

A unified approach to computations with permutation and matrix groups

Ákos Seress*

Abstract. We survey algorithms to compute with large finite permutation and matrix groups. Particular attention will be given to handling both types of groups with similar methods, using structural properties to answer even basic questions such as the order of the input group.

Mathematics Subject Classification (2000). Primary 20B40; Secondary 20B15.

Keywords. Computational group theory, permutation group algorithm, matrix group algorithm.

1. Introduction

There are two basic methods to input a group into a computer: (a) by a presentation, using abstract generators and relations, and (b) by a “concrete” representation as a permutation group or matrix group, defined by a set of generating permutations or matrices. In this survey, we are concerned with groups given as in (b), and with black-box groups (see Definition 3.1), which are a common generalization of permutation and matrix groups. We shall concentrate on the basic questions how to determine the order of the input group G , how to set up a data structure to test membership in G , and how to compute a composition series. For readers interested in a broader range of topics, we recommend our brief survey [47] describing all areas of computational group theory and the more thorough coverage in the recent book [29]. Two monographs providing a comprehensive coverage of the subareas finitely presented groups and permutation groups are [51] and [48], respectively.

Given a set X of permutations or of invertible matrices over a finite field, X generates a finite group G so the undecidability issues related to generator-relator presentations and to infinite matrix groups do not arise. However, $|G|$ can be exponentially large in terms of the input length, so brute-force methods like listing all elements of G are out of question and we have to design efficient algorithms to deal with G .

There are two widely accepted notions of efficiency. On one hand, in practice, it means that we obtain results in reasonable time in the actual computations we perform. On the other hand, in theory, efficiency means fast asymptotic running time. Historically, group computations developed on these two tracks separately,

*Partially supported by the NSA and the NSF.

but in the last fifteen years or so the two approaches have started to converge. This convergence is not surprising. As we deal with larger and larger inputs, only those practical methods that are asymptotically efficient survive; conversely, the rigorous complexity analysis inspires new algorithms that may have practical implementations. This unification of theory and practice is one of the aspects the title of this paper refers to. Implementations of asymptotically fast algorithms are finding their way into *GAP* [28] and *MAGMA* [19], the two large computer algebra systems for group computations.

There are two other aspects of unification. One of them is within the matrix group setting. As we shall discuss in Section 3, there are two approaches to matrix group computations and we shall mention the recent efforts to combine them. The other aspect is the uniform treatment of permutation groups and matrix groups, by breaking them into manageable pieces as the image and kernel of appropriate group homomorphisms. This approach is the standard one for matrix groups, but a recently developed data structure enables us to handle both types of groups the same way.

The most recent ICM talk about computational group theory was given eight years ago, by Bill Kantor [30]. His major emphasis was the use of the classification of finite simple groups (CFSG) in the topic and the handling of simple groups. We shall also report the latest developments in simple group management, but this paper is more focused on the *reduction* to the simple group case. There are interesting and deep mathematical problems in both subareas, although the management of simple groups requires mostly group theoretical arguments while the reduction to the simple group case uses a mixture of group theoretic, combinatorial and computer science (design of data structures) methods. Consequences of CFSG are required in the analysis of many reduction algorithms as well.

2. Permutation groups

The fundamental data structures for computation with permutation groups were introduced by Sims [50]; they are called base and strong generating set (SGS). A *base* of $G \leq \text{Sym}(\Omega)$ is a sequence of points $B = (\beta_1, \dots, \beta_m)$ from Ω such that the pointwise stabiliser $G_B = 1$. A base B naturally defines a subgroup chain

$$G = G^{[1]} \geq G^{[2]} \geq \dots \geq G^{[m]} \geq G^{[m+1]} = 1$$

where $G^{[i]} := G_{(\beta_1, \dots, \beta_{i-1})}$ is the pointwise stabilizer of $\{\beta_1, \dots, \beta_{i-1}\}$. The base is called *non-redundant* if $G^{[i+1]}$ is a proper subgroup of $G^{[i]}$ for all $i \leq m$. For a non-redundant base B , we have $\log |G| / \log N \leq |B| \leq \log |G|$, where $|\Omega| = N$. (As usual in complexity theory, we write logarithms to base 2.)

A *strong generating set* (SGS) for G relative to B is a generating set S for G with the property that

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

Given $G = \langle X \rangle \leq \text{Sym}(\Omega)$, Sims's algorithm constructs a non-redundant base B and an SGS S relative to B . Once S is known, it is easy to construct (right) transversals T_i

for $G^{[i]} \bmod G^{[i+1]}$. Crucially, $|T_i| \leq N$ is “small”. For all γ in the orbit $\beta_i^{G^{[i]}}$, the transversal T_i contains some $r_\gamma \in G^{[i]}$ with $\beta_i^{r_\gamma} = \gamma$. These transversals can be used to compute $|G| = \prod_{i=1}^m |T_i|$ and to factor any $g \in G$ as a product $g = r_m \dots r_1$, for some $r_i \in T_i$. This factorization is unique and it can be done by an efficient algorithm called *sifting*. First, we take $r_1 \in T_1$ such that $\beta_1^{r_1} = \beta_1^g$. Then $g_2 := gr_1^{-1} \in G^{[2]}$, and we can take $r_2 \in T_2$ such that $\beta_2^{g_2} = \beta_2^{r_2}$, etc. Sifting can be used to test membership in G . For details, we refer to [48, Ch. 4].

