J. Symbolic Computation (2002) **34**, 213–228
doi:10.1006/jsco.2002.0559
Available online at http://www.idealibrary.com on IDEAL®

Efficient Collection in Infinite Polycyclic Groups

VOLKER GEBHARDT†

School of Mathematics and Statistics, University of Sydney, Sydney, NSW 2006, Australia

We present and analyse an algorithm for collection in polycyclic groups which has better complexity in terms of the exponents occurring in the collected words than previously used collection algorithms, and yields a much better performance especially for infinite polycyclic groups.

1. Introduction

Each polycyclic group G can be described by a *polycyclic presentation* of the form

$$\begin{aligned}\langle x_1,\dots,x_n \mid x_i^{m_i} &= w_{i,i} \quad (i\in I),\\ x_j^{x_i} &= w_{i,j} \quad (1\le i<j\le n),\\ x_j^{x_i^{-1}} &= w_{-i,j} \quad (1\le i<j\le n;\ i\notin I)\\ (x_j^{-1})^{x_i} &= w_{i,-j} \quad (1\le i<j\le n; j\notin I),\\ (x_j^{-1})^{x_i^{-1}} &= w_{-i,-j} \quad (1\le i<j\le n;\ i,j\notin I)\rangle,\end{aligned} \tag{1}$$

where $I\subseteq\{1,\dots,n\}$, $m_i>1$ for $i\in I$, and the words $w_{i,j}$ are of the form $w_{i,j}=x_{|i|+1}^{l(i,j,|i|+1)}\dots x_n^{l(i,j,n)}$, with $l(i,j,k)\in\mathbb{Z}$, $0\le l(i,j,k)<m_k$ if $k\in I$.

For $1\le i\le n$, let G_i be the subgroup of G generated by $x_i,\dots,x_n$ and define G_{n+1} to be the trivial group. The presentation is called *consistent*, if $|G_i/G_{i+1}|=m_i$ whenever $i\in I$ and G_i/G_{i+1} is infinite whenever $i\notin I$.

Given a consistent polycyclic presentation as above, every element u of G can be uniquely written in *normal form* as $u=x_1^{\alpha_1}\cdots x_n^{\alpha_n}$, where $\alpha_i\in\mathbb{Z}$, $0\le\alpha_i<m_i$ if $i\in I$. We call $(\alpha_1,\dots,\alpha_n)$ the *vector of exponents* for u. Given two words u and v in normal form, the product $u\cdot v$ can be reduced to normal form by repeatedly substituting minimal non-normal subwords using the relations from (1); a process called *collecting*. While several strategies for selecting minimal non-normal subwords for substitution have been developed, the best performance in most situations is obtained by choosing the leftmost minimal non-normal subword (Vaughan-Lee, 1990; Leedham-Green and Soicher, 1990). This strategy, called *Collection from the Left*, is now well-established for computing with polycyclic presentations, and is used in computer algebra programs like MAGMA (Bosma *et al.*, 1997) and GAP (GAP, 2000).

Until recently, polycyclic presentations have mostly been used for representing finite soluble groups and in particular p-groups. In these situations we have $I=\{1,\dots,n\}$ and

†E-mail: volker@maths.usyd.edu.au

0747–7171/02/030213 + 16 $35.00/0

hence all entries of the exponent vectors for elements are bounded, usually by relatively small primes. Consequently, the influence of the order of magnitude of the exponents on the cost of collection was only of little importance and the attention was focused on the influence of the number of generators or the θ-class of the presentations (Leedham-Green and Soicher, 1990). However, the past few years have seen a growing interest in infinite polycyclic groups and a number of important algorithms have recently been developed for this class of groups (Lo, 1998; Eick, 2001). In infinite polycyclic groups, there is no bound on the entries of the exponent vectors for elements and hence the need arises to work efficiently with large exponents in the collection process.

The previously published versions of collection from the left perform rather badly in this respect. We will establish in Theorem 3.4 that the cost of collecting a product ux_i^β, where u is in normal form, is asymptotically bounded above by

$$(n-1)! \cdot K^{2\cdot(n-1)} + o\,(K^{2\cdot(n-1)}),$$

where K is a bound for the absolute values of all exponents occurring during the collection. Concerning lower bounds, we will see in Section 3 that for non-commuting generators x_i and x_j, the cost of collecting the product $x_j^\alpha x_i^\beta$ $(1 \le i < j \le n; \alpha, \beta > 0)$ into normal form grows at least linearly in α and β, since the conjugate relation $x_j^{x_i} = w_{i,j}$ is used β times and each time the resulting expression $(w_{i,j})^\alpha$ is processed at a cost proportional to α.

In this paper we present a modification of collection from the left which avoids this problem by employing "repeated squaring" in a twofold way. Firstly, we process expressions of the form $(w_{i,j})^\alpha$ by means of repeated squaring, using recursive collection. Secondly, we obtain normal form expressions for conjugates of the form $x_j^{(x_i{}^\beta)}$ by computing the βth power of the automorphism of G_{i+1} induced by conjugation with x_i using repeated squaring techniques.[†] We prove in Theorem 4.13 that, using these improvements, the cost of collecting a product ux_i^β, where u is in normal form, is asymptotically bounded above by

$$6^{n-1} \cdot ((n-1)!)^2 \cdot (\log_2 K)^{2\cdot(n-1)} + o\,((\log_2 K)^{2\cdot(n-1)}),$$

where K is a bound for the absolute values of all exponents occurring during the collection. Furthermore, we determine theoretical crossover points between the two algorithms and present detailed runtime statistics.

The new collection algorithm has been implemented in C as part of the Magma kernel and is used in Magma version 2.8; a hybrid version, avoiding performance penalties for small exponents, is used from version 2.9 onwards.

2. Collection from the Left

In this section we recall collection from the left as described in Vaughan-Lee (1990). However, we present it as specialization of a slightly more general algorithm.

Notation 2.1. (Collector from the Left) A *collector from the left* represents a word in the generators of G in two parts, a collected part $x = x_1^{\alpha_1} \cdots x_n^{\alpha_n}$ in normal form, which is stored as vector $a = (\alpha_1, \ldots, \alpha_n)$ of exponents, and an uncollected part,

[†]Similar constructions seem to have been used in implementations before. However, neither a detailed description, a complexity analysis nor runtime statistics have been published.

which is stored as a stack of generator powers $s = (x_{i_1}^{\beta_1}, \ldots, x_{i_t}^{\beta_t})$, where t indicates the top of the stack, and $\beta_j \in \mathbb{Z} \setminus \{0\}$, $0 < \beta_j < m_{i_j}$ if $i_j \in I$, for all j. This represents the word $x \cdot x_{i_t}^{\beta_t} \cdots x_{i_1}^{\beta_1}$.

