CONSTRUCTING COMPOSITION FACTORS FOR A LINEAR GROUP IN POLYNOMIAL TIME

DEREK HOLT, C.R. LEEDHAM-GREEN, AND E.A. O'BRIEN

In memory of our friend Kay Magaard

Abstract. We present a Las Vegas polynomial-time algorithm that takes as input a subgroup of $\mathrm{GL}(d, \mathbb{F}_q)$ and, subject to the existence of certain oracles, determines its composition factors, provided that none of those factors is isomorphic to one of ${}^2B_2(2^{2k+1})$, ${}^2F_4(2^{2k+1})$, ${}^3D_4(2^k)$, or ${}^2G_2(3^{2k+1})$, for any k.

1. Introduction

In 1987 Luks [41] provided the first polynomial-time algorithm to construct the composition factors of a permutation group. This result has important implications: Kantor [33] employed it to obtain polynomial-time construction of Sylow subgroups, and Babai, Luks & Seress [7] use it as a building block for a family of polynomial-time algorithms for permutation groups. For an extensive related discussion, see Seress [45, §6.2].

Our goal in this paper is to provide the first polynomial-time algorithm to solve this problem for linear groups defined over finite fields. In effect, the algorithm is an outcome of the "matrix group recognition" project, a major topic of research over the past 25 years. For an overview of the project, see [43].

Let $G = \langle X \rangle \leq \mathrm{GL}(d, \mathbb{F}_q)$ where $\mathbb{F}_q$ is a finite field of order $q = p^e$. In summary, the fundamental aim of the project is to identify the composition factors of G, and to solve the *constructive membership problem* in G. This means to decide whether a given $g \in \mathrm{GL}(d, \mathbb{F}_q)$ lies in G and, if so, to write g as a *word over* X: namely, as a word in the alphabet $X \cup X^{-1}$. In practice, we construct a compressed version of the word as a *straight line program* [45, p. 10]; this ensures that its length (and cost of evaluation) is polynomially bounded.

Two approaches have dominated the research undertaken. Babai & Beals [8] initiated the *black-box approach*: it aims to construct a specific characteristic series of subgroups for an arbitrary finite group G that can be refined to provide a composition series; the

Date: February 17, 2020.

Key words and phrases. finite linear groups; composition factors; algorithm; polynomial-time.

O'Brien was supported by a University of Auckland Hood Fellowship and by the Hausdorff Institute, Bonn, while this work was carried out. All authors were supported by the Marsden Fund of New Zealand via grant UOA 1626. We thank both James Wilson and the referee for many helpful suggestions.

associated algorithms are independent of the given representation of G. In 2009, Babai, Beals & Seress [11] proved that, subject to the availability of certain oracles, there exists a Monte Carlo polynomial-time black-box algorithm to construct this characteristic series for $G = \langle X \rangle \leq \mathrm{GL}(d, \mathbb{F}_q)$ when q is odd, to identify the composition factors, and to solve the constructive membership problem for G. (For the definitions of Monte Carlo and Las Vegas algorithms, see Section 2.1 below.) If q is even, then they can construct a composition series for $G/\mathrm{Rad}(G)$ and identify its composition factors, where $\mathrm{Rad}(G)$ is the soluble radical of G. Their computations in the soluble radical rely on the work of Luks [42].

The algorithms of [11] rely on two number-theoretic oracles. The first is a *discrete log oracle*: for given nonzero μ and fixed primitive element ω of a finite field F, it returns the unique $k \in \{0, \dots, |F| - 1\}$ such that $\mu = \omega^k$. It is needed for fields of order q^i for $1 \leq i \leq d$. The second oracle factorises numbers of the form $q^i - 1$ for $1 \leq i \leq d$. Both are needed to solve problems in abelian matrix groups. The algorithms can be upgraded to Las Vegas provided that polynomial-time black-box constructive membership algorithms and short presentations (both defined below) are available for all nonabelian composition factors of G.

