An introduction to Computational Group Theory

Ákos Seress
The Ohio State University
Columbus, OH 43210

Can one rotate *only one* corner piece in Rubik's cube? What are the energy levels of the buckyball molecule [53]? Are the graphs on Fig. 1 isomorphic? What is the Galois group of the polynomial $x^8 + 2x^7 + 28x^6 + 1728x + 3456$ [71]? What are the possible symmetry groups of crystals [15]? These are all questions which, directly or by a not so obvious way, lead to problems in Computational Group Theory.

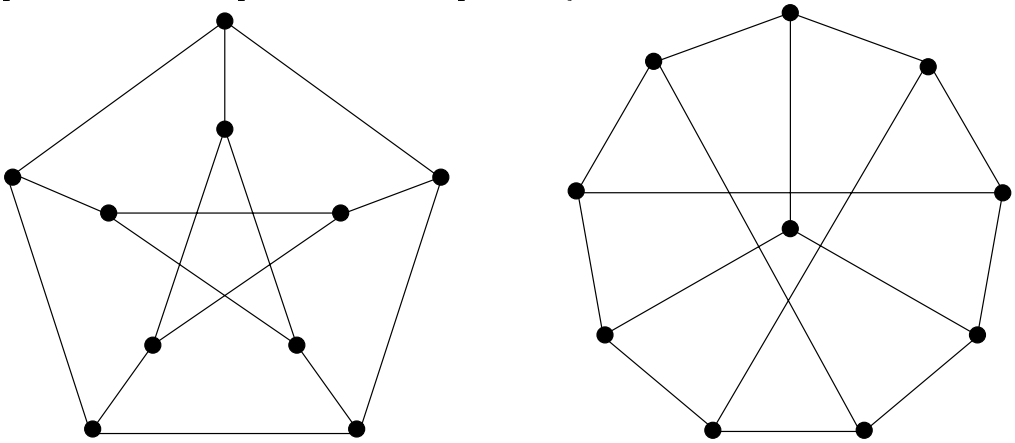

Fig. 1

Algebraic structures are well suited for machine computations. One reason for that is that we can describe large objects very concisely, by a set

Partially supported by NSF Grant CCR-9503430 and by the Alexander von Humboldt Foundation.

of generators: for example, 50 bits are enough to define $\mathrm{GL}_5(2)$, a group of order 9999360, by two 0-1 matrices of size 5×5. Even more importantly, often we can find a generating set which reflects the structure of the group so that structural and quantitative properties can be read off easily.

Computational Group Theory (CGT) is one of the oldest and most developed branches of computational algebra. Although most general purpose symbolic algebra programs can handle groups to a certain extent, there are two systems which are particularly well suited for computations with groups: GAP [95] and MAGMA [20]. Also, numerous stand-alone programs and smaller systems are available.

GAP can be obtained by anonymous ftp from servers on three continents; the addresses can be found on the World Wide Web page

$$\texttt{http://www-groups.dcs.st-and.ac.uk}.$$

For the availability of MAGMA, please see the World Wide Web page

$$\texttt{http://www.maths.usyd.edu.au:8000/comp/magma}.$$

The important subareas of CGT correspond to the most frequently used representations of groups: permutation groups, matrix groups, and groups defined by generators and relators, as well as to the perhaps most powerful tool for the investigation of groups, representation theory. Also, there are specialized and more efficient algorithms for special classes such as nilpotent or solvable groups. In this survey, in each subarea we attempt to indicate the basic ideas, and the size of jobs which can be handled by the current systems on a reasonable machine. Of course, we cannot be comprehensive here; [19], [3], [17], [18], [37], [38], [96], [103] indicate the breadth of the field, and their reference lists can be starting points for further reading. A case study, using a wide variety of methods, is described in [79].

We start with some historical remarks. Algorithmic questions permeated group theory from its inception in the last century. As examples, consider that Galois' work was inspired by the solvability of equations. Thirty years later, Mathieu announced the existence of the 5-transitive group M_{24}, but needed twelve years to find the "clarity and elegance necessary to present it". Had he access to a computer, this period could probably have been shortened significantly. Jordan, Hölder, Cole, and others could also have used the machine in their quest to classify groups of small order.

The "official" starting date of CGT may be pinned down in 1911, when Dehn proposed the solution of the word problem: find an algorithm to decide whether, in a group defined by a finite set of abstract generators and relators, a word in the generators represents the identity. Dehn's question was motivated by topological considerations; even today, it is hard to draw a sharp border between combinatorial group theory and topology. The flourishing of CGT started in the sixties [55], when e.g. the basic methods for permutation group manipulation [98], [99] and for the computation of character tables [32] were established, and term rewriting procedures were introduced [52]. Not much later, the first large applications such as Sims' existence proof for Lyons' sporadic simple group [100] arose, and the development of the first integrated system, the Aachen–Sydney Group System, started. Since then, the area is growing rapidly, both in terms of the design, implementation, and application of algorithms, as well as the number of mathematicians involved in this development. Nowadays, some of the major lines of development are the integration of consequences of the classification of finite simple groups and methods suggested by complexity theoretical considerations into practical algorithms, and the systematic use of randomization. A more detailed history is in [79].

Acknowledgements. I am indebted to G. Havas, D. Holt, W. Kantor, K. Lux, J. Neubüser, and E. O'Brien for their helpful comments. The part of Section 2 about quotient group methods was written by J. Neubüser.

1 Finitely presented groups

Let $G = \langle E|\mathcal{R}\rangle$ be a presentation for a group G: $E = \{g_1, ..., g_n\}$ is a finite set of generators, and $\mathcal{R} = \{r_1 = 1, ..., r_m = 1\}$ is a set of defining relations. Each r_i is a word, using the generators in E and their inverses. The basic questions are to decide whether G is finite, and to determine whether a given word represents the identity of G.

By the celebrated result of Novikov and Boone, these questions are *undecidable*: they cannot be answered by a recursive algorithm. Nevertheless, because of the practical importance of the problem, a lot of effort is devoted to the development of methods for investigating finitely presented groups.

One basic method is the *Todd–Coxeter coset enumeration procedure.* Given $G = \langle E|\mathcal{R}\rangle$ and $H = \langle h_1, ..., h_k\rangle$, where $H \leq G$ and each h_j is a word in the

generators of G and their inverses, our goal is to compute the permutation representation of G on the right cosets of H.

We set up a *coset table*: this is a matrix M, where the rows are labelled by positive integers, representing cosets of H, and the columns are labelled by the elements of $\overline{E} := \{g_1, ..., g_n, g_1^{-1}, ..., g_n^{-1}\}$. The entries (if defined) are positive integers, $M(k, g) = l$ if we know that $kg = l$ for the cosets k, l and for $g \in \overline{E}$. Originally, we have a $1 \times |\overline{E}|$ table with no entries, where 1 denotes the coset $H \cdot 1$. As new cosets are defined, we add rows to the coset table.

Of course, we have to detect when two words, defining different rows of the table, actually belong to the same coset of H. To this end, for each relation $r_i = g_{i_1} g_{i_2} \cdots g_{i_t}$, we also maintain a *relation table*. This is a matrix M_i, with rows labelled by the cosets 1,2,... as defined in M, and columns labelled by the elements of the sequence $(g_{i_1}, g_{i_2}, \ldots, g_{i_t})$. The entry $M_i(k, g_{i_j})$, if defined, is the number of the coset $kg_{i_1} \cdots g_{i_j}$. Initially, we have $M_i(k, g_{i_t}) = k$ for each row number k, since $r_i = 1$ in G. Whenever a new coset is defined, we fill all entries of the relation tables which we can.

Finally, for each generator $h_j = g_{j_1} \cdots g_{j_t}$ of H, we maintain a *subgroup table*. This is a matrix S_j with only one row, corresponding to the coset $H \cdot 1$, and columns labelled by the factors of h_j. The rule for filling entries is the same as for the M_i; originally, $S_j(1, g_{j_t}) = 1$, since $Hh_j = H$.