Moreover, a collector has an algorithm $\mathcal{C}$, the *collection algorithm*, which modifies the represented word without changing the represented element of G as follows. It reduces the absolute value $|\beta_t|$ of the exponent of the topmost item $x_{i_t}^{\beta_t}$ on the stack, removes items with exponent 0, modifies x and possibly pushes a finite number of generator powers $x_{k_1}^{\gamma_1}, \ldots, x_{k_r}^{\gamma_r}$, onto the stack, where for all j we have $k_j > i_t$ and $\gamma_j \in \mathbb{Z} \setminus \{0\}$, $0 < \gamma_j < m_{k_j}$ if $k_j \in I$.

To collect a product $u \cdot v$ into normal form, where $u = x_1^{\gamma_1} \cdots x_n^{\gamma_n}$ and $v = x_1^{\delta_1} \cdots x_n^{\delta_n}$ are in normal form, the exponent vector representing the collected part is initialized with $(\gamma_1, \ldots, \gamma_n)$ and the collector stack is initialized with $(x_n^{\delta_n}, \ldots, x_1^{\delta_1})$. Then, the collection algorithm $\mathcal{C}$ is applied repeatedly. Since during the collection of a power of x_i from the stack, only powers of generators x_k for $k > i$ are pushed onto the collector stack, this process must terminate, that is, the stack becomes empty at some point and the collected part then is equal to the normal form of $u \cdot v$.

2.1. A general version of collection from the left

Let the topmost generator power on the collector stack be x_i^{β}. To work it off the stack, x_i^{β} must be moved in place past the postfix $x_{i+1}^{\alpha_{i+1}} \cdots x_n^{\alpha_n}$ of the collected part. In order to simplify notation, we set $y := x_i^{\operatorname{sgn}(\beta)}$, $y_k := x_k^{\operatorname{sgn}(\alpha_k)}$ $(k = 1, \ldots, n)$ and redefine $\beta := |\beta|$ and $\alpha_k := |\alpha_k|$ $(k = 1, \ldots, n)$.

Algorithm 2.2. (Elementary Collection Step) *The following algorithm is a collection algorithm in the sense of Notation* 2.1.

1. *Choose* $\sigma \in \{1, \ldots, \beta\}$.
2. *Set* $\beta := \beta - \sigma$ *and* $\alpha_i := \alpha_i \pm \sigma$*; if* $\beta = 0$*, remove* y^{β} *from the stack.*
3. *If* $i \in I$ *and* $\alpha_i \geq m_i$*, set* $\alpha_i := \alpha_i - m_i$ *and push the right-hand side* $w_{i,i}$ *of the power relation for* x_i *onto the collector stack.*
4. *For* $k := n, \ldots, i+1$*, push any word for* $(y_k^{\alpha_k})^{y^{\sigma}}$ *involving only the generators* $x_{i+1}, \ldots, x_n$ *onto the collector stack and set* $\alpha_k := 0$.

2.2. Collection from the left as described in Vaughan-Lee (1990)

Normal form expressions for conjugates of the form $(y_k^{\alpha_k})^{y^{\sigma}}$ can only be read directly from the polycyclic presentation if $\alpha_k = 1 = \sigma$ or if y and y_k commute. For this reason, the version of collection from the left described in Vaughan-Lee (1990) basically corresponds to the following specialization of Algorithm 2.2, which we will refer to as Algorithm 2.3. For details see Vaughan-Lee (1990).

1. In step 1 of Algorithm 2.2, $\sigma := 1$ is chosen, unless y commutes with y_k for all $k = i+1, \ldots, n$, in which case $\sigma := \beta$ is chosen. Hence, the item y^{β} on the collector stack in general is treated as a sequence of β copies of y.

2. For non-commuting generators y and y_k (implying $\sigma = 1$), α_k copies of the normal form expression for ${y_k}^y$, which can be read from the polycyclic presentation, are pushed onto the collector stack in step 4 of Algorithm 2.2.

3. Analysing Collection from the Left

In this section we investigate the cost of working the topmost item y^β off the collector stack using Algorithm 2.3 as collection algorithm. As a measure for the cost, we take the total number of stack items which have to be worked off the stack until the collection of y^β is complete, that is, we include the cost for working off the stack all generator powers, which may be pushed onto the stack during this process. This is formalized in Definition 3.1.

Definition 3.1. (Cost of Collecting a Generator Power) Assume a collector state as in Notation 2.1 with collected part given by a and a collector stack s containing t items with topmost item x_i^β. Define $P(a, s, \mathcal{C})$ to be the number of passes through the collection algorithm $\mathcal{C}$ which have to be performed until the number of items on the collector stack reaches $t - 1$. We call $P(a, s, \mathcal{C})$ the *cost of collecting* x_i^β.

Definition 3.2. Let A and B be fixed positive integers. For $i \in \{1, \ldots, n\}$ define $N_1(i)$ recursively by

$$N_1(n) := 1 \qquad \text{and} \qquad N_1(i) := 1 + (1 + BA \cdot (n - i)) \cdot \sum_{r=i+1}^{n} N_1(r) \qquad \text{for } i < n. \quad (2)$$