By contrast, the *geometric approach* investigates whether $G = \langle X \rangle \leq \mathrm{GL}(d, \mathbb{F}_q)$ satisfies certain geometric properties in its action on its underlying vector space $V = \mathbb{F}_q^d$. For example, G acts reducibly if it fixes a nonzero proper subspace of V, and it acts imprimitively if it permutes the summands of a direct sum decomposition of V. A classification of the maximal subgroups of classical groups by Aschbacher [1] underpins this approach: in summary, either G preserves a linear structure in its action on V, and has a normal subgroup related to this structure, so providing a *reduction*; or it has a normal absolutely irreducible subgroup that is simple modulo scalars. The associated algorithms recursively exploit this reduction to construct a composition series for G. The outcome is reported in [4], where the algorithm CompositionTree is described. It takes as input $G = \langle X \rangle \leq \mathrm{GL}(d, \mathbb{F}_q)$ and outputs a *composition tree*, a data structure, for G. The tree allows us to list both a composition and chief series for G, and to solve membership and other problems for G.

Central to the CompositionTree algorithm are *short presentations* for the simple groups that occur as composition factors of G. For each finite nonabelian simple group S, we have defined a specific sequence of *standard generators*. A *constructive recognition algorithm* for S takes as input a group $G = \langle X \rangle$ known to be isomorphic to S, computes standard generators of G as words over X, and uses the standard generators to establish an isomorphism between G and (a central quotient of) the *standard copy* of S, a specific representation of S. The isomorphism is realised by an algorithm that solves the constructive membership problem in G. The constructive recognition algorithm returns the standard generators and the constructive membership algorithm for G. Babai & Szemerédi [5] define the *length* of a presentation to be the number of symbols required to write it down. A presentation on our standard generators for every finite nonabelian simple group S is known; with the exception of one family of finite simple groups, this presentation is *short* in the sense that its length is bounded by a function which is polynomial in $\log |S|$; it is not known whether short presentations exist for the small Ree groups ${}^2G_2(3^{2k+1})$. For details of the standard generators and presentations, see [15, 17, 38, 40, 48]. Ultimately, these presentations for the composition factors of G are combined to write down a presentation for G, allowing us to verify the correctness of the output of the resulting Las Vegas CompositionTree algorithm. The outcome is efficient in practice; an implementation of CompositionTree and its associated algorithms is available in Magma [14].

In the introduction to [4], we wrote that "Serious obstructions remain before we have a provably polynomial-time algorithm to compute a composition tree". Here we revisit the topic and obtain the following result.

Theorem 1.1. *There is a Las Vegas polynomial-time algorithm that takes as input a group $G = \langle X\rangle \leq \mathrm{GL}(d, \mathbb{F}_q)$ and, subject to the existence of a discrete log oracle for $\mathbb{F}_{q^i}$ and an oracle to factorise integers of the form $q^i - 1$ for $1 \leq i \leq d$, and to the availability of polynomial-time constructive recognition algorithms and short presentations for the nonabelian composition factors of G, it constructs a composition tree for G.*

By "constructs a composition tree for G", we mean solving the basic problems discussed earlier: compute a composition series for G, identify the factors in this series, and provide a solution to the constructive membership problem in G. We also provide an isomorphism between each nonabelian composition factor of G and (a central quotient of) the standard copy of that factor.

The following corollary reflects the current status of constructive recognition algorithms for the various families of finite simple groups.

Corollary 1.2. *There is a Las Vegas polynomial-time algorithm that takes as input a group $G = \langle X\rangle \leq \mathrm{GL}(d, \mathbb{F}_q)$ that has no composition factor isomorphic to ${}^2B_2(2^{2k+1})$, ${}^2F_4(2^{2k+1})$, ${}^3D_4(2^k)$, or ${}^2G_2(3^{2k+1})$, for any k, and, subject to the existence of a discrete log oracle for $\mathbb{F}_{q^i}$ and an oracle to factorise integers of the form $q^i - 1$ for $1 \leq i \leq d$, it constructs a composition tree for G.*

As we shall explain in Section 2.3, the oracle to factorise integers of the form $q^i - 1$ allows us to calculate and factorise the orders of elements of G in polynomial time. Corollary 1.2 is a direct consequence of Theorem 5.1, which is proved using Theorems 3.1 and 4.1. Although the arguments used in the proofs of these theorems constitute a proof of Theorem 1.1, we preferred to formulate them so that they provide more information on what we can do in the cases excluded by the corollary.

