AAECC 15, 213–231 (2004)
Digital Object Identifier (DOI) 10.1007/s00200-004-0157-5

AAECC
Applicable Algebra in Engineering, Communication and Computing

# DFT-based Word Normalization in Finite Supersolvable Groups

**Meinard Müller, Michael Clausen**

Universität Bonn, Institut für Informatik III, Römerstrasse 164, 53117 Bonn, Germany
(e-mail: {meinard, clausen}@cs.uni-bonn.de)

Received: August 28, 2002; revised version: March 24, 2004
Published online: September 6, 2004 – © Springer-Verlag 2004

**Abstract.** In this paper we discuss the word normalization problem in pc presented finite supersolvable groups: given two group elements $a$ and $b$ in normal form the normal form of the product $a \cdot b$ is to be computed. As an alternative to classical collection strategies we present a new DFT-based strategy, which uses fragments of certain irreducible representations of the underlying group. This strategy allows an explicit running time analysis. For example, in the special case of a pc presented $p$-group $G$ of order $p^n$ one needs at most $5 \cdot p \cdot n^2$ additions in $\mathbb{Z}_e := \mathbb{Z}/e\mathbb{Z}$ for the computation of the normal form, where $e$ denotes the exponent of $G$. Interpreting pc presentations as polynomials in multivariate non-commutative polynomial rings we derive an algorithm for fast polynomial division.

**Keywords:** Discrete Fourier transform, Word normalization, Power-commutator presentation, Supersolvable groups

## 1 Introduction

In this paper we are going to study word normalization problems in *finite supersolvable groups*. Recall that a finite group $G$ is said to be *supersolvable* if it has a chain of subgroups

$$\mathcal{C} = (G = G_n \supset G_{n-1} \supset \ldots \supset G_1 \supset G_0 = \{1\}), \tag{1}$$

where each subgroup $G_{i-1}$ is normal in $G$, denoted by $G_{i-1} \trianglelefteq G$, and each factor $G_i/G_{i-1}$ is of prime order $p_i$, $1 \le i \le n$. Such a series is also called a *chief series* of $G$. (Note that for solvable groups there exists a series as in (1) with the weaker condition $G_i \lhd G_{i+1}$, $0 \le i < n$. Hence, each supersolvable group is solvable. On the other hand, all abelian groups, all $p$-groups and all nilpotent groups of finite order are supersolvable.) For each $i \in [1 : n] := \{1, 2, \ldots , n\}$

let $g_i$ be an element in $G_i \setminus G_{i-1}$, then $G$ is generated by $g_1, \dots, g_n$. Even more, each element $g \in G$ has the unique expression

$$g = g_n^{e_n} \cdot g_{n-1}^{e_{n-1}} \cdot \dots \cdot g_1^{e_1} \quad (0 \le e_i < p_i), \tag{2}$$

which is also referred to as *the normal form* of $g$. The multiplication in $G$ is completely described, if the normal forms of all powers $g_i^{p_i}$ and all commutators $[g_i, g_j] := g_i^{-1} g_j^{-1} g_i g_j$ are known. More formally, every supersolvable group has a *power-commutator presentation* (pc presentation) of the form

$$G = \langle g_1, \dots, g_n \mid g_i^{p_i} = u_i \ (1 \le i \le n),\ [g_i, g_j] = w_{ij} \ (1 \le i < j \le n) \rangle, \tag{3}$$

with generators $g_i \in G_i \setminus G_{i-1}$ and words $u_i \in G_{i-1}$ and $w_{ij} \in G_i$, all given in normal form:

$$u_i = g_{i-1}^{a_{i,i-1}} \cdot \dots \cdot g_1^{a_{i,1}} \qquad \text{and} \qquad w_{ij} = g_i^{b_{ij,i}} \cdot \dots \cdot g_1^{b_{ij,1}}. \tag{4}$$

Conversely, every pc presentation of the form (3) defines a supersolvable group. The pc presentation is called *consistent* if the resulting group has order $p_1 p_2 \dots p_n$. We refer to [12] for further details.

A classical word normalization problem is the task to compute for two elements $g^\alpha := g_n^{\alpha_n} \cdot \dots \cdot g_1^{\alpha_1}$ and $g^\beta$, both in normal form, the normal form of their product: $g^\gamma = g^\alpha \cdot g^\beta$. More generally, if $\phi\colon [1 : L] \to [1 : n]$ and $\psi\colon [1 : L] \to \mathbb{Z} \setminus \{0\}$, then one could ask for a fast algorithm that on input $(\phi, \psi)$ outputs the normal form of the word

$$g_\varphi^\psi \ := \ g_{\varphi(1)}{}^{\psi(1)} \cdot g_{\varphi(2)}{}^{\psi(2)} \cdot \dots \cdot g_{\varphi(L)}{}^{\psi(L)}. \tag{5}$$

Even more general is the task to compute the normal form of a regular expression in the generators like $((g_1^{23} \cdot g_6^5 \cdot g_3^9)^{-35} \cdot ((g_4^{12} \cdot g_6^2)^{42} \cdot (g_3^4 \cdot g_4^5)^{19})^{-33})^{14}$ or of a group element defined by a straightline program, see, e.g., [3]. There are also word normalization problems in disguise. One example, studied in Section 3, is the computation of the orders of the generators.

Classical strategies for solving word normalization problems of the form $g^\gamma = g^\alpha \cdot g^\beta$ involve various kinds of *collection* processes (see, e.g., [12]) or Hall polynomials combined with interpolation techniques (see, e.g., [13]). To the best of our knowledge, there is no strategy that is always superior to all other strategies. Furthermore, the only known upper bounds for the number of arithmetic operations needed to compute the normal form are exponential in the input size.

For the class of pc presented finite supersolvable groups, we propose a new DFT-based word normalization strategy, which principally differs from classical collection strategies used in computer algebra system such as GAP or MAGMA.

To determine the normal form, we use certain fragments of irreducible representations of the underlying group which are computed in a preprocessing step. Our DFT-based strategy has the following advantages over classical strategies:

- It allows an explicit running time analysis. The upper bound for the number of operations needed in the normal form computation is roughly linear in the length of the word to be normalized, and quadratic in the length $n$ of the chief series $\mathcal{C}$ of $G$.
- It works efficiently even in the case of normalizing regular expressions. The crucial point is that such a regular expression has not to be resolved in order to compute the normal form.

Before we explain the basic idea of our approach, we briefly recall some facts from representation theory (see [4, 11]). Let $G$ denote a finite group of order $N$. The vector space $\mathbb{C}G := \{a | a : G \to \mathbb{C}\}$ of all complex valued functions on $G$ with multiplication $(ab)(x) := \sum_{g \in G} a(g)b(g^{-1}x)$ for $a, b \in \mathbb{C}G$ and all $x \in G$ defines an associative algebra, the so-called *group algebra* of $G$. By Wedderburn's Theorem, the group algebra $\mathbb{C}G$ (the *signal domain*) is isomorphic to an algebra of block diagonal matrices (the *spectral domain*):

$$D = \oplus_{k=1}^{h} D_k : \mathbb{C}G \longrightarrow \oplus_{k=1}^{h} \mathbb{C}^{d_k \times d_k}. \tag{6}$$

Here, the number $h$ of blocks equals the number of conjugacy classes of $G$ and the projections $D_1, \dots , D_h$, also denoted by $\mathrm{Irr}(G)$, form a transversal of irreducible matrix representations of $\mathbb{C}G$. Every such $D$ is called a *discrete Fourier transform* (DFT) of $G$. One can show that $D$ is determined up to the ordering of the $h$ blocks and the choice of basis in each of these blocks. For supersolvable groups $G$, Baum and Clausen [1] have designed an efficient algorithm, in the following also denoted as BC-Algorithm, to compute a very special DFT $D$ of $G$: all representing matrices $D(g_i)$ on the generators $g_i$, $1 \leq i \leq n$, are monomial having only $e$-th roots of unity as non-zero entries. Here $e$ denotes the exponent of $G$, which is the least common multiple of the orders of the elements of $G$. This allows very efficient computations in the spectral domain. Furthermore, since all matrix operations in the spectral domain will be multiplications or inversions, one can compute purely symbolically in the additive group $\mathbb{Z}_e := \mathbb{Z}/e\mathbb{Z}$.