Proposition 3.3. *Assuming $A = B$ and considering $N_1(i)$ as a function of A only, that is, regarding n and i as fixed, we have*

$$N_1(i) = (n - i)! \cdot A^{2\cdot(n-i)} + o(A^{2\cdot(n-i)}).$$

Proof. This follows from the definition by induction, writing

$$N_1(i) = \sum_{k=0}^{n-i} n_k^{(i)} \cdot A^{2k}$$

and comparing leading coefficients. □

Theorem 3.4. *Assume a collector state as in Notation* 2.1 *and let the collection algorithm $\mathcal{C}$ be given by Algorithm* 2.3. *Let the topmost item on the collector stack be y^β, where $y = x_i^{\pm 1}$ and $\beta > 0$. Assume that during the entire collection of y^β, the absolute values of the exponents $\alpha_1, \ldots, \alpha_n$ and the absolute values of the exponents of the generator powers on the collector stack remain bounded by the constants A and B, respectively. Then,*

$$P(a, s, \mathcal{C}) \leq N_1(i).$$

Proof. If $i = n$, $P(a, s, \mathcal{C}) = 1 = N_1(n)$, so assume $i < n$ and suppose the theorem holds for states of the collector for which the topmost item on the collector stack is a power of $x_{i'}$, where $i' > i$. According to Algorithm 2.3, β occurrences of y have to be moved in place one after another. Note that during these β collections, at most one reduction of

the exponent α_i in the collected part can occur. Since every word pushed onto the stack during collecting y^β only involves the generators $x_{i+1}, \ldots, x_n$, the theorem can be used to estimate the cost of collecting these generator powers and every word in normal form which is pushed onto the stack can be collected at a cost of at most $\sum_{r=i+1}^{n} N_1(r)$. Using the bound A on the absolute values of the exponents α_k in the collected part and $\beta \leq B$, we obtain

$$P(a,s,\mathcal{C}) \leq \underbrace{1}_{\text{(a)}} + \underbrace{\sum_{r=i+1}^{n} N_1(r)}_{\text{(b)}} + \underbrace{\beta \cdot A \cdot (n-i) \cdot \sum_{r=i+1}^{n} N_1(r)}_{\text{(c)}}$$
$$\leq 1 + (1 + BA \cdot (n-i)) \cdot \sum_{r=i+1}^{n} N_1(r) = N_1(i).$$

Here, the term (a) counts the pop of y^β itself, the term (b) accounts for the right-hand side of the power relation for x_i which may be pushed onto the stack during reducing the exponent α_i in the collected part and, finally, the term (c) accounts for the right-hand sides of the conjugate relations which are pushed onto the stack each with a multiplicity of at most A, while moving the β occurrences of y into place. □

Remark 3.5.

1. It is easy to see, that the factor $(1+BA\cdot(n-i))$ in definition (2) of the cost function N_1 cannot be improved, that is, that the estimate of the number of words pushed onto the stack used in the proof of Theorem 3.4 is sharp. Consider, for example, collecting the product $x_1^B x_2^A \cdots x_n^A \cdot x_1^B$ in the group defined by the presentation

   $$\langle x_1, \ldots, x_n \mid x_1^{2B} = 1,\ x_k^{x_1} = x_k^{-1}\ (k = 2, \ldots, n),\ [x_k, x_l] = 1\ (k, l > 1)\rangle.$$

   In this collection, the power relation for x_1 is used once and conjugate relations involving x_1 are used $A \cdot B \cdot (n-1)$ times.

   Moreover, for a generic polycyclic group, generator powers for all generators $x_{i+1}, \ldots, x_n$ may be present at least in some of the relations used for collecting x_i, which means that the factor $\sum_{r=i+1}^{n} N_1(r)$ in (2) cannot be improved in general, either.

   In practice, however, most relations in a polycyclic presentation do not involve all the generators permitted by (1), and hence the cost estimate of Theorem 3.4 is usually extremely pessimistic; see Section 5.2. Still, it is obvious from Algorithm 2.3 (and an example similar to the above) that the cost can grow at least linearly in the bounds on the exponents.
2. Note that the exponential growth of the cost function $N_1(1)$ in n agrees with the results of Leedham-Green and Soicher (1990). However, as pointed out in Leedham-Green and Soicher (1990), collection from the left is exponential in the number of polycyclic generators even for finite Abelian polycyclic groups; this behaviour is caused by power relations. In contrast to this, it is clear from the proof of Theorem 3.4, that the complexity of collection from the left in terms of bounds on the exponents of collected words is governed by the conjugate relations. Since the analysis of Leedham-Green and Soicher (1990) mainly targets finite soluble groups, it does not discuss complexity in terms of exponents.

4. The Improved Algorithm

4.1. repeated squaring for exponents in the collected part

The first improvement can be achieved by addressing item 2 listed in Section 2.2. Instead of working with α_k copies of a normal form expression w for ${y_k}^y$, compute w^{α_k} in normal form by repeated squaring.

To do this, the collector must be called recursively. Since all words for which a power has to be computed in the course of collecting a power of x_i only involve generators x_k where $k > i$, this recursion is finite.

Algorithm 4.1. (Computing the Power of a Word) *Let w be a word in normal form and $\alpha \in \mathbb{N}$. The following algorithm computes the normal form u of w^α.*

- *Set $u := 1$.*
- *While $\alpha > 0$ do*
  - *If α is odd, set $u := u \cdot w$ in normal form.*
  - *Set $\alpha := \lfloor \frac{\alpha}{2} \rfloor$; if $\alpha > 0$, set $w := w \cdot w$ in normal form.*

Lemma 4.2. (Cost of Computing a Power) *Let $\alpha > 0$, let w be a word in normal form in the generators $x_{i+1}, \ldots, x_n$, and assume that the cost of any collection of a power of x_j in Algorithm 4.1 is bounded by $N(j)$. Then the cost of computing w^α using Algorithm 4.1 is bounded by*

$$2 \cdot (\lfloor \log_2 \alpha \rfloor + 1) \cdot \sum_{j=i+1}^{n} N(j).$$

Proof. The normal form expression for every power of w has the form $x_{i+1}^{\gamma_{i+1}} \cdots x_n^{\gamma_n}$ for some exponents $\gamma_{i+1}, \ldots, \gamma_n$, whence according to the assumption and Definition 3.1, the cost of every multiplication in Algorithm 4.1 is bounded by $\sum_{j=i+1}^{n} N(j)$. □

Remark 4.3. (Crossover Point) Using Lemma 4.2, we can estimate the crossover point, that is, the minimal exponent $\alpha > 0$, for which it pays to compute the normal form expression u of w^α first using Algorithm 4.1 and then to collect u, as opposed to collecting α copies of w.