In particular, we can handle groups having ${}^2G_2(3^{2k+1})$ as composition factors but, since no short presentations are known for the groups ${}^2G_2(3^{2k+1})$, our algorithm is only Monte Carlo. We can handle individual groups from the other excluded classes for small k by treating them as "sporadic groups".

The serious obstructions alluded to in [4] to a polynomial-time algorithm arose principally from our inability to find (or prove the non-existence of) Aschbacher reductions of matrix groups in polynomial time. We overcome that problem by proving in Theorem 3.1 that we can in Monte Carlo polynomial time find a nontrivial element in a proper normal subgroup of a nonabelian black-box group, and then prove in Theorem 4.1 that we can use such elements effectively to find Aschbacher reductions of matrix groups. There have also been significant recent advances in the development of algorithms for the constructive recognition of the finite exceptional groups of Lie type.

Our primary objective is to prove the theorem and corollary as stated, without considering the degrees of the polynomials involved. It is easy to produce explicit bounds, but they are too large to be of practical interest. Our implementation of the algorithm of [4] rarely exhibits the difficulties that the algorithm presented here is designed to avoid; this justifies our decision to pay little attention to practical performance.

The discrete log oracle is used in the constructive recognition of simple groups of Lie type, and to determine the order and structure of certain abelian subgroups of $\mathrm{GL}(d, \mathbb{F}_q)$ that may be rewritten over an extension field $\mathbb{F}_{q^i}$ for some $i \in \{1, \ldots, d\}$. The most efficient existing algorithms to solve the discrete log problem run in sub-exponential time (see [46, Chapter 4]).

A complete or partial factorisation of integers of the form $q^i - 1$ for certain $i \in \{1, \ldots, d\}$ is needed. A partial factorisation into 'small' primes and certain coprime residues that are products of 'large' primes can be carried out in polynomial time. Further factorisation is only needed if G has a composition factor of order a prime dividing such a residue. That a residue is prime may be determined in polynomial time.

In Section 2 we discuss black-box groups, Monte Carlo and Las Vegas algorithms, and procedures to generate random elements of black-box groups. We also summarise the current status of constructive recognition algorithms for the finite simple groups. In Section 3 we prove the main technical result of the paper by presenting a Monte Carlo algorithm that takes as input a nonabelian black-box group G, and either identifies G as a finite simple group, or outputs a nontrivial element of a proper normal subgroup of G. In Section 4 we show how this algorithm underpins a Las Vegas polynomial-time algorithm that takes as input $G \leq \mathrm{GL}(d, \mathbb{F}_q)$ and either finds an Aschbacher reduction of G, or proves that G is nearly simple and identifies its nonabelian composition factor. We use this to prove Theorem 1.1 in Section 5.

2. Black-box groups and algorithms

The concept of a *black-box group* was introduced in [5]. In this model, the elements of a finite group G are encoded by bit-strings of uniform length N, so G has an *encoding* of length N and $|G| \leq 2^N$. The encoding of an element is not required to be unique, but distinct group elements have distinct encodings. Not all bit-strings are required to represent group elements.

Three oracles are supplied. One takes as input encodings of an ordered pair (g, h) of elements of G and returns an encoding of gh; a second takes as input an encoding of $g \in G$

and returns an encoding of g^{-1}; the third takes as input an encoding of g and returns TRUE or FALSE according as g is or is not the identity element of G. Both permutation groups and matrix groups defined over finite fields are covered by this model.

A *black-box algorithm* for a black-box group G takes, as part of its input, a generating set X of G, and the three oracles described above. Such an algorithm does not use specific features of the group representation, nor particulars of how group operations are performed. The size of the input is $N|X|$. In calculating the complexity of the algorithm, each call to one of the oracles is regarded as a single operation taking constant time. (For permutation groups and matrix groups defined over finite fields, these operations take time polynomial in N.)

2.1. **Monte Carlo and Las Vegas algorithms.** A *Monte Carlo* algorithm is a randomised algorithm that takes a positive real number $\epsilon < 1/2$ as part of its input, and it may, with probability at most ϵ, return an incorrect answer. Such an algorithm is said to run in polynomial time if its running time is bounded by a polynomial function of its input length and $\log \epsilon^{-1}$. A *Las Vegas* algorithm is a Monte Carlo algorithm that never gives an incorrect answer: it either returns a correct answer or it reports failure, and the latter occurs with probability at most ϵ. A Monte Carlo algorithm is upgraded to Las Vegas if it is combined with an algorithm that can decide whether or not its output is correct. See [45, §1.3] for a discussion of these concepts.