There are many examples, where computations can be speeded up using DFT-based methods. The most prominent example is polynomial multiplication or cyclic convolution which amounts to multiplication in the group algebra $\mathbb{C}C_N$, where $C_N$ denotes the cyclic group of order $N$. Here the corresponding DFT can be used to transfer the problem from the signal to the spectral domain, where convolution reduces to pointwise multiplication. Another more algebraic example is the efficient DFT-based computation of character tables of $p$-groups, see [14].

To explain the main idea of our DFT-based normalization algorithm we consider the problem of computing the normal form of the product $a \cdot b$, where

$a, b \in G$. Intuitively, we translate the problem by means of a suitable DFT $D$ into the spectral domain: $D(a \cdot b) = D(a) \cdot D(b)$. Here, the multiplication can be done efficiently. The translation of the product back into the signal domain, which is in this case simply the group, can be done by using only fragments of certain irreducible matrix representations – one for each generator $g_i$, $1 \leq i \leq n$. As a special case of our normalization algorithm, we obtain the following:

**Corollary 1.1** *Let $G$ be a pc presented $p$-group of order $p^n$ and exponent $e$, with corresponding chief series $\mathcal{C}$. Then, given (suitable parts of) a $\mathcal{C}$-adapted DFT, normalization of the product of two normal words in $G$ can be done with at most $5 \cdot p \cdot n^2$ additions in $\mathbb{Z}_e$.*

The rest of this paper is organized as follows. In Section 2, we introduce some notation and sketch the main ideas of the BC-Algorithm. Section 3 discusses some basic properties of $e$-monomial matrices and explains how to compute efficiently the orders of the generators from parts of the DFT-information. In Section 4, we describe our DFT-based word normalization procedure, referred to as the WN-Algorithm, and give a detailed complexity analysis. Section 5 gives some implementation details and running times. As an application, we discuss in Section 6 how pc presentations can be interpreted as certain Gröbner bases in non-commutative polynomial rings. In this context, the WN-Algorithm can be used for fast polynomial division.

## 2 BC-Algorithm

We assume that the reader is familiar with the basic notions and facts from representation theory (see, e.g., [11] as a standard reference). In the following, we briefly recall the main ideas of the BC-Algorithm thereby we fix the notation and describe a memory efficient data structure to store the DFT data. For a more detailed account, the reader is referred to [1, 4, 10].

For the rest of this section, let $G$ denote a finite supersolvable group. The BC-Algorithm, which requires as input a consistent pc presentation of $G$, is based on the following version of Clifford's Theorem:

**Theorem 2.1** (Clifford's Theorem) *Let $G$ and $\mathcal{C}$ be as above. Given an irreducible representation $F$ of $\mathbb{C}G_{i-1}$, $0 < i \leq n$, one of the following two cases holds:*

(1) *$F$ extends to $p_i$ pairwise inequivalent irreducible representations $D_0, \ldots, D_{p_i-1}$ of $\mathbb{C}G_i$ of degree $\deg(F)$. Moreover, if $\chi^0, \chi^1, \ldots, \chi^{p_i-1}$ are the linear characters of the cyclic group $G_i/G_{i-1}$ in a suitable order, we have $D_k = \chi^k \otimes D_0$.*

(2) *The induction of $F$ to $\mathbb{C}G_i$ is an irreducible representation of degree $p_i \cdot \deg(F)$.*

*Up to equivalence, all irreducible representations of $\mathbb{C}G_i$ can be obtained in this way.*

This allows us to construct the irreducible representations of $G$ iteratively in a bottom-up fashion along the chief series $\mathcal{C}$. Analogously to $G$, define $\mathrm{Irr}(G_i)$ for the subgroups $G_i$. Then, at each level $i$, $1 \leq i \leq n$, a set $\mathrm{Irr}(G_i)$ is computed from $\mathrm{Irr}(G_{i-1})$. In view of the efficiency the main point is that the BC-Algorithm does not construct a transversal $\mathrm{Irr}(G_i)$ arbitrarily but a very special set of representations with the following properties: firstly, the representations $F \in \mathrm{Irr}(G_i)$ are "optimally" adapted to the chief series $\mathcal{C}_i := (G_i \supset G_{i-1} \supset \ldots \supset G_0 = \{1\})$ of $G_i$ in the sense that the restriction $F{\downarrow}G_\ell$ of $F$ to any subgroup $G_\ell$, $1 \leq \ell < i$, is not only *equivalent* – as stated in Clifford's Theorem – but even *equal* to the direct sum of representations of $\mathrm{Irr}(G_\ell)$. Furthermore, equivalent irreducible constituents are equal. This property is referred to as $\mathcal{C}_i$*-adaptivity* of $F$ and ensures that all data already computed up to level $i-1$ can be used without modification for the construction in the following levels $i, \ldots, n$. A transversal of $\mathcal{C}_i$-adapted irreducible matrix representations of $\mathbb{C}G_i$ will be denoted by $\mathrm{Irr}(G_i, \mathcal{C}_i)$. Secondly, for the class of supersolvable groups every $\mathcal{C}_i$-adapted representation $F$ is automatically *monomial*, i.e., all matrices $F(g)$, $g \in G_i$, are monomial having exactly one non-zero entry in each column and row. (This gives an alternative proof of the fact that supersolvable groups are M-groups, see, e.g., [11], Section 8.5.) Thirdly, all non-zero matrix entries of $F(g)$ are $e$-th roots of unity, where $e$ denotes the exponent of $G$ (such matrices will be called *e-monomial*). Since all matrix manipulations of the BC-Algorithm are matrix multiplications and inversions the arithmetic operations can be done purely symbolically in the additive group $\mathbb{Z}_e$, i.e., thus the algorithm never runs into numerical problems. We summarize the result in the following theorem, whose proof can be found in [4].

**Theorem 2.2** *Let $G$ be a finite supersolvable group of order $N$ with exponent $e$ and chief series $\mathcal{C}$. Then the BC-Algorithm constructs from a consistent pc presentation of $G$ a transversal* $\mathrm{Irr}(G, \mathcal{C})$ *of e-monomial $\mathcal{C}$-adapted irreducible representations with $O(N \log N)$ operations in the additive group $\mathbb{Z}_e$.*

The output data of the BC-Algorithm can be realized by a compact data structure denoted by `DS(`$G$`)`. Each $e$-monomial matrix $A \in \mathbb{C}^{m \times m}$ can be written as

$$A = \alpha \cdot \mathrm{diag}(\omega^{a_1}, \ldots, \omega^{a_m}) \tag{7}$$

with a permutation $\alpha \in S_m$ and non-trivial coefficients $\omega^{a_1}, \ldots, \omega^{a_m}$, where $\omega := \exp(2\pi\sqrt{-1}/e)$. (Actually, the BC-Algorithm works with any primitive $e$-th root of unity $\omega$.) The permutation $\alpha \in S_m$ is interpreted as permutation

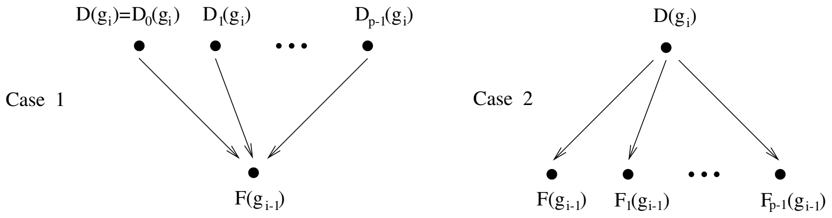

**Fig. 1.** Local structure of `DS(`$G$`)` around $D$, where $p := p_i$

matrix, whose $j$-th column has entry 1 at position $\alpha(j)$, $1 \le j \le m$. Therefore, $A$ can be described by the $2m$ positive integers $\alpha(1), \dots, \alpha(m)$ and $a_1, \dots, a_m$, which are all bounded by $N$. In view of the implementation we assume that these integers as well as all pointers can each be stored in a 32-bit word or `double`.