When the last entry is filled in a row of a relation table or a subgroup table, we also get an extra piece of information $kg = l$ for some cosets k, l and $g \in \overline{E}$. This is called a *deduction*. If the entry $M(k, g)$ is not yet defined then we fill the entries $M(k, g)$, $M(l, g^{-1})$ and all possible entries in the relation and subgroup tables; this way, we may get further deductions. If $M(k, g)$ is already defined but $l' := M(k, g) \neq l$, then we realize that l, l' denote the same coset of H. This is called a *coincidence*. We replace all occurrences of l, l' by the smaller of these two numbers and fill the entries of the tables which we can. This may lead to further deductions and coincidences. The process stops when all entries of the coset table, the relation tables, and subgroup tables are filled.

We illustrate these ideas by enumerating $G = \langle g_1, g_2 | g_1^2 = 1, g_2^2 = 1, (g_1 g_2)^3 = 1 \rangle \cong S_3$ on the cosets of the subgroup $H = \langle g_1 g_2 g_1 g_2 \rangle$ of order 3. Since both generators are involutions, we have $\overline{E} = E$. Also, we maintain only one relation table, corresponding to $(g_1 g_2)^3 = 1$; the other two relators tell us that at the definition of new cosets, we should multiply previous cosets by g_1, g_2 alternatingly. Fig.2 shows the coset table CT, relation table RT, and subgroup

table ST after the definition of the cosets $1 := H, 2 := 1g_1, 3 := 1g_2, 4 := 2g_2$. At that moment, the last entry (in the second column) of ST is filled and we get the deduction $4g_1 = 3$, which also implies $3g_1 = 4$. Then all entries in CT are known, and we can complete RT; this leads to the coincidences $1 = 4$ and $2 = 3$.

CT	g_1	g_2
1	2	3
2	1	4
3		1
4		2

RT	g_1	g_2	g_1	g_2	g_1	g_2
1	2	4			3	1
2	1	3			4	2
3			4	2	1	3
4			3	1	2	4

ST	g_1	g_2	g_1	g_2
1	2	4	3	1

Fig. 2

Taking minimal care in defining new cosets (namely, if a coset k is defined then sooner or later we define kg for all $g \in \overline{E}$), it is guaranteed that the algorithm terminates if $|G : H| < \infty$. However, there is no recursive function of $|G : H|$ and the input length which would bound the number of cosets defined during the procedure. It is easy to give presentations of the trivial group (cf. e.g., [103, 5.8]) such that no commonly used variant of the Todd–Coxeter algorithm can handle them. For different coset enumeration strategies and performance statistics, we refer to [40]. Success mostly depends on the number of entries defined in the coset table rather than $|G : H|$. There are instances of successful enumeration with $|G : H| > 10^6$.

A possibility for saving memory space is to enumerate double cosets HgK, instead of right cosets of H. Following a suggestion of Conway, Linton [61] implemented such procedure. It works well when H has reasonably small index in G, and we have detailed knowledge about K: for example, if we know the possibilities for $K \cap K^g$, for $g \in G$.

If we do not have a candidate for a small index subgroup H in G, we may try programs which find some or all subgroups of G with index at most a given bound n. These programs consider all coset tables with at most n rows, and use a backtrack search to eliminate those which are not consistent with the given relators. Depending on the complexity of the presentation, in some cases we can expect success for values of n up to about 100.

Another line of attack is to look for quotient groups with specific structure. The simplest case is to find G/G': to this end, we only have to add the

relators $[g_i, g_j] = 1$ to $\mathcal{R}$ for all $g_i, g_j \in E$, and find the cyclic decomposition of the resulting abelian group. In an abelian group $G = \langle E|\mathcal{R}\rangle$, the exponents of the generators in the relators define an $|E| \times |\mathcal{R}|$ integer matrix S, and the cyclic decomposition can be read out from the Smith normal form (SNF) of S. The SNF can be computed by row and column operations of S; however, the integer entries occurring during the computation are typically much larger than the input and the final result, and the procedure has exponential worst case complexity [36]. By different methods, the SNF can be computed in polynomial time. The current asymptotically fastest variant is described in [105]. In matrices with hundreds of rows, recent practical approaches [41], [42] use modular arithmetic and lattice reduction techniques [58].

Harder, but still manageable is to find large nilpotent or even polycyclic quotient groups. We shall discuss such methods at the end of Section 2.

So far, we have seen methods which found factor groups of G or a permutation representation on the cosets of a subgroup. To proceed further, we often need presentations for subgroups. Schreier's subgroup lemma provides a way to obtain generators for a subgroup $H \leq G$.

Lemma 1.1 *Let T be a right transversal for H in $G = \langle S\rangle$ and, for $g \in G$, let $\overline{g}$ denote the element of T such that $g \in H\overline{g}$. Then*

$$\{ts\overline{ts}^{-1} : t \in T, s \in S\}$$

generates H.

Using that words representing subgroup elements can be rewritten as products of Schreier generators, one may obtain a presentation for H. Although there are methods to eliminate some unnecessary relators [78], this presentation is usually highly redundant. It can be shortened by applying the so-called *Tietze transformations*: (i) if a generator occurs only once in some relator, then this generator can be expressed as a product of the others; (ii) the product of two generators g_i, g_j can be introduced as a new generator g_k; (iii) if $uv = 1$ is a relator then a relator $w_1uw_2 = 1$ can be replaced by $w_1v^{-1}w_2 = 1$ (u, v, w_1, w_2 are words). In principle, any two presentations of a given group can be transformed to each other by a finite sequence of Tietze transformations, but Novikov–Boone implies that there is no recursive function of the input length which would bound the number of necessary steps.

It is often worthwhile to do Tietze transformations interactively, guiding the computer to the type of presentation we try to achieve.

An alternative method to coset enumeration is the *Knuth–Bendix term-rewriting procedure* [52], [103]. We collect a list of pairs of words (u, v) such that u, v represent the same element of G. These pairs are called rewriting rules since we can replace a word w_1uw_2 by w_1vw_2. The goal is to collect a *confluent* system of rules: no matter in which order are the rules applied, every word in the generators is converted into a unique normal form. Although the usual problems of undecidability arise, [103, 5.8] describes instances when a version of the Knuth–Bendix procedure can handle presentations which are beyond the capability of the Todd–Coxeter methods. Even more importantly, Knuth–Bendix methods can sometimes solve the word problem for infinite groups, which can almost never be done by coset enumeration techniques.

An interesting recent development is the definition and the algorithmic handling of *automatic groups* [35], [43]. These are groups with solvable word problem, and include important group classes occurring in topology such as the fundamental groups of compact hyperbolic and Euclidean manifolds, and of hyperbolic manifolds of finite volume, word hyperbolic groups and groups satisfying various small cancellation properties.

2 Polycyclic groups

In the rest of this survey, we shall almost exclusively deal with groups for which the undecidability of the word problem vanishes. This will clearly be the case with finite groups, given e.g. as permutation or matrix groups, but it is also the case for groups given by polycyclic presentations.

A group is called *polycyclic* if it has a finite subnormal series of the form

$$G = G_1 \rhd G_2 \rhd \cdots \rhd G_n \rhd G_{n+1} = 1, \tag{1}$$

with cyclic factors G_i/G_{i+1}. If, for $1 \leq i \leq n$, we pick $a_i \in G_i \setminus G_{i+1}$ such that $G_i = \langle G_{i+1}, a_i \rangle$, then each $g \in G$ can be written uniquely in the form $g = a_1^{e_1} a_2^{e_2} \cdots a_n^{e_n}$, where $e_i \in \mathbb{Z}$ for infinite cyclic factors G_i/G_{i+1}, and $0 \leq e_i < |G_i : G_{i+1}|$ for finite factor groups. This description of g is called a *collected word* for g. For finite polycyclic groups, we also have

$$\begin{aligned} a_i^{|G_i:G_{i+1}|} &= a_{i+1}^{\varepsilon_{i;i+1}} a_{i+2}^{\varepsilon_{i;i+2}} \cdots a_n^{\varepsilon_{i;n}}, \text{ for } 1 \leq i \leq n, \\ a_j a_i &= a_i a_{i+1}^{\varepsilon_{i,j;i+1}} a_{i+2}^{\varepsilon_{i,j;i+2}} \cdots a_n^{\varepsilon_{i,j;n}}, \text{ for } 1 \leq i < j \leq n. \end{aligned} \tag{2}$$

These are the relators for the so-called power-conjugate presentation (pcp) for G. (For infinite polycyclic groups, similar relators have to be added to (2) for the inverses of the a_i of infinite order.)