Sims’s algorithm is based on elementary group theory. The running time of the asymptotically fastest versions is $O(|X|N^2 \log^c |G|)$ for some absolute constant c . A factor N can be shaved off the running time by randomization:

Theorem 2.1 ([7]). *Given $G = \langle X \rangle \leq \text{Sym}(\Omega)$ with $|\Omega| = N$, a base B and an SGS relative to B can be computed in $O(|X|N \log^c |G|)$ time, by a Monte Carlo algorithm with an arbitrarily small positive but fixed upper bound on the probability of incorrect output.*

Recall that a randomized algorithm is called *Monte Carlo* if there is a chance of an incorrect output but an upper bound for the probability of error can be prescribed by the user. On the contrary, a *Las Vegas* algorithm never returns an incorrect answer but it may report failure with probability bounded by the user.

The algorithm in Theorem 2.1 is still elementary. The quadratic $O(N^2)$ bottlenecks are broken by randomization, and by combinatorial tricks like working with base images instead of full permutations or to test membership in certain large subsets of G without listing those subsets. If $\log |G|$ is bounded from above by a polylogarithmic, $\log^c N$, function of N then the running time is a *nearly linear*, $O((|X|N) \log^c(|X|N))$, function of the input length $|X|N$. This motivated the following definition. An infinite family \mathcal{G} of permutation groups is called *small-base* if every group $G \in \mathcal{G}$ of degree m satisfies $\log |G| < \log^c m$ for some fixed constant c . Important families of groups are small-base, including all permutation representations of non-alternating simple groups.

The algorithm in Theorem 2.1 is also practical. In *GAP*, currently permutation group computations are based on an implementation of this algorithm. The certainty of a correct answer, if desired, is obtained by a quadratic algorithm of Sims that checks the correctness of a base and SGS (see [48, Section 8.2]).

For arbitrary inputs, where $\log |G|$ may become comparable to N , no deterministic version of Sims’s algorithm is known to run faster than $O(N^5 + |X|N^2)$. Fortunately, ideas from a purely theoretical development come to the rescue. In [9], an algorithm is described to handle permutation groups in the parallel computational model NC. Informally, in NC we can work with polynomially many, n^c , processors, but we have only polylogarithmic, $\log^c n$, time in terms of the input length n . Note that sifting is inherently sequential (we have to know the coset representatives r_1, \dots, r_i before r_{i+1} can be computed in a factorization $g = r_m \dots r_1$) so a Sims-based approach may work in NC only for small-base groups. The algorithm in [9] is based on entirely different

principles, exploring the structure of the input group G . By the time it computes $|G|$, it also obtains a composition series for G . Some of these ideas were also used in the more realistic domain of sequential computations to break the long-standing $O(N^5)$ barrier:

Theorem 2.2 ([10]). *Given $G = \langle X \rangle \leq \text{Sym}(\Omega)$ with $|\Omega| = N$, there is a deterministic algorithm with $O(N^4 \log^c N + |X|N^2)$ running time to compute $|G|$ and to set up a data structure for testing membership in G .*

The algorithm in Theorem 2.2 detects all large alternating composition factors of G and handles them by special methods, while the rest of the group is handled using Sims's ideas.

How can we detect large alternating sections in a permutation group? Combinatorial reduction (action on orbits, and then action on blocks of imprimitivity) leads to primitive permutation groups. At that point, we invoke a consequence of CFSG [20]: any primitive permutation group $H \leq \text{Sym}(\Delta)$ of degree n is a small-base group, unless $n = \binom{m}{k}^r$ for some positive integers m, k, r and Δ can be identified with r -tuples of k -sets of an m -element set, $A_m^r \leq H \leq S_m \wr S_r$, and H acts naturally on these sequences in the so-called product action of wreath products. Such an H is called a *group of Cameron type* in [10]. In this very special situation, [10] gives a combinatorial algorithm to construct a collection Σ of mr subsets of Δ so that H acts on Σ in the natural imprimitive action of wreath products.

The input group G is handled by a recursive procedure. We define a homomorphism $\varphi: G \rightarrow \text{Sym}(\Delta)$ for some Δ , process $\text{Im}(\varphi)$, obtain generators for $\text{Ker}(\varphi)$, and process $\text{Ker}(\varphi)$. If G is transitive then Δ is an orbit and $\text{Im}(\varphi)$ is the restriction of G to this orbit; and if G is transitive but imprimitive then Δ is a block system and $\text{Im}(\varphi)$ is the action on this block system. When the recursion arrives to a primitive group then we test whether it is of Cameron type. If not, then Sims's base-SGS method is used to process it. If it is of Cameron type then a further homomorphism is constructed to the natural imprimitive action. The full alternating and symmetric groups A_m and S_m , encountered in their natural action on m points, are handled by combinatorial methods, as a special case of the constructive recognition of almost simple groups (see Section 3.1).