The BC-Algorithm computes $\mathrm{Irr}(G, \mathcal{C})$, where each representation $D \in \mathrm{Irr}(G, \mathcal{C})$ is given by its representing $e$-monomial matrices $D(g_1), \dots, D(g_n)$ on the generators of the pc presentation of $G$. Using Eq. (7) one can easily show that this leads to a memory requirement of $2N \log N$ `doubles`. Using the $\mathcal{C}$-adaptivity one can do even better. To this means we introduce the *$\mathcal{C}$-character graph* of $G$, which consists of $n+1$ levels. Each node of level $i$, $0 \le i \le n$, corresponds uniquely to a representation $D \in \mathrm{Irr}(G_i, \mathcal{C}_i)$ (which in turn corresponds to an irreducible character). Edges do exist at most between nodes of adjacent levels. More precisely, there is an edge between $D \in \mathrm{Irr}(G_i, \mathcal{C}_i)$ and $F \in \mathrm{Irr}(G_{i-1}, \mathcal{C}_{i-1})$ iff $D$ restricted to $G_{i-1}$ contains $F$ as a direct summand. (As an example, Figure 2 shows the character graph of $G_{128}$ given by Eq. (9).) In order to store $\mathrm{Irr}(G, \mathcal{C})$, it suffices to store the character graph of $G$, where we need a pointer for each edge, and for each node corresponding to $D \in \mathrm{Irr}(G_i, \mathcal{C}_i)$ we store the $e$-monomial matrix $D(g_i)$ given by $2\deg(D)$ integers. Based on Clifford's Theorem, Figure 1 shows the local structure of `DS(`$G$`)` around $D$.

In Case 1 it follows again by Clifford's Theorem (using the notation of Theorem 2.1) that $D_k(g_i) = \chi^k(g_i G_{i-1}) D_0(g_i)$ for $1 \le k < p$. Therefore it suffices to store the matrix $D_0(g_i)$ and, for $1 \le k < p$, just the integers $\chi^k(g_i G_{i-1})$ and a pointer to $D_0(g_i)$. A straightforward analysis of `DS(`$G$`)` leads to the following result (see [10]):

**Lemma 2.3** *The data structure* `DS(`$G$`)` *for a finite supersolvable group $G$ requires at most* $12\,N$ `doubles` *for all pointers and integers.*

## 3 $e$-Monomial Matrices

In this section, we summarize basic properties of $e$-monomial matrices which will be needed in the analysis of the WN-Algorithm. Furthermore, we prove a formula for the order of an $e$-monomial matrix which will be exploited as an important first step in the word normalization.

## 3.1 Basic Properties

As in (7), let $A = \alpha\,\mathrm{diag}(\omega^{a_1}, \dots, \omega^{a_N})$ and $B = \beta\,\mathrm{diag}(\omega^{b_1}, \dots, \omega^{b_N})$ denote $e$-monomial matrices of dimension $N \in \mathbb{N}$ with permutations $\alpha, \beta \in S_N$ and $\omega := \exp(2\pi\sqrt{-1}/N)$. Then it is easily checked that

$$\beta^{-1}\mathrm{diag}(a_1, \dots, a_N)\beta = \mathrm{diag}(a_{\beta(1)}, \dots, a_{\beta(N)}). \tag{8}$$

From this follows the next lemma which tells us how to compute with $e$-monomial matrices.

**Lemma 3.1** *Let $A$ and $B$ be e-monomial matrices as above and let $A_m := (\alpha(m), a_m)$ for $1 \le m \le N$. Then the following holds:*

$$\begin{aligned}
(A^{-1})_m &= (\alpha^{-1}(m),\ (-a_{(\alpha^{-1}(m))}) \bmod e),\\
(AB)_m &= (\alpha(\beta(m)),\ (a_{\beta(m)} + b_m) \bmod e),\\
(A^r)_m &= (\alpha^r(m),\ (a_{\alpha^{r-1}(m)} + \dots + a_{\alpha(m)} + a_m) \bmod e).
\end{aligned}$$

The formula for the multiplication of two matrices can easily be generalized to $L$ matrices, $L \in \mathbb{N}$. Conducting the multiplication from right to left, one obtains the following lemma.

**Lemma 3.2** *Let $A_1, \dots, A_L \in \mathrm{GL}(N, \mathbb{C})$ be e-monomial matrices and let $m \in \{1, \dots, N\}$. Then $(A_1 \cdot \ldots \cdot A_{L-1} \cdot A_L)_m$ can be computed with $L - 1$ additions in $\mathbb{Z}_e$.*

The next lemma tells us, how to compute $(A^r)_m$ efficiently for $r \gg N$ using the binary method, where only additions (and no multiplications) in $\mathbb{Z}_e$ are used.

**Lemma 3.3** *Let $A = \alpha\,\mathrm{diag}(\omega^{a_1}, \dots, \omega^{a_N})$ and $r \in \mathbb{N}$. Assume that $m \in \{1, \dots, N\}$ lies in a cycle of $\alpha$ of length $M$, $1 \le M \le N$. Then $(A^r)_m$ can be computed with at most $r - 1$ or, if $r \ge M$, with at most $2\lfloor\log(r \text{ div } M)\rfloor + M$ additions in $\mathbb{Z}_e$.*

*Proof.* The upper bound $r - 1$ follows from Lemma 3.2. If $r \ge M$ write $r = s \cdot M + t$ with $0 \le t < M$. Let $\sigma_i := a_{\alpha^i(m)}$, $\Sigma_k := \sum_{i=0}^{k} \sigma_i$, and $\Sigma_{-1} := 0$. As $\alpha^M(m) = m$, we obtain $\sigma_i = \sigma_{i+M}$ and thus

$$(A^r)_m = (\alpha^r(m), \Sigma_{r-1} \bmod e) = (\alpha^t(m), \Sigma_{t-1} + s \cdot \Sigma_{M-1} \bmod e).$$

Hence $\Sigma_{r-1} \bmod e$ can be computed with $M + 2\lfloor \log s \rfloor$ additions in $\mathbb{Z}_e$ as follows.

(i) Compute $\Sigma_{t-1} \bmod e$ with $t-1$ additions in $\mathbb{Z}_e$.
(ii) Compute $\Sigma_{M-1} = \Sigma_{t-1} + (\Sigma_{M-1} - \Sigma_{t-1}) \bmod e$ with $M-t$ additions in $\mathbb{Z}_e$.
(iii) Compute $s \cdot \Sigma_{M-1} \bmod e$ with the binary method with at most $2\lfloor \log s \rfloor$ additions in $\mathbb{Z}_e$.
(iv) Compute $\Sigma_{r-1} \bmod e = (\Sigma_{t-1} \bmod e) + (s \cdot \Sigma_{M-1} \bmod e) \bmod e$ with one addition in $\mathbb{Z}_e$.

