Simple Groups in Computational Group Theory

William M. Kantor

Abstract. This note describes recent research using structural properties of finite groups to devise efficient algorithms for group computation.

1991 Mathematics Subject Classification: 20B40, 20D05, 68Q40
Keywords and Phrases: group theoretic algorithms, simple groups

1. Introduction

Numerous applications already have been made of the monumental classification of the finite simple groups (CFSG) within algebra, combinatorics, model theory and computer science. See [GK] for a recent survey. Here I will only consider applications to computational group theory, in the intersection of algebra and computer science, emphasizing results whose statements make it far from clear how any simple group information could be applied.

Two computer systems are widely available for computational group theory: GAP (developed in Aachen by Neubüser and Schönert [Sch], but now moved to St. Andrews); and Magma (developed in Sydney by Cannon [CP]). These systems have been used for important applications within group theory and other parts of mathematics (cf. [Ser1]), occasionally with help from CFSG. One of the most important relationships between computer computations and simple groups was the construction and study of many sporadic (and other specific) simple groups. However, this brief survey focuses on the mathematics behind the algorithms: while practicality is certainly a very important aspect, additional important ones are the discovery of new ways to view standard group-theoretic results, the need for new results about groups, and complexity questions within computer science.

Introductory example The following purely mathematical result can be explained to undergraduates:

Theorem 1 [IKS]. *If p is a prime divisor of the order of a subgroup G of S_n, then the probability that a random element of G has order divisible by p is at least $1/n$. This bound is tight if and only if n is a power of p.*

A similar result is in [Ga]. There is a very practical motivation: assuming that there is a mechanism for finding random elements of G (cf. §§2,5), the theorem states that it "only" takes $O(n)$ samples in order very likely to obtain an element of order divisible by p, and hence one of order p. The proof reduces to the case of a simple group G, in which case much more precise information is obtained about the proportion of elements of G of order divisible by any given prime.

It should be emphasized that the standard proofs of Cauchy's Theorem, or more generally of Sylow's Theorem, are not likely to be used in actual computations to obtain elements or subgroups of complicated groups. Therefore, other ideas are needed for computations.

2. The permutation group setting; polynomial time

The basic computational situation discussed here is as follows: a group is given, specified as $G = \langle S \rangle$ in terms of some (arbitrary) generating set S of its elements[1]. The goal is then to find properties of G efficiently, such as $|G|$, the derived series, a composition series, Sylow subgroups, and so on. In this section S will be a set of permutations of an n-element set, and then the word "efficiently" might mean "in time polynomial (or nearly linear) in the input length $|S|n$ of the problem".

Many permutation group algorithms are described in detail in a new book by Seress [Ser2]. Numerous other aspects of computational group theory are surveyed in [Si3,Ser1], which also discuss different ways groups are commonly input into computers, e.g., via presentations. See [Ba1] for additional background, especially regarding complexity questions within various models of computation.

The development of efficient computer algorithms for permutation groups was begun by Sims [Si1,Si2] (who then used his ideas for existence proofs for sporadic simple groups). Of fundamental importance was his use of a *base* $B = \{\alpha_1, \dots, \alpha_b\}$ for G: any set of points whose pointwise stabilizer is 1 (possibly B consists of $n-1$ points). Let $G_{(i)}$ be the pointwise stabilizer[2] of $\alpha_1, \dots, \alpha_i$, so that $G = G_{(0)} \geq G_{(1)} \geq \cdots \geq G_{(b)} = 1$ and $|G| = \Pi_1^b |G_{(i-1)} : G_{(i)}|$; here $|G_{(i-1)} : G_{(i)}|$ is the length of the orbit $\mathcal{O}_i$ of α_i under $G_{(i-1)}$. Sims developed a data structure to find a base and (generators for) all of these subgroups $G_{(i)}$ and orbits $\mathcal{O}_i$ simultaneously and efficiently. This yielded $|G|$ using only elementary group theory: it did not involve structural properties of groups. The first version of Sims's order algorithm analyzed in polynomial time is in [FHL];[3] $O(|S|n^2 + n^5)$ versions are in [Kn,Je], and these methods cannot decrease the exponent 5 [Kn].