We finish this section by announcing a Las Vegas upgrade of Theorem 2.1.

Theorem 2.3 ([32], [33]). *Given $G = \langle X \rangle \leq \text{Sym}(\Omega)$ with $|\Omega| = N$ and G having no composition factors of Lie type 2G_2 and 2F_4 , a base and SGS for G can be computed in $O(|X|N \log^c |G|)$ time, by a Las Vegas algorithm with an arbitrarily small but fixed positive upper bound on the probability of incorrect output.*

Although the statements of Theorems 2.1 and 2.3 are similar, the proofs are based on entirely different principles. The algorithm in Theorem 2.3 computes a composition series for G by a Monte Carlo algorithm, recognizes constructively the composition factors (see Section 3.1), and then uses the isomorphisms set up by the recognition algorithms to write a presentation for G . Evaluation of this presentation verifies

the correctness of the entire computation. The groups ${}^2G_2(q)$ have to be excluded because currently it is not known that they have short presentations suitable for the time requirement of this application; the groups ${}^2F_4(q)$ are excluded because the known constructive recognition algorithm is not fast enough.

There is a large library of Monte Carlo algorithms that run in nearly linear time for small-base inputs (see [48, Ch. 5 and 6]). The significance of Theorem 2.3 is that if the initial base-SGS computation is correct for some input group G then all of these nearly linear-time algorithms are *automatically upgraded to Las Vegas*.

Summarizing, we saw that the basic tasks of finding the order and setting up membership testing in permutation groups can be performed by elementary methods in polynomial time, but randomization and the structural exploration of the input group provide much faster algorithms.

3. Matrix groups

The basic problems for matrix groups over finite fields, such as membership and order, seem to be much harder than the corresponding problems for permutation groups. The fundamental difference is that there is no longer, in general, a decreasing sequence of subgroups from G to 1 in which all successive indices are small; this makes an analogue of Sims' base-SGS approach infeasible. For permutation groups, the natural divide-and-conquer approach leads to primitive groups, and those groups can be reduced to symmetric groups or else they are small-base groups. In contrast, a large variety of primitive irreducible matrix groups has order $\exp(\Omega(d^2))$ (here d is the dimension of the matrices). Finally, even for 1×1 matrices, the problems are closely related to discrete logarithm computations.

For later use, we define two versions of the discrete logarithm problem:

(DL1) Given $a, b \in \text{GF}(q)^*$, determine whether $a \in \langle b \rangle$.

(DL2) Given $a, b \in \text{GF}(q)^*$, determine whether $a \in \langle b \rangle$. If the answer is yes then find an exponent x such that $a = b^x$.

Finding the order of $G = \langle X \rangle \leq \text{GL}(1, q)$ is between these two problems in difficulty: version (DL1) can be reduced to it in polynomial time, while it can be reduced to version (DL2) in polynomial time. We note that neither version of the discrete logarithm problem has at present a polynomial-time solution, although subexponential algorithms exist even for the more difficult version (DL2) [39].

Despite all the difficulties listed above, significant progress has been made recently on matrix groups and associated data structures, and currently this is the most active area of computational group theory. However, contrary to the permutation group case, it seems that randomization and a full structural exploration of the input is not only a speedup, but an essential and unavoidable tool. This means that we have to set up a

recursive scheme of homomorphisms, breaking the input into the image and kernel. This reduction bottoms out at matrix groups H that are almost simple modulo scalars. At these terminal stages of the recursion, we have to find the name of the isomorphism type of H , and then set up an identification with a standard permutation or matrix representation of this isomorphism type.

First, we discuss the methods for handling almost simple groups. For that, we need two definitions.

Definition 3.1. A *black-box group* is a group whose elements are encoded as words of length at most N over some alphabet T and some bound N . Not every word represents a group element and the same group element may be represented by more than one word. Moreover, an oracle (the “black box”) performs the following three operations: given (words representing) $g, h \in G$, it can compute (a word representing) gh, g^{-1} , and it can decide whether $g = 1$.

Our definition is slightly more general than the original one [11], where only 0-1 strings of uniform lengths are allowed. The primary examples of black-box groups are permutation groups and matrix groups, but there are two other important examples. One of them is a power-conjugate presentation for a finite solvable group, where each group element has a canonical form $a_1^{e_1} a_2^{e_2} \dots a_m^{e_m}$ for a suitable generating sequence (a_1, \dots, a_m) . More important for our present discussion is that permutation groups G can be considered as black-box groups where the alphabet is an SGS for G (see [48, Section 5.3]). In small-base groups, group operations using words in the strong generators are asymptotically much faster than permutation multiplications. These special types of black-box groups play an important role, for example, in the proof of Theorem 2.3. In these black-box groups, we lose all information stored implicitly in the cycle structure of permutations, and algorithms can utilize only the three black-box operations defined above.