From this follows the other upper bound. In (iii) the product $s \cdot \Sigma_{M-1} \bmod e$ in $\mathbb{Z}_e$ was computed by additions in $\mathbb{Z}_e$ to avoid multiplications in $\mathbb{Z}_e$. □

## *3.2 The Order of Generators and e-Monomial Matrices*

Given a consistent pc presentation, one might ask for the orders $\mathrm{ord}(g_i)$ of the generators $g_i$. (By the way, we need this information in the next section to obtain faster algorithms.) As $g_i \in G_i \setminus G_{i-1}$, $p_i$ divides the order of $g_i$. More precisely, we have, $\mathrm{ord}(g_i) = p_i \cdot \mathrm{ord}(g_i^{p_i})$. Next we could look at $g_i^{p_i} \in G_{i-1}$ which is known in normal form, $g_i^{p_i} = u_i$. From this normal form we can easily compute the smallest $j$ such that $g_i^{p_i} \in G_j$. If $j = 0$, then $u_i = 1$ and we are done. Otherwise, we run into a word normalization problem, for we have to compute the canonical form of $u_i^{p_j}$ in order to obtain $\mathrm{ord}(g_i) = p_i \cdot p_j \cdot \mathrm{ord}(u_i^{p_j})$, and so on.

With the help of our DFT-data, the computation of the orders of the generators becomes quite easy. From the BC-algorithm we know, for each $i$, the matrices $D_{i,0}(g_i), \ldots, D_{i,h_i-1}(g_i)$. As the DFT for $G_i$ is an isomorphism, the order of $g_i$ equals the order of the block diagonal matrix $D_{i,0}(g_i) \oplus \ldots \oplus D_{i,h_i-1}(g_i)$, which in turn is the lowest common multiple of the orders of the $e$-monomial matrices $D_{i,s}(g_i)$, $s \in [0 : h_i - 1]$. So to solve our problem, we need to determine the orders of $e$-monomial matrices.

We are going to compute the order of the $e$-monomial matrix $A = \alpha\, \mathrm{diag}(\epsilon_1, \ldots, \epsilon_N)$ setting $\epsilon_j := \omega^{a_j}$ for $1 \le j \le N$. Suppose $\alpha$ consists of $k$ cycles of length $\ell_1, \ldots, \ell_k$ and let $C_\kappa$ denote the set of all elements in the $\kappa$-th cycle of $\alpha$. E.g., if $\alpha = (a \ldots b)(c \ldots d) \ldots (y \ldots z)$ then $C_1 = \{a, \ldots, b\}$, $C_2 = \{c, \ldots, d\}$ and $C_k = \{y, \ldots, z\}$. It is well-known that

$$\mathrm{ord}(\pi) = \mathrm{lcm}(\ell_1, \ldots, \ell_k).$$

For $\kappa \in [1 : k]$ let

$$\lambda_\kappa := \left( \prod_{j \in C_\kappa} \epsilon_j \right)^{\mathrm{ord}(\pi)/\ell_\kappa} \qquad \text{and} \quad m_\kappa := \left( \sum_{j \in C_\kappa} a_j \right) \cdot (\mathrm{ord}(\pi)/\ell_\kappa) \bmod e.$$

**Lemma 3.4** *Keeping the above notation, the following holds:*

(1) $\operatorname{ord}(A) = \operatorname{ord}(\alpha) \cdot \operatorname{lcm}(\operatorname{ord}(\lambda_1), \dots, \operatorname{ord}(\lambda_k))$.
(2) $\operatorname{ord}(\lambda_\kappa) = e/\gcd(e, m_\kappa)$, *for each* $\kappa \in [1 : k]$.

*Proof.* Our claim follows immediately from the well-known fact that for a group element $g$ of order $m$ and a positive integer $r$ the order of $g^r$ equals $m/\gcd(m, r)$. This fact should be combined with the formula

$$A^r = \alpha^r \operatorname{diag}\left(\prod_{\rho=1}^{r-1} \epsilon_{\alpha^{\rho-1}(1)}, \dots, \prod_{\rho=1}^{r-1} \epsilon_{\alpha^{\rho-1}(N)}\right)$$

whose validity follows from Lemma 3.1. □

## 4 DFT-based Word Normalization

In this section, we describe a DFT-based strategy for the word normalization problem in a pc presentated finite supersolvable group $G$ of exponent $e$, whose presentation defines a chief series $\mathcal{C}$. We fix the primitive $e$-th root of unity $\omega := \exp(2\pi\sqrt{-1}/e)$. Using the notation from Section 2, we are going to normalize $g_\varphi^\psi = g_{\varphi(1)}^{\psi(1)} \cdot \dots \cdot g_{\varphi(L)}^{\psi(L)}$, see (5). There are two trivial preprocessing steps: first of all, by summing up exponents one can assume $\varphi(\ell) \neq \varphi(\ell + 1)$ for $1 \leq \ell < L$. Secondly, according to the last section, we know the orders of the generators. Thus we can replace each $\psi(\ell)$ by $(\psi(\ell) \bmod \operatorname{ord}(g_{\varphi(\ell)}))$, for $1 \leq \ell \leq L$.

Suppose $g_\varphi^\psi$ has normal form $g_n^{e_n} \cdot \dots \cdot g_1^{e_1}$. Our task is to compute the exponents $e_n, \dots, e_1$. A general strategy could proceed as follows. To compute $e_n$ switch to the factor group $G/G_{n-1}$. Then

$$g_n^{\sum_{\ell:\varphi(\ell)=n} \psi(\ell)} G_{n-1} = g_\varphi^\psi G_{n-1} = g_n^{e_n} G_{n-1},$$

hence $e_n = \sum_{\ell:\varphi(\ell)=n} \psi(\ell) \bmod p_n$. Next consider $w_{n-1} := g_n^{-e_n} g_\varphi^\psi$ which is equal to $g_{n-1}^{e_{n-1}} \cdot \dots \cdot g_1^{e_1} \in G_{n-1}$. Although $w_{n-1}$ is an element of $G_{n-1}$, it is given as a word in possibly all generators. Taken this into account, we switch to $G/G_{n-2}$ and have to extract from the equation $w_{n-1}G_{n-2} = g_{n-1}^{e_{n-1}} G_{n-2}$ the exponent $e_{n-1}$. Inductively, suppose we already know the exponents $e_n, \dots, e_{i+1}$, for some $i \geq 1$. Then define $w_i := g_{i+1}^{-e_{i+1}} \cdot \dots \cdot g_n^{-e_n} g_\varphi^\psi = g_i^{e_i} \cdot \dots \cdot g_1^{e_1} \in G_i$. The task is to extract from $w_i G_{i-1} = g_i^{e_i} G_{i-1}$ the exponent $e_i$. The goal of this section is to show, how this extraction can be done efficiently using partial information about the $\mathcal{C}$-adapted DFT of $G$.

The BC-Algorithm enumerates the $\mathcal{C}_i$-adapted irreducible representations of $G_i$ in a specific way. For each $i$, $D_{i,0}$ is the trivial representation, whereas $D_{i,1}$ is the one-dimensional representation with kernel $G_{i-1}$ specified by $D_{i,1}(g_i) =$

$\varrho_i := \exp(2\pi\sqrt{-1}/p_i)$. For each $i$ let $D_{n,\nu_i}$ be the first irreducible representation of $G$ with the property that its restriction to $G_i$ contains $D_{i,1}$. From Clifford Theory (see, e.g., [7]) we know that in this case $D_{n,\nu_i}\downarrow G_i$ is a direct sum of one-dimensional representations, among them also $D_{i,1}$. Let $\mu_i$ be the minimal $m$ such that the $m$-th constituent of $D_{n,\nu_i}\downarrow G_i$ equals $D_{i,1}$. We remark that both $\nu_i$ and $\mu_i$ can easily be extracted from `DS(`$G$`)`.

Now we are prepared to present our DFT-based normalization procedure. In a preprocessing phase, the data structure `DS(`$G$`)` as well as the orders of the generators for a pc presented finite supersolvable group have been precomputed.