Once $|G|$ can be found, many other properties of G, such as the derived series, solvability and nilpotence, can be determined in polynomial time. More important from an algorithmic point of view was a Membership Test: *given $h \in S_n$, decide whether or not $h \in G$; and if it is, obtain h from the generating set S.* The first of these is easy: one could test whether or not $|G| = |\langle S \cup \{h\} \rangle|$; the second depends on the data structure in the order algorithm. A random element of G is now easily obtained as $t_b t_{b-1} \cdots t_1$ where, for each i, t_i is a random element of a transversal for $G_{(i)}$ in $G_{(i-1)}$. The above ideas were implemented in GAP and Magma.

Obstacles to polynomial-time computation In polynomial time it is not possible to list the elements of any given permutation group. There are more serious obstacles to the algorithmic study of permutation groups. Consider the following problems for a given $G = \langle S \rangle \leq S_n$.

Centralizer: Given $t \in G$ of order 2, find its centralizer $C_G(t)$.

Intersection: Given subgroups $H, K \leq G$, find their intersection $H \cap K$.

---

[1] It is standard to have groups specified by generating sets. A familiar example is the group of Rubik's cube.

[2] Such subgroups are typically involved in "solving" Rubik's cube.

[3] Sims has informed me that his original version also was a polynomial-time algorithm.

Here $C_G(t)$, H, K and $H \cap K$ are specified by generating sets. Luks observed that Centralizer and Intersection are polynomial-time equivalent and, what is more surprising, that the following problem reduces to these in polynomial time.

GRAPH ISOMORPHISM: Given two n-vertex graphs, are they isomorphic?

There are practical algorithms for Graph Isomorphism; the main one is due to B. McKay and is contained in both GAP and MAGMA. However, it is a long-standing open question whether there is a polynomial-time algorithm for Graph Isomorphism. This would be settled by a polynomial-time algorithm for Centralizer, but it seems unlikely that one exists. Thus, this leaves the awkward problem that centralizers, normalizers and intersections of arbitrary subgroups cannot be used in any of the algorithms considered here. This is discussed at length in [Lu3].

A remarkable result of Luks [Lu1] combined "elementary" group theory with group-theoretic algorithms (e.g., for intersecting a *solvable* group with any subgroup of S_n) to obtain a polynomial-time Graph Isomorphism algorithm assuming that the valences of the vertices are bounded.

3. POLYNOMIAL-TIME COMPUTATION USING CFSG

Structural properties of finite groups can help lead to algorithms and then be involved in proving their validity and timing. This is where CFSG enters. The breakthrough in the complexity of permutation group algorithms was Luks's use of CFSG to determine a composition series:

THEOREM 2 [Lu2,Be]. *There is a polynomial-time algorithm that determines a composition series of any given $G = \langle S \rangle \leq S_n$.*

The proof used a familiar consequence of CFSG, the validity of "Schreier's conjecture": the outer automorphism group of every finite simple group is solvable. A dozen years after it was first obtained, as part of his thesis Beals observed that a slight modification of Luks's algorithm eliminates any need for CFSG! This is the premier example of a consequence of CFSG in which CFSG was eventually removed. It seems as if there should be more instances of other consequences of CFSG (in various areas) which, once known to be true, can then be proved in better or simpler ways. An example begging for such a new proof is Theorem 1.

Standard methods for finding Sylow subgroups use exponential time in the worst case. However:

THEOREM 3 [Ka1]. *There are polynomial-time algorithms for the following problems. Given $G = \langle S \rangle \leq S_n$ and a prime p dividing $|G|$,*

(i) *Find a Sylow p-subgroup of G containing any given p-subgroup of G;*
(ii) *Given Sylow p-subgroups P_1 and P_2 of G, find $g \in G$ with $P_1^g = P_2$; and*
(iii) *Find the normalizer of a Sylow p-subgroup.*

The original arguments in [Ka1] were streamlined in [Ka2,KLM]. The basic idea is to reduce finding Sylow subgroups to finding them in simple groups and to conjugating them in arbitrary groups; then to reduce conjugating them to the simple group case; and finally to solve these types of problems for simple groups on a case-by-case basis. The algorithms in the theorem are impractical, but versions will go into GAP based on [Mo].