The pcp's are very compact, but still relatively easily manageable representations of polycyclic groups. Sims [103, Ch. 9] discusses algorithms which are efficient even for infinite polycyclic groups: for example, it is possible to test membership in subgroups, compute normal closures and so derived and lower central series, and manage homomorphisms. For finite solvable groups, pcp's can be used to determine virtually all structural properties [54], [73], [39], [24], [34], [104]. Among others, conjugacy classes of elements, complements to normal subgroups, normalizers, Frattini subgroups can be computed, and it is possible to find automorphism groups. For these tasks, if there are algorithms in other representations of groups, they can handle only groups of much smaller order. There are examples of structural exploration of polycyclic groups with $\log |G|$ in the thousands; currently, groups of that size can not be handled in any other representation.

We indicate two basic methods, which are used in almost all algorithms dealing with finite solvable groups. The first one is to exploit analogies with linear algebra. Each collected word can be represented as an integer vector of length n, the coordinates being the exponents of the a_i. The relators (2) imply that if $w_1, , w_2$ are collected words and the first m positions of w_1 are 0, then the first m positions in the collected word for $w_1 w_2$ are the same as in w_2. Thus, given a list of generators for a subgroup $H \leq G$, a non-commutative version of Gaussian elimination can be used to obtain generators $h_1, ..., h_k$ for H in row-echelon form. (Besides the usual steps of row reduction, powers and commutators using the newly obtained vectors have to be formed, corresponding to the construction of Schreier generators in Lemma 1.1.) Then each $h \in H$ can be written uniquely in the form

$$h = h_1^{d_{i_1}} h_2^{d_{i_2}} \cdots h_k^{d_{i_k}}, \tag{3}$$

with exponents $0 \leq d_{i_j} < |G_{i_j} : G_{i_{j+1}}|$. The procedure can be used for membership testing in H as well, by attempting to factorize any given $g \in G$ as in (3). When membership testing is available, we can compute normal closures of subgroups, derived series and lower central series, and handle homomorphisms. We shall see an analogous procedure for permutation groups in Section 3.

The other basic method is to consider larger and larger factor groups of G. We construct a subnormal chain (1) which is a refinement of a chain of normal subgroups $G = N_1 \rhd N_2 \rhd \cdots \rhd N_m = 1$, with N_i/N_{i+1} elementary abelian. Then we use induction to extend information from G/N_i to G/N_{i+1}. Since G/N_i acts as a group of linear transformations on N_i/N_{i+1}, often linear algebra methods can be used.

The time-critical part of computations with a pcp is usually to find the collected word for the product of two collected words. The pairs of words in the equations in (2) comprise a confluent rewriting system, so applying these rules in any order, we shall get the collected word. Of the various possible strategies, "collection from the left" seems to be presently the method of choice, which means that we try to eliminate the first occurrence $a_j a_i$, $j > i$, from the left end of the word [57]. In nilpotent groups, it is possible to determine formulae for the exponents in the product of two collected words, which usually provides a faster way for multiplication.

Given the usefulness of pcp's, one may try to construct one from other representations. From a permutation representation, a pcp can be obtained relatively painlessly [16] (although, for some tasks, there are permutation group algorithms which may run faster than the conversion to a pcp and the application of the polycyclic method [63]). In most applications, the groups arise as finitely presented groups, and there is a collection of algorithms for finding pcp's of certain factor groups.

The first of these was a pQ-algorithm [70] for finding p-quotients of a finitely presented group. This was soon turned into a powerful tool [83], and used for the study of *Burnside groups*. Let F_n be the free group of rank n and q a power of a prime. The Burnside group $B(n,q)$ is the largest factor group of F_n with all elements of order dividing q. $B(n,q)$ may be infinite but, as recently was proved by Zelmanov, it has a largest finite factor group $\overline{B}(n,q)$. As early as 1975, it was determined that $|\overline{B}(4,4)| = 2^{422}$ [1]. (In this particular instance of Burnside groups, we have $B(4,4) = \overline{B}(4,4)$.) Present implementations can determine quotients of Burnside groups of order p^k with k in the thousands for small p [84].

The family of available quotient group algorithms nowadays includes an NQ-method which determines possibly infinite nilpotent quotients of a finitely presented group [85], at least two different SQ-methods for finding finite solvable factor groups [91], [86], and a PCQ-method for finding infinite polycyclic factor groups [102], [62]. All of these algorithms proceed by

induction. As a first step, an epimorphism $\varphi : G \to G/G'$ is constructed (cf. Section 1). Suppose that an epimorphism φ is known onto a nilpotent or polycyclic group H. Then an attempt is made to lift φ to an epimorphism of G onto an extension group K of H by a normal subgroup N that is central in the case of pQ and NQ, or just abelian in the other cases.

The methods for finding such N and K and for lifting φ differ in the various quotient methods. In the case of pQ, they amount to solving linear equations over $\mathrm{GF}(p)$ that on one hand represent confluence conditions for the pcp presentation and on the other hand stem from the relations for G. In the case of NQ, congruences occur instead of equations, while in the case of SQ's modular representations of H are considered. Finally, in the case of PCQ's, Gröbner basis techniques come into play. It should be no wonder that the time requirement for NQ, and especially for SQ and PCQ is much higher than for pQ, and so applications of these are far more restricted.

3 Permutation groups

The situation is quite satisfactory in the case of permutation group algorithms as well: most structural properties of permutation groups of reasonable degree and order can be readily computed. Working with permutation representations is usually the method of choice when studying simple groups and their subgroup structure, provided that the simple group has a permutation representation of sufficiently small degree. "Reasonable" and "sufficiently small" degree, of course, depends on the available software and hardware; currently, it is in the low hundred thousands, while the logarithm of the group order may be in the low hundreds.

This is the area of CGT where the complexity analysis of algorithms is the most developed. The reason for that is the connection with the celebrated *Graph Isomorphism Problem*. The decisive result in establishing the connection is Luks' polynomial time algorithm for the isomorphism testing of graphs with bounded valence [64], where the isomorphism problem is reduced to finding setwise stabilizers of subsets in the permutation domain of groups with composition factors of bounded size. This result not only established a link between complexity theory and CGT, but provided new methodology for permutation group algorithms.

Because of the approach from two different point of views, complexity the-

oretic and practical, algorithms for numerous tasks were developed independently in the two contexts. In the last five years, a remarkable convergence of the approaches occurred: for example, in GAP, most of the permutation group library uses implementations of algorithms with the fastest known asymptotic running times.

The basic ideas of permutation group manipulation are due to Sims [98, 99]. A *base* for $G \leq \mathrm{Sym}(\Omega)$ is a sequence $B = (\beta_1, ..., \beta_m)$ of points from Ω such that the pointwise stabilizer $G_B = 1$. The sequence B defines a subgroup chain

$$G = G^{[1]} \geq G^{[2]} \geq \cdots \geq G^{[m]} \geq G^{[m+1]} = 1 \tag{4}$$

where $G^{[i]} = G_{(\beta_1,...,\beta_{i-1})}$. A *strong generating set* (SGS) relative to B is a generating set S for G with the property that

$$\langle S \cap G^{[i]} \rangle = G^{[i]}, \text{ for } 1 \leq i \leq m+1. \tag{5}$$

Given an SGS, the orbits $\beta_i^{G^{[i]}}$ can be easily computed, and the product of orbit sizes gives $|G|$ immediately. Also, keeping track of elements of $G^{[i]}$ in the orbit algorithm which carry β_i to points in $\beta_i^{G^{[i]}}$, we obtain (right) transversals R_i for $G^{[i]}$ mod $G^{[i+1]}$. (In practice, in order to save memory space, the transversal elements are stored only as words in the strong generators.) Every $g \in G$ can be written uniquely in the form

$$g = r_m r_{m-1} \cdots r_1 \tag{6}$$

with $r_i \in R_i$. This decomposition can be done algorithmically: given $g \in G$, find the coset representative $r_1 \in R_1$ such that $\beta_1^g = \beta_1^{r_1}$. Then compute $g_2 := g r_1^{-1} \in G^{[2]}$; find $r_2 \in R_2$ such that $\beta_2^{g_2} = \beta_2^{r_2}$; compute $g_3 := g_2 r_2^{-1} \in G^{[3]}$; etc. This factorization procedure is called *sifting* and may be considered as a permutation group version of Gaussian elimination. Sifting can also be used for *testing membership* in G. Given $h \in \mathrm{Sym}(\Omega)$, we attempt to sift it; $h \in G$ if and only if we can factorize h as a product of coset representatives.

