torsion free nilpotent group with a presentation as above. Then each $x \in G$ may be written uniquely in the form $x = x_1^{e_1} \cdots x_n^{e_n}$. Given two elements $x = x_1^{e_1} \cdots x_n^{e_n}, y = x_1^{y_1} \cdots x_n^{y_n}$, their product can be written uniquely as:

$$xy = x_1^{e_1} \cdots x_n^{e_n} x_1^{y_1} \cdots x_n^{y_n} = x_1^{z_1} \cdots x_n^{z_n}.$$

Philip Hall [2] showed that there are rational polynomials $f_1, \cdots, f_n$ that describe the multiplication of elements in G, a finitely generated torsion free nilpotent group. In particular, if $(e_1, \cdots, e_n)$, $(y_1, \cdots, y_n)$, and $(z_1, \cdots, z_n)$ are the vector of exponents for $x, y,$ and xy, respectively, then for $1 \leq r \leq n$ we have

$$z_r = f_r(e_1, \cdots, e_n, y_1, \cdots, y_n).$$

In [5] Leedham-Green and Soicher show how to compute polynomials in variables corresponding to the $e_i, y_i, c_{i,j,k}$ such that they describe the multiplication of elements in the group G by an algorithm referred to as "Deep Thought". In [6] W. Nickel presents an algorithm for computing a presentation of a finitely generated torsion free nilpotent group given by a polycyclic presentation by unitriangular matrices over $\mathbb{Z}$. The algorithm by Nickel uses the polynomials computed by "Deep Thought." There is another algorithm due to DeGraaf and Nickel that computes a faithful unitriangular representation of finitely generated torsion-free nilpotent groups in [1], but the algorithm presented by Nickel in [6] is more efficient.

2. Matrix Representations of Finitely Generated Torsion-free Nilpotent Groups

W. Nickel introduced an algorithm to compute a matrix representation for a finitely generated torsion-free nilpotent group given by a nilpotent presentation in [6]. The crux of the algorithm computes a $\mathbb{Q}$-basis for a finite dimensional faithful G-module, where G is a finitely generated torsion-free nilpotent group given by a nilpotent presentation.

In [6], a faithful finite dimensional G-module is constructed as follows. Let G be a finitely generated torsion-free nilpotent group with nilpotent generating sequence $a_1, \cdots, a_n$ and multiplication polynomials $q_1, \cdots, q_n$. Let G act on the dual of the group ring $(\mathbb{Q}G)^*$ by defining $f^g(h) := f(hg^{-1})$. By identifying $a_1^{x_1} \cdots a_n^{x_n}$ with $x_1, \cdots, x_n$ the image of $f \in (\mathbb{Q}G)^*$ under the action of G can be described using the multiplication polynomials $q_1, \cdots, q_n$; that is, $f^g = f(q_1, \cdots, q_n)$. With this action one can construct a finite dimensional faithful G-submodule of $(\mathbb{Q}G)^*$ by using the following lemma.

Lemma 4. [6] *The submodule M of $(\mathbb{Q}G)^*$ generated by $t_i : G \to \mathbb{Z}$ with $t_i(a_1^{x_1} \cdots a_n^{x_n}) = x_i$ is a finite dimensional faithful G-module.*

To construct a basis a method called *Insert* is used. *Insert* adds a polynomial to a given basis of polynomials if that polynomial is not in the span of the basis; that is, if the polynomial cannot be written as a linear combination of the basis elements. This is done by first placing an ordering on the monomials. The leading monomial of a polynomial is then the monomial that is largest with respect to the ordering. *Insert* takes a basis of polynomials in ascending order and a polynomial f, and determines if f is in the span of the basis. If f is not in the span of the basis, then it will be added to the basis. *Insert* determines if f is in the span of the basis by subtracting the appropriate multiple of each polynomial in the basis, starting with the polynomial that is largest with respect to the ordering and continuing in descending order. If $f \neq 0$ after this process is completed, then f is not in the span of the basis and is added to the original basis.

In order to determine a $\mathbb{Q}$-basis for a finite dimensional G-module generated by $f_1, \cdots, f_k$, the algorithm uses *Insert* to build up a basis from $f_1, \cdots, f_k$. Then the algorithm works up the central series of G beginning with G_n. For each j the algorithm will compute a basis for the G_j-module M_j from a basis of the G_{j+1}-module M_{j+1} generated by $f_1, \cdots, f_k$. In order to obtain a generating set for M_j it is enough to close the module M_{j+1} under the action of powers of a_j (since G_j/G_{j+1} is cyclic) and to apply powers of a_j to each basis element of M_{j+1}. The algorithm for computing a $\mathbb{Q}$-basis is presented in Figure 1.