In some situations, we also consider permutation and matrix groups with their natural group operations as black-box groups, because permutation group theoretic notions like orbits or cycle structure, or geometric notions like invariant subspaces or characteristic polynomials in the matrix group case, do not help. For example, we have no better methods for generating random elements in a matrix group than creating new group elements from the given generators by multiplications and inversions, that is, by black-box operations [4], [22]. Black-box group algorithms, with the natural permutation operations, are also used in computations of normal closures, derived series, and related algorithms both in theory [23], [6] and in *GAP*.

The second definition we require is of a straight-line program (SLP). It is a data structure to circumvent problems with overly long words in generators.

Definition 3.2. Given $G = \langle X \rangle$, a *straight-line program of length m* reaching some $g \in G$ is a sequence of expressions (w_1, \dots, w_m) such that for each i one of the following holds: w_i is a symbol for some element of X , $w_i = (w_j, -1)$ for some $j < i$, or $w_i = (w_j, w_k)$ for some $j, k < i$, such that, if the expressions are evaluated,

then the value of w_m is g . Here, $(w_j, -1)$ is evaluated as the inverse of the evaluated value of w_j , and (w_j, w_k) is evaluated as the product of the evaluated values of w_j and w_k .

3.1. Recognition of almost simple groups. Now we are ready to discuss matrix groups that are almost simple modulo scalars or, more generally, almost simple black-box groups. There are two basic tasks: non-constructive recognition and constructive recognition. *Non-constructive recognition* of an almost simple group means to name the isomorphism type. The first such algorithm is in [41], where it is decided whether a given group $G \leq \text{GL}(d, p^e)$ contains $\text{SL}(d, p^e)$. A sample of random elements is taken, and we look for elements whose order is divisible by some primitive prime divisor (ppd) of $p^{de} - 1$ and of $p^{(d-1)e} - 1$. (Recall that a *primitive prime divisor* of $p^n - 1$ is a prime $r \mid p^n - 1$ which does not divide $p^i - 1$ for any $i < n$.) If $G \geq \text{SL}(d, p^e)$ then elementary estimates show that both kinds of ppd's occur frequently enough so that a small sample of random elements detects them; however, CFSG is invoked to prove that if both kinds of ppd's occur then indeed $G \geq \text{SL}(d, p^e)$. Subsequently, similar algorithms were designed to recognize the other classical groups in their natural matrix representations [21], [43]. The culmination of this type of results is in [8]: given an almost simple black-box group G of Lie type and given the characteristic p of G , the isomorphism type of G can be computed. This algorithm still looks for elements whose order is divisible by various ppd's and pairs of ppd's. We note that the ppd property can be checked in the black-box group setting, without computing element orders. If we know only that a matrix group is simple of Lie type modulo scalar matrices, its characteristic can be determined by recent algorithms in [38] and [49]. All algorithms mentioned in this paragraph are Monte Carlo with polynomial running time, and have efficient implementations.

In applications in recursive schemes breaking down arbitrary matrix groups to almost simple pieces, non-constructive recognition is not sufficient; we need the more involved constructive recognition. Given an almost simple black-box group $G = \langle X \rangle$, *constructive recognition* of G is a Las Vegas algorithm that, besides naming the isomorphism type of G , computes an isomorphism $\varphi: G \rightarrow C$ with a standard permutation representation or (projective) matrix representation of this isomorphism type. The isomorphism φ is defined by giving the images of a new generating set $Y \leq G$. Moreover, we require that, given any $g \in G$, a short SLP reaching g from Y can be computed, and given any $h \in C$, $\varphi^{-1}(h)$ can be computed.

The first constructive recognition algorithm, for black-box groups $\text{GL}(n, 2)$, was given in [24]. Subsequently, constructive recognition of all classical groups (in [32]) and exceptional groups G (in [31]) was accomplished. These algorithms require the characteristic p of G as part of the input. The rough idea is the following. Since in the black-box setting we do not have a vector space to work with, the algorithms construct a large elementary abelian p -section P of G such that the conjugation action of a maximal parabolic $H \leq G$ on P is isomorphic to the natural matrix action of H , and subsequently extend this matrix action to arbitrary elements of G . Constructive

recognition of alternating and symmetric groups is much easier [12], [13], [14]. A recent algorithm [2] recognizes constructively about half of the sporadic groups, using a generalization of Sims's sifting through subset chains instead of subgroup chains.

An exciting new method by Ryba toward the constructive recognition of some Lie-type groups is described in [45] and [46]. Let p be an odd prime, and let G be a group of untwisted Lie type defined over a field of characteristic p . Suppose further that the associated Lie algebra of G is simple. Then, given any absolutely irreducible characteristic p representation $G = \langle X \rangle$, a polynomial-time Las Vegas algorithm computes the action of the generator set X on the Chevalley basis of the Lie algebra of G . Hence the constructive recognition problem is reduced to consideration of the adjoint representation.

Ryba is currently working on the extension of this algorithm to all Lie-type groups. Although his methods are still under development, they seem to have the potential to become the major tool of constructive recognition, reducing the use of the methodology of [32], [31] only to the natural and cross-characteristic representations of Lie-type groups.