**WN-Algorithm: Word Normalization via DFT**
**Input:** $(\varphi, \psi)$ /* this describes the word $g_\varphi^\psi = g_{\varphi(1)}^{\psi(1)} \cdot \ldots \cdot g_{\varphi(L)}^{\psi(L)}$.
**Output:** $(e_1, \ldots, e_n)$ /* $g_n^{e_n} \cdot \ldots \cdot g_1^{e_1}$ is the normal form of $g_\varphi^\psi$ */

```
(1) $e_n := \sum_{\ell:\varphi(\ell)=n} \psi(\ell) \bmod p_n$
(2) for $i = n - 1$ downto 1 do
(3)     Compute the $\mu_i$-th entry $\varrho_i^{x_i}$ of the diagonal matrix
        $D(w_i) = D(g_{i+1}^{-1})^{e_{i+1}} \cdot \ldots \cdot D(g_n^{-1})^{e_n} \cdot D(g_{\varphi(1)})^{\psi(1)} \cdot \ldots \cdot D(g_{\varphi(L)})^{\psi(L)}$,
        where $D := D_{n,\nu_i}$.
(4) $e_i := x_i \bmod p_i$
(5) endfor
```

**Theorem 4.1** *The WN-algorithm is correct.*

*Proof.* We have already seen that line (1) correctly computes $e_n$. Now let $i \in [1 : n-1]$ and suppose that $e_n, \ldots, e_{i+1}$ have already been computed correctly by the WN-algorithm.

By definition of $\nu_i$ and $\mu_i$, line (3) computes $D_{i,1}(w_i) = D_{i,1}(g_i)^{e_i} \cdot \ldots \cdot D_{i,1}(g_1)^{e_1} = D_{i,1}(g_i)^{e_i} = \varrho_i^{e_i}$, since all $g_j$, $j < i$, are in the kernel of $D_{i,1}$. □

*Example 4.2* We illustrate the WN-Algorithm by means of the group $G_{128}$ of order 128 given by the following pc presentation (trivial commutator relations are omitted):

$$\begin{aligned} G_{128} = \langle g_7, g_6, g_5, g_4, g_3, g_2, g_1 | g_1^2 = g_2^2 = g_4^2 = g_5^2 = g_6^2 = 1, g_3^2 = g_1, g_7^2 = g_4, \\ [g_2, g_6] = [g_2, g_7] = [g_3, g_4] = [g_3, g_5] = [g_3, g_6] = g_1, [g_3, g_7] = g_2, \\ [g_4, g_5] = g_2 \cdot g_1, [g_4, g_6] = g_3 \cdot g_1, [g_5, g_7] = g_3, [g_6, g_7] = g_5 \rangle \end{aligned} \tag{9}$$

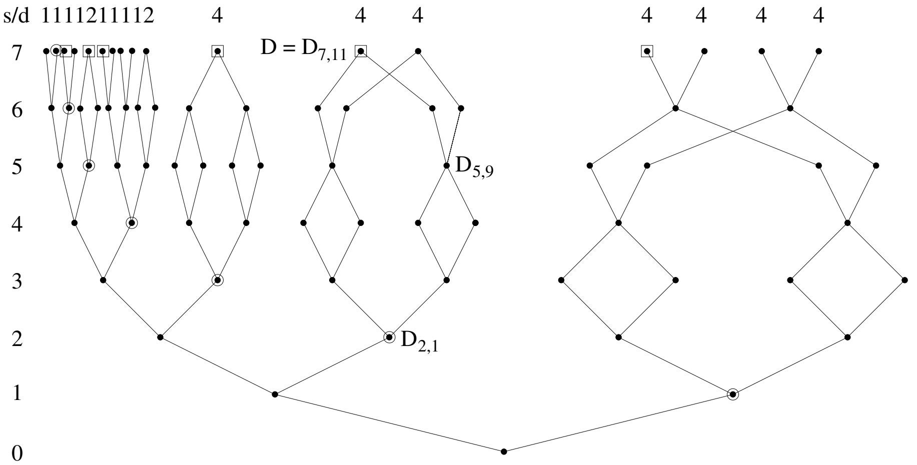

**Fig. 2.** The character graph of $G_{128}$

The character graph of $G_{128}$, which can be regarded as the "skeleton" of the data structure $\mathtt{DS}(G)$, is shown in Figure 2. The representations $D_{i,1}$, $i \in [1:7]$, are marked by a circle, and the corresponding representations $D_{7,\nu_i}$ of $G = G_7$ are marked by a square. For example, for $i = 2$ we obtain $\nu_2 = 11$, and the restriction of $D := D_{7,11}$ to $G_2$ equals the 4-fold direct sum of $D_{2,1}$. Hence, in this case, $\mu_2 = 1$. Each non-trivial entry of $D(w_2)$ equals $D_{2,1}(w_2)$.

In the analysis of the WN-Algorithm we need the following entity

$$C(\varphi,\psi) := L + \sum_{\ell=1}^{L} \lfloor \log(\psi(\ell)) \rfloor. \tag{10}$$

$C(\varphi,\psi)$ describes roughly the binary description lengths of all exponents of the word $g_\varphi^\psi$. Then one has the following upper complexity bounds for the WN-Algorithm.

**Theorem 4.3** *Let $G$ be a finite pc presented supersolvable group with chief series $\mathcal{C}$ of length $n$, exponent $e$ and maximal prime factor $p_{\max}$. Furthermore, let (suitable parts) of $\mathtt{DS}(G)$ and all inverse matrices be given (computed in a preprocessing step by the BC-Algorithm and stored with ($O(|G|)$ memory requirement). Let $d_{\max} := \max_{D\in \mathrm{Irr}(G,\mathcal{C})}(\deg(D))$ and let $A(\varphi,\psi)$ denote the number of additions in $\mathbb{Z}_e$ needed by the WN-Algorithm to compute the normal form of a word $w = g_\varphi^\psi$. Then one has the following worst case upper complexity bounds:*

$$A(\varphi,\psi) \le 2n \cdot \sum_{\ell=1}^{L} |\psi(\ell)| + n^2(p_{\max}-1) \tag{11}$$

*and the in general better bound*

$$A(\varphi, \psi) \leq 2n \cdot C(\varphi, \psi) + n^2 \cdot (\log(p_{\max}) + \min(p_{\max}, d_{\max}) + 1)$$
$$+2n \cdot \sum_{\ell=1}^{L} \min(|\psi(\ell)|, d_{\max}). \tag{12}$$

*Proof.* In the proof of the upper bound (11) the binary method is not applied, whereas in the proof the upper bound (12) the binary method is used. Furthermore, we only count additions in $\mathbb{Z}_e$ and assume that table look-ups are free of cost. (Actually, the number of table look-ups is of the same order as the number of additions in $\mathbb{Z}_e$.) Using the notation as in the description of the WN-Algorithm, in the $i$-th step, $1 \leq i \leq n$, we have to compute $m$-th entry of the product

$$D(w_i) = D(g_{i+1}^{-1})^{e_{i+1}} \cdot \ldots \cdot D(g_n^{-1})^{e_n} \cdot D(g_{\varphi(1)})^{\psi(1)} \cdot \ldots \cdot D(g_{\varphi(L)})^{\psi(L)}, \tag{13}$$

where $m := \mu_i$ and $D := D_{n,\nu_i}$. Only the data from $\texttt{DS}(G)$ and the inverse matrices are available.