Quotient groups If $N \trianglelefteq G \leq S_n$, one can ask for properties and subgroups of G/N. For some questions (a composition series, Sylow subgroups) the transitions from algorithms for G to ones for G/N are elementary. However, it is not necessarily possible for G/N to be represented as a permutation group of degree $< 2^{n/4}$, as P. Neumann has observed when G is merely a direct product of dihedral groups. Hence, there may not be a permutation representation of a quotient group G/N to which previous results could be applied, so that some algorithms have to be developed from scratch for G/N. One of the most unexpected examples of this is the center. It is not difficult to find $Z(G)$ efficiently, but no elementary method is known for $Z(G/N)$. In [KL] this was computed in polynomial time, using the preceding algorithmic version of Sylow's Theorem via the following result: *There is a polynomial-time algorithm for computing* $\mathrm{Core}_G(H)$*, given* $H \leq G \leq S_n$.

Here, the *core* $\mathrm{Core}_G(H) = \cap\{H^g \mid g \in G\}$ is the largest normal subgroup of G contained in H. As noted earlier, it is not known how to intersect two subgroups of an arbitrary group in polynomial time, and this is probably impossible, presenting an apparent obstacle to computing $C = \mathrm{Core}_G(H)$. Nevertheless, C can be found as follows: for each prime $p \big| |G|$ find a Sylow p-subgroup Q of G and let $P := Q$; while $\langle P^G \rangle \not\leq H$ find $g \in G$ with $P^g \not\leq H$ and replace P by $P \cap H^{g^{-1}}$; this only involves intersecting with p-groups (cf. [Lu1]). Then C is generated by these subgroups P, one for each p. Now $L/N = Z(G/N)$ can be computed as follows: let $\hat{G} = \{(g, g) \mid g \in G\}$, acting on the disjoint union of two copies of our n-set, and let $A = \hat{G}(1 \times G)$, $B = \hat{G}(1 \times N)$ and $C = \mathrm{Core}_A(B)$; then L is the projection of C onto the first copy of G.

The results and methods in [KL] led to the Quotient Group Thesis: *If a problem is in polynomial time for permutation groups then it is also in polynomial time for quotients of permutation groups.* It is not clear how this thesis could ever be proved, but it holds for all "standard" questions concerning groups given as permutation groups. However, ridiculously different methods, involving simple groups, appear to be needed in the cases of permutation groups and their quotients.

Faster algorithms As already mentioned, finding $|G|$ requires time $O(|S|n^2 + n^5)$ by the methods in [Si1,Si2,FHL,Kn,Je]. The exponent 5 can be decreased by a very different method based on a nonalgorithmic CFSG-based property of primitive permutation groups [Ca]: *If* $G \leq S_n$ *is primitive and* $|G| > n^{2\log n}$*, then* $n = \binom{m}{l}^k$ *for some* m, l, k*, and* G *is a subgroup of* $S_m \mathrm{wr} S_k$ *with socle* $(A_m)^k$ *acting on the ordered* k*-tuples of* l*-subsets of an* m*-set.* Thus, reductions to primitive groups lead either to groups that are not too big or to ones that are easily understood. This in turn led to an $O(|S|n^3 \log^c n)$ *time algorithm for finding* $|G|$ [BLS2].

This faster method arose from the study of the theoretical feasibility of parallel computation with permutation groups, in which one allows a polynomial number of processors running in time $O(\log^c(|S|n))$ (the complexity class NC). Sims's method for order and membership testing is unavailable: it requires sequentially using a pointwise stabilizer sequence. Nevertheless, methods that later decreased the exponent 5 had already put those problems into NC [BLS1]—and are also used in new methodology described in the next section and employed in GAP. These algorithms rely heavily on CFSG, as do parallel algorithms for finding a composition series [BLS1] and Sylow subgroups [KLM].

4. Nearly linear algorithms

It takes time at least $|S|n$ to read $|S|$ permutations. It is remarkable that quite a few algorithms run in time not too far from this linear lower bound. A *nearly linear* algorithm for a permutation group $G = \langle S \rangle \leq S_n$ is one running in time $O(|S|n \log^c |G|)$ for some constant c. When applied to groups having a base of size $O(\log^{c'} n)$ for some c', $O(|S|n \log^c |G|)$ becomes $O(|S|n \log^{c+c'} n)$, which is very close to linear in the input length. Note that the classical simple groups, in all of their permutation representations, have bases of size $O(\log^2 n)$. It appears that most practical group-theoretic computations involve either small-base groups or alternating or symmetric groups.