Let the cost of any collection of a power of x_j during the computation of u and during the final collection of u be bounded by $N(j)$ and let the cost of any collection of a power of x_j during multiplication of α copies of w be bounded by $\widetilde{N}(j)$. Then, the cost of the first approach is at most $(2 \cdot (\lfloor \log_2 \alpha \rfloor + 1) + 1) \cdot \sum_{j=i+1}^{n} N(j)$, and the cost for the second approach is at most $\alpha \cdot \sum_{j=i+1}^{n} \widetilde{N}(j)$. Assuming $N(j) = \widetilde{N}(j)$ and equating both yields a theoretical crossover point $\alpha_c \approx 10$. In fact, explicitly counting the number of multiplications in Algorithm 4.1 for exponents less than 10 instead of using the bound $2 \cdot (\lfloor \log_2 \alpha \rfloor + 1)$ yields $\alpha_c \approx 5$.

Note that this is only a rough guide, since we just compared upper bounds for the costs. Moreover, the cost of collecting a product $x x_j^\beta$ into normal form depends on both β and the exponents in the collected part x, whence $N(j) = \widetilde{N}(j)$ is not strictly satisfied, since the intermediate collector states for both approaches will in general be different.

In practice, however, the exponents occurring in the intermediate collector states have the same order of magnitude for both approaches, justifying the assumption $N(j) = \widetilde{N}(j)$.

4.2. repeated squaring for exponents of generator powers on the stack

Assume a collector state as in Notation 2.1 and let the topmost item on the collector stack be y^β, where $y = x_i^{\pm 1}$ and $\beta > 0$. We fix this notation for this section.

To address item 1 listed in Section 2.2, we compute conjugates of the form $(x_k^\sigma)^{y^\beta}$ $(k = i+1, \ldots, n, \sigma \in \{-1, +1\})$ by considering $(x_k^\sigma)^y$ as image of x_k^σ under the automorphism of G_{i+1} induced by conjugation with y and computing powers of this automorphism by repeated squaring.

Notation 4.4. Define $\mathcal{G}_i := \{x_{i+1}, \ldots, x_n\} \cup \{x_k^{-1} \mid k \in \{i+1, \ldots, n\} \setminus I\}$.

Lemma 4.5. (Evaluating an Automorphism) *Let φ be an automorphism of G_{i+1} and assume that the images $\varphi(g)$ are known in normal form for all $g \in \mathcal{G}_i$. Then for any $x \in G_{i+1}$ in normal form, $\varphi(x)$ can be obtained by computing at most $n - i$ powers of normal form expressions in G_{i+1} and collecting at most $n - i - 1$ products of normal form expressions in G_{i+1}.*

Proof. Let $x = y_{i+1}^{\gamma_{i+1}} \cdots y_n^{\gamma_n}$ be in normal form with $y_k \in \mathcal{G}_i$ and $\gamma_k \geq 0$ for $k = i+1, \ldots, n$. Then, $\varphi(x) = \varphi(y_{i+1})^{\gamma_{i+1}} \cdots \varphi(y_n)^{\gamma_n}$, proving the claim. □

Algorithm 4.6. (Compute Higher Conjugates) *Let $\mathcal{G} \subseteq \mathcal{G}_i$, y and β as above. The following algorithm computes the conjugates $\overline{g} := g^{y^\beta}$ for all $g \in \mathcal{G}$.*

1. *Initialize $\overline{g} := g$ for all $g \in \mathcal{G}$.*
2. *Define $\varphi : G_{i+1} \to G_{i+1}$ by $\varphi(g) := g^y$ for all $g \in \mathcal{G}_i$.*
3. *While $\beta > 0$ do*
   - *If β is odd, set $\overline{g} := \varphi(\overline{g})$ in normal form for all $g \in \mathcal{G}$.*
   - *Set $\beta := \lfloor \frac{\beta}{2} \rfloor$; if $\beta > 0$, compute $\varphi^2(g) = \varphi(\varphi(g))$ in normal form for all $g \in \mathcal{G}_i$ and set $\varphi := \varphi^2$.*

Remark 4.7. Note that the normal forms of the conjugates g^y $(g \in \mathcal{G}_i)$ required in step 2 of Algorithm 4.6 can be read directly from the polycyclic presentation.

By Lemma 4.5, the images of elements in normal form under φ which are needed in step 3 can be obtained by computing powers and products of normal form expressions in G_{i+1}, requiring recursive use of the collector. Obviously, this recursion is finite. Power computations can be done, for example, using Algorithm 4.1. Note, however, that a naive approach may be faster in some cases; see Remark 4.3.

Lemma 4.8. (Cost of Computing Conjugates) *Let $\mathcal{G} \subseteq \mathcal{G}_i$, y and β as above. Assume that the cost of any collection of a power of x_j in Algorithm 4.6 is bounded by $N(j)$. Assume further that any computation of a power of a normal form expression performed in Algorithm 4.6 involves no more than M_1 multiplications of words in normal form. Then the cost of computing the conjugates g^{y^β} for all $g \in \mathcal{G}$ using Algorithm 4.6 is*

bounded by

$$(\lfloor \log_2 \beta \rfloor + 1) \cdot (|\mathcal{G}| + |\mathcal{G}_i|) \cdot (n-i) \cdot (M_1 + 1) \cdot \sum_{j=i+1}^{n} N(j).$$

PROOF. According to the assumption and Lemma 4.5, at most $(\lfloor \log_2 \beta \rfloor + 1) \cdot (|\mathcal{G}| + |\mathcal{G}_i|) \cdot (n-i) \cdot (M_1 + 1)$ products of normal form expressions for elements in G_{i+1} have to be computed in Algorithm 4.6. Since every such normal form expression is of the form $x_{i+1}^{\gamma_{i+1}} \cdots x_n^{\gamma_n}$ for some exponents $\gamma_{i+1}, \ldots, \gamma_n$, the cost for collecting each product into normal form is bounded by $\sum_{j=i+1}^{n} N(j)$ according to the assumption and Definition 3.1. □

REMARK 4.9. (CROSSOVER POINT) Using Lemma 4.8, we can estimate the crossover point, that is, the minimal value of β, for which it pays to compute the expressions in normal form for all needed conjugates g^{y^β} $(g \in \mathcal{G})$ by using Algorithm 4.6, as opposed to working exclusively with the relations of the polycyclic presentation and collecting β instances of y separately.

Assume that for the first approach, the cost of any collection of a power of x_j is bounded by $N(j)$. Let M_1 be the bound on the number of word multiplications per power computation from Lemma 4.8 and let M_2 be a bound on the number of word multiplications, necessary for any power computation during raising the conjugates g^{y^β} $(g \in \mathcal{G})$ to the powers indicated by the exponents of the collected part, as discussed in Section 4.1.