The running times of the algorithms in [32], [31] are polynomials in the rank r and the defining field size q of the Lie-type input group G . However, the length of the input may be only $O(r^2 \log q)$, so for large q the running time is exponential. An idea to overcome this difficulty is in [25], where the groups $SL(2, q)$ in their standard 2×2 matrix setting are recognized in polynomial time of the input length *plus* polynomially many calls to an oracle solving version (DL2) of the discrete logarithm problem in $GF(q)$. Later this algorithm was extended to arbitrary matrix representations of $PSL(2, q)$ [26]. This motivated the following definition.

Definition 3.3. Let G be an almost simple group of Lie type defined over the field $GF(p^l)$. We say that G is *constructively recognizable with a discrete logarithm oracle*, in short G is *CRDLO*, if for *any* quasisimple representation of G in characteristic p , G can be constructively recognized in time polynomial in the input length plus the time of polynomially many calls to a discrete logarithm oracle in $GF(p^l)$. (Note that the field of definition $GF(p^l)$ may be different from the field $GF(p^e)$ over which the input matrices are given. Recall that a group G is called *quasisimple* if $G/Z(G)$ is simple and G equals its derived subgroup.)

Theorem 3.4 ([16], [15], [17]). *All classical groups are CRDLO.*

The algorithms of this theorem are based on [32], using an oracle to handle $SL(2, q)$ subgroups. In turn, this oracle is based on the methods of [26]. The case of special linear, symplectic, and unitary groups is a more or less straightforward modification of [32], but the case of orthogonal groups involves significant additional technical difficulties.

3.2. The general case. Now we turn to the case of arbitrary matrix groups. There are two basic methods for the breakup of the input into manageable pieces (which,

in most cases, amounts to the reduction to almost simple groups). The *geometric approach*, summarized in [37], is based on Aschbacher's classification of matrix groups [3]. This classification defines eight types of geometric subgroups of $GL(d, q)$, and groups $G \leq GL(d, q)$ belonging to seven of these types have a naturally associated $N \triangleleft G$ which enables the recursive handling of G/N and N . These classes consist of reducible groups, imprimitive groups, normalizers of extraspecial groups, and so on. For example, in the case of reducible groups we can consider the homomorphism defined by the restriction to the action on an invariant subspace, and the kernel of this action. The eighth geometric category contains the classical groups in their natural representation. The groups not belonging to any of the eight geometric categories are almost simple modulo scalars. After contributions by many people (see [44] for an overview), O'Brien has a working implementation of the reduction to the almost simple case.

By contrast, the *black-box group approach*, initiated by Babai and Beals [12], tries to determine the abstract group-theoretic structure of G . Every finite group G has a series of characteristic subgroups $1 \leq M_1 \leq M_2 \leq M_3 \leq G$, where M_1 is solvable, M_2/M_1 is isomorphic to a direct product $T_1 \times \cdots \times T_k$ of nonabelian simple groups, M_3/M_2 is solvable, and G/M_3 is a permutation group, permuting the simple groups T_i . Given $G = \langle X \rangle \leq GL(d, p^e)$, [5] constructs subgroups H_1, \dots, H_k such that $H_i/S_i \cong T_i$ for some solvable group S_i . Having these H_i at hand, it is possible to construct the permutation group $G/M_3 \leq S_k$, which then can be handled by permutation group methods. Moreover, using the results of [1], [8], the simple groups T_i can be non-constructively recognized.

The Babai–Beals algorithm and its extension by [1], [8] are Monte Carlo, and run in polynomial time in the input length.

Contrary to the geometric approach, [5] does not use the geometry associated with the matrix group action of G . The fact that $G \leq GL(d, p^e)$ is only used when appealing to a simple consequence of [35], [27]: if T_i is of Lie type in characteristic different from p , then T_i has a permutation representation of degree polynomial in d .

These results can be extended significantly further.

Theorem 3.5 ([34]). *Given $G = \langle X \rangle \leq GL(d, p^e)$, there is a Las Vegas algorithm that computes the following.*

- (i) *The order of G .*
- (ii) *A series of subgroups $1 = N_0 \triangleleft N_1 \triangleleft \cdots \triangleleft N_{m-1} \triangleleft N_m = G$, where N_i/N_{i-1} is a nonabelian simple group or a cyclic group for all i .*
- (iii) *A presentation of G .*
- (iv) *Given any $g \in GL(d, p^e)$, the decision whether $g \in G$, and if $g \in G$, then a straight-line program from X , reaching g .*

The algorithm uses an oracle to solve version (DL2) of the discrete logarithm problem in fields of characteristic p and size up to p^{ed} . In the case when all composition factors of Lie type that are in characteristic p are CRDLO, the running time

is polynomial in the input length $|X|d^2e \log p$, plus the time requirement of polynomially many calls to the discrete logarithm oracle. Note that in (ii) the cyclic factors N_i/N_{i-1} may not be simple of prime order because we do not assume that we can factor large integers.

The proof proceeds by continuing the Babai–Beals algorithm when that approach bottoms out. The key idea is that, using the notation introduced in the discussion before Theorem 3.5, for those simple groups T_i that are of Lie type of characteristic p , H_i can be written in an appropriate basis in an upper triangular 3×3 block matrix form such that T_i acts in a quasisimple representation on the block $(2, 2)$. Hence the CRLDO constructive recognition algorithms can be applied. Finally, the subgroup M_1 is handled using a modification of Luks’s deterministic algorithm [40] for solvable matrix groups.