We sketch the basic idea of an SGS construction. Given $G = \langle T \rangle$, we maintain a sequence $B = (\beta_1, ..., \beta_k)$ of already known elements of a base, and approximations S_i for a generating set of the stabilizer $G_{(\beta_1,...,\beta_{i-1})}$. During the construction, we always maintain the property that for all i, $\langle S_i \rangle \geq \langle S_{i+1} \rangle$. We say that the data structure is *up-to-date below level j* if

$$\langle S_i \rangle_{\beta_i} = \langle S_{i+1} \rangle \tag{7}$$

holds for all $j < i \leq k$. It is easy to see that $\bigcup S_i$ is an SGS for G if and only if it generates G and the data structure is up-to-date below level 0.

In the case when the data structure is up-to-date below level j, we compute a transversal R_j for $\langle S_j \rangle$ mod $\langle S_j \rangle_{\beta_j}$. Then we test whether (7) holds for $i = j$. By Lemma 1.1, this can be done by sifting the Schreier generators obtained from R_j and S_j in the group $\langle S_{j+1} \rangle$. If all Schreier generators are in $\langle S_{j+1} \rangle$ then we say that the data structure is up-to-date below level $j-1$; otherwise, we add a non-trivial Schreier generator h to S_{j+1}, and say that the data structure is up-to-date below level $j+1$. In the case $j = k$, we also choose a new base point β_{k+1} from the support of h.

We start the algorithm by choosing $\beta_1 \in \Omega$ which is moved by at least one generator in T and setting $S_1 := T$. At that moment, the data structure is up-to-date below level 1; the algorithm terminates when the data structure becomes up-to-date below level 0.

Sims' algorithm is of fundamental importance to permutation group manipulation. Besides computing $|G|$ and setting up membership testing in G, the method can be applied for computing normal closures of subgroups, derived series and lower central series, handling homomorphisms, and finding the pointwise stabilizer of any subset of Ω. In practice, usually a random variant (heuristic [59] or Monte Carlo [7]) is applied. Then another algorithm proposed by Sims, the so-called "Verify-Routine", can be run to guarantee the correctness of the answer.

A second generation of algorithms uses divide and conquer techniques by utilizing the imprimitivity block structure of the input group, thereby reducing the problems to primitive groups. Besides [64], we mention [65], [81] for computing a composition series, [49], [50] for Sylow subgroups, and the asymptotically fastest deterministic SGS constructions [9], [8]. Although, in [8], an in-depth structural examination of G is necessary before $|G|$ is obtained, namely, the computation of a composition series, the algorithm runs almost a factor n^2 faster than the asymptotically fastest versions of Sims' original method. The required group theoretic arsenal for these algorithms include consequences of the classification of finite simple groups and, in the case of [49], [50], detailed knowledge of the classical simple groups. (We mention, however, that a composition series can be computed without using the simple group classification [12].)

The running time of most algorithms for $G = \langle S \rangle \leq S_n$ does not only depend on the input length $|S|n$, but also on the order of G. Moreover, in

groups of current interest, it frequently happens that the degree of G is in the tens of thousands or even higher, so even a $\Theta(n^2)$ algorithm may not be practical. On the other hand, $\log |G|$ is often small, bounded from above by a polylogarithmic function of n. Therefore, a recent trend is to search for algorithms with running time of the form $O(n|S| \log^k |G|)$. These notions are formalized in the following way.

With reference to some constant c, an (infinite) family $\mathcal{G}$ of permutation groups is called a family of *small-base groups* if all $G \in \mathcal{G}$ of degree n admit bases of size $O(\log^c n)$. Equivalently, $\log |G| = O(\log^{c'} n)$ for a constant c' and each $G \in \mathcal{G}$ of degree n. For example, all permutation representations of finite simple groups except the alternating ones, and all primitive groups which do not contain alternating subgroups in their socle belong (with $c = 2$) to the small-base category. Algorithms with running time $O(n|S| \log^k |G|)$ are called *nearly linear*, since their running time is $O(n \log^{c''} n)$ on small-base inputs.

There is a large library of nearly linear algorithms, e.g., [7], [13], [75], [96]. Roughly, everything that can be done in polynomial time can also be done in nearly linear time in groups which do not have composition factors of exceptional Lie type. The price we pay for the speedup is that most algorithms are random *Monte Carlo*: they may return an incorrect answer, with probability of error bounded by the user. (However, if the original SGS construction is correct, further computations also return correct answers.) Most of these algorithms (except the Sylow subgroup computations) are implemented as the default functions in GAP. Although there is a recent theoretical result [51] which upgrades SGS constructions, and so all nearly linear time algorithms, to Las Vegas (i.e., to guaranteed correct output) for groups with no exceptional Lie type composition factors, the practicability of this method is not demonstrated yet.

At present, there are no polynomial time algorithms for some important tasks like computing the centralizer of elements, the intersection of two subgroups, or setwise stabilizers of subsets of the permutation domain. These tasks are polynomially equivalent, and Graph Isomorphism can be reduced to them [67]. Therefore, it is suspected that no polynomial time algorithms exist.

Practical algorithms for these problems use *backtrack methods* [16]. Let B be a fixed base of G. The images of base points uniquely determine the elements of G, while the image of an initial segment of B defines a coset of

some $G^{[i]}$ (cf. (4)). The images of all initial segments define a partial order by inclusion, which is called the search tree for G. Traditional backtrack methods systematically examine the elements of the search tree. Especially valuable is when a node close to the root can be eliminated, because all elements of G less than this node (i.e., elements of the appropriate coset) can be excluded at once. A new generation of backtrack methods was developed by Leon [60], based on ideas from B. McKay's graph isomorphism testing program [72]. Nowadays, centralizers in small-base groups of degree in the tens of thousands can be computed. The computation of normalizers of subgroups seems to be harder: in theory, polynomial time equivalence with centralizers is not known, and practical computation is much more complicated [106].

4 Representation theory

As John Conway writes in the introduction of the Atlas [27], "the ordinary character table is beyond doubt the most compendious way of conveying information about a group to the skilled reader". However, computing a character table from scratch is not an easy task. Therefore, GAP contains tables for the most frequently used groups, including all tables in the Atlas and tables for most of the maximal subgroups of sporadic groups.

A basic idea for computing the complex irreducible characters of a group G is due to Burnside. In the group algebra $\mathbb{C}G$ of G, the sum S_C of the elements of a conjugacy class C is in the center of $\mathbb{C}G$; in fact, taking these sums for all conjugacy classes, we get a basis for $Z(\mathbb{C}G)$. For a fixed conjugacy class C, the products $S_C S_{C_i}$ decompose as a linear combination of the S_{C_j}'s, and the coefficients, taken for all possible C_i, define a matrix M_C. Since the S_{C_j}'s are in $Z(\mathbb{C}G)$, the matrices M_{C_j} commute, and are simultaneously diagonalizable. The irreducible characters can be easily computed from the entries in the diagonal forms.

To perform the computation sketched above, we must know the conjugacy classes and the class-multiplication coefficients, which boils down to an algorithm to determine which conjugacy class a given group element belongs to. Dixon [32] performs the diagonalization over prime fields $\mathbb{Z}_p$ for (large) primes p, and the result is lifted back to $\mathbb{C}$. Schneider [94] introduces strategies to select a small collection of the M_{C_j}'s, from which the new basis for

the diagonal form can be computed. The Dixon–Schneider method can be used for groups of moderate order (about $|G| < 10^9$), and with not too many conjugacy classes (in the low hundreds).