Randomized algorithms I will need informal definitions of Las Vegas and Monte Carlo algorithms. Both are randomized algorithms. *The output of a* Monte Carlo algorithm *may be incorrect*, but that only can happen with a small, user-prescribed probability. So there is always an output, but there is also the uncomfortable possibility of error. *The output of a* Las Vegas algorithm *is correct*, but there is a small, user-prescribed probability that nothing is output. This is more comforting.

Most known nearly linear algorithms are Monte Carlo. These are of more than theoretical interest, since a large part of the permutation group library in GAP is based on implementations of nearly linear algorithms for many of the problems discussed so far: finding $|G|$, the derived series and a composition series; and soon, finding and conjugating Sylow subgroups with some restrictions on the noncyclic composition factors [Mo]. Many of these algorithms are described in detail in [Ser2]. In GAP the possibility of erroneous output presently is avoided by applying an $O(|S|n^2 \log^c |G|)$ algorithm essentially due to Sims in order to check the correctness of point stabilizer constructions [Ser2]. Now, under mild restrictions, all of these nearly linear Monte Carlo algorithms can be upgraded to nearly linear Las Vegas ones, using algorithms Seress will program into GAP:

Theorem 4 [KS1,KS2,KM]. *There are* nearly linear Las Vegas *algorithms which, when given $G = \langle S \rangle \leq S_n$ with no composition factor isomorphic to any $PSU(3, q)$, ${}^2B_2(q)$, ${}^2G_2(q)$ or ${}^2F_4(q)$, determine the following*: $|G|$, *membership in G, a composition series for G, and everything else previously found only by Monte Carlo algorithms.*

The proof starts with a known Monte Carlo algorithm that finds a composition series for G. For an alleged simple group that is a (composition) factor of this series, determine its order and isomorphism type and use a very fast constructive recognition test (cf. §5). If the test fails, output nothing (the probability of this is small); otherwise verify the precise composition factors of G, hence find $|G|$. Knowing $|G|$ with certainty in turn allows the outputs of all other known nearly linear algorithms to be verified with certainty.

Following [BLS1], this approach departs significantly from the standard methods based on Sims's point stabilizer ideas: by the time we have $|G|$ we have a composition series for G. Very fast simple group recognition algorithms were essential.

These have much wider applicability, as will be seen in the next section.

5. Black box groups

In §2 I mentioned a general computational setting for a group $G = \langle S\rangle$. The most-studied case of that setting is permutation groups. Another very natural setting is matrix groups: S is a set of invertible matrices over some field, which here is always a finite field. The questions remain the same: efficiently find properties of G, such as $|G|$, solvability, a composition series, etc. If $S \subset GL(d,q)$ then the input length is $|S|d^2 \log q$ (since $\log q$ bits are required to write each of the d^2 entries). These problems seems to be very hard. However, under reasonable additional conditions, and allowing probabilistic algorithms, this has become an actively studied area. Some of the most interesting results have stemmed from ignoring the representation of G on $\mathbb{F}_q^d$ implicit in the above description (hence ignoring eigenvalues, minimal polynomials and so on), and abstracting to the following notion.

A *black box group* is a group G, whose elements are *encoded by binary strings* of the same length N, such that routines ("oracles") are provided for

$$\begin{cases} \text{multiplying two elements,} \\ \text{inverting an element,} \\ \text{deciding whether an element} = 1. \end{cases}$$

Here G is specified as $G = \langle S\rangle$ for some set S of elements. Note that $|G| \le 2^N$; not all strings correspond to group elements.

The basic examples are permutation groups and matrix groups. However, considering permutation groups as black box groups, so that group operations can be performed much faster than permutation multiplication, has recently become a crucial tool within the study of permutation groups [Ser2,KS1]. Moreover, experience has shown that allowing fewer tools sometimes forces new and better methods.

According to an amazing result of Babai [Ba2], *one can find a nearly uniformly distributed random element of a black box group $G = \langle S\rangle$ using $O(|S|N^5)$ group operations.* This tour de force involves combinatorial methods but nothing about the structure of G; note that $|G|$ is never known here. A practical heuristic algorithm in [CLMNO] for finding random group elements is adequate for Las Vegas algorithms, in which correctness of the output is ultimately verified (cf. [Ba3]).