Usually the randomisation employed in an algorithm is controlled by a random number generator. Since we consider only randomised algorithms whose input includes a group $G = \langle X \rangle$, we assume equivalently that the non-deterministic behaviour of the algorithm is controlled by a random element generator that outputs nearly uniformly distributed random elements of G as words over X. The proofs of our results depend on this property.

2.2. **Constructing random elements.** An algorithm constructs an ϵ-uniformly distributed element x of a finite group G if

$$(1-\epsilon)/|G| < \mathrm{Prob}(x = g) < (1+\epsilon)/|G| \text{ for all } g \in G.$$

If $\epsilon < 1/2$, then the algorithm constructs *nearly uniformly distributed* random elements of G. Babai [6] presents a Monte Carlo polynomial-time black-box algorithm to construct such elements. An alternative is the *product replacement algorithm* of Celler *et al.* [19]; that this runs in polynomial time was established by Pak [44]. See [45, §2.2] for a discussion of these methods.

The two algorithms cited in the preceding paragraph make no use of the 'is-identity' oracle, which tests whether a given group element is equal to the identity element. The elements that they return are defined as words over the input generating set X. These properties of the method used to generate random elements are assumed to hold in the proof of Theorem 3.1, which relies essentially on the following behaviour.

Recall that the sequence of random objects generated by an algorithm is controlled by an initial seed that can be set immediately before running the algorithm; if we reset the seed to be the same as in a previous run, then the same sequence of random objects will be generated in the subsequent run. Suppose that G and H are two black-box groups with the property that they accept the same bit strings as representing group elements, their generating sets are represented by the same set X of bit strings, and they have identical 'product' and 'inverse' oracles, but possibly different 'is-identity' oracles. This situation arises in particular when $G \cong H/K$, and the 'is-identity' oracle for G tests elements of H for membership of K. Suppose that we set the initial seed and construct a sequence of random elements of G; then we reset the seed to the same value, and now construct the same number of random elements of H. Then we obtain the same sequence of random elements as *words over* X in the two runs. If the random elements of H are nearly uniformly distributed, then so are the random elements of H/K.

2.3. **An order oracle.** Let G be a black-box group. We assume that an *order oracle* FactoredOrder is available: it returns the prime factorisation of the order of a given element of G. We regard each call to this oracle as a single operation. A $\mathrm{BB}^{\mathrm{o}+}$ algorithm is a black-box algorithm that uses this additional oracle; a $\mathrm{BB}^{\mathrm{od}+}$ algorithm is one which, in addition, uses an oracle to compute discrete logs in certain finite fields. Note that the order and discrete log oracles are used only to enable black-box algorithms to run faster; they do not change their functionality.

For a discussion of an order oracle for black-box groups, see [39, §5.1]. An algorithm that, given the prime factorisation of integers of the form $q^i - 1$ for $1 \leq i \leq d$, determines the factorised order of an element of $\mathrm{GL}(d, \mathbb{F}_q)$ or $\mathrm{PGL}(d, \mathbb{F}_q)$ in polynomial time is presented in [20]. Factorisations of numbers of the form $q^i - 1$ are available as part of the Cunningham Project [18].

2.4. **Critical algorithms and their realisation.** For a discussion of the families of finite simple groups, see, for example, [47]. In particular, let $G(q)$ denote a finite quasisimple classical or exceptional group of Lie type over a field of order q.

Liebeck & O'Brien [39] present a Monte Carlo $\mathrm{BB}^{\mathrm{o}+}$ algorithm to determine the defining characteristic of a finite simple group of Lie type. Babai *et al.* [10] present a Monte Carlo black-box algorithm that, given as input a black-box group G isomorphic to a simple group of Lie type of known characteristic, determines the standard name of G. Using [32] and [48], we extended this to include the alternating and sporadic groups. All of these run in time polynomial in the size of the input.

A Las Vegas black-box algorithm to solve the constructive recognition problem for alternating groups is described in [32]. It runs in time polynomial in the size of the input.