For larger groups, we may try to compute the character table interactively [80], [69]. We may know some (not necessarily irreducible) characters from a matrix or a permutation representation of G, and there are machine commands to induce characters of subgroups, restrict characters of overgroups, and extend characters of factor groups. We may also take products of known characters (corresponding to tensor products of representations) and, if the power maps for conjugacy classes are known, we can compute the generalized characters defined by $\chi^{(n)}(g) := \chi(g^n)$.

Once a set of characters is collected, the hunt for the irreducible ones may begin. Taking scalar products, we may subtract multiples of the known irreducible characters to decrease the norms of the characters in our collection. Earlier methods tried to split characters as the sum of characters with smaller norms. Nowadays, the most effective method seems to be the application of the basis reduction algorithm [58] to the set of known characters, to obtain a basis of the lattice $\mathcal{L}$ of the generalized characters which consists of elements of small norm. Then, following an idea of W. Plesken, we may consider embeddings of the basis vectors of $\mathcal{L}$ into the lattice $\mathbb{Z}^m$ as vectors with the indicated norm, and find the standard basis of $\mathbb{Z}^m$, which corresponds to the irreducible characters, as linear combinations of the basis vectors of $\mathcal{L}$. Quite remarkably, despite the numerous possibilities for the embedding, this method usually produces quickly some irreducible characters.

The method of generating characters by hunting for irreducibles can be iterated: when a larger collection of irreducible characters is known, it may be worthwhile to induce characters of smaller subgroups, or compute higher tensor powers. Initially, we may restrict the computations to rational characters, because of the cost of handling the irrational numbers. When irrational characters are finally computed, we can use the fact that if n is the order of some $g \in G$, then each character value on g can be written as an integer linear combination of the powers of a primitive n^{th} root of unity.

Special algorithms exist for the computation of character tables in several classes of solvable groups [26], [25, Ch. 8].

The other large area of representation theory is the theory of modular representations. Brauer characters are even harder to compute than ordinary ones, so GAP also contains a library of Brauer character tables, including all

tables in the modular extension of the Atlas [48].

In the case of representations over the complex numbers, most computations deal only with the character tables. On the other hand, over finite fields, the emphasis is also on the computation with the representations themselves. One of the reasons is that modular representations occur naturally in different areas of CGT: e.g., when considering groups acting on the elementary abelian factors (Section 2), or when studying matrix groups (Section 5).

Given a representation $\varphi : G \to \mathrm{GL}_d(q)$, specified by the matrix images $x_1, ..., x_t$ of the generators of G, acting on a d-dimensional $\mathrm{GF}(q)$-module M, a basic problem is to find invariant submodules, or to prove that G acts irreducibly on M. In theory, this can be done in deterministic time polynomial in $d \log q$ [92]; in practice, an idea of Parker [90], the so-called Meat-Axe, is used. We describe the generalization by Holt and Rees [46]. Let A be the algebra generated by $x_1, ..., x_t$, and let $a \in A$. We compute an irreducible factor $p(x)$ of the characteristic polynomial of a, $b := p(a)$, the null-space N of b, the A-closure L of a nonzero vector in N, and the A^T-closure L' of a vector in the null-space of the transpose b^T. If $L \lneqq M$ or $L' \lneqq M^T$ then we found an invariant submodule, and it can be shown that if $\dim(N) = \deg(p)$, $L = M$, and $L' = M^T$, then M is irreducible.

Note that we do not have to search through all elements of any submodule of M during the algorithm. The efficiency of proving irreducibility depends on whether we can find $a \in A$ with a factor of the characteristic polynomial satisfying $\dim(N) = \deg(p)$. It turns out [46] that, with the exception of a particular, well-characterized situation, a uniformly distributed random element $a \in A$ has such a factor with probability at least $1/7$. Recently, this bad case was handled efficiently as well. We shall return to the problem of generating random elements in Section 6.

In studying the structure of large dimensional modules, sometimes the *condensation method* [93], [69], [68] can be applied. This means that given $G \leq \mathrm{GL}_d(\mathbb{F})$, we find a subalgebra $A \leq \mathbb{F}G$ and an A-module N of possibly much smaller dimension than M such that the submodule lattices of M and N are isomorphic. For example, in [31], a 976841775-dimensional module of the sporadic simple Thompson group was condensed to 1403 dimensions. Let $H \leq G$ such that $(|H|, |\mathbb{F}|) = 1$, and let $N := M^H$ be the fixed point space

of H. We define

$$e_H := \frac{1}{|H|}\Sigma_{h\in H} h \in \mathbb{F}G \text{ and } A := e_H \mathbb{F}G e_H \leq \mathbb{F}G. \tag{8}$$

It can be shown that if $S^H \neq 0$ for all factors S in a composition series of M then the submodule lattices of M and N are the same.

5 Matrix groups

Matrix groups are important and very compact representations of groups, but pose serious computational problems. For groups over the integers, the membership problem is undecidable already in four dimensions [74]. (However, on the positive side, the finiteness of matrix groups over $\mathbb{Z}$ can be determined by a practical polynomial time algorithm [5].) There are difficulties with matrix groups defined over finite fields as well. Rewriting matrix groups as permutation groups on the underlying vector space, which was the early approach, results in an exponential blowup of the input size. Moreover, two basic ingredients, which were crucial for the efficient handling of solvable and permutation groups, are missing: there may be no subgroup chain with small indices, analogous to (1) or (4), and it is not clear how to generalize Gaussian elimination. Large fields also cause problems, since the discrete logarithm problem arises already for 1×1 matrices. Nevertheless, despite all these difficulties, there is a concentrated attack from both the practical and theoretical sides to determine the structure of matrix groups over finite fields. Currently, this is the most active area of CGT.

Most effort concentrates on the *matrix recognition project*: given a matrix group $G \leq \mathrm{GL}_d(q)$ by a list of generators, find at least one category of Aschbacher's classification to which G belongs. Aschbacher's theorem [2] classifies subgroups of $\mathrm{GL}_d(q)$ into nine categories: roughly, modulo scalars G is an almost simple group; or isomorphic to a subgroup of $\mathrm{GL}_d(q')$ for some $q'|q$; or there is a normal subgroup of G, naturally associated with the action of G on the underlying vector space. Once a category is found, it can be used to explore the structure of the group further. For example, if the group acts imprimitively and we find a decomposition $V = V_1 \oplus V_2 \oplus \cdots \oplus V_m$ of the underlying vector space such that the V_i are permuted by G, then we can construct a homomorphism $\varphi : G \to S_m$. The image of φ is a permutation

group of low degree, so it can be handled easily. In this case, the "naturally associated" normal subgroup is the kernel of φ; we can obtain generators for it, and apply our methods recursively to the action on the lower dimensional vector spaces V_i.

The first subproblem solved both in theory and practice is the recognition of the classical almost simple groups, in their natural representation. Neumann and Praeger [82] introduced the basic ideas. We say that a number is $\mathrm{ppd}(q,d)$ if it has a prime divisor p such that $p|q^d-1$ and $p \not| q^i-1$ for $1 \le i \le d-1$. In [82], precise estimates are given for the proportion of elements in groups $G \ge \mathrm{SL}_d(q)$ whose order is $\mathrm{ppd}(q,d)$ and for those whose order is $\mathrm{ppd}(q,d-1)$. It turns out that among $5.5d$ randomly and independently chosen elements, both types occur with probability > 0.99. On the other hand, the (short) list of subgroups containing both types of elements but not containing $\mathrm{SL}_d(q)$ is determined, and special tests are devised to eliminate them. This method was extended in [87] in a highly nontrivial way to the other classical groups. An alternative approach is described in [23]. Here, random elements of G are chosen, their order is determined, and groups of the Aschbacher classification which cannot contain elements of this order are eliminated.

The classical matrix groups can also be recognized *constructively*; the output is not only a proof that G is a classical group, but every element of G can be expressed effectively in terms of the given generating set. [22] handles the case $G \ge \mathrm{SL}_d(q)$ in the natural representation, [30] the case $G = \mathrm{SL}_d(2)$ in *any* representation, and [51] handles all classical groups, in any representation.