The matrix representation for each generator a_k of G can be calculated by decomposing the image of each basis element under a_k in terms of the basis. The ordering utilized in *Insert* is the reverse lexicographic ordering; that is, $x_1^{k_1} \cdots x_n^{k_n} < x_1^{l_1} \cdots x_n^{l_n}$ if there is an i $(1 \leq i \leq n)$ such that $k_j = l_j$ for $i < j \leq n$ and $k_i < l_i$. Suppose that $b_1, \cdots, b_m$ is the basis produced

FIGURE 1. Matrix Representation Algorithm (building a basis), [6]

Input: A finitely generated torsion- free nilpotent group G with a nilpotent generating sequence $a_1, \cdots, a_n$ and corresponding multiplication polynomials $q_1, \cdots, q_n$; a list of polynomials $f_1, \cdots, f_k$.

Output: A $\mathbb{Q}$-basis B for the G-module generated by $f_1, \cdots, f_k$.

Algorithm:

$B := []$;
for j **in** $[1 \cdots k]$ **do** Insert(B, f_j); **od**;
for j **in** $[n, n-1 \cdots 1]$ **do**
  #B is a basis for M_{j+1}
  # Exponents after multiplication by a_i^{-1} from the right:
  $q^{(j)} := [q_1(x_1, \cdots, x_n, y_i = -\delta_{ij}), \cdots, q_n(x_1, \cdots, x_n, y_i = -\delta_{ij})]$;
  for f **in** Copy(B) **do**
    # Add to B images of f under powers of a_j
    repeat
      # Compute f^{a_j}
      $f^{a_j} := f(q_1^{(j)}, \cdots, q_n^{(j)})$;
      r:=Insert(B, f^{a_j});
      $f := f^{a_j}$;
    until r=0;
  od;
od;
return B;

using this ordering. Let $q_1^{(j)}, \cdots q_n^{(j)}$ be the polynomials that describe the multiplication of an arbitrary element of the group with a_j^{-1}:

$$a_1^{x_1} \cdots a_n^{x_n} \cdot a_j^{-1} = a_1^{q_1^{(j)}} \cdots a_n^{q_n^{(j)}}$$

The polynomials $q_i^{(j)}$ are obtained from the multiplication polynomials q_i by setting $y_i = 0$ if $i \neq j$ and $y_i = -1$ if $i = j$. Then for each basis element b_k, one can compute $b_k(q_1^{(j)}, \cdots, q_n^{(j)})$ and decompose this element into a linear combination $c_{k_1} b_1 + \cdots + c_{k_n} b_m$ of basis vectors. The coefficients $c_{k_1}, \cdots, c_{k_n}$ are the k^{th} row of the matrix representing a_j. For more information on this algorithm see [6].

3. Complexity Analysis

The algorithm for finding matrix representations of torsion-free finitely generated nilpotent groups formulated in [6] was implemented in the **GAP** package polycyclic. The algorithm was implemented using the coordinate functions $t_i : G \to \mathbb{Z}$ given by $a_1^{x_1} \cdots a_n^{x_n} \mapsto x_i$ for $1 \leq i \leq n$ as the input polynomials $f_1, \cdots, f_k$. In order to analyze the complexity of the

algorithm utilized for building a $\mathbb{Q}$-basis (see Figure 1), we must analyze how the action of $a_j \in G$ affects each coordinate function t_i. Recall that $t_i^{a_j} := t_i(q_1^{(j)}, \cdots, q_n^{(j)})$ where $q_i^{(j)} = q_i(x_1, \cdots, x_n, y_i = -\delta_{ij})$ and the q_i are the multiplication polynomials computed via "Deep Thought". From the nature of the polynomials q_i it follows that $q_i^{(j)} = x_i$ for $1 \leq i < j$, $q_j^{(j)} = x_j - 1$, and $q_i^{(j)} = x_i + \overline{q_i}^{(j)}(x_1, \cdots, x_{i-1})$ for $j < i \leq n$. To see this, we will follow the exposition of [6]. We may rewrite the product, $a_1^{x_1} \cdots a_n^{x_n} a_1^{y_1} \cdots a_n^{y_n}$, of two elements $x = a_1^{x_1} \cdots a_n^{x_n}$, $y = a_1^{y_1} \cdots a_n^{y_n} \in G$ as

$$(a_1^{x_1} \cdots a_i^{x_i})(a_{i+1}^{x_{i+1}} \cdots a_n^{x_n} a_1^{y_1} \cdots a_i^{y_i})(a_{i+1}^{y_{i+1}} \cdots a_n^{y_n}),$$

which is equal to

$$(a_1^{x_1} \cdots a_i^{x_i} a_1^{y_1} \cdots a_i^{y_i})(a_{i+1}^{'x_{i+1}} \cdots a_n^{'x_n} a_{i+1}^{y_{i+1}} \cdots a_n^{y_n})$$