- Since $D \downarrow G_{i-1}$ is trivial, the factors of the form $D(g_j)^r$, $r \in \mathbb{Z}$, $1 \leq j < i$, are free of charge.
- The matrices $D(g_j)$, $i \leq j < n$, are not given explicitly in $\texttt{DS}(G)$. $D(g_j)$ is a block-diagonal matrix with blocks $F(g_j)$ for suitable representations $F \in \mathrm{Irr}(G_j, \mathcal{C}_j)$ of same degree $\deg(F)$. In $\texttt{DS}(G)$ the matrices $F(g_j)$ are given up to a constant factor $c \in \mathbb{C}$ (which is an $e$-th root of unity). The computation of a non-trivial entry of a factor of the form $D(g_j)^r$, $r \in \mathbb{N}$, $i \leq j < n$, can be reduced without additional cost to computing a non-trivial entry of $F_k(g_j)^r$ for some suitable $F_k$ among the representations $F$ of the block structure. (This $F_k$ can be read off the character graph which is part of $\texttt{DS}(G)$ without cost.) In the data structure $\texttt{DS}(G)$ the representation $F_k$ is given up to a constant $c$, which is a primitive $e$-th root of unity. ($F_k = c \cdot F_0$; see the discussion before Lemma 2.3.) Considering that this factor $c$ leads to an additional cost of $\min(r-1, M)$ additions in $\mathbb{Z}_e$, it follows from Lemma 3.3 that a non-trivial entry of $F_k(g_j)^r$ (and, therefore, also one of $D(g_j)^r$) can be computed with
$$\min\left(2r-1,\ 2\lfloor\log(r \text{ div } M)\rfloor + 2M\right) \tag{14}$$
additions in $\mathbb{Z}_e$ with some $M \leq \deg(F)$. Therefore, the number of necessary additions in $\mathbb{Z}_e$ is also bounded by
$$2\lfloor\log(r \text{ div } M)\rfloor + \min(2r-1, 2M). \tag{15}$$
- For factors of the form $D(g_j)^r$ with negative $r \in \mathbb{Z}_{<0}$, we use the inverse matrices computed in the preprocessing step and proceed as above.

Now, we prove upper bound (11) for $A(\varphi, \psi)$. Let $A(i)$ denote the number of additions in $\mathbb{Z}_e$ needed to compute the $m$-th entry of $D(w_i)$ in the $i$-th step. The product in (13) has $n - i + L$ factors. By Lemma 3.2 and the bound $2r - 1$ in (14) one obtains

$$A(i) \leq n - i + L - 1 + \sum_{\ell=i+1}^{n} (2e_\ell - 1) + \sum_{\ell=1}^{L} (2|\psi(\ell)| - 1).$$

Setting $p_{\max} := \max_{1 \leq i \leq n}(p_i)$ and noting that $e_\ell \leq p_{\max} - 1$, one obtains an upper bound for $A(\varphi, \psi)$ by summing over the steps $i = 1, \dots, n$:

$$\begin{aligned} A(\varphi, \psi) &= \sum_{i=1}^{n} A(i) \\ &\leq \frac{n(n-1)}{2} + n \cdot L - n + \frac{n(n-1)}{2}(2(p_{\max} - 1) - 1) \\ &\quad + n \cdot \sum_{\ell=1}^{L} (2|\psi(\ell)| - 1) \\ &= n(n-1)(p_{\max} - 1) - n + 2n \cdot \sum_{\ell=1}^{L} |\psi(\ell)| \\ &\leq 2n \cdot \sum_{\ell=1}^{L} |\psi(\ell)| + n^2(p_{\max} - 1). \end{aligned}$$

Now, we prove the upper bound (12) for $A(\varphi, \psi)$. Noting that the cycle length $M$ is bounded by $d_{\max} := \max_{D \in \mathrm{Irr}(G,\mathcal{C})}(\deg(D))$ and $\log(|r| \operatorname{div} M)$ is bounded by $\log |r|$, we obtain from Lemma 3.2 and (15)

$$\begin{aligned} A(i) \leq n - i + L - 1 + &\sum_{\ell=i+1}^{n} \big(2\lfloor \log e_\ell \rfloor + \min(2e_\ell - 1, 2d_{\max})\big) \\ + &\sum_{\ell=1}^{L} \big(2\lfloor \log |\psi(\ell)| \rfloor + \min(2|\psi(\ell)| - 1, 2d_{\max})\big). \end{aligned}$$

Using $e_\ell \leq p_{\max} - 1$ and $\sum_{\ell=1}^{L} \lfloor \log(|\psi(\ell)|) \rfloor = C(\varphi, \psi) - L$, which follows from the definition (10) of $C(\varphi, \psi)$, we obtain

$$\begin{aligned} A(i) &\leq n - i + L - 1 + \sum_{\ell=i+1}^{n} \Big(2\lfloor \log(p_{\max} - 1) \rfloor + \min(2p_{\max} - 3, 2d_{\max})\Big) \\ &\quad + 2 \cdot C(\varphi, \psi) - 2 \cdot L + \sum_{\ell=1}^{L} \big(\min(2|\psi(\ell)| - 1, 2d_{\max})\big) \\ &= n - i + L - 1 + 2 \cdot (n - i)\Big(\lfloor \log(p_{\max} - 1) \rfloor + \min(p_{\max} - 1, d_{\max})\Big) \\ &\quad + 2 \cdot C(\varphi, \psi) - 2 \cdot L + 2 \cdot \sum_{\ell=1}^{L} \big(\min(|\psi(\ell)|, d_{\max})\big) \end{aligned}$$

$$\leq 2 \cdot (n-i)\Big(\log(p_{\max}) + \min(p_{\max}, d_{\max}) + 1\Big) + 2 \cdot C(\varphi, \psi) - L + 2 \cdot \sum_{\ell=1}^{L} \min(|\psi(\ell)|, d_{\max}).$$

Altogether on gets the following bound for $A(\varphi, \psi)$:

$$\begin{aligned} A(\varphi, \psi) &= \sum_{i=1}^{n} A(i) \\ &\leq 2 \cdot \frac{n(n-1)}{2} \cdot \big(\log(p_{\max}) + \min(p_{\max}, d_{\max}) + 1\big) \\ &\quad + 2n \cdot C(\varphi, \psi) - n \cdot L + 2n \cdot \sum_{\ell=1}^{L} \min(|\psi(\ell)|, d_{\max}) \\ &\leq 2n \cdot C(\varphi, \psi) + n^2 \cdot \big(\log(p_{\max}) + \min(p_{\max}, d_{\max}) + 1\big) \\ &\quad + 2n \cdot \sum_{\ell=1}^{L} \min(|\psi(\ell)|, d_{\max}). \end{aligned}$$

□

For most groups these upper bounds are only very rough. For example, if the maximal prime factor $p_{\max}$ and the maximal degree $d_{\max}$ are small compared to $|G|$ und $C(\varphi, \psi)$, the running time is essentially $n \cdot C(\varphi, \psi)$ up to a constant factor. As special case of Theorem 4.3 we get Corollary 1.1 and the following result:

**Corollary 4.4** *Let $G$ be a pc presented $p$-group of order $p^n$. Under the assumptions of Theorem 4.3 the following holds:*

(1) *The normal form of the product of elements in $G$ given in normal form can be computed with $5 \cdot p \cdot n^2$ additions in $\mathbb{Z}_e$.*
(2) *The normal form of the inverse of an element in $G$ given in normal form can be computed with $3 \cdot p \cdot n^2$ additions in $\mathbb{Z}_e$.*

*Proof.* Let $a, b$ be words in normal form, then all exponents of $w = a \cdot b$ are bounded by $(p-1)$ and $L$ is bounded by $2 \cdot n$. Hence, claim (1) follows directly from the first upper bound of Theorem 4.3. Let $g = g_n^{e_n} \cdot \ldots \cdot g_1^{e_1}$ be given in normal form, then the inverse is given by $g^{-1} = g_1^{-e_1} \cdot \ldots \cdot g_n^{-e_n}$. Claim (2) follows also from the first upper bound of Theorem 4.3. □

There is still place for improvements and generalization of the WN-Algorithm. Note that not all irreducible representations of $\mathrm{Irr}(G, \mathcal{C})$ are actually needed, so that in the preprocessing step one can restrict on the computation of the necessary representations. Furthermore, the WN-Algorithm works in the same way for any regular expression in the generators $g_1, \ldots, g_n$ of $G$.

## 5 Implementation

The WN-Algorithm has been implemented in the programming language C/C++. Numerous tests have been conducted using an Intel Pentium III, 700 MHz with 128 MB RAM. In the following tables all running times are measured in milliseconds.