At present the most general theorem concerning black box groups is

Theorem 5 [BB,KS1]. *There is a Las Vegas algorithm for the following problem. Suppose that a black box group $G = \langle S\rangle$ is given, together with a list of primes that contains all prime divisors of $|G|$ as well as oracles for handling elementary abelian subgroups of G and discrete logarithms. Then $|G|$ and a composition series for G can be computed in time polynomial in both the input length and the size of the largest field involved in defining the Lie type composition factors of G.*

The additional oracle hypothesis presumes access to "discrete logarithms" (*write any given element of $\mathbb{F}_p^*$ as a power of a given generator*) and "handling elementary abelian subgroups" E (i.e., *the ability to do all standard linear algebra in E*). As originally stated in [BB] the polynomial timing in the theorem also involved a polynomial in the smallest integer $\nu(G)$ such that all nonabelian composition factors of G have faithful permutation representations of degree at most

$\nu(G)$. The slightly stronger result stated above was obtained in [KS1] using simple group recognition algorithms, discussed below. Sylow subgroups can also be found in the situation of the theorem, but the permutation group literature [Ka1] is no longer available for this as it was in [BB].

Recognizing simple groups All modern Sylow subgroup algorithms for permutation groups reduce to the case of simple groups [Ka1,Ka2,Mo,KLM,CCH]. For any given simple permutation group one first determines an explicit isomorphism with a known simple group, afterwards studying Sylow subgroups of the concrete simple groups. Deterministic algorithms producing such isomorphisms are in [Ka1,Ka2,KLM].

Outside the permutation group setting, the problem of *recognizing* simple groups began with algorithms [NeP,NiP,CLG1] for deciding whether a given subgroup $G = \langle S\rangle \le GL(d,q)$ contains $SL(d,q)$ or a classical group as a normal subgroup. These were *nonconstructive* recognition algorithms, outputting either "G contains a normal classical group", or "G probably does not contain any classical group of $d\times d$ matrices as a normal subgroup". These rely heavily on CFSG: they search for certain matrices in G that occur with high probability in the relevant classical groups, and then use a far-reaching nonalgorithmic consequence of CFSG determining the subgroups of $GL(d,q)$ containing such elements [GPPS].

This suggested the need for *constructive* recognition algorithms: given $G = \langle S\rangle \le GL(d,q)$ containing $SL(d,q)$, the algorithm in [CLG2] writes any given element of $SL(d,q)$ in terms of S. This has also been done for the symplectic groups [Ce]. The nonconstructive recognition algorithms are Monte Carlo (more precisely, *one-sided Monte Carlo* [Ba3], since an output such as "contains $SL(d,q)$" is guaranteed to be correct), and run in time polynomial in the input length; the constructive ones can be viewed as Las Vegas algorithms whose timing also depends on a small power of q, so that these do not run in polynomial time.

The black box version of constructive recognition is conceptually harder: there is no longer a vector space available, hence no linear algebra to rely on. The goal is an effective isomorphism[4] $\varphi\colon G \to H$ to a concrete version H of the simple black box group G, whereas only the name of H is output in nonconstructive recognition. It was not at all clear that one could recognize a black box simple group in this manner, but in [CFL] this was shown to be possible when $G \cong PSL(d,2)$.

More generally, *there is a Las Vegas algorithm which, when given a black box group G isomorphic to a simple group of Lie type of known characteristic, constructively recognizes G in time polynomial in the input length and field size* [KS1] (cf. [KM]). The characteristic assumption is removed in [KS3] by assuming instead that there is a method (an oracle) for finding the order of any given element of G. When the characteristic is known, the idea is to (probably) construct an element in a large conjugacy class, one of whose powers is a (long) root element; then construct larger subgroups using random conjugates of these root elements; and ultimately make a recursive call to a group of rank one less than that of G (if G does not already have rank 1). The final step of the algorithm verifies the correctness of the output isomorphism by computing a presentation for G (see below).

[4] Able to find $g\varphi$ or $h\varphi^{-1}$ for any *given* $g \in G$ or $h \in H$.

Here the isomorphism type of G is not part of the input. Note that the Lie rank l and logarithm of the field size q are polynomial in N, since $N \geq \log|G| \geq (l^2 \log q)/4$. While q appears in timing estimates of all known constructive recognition algorithms, it should not be needed, at least when l is large. Analogous results for alternating groups are in [BLNPS].