The algorithms of [5] and [34] are not practical. However, [5] is a cookie jar of new ideas, which should be used in implementations. Hence, we recently started a project of designing new reduction algorithms for those Aschbacher categories where the current algorithms do not have fast asymptotic running time, combining geometric and black-box methods. This is the second level of unification mentioned in the introduction. Although this project is quite new and there is only one paper [18] in print (about the category of normalizers of extraspecial groups), algorithms for three other categories (imprimitive, tensor product, and tensor induced) are in the offing.

4. A new data structure

In this section we discuss an implementation aspect of computations with permutation and matrix groups. As we have seen in the previous sections, the asymptotically most efficient permutation group algorithms and all existing matrix group algorithms break up the input into manageable pieces. Hence, in order to implement these algorithms, we need a data structure for a recursive scheme which facilitates divide-and-conquer techniques to pass from a group G to a normal subgroup N and the factor group G/N , and then to put together the results of those two smaller computations.

In [42] we describe the design and implementation of such a data structure. This data structure opens up the possibility to handle theoretical algorithms that were considered too complicated for implementation. The homomorphism mechanism is on the black-box level, so permutation groups and matrix groups can be treated in a uniform way. Also, for any group occurring in the recursive scheme, the image of the homomorphism may be either a permutation, matrix, or black-box group, so we can switch between the different types as best suited for the particular application. Once a homomorphism $\varphi: H \rightarrow K$ is defined for some inner node H of the recursion tree, the computation of $\text{Ker}(\varphi)$ and the combination of the results for $\text{Im}(\varphi)$ and $\text{Ker}(\varphi)$ are done by generic procedures, so in applications we can concentrate on finding suitable homomorphisms φ and the handling of the leaf nodes (which usually means constructive recognition of that node).

The first success story is the implementation of a randomized version of the algorithm of Theorem 2.2. All that was needed was the design and implementation of a randomized speedup for processing groups of Cameron type [36], and an implementation of constructive recognition of alternating and symmetric groups in their natural representation, as described in [48, Section 10.2.4]. The rest of the algorithm of Theorem 2.2 is done automatically by the generic recursive procedure. The final result is the first practical treatment of *all* permutation groups: for small-base inputs, the algorithm reverts to the base-SGS method with minimal overhead (and sometimes it runs faster even on small-base inputs), and on larger inputs there are very substantial savings compared to the straightforward call of the current default base-SGS computation.

The mathematical idea behind the recursion scheme is simple, but there were formidable challenges in the design of the data structure. To pull back the results from N and G/N to G , we need new data types, permutations and matrices with memory, that “remember” how they were obtained from the generators of G by storing a straight-line program (SLP). This results in conflicting requirements. On one hand, these new data types must behave like permutations or matrices, so the *existing and future permutation and matrix group algorithms can be applied to them without rewriting the library of GAP functions* and we can incorporate permutation and matrix group algorithms by other developers who may not need to know of our recursive scheme. On the other hand, the steps of the applied permutation and matrix group algorithms must be recorded in the SLP, although these algorithms are not even aware that this SLP exists. We also need a new type of homomorphism template, flexible enough to accommodate the wide variety of methods that can create factor groups. Exploring an unknown G , we may try a lot of different methods to pass to a factor group; these methods must be prioritized by an automatic method selection mechanism, while at the same time this mechanism must be transparent enough for users to change the order in which applicable methods are called if some extra information is known or suspected about G .

There is a long list of novel tricks incorporated in the new framework: for example, how to balance the recursive tree (that the branches are about the same length, speeding up traversing the tree); how to pass information to the children of a node, so each node can have its own individualized method selection process for a more efficient way to find a homomorphism from this node; and the introduction of an analogue of strong generating sets in the matrix group setting, which enables the writing of shorter straight-line programs to reach group elements. Current work concentrates on adding new homomorphism methods to the recursive scheme for the geometric subgroups in Aschbacher’s classification, combining black-box and geometric methods as mentioned at the very end of the previous section, and implementing methods for the end nodes of recursion, to handle almost simple matrix and black-box groups. There will also be methods to handle solvable groups given by power-conjugate presentations, thereby including the third large category of black-box groups in the same scheme. I am quite enthusiastic about this new framework: I think we have found the tool for

the uniform treatment of permutation, matrix, and solvable groups, at the same time unifying the theoretical and practical sides of computations with these groups.

Acknowledgement. I am indebted to Bill Kantor for his very helpful comments.