since G_{i+1} is normal in G. The expression $a_{i+1}^{'x_{i+1}} \cdots a_n^{'x_n} a_{i+1}^{y_{i+1}} \cdots a_n^{y_n}$ can be computed in G_{i+1} and does not involve $a_1 \cdots a_i$. Hence, q_i is determined by $a_1^{x_1} \cdots a_i^{x_i} a_1^{y_1} \cdots a_i^{y_i}$ and q_i only depends on $x_1, \cdots, x_i, y_1 \cdots, y_i$. Since a_i is central in G/G_{i+1} we have that $q_i = x_i + y_i + \overline{q_i}$ with $\overline{q_i} \in \mathbb{Q}[x_1, \cdots, x_{i-1}, y_1, \cdots, y_{i-1}]$. Since multiplying $a_1^{x_1} \cdots a_n^{x_n}$ by a_j^{-1} from the right does not affect $a_1, \cdots, a_{j-1}$ we have that $q_i^{(j)} = x_i$ for $1 \leq i < j$. Moreover, we have $q_j^{(j)} = x_j - 1$ and by entering $y_i = -\delta_{ij}$ we have $q_i^{(j)} = x_i + \overline{q_i}^{(j)}(x_1, \cdots, x_{i-1})$ for $j < i \leq n$ with $\overline{q_i}^{(j)} \in \mathbb{Q}[x_1, \cdots, x_{i-1}]$.

Here we would like to introduce some notation. Since we are required to close the module under powers of each a_j for $j = 1, \cdots, n$, we would like to describe the multiplication of an arbitrary group element with a_j^k for $k \in \mathbb{N}$:

$$a_1^{x_1} \cdots a_n^{x_n} \cdot a_j^{-k} = a_1^{q_{1,k}^{(j)}} \cdots a_n^{q_{n,k}^{(j)}}.$$

The polynomials $q_{i,k}^{(j)}$ are obtained from the multiplication polynomials q_i by setting $y_i = 0$ if $i \neq j$ and $y_i = -k$ if $i = j$, which we denote $-k\delta_{ij}$. Note that by the same reasoning as above we have $q_{i,k}^{(j)} = x_i$ for $1 \leq i < j$, $q_{j,k}^{(j)} = x_j - k$, and $q_{i,k}^{(j)} = x_i + \overline{q_{i,k}}^{(j)}(x_1, \cdots, x_{i-1})$ for $j < i \leq n$ with $\overline{q_{i,k}}^{(j)} \in \mathbb{Q}[x_1, \cdots, x_{i-1}]$.

In order to understand the action of a_j^k, for $k \in \mathbb{N}$, on each t_i we look at three cases: $1 \leq i < j$, $i = j$, and $j < i \leq n$.

(1) $1 \leq i < j$:

$$\begin{aligned}
t_i^{a_j^k} &:= t_i(q_{1,k}^{(j)}, \cdots, q_{n,k}^{(j)}) \\
&= t_i(x_1, \cdots, x_j - k, x_{j+1} + \overline{q_{j+1,k}}^{(j)}(x_1, \cdots, x_j), \cdots, x_n + \overline{q_{n,k}}^{(j)}(x_1, \cdots, x_{n-1})) \\
&= x_i \\
&= t_i
\end{aligned}$$

(2) $i = j$:

$$\begin{aligned} t_j^{a_j^k} &:= t_j(q_{1,k}^{(j)}, \cdots, q_{n,k}^{(j)}) \\ &= t_i(x_1, \cdots, x_j - k, x_{j+1} + \overline{q_{j+1,k}}^{(j)}(x_1, \cdots, x_j), \cdots, x_n + \overline{q_{n,k}}^{(j)}(x_1, \cdots, x_{n-1})) \\ &= x_j - k \\ &= t_j - k \end{aligned}$$

(3) $j < i \leq n$:

$$\begin{aligned} t_i^{a_j^k} &:= t_i(q_{1,k}^{(j)}, \cdots, q_{n,k}^{(j)}) \\ &= t_i(x_1, \cdots, x_j - k, x_{j+1} + \overline{q_{j+1,k}}^{(j)}(x_1, \cdots, x_j), \cdots, x_n + \overline{q_{n,k}}^{(j)}(x_1, \cdots, x_{n-1})) \\ &= x_i + \overline{q_{i,k}}^{(j)}(x_1, \cdots, x_{i-1}) \\ &= t_i + \overline{q_{i,k}}^{(j)}(x_1, \cdots, x_{i-1}) \end{aligned}$$

By utilizing this information on the action of a_j^k on each t_i we are able to determine the complexity of the algorithm presented in Figure 1 when the coordinate functions are used. We begin by finding an upper bound on the size of the matrices produced. It is clear from the algorithm presented in [6] that the size of the matrix representation produced depends on the size of the $\mathbb{Q}$-basis constructed using the algorithm presented in Figure 1. To determine a bound, we begin by analyzing the main loop of the algorithm in Figure 1. The number of iterations the main loop in Figure 1 undergoes throughout the algorithm is directly related to the size of the basis produced.