Then the cost for computing all conjugates g^{y^β} $(g \in \mathcal{G})$, raising them to the needed power, and collecting the resulting words in normal form is at most

$$((\lfloor \log_2 \beta \rfloor + 1) \cdot (|\mathcal{G}| + |\mathcal{G}_i|) \cdot (n-i) \cdot (M_1 + 1) + |\mathcal{G}| \cdot (M_2 + 1)) \cdot \sum_{j=i+1}^{n} N(j). \quad (3)$$

Assuming analogous bounds $\widetilde{N}(j)$ and $\widetilde{M_2}$ for the second approach, the cost of β times raising the conjugates g^y $(g \in \mathcal{G})$ to some power, and collecting the resulting words in normal form is at most

$$\beta \cdot |\mathcal{G}| \cdot (\widetilde{M_2} + 1) \cdot \sum_{j=i+1}^{n} \widetilde{N}(j). \quad (4)$$

Assuming $N(j) = \widetilde{N}(j)$ and $M_2 = \widetilde{M_2}$, equating (3) and (4) yields a theoretical crossover point β_c given by

$$\frac{\beta_c - 1}{\lfloor \log_2 \beta_c \rfloor + 1} = \left(1 + \frac{|\mathcal{G}_i|}{|\mathcal{G}|}\right)(n-i) \cdot \frac{M_1 + 1}{M_2 + 1}. \quad (5)$$

The assumptions $N(j) = \widetilde{N}(j)$ and $M_2 = \widetilde{M_2}$ can be justified by comments similar to the ones in Remark 4.3. We see that the repeated squaring method used in Algorithm 4.6 becomes more expensive with growing $n - i$, since the cost of evaluating φ in step 3 of Algorithm 4.6 is proportional to $n - i$ by Lemma 4.5. Moreover, the cost of computing φ^2 is proportional to $|\mathcal{G}_i|$, whence the overhead from this precomputation is the more important the larger the ratio $\frac{|\mathcal{G}_i|}{|\mathcal{G}|}$ is.

Note that M_2 is determined by the order of the exponents in the collected part during the collection, whereas M_1 is determined by the order of the exponents in the conjugates g^{y^γ} $(g \in \mathcal{G}_i,\ 0 \leq \gamma < \beta)$. Hence, the ratio $\frac{M_1+1}{M_2+1}$ is not known when a decision between

the approaches has to be made. The simplest possibility is to assume $M_1 \approx M_2$, which may result in bad choices; the actual crossover point will be smaller (larger) than the estimated one, if the exponents in the conjugates g^{y^γ} ($g \in \mathcal{G}_i$, $0 \leq \gamma < \beta$) on average are smaller (larger) than the exponents in the collected part.

However, if the caching mentioned in Remark 4.15 is applied, information about the exponents in the conjugates $g^{y^{\pm 2^l}}$ ($g \in \mathcal{G}_i$, $l = 0, 1, \ldots$) is available and can in principle be used for determining the crossover point.

4.3. A more efficient version of collection from the left

Using the results obtained so far in this section, we can formulate a collection algorithm in the sense of Notation 2.1, which is more efficient for computations involving large exponents than Algorithm 2.3. Like Algorithm 2.3, this collection algorithm can be seen as a specialization of Algorithm 2.2; we refer to it as Algorithm 4.10.

1. In step 1 of Algorithm 2.2, $\sigma := \beta$ is chosen. This means that the item y^β on the collector stack is removed from the collector stack in one pass.
2. In step 4 of Algorithm 2.2, first normal form expressions c_k for the conjugates ${y_k}^{y^\beta}$ ($k = i+1, \ldots, n$) are computed using Algorithm 4.6. Subsequently, for $k = n, \ldots, i+1$ the normal form of $c_k^{\alpha_k}$ is computed using Algorithm 4.1 and pushed onto the collector stack.

Definition 4.11. Let A, B and C be fixed positive integers. For $i \in \{1, \ldots, n\}$ define $N_2(i)$ recursively by $N_2(n) := 1$ and

$$N_2(i) := 1 + (3 \cdot (n-i)^2 \cdot (\lfloor \log_2 B \rfloor + 1) \cdot (2(\lfloor \log_2 C \rfloor + 1) + 1) + 2 \cdot (n-i) \cdot (\lfloor \log_2 A \rfloor + 1) + (n-i+1)) \cdot \sum_{r=i+1}^{n} N_2(r)$$

for $i < n$.

Proposition 4.12. *Assuming $A = B = C$ and considering $N_2(i)$ as a function of A only, that is, regarding n and i as fixed, we have*

$$N_2(i) = 6^{n-i} \cdot ((n-i)!)^2 \cdot (\log_2 A)^{2\cdot(n-i)} + o((\log_2 A)^{2\cdot(n-i)}).$$

Proof. This follows from the definition by induction, writing

$$N_2(i) = \sum_{k=0}^{2\cdot(n-i)} n_k^{(i)} \cdot (\log_2 A)^k$$

and comparing leading coefficients. □

Theorem 4.13. *Assume a collector state as in Notation 2.1 and let the collection algorithm $\mathcal{C}$ be given by Algorithm 4.10. Let the topmost item on the collector stack be y^β, where $y = x_i^{\pm 1}$ and $\beta > 0$. Let A, B and C be constants such that during the entire collection of y^β, the absolute values of the exponents $\alpha_1, \ldots, \alpha_n$ are bounded by A, the absolute values of the exponents of the generator powers on the collector stack are bounded by B,*

and the absolute values of the exponents of the elements $\overline{g}$ $(g \in \mathcal{G})$ *and* $\varphi(g)$ $(g \in \mathcal{G}_i)$ *in every step of Algorithm* 4.6 *are bounded by* C*. Then,*

$$P(a, s, \mathcal{C}) \leq N_2(i).$$

PROOF. If $i = n$, $P(a, s, \mathcal{C}) = 1 = N_2(n)$, so assume $i < n$ and suppose the theorem holds for states of the collector for which the topmost item on the collector stack is a power of $x_{i'}$, where $i' > i$.