References

- [1] Altseimer, C., Borovik, A. V., Probabilistic recognition of orthogonal and symplectic groups. In *Groups and Computation III* (ed. by W. M. Kantor, Á. Seress), Ohio State Univ. Math. Res. Inst. Publ. 8, Walter de Gruyter, Berlin, New York 2001, 1–20.
- [2] Ambrose, S., Neunhöffer, M., Praeger, C. E., Schneider, C., Generalised sifting in black-box groups. *London Math. Soc. J. Comput. Math.* **8** (2005), 217–250.
- [3] Aschbacher, M., On the maximal subgroups of the finite classical groups. *Invent. Math.* **76** (1984), 469–514.
- [4] Babai, L., Local expansion of vertex-transitive graphs and random generation in finite groups. In *Proc. 23rd ACM STOC 1991*, 164–174.
- [5] Babai, L., Beals, R., A polynomial-time theory of black box groups. In *Groups St. Andrews 1997 in Bath, I* (ed. by C. M. Campbell, E. F. Robertson, N. Ruskuc, G. C. Smith), London Math. Soc. Lecture Note Ser. 260, Cambridge University Press, Cambridge 1999, 30–64.
- [6] Babai, L., Cooperman, G., Finkelstein, L., Luks, E. M., Seress, Á., Fast Monte Carlo algorithms for permutation groups. *J. Comput. System Sci.* **50** (1995), 296–308.
- [7] Babai, L., Cooperman, G., Finkelstein, L., Seress, Á., Nearly linear time algorithms for permutation groups with a small base. In *Proc. International Symposium on Symbolic and Algebraic Computation (ISSAC '91)*, ACM Press, New York 1991, 200–209.
- [8] Babai, L., Kantor, W. M., Pálffy, P. P., Seress, Á., Black box recognition of finite simple groups of Lie type by statistics of element orders. *J. Group Theory* **5** (2002), 383–401.
- [9] Babai, L., Luks, E. M., Seress, Á., Permutation groups in NC. In *Proc. 19th ACM STOC 1987*, 409–420.
- [10] Babai, L., Luks, E. M., Seress, Á., Fast management of permutation groups I. *SIAM J. Algorithms* **26** (1997), 1310–1342.
- [11] Babai, L., Szemerédi, E., On the complexity of matrix group problems I. In *Proc. 25th IEEE FOCS 1984*, 229–240.
- [12] Beals, R., Babai, L., Las Vegas algorithms for matrix groups. In *Proc. 34th IEEE FOCS 1993*, 427–436.
- [13] Beals, R., Leedham-Green, C. R., Niemeyer, A. C., Praeger, C. E., Seress, Á., A black-box group algorithm for recognizing finite symmetric and alternating groups, I. *Trans. Amer. Math. Soc.* **355** (2003), 2097–2113.
- [14] Bratus, S., Pak, I., Fast constructive recognition of a black-box group isomorphic to S_n or A_n using Goldbach's conjecture. *J. Symbolic Comput.* **29** (2000), 33–57.
- [15] Brooksbank, P. A., Fast constructive recognition of black box unitary groups. *London Math. Soc. J. Comput. Math.* **6** (2003), 162–197.

- [16] Brooksbank, P. A., Kantor, W. M., On constructive recognition of a black box $\text{PSL}(d, q)$. In *Groups and Computation III* (ed. by W. M. Kantor, Á. Seress), Ohio State Univ. Math. Res. Inst. Publ. 8, Walter de Gruyter, Berlin, New York 2001, 95–111.
- [17] Brooksbank, P. A., Kantor, W. M., Fast constructive recognition of black box orthogonal groups. Preprint, 2006.
- [18] Brooksbank, P. A., Niemeyer, A. C., Seress, Á., A reduction algorithm for matrix groups with an extraspecial normal subgroup. In *Finite Geometries, Groups, and Computation* (ed. by A. Hulpke, R. Liebler, T. Penttila, Á. Seress), Walter de Gruyter, Berlin, New York 2006, 1–16.
- [19] Bosma, W., Cannon, J., Playoust, C., The MAGMA algebra system I: The user language. *J. Symbolic Comput.* **24** (1997), 235–265.
- [20] Cameron, P. J., Finite permutation groups and finite simple groups. *Bull. London Math Soc.* **13** (1981), 1–22.
- [21] Celler, F., Leedham-Green, C. R., A non-constructive recognition algorithm for the special linear and other classical groups. In *Groups and Computation II* (ed. by L. Finkelstein, W. M. Kantor), Amer. Math. Soc. DIMACS Ser. 28, Amer. Math. Soc., Providence, RI, 1997, 61–67.
- [22] Celler, F., Leedham-Green, C. R., Murray, S. H., Niemeyer, A. C., O’Brien, E. A., Generating random elements of a finite group. *Comm. Algebra* **23** (1995), 4931–4948.
- [23] Cooperman, G., Finkelstein, L., Combinatorial tools for computational group theory. In *Groups and Computation* (ed. by L. Finkelstein, W. M. Kantor), Amer. Math. Soc. DIMACS Ser. 11, Amer. Math. Soc., Providence, RI, 1993, 53–86.
- [24] Cooperman, G., Finkelstein, L., Linton, S., Recognizing $GL_n(2)$ in non-standard representation. In *Groups and Computation II* (ed. by L. Finkelstein, W. M. Kantor), Amer. Math. Soc. DIMACS Series 28, Amer. Math. Soc., Providence, RI, 1997, 85–100.
- [25] Conder, M. D. E., Leedham-Green, C. R., Fast recognition of classical groups over large fields. In *Groups and Computation III* (ed. by W. M. Kantor, Á. Seress), Ohio State Univ. Math. Res. Inst. Publ. 8, Walter de Gruyter, Berlin, New York 2001, 113–121.
- [26] Conder, M. D. E., Leedham-Green, C. R., O’Brien, E. A., Constructive recognition of $\text{PSL}(2, q)$. *Trans. Amer. Math. Soc.* **358** (2006), 1203–1221.
- [27] Feit, W., Tits, J., Projective representation of minimum degree of group extensions. *Canad. J. Math.* **30** (1978), 1092–1102.
- [28] The GAP Group, GAP – Groups, Algorithms, and Programming, Version 4.4. Aachen–St Andrews 2005; <http://www.gap-system.org>.
- [29] Holt, D., Eick, B., O’Brien, E. A., *Handbook of Computational Group Theory*. Chapman and Hall/CRC Press, Boca Raton, FL, 2005.
- [30] Kantor, W. M., Simple groups in computational group theory. *Proceedings of the International Congress of Mathematicians* (Berlin, 1998), Vol. II, Doc. Math., J. DMV, Extra Vol. ICM Berlin, 1998, 77–86.
- [31] Kantor, W. M., Magaard, K., Black-box exceptional groups of Lie type. In preparation.
- [32] Kantor, W. M., Seress, Á., Black box classical groups. *Mem. Amer. Math. Soc.* **149** (2001), Nr. 708.