Theorem 5. *When the algorithm presented in* [6] *is implemented using the coordinate functions t_i for $1 \leq i \leq n$ the worst case dimension of the matrix representation produced depends quadratically on the Hirsch length of the group; that is, the dimension of the matrix representation is $O(n^2)$ where n is the Hirsch length of the group..*

Proof. We will prove the cardinality of the basis depends quadratically on the Hirsch length of the group by using induction on the number of times the main loop in Figure 1 is repeated.

Recall that $t_i^{a_j^k} := t_i(q_{1,k}^{(j)}, \cdots q_{n,k}^{(j)})$ where $q_{i,k}^{(j)} = x_i$ for $1 \leq i < j$, $q_{j,k}^{(j)} = x_j - k$, and $q_{i,k}^{(j)} = x_i + \overline{q_{i,k}}^{(j)}(x_1, \cdots, x_{i-1})$ for $j < i \leq n$.

First note that we may write $\overline{q_{i,k}}^{(j)} = r_{i,k}^{(j)} + \sum_{m=1}^{i-1} c_m t_m$ for appropriate $c_m \in \mathbb{Q}$, $r_{i,k}^{(j)} \in \mathbb{Q}[x_1, \cdots, x_{i-1}]$, where $r_{i,k}^{(j)} = 0$ or cannot be written as a linear combination of the coordinate functions t_l for $1 \leq l \leq n$.

The algorithm presented in [6] (see Figure 1) begins by building a basis from the coordinate functions $t_1, \cdots, t_n$ using *Insert*. By definition $t_i(x_1, \cdots, x_n) = x_i$; hence, *Insert* will form the basis $B = \{t_1, \cdots, t_n\}$.

Since we are considering the worst case scenario, we will assume that each polynomial $r_{i,k}^{(j)}$ is not a linear combination of $t_1, \cdots, t_n, r_{m,k}^{(j)}$ for $m \neq i$.

Claim: The basis produced in the worst case scenario is $B = \left\{t_1, \cdots, t_n, r_{n,k_{n,n-1}}^{(n-1)}, \cdots, r_{n,k_{n,1}}^{(1)}, r_{n-1,k_{n-1,n-2}}^{(n-2)}, \cdots, r_{n-1,k_{n-1,1}}^{(1)}, \cdots, r_{3,k_{3,2}}^{(2)}, r_{3,k_{3,1}}^{(1)}, r_{2,k_{2,1}}^{(1)}, r_1\right\}$ for $k_{i,j} = 1, \cdots, m_{i,j}$ where $m_{i,j}$ denotes the number of terms in the polynomial $\overline{q_i}^{(j)}$.

We must first consider when the main loop in Figure 1 is iterated one time.

The loop begins by adding images of each t_i under the action of a_n. We may assume that the algorithm begins by adding images of t_n under the action of a_n to the basis. We know that $t_n^{a_n} = t_n - 1$, which clearly is not in the span of the basis. *Insert*$(B,\ t_n^{a_n})$ will begin by subtracting t_n from $t_n - 1$, leaving $r = -1$ which will be added to the basis. As it is necessary to close the G-module under the action of powers of a_n, we must repeat the process for $t_n^{a_n}$. Hence, the algorithm computes $(t_n^{a_n})^{a_n} = t_n^{a_n^2} = t_n - 2$, which is the span of the basis; that is $r = 0$ and the process terminates. For $1 \leq i \leq n-1$, we have that $t_i^{a_n} = t_i$. Hence, for t_i with $1 \leq i \leq n-1$ no new polynomials will be added to the basis. After this process is completed for a_n the resulting basis will be $B = \{t_1, \cdots, t_n, r_1 = -1\}$.

Base case: The main loop in Figure 1 is repeated twice.
The second repetition of the loop repeats the process for a_{n-1}.
We begin by noting that for $1 \leq i \leq n-2$ we have $t_i^{a_{n-1}} = t_i$ and $t_{n-1}^{a_{n-1}} = x_{n-1} - 1 = t_{n-1} + r_1$, which are in the span of the basis. Hence, we need only to consider the case in which images of t_n under the action of a_{n-1}^k for $k \in \mathbb{N}$ are added to the basis. By the nature of the coordinate functions and since the polynomials $\overline{q_{n,k}}^{(n-1)}$ and $\overline{q_{n,j}}^{(n-1)}$ differ only by coefficients to close under the action of powers of a_{n-1} the inner most loop will be repeated at most $m_{n,n-1}$ times, where $m_{n,n-1}$ denotes the number of terms in the polynomial $\overline{q_n}^{(n-1)}$; that is, we need to add images of t_n under the action of a_{n-1}^k for $k = 1, \cdots, m_{n,n-1}$.