Presentations Las Vegas algorithms recognizing simple black box groups eventually need to verify a presentation of a known simple group, which for use in Theorem 4 must be written in time polynomial only in the input length. In view of the time required to multiply out a permutation g given as a product of generators, verifying that $g = 1$ involves the lengths of presentations[5]. For all simple groups except, perhaps, $PSU(3,q)$, ${}^2B_2(q)$ and ${}^2G_2(q)$, there is a presentation of length $O(\log^c |G|)$, using $c = 2$ and in most cases even $c = 1$; the proof in [BGKLP] uses simple tricks to adapt the usual Curtis-Steinberg-Tits presentations for these groups. It is perhaps surprising that the case of the very familiar groups $PSU(3,q)$ has remained open for almost 10 years. Short presentations have the following nonalgorithmic consequence needed in the proof of Theorem 4: *Every finite group G, with no composition factor of the form $PSU(3,q)$, ${}^2B_2(q)$ or ${}^2G_2(q)$, has a presentation of total length $O(\log^3|G|)$.* The exponent 3 is best possible.

These short presentations also were used in [Ma] to prove the existence of a constant c such that there are at most $n^{cd\log n}$ d-generator groups of order n with no composition factor $PSU(3,q)$, ${}^2B_2(q)$ or ${}^2G_2(q)$, and each such group can be defined by means of $cd\log n$ relations in those d generators. As with Theorem 1, we see that algorithmic needs have led to a result about finite groups. Of course, it is not surprising that many applications of CFSG have needed new properties of simple groups (cf. [GK]).

Acknowledgment: I am grateful to L. Babai, E. Luks, J. Neubüser, Á. Seress and C. Sims for very helpful comments.

References

[Ba1] L. Babai, Computational complexity in finite groups, pp. 1479-1489 in: Proc. ICM, Kyoto 1990.

[Ba2] L. Babai, Local expansion of vertex-transitive graphs and random generation in finite groups, pp. 164–174 in: Proc. ACM STOC 1991.

[Ba3] L. Babai, Randomization in group algorithms: conceptual questions, [FK] 1–17.

[BB] R. Beals and L. Babai, Las Vegas algorithms for matrix groups, pp. 427–436 in: Proc. IEEE FOCS 1993.

[BCFLS] L. Babai, G. Cooperman, L. Finkelstein, E. M. Luks and Á. Seress, Fast Monte Carlo algorithms for permutation groups, pp. 90–100 in: Proc. ACM STOC 1991.

[BCFS] L. Babai, G. Cooperman, L. Finkelstein and Á Seress, Nearly linear time algorithms for permutation groups with a small base, pp. 200–209 in: Proc. ISSAC 1991.

[Be] R. Beals, An elementary algorithm for computing the composition factors of a permutation group, pp. 127-134 in: Proc. ISSAC 1993.

[5] The *length* of a presentation $\langle X \mid R\rangle$ is $|X| + \Sigma_{r\in R}\, l_X(r)$.

[BGKLP] L. Babai, A. J. Goodman, W. M. Kantor, E. M. Luks and P. P. Pálfy, Short presentations for finite groups. J. Algebra 194 (1997) 79–112.

[BLNPS] R. Beals, C. R. Leedham-Green, A. C. Niemeyer, C. E. Praeger and Á. Seress, A mélange of black box algorithms for recognising finite symmetric and alternating groups (in preparation).

[BLS1] L. Babai, E. M. Luks and Á. Seress, Permutation groups in NC, pp. 409–420 in Proc. ACM STOC 1987.

[BLS2] L. Babai, E. M. Luks and Á. Seress, Fast management of permutation groups I. SIAM J. Comput. 26 (1997) 1310–1342; II (in preparation).

[Ca] P. Cameron, Finite permutation groups and finite simple groups. BLMS 13 (1981) 1–22.

[CCH] J. Cannon, B. Cox and D. F. Holt, Computing Sylow subgroups in permutation groups (to appear).

[Ce] F. Celler, Matrixgruppenalgorithmen in GAP. Ph. D. thesis, RWTH Aachen 1997.

[CFL] G. Cooperman, L. Finkelstein and S. Linton, Recognizing $GL_n(2)$ in non-standard representation, [FK] 85–100.

[CLG1] F. Celler and C. R. Leedham-Green, A non-constructive recognition algorithm for the special linear and other classical groups, [FK] 61–67.

[CLG2] F. Celler and C. R. Leedham-Green, A constructive recognition algorithm for the special linear group (to appear in Proc. ATLAS Conf.).