- [33] Kantor, W. M., Seress, Á., Permutation group algorithms via black box recognition algorithms. In *Groups St. Andrews 1997 in Bath, II* (ed. by C. M. Campbell, E. F. Robertson, N. Ruskuc, G. C. Smith), London Math. Soc. Lecture Note Ser. 261, Cambridge University Press, Cambridge 1999, 436–446.
- [34] Kantor, W. M., Seress, Á., Computing with matrix groups. In *Groups, Combinatorics, and Geometry* (ed. by A. A. Ivanov, M. W. Liebeck, J. Saxl), World Scientific, River Edge, NJ, 2003, 123–137.
- [35] Landazuri, V., Seitz, G. M., On the minimal degrees of projective representations of the finite Chevalley groups. *J. Algebra* **32** (1974), 418–443.
- [36] Law, M., Niemeyer, A. C., Praeger, C. E., Seress, Á., A reduction algorithm for large-base primitive permutation groups. *London Math. Soc. J. Comput. Math.* **9** (2006), 159–173.
- [37] Leedham-Green, C. R., The computational matrix group project. In *Groups and Computation III* (ed. by W. M. Kantor, Á. Seress), Ohio State Univ. Math. Res. Inst. Publ. 8, Walter de Gruyter, Berlin, New York 2001, 229–247.
- [38] Liebeck, M. W., O’Brien, E. A., Finding the characteristic of a group of Lie type. Submitted, 2005.
- [39] Lovorn, R., Rigorous, subexponential algorithms for discrete logarithms over finite fields. Ph. D. thesis, U. of Georgia, 1992.
- [40] Luks, E. M., Computing in solvable matrix groups. In *Proc. 33rd IEEE FOCS. 1992*, 111–120.
- [41] Neumann, P. M., Praeger, C. E., A recognition algorithm for special linear groups. *Proc. London Math. Soc.* (3) **65** (1992), 555–603.
- [42] Neunhöffer, M., Seress, Á., A data structure for a uniform approach to computations with finite groups. In *Proc. International Symposium on Symbolic and Algebraic Computation (ISSAC’06)*, ACM Press, New York 2006, 254–261.
- [43] Niemeyer, A. C., Praeger, C. E., A recognition algorithm for classical groups over finite fields. *Proc. London Math. Soc.* (3) **77** (1998), 117–169.
- [44] O’Brien, E. A., Towards effective algorithms for linear groups. In *Finite Geometries, Groups, and Computation* (ed. by A. Hulpke, R. Liebler, T. Penttila, Á. Seress), Walter de Gruyter, Berlin, New York 2006, 163–190.
- [45] Ryba, A., Computer construction of split Cartan subalgebras. *J. Algebra*, to appear.
- [46] Ryba, A., Identification of matrix generators of a Chevalley group. Submitted, 2005.
- [47] Seress, Á., An introduction to Computational Group Theory. *Notices Amer. Math. Soc.* **46** (1997), 671–679.
- [48] Seress, Á., *Permutation Group Algorithms*. Cambridge Tracts in Math. 152, Cambridge University Press, Cambridge 2003.
- [49] Seress, Á., Large element orders and the characteristic of Lie-type simple groups. Submitted, 2005.
- [50] Sims, C. C., Computation with permutation groups. In *Proc. Second Symp. on Symbolic and Algebraic Manipulation*, ACM Press, New York 1971, 23–28.
- [51] Sims, C. C., *Computation with Finitely Presented Groups*. Encyclopedia Math. Appl. 48, Cambridge University Press, Cambridge 1994.

Department of Mathematics, The Ohio State University, Columbus, OH 43210, U.S.A.

E-mail: akos@math.ohio-state.edu