From above we know that

$$\begin{aligned} t_n^{a_{n-1}^k} =& t_n(x_1, \cdots, x_{n-1} - k, x_n + \overline{q_{n,k}}^{(n-1)}(x_1, \cdots, x_{n-1})) \\ =& x_n + \overline{q_{n,k}}^{(n-1)}(x_1, \cdots, x_{n-1}) \\ =& t_n + \overline{q_{n,k}}^{(n-1)} \\ =& t_n + r_{n,k}^{(n-1)} + \sum_{m=1}^{n-1} c_m t_m. \end{aligned}$$

Insert$(B, t_n^{a_{n-1}})$ will subtract appropriate multiples of $t_1, \cdots, t_n$ from $t_n^{a_{n-1}^k}$ leaving $r = r_{n,k}^{(n-1)}$ to be added to the basis for $k = 1, \cdots, m_{n,n-1}$. Hence, after two iterations of the loop the resulting basis is

$B = \left\{t_1, \cdots, t_n, r^{(n-1)}_{n,k_{n,n-1}}, r_1\right\}$ for $k_{n,n-1} = 1, \cdots, m_{n,n-1}$.

Inductive assumption: Note that the j^{th} iteration of the loop will add images of $t_1, \cdots, t_n$ under the action of powers of a_{n-j+1}. Suppose for $2 < j < n$ iterations of the loop the resulting basis is
$B = \left\{t_1, \cdots, t_n, r^{(n-1)}_{n,k_{n,n-1}}, \cdots, r^{(n-j+1)}_{n,k_{n,n-j+1}}, r^{(n-2)}_{n-1,k_{n-1,n-2}}, \cdots, r^{(n-j+1)}_{n-1,k_{n-1,n-j+1}}, \cdots, r^{(n-j+1)}_{n-j+2,k_{n-j+2,n-j+1}}, r_1,\right\}$
for $k_{h,i} = 1, \cdots, m_{h,i}$ where each $m_{h,i}$ represents the number of terms in the polynomial $\overline{q_h}^{(i)}$.

$(j+1)^{st}$ *iteration of the loop*: The main loop in Figure 1 will be performed for a_{n-j}.
We begin by noting that

$$\begin{aligned} t_i^{a^k_{n-j}} &:= t_i(q^{(n-j)}_{1,k}, \cdots, q^{(n-j)}_{n,k}) \\ &= t_i(x_1, \cdots, x_{n-j} - k, x_{n-j+1} + r^{(n-j)}_{n-j+1,k} + \sum_{m=1}^{n-j} c_m t_m, \cdots, x_n + r^{(n-j)}_{n,k} + \sum_{m=1}^{n-1} d_m t_m) \end{aligned}$$

We would like to see which polynomials will be added to the basis in this step. We will look at three cases:

- $1 \leq i \leq n-j-1$:
  $t_i^{a^k_{n-j}} = x_i = t_i$. This is in the span of the basis, and so the algorithm will have $r = 0$ and the process terminates.
- $i = n-k$:
  $t_{n-j}^{a^k_{n-j}} = x_{n-j} - k = t_{n-j} + kr_1$. This is in the span of the basis, and so the algorithm will have $r = 0$ and the loop terminates.
- $n-j+1 \leq i \leq n$:
  $t_i^{a^k_{n-j}} = x_i + r^{(n-j)}_{i,k} + \sum_{m=1}^{i-1} c_m t_m$. By assumption $r^{(n-j)}_{i,k}$ is not in the span of the basis for each $n-j+1 \leq i \leq n$ and $k \in \mathbb{N}$. The algorithm will then add $r^{(n-j)}_{i,k_{i,n-j}}$ for $k_{i,n-j} = 1, \cdots, m_{i,n-j}$ to the basis for each i, resulting in the basis
  $B = \left\{t_1, \cdots, t_n, r^{(n-1)}_{n,k_{n,n-1}}, \cdots, r^{(n-j)}_{n,k_{n,n-j}}, r^{(n-2)}_{n-1,k_{n-1,n-2}}, \cdots, r^{(n-j)}_{n-1,k_{n-1,n-j}}, \cdots, r^{(n-j)}_{n-j+2,k_{n-j+2,n-j}}, r^{(n-j)}_{n-j+1,k_{n-j+1,n-j}}, r_1\right\}$
  for $k_{h,i} = 1, \cdots, m_{h,i}$.