Concerning the other cases of the Aschbacher classification, the Meat-Axe (cf. Section 4) can be used to test irreducibility and absolute irreducibility of G. There are also programs computing imprimitivity and tensor decompositions [44], [56]. The exploration of groups beyond classifying them in one of the Aschbacher classes has also started. The implementations of the algorithms indicated in the previous three paragraphs can handle groups of dimension in the low hundreds, over moderately sized fields ($q < 2^{16}$).

Another line of attack is to try to construct permutation representations of subgroups of $\mathrm{GL}_d(q)$, while avoiding the blowup to q^d points. In [28], a permutation representation of Lyons' sporadic simple group Ly on 9.6 million points is constructed from $Ly \le \mathrm{GL}_{111}(5)$, while [76] describes general strategies to choose base points from the vector space. The theoretical paper

[11] handles *all* matrix groups, in time polynomial in (ν+ input size), where ν is the smallest number such that all composition factors of G have faithful permutation representations on at most ν points.

While randomization can often be used to speed up permutation group algorithms, it seems to be an indispensable tool for the study of matrix groups. The only exception so far is Luks' algorithm [66] for determining the structure of solvable matrix groups.

6 Black box groups

Algorithms usually try to exploit the specific features of the given representation of the input group. For example, for very large m, the fastest way to compute the m^{th} power of a permutation is to find the images of the cycles separately. For a cycle C of length c, $C^m = C^k$, where k is the remainder of m divided by c. On the other hand, the m^{th} power of a group element can be obtained by repeated squaring, using the binary decomposition of m, with $O(\log m)$ group operations in *any* representation of the input group.

Black box group algorithms are algorithms of the latter type. The elements of a black box group G are encoded as strings of length at most N over a finite alphabet Q, and group operations are performed by an oracle (the black box). The algorithm can use only the following operations: given $g, h \in G$, we can compute (a string representing) gh; g^{-1}; and decide whether $g = 1$. (This is a slight generalization of [10], where black box groups were first defined.) Note that we have an upper bound, e.g., $|G| < N|Q|^N$, on the order of G.

A natural example of black box groups are matrix groups $G \leq \mathrm{GL}_d(q)$, with $Q = \mathrm{GF}(q)$ and $N = d^2$, since there are algorithms for matrix groups which need only black box group operations. In fact, [11], [30], [51] work in this more general setting. Another important example, relevant for nearly linear algorithms (cf. Section 3), is permutation groups, with Q an SGS of G. The elements of G are represented as in (6), and then each coset representative is written as a word in the strong generators. It is possible to find an SGS Q such that $N = O(\log |G|)$ and then, using that the base images uniquely determine the elements of G, the black box group operations can be carried out in $O(\log^2 |G|)$ time. For small-base groups, this is faster than ordinary permutation multiplication. However, we lost all information

implicit in the cycle structure of permutations, and only the black box group operations can be carried out.

We saw in Section 5 that random elements are crucial for most matrix group algorithms. So far, all efforts to generate random elements in matrix groups are essentially black box group algorithms: they multiply previously available group elements. In theory, concerning polynomial time construction, the situation is satisfactory: Babai [4] gives a Monte Carlo algorithm which, after a preprocessing phase consisting of $O(\log^5 |G|)$ group operations, constructs independent, nearly uniformly distributed elements, at a cost of $O(\log |G|)$ group operations per random element. (Nearly uniform means that each element of G is selected with probability between $(1-\varepsilon)/|G|$ and $(1+\varepsilon)/|G|$ for some small ε.) In practice, the heuristic product replacement algorithm of [21] is used. This starts with a list $(g_1, g_2, ..., g_m)$ of generators; at each step, two indices i, j are selected randomly, and one of g_i, g_j is replaced by the product $g_i g_j$. After K replacements as preprocessing, we start to output the newly created elements of the list as random elements of G. It is shown in [21] that if m is at least twice the size of a minimal generating set of G then the limit distribution of the sequence $(g_1, g_2, ..., g_m)$ is uniform among all generating sequences of length m, and computational evidence shows that in applications for matrix groups of dimension in the low hundreds, K can be chosen around 100. Although the constructed random elements are not independent, the ratio of elements with properties relevant for the algorithms (e.g., the ppd property) seems in experimental tests to be close to the ratio of such elements in the group.

Another black box group method which has nice applications is the method of random subproducts, introduced in an early version of [9]. Given a list of group elements $L = (g_1, g_2, ..., g_m)$ of G, a *random subproduct* of L is $g_1^{\varepsilon_1} g_2^{\varepsilon_2} \cdot \ldots g_m^{\varepsilon_m}$, where the ε_i are chosen independently and uniformly from the set $\{0, 1\}$. Of course, random subproducts are very far from uniformly distributed random elements even if L consists of a generator list of G, but they can emulate certain properties of random elements successfully. Namely, a lot of algorithms use the following property of uniformly distributed random elements g: if a subgroup $K \lneq G$ then $g \notin K$ with probability at least $1/2$. It is easy to see that a random subproduct of generators has the same property, and there are surprisingly powerful applications [6], [29] of this simple observation. In practice, random subproducts are used in GAP to speed up even traditional permutation group algorithms such as normal closure.

7 Applications

We finish this survey by briefly mentioning some available data bases, applications of CGT, and standalone programs.

In the 19^{th} century, a central problem of group theory was the classification of groups of a given order. Currently, GAP contains a library of the 174366 groups of order at most 1000, but not 512 or 768 [14]. Among these, 56092 are of order 256 [89], and it is expected that there are more than a million groups of order 512.

The 42038 transitive permutation groups of degree at most 31 are also available [47]. In principle, this list can be used to compute Galois groups; the computations are practical up to degree 15. Non-affine primitive permutation groups are listed up to degree 1000 [33] and solvable permutation groups up to degree 255 [97]. The list of 4-dimensional space groups [15] and conjugacy class representatives of irreducible maximal finite subgroups of $\mathrm{GL}_n(\mathbb{Q})$ for $n \leq 24$ [77] is now also on-line. In Section 4, we already mentioned the availability of ordinary and modular character tables. Probably, just the sizes of these databases indicate the computational difficulties obtaining them.

The applications of CGT in group theory are too numerous to list here. We just mention one of them, the role of CGT in the construction of some of the sporadic finite simple groups [100], [101], [88].

Both GAP and MAGMA support the application of CGT to related fields such as other areas of abstract algebra, graph theory, and coding theory. MAGMA covers other areas of symbolic computation as well; it has a particularly strong number theory component. Finally, we mention two standalone programs, which may be used to investigate finitely presented groups. QUOTPIC [45] is a user-friendly graphical interface to display factor groups of fp-groups, while the system *Magnus*, in the beta-release stage, works with infinite fp-groups.

References

[1] William A. Alford, George Havas, and M.F. Newman. Groups of exponent four. *Notices Amer. Math. Soc.*, 22:A.301, 1975.

[2] M. Aschbacher. On the maximal subgroups of the finite classical groups. *Invent. Math.*, 76:469–514, 1984.

[3] Michael D. Atkinson, editor. *Computational Group Theory*, London, New York, 1984. (Durham, 1982), Academic Press.

[4] László Babai. Local expansion of vertex-transitive graphs and random generation in finite groups. In *Proc. 23rd ACM Sympos. Theory Comp.*, pages 164–174, 1991.

[5] László Babai, Robert Beals, and Daniel Rockmore. Deciding finiteness of matrix groups in deterministic polynomial time. In *Proc. of International Symposium on Symbolic and Algebraic Computation* ISSAC '93, pages 117–126. (Kiev), ACM Press, 1993.

[6] László Babai, Gene Cooperman, Larry Finkelstein, Eugene M. Luks, and Ákos Seress. Fast Monte Carlo algorithms for permutation groups. *J. Comp. Syst. Sci.*, 50:296–308, 1995.

[7] László Babai, Gene Cooperman, Larry Finkelstein, and Ákos Seress. Nearly linear time algorithms for permutation groups with a small base. In *Proc. of International Symposium on Symbolic and Algebraic Computation* ISSAC '91, pages 200–209. (Bonn), ACM Press, 1991.

[8] László Babai, Eugene M. Luks, and Ákos Seress. Fast management of permutation groups II. In preparation.