Table 1 illustrates the running time behaviour of the WN-Algorithm for fixed words but various group sizes $|G|$ with chief series $\mathcal{C}$ of different length $n$. We used as groups $m$-fold direct products of the symmetric group $S_3$ for increasing $m$. The forth column $t$(DFT) represents the running time of the preprocessing step computing the `DS(G)` by the BC-Algorithm. Columns five to seven show the running times $t(w(j))$ needed to normalize the words $w(j) = g_{\varphi(j)}^{\psi(j)}$, $j = 1, 2, 3$. The last row shows the entity $C(\varphi(j), \psi(j))$. The words $w(j)$ were generated by some random functions $\varphi$ and $\psi$, equally distributed in $[1 : n]$ and $[1 : 6]$ respectively. (Note that $e = 6$ is the exponent of all groups $G = (S_3)^m$).)

Looking at the columns of the table, one recognizes that the running times are roughly linear in $n$. From the rows of the table follows that the running times are also about linear in $C(\varphi, \psi)$. In other words, the running time behaviour of the implementation of WN-Algorithm can be roughly decribed by $n \cdot C(\varphi, \psi)$. This corresponds to the second upper bound of Theorem 4.3 since the remaining summands are neglectable for small $p_{\max}$ and $d_{\max}$ and independent of $C(\varphi, \psi)$.

To test the implementation on more complex groups, we have generated a library of several thousands of consistent pc presentations defining supersolvable groups of various orders up to $10^7$ using the computer algebra system GAP [6]. In Table 2, $\mathrm{Lib}_N(k)$ denotes the $k$-th pc presentation of our library defining a group $G$ of order $N$ and $\mathrm{Syl}_2(S_{16})$ is a 2-Sylow group of the symmetric group $S_{16}$. Again one recognizes that the running time of the WN-Algorithm does not depend on the group order $|G|$, but on the length $n$ of the chief series corresponding to the hight of the character graph. For example, the running times for the group $G = \mathrm{Syl}_2(S_{16})$ ($n = 15$) are about three times the ones of the group

**Table 1.** Running times depending on $n$

| $G$ | $\lvert G\rvert$ | $n$ | $t$(DFT) | $t(w(1))$ | $t(w(2))$ | $t(w(3))$ |
|---|---|---|---|---|---|---|
| $(S_3)$ | 6 | 2 | $< 1$ | 17 | 133 | 951 |
| $(S_3)^2$ | 36 | 4 | $< 1$ | 30 | 218 | 1808 |
| $(S_3)^3$ | 216 | 6 | $< 1$ | 55 | 301 | 2846 |
| $(S_3)^4$ | 1296 | 8 | 1 | 56 | 436 | 4040 |
| $(S_3)^5$ | 7776 | 10 | 17 | 92 | 558 | 5454 |
| $(S_3)^6$ | 46656 | 12 | 82 | 103 | 692 | 6897 |
| $(S_3)^7$ | 279936 | 14 | 281 | 128 | 856 | 8499 |
| $(S_3)^8$ | 679616 | 16 | 1370 | 100 | 1035 | 10283 |
| $(S_3)^9$ | 10077696 | 18 | 5124 | 120 | 1225 | 12183 |
| $(S_3)^{10}$ | 60466176 | 20 | 22186 | 252 | 1464 | 14386 |
| $C(\varphi, \psi)$ | | | | $2.2 \cdot 10^4$ | $2.2 \cdot 10^5$ | $2.2 \cdot 10^6$ |

**Table 2.** Running times for different groups

| $G$ | $\|G\|$ | $n$ | $t(\mathrm{DFT})$ | $t(w(1))$ | $t(w(2))$ | $t(w(3))$ | $t(w(4))$ | $t(w(5))$ |
|---|---|---|---|---|---|---|---|---|
| $\mathrm{Lib}_{1024}(1)$ | 1024 | 10 | 4 | 7 | 76 | 481 | 4670 | 46690 |
| $\mathrm{Lib}_{2187}(1)$ | 2187 | 7 | 2 | 2 | 46 | 320 | 3015 | 29065 |
| $\mathrm{Lib}_{7560}(1)$ | 7560 | 8 | 119 | 4 | 50 | 371 | 3810 | 36336 |
| $\mathrm{Lib}_{7560}(54)$ | 7560 | 8 | 6 | 3 | 54 | 373 | 3439 | 34420 |
| $\mathrm{Lib}_{7776}(1)$ | 7776 | 10 | 21 | 8 | 72 | 470 | 4609 | 45380 |
| $\mathrm{Lib}_{15625}(35)$ | 15625 | 6 | 6 | 2 | 31 | 274 | 2470 | 23411 |
| $\mathrm{Lib}_{16000}(1)$ | 16000 | 10 | 77 | 6 | 69 | 472 | 4521 | 45229 |
| $\mathrm{Lib}_{22287}(1)$ | 22287 | 4 | 96 | 1 | 25 | 196 | 1695 | 15905 |
| $\mathrm{Lib}_{23940}(1)$ | 23940 | 7 | 113 | 5 | 41 | 317 | 2975 | 28340 |
| $\mathrm{Syl}_2(S_{16})$ | 32768 | 15 | 146 | 12 | 120 | 830 | 8299 | 82611 |
| $\mathrm{Lib}_{42875}(1)$ | 42875 | 6 | 55 | 3 | 38 | 274 | 2570 | 24335 |
| $\mathrm{Lib}_{179894}(3)$ | 179894 | 5 | 260 | 1 | 31 | 237 | 2185 | 20380 |
| $C(\varphi, \psi)$ | | | | $10^3$ | $10^4$ | $10^5$ | $10^6$ | $10^7$ |

$G = \mathrm{Lib}_{42875}(1)$ ($n = 6$) and two times the ones of $G = \mathrm{Lib}_{7560}(1)$ ($n = 8$). The words being used in these examples have $\psi(\ell) = 1$ for all $1 \leq \ell \leq L(w)$, i.e., $C(\varphi, \psi) = L$.

We compared the running times of the WN-Algorithm with the running times of a GAP-routine which computes the word normalization using the collection strategies described in [12]. See also [8] or [15] for further details. As for the WN-Algorithm the running time of the GAP-routine was also about linear in the complexity $C(\varphi, \psi)$ and did not essentially depend on the group size $|G|$. However, there was one striking qualitative difference in the running time behaviour: in contrast to the WN-Algorithm the running time of the GAP-routine did not depend on $n$ but on the complexity of the pc presentation defining the group $G$, i.e., the number and sizes of the non-trivial coefficients $a_{i,\ell}$ und $b_{ij,\ell}$ in the pc presentation (see (3) and (4)). These experimental results reflect that our DFT-based strategy for the word normalization and conventional collecting strategies are based on different principles. However, recall that the WN-Algorithm needs some preprocessing step to compute the irreducibe representations and memory capacity linear in the group order $|G|$ to store `DS(`$G$`)`.

## 6 Multivariate Polynomial Division

For a finite pc presented group $G$ the group algebra $\mathbb{C}G$ can be identified with the quotient of a non-commutative polynomial ring modulo a suitable ideal $I_G$. Choosing a certain monomial ordering, the pc presentation of $G$ defines a reduced Göbner basis of $I_G$. In this case, polynomial division modulo $I_G$ can be done via word normlization by means of the WN-Algorithm. In the following we go into some more detail.