Hence, for any $1 \leq j < n$ we know after $j+1$ iterations the resulting basis will be
$B = \left\{t_1, \cdots, t_n, r^{(n-1)}_{n,k_{n,n-1}}, \cdots, r^{(n-j)}_{n,k_{n,n-j}}, r^{(n-2)}_{n-1,k_{n-1,n-2}}, \cdots, r^{(n-j)}_{n-1,k_{n-1,n-j}}, \cdots, r^{(n-j)}_{n-j+2,k_{n-j+2,n-j}}, r^{(n-j)}_{n-j+1,k_{n-j+1,n-j}}, r_1\right\}$
for $k_{h,i} = 1, \cdots, m_{h,i}$ where each $m_{h,i}$ represents the number of terms in the polynomial $\overline{q_h}^{(i)}$. The main loop in Figure 1 will be repeated n times (one for each element in the nilpotent generating sequence), and the resulting basis will be
$B = \left\{t_1, \cdots, t_n, r^{(n-1)}_{n,k_{n,1}}, \cdots, r^{(1)}_{n,k_{n,1}}, r^{(n-2)}_{n-1,k_{n-1,n-2}}, \cdots, r^{(1)}_{n-1,k_{n-1,1}}, \cdots, r^{(2)}_{3,k_{3,2}}, r^{(1)}_{3,k_{3,1}}, r^{(1)}_{2,k_{2,1}}, r_1\right\}$
for $k_{i,j} = 1, \cdots, m_{i,j}$ as desired.

The cardinality of this basis is at most
$n + \sum_{i=1}^{n-1} m_{n,i} + \sum_{i=1}^{n-2} m_{n-1,i} + \cdots + \sum_{i=1}^{2} m_{3,i} + m_{2,1} + 1.$

Let m denote the largest of the $m_{i,j}$. Then the cardinality of the basis is bounded above by

$$\begin{aligned} n + (n-1)m + (n-2)m + \cdots + 2m + m + 1 \leq & m \sum_{i=1}^{n} i + 1 \\ \leq & \frac{m}{2} n(n+1) + 1. \end{aligned}$$

Thus, the dimension of the matrix representation is $O(n^2)$.

□

Theorem 6. *The running time of the algorithm presented in Figure 1 using the coordinate functions t_i for $1 \leq i \leq n$ is $O(n^{l+2})$, where n is the Hirsch length of the group and l is the highest degree of the polynomials $t_i^{a_j}$.*

Proof. Before determining the worst case complexity of the algorithm presented in Figure 1, we must determine the number of steps needed by the method *Insert*. *Insert* takes a basis of polynomials in ascending order and a polynomial f, and determines if f is in the span of the basis by subtracting the appropriate multiple of each polynomial in the basis, starting with the polynomial that is largest with respect to the ordering and continuing in descending order. If $f \neq 0$ after this process is completed, then f is not in the span of the basis and is added to the original basis. It is clear that the number of subtractions used on a given polynomial f by *Insert* is at most the number of terms in f. Each polynomial $t_i^{a_j}$ is a polynomial in n variables since $t_i^{a_j} = t_i(q_1^{(j)}, \cdots, q_n^{(j)})$. An l^{th} degree polynomial with n variables has at most $\sum_{k=1}^{l} \binom{k+n-1}{k}$ terms, which is bounded above by $\frac{c}{l!}(n+l-1)^l$ for an appropriate $c \in \mathbb{N}$. We will denote the number of terms in a given polynomial f by m_f.

Recall the algorithm to form a $\mathbb{Q}$-basis of the G-module generated by the coordinate functions t_i for $1 \leq i \leq n$ begins by building a basis from the coordinate functions $t_1, \cdots, t_n$ by implementing $Insert(B, t_i)$ for $1 \leq i \leq n$. It is clear from the above remarks that this process will take a total of $\sum_{i=1}^{n} m_{t_i}$ steps.

The next step to determine the complexity of the algorithm for building a $\mathbb{Q}$-basis is to determine how many iterations of the innermost loop in Figure 1 occur for each a_j. In the proof of Theorem 5 one can see how many iterations of Insert need to be performed for each a_j, where $1 \leq j \leq n$. It is clear from the proof of Theorem 5 that when the first iteration of the loop

is performed (closing under the action of a_n) *Insert* is repeated $n+1$ times. For the $k+1^{st}$ iteration the algorithm will close the module under the action of a_{n-k}. For t_i, with $1 \leq i \leq n-k$, $t_i^{a_{n-k}}$ is already in the basis and hence $n-k$ iterations of *Insert* are performed. For t_i with $n-k+1 \leq i \leq n$, $t_i^{a_{n-k}}$ is not in the span of the basis and *Insert* is performed $m_{i,n-k}$ times, where $m_{i,n-k}$ denotes the number of terms in the polynomial $\overline{q_i}^{(n-k)}$, for each i (see proof of Theorem 5). Hence, here *Insert* will be performed $\sum_{i=n-k+1}^{n} m_{i,n-k}$ times. Thus, the total number of times Insert will be performed in the $k+1^{st}$ iteration of the loop is $(n-k)+\sum_{i=n-k+1}^{n} m_{i,n-k}$. As above let m denote the maximum of the $m_{i,j}$.