[9] László Babai, Eugene M. Luks, and Ákos Seress. Fast management of permutation groups I. *SIAM J. Computing*, 26, 1997, to appear.

[10] László Babai and Endre Szemerédi. On the complexity of matrix group problems, I. In *Proc. 25th IEEE Sympos. Foundations Comp. Sci.*, pages 229–240, 1984.

[11] Robert Beals and László Babai. Las Vegas algorithms for matrix groups. In *Proc. 34rd IEEE Sympos. Foundations Comp. Sci.*, pages 427–436, 1993.

[12] Robert M. Beals. An elementary algorithm for computing the composition factors of a permutation group. In *Proc. of International Symposium on Symbolic and Algebraic Computation* ISSAC '93, pages 127–134. (Kiev), ACM Press, 1993.

[13] Robert M. Beals and Ákos Seress. Structure forest and composition factors for small base groups in nearly linear time. In *Proc. 24th ACM Sympos. Theory Comp.*, pages 116–125, 1992.

[14] Hans Ulrich Besche and Bettina Eick. Construction of finite soluble groups. In preparation.

[15] Harold Brown, Rolf Bülow, Joachim Neubüser, Hans Wondratschek, and Hans Zassenhaus. *Crystallographic groups of four-dimensional space*. Wiley-Interscience, New York, Chicester, Brisbane, Toronto, 1978.

[16] G. Butler. *Fundamental Algorithms for Permutation Groups*, volume 559 of *Lecture Notes in Comput. Sci.* Springer-Verlag, Berlin, Heidelberg, New York, 1991.

[17] John Cannon, editor. *Computational Group Theory I*, volume 9(5–6) of J. Symbolic Comput., 1990.

[18] John Cannon, editor. *Computational Group Theory II*, volume 12(4–5) of J. Symbolic Comput., 1990.

[19] John Cannon and George Havas. Algorithms for groups. *Australian Computer Journal*, 24(2):51–60, 1992.

[20] John Cannon and Catherine Playoust. *An Introduction to* Magma. School of Mathematics and Statistics, Sydney University, 1993.

[21] Frank Celler, Charles R. Leedham-Green, Scott H. Murray, Alice C. Niemeyer, and E.A. O'Brien. Generating random elements of a finite group. *Comm. Algebra*, 23:4931–4948, 1995.

[22] Frank Celler and C.R. Leedham-Green. A constructive recognition algorithm for the special linear group. In preparation.

[23] Frank Celler and C.R. Leedham-Green. A non-constructive recognition algorithm for the special linear and other classical groups. In [38].

[24] Frank Celler, Joachim Neubüser, and Charles R. B. Wright. Some remarks on the computation of complements and normalizers in soluble groups. *Acta Appl. Math.*, 21:57–76, 1990.

[25] Michael Clausen and Ulrich Baum. *Fast Fourier Transforms.* Bibliographisches Institut & F. A. Brockhaus AG, Mannheim, 1993.

[26] S. B. Conlon. Calculating characters of p-groups. In [17], pages 535–550.

[27] J.H. Conway, R.T. Curtis, S.P. Norton, R.A. Parker, and R.A. Wilson. *Atlas of finite groups.* Clarendon Press, Oxford, 1985.

[28] G. Cooperman, L. Finkelstein, M. Tselman, and B. York. Constructing permutation representations for matrix groups. *J. Symbolic Comput.*, to appear.

[29] Gene Cooperman and Larry Finkelstein. Combinatorial tools for computational group theory. In [37], pages 53–86.

[30] Gene Cooperman, Larry Finkelstein, and Steven A. Linton. Recognizing $\mathrm{GL}_n(2)$ in non-standard representation. In [38].

[31] Gene Cooperman, Gerhard Hiss, Klaus Lux, and Jürgen Müller. The Brauer tree of the principal 19-block of the sporadic simple Thompson group. Manuscript.

[32] John D. Dixon. High speed computation of group characters. *Numer. Math.*, 10:446–450, 1967.

[33] John D. Dixon and Brian Mortimer. The primitive permutation groups of degree less than 1000. *Math. Proc. Camb. Phil. Soc.*, 103:213–238, 1988.

[34] Bettina Eick. Special presentations for finite soluble groups and computing (pre-)Frattini subgroups. In [38].

[35] D. B. A. Epstein et al. *Word Processing in Groups.* Jones and Bartlett, Boston, 1992.

[36] Xin Gui Fang and George Havas. On the worst-case complexity of Gaussian elimination. In *Proc. of International Symposium on Symbolic and Algebraic Computation* ISSAC '97. ACM Press, 1997.

[37] Larry Finkelstein and William M. Kantor, editors. *Groups and Computation*, volume 11 of *Amer. Math. Soc. DIMACS Series*. (DIMACS, 1991), 1993.

[38] Larry Finkelstein and William M. Kantor, editors. *Groups and Computation*, volume 28 of *Amer. Math. Soc. DIMACS Series*. (DIMACS, 1995), 1997.

[39] Steven P. Glasby and Michael C. Slattery. Computing intersections and normalizers in soluble groups. In [17], pages 637–651.

[40] George Havas. Coset enumeration strategies. In *Proc. of International Symposium on Symbolic and Algebraic Computation* ISSAC '91, pages 191–199. (Bonn), ACM Press, 1991.

[41] George Havas, Derek F. Holt, and Sarah Rees. Recognizing badly presented Z-modules. *Linear Algebra Appl.*, 192:137–163, 1993.

[42] George Havas, B. S. Majewski, and K. R. Matthews. Extended gcd algorithms. Technical report 302, The Univ. of Queensland, 1995.

[43] Derek Holt. The Warwick Automatic group software. In G. Baumslag, D. Epstein, R. Gilman, H. Short, and C. Sims, editors, *Geometric and Computational Perspectives on Infinite Groups*, volume 25 of *Amer. Math. Soc. DIMACS Series*, pages 69–82. (DIMACS, 1994), 1995.

[44] Derek F. Holt, C.R. Leedham-Green, E.A. O'Brien, and Sarah Rees. Testing matrix groups for primitivity. *J. Algebra*, pages 795–817, 1996.

[45] Derek F. Holt and Sarah Rees. A graphics system for displaying finite quotients of finitely presented groups. In [37], pages 113–126.

[46] Derek F. Holt and Sarah Rees. Testing modules for irreducibility. *J. Austral. Math. Soc. Ser. A*, 57:1–16, 1994.

[47] Alexander Hulpke. *Konstruktion transitiver Permutationsgruppen*. PhD thesis, RWTH Aachen, 1996.

[48] Christoph Jansen, Klaux Lux, Richard Parker, and Robert Wilson. *An Atlas of Brauer Characters*, volume 11 of *London Math. Soc. Monographs, (N. S.)*. Clarendon Press, Oxford, 1995.

[49] William M. Kantor. Sylow's theorem in polynomial time. *J. Comp. Syst. Sci.*, 30:359–394, 1985.

[50] William M. Kantor. Finding Sylow normalizers in polynomial time. *J. of Algorithms*, 11:523–563, 1990.

[51] William M. Kantor and Ákos Seress. Black box classical groups. Manuscript.

[52] D. E. Knuth and P. B. Bendix. Simple word problems in universal algebras. In [55], pages 263–297.

[53] B. Kostant. The graph of the truncated icosahedron and the last letter of galois. *Notices Amer. Math. Soc.*, 42:959–968, 1995.

[54] R. Laue, J. Neubüser, and U. Schoenwaelder. Algorithms for finite soluble groups and the SOGOS system. In [3], pages 105–135.

[55] J. Leech, editor. *Computational problems in abstract algebra*, Oxford, 1970. (Oxford, 1967), Pergamon Press.

[56] C.R. Leedham-Green and E.A. O'Brien. Recognising tensor products of matrix groups. *Internat. J. Algebra Comput.*, 1997, to appear.

[57] C.R. Leedham-Green and L.H. Soicher. Collection from the left and other strategies. In [17], pages 665–675.

[58] A. K. Lenstra, H. W. Lenstra, and László Lovász. Factoring polynomials with rational coefficients. *Math. Ann.*, 261:515–534, 1982.

[59] Jeffrey S. Leon. On an algorithm for finding a base and strong generating set for a group given by generating permutations. *Math. Comp.*, 20:941–974, 1980.