A Las Vegas black-box algorithm to solve the constructive recognition problem for classical groups is described in [26]. The situation for exceptional groups is more complicated. We exclude ${}^2B_2(2^{2k+1})$, ${}^2F_4(2^{2k+1})$, ${}^3D_4(2^k)$, and ${}^2G_2(3^{2k+1})$ from discussion for now. Kantor & Magaard [35] present Las Vegas black-box algorithms to solve the problem for the remaining exceptional groups. The algorithms of [26, 35] take as input a representation of

$G(q)$ and run in time polynomial in the size of the input subject to the existence of the following oracles:

- discrete log oracles for $\mathbb{F}_{q^e}$, the field of definition of $G(q)$ (so $e \leq 3$);
- an oracle to recognise constructively central quotients of $\mathrm{SL}(2, \mathbb{F}_{q^e})$.

To realise the $\mathrm{SL}(2, \mathbb{F}_{q^e})$-oracle was challenging: for polynomial-time solutions in odd and even characteristic respectively, see [13, 34].

Recall, from [36], that a faithful linear or projective representation of $G(q)$ in characteristic r coprime to q has degree that is a non-constant polynomial in q; since such input has size bounded below by q, it is easy to design constructive recognition algorithms which achieve polynomial time complexity. For matrix representations in *defining characteristic*, more efficient algorithms are needed. By [24], for these representations, the $\mathrm{SL}(2, \mathbb{F}_{q^e})$-oracle is reduced to a discrete log oracle for $\mathbb{F}_{q^e}$. The constructive membership problem for classical groups is solved in polynomial time using the algorithms of [25]. Subject to an order oracle, and discrete log oracle for $\mathbb{F}_{q^e}$ where $e \leq 3$, the work of [22, 23, 40] provides constructive recognition algorithms, which are both practical and run in Las Vegas polynomial time, for defining characteristic representations of those exceptional groups handled by [35].

We comment briefly on the exclusions among the exceptional groups. The polynomial-time constructive recognition algorithms of [16, 3] for ${}^2B_2(2^{2k+1})$ and ${}^2G_2(3^{2k+1})$ apply only to their smallest dimensional defining characteristic faithful irreducible matrix representations. Bäärnhielm [2] gives a constructive recognition algorithm for the equivalent representation of ${}^2F_4(2^{2k+1})$. Using [23, 40], we obtain an $O(2^k)$ constructive recognition algorithm for defining characteristic representations of ${}^3D_4(2^k)$. All assume the availability of a discrete log oracle.

Building on this work, we assert the existence of Monte Carlo polynomial-time algorithms to solve the following tasks. The first of these is $\mathrm{BB}^{\mathrm{o}+}$, the second is $\mathrm{BB}^{\mathrm{od}+}$.

- NameSimple(G, ϵ). If G is a nonabelian simple black-box group of order less than 2^N then, with probability at least $1 - \epsilon$, the name of G is returned. Otherwise the algorithm returns False or an incorrect name.

- StandardGens(G, S, ϵ) where S is the name of a nonabelian simple group of order less than 2^N and not one of ${}^2B_2(2^{2k+1})$, ${}^2F_4(2^{2k+1})$, ${}^3D_4(2^k)$, or ${}^2G_2(3^{2k+1})$. Recall from Section 1 that each finite nonabelian simple group S has a specified sequence of *standard generators*. If the black-box group $G = \langle X \rangle$ is isomorphic to S, then, with probability at least $1 - \epsilon$, the algorithm returns True, a sequence of standard generators of G defined as words over X, and an algorithm to solve the constructive membership problem in G. Otherwise the algorithm returns False. StandardGens may fail to construct standard generators of a correctly named input simple group.

If $S = {}^2G_2(3^{2k+1})$, then we cannot currently construct standard generators in G in polynomial time, but the black-box algorithm of [28] solves the constructive membership problem in G in polynomial time.

We also use the following black-box algorithm of [45, Theorem 2.3.9 and Remark 2.3.5]. NormalClosure(G, g, ϵ) takes as input a black-box group G, an element g of G, and some ϵ with $0 < \epsilon < 1/2$. It returns a sequence of elements of G that, with probability at least $1 - \epsilon$, generates the normal closure $\langle g^G \rangle$ of g in G. If N is the encoding length of G, then the number of operations is bounded by $C\lceil (|X| + N) \log \epsilon^{-1} \rceil$ for some absolute constant C. NormalClosure operates *correctly* if it returns generators of $\langle g^G \rangle$.

