An Efficient Computer Program for the Symbolic Construction of Irreducible Representations of Supersolvable Groups

Meinard Müller Michael Clausen

04.06.2001

Abstract

Baum and Clausen have presented in [1] an algorithm, refered to as BC-Algorithm, for the construction of a complete list of pairwise inequivalent ordinary irreducible representations for a given finite supersolvable group G. This algorithm is almost optimal in the sense that the running time is proportional to the length of the output up to some logarithmic factors. In this paper we describe an implementation, which realizes this algorithm for the first time efficiently concerning the running time as well as the memory requirements. Extensive tests have shown that the actual running time behaviour of the computer program reflects very well the theoretical complexity bounds. Furthermore, all arithmetic operations of the BC-Algorithm are done symbolically over $\mathbb{Z}/\mathbb{Z}_e$, where e denotes the exponent of G.

Introduction

Representation theory is not only one of the most powerful tools for the investigation of structural properties of groups but has also been applied to real world applications in fields such as signal processing, coding theory, quantum mechanics and crystallography among others. Therefore, there is an increased interest in not only investigating representations theoretically but also to construct them explicitly, e.g., in form of matrix representations. In computational group theory, there has been much work devoted to the design of efficient algorithms, which have been implemented in various computer algebra systems such as `GAP` and MAGMA. In this paper we describe a computer program, which realizes the BC-Algorithm for the first time constructing a *transversal* of irreducible representation (i.e., a representative of each isomorphism class) for the special class of finite supersolvable groups.

The rest of the paper is organized as follows. In Section 1 we introduce the notation and sketch the main ideas of the BC-Algorithm. For the implementation a memory efficient data structure has been designed, which will be described in Section 2. The computer program has been tested and analysed extensively for several thousand groups. We will summarize and discuss the results in Section 3.

1 BC-Algorithm

Let G denote a group of finite order $N := |G|$. By Wedderburn's structure theorem, the complex group algebra $\mathbb{C}G := \{a|a : G \to \mathbb{C}\}$ of a finite group G, with convolution $(ab)(x) := \sum_{g\in G} a(g)b(g^{-1}x)$ for $a, b \in \mathbb{C}G$ and all $x \in G$, is isomorphic to an algebra of block diagonal matrices, $\mathbf{D} = \oplus_{k=1}^{h} D_k : \mathbb{C}G \longrightarrow \oplus_{k=1}^{h} \mathbb{C}^{d_k\times d_k}$. Here, the number h of blocks equals the number of conjugacy classes of G and the projections $D_1, \ldots, D_h$ form a transversal of irreducible matrix representations of $\mathbb{C}G$, also denoted by $\mathrm{Irr}(G)$. Every such $\mathbf{D}$ is called a *discrete Fourier transform* (DFT) of G. One can show, that $\mathbf{D}$ is determined up to the ordering of the h blocks and the choice of basis in each of these blocks. For the class of supersolvable groups the BC-Algorithm constructs such a transversal $\mathrm{Irr}(G)$, or equivalently a DFT $\mathbf{D}$. A finite supersolvable group G has by definition a chief series

$$\mathcal{T} = (G = G_n \supset G_{n-1} \supset \ldots \supset G_1 \supset G_0 = \{1\}), \qquad G_i \lhd G,$$

with chief factors G_i/G_{i-1} of prime order $p_i := [G_i : G_{i-1}]$. The input of the BC-Algorithm is a consistent *power-commutator presentation* (pc-presentation) corresponding to $\mathcal{T}$:

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

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. Analogous to G, define $\mathrm{Irr}(G_i)$ and $\mathcal{T}_i$ for the subgroups G_i. The central idea of the BC-Algorithm is based on the following version of Clifford's Theorem:

Theorem 1.1 (Clifford's Theorem). *Let G and $\mathcal{T}$ be as above. Given an irreducible representation F of $\mathbb{C}G_{i-1}$, $0 < i \le n$, one of the following two cases holds:*

(1) *F extends to $p := p_i$ pairwise inequivalent irreducible representations $D_0, \ldots, D_{p-1}$ of $\mathbb{C}G_i$ of the same degree $\deg(F)$. Moreover, if $\chi^0, \chi^1, \ldots, \chi^{p-1}$ 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 this way.

This allows us to construct the irreducible representations of G iteratively in a bottom-up fashion along the chief series $\mathcal{T}$, where at each level i, $1 \le i \le n$, 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 any transversal $\mathrm{Irr}(G_i)$ but a very special set of representations with the following properties: First, the constructed representations of $\mathrm{Irr}(G_i)$ are *monomial*, meaning that all matrices $D(g)$, $g \in G$, $D \in \mathrm{Irr}(G_i)$, are monomial (having exactly one non-zero entry in each column and row). Second, all matrix entries of $D(g)$ are eth 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 $\mathbb{Z}/\mathbb{Z}_e$, i.e., there are no numerical problems. Third, the representations $D \in \mathrm{Irr}(G_i)$ are "optimally" adapted to the

chief series $\mathcal{T}_i$ of G_i in the sense that the restriction of D to any subgroup G_ℓ, $1 \leq \ell < i$, is not only *equivalent* - as stated in Clifford's Theorem - but even *equal* to the direct sum of representations of $\mathrm{Irr}(G_\ell)$. This property is refered to as $\mathcal{T}_i$*-adaptivity* of D and ensures that all data already computed up to level $i-1$ can be used without modification for the construction in the following levels $i, \ldots n$. To achieve $\mathcal{T}_i$-adaptivity the so-called *intertwining-spaces* come into play, whose computation, as it turns out, are the bottle-neck of the BC-Algorithm. We summarize the result in the following theorem:

Theorem 1.2. *Let G be a finite supersolvable group of order $N := |G|$, with exponent e and chief series $\mathcal{T}$. Then the BC-Algorithm constructs from a consistent pc-presentation of G a transversal $\mathrm{Irr}(G)$ of e-monomial $\mathcal{T}$-adapted irreducible representations with $O(N \log^2 N)$ operations in the additive group $\mathbb{Z}/\mathbb{Z}_e$.*

As we will see in the next section the output length of the BC-Algorithm is - using an appropriate data structure - linear in N. Therefore, the running time is - up to the polylogarithmic factor $\log^2 N$ - linear in the output lenght N and can in this sense be considered as almost optimal.

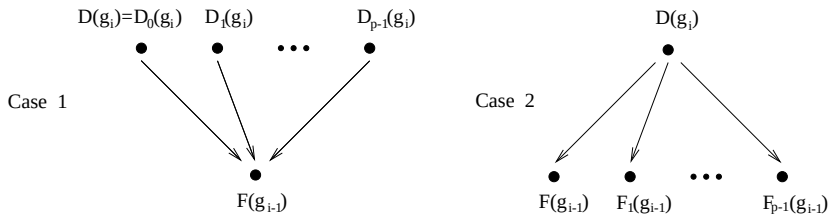
2 Data Structure

In the case of supersolvable groups G of order $N := |G|$ the output data of the BC-Algorithm can be realized by a compact data structure, denoted by `DS(`G`)`, since the representations are e-monomial. Each e-monomial matrix $M \in \mathbb{C}^{m \times m}$ can be written as

$$M = \pi \cdot \mathrm{diag}(\omega^{a_1}, \ldots, \omega^{a_m}) \tag{1}$$

with a permutation $\pi \in S_m$ and non-trivial coefficients $\omega^{a_1}, \ldots, \omega^{a_m}$, where we have fixed a primitive eth root of unity ω. Therefore M can be described by the $2m$ positive integers $\pi(1), \ldots, \pi(m)$ and $a_1, \ldots, a_m$, which are all bounded by N. In the following we want to make the assumption, that these integers as well as pointers can each be stored in a 32-bit word or `double` (this is no restriction with regard to the applications we have in mind).

The BC-Algorithm computes $\mathrm{Irr}(G)$, where each representation $D \in \mathrm{Irr}(G)$ is given by its representing e-monomial matrices $D(g_1), \ldots, D(g_n)$ on the generators of the pc-presentation of G. Using (1) one can easily show that this leads to a memory requirement of $2N \log N$ `double`s. Using the $\mathcal{T}$-adaptivity one can do even better. To this means we introduce the $\mathcal{T}$-character graph of G, which consits of $n+1$ levels. Each node of level i, $0 \leq i \leq n$, corresponds uniquely to a representation $D \in \mathrm{Irr}(G_i, \mathcal{T}_i)$ (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 $D \in \mathrm{Irr}(G_i)$ and $F \in \mathrm{Irr}(G_{i-1})$ iff D restricted to G_{i-1} contains F as a direct summand. In order to store $\mathrm{Irr}(G)$, it suffices to store the character graph of G, where we need a pointer for each edge, and for each node corresponding to $D \in \mathrm{Irr}(G_i)$ we store the e-monomial matrix $D(g_i)$ given by $2\deg(D)$ integers. By Clifford's Theorem the so defined data structure `DS(`G`)` is locally in D of the following form, where $p := p_i$:



In Case 1 it follows again by Clifford's Theorem (using the notation of Theorem 1.1) that $D_k(g_i) = \chi^k(g_i G_{i-1}) D_0(g_i)$ for $1 \leq k \leq p$. Therefore it suffices to store the matrix $D_0(g_i)$ and for $1 \leq k \leq p$ just the integers $\chi^k(g_i G_{i-1})$ and a pointer on $D_0(g_i)$. A straightforward analysis of `DS(`G`)` leads to the following result:

Lemma 2.1. *The data structure* `DS(`G`)` *for a finite supersolvable group G requires at most* $12\,N$ `double`*s for all pointers and integers.*

3 Implementation

The BC-Algorithm has been implemented in the programming language C/C++. Tests were run on an Intel Pentium III, 700 MHz with 128 MByte RAM, which allowed experiments up to group orders $N \approx 10^7$. The following table shows the running times t in milliseconds (ms) and the memory requirements (MR) in byte needed by the BC-Algorithm. In this example, the groups G are m-fold direct products of the symmetric group S_3 of order 6 for $m = 3, \ldots, 10$. Even though the groups are of very simple nature - the irreducible representation could simply be computed by forming tensor products - they illustrate very well the running time behaviour of the implementation of the BC-Algorithm which does not essentially depend on the complexity of the pc-presentations.