[60] Jeffrey S. Leon. Permutation group algorithms based on partitions, I: Theory and algorithms. In [18], pages 533–583.

[61] S.A. Linton. Double coset enumeration. In [18], pages 415–426.

[62] Eddie H. Lo. A polycyclic quotient algorithm. In [38].

[63] E. M. Luks, F. Rákóczi, and C. R. B. Wright. Computing normalizers in permutation p-groups. In *Proc. of International Symposium on Symbolic and Algebraic Computation* ISSAC '94, pages 139–146. (Oxford), ACM Press, 1994.

[64] Eugene M. Luks. Isomorphism of graphs of bounded valence can be tested in polynomial time. *J. Comp. Syst. Sci.*, 25:42–65, 1982.

[65] Eugene M. Luks. Computing the composition factors of a permutation group in polynomial time. *Combinatorica*, 7:87–99, 1987.

[66] Eugene M. Luks. Computing in solvable matrix groups. In *Proc.* 33*rd IEEE Sympos. Foundations Comp. Sci.*, pages 111–120, 1992.

[67] Eugene M. Luks. Permutation groups and polynomial-time computation. In [37], pages 139–175.

[68] Klaus Lux. Algorithmic Methods in Modular Representation Theory. Habilitationsschrift, RWTH Aachen, 1997.

[69] Klaus Lux and Herbert Pahlings. Computational aspects of representation theory of finite groups. In G. O. Michler and C. M. Ringel, editors, *Representation Theory of Finite Groups and Finite-Dimensional Algebras*, volume 95 of *Progress in Math.*, pages 37–64. Birkhäuser Verlag, 1991.

[70] I.D. Macdonald. A computer application to finite p-groups. *J. Austral. Math. Soc. Ser. A*, 17:102–112, 1974.

[71] B. H. Matzat. Konstruktion von Zahl- und Funktionenkörper mit vergegebener Galoisgruppe. *J. Reine und Angew. Math.*, 349:179–220, 1984.

[72] Brendan McKay. Nauty users guide (version 1.5). Technical report, Dept. Comp. Sci., Austral. Nat. Univ., 1990.

[73] M. Mecky and J. Neubüser. Some remarks on the computation of conjugacy classes of soluble groups. *Bull. Australian Math. Soc.*, 40:281–292, 1989.

[74] K. A. Mihailova. The occurence problem for direct products of groups (in Russian). *Dokl. Akad. Nauk. SSSR*, 119:1103–1105, 1958.

[75] Prabhav Morje. On nearly linear time algorithms for Sylow subgroups of small-base permutation groups. In [38].

[76] Scott H. Murray and E.A. O'Brien. Selecting base points for the Schreier-Sims algorithm for matrix groups. *J. Symbolic Comput.*, 19:577–584, 1995.

[77] G. Nebe and W. Plesken. Finite Rational Matrix Groups. *Memoirs Amer. Math. Soc.*, 116(556), 1995.

[78] J. Neubüser. An elementary introduction to coset table methods in computational group theory. In C. M. Campbell and E. F. Robertson, editors, *Groups – St Andrews* 1981, volume 71 of *London Math. Soc. Lecture Note Ser.*, pages 1–45. Cambridge University Press, 1982.

[79] J. Neubüser. An invitation to computational group theory. In C. M. Campbell, T. C. Hurley, E. F. Robertson, S. J. Tobin, and J. J. Ward, editors, *Groups'93 – Galway/St. Andrews*, volume 212 of *London Math. Soc. Lecture Note Ser.*, pages 457–475. Cambridge University Press, 1995.

[80] J. Neubüser, H. Pahlings, and W. Plesken. CAS; design and use of a system for the handling of characters of finite groups. In [3], pages 145–183.

[81] Peter M. Neumann. Some algorithms for computing with finite permutation groups. In E.F. Robertson and C.M. Campbell, editors, *Proc. of Groups–St Andrews* 1985, volume 121 of *London Math. Soc. Lecture Note Ser.*, pages 59–92. Cambridge University Press, 1987.

[82] Peter M. Neumann and Cheryl E. Praeger. A recognition algorithm for special linear groups. *Proc. London Math. Soc.* (3), 65:555–603, 1992.

[83] M.F. Newman. Calculating presentations for certain kinds of quotient groups. In *Proc. ACM Symposium on Symbolic and Algebraic Computation* SYMSAC '76, pages 2–8. (New York), ACM Press, 1976.

[84] M.F. Newman and E.A. O'Brien. Application of computers to questions like those of Burnside, II. *Internat. J. Algebra Comput.*, 6:593–605, 1996.

[85] Werner Nickel. Computing nilpotent quotients of finitely presented groups. In G. Baumslag, D. Epstein, R. Gilman, H. Short, and C. Sims, editors, *Geometric and Computational Perspectives on Infinite Groups*, volume 25 of *Amer. Math. Soc. DIMACS Series*, pages 175–191. (DIMACS, 1994), 1995.

[86] Alice C. Niemeyer. A soluble quotient algorithm. *J. Symbolic Comput.*, 18:541–561, 1994.

[87] Alice C. Niemeyer and Cheryl E. Praeger. A recognition algorithm for classical groups over finite fields. *Proc. London Math. Soc.*, 1997, to appear.

[88] Simon Norton. The construction of J_4. In Bruce Cooperstein and Geoffrey Mason, editors, *Proc. Santa Cruz Conference on Finite Groups*, pages 271–277. Amer. Math. Soc., 1980.

[89] E.A. O'Brien. The groups of order dividing 256. *Bull. Austral. Math. Soc.*, 39:159–160, 1989.

[90] R.A. Parker. The computer calculation of modular characters (the Meat-Axe). In [3], pages 267–274.

[91] W. Plesken. Towards a soluble quotient algorithm. *J. Symbolic Comput.*, 4:111–122, 1987.

[92] Lajos Rónyai. Computing the structure of finite algebras. *J. Symbolic Comput.*, 9:355–373, 1990.

[93] A. J. E. Ryba. Computer condensation of modular representations. In [17], pages 591–600.

[94] Gerhard J. A. Schneider. Dixon's character table algorithm revisited. In [17], pages 601–606.

[95] Martin Schönert *et al.* GAP – *Groups, Algorithms and Programming.* Lehrstuhl D für Mathematik, RWTH, Aachen, 1994.

[96] Ákos Seress. Nearly linear time algorithms for permutation groups: an interplay between theory and practice. *Acta Appl. Math.*, 1997, to appear.

[97] M.W. Short. *The Primitive Soluble Permutation Groups of Degree less than* 256, volume 1519 of *Lecture Notes in Math.* Springer-Verlag, Berlin, Heidelberg, New York, 1992.

[98] Charles C. Sims. Computational methods in the study of permutation groups. In [55], pages 169–183.

[99] Charles C. Sims. Computation with permutation groups. In *Proc. Second Symp. on Symbolic and Algebraic Manipulation*. ACM Press, 1971.

[100] Charles C. Sims. The existence and uniqueness of Lyons' group. In T. Hagen, M. P. Hale., and E. E. Shult, editors, *Finite Groups '72*, pages 138–141. North Holland, 1973.

[101] Charles C. Sims. How to construct a baby monster. In M. J. Collins, editor, *Finite simple groups II*, pages 339–345. (Durham 1978), Academic Press, 1980.

[102] Charles C. Sims. Implementing the Baumslag-Cannonito-Miller Polycyclic Quotient Algorithm. In [17], pages 707–723.

[103] Charles C. Sims. *Computation with finitely presented groups*. Cambridge University Press, 1994.

[104] Michael J. Smith. *Computing automorphisms of finite soluble groups*. PhD thesis, Australian National University, 1994.

[105] A. Storjohann and G. Labahn. Asymptotically fast computation of Hermite normal forms of integer matrices. In *Proc. of International Symposium on Symbolic and Algebraic Computation* ISSAC '96, pages 259–266. (Zürich), ACM Press, 1996.

[106] Heiko Theißen. *Eine Methode zur Normalisatorberechnung in Permutationsgruppen mit Anwendungen in der Konstruktion primitiver Gruppen*. PhD thesis, RWTH Aachen, 1997.