The foundations of the theory of non-commutative Gröbner bases was already laid down by Bergman [2]. For a detailed treatment of this theory we refer to [9]. Let $\mathbb{C}\langle X\rangle = \mathbb{C}\langle X_n, \ldots, X_1\rangle$ denote the *non-commutative polynomial ring* over $\mathbb{C}$ in the variables $X := \{X_n, \ldots, X_1\}$. In the following, an ideal $I \subset \mathbb{C}\langle X\rangle$ is always understood to be two-sided. Let $G$ be a finite supersolvable group with a pc presentation as in (3). Then $X_i \mapsto g_i$, $i = 1, \ldots, n$, extends to a surjective homomorphism $\varphi : \mathbb{C}\langle X_n, \ldots, X_1\rangle \to \mathbb{C}G$ with kernel $I_G := \ker(\varphi)$. The pc relations of $G$ define polynomials in $I_G$ by

$$\begin{aligned} f_i &:= X_i^{p_i} - X_{i-1}^{a_{i,i-1}} \cdot \ldots \cdot X_1^{a_{i,1}}, && 1 \le i \le n,\\ f_{ij} &:= X_i X_j - X_j X_i X_i^{b_{ij,i}} \cdot \ldots \cdot X_1^{b_{ij,1}}, && 1 \le i < j \le n. \end{aligned}$$

We fix the so called *basic wreath-product ordering* denoted by $\succ_{\mathrm{w}}$ as monomial ordering on $\mathbb{C}\langle X\rangle$ (see [12] for a definition). Then the leading terms of $f_i$ and $f_{ij}$ with respect to $\succ_{\mathrm{w}}$ are given by $\mathrm{LT}(f_i) = X_i^{p_i}$ and $\mathrm{LT}(f_{ij}) = X_i X_j$ and the following theorem can be proved easily:

**Theorem 6.1** *The set* $\mathcal{G}_{\mathrm{w}} := \{f_i, 1 \le i \le n\} \cup \{f_{ij}, 1 \le i < j \le n\}$ *is the reduced Gröbner basis of* $I_G \subset \mathbb{C}\langle X_n, \ldots, X_1\rangle$ *with respect to the basic wreath-product ordering* $\succ_{\mathrm{w}}$.

The *reduction process* or *non-commutative multivariate polynomial division* is analogous to the commutative case. For example, let $f \in \mathbb{C}\langle X\rangle$ and w.l.o.g. let the leading coefficient of $f$ be one. Denote the leading term of $f$ by $\mathrm{LT}(f)$, then $f = \mathrm{LT}(f) + f'$ with suitable $f' \in \mathbb{C}\langle X\rangle$. A monomial $m$ contains $\mathrm{LT}(f)$ as subword, if $m = m_1 \mathrm{LT}(f) m_2$ for suitable monomials $m_1, m_2$. The *reduction* of $m$ w.r.t. $f$ is defined as $m_1(-f')m_2$. In this way, the reduction of an element in $\mathbb{C}\langle X\rangle$ w.r.t. to an arbitrary set of polynomials can be defined. As in the commutative case the reduction process w.r.t. a Gröbner basis $\mathcal{G}_{\mathrm{w}}$ leads to a uniquely determined remainder (see [9]). We denote this remainder by $\bar{f}^{\mathcal{G}_{\mathrm{w}}}$.

Let $\mathcal{G}_{\mathrm{w}}$ be the Gröbner basis of $I_G$ with monomial ordering $\succ_{\mathrm{w}}$ as in Theorem 6.1. In this special case, we suggest to compute the unique remainder $\bar{f}^{\mathcal{G}_{\mathrm{w}}}$ modulo $I_G$ of a given polynomial $f \in \mathbb{C}\langle X\rangle$ via word normalization instead of computing it via stepwise reduction.

**PD-Algorithm: Polynomial division via word normalization**
**Input:** Polynomial $f = \sum_{(\varphi,\psi)} c_{(\varphi,\psi)} X_\varphi^\psi \in \mathbb{C}\langle X\rangle$
**Output:** $\bar{f}^{\mathcal{G}_{\mathrm{w}}}$ /* reduced polynomial with respect to $\mathcal{G}_{\mathrm{w}}$*/

(1) **Normalization step:** Compute the normal form of all monomials $X_\varphi^\psi$ with $c_{(\varphi,\psi)} \neq 0$ by means of the WN-Algorithm. These computations can be done in parallel.
(2) **Summation step:** Sum up the coefficients $c_{(\varphi,\psi)} \neq 0$, whose monomials $X^\alpha$ are equivalent to the same normal form computed in (2).
(3) Define the polynomial $\bar{f}$, whose terms consist of the monomials of step (1) with the corresponding coefficients computed in (2). Then $\bar{f}^{\mathcal{G}_{\mathrm{w}}} = \bar{f}$.

**Theorem 6.2** *The PD-Algorithm is correct.*

*Proof.* $\bar{f}$ is congruent to $f$ modulo $I_G$ and $\bar{f}$ is not further reducible with respect to $\mathcal{G}_{\mathrm{w}}$. Therefore, one has $\bar{f} = \bar{f}^{\mathcal{G}_{\mathrm{w}}}$. □

In contrast to the standard reduction algorithm, which proceeds "horizontally", the PD-Algorithm proceeds "vertically". In particluar, the running time of the standard algorithm to compute $\bar{f}^{\mathcal{G}_{\mathrm{w}}}$ is in general linear in the multidegree of $f$, whereas the one of the PD-Algorithm is logarithmic in the multidegree. Hence, the PD-Algorithm is much faster in case the multidegrees are large.

For an arbitrary monomial ordering $\succ$ with Gröbner basis $\mathcal{G}$ of $I_G$ one can proceed as follows:

- For a given $f \in \mathbb{C}\langle X\rangle$ compute the normal form $\bar{f}^{\mathcal{G}_{\mathrm{w}}}$ with respect to the wreath product ordering via the PD-Algorithm. The total degree $|\alpha|$ of each monomial $X^\alpha$ of this polynomial is bounded by $|\alpha| = \alpha_1 + \cdots + \alpha_n < p_1 + \ldots + p_n$.
- Then, reduce $\bar{f}^{\mathcal{G}_{\mathrm{w}}}$ w.r.t. the Gröbner basis $\mathcal{G}$ by the standard algorithm to get the desired remainder $\bar{f}^{\mathcal{G}}$. This can be done fast since the degree of $\bar{f}^{\mathcal{G}_{\mathrm{w}}}$ is already small.

## References

1. Baum, U., Clausen, M.: Computing irreducible representations of supersolvable groups. Mathematics of Computation. Volume **63**(207), 351–359 (1994)
2. Bergman, G.: The Diamond Lemma for Ring Theory. Adv. Math. **29**, 178–218 (1978)
3. Bürgisser, P., Clausen, M., Shokrollahi, M.A.: Algebraic Complexity Theory. Grundlehren der mathematischen Wissenschaften, **315**, Springer Verlag, Berlin, 1997
4. Clausen, M., Baum, U.: Fast Fourier Transforms. BI-Wissenschaftsverlag, Mannheim, 1993
5. Clausen, M., Müller, M.: A fast program generator of FFTs. Proceedings AAECC-13, Honolulu, LNCS **1719**, 29–42 (1999)
6. The GAP Group, GAP – Groups, Algorithms and Programming. Version 4.3; 2002 (http://www.gap-system.org.)
7. Isaacs, I.: Character Theory of Finite Groups. Academic Press, Inc. 1976
8. Leedham-Green, C.R., Soicher, L.H.: Collection from the left and other strategies. J. Symbolic Computation **9**, 665–675 (1990)
9. Mora, F.: Gröbner Bases for Non-Commutative Polynomial Rings. Proceedings AAECC-3, Grenoble, LNCS **229**, 353–362 (1985)
10. Müller, M.: Beiträge zur Algorithmik verallgemeinerter diskreter Fouriertransformationen. PhD thesis, Universität Bonn, Institut für Informatik, 2001
11. Serre, J.-P.: Linear Representations of Finite Groups. Graduate Texts in Mathematics, Springer-Verlag, 1986
12. Sims, C.C.: Computation with finitely presented groups. Cambridge University Press, 1994

13. Sims, C.C.: Fast multiplication and growth in groups. ISSAC'98, Rostock, Germany, 1998, pp. 165–170
14. Thümmel, A.: Computing character tables of $p$-groups. Proceedings ISSAC'96, Zürich, Switzerland, 1996, pp. 150–154
15. Vaughan-Lee, M.R.: Collection from the left. J. Symbolic Computation **9**, 725–733 (1990)