[CLMNO] F. Celler, C. R. Leedham-Green, S. H. Murray, A. C. Niemeyer and E. A. O'Brien, Generating random elements of a finite group. Comm. in Alg. 23 (1995) 4931–4948.

[CP] J. Cannon and C. Playoust, An introduction to Magma. School of Math. and Stat., Univ. Sydney, 1993. magma@maths.usyd.edu.au

[FHL] M. Furst, J. Hopcroft and E. Luks, Polynomial–time algorithms for permutation groups, pp. 36–41 in Proc. IEEE FOCS 1980.

[FK] L. Finkelstein and W. M. Kantor, Groups and Computation II, AMS 1997.

[Ga] A. Gambini, Zur Komplexität einiger gruppentheoretischer Algorithmen, Ph. D. thesis, Univ. Freiburg 1992.

[GK] R. M. Guralnick and W. M. Kantor, Some applications of the classification of finite simple groups (in preparation).

[GPPS] R. M. Guralnick, T. Penttila, C. E. Praeger and J. Saxl, Linear groups with orders having certain primitive prime divisors (submitted).

[IKS] I. M. Isaacs, W. M. Kantor and N. Spaltenstein, On the probability that a group element is p–singular. J. Algebra 176 (1995) 139–181.

[Je] M. R. Jerrum, A compact representation for permutation groups. J. Algorithms 7 (1986) 60–78.

[Ka1] W. M. Kantor, Sylow's theorem in polynomial time. J. Comp. Syst. Sci. 30 (1985) 359–394.

[Ka2] W. M. Kantor, Finding Sylow normalizers in polynomial time. J. Algor. 11 (1990) 523–563.

[KL] W. M. Kantor and E. M. Luks, Computing in quotient groups, pp. 524–534 in: Proc. ACM STOC 1990.

[KLM] W. M. Kantor, E. M. Luks and P. D. Mark, Sylow subgroups in parallel (to appear in J. Algorithms).

[KM] W. M. Kantor and K. Magaard, Black box exceptional groups of Lie type (in preparation).

[Kn] D. E. Knuth, Efficient representation of perm groups. Combinatorica 11 (1991) 33–43.

[KS1] W. M. Kantor and Á. Seress, Black box classical groups (submitted).

[KS2] W. M. Kantor and Á. Seress, Permutation group algorithms via black box recognition algorithms (to appear in Proc. Bath Conf.).

[KS3] W. M. Kantor and Á. Seress (in preparation).

[Lu1] E. M. Luks, Isomorphism of graphs of bounded valence can be tested in polynomial time. J. Comp. Syst. Sci. 25 (1982) 42–65.

[Lu2] E. M. Luks, Computing the composition factors of a permutation group in polynomial time. Combinatorica 7 (1987) 87–99.

[Lu3] E. M. Luks, Permutation groups and polynomial time computation, pp. 139–175 in: Groups and Computation (eds. L. Finkelstein, W. M. Kantor), AMS 1993.

[Ma] A. Mann, Enumerating finite groups and their defining relations (to appear in J. Group Theory).

[Mo] P. Morje, A nearly linear algorithm for Sylow subgroups of permutation groups. Ph.D. thesis, Ohio State U. 1995.

[NeP] P. M. Neumann and C. E. Praeger, A recognition algorithm for special linear groups, PLMS 65 (1992) 555–603.

[NiP] A. C. Niemeyer and C. E. Praeger, Implementing a recognition algorithm for classical groups, [FK] 273–296.

[Sch] M. Schönert et. al., GAP: Groups, Algorithms, and Programming. Lehrstuhl D für Mathematik, RWTH Aachen, 1994. gap@dcs.st-and.ac.uk

[Ser1] Á. Seress, An introduction to computational group theory. Notices AMS 44 (1997) 671–679.

[Ser2] Á. Seress, Permutation group algorithms (Cambridge University Press, to appear).

[Si1] C. C. Sims, Computational methods in the study of permutation groups, pp. 169–183 in: Computational problems in abstract algebra (ed. J. Leech), Pergamon 1970.

[Si2] C. C. Sims, Computation with permutation groups, pp. 23–28 in: Proc. Symp. Symb. Alg. Manipulation (ed. S. R. Petrick), ACM 1971.

[Si3] C. C. Sims, Group-theoretic algorithms, a survey, pp. 979–985 in: Proc. ICM, Helsinki 1978.

William M. Kantor
U. of Oregon
Eugene, OR 97403, USA