Since every word pushed onto the stack during collecting y^β only involves the generators $x_{i+1}, \ldots, x_n$, the theorem can be used to estimate the cost of collecting these generator powers and every word in normal form which is pushed onto the stack can be collected at a cost of at most $\sum_{r=i+1}^{n} N_2(r)$. Using Lemma 4.2 and Lemma 4.8, we then obtain

$$\begin{aligned} P(a, s, \mathcal{C}) \leq\ & \underbrace{1}_{\text{(a)}} + \underbrace{\sum_{r=i+1}^{n} N_2(r)}_{\text{(b)}} \\ & + \underbrace{3 \cdot (n-i)^2 \cdot (\lfloor \log_2 B \rfloor + 1) \cdot (2(\lfloor \log_2 C \rfloor + 1) + 1) \cdot \sum_{r=i+1}^{n} N_2(r)}_{\text{(c)}} \\ & + \underbrace{(n-i) \cdot 2(\lfloor \log_2 A \rfloor + 1) \cdot \sum_{r=i+1}^{n} N_2(r)}_{\text{(d)}} + \underbrace{(n-i) \cdot \sum_{r=i+1}^{n} N_2(r)}_{\text{(e)}} \\ =\ & N_2(i). \end{aligned}$$

Here, term (a) counts the pop of y^β itself and term (b) accounts for the right-hand side of the power relation for x_i which may be pushed onto the stack during reducing the exponent α_i in the collected part. Term (c) is a bound for computing the higher conjugates ${y_k}^{y^\beta}$ $(k = i+1, \ldots, n)$ in normal form using Algorithm 4.6, obtained by applying Lemma 4.8 with $M_1 = 2(\lfloor \log_2 C \rfloor + 1)$ and using $|\mathcal{G}_i| \leq 2(n-i)$ and $|\mathcal{G}| \leq n-i$. Term (d) estimates the cost of computing the conjugates $(y_k^{\alpha_k})^{y^\beta}$ in normal form as powers of the words ${y_k}^{y^\beta}$ $(k = i+1, \ldots, n)$ using Algorithm 4.1, by applying Lemma 4.2. Term (e), finally, accounts for the cost of collecting the normal form expressions for the conjugates $(y_k^{\alpha_k})^{y^\beta}$ $(k = i+1, \ldots, n)$. □

REMARK 4.14.

1. As was the case for the analysis performed in Section 3, the theoretical cost estimate given in Theorem 4.13 is usually very pessimistic; see Section 5.2.
2. Like the cost function $N_1(1)$, the cost function $N_2(1)$ exhibits an exponential growth in n and its growth in terms of exponent bounds is governed by the influence of the conjugate relations.

REMARK 4.15. In principle, the cost of computing higher conjugates using Algorithm 4.6 can be reduced significantly by caching the normal form expressions for $g^{y^{\pm 2^l}}$

($g \in \mathcal{G}_i,\ l \in \mathbb{N}$), instead of recomputing them for every collection. This could be interpreted as adding further (redundant) relations to the polycyclic presentation. The factor $3 \cdot (n-i)^2$ in the definition of the cost function (Definition 4.11) could be replaced by a factor of $(n-i)^2$, yielding an asymptotic complexity of $N_2^{cached}(i) = 2^{n-i} \cdot ((n-i)!)^2 \cdot (\log_2 A)^{2\cdot(n-i)} + o((\log_2 A)^{2\cdot(n-i)})$.

As mentioned in Remark 4.9, the cached information about the exponents in the higher conjugates would also facilitate a more reliable estimate of the crossover point between Algorithm 4.6 and the naive approach.

5. Comparing the Two Collection Algorithms

Theorems 3.4 and 4.13 give an estimate of the cost of collection, assuming upper bounds for the exponents of *all intermediate collector states* of a collection as opposed to working with bounds on the exponents of the *input words*.

Both from the complexity analysis point of view and for the purpose of comparing the two collection algorithms, it would be highly desirable to express the bounds used in Theorems 3.4 and 4.13 in terms of bounds on the exponents of the input words. Unfortunately, estimating the growth of exponents in the course of a collection is not only extremely complicated, but it also leads to theoretical bounds for the exponents of intermediate collector states, which are orders of magnitude larger than the maximal exponents actually observed. Because of this, any comparison of the collection algorithms, that is based on worst case bounds for exponents of intermediate collector states derived theoretically from bounds on the exponents of input words, would be meaningless in practice.

Instead of working with bounds on the exponents of input words, we therefore decided to work with *a posteriori* bounds, that is, the maximal exponents actually observed during a collection. Of course, even when collecting the same word, the *a posteriori* exponent bounds for the two collection algorithms will in general be different. Hence, care has to be taken when using Theorems 3.4 and 4.13 with such *a posteriori* bounds for a comparison of Algorithms 2.3 and 4.10.

Looking at Algorithms 2.3 and 4.10, we see that the intermediate collector states are of a very similar structure. Once the higher conjugates $g^{y^{2^l}}$ are known for $g \in \mathcal{G}_i$ and $l \in \mathbb{N}$, the main difference between the algorithms is that the order in which a product of generators is evaluated is different. Significant differences in the *a posteriori* bounds hence can occur only for two reasons:

- Computing some higher conjugate $g^{y^{2^l}}$ in Algorithm 4.10 involves products which do not have to be evaluated at all in Algorithm 2.3, for example, because the exponent of g in the collected part always remains zero, and the exponents occurring in the evaluation of these products are much larger than the maximal exponents in the rest of the collection.

  This can certainly happen, but one would expect this to be a rather exceptional case. Moreover, if the caching mentioned in Remark 4.15 is applied, this effect can only occur during some initial phase until the cache is sufficiently filled.
- For one collection algorithm, exponents in intermediate collector states get much larger than the exponents in the final result and cancel only towards the end of the collection, while for the other collection algorithm, cancellations happen more evenly distributed during the collection, preventing exponents from getting large.

While it is likely that examples exhibiting this behaviour can be constructed favouring either collection algorithm, such a situation appears to be rare.

With regard to the above considerations, we postulate that in almost all situations, the *a posteriori* bounds for the collection algorithms, Algorithms 2.3 and 4.10, are of the same order of magnitude:

POSTULATE 5.1. *Consider a collector state as in Notation* 2.1. *Let* A_1 *and* B_1 *be the minimal possible values of the constants* A *and* B *from Theorem* 3.4 *and let* A_2, B_2 *and* C_2 *be the minimal possible values of the constants* A, B *and* C *from Theorem* 4.13. *Then, in most situations,*

$$\max\{A_1, B_1\} \approx \max\{A_2, B_2, C_2\}.$$

In all computations performed so far, not a single case has been encountered, where this assumption was violated. The examples given in Section 5.2 corroborate the claim that Postulate 5.1 is justified.