The total number of iterations of Insert performed in the main loop of Figure 1 is then

$$\begin{aligned}(n+1)+\sum_{k=1}^{n-1}(n-k)+\sum_{k=1}^{n-1}\sum_{i=n-k+1}^{n} m_{i,n-k} &= 1+\sum_{k=1}^{n} k+\sum_{k=1}^{n-1}\sum_{i=n-k+1}^{n} m_{i,n-k}\\ &=1+\frac{1}{2}n(n+1)+\sum_{k=1}^{n-1}\sum_{i=n-k+1}^{n} m_{i,n-k}\\ &\leq 1+\frac{1}{2}n(n+1)+\sum_{k=1}^{n-1} km\\ &=1+\frac{1}{2}n(n+1)+\frac{m}{2}n(n-1)\\ &=1+\frac{m+1}{2}n^2+\frac{1-m}{2}n\end{aligned}$$

Let l be the highest degree of the polynomials utilized in the algorithm. Then we know the maximum number of steps that *Insert* requires at any step is bounded above by $\frac{c}{l!}(n+l-1)^l$. Then the number of steps required by the main loop in Figure 1 is bounded above by

$$\frac{c}{l!}(n+l-1)^l(1+\frac{m+1}{2}n^2+\frac{1-m}{2}n)$$

and the total number of steps required by the algorithm is bounded above by $\sum_{i=1}^{n} m_{t_i}+\frac{c}{l!}(n+l-1)^l(1+\frac{m+1}{2}n^2+\frac{1-m}{2}n) \leq \frac{c}{l!}(n+l-1)^l(1+\frac{m+1}{2}n^2+\frac{3-m}{2}n)$. Thus, the total running time of the algorithm is $O(n^{l+2})$.

□

4. Modified Algorithm

From Theorem 5, we see that the $\mathbb{Q}$-basis of the G-module generated by the coordinate functions $t_1, \cdots, t_n$ is a subset of $B = \left\{t_1, \cdots, t_n, r^{(n-1)}_{n,k_{n,n-1}}, \cdots, r^{(1)}_{n,k_{n,1}}, r^{(n-2)}_{n-1,k_{n-1,n-2}}, \cdots, r^{(1)}_{n-1,k_{n-1,1}}, \cdots, r^{(2)}_{3,k_{3,2}}, r^{(1)}_{3,k_{3,1}}, r^{(1)}_{2,k_{2,1}}, r_1\right\}$ for $k_{i,j} = 1, \cdots, m_{i,j}$ where $m_{i,j}$ denotes the number of terms in the polynomial $\overline{q_i}^{(j)}$. Using this fact we may modify the algorithm presented by Nickel in [6].

In order to find a $\mathbb{Q}$-basis of the G-module generated by the coordinate functions $t_1, \cdots, t_n$ we must begin by building a basis up from the coordinate functions using *Insert* as is done in [6]. By the nature of the coordinate functions, it is clear that the action of each a_j for $1 \leq j \leq n$ will only add new polynomials to the basis at certain steps in the algorithm. Recall that in the first time the main loop in Figure 1 is repeated the only polynomial that could be added to the basis is when $Insert(B, t_n^{a_n})$ is applied. Then in the $k+1^{st}$, where $1 \leq k \leq n-1$, iteration of the loop the only polynomials that could be added to the basis arise from closing under powers of a_{n-k} for $n-k+1 \leq i \leq n$; that is, we need only perform $Insert(B, t_n^{a_n})$ and $Insert(B, t_i^{a_j^p})$ for $i > j$, $1 \leq j \leq n-1$ and for $p \in \mathbb{N}$. This is clear in the proof of Theorem 5. Hence, in order to determine the $\mathbb{Q}$-basis of the G-module generated by $t_1, \cdots, t_n$ it is enough to implement $Insert$ for only these polynomials. The pseudo-code for the algorithm is displayed in Figure 2.

Theorem 7. *The algorithm presented in Figure 2 has running time $O(n^{l+2})$, where n is the Hirsch length of the group and l is the highest degree of the polynomials $t_i^{a_j}$.*

Proof. We denote the number of terms in a polynomial f by m_f. Then the number of steps required for $Insert(B, t_i)$ for $1 \leq i \leq n$ is at most m_{t_i} and the number of steps of $Insert(B, t_n^{a_n})$ is at most m_{t_n}.

Next, we must determine the number of times $Insert$ is performed for the $k+1^{st}$ iteration of the innermost **for** loop in Figure 3; that is, we must determine the number of times $Insert$ is performed to close the module under the action of a_{n-k}. As the only difference between the modified algorithm and the algorithm presented in [6] when implemented with the coordinate functions is the **while** loop, the number of times $Insert$ is performed for each a_{n-k} only differs by $n-k$ (see proof of 6). Thus, the total number of times Insert will be performed in the $k+1^{st}$ iteration of the loop is $\sum_{i=n-k+1}^{n} m_{i,n-k}$. Let m denote the maximum of the $m_{i,j}$.