G	$\lvert G\rvert$	n	h	$d^1(G)$	MR (byte)	MR/$d^1(G)$	t (ms)	$t/d^1(G)$
$(S_3)^3$	216	6	27	64	4728	73.9	<1	0
$(S_3)^4$	1296	8	81	256	14476	56.5	2	0.008
$(S_3)^5$	7776	10	243	1024	45704	44.6	18	0.018
$(S_3)^6$	46656	12	729	4096	147516	36.0	74	0.018
$(S_3)^7$	279936	14	2187	16384	485656	29.6	270	0.017
$(S_3)^8$	679616	16	16561	65536	1631084	24.9	1078	0.016
$(S_3)^9$	10077696	18	19683	262144	5591592	21.3	4844	0.019
$(S_3)^{10}$	60466176	20	59049	1048576	19570204	18.7	22105	0.021

A more careful analysis of the BC-Algorithm and `DS(`G`)` reveals, that the upper bounds $O(N \log^2 N)$ for the running time and $O(N)$ for the memory requirement are in general rough upper bounds, which are only sharp in case of G being abelian. In general, the complexity of BC-Algorithm depends very much on the number and degrees of the irreducible representations of $\mathrm{Irr}(G)$. On can show that replacing N by the number $d^1(G) := \sum_{D \in \mathrm{Irr}(G)} \deg(D)$ in the upper bounds reflect much better the running time be-

haviour. (Note that $d^1(G) \leq N$ and $d^1(G) = N$ iff G is abelian.) The table shows that the running time as well as the memory requirement do not explode with increasing group order N but are about linear in $d^1(G)$.

To test the implementation on more complex groups, we have generated a library of several thousand consistent pc-presentations defining supersolvable groups of different order up to 10^7. This was done by a randomized algorithm using the computer algebra system `GAP`. The running time behaviour of the implementation of the BC-Algorithm for groups of different order is illustrated by the following table. Here $\mathrm{Lib}_N(k)$ denotes the kth 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}.

G	$\lvert G\rvert$	n	h	$d^1(G)$	MR (byte)	MR/$d^1(G)$	t (ms)	$t/d^1(G)$
$\mathrm{Lib}_{1024}(1)$	1024	10	175	288	23028	80.0	4	0.0139
$\mathrm{Lib}_{2187}(1)$	2187	7	155	351	22612	64.4	3	0.0086
$\mathrm{Lib}_{7560}(1)$	7560	8	2295	4050	434272	107.0	111	0.0274
$\mathrm{Lib}_{7560}(54)$	7560	8	945	2016	58084	28.8	10	0.0050
$\mathrm{Lib}_{7776}(1)$	7776	10	333	1152	42936	37.3	15	0.0130
$\mathrm{Lib}_{15625}(35)$	15625	6	265	1225	32396	26.4	8	0.0065
$\mathrm{Lib}_{16000}(1)$	16000	10	424	2112	100408	47.5	67	0.0317
$\mathrm{Lib}_{22287}(1)$	22287	4	22287	22287	1304644	58.5	96	0.0043
$\mathrm{Lib}_{23940}(1)$	23940	7	3990	6840	483464	70.7	116	0.0170
$\mathrm{Syl}_2(S_{16})$	32768	15	230	2064	134760	65.3	144	0.0698
$\mathrm{Lib}_{42875}(1)$	42875	6	1595	4095	223812	54.7	35	0.0086
$\mathrm{Lib}_{179894}(3)$	179894	5	179894	179894	7441040	41.4	253	0.0014

The implementation described in this paper is part of the computer system SUGAR - SUpersolvable Groups and Algorithmic Representation Theory - which also comprises routines for fast evaluation of representations (generalized fast Fourier transforms), for construction of pc-presentations and for signal processing among others. First experiments have been conducted to apply these generalized discrete Fourier transforms to digital signal processing for efficient randomized signal compression. For details and further links to the literature we refer to [3].

References

[1] Baum, U., Clausen, M.: Computing irreducible representations of supersolvable groups. Mathematics of Computation, Volume **63**, Number 207 (1994) 351–359.

[2] Clausen, M., Baum, U.: Fast Fourier Transforms. BI-Wissenschaftsverlag, 1993.

[3] Müller, M.: Beiträge zur Algorithmik verallgemeinerter diskreter Fouriertransformationen. PhD thesis, Universität Bonn, Institut für Informatik, 2001.

Authors: Meinard Müller, Michael Clausen, Universität Bonn, Institut für Informatik III, Römerstr. 164, 53117 Bonn, Germany, {meinard, clausen}@cs.uni-bonn.de