Assuming the validity of Postulate 5.1, introducing a common *a posteriori* bound K for the absolute values of the exponents in all intermediate collector states, and using this value for the constants A and B from Theorem 3.4 and for the constants A, B and C from Theorem 4.13, we obtain from Proposition 3.3 an upper bound for the asymptotic complexity of Algorithm 2.3 of

$$(n-1)! \cdot K^{2\cdot(n-1)} + o(K^{2\cdot(n-1)})$$

and from Proposition 4.12 an upper bound for the asymptotic complexity of Algorithm 4.10 of

$$6^{n-1} \cdot ((n-1)!)^2 \cdot (\log_2 K)^{2\cdot(n-1)} + o((\log_2 K)^{2\cdot(n-1)}).$$

As seen in Remark 3.5 (1),

$$(n-1) \cdot K^2$$

is a lower bound for the asymptotic complexity of Algorithm 2.3, showing the superiority of Algorithm 4.10 for large exponents.

5.1. A HYBRID METHOD

To avoid performance penalties for small exponents, mainly caused by the overhead of setting up the data structures used for computing the higher conjugates in Algorithm 4.6, a hybrid collector suggests itself.

Such a hybrid collector can choose between Algorithms 4.10 and 2.3 independently for every generator power y^β which has to be worked off the stack, the choice depending on β, $n-i$ and possibly on further criteria. Similarly, powers of words can be computed either using repeated squaring or ordinary multiplication, depending on the exponent.

Timing results for such a hybrid collector are included in Section 5.2. For computing powers of words, we use the crossover point $\alpha_c = 5$ established in Remark 4.3. In regard of the difficulties mentioned in Remark 4.9, the crossover between Algorithms 4.10 and 2.3 was tuned in experiments, starting from the estimate given in Remark 4.9.

Table 1. Squaring of random words in the group G_1; see text.

	m	2	5	10	40	100	1000	10 000	100 000
A1	bound	104	226	520	1175	4867	—	—	—
	pops	205 468	$1.2 \cdot 10^6$	$8.0 \cdot 10^6$	$6.4 \cdot 10^7$	$5.2 \cdot 10^9$	—	—	—
	time[s]	0.8	4.8	30	240	24 000	—	—	—
A2	bound	86	167	325	1001	2879	24740	282 658	$2.4 \cdot 10^6$
	pops	676 763	829 949	$1.2 \cdot 10^6$	$1.6 \cdot 10^6$	$2.9 \cdot 10^6$	$6.7 \cdot 10^6$	$1.4 \cdot 10^7$	$1.9 \cdot 10^7$
	time[s]	7.8	9.7	14	19	35	83	180	240
A3	bound	107	250	565	1351	5762	54 147	—	—
	pops	125 030	357 481	$1.2 \cdot 10^6$	$4.5 \cdot 10^6$	$5.7 \cdot 10^7$	$5.7 \cdot 10^9$	—	—
	time[s]	1.2	3.5	12	46	580	64 000	—	—
A4	bound	106	248	539	1432	6961	41 349	635 067	$5.7 \cdot 10^6$
	pops	105 629	305 703	654 655	$1.1 \cdot 10^6$	$2.0 \cdot 10^6$	$5.8 \cdot 10^6$	$1.4 \cdot 10^7$	$1.9 \cdot 10^7$
	time[s]	0.7	2.5	5.9	11	21	67	170	230

5.2. TIMING RESULTS

In this section we give timing results for four collection algorithms:

A1 Algorithm 2.3, the established collection algorithm.
A2 Algorithm 4.10, the collection algorithm presented in Section 4.
A3 A modification of Algorithm 2.3, which makes use of Algorithm 4.1 for computing powers of conjugates, but does not compute higher conjugates as described in Section 4.2.
A4 A hybrid algorithm as sketched in Section 5.1.

Note that in the implementations of all these algorithms, several obvious optimizations have been applied. For example, commuting generators are treated more efficiently than in the literal versions of Algorithms 2.3 and 4.10.

The performance of the collection algorithms for different orders of exponents is tested by computing the squares of random elements. For various values of a parameter m, we choose a set of random elements such that exponents not bounded by power relations are randomly distributed in $\{-m, \ldots, m\}$. For every such set and every collection algorithm, we then compute the squares of all elements in the set, recording the average values of the *a posteriori* bound, the number of stack pops during the entire collection, and the running time.

We perform this test for the following groups.

1. A group G_1 presented on 25 generators with five power relations and 423 non-trivial conjugate relations. The maximal exponent of the right-hand side of a polycyclic relation is 15. This is a fairly complicated presentation with many generators (Table 1).
2. The group

$$G_2 = \langle\, a, b, c, d, e \,|\, c^6 = e^3 = 1, b^a = b^{a^{-1}} = b^{-1}, c^b = c^{b^{-1}} = ce, \\ d^b = d^{b^{-1}} = d^{-1}, d^c = de, e^b = e^{b^{-1}} = e^2 = e^c, \\ \text{remaining conjugate relations trivial} \,\rangle\,.$$

Table 2. Squaring of random words in the group G_2; see text.

	m	2	5	10	20	40	100	1000	10 000	100 000	1 000 000
	bound	4	7	13	26	52	131	1307	12 440	—	—
A1	pops	28	70	167	464	1546	8267	764 710	$7.0 \cdot 10^7$	—	—
	time[ms]	0.1	0.2	0.4	1.0	3.3	17	1600	150 000	—	—
	bound	4	7	13	26	52	131	1307	12 440	131 477	$1.2 \cdot 10^6$
A2	pops	37	61	77	94	113	140	207	275	340	409
	time[ms]	0.3	0.4	0.5	0.7	0.8	1.0	1.6	2.2	2.3	3.0
	bound	4	7	13	26	52	131	1307	12 440	131 477	—
A3	pops	34	81	158	324	712	1999	27 096	343 397	$4.4 \cdot 10^6$	—
	time[ms]	0.2	0.5	0.9	1.9	4.4	13	180	2300	31 000	—
	bound	4	7	13	26	52	131	1307	12 440	131 477	$1.2 \cdot 10^6$
A4	pops	33	75	97	108	124	147	217	291	346	406
	time[ms]	0.2	0.3	0.5	0.6	0.8	1.0	1.4	2.0	2.5	2.8