FIGURE 2. Building a Basis for a G-module using Coordinate Functions

Input: A finitely generated torsion-free nilpotent group G with nilpotent generating sequence $a_1, \cdots, a_n$ and multiplication polynomials $q_1, \cdots, q_n$; coordinate functions $t_1, \cdots, t_n$.
Output: A $\mathbb{Q}$-basis B for the G-module generated by $t_1, \cdots, t_n$.
Algorithm:

$B := []$;
  for j **in** $[1, \cdots n]$ **do** $Insert(B, t_j)$; **od**;
  do $Insert(B, t_n^{a_n})$; **od**;
  for j **in** $[n-1 \cdots 1]$ **do**
    for i **in** $[1, \cdots n]$ **do**
      while $i > j$ **do**
        repeat
          # Compute $t_i^{a_j}$
          $t_i^{a_j} := t_i(q_1^{(j)}, \cdots, q_n^{(j)})$;
          r:=Insert$(B, t_i^{a_j})$;
          $t_i := t^{a_j}$;
      until r=0;
      od;
    od;
  od;
return B;

The total number of iterations of Insert performed is then

$$\begin{aligned}\sum_{k=1}^{n-1}\sum_{i=n-k+1}^{n} m_{i,n-k} \leq \sum_{k=1}^{n-1} km \\ = \frac{m}{2}n(n-1) \\ = \frac{m}{2}n^2 - \frac{m}{2}n\end{aligned}$$

Let l be the highest degree of the polynomials utilized in the algorithm. Then we know the maximum number of steps that *Insert* requires at any step is bounded above by $\frac{c}{l!}(n+l-1)^l$ for some $c \in \mathbb{N}$. Then the number of steps required by the main loop in Figure 2 is bounded above by

$$\frac{c}{l!}(n+l-1)^l(\frac{m}{2}n^2 - \frac{m}{2}n)$$

and the total number of steps required by the algorithm is bounded above by

$$\sum_{i=1}^{n} m_{t_i} + m_{t_n} + \frac{c}{l!}(n+l-1)^l(\frac{m}{2}n^2 - \frac{m}{2}n) \leq$$
$$\frac{c}{l!}(n+l-1)^l(1 + \frac{m}{2}n^2 + \frac{2-m}{2}n).$$

Thus, the total running time of the algorithm is $O(n^{l+2})$. □

5. Closing Remarks

We have shown that the algorithm in [6] has polynomial time complexity with respect to the Hirsch length if the coordinate functions are utilized, and that the dimension of the matrix representation produced depends quadratically on the Hirsch length of the group. It is possible to improve the bound on the dimension and time complexity by using a different set of functions rather than the coordinate functions. Unfortunately, there is no known way to choose these functions.

The modified algorithm is only for generating a $\mathbb{Q}$-basis for the G-module generated by the coordinate functions unlike the algorithm presented in [6], which works for a general set of functions $f_1, \cdots, f_k$. We would like to note that the running time of the modified algorithm can have better running times than the algorithm presented in [6] since the running time is dependent on the polynomials $t_i^{a_j}$. While this algorithm potentially decreases running time, it does not affect the dimension of the matrices produced.

Acknowledgement

The authors would like to thank the anonymous referee for his helpful comments. Delaram Kahrobaei is grateful to Professor Derek Holt for providing the opportunity for her to visit Warwick University in summer 2010 and very stimulating conversations regarding this problem.

References

[1] W. A. De Graaf and W. Nickel. Constructing faithful representations of finitely-generated torsion-free nilpotent groups. *J. Symbolic Computation*, 33:31–41, 2002.

[2] Philip Hall. Nilpotent groups. *Notes of lectures given at the Canadian Mathematical Congress, University of Alberta*, pages 1–42, 1957.

[3] D. Holt, B. Eick, and E. O'Brien. *Handbook of computational group theory*. Discrete Mathematics and its Applications (Boca Raton), Chapman and Hall/CRC, Boca Raton, FL, 2005.

[4] A.I. Kostrikin and I.R. Shafarevich. *Encyclopedia of Mathematical Sciences, Volume 37, Algebra IV: Infinite Groups, Linear Groups*. Springer-Verlag, 1991.

[5] C.R. Leedham-Green and L.H. Soicher. Symbolic collection using deep thought. *LMS J. Comput. Math.*, 1:9–24, 1998.

[6] W. Nickel. Matrix representations for torsion-free nilpotent groups by deep thought. *Journal of Algebra*, 300:376–383, 2006.

California University of Pennsylvania
E-mail address: Habeeb@CalU.edu

CUNY Graduate Center and City Tech, City University of New York
E-mail address: DKahrobaei@GC.Cuny.edu