Table 3. Squaring of random words in the group G_3; see text.

	m	2	5	10	20	40	100	1000	10 000
	bound	3846	42 004	308 217	$2.2 \cdot 10^6$	$1.7 \cdot 10^7$	$2.8 \cdot 10^8$	—	—
A1	pops	225	2986	23 306	173 585	$1.4 \cdot 10^6$	$2.1 \cdot 10^7$	—	—
	time[ms]	0.3	4.2	33	250	1900	31 000	—	—
	bound	3864	42 097	308 574	$2.2 \cdot 10^6$	$1.7 \cdot 10^7$	$2.8 \cdot 10^8$	$2.1 \cdot 10^{11}$	$2.1 \cdot 10^{14}$
A2	pops	64	203	379	618	925	1478	3371	6246
	time[ms]	0.4	1.0	1.9	3.2	4.9	7.7	18	36
	bound	3779	41 952	308 188	$2.2 \cdot 10^6$	$1.7 \cdot 10^7$	$2.8 \cdot 10^8$	$2.1 \cdot 10^{11}$	—
A3	pops	44	239	888	3392	13 955	97 427	$1.1 \cdot 10^7$	—
	time[ms]	0.2	0.9	3.3	13	52	370	43 000	—
	bound	3779	41 830	308 491	$2.2 \cdot 10^6$	$1.7 \cdot 10^7$	$2.8 \cdot 10^8$	$2.1 \cdot 10^{11}$	$2.1 \cdot 10^{14}$
A4	pops	41	215	433	697	1003	1513	3448	6251
	time[ms]	0.2	0.9	2.1	3.5	5.2	7.9	19	36

A pretty simple presentation (Table 2).

3. The group

$$G_3 = \langle\, a, b, c, d \,|\, b^a = bc^{32}, b^{a^{-1}} = bc^{-32}d^{1024},$$
$$c^a = cd^{32} = c^b, c^{a^{-1}} = cd^{-32} = c^{b^{-1}} \,\rangle\,.$$

The exponents in the normal forms of higher conjugates $x_j^{x_i^\beta}$ grow exponentially in $|\beta|$ for this presentation (Table 3).

All tests listed in this section were run on a Sun E450 with four 400 MHz UltraSparc-II CPUs and 4 GB of RAM, using a development version of Magma.

We note the following points.

- Postulate 5.1 is a very good description of the observed *a posteriori* bounds. For one of the examples the *a posteriori* bounds are exactly equal, in the other cases

the differences are minor. The maximal observed differences are a factor of about 2 (for the group G_1), in fact favouring Algorithm 4.10.

- Comparing the three "pure" Algorithms A1, A2 and A3 in terms of the number of stack pops needed, A2 yields the minimal number of stack pops except for very small values of m ($m \leq 2 \ldots 5$, depending on the presentation). However, the time spent per stack pop for algorithm A1 is significantly smaller than for the other algorithms; A2 spends the largest amount of time per stack pop.

  Because of both these effects, A1 or A3 tend to give smaller running times than A2 for small values of m ($m \leq 5 \ldots 40$, depending on the presentation). For larger values of m, A2 clearly yields the shortest running times, and in fact for large values of m ($m \geq 1\,000 \ldots 10\,000$, again depending on the presentation), Algorithm 4.10 is the only of the "pure" collection algorithms with acceptable running times.
- Over the entire range of exponents, the hybrid method A4 in general yields a performance better than or similar to the best of the "pure" algorithms. It is therefore the most practical of the tested collection algorithms.

  Note that in some cases, the hybrid method is even significantly faster than the fastest "pure" algorithm. The reason for this is that decisions about which algorithm to use are made independently for every generator power which is popped off the stack, and each power of a word which is computed. Hence, in theory, the optimal method can be used in each step of the collection. In practice, however, as mentioned in Remark 4.9, the decision will not be optimal in all cases. In fact, the choice of the crossover point used in the hybrid algorithm A4 assumes that the exponents in the higher conjugates $x_j^{x_i^{\pm 2^l}}$ are approximately of order 2^l. This is more or less the case for the group G_1, but is not satisfied at all for the groups G_2 (where they are bounded by 2) and G_3 (where they grow much faster). It is therefore not surprising, that the hybrid method A4 does not perform quite as well for the groups G_2 and G_3 as it does for the group G_1.

6. Conclusions

Because of its better complexity in terms of exponents of elements, Algorithm 4.10 significantly speeds up computing in polycyclic groups if the entries of the exponent vectors reach absolute values of about $10 \ldots 100$, depending on the presentation. Hence, it is certainly relevant for practical purposes not only in infinite polycyclic groups, but also for finite soluble groups, unless all cyclic factors have rather small order. In fact, collection using Algorithm 4.10 is the only practical way of computing with elements whose exponents exceed orders of $1\,000 \ldots 100\,000$, depending on the presentation.

The most flexible collection algorithm can be obtained using a hybrid strategy, choosing for each individual collection step the most suitable algorithm; this offers the benefits of efficient treatment of large exponents and avoids performance penalties due to unnecessary overheads for small exponents.

Acknowledgement

I would like to thank the unknown referee for his very valuable and helpful comments on an earlier version of this article.

References

Bosma, W., Cannon, J., Playoust, C. (1997). The MAGMA algebra system I: the user language. *J. Symb. Comput*, **24**, 235–265.

Eick, B. (2001). On the fitting subgroup of a polycyclic-by-finite group and its applications. *J. Algebra*, **242**, 176–187.

GAP (2000). *GAP—Groups, Algorithms, and Programming*. Aachen, St Andrews, The GAP Group; `http://www-gap.dcs.st-and.ac.uk/~gap`.

Leedham-Green, C. R., Soicher, L. H. (1990). Collection from the left and other strategies. *J. Symb. Comput.*, **9**, 665–675. (*Computational Group Theory*, Part 1).

Lo, E. H. (1998). Finding intersections and normalizers in finitely generated nilpotent groups. *J. Symb. Comput.*, **25**, 45–59.

Vaughan-Lee, M. R. (1990). Collection from the left. *J. Symb. Comput.*, **9**, 725–733. (*Computational Group Theory*, Part 1).

Received December 6 2001
Accepted 18 May 2002