3. Deciding simplicity

Theorem 3.1. *There is a Monte Carlo polynomial-time* $\mathrm{BB}^{\mathrm{od}+}$ *algorithm* TestSimple *that takes as input a nonabelian black-box group* $G = \langle X \rangle$ *and* $\epsilon \in (0, 1/2)$ *and outputs one of the following:*

(i) True, *the name of* G*, and a constructive membership algorithm for* G*;*
(ii) False *and* $w \in G$*;*
(iii) Fail, *possibly with the report that* G *may have one of the composition factors excluded by condition* (b) *below.*

This output is deemed to be correct *if one of the following holds:*

(1) *G is simple,* True *is returned, the correct name for G is returned, and the constructive membership algorithm for G that is returned writes elements g of G as words over X that evaluate to g; or*
(2) False *and w are returned, and w is nontrivial and lies in a proper normal subgroup of G.*

Otherwise the output is deemed to be incorrect*; in particular, the output* Fail *is incorrect.*

Suppose in addition that G satisfies at least one of the following conditions:

(a) *G is not perfect;*
(b) *G has no quotient that is isomorphic to ${}^2B_2(2^{2k+1})$, ${}^2F_4(2^{2k+1})$, or ${}^3D_4(2^k)$, for any k.*

Then the probability that the output is incorrect is less than ϵ. If the output is (i)*, then it is guaranteed to be correct except when $G \cong {}^2G_2(3^{2k+1})$ for some k.*

We first describe TestSimple and then prove Theorem 3.1. TestSimple makes frequent calls to an auxiliary function UpdateWitness. The "witness" in question is $1 \neq w \in G$. As input, UpdateWitness takes the existing witness w together with $h \in G$, which we think of as a candidate for membership in a proper normal subgroup of G. With high probability, the new value of w output by UPDATEWITNESS will lie in a proper normal subgroup of G if at least one of h or the original w does. So, after the complete run of TESTSIMPLE, if any of the elements h to which UPDATEWITNESS was applied lies in a proper normal subgroup of G, then so does the final value of w.

Another variable DONE is set to TRUE if at any point during the run of TESTSIMPLE we know (with high probability) that the current witness w lies in a proper normal subgroup of G. We then immediately halt the run of TESTSIMPLE and return FALSE and w.

Here is the pseudocode for UPDATEWITNESS. The group G, its generating set X, and ϵ, all referenced by UPDATEWITNESS, comprise the input to TESTSIMPLE, and are never changed. The witness w is initialised to a nontrivial commutator $[x, y]$ for $x, y \in X$ (recall G is nonabelian), and DONE is initialised to FALSE.

Function UPDATEWITNESS($h, w,$ DONE)

```
$Y$ := NormalClosure($G, h, \epsilon/c$) for some positive integer $c$;
for $y$ in $Y$ do
    if $[w, y] \neq 1$ then $w := [w, y]$; return $w$, Done; end if;
end for;
Done := True;
for $x$ in $X$ do
    if $[h, x] \neq 1$ then return $w$, Done; end if;
end for;
$w := h$;
return $w$, Done;
```

The value of the integer c in the first line is specified in Lemma 3.3 below. Let us assume that NORMALCLOSURE operates correctly, so $\langle Y\rangle = \langle h^G\rangle$. In most runs of UPDATEWITNESS we do not expect w to lie in the centraliser of $\langle h^G\rangle$ in G, and w is replaced by a nontrivial commutator of itself with a generator of that normal closure. But if $w \in C_G(\langle h^G\rangle)$ then DONE is set to TRUE. In this case, the next step tests if $h \in Z(G)$. If so, then we replace the witness w by h, which lies in the proper normal subgroup $Z(G)$ of G (recall G is nonabelian). If not, then $C_G(\langle h^G\rangle)$ is a proper normal subgroup of G that contains w, so we leave w unchanged.

The procedure TESTSIMPLE also uses modifications of NAMESIMPLE and STANDARDGENS, which we call NAMESIMPLE$^+$ and STANDARDGENS$^+$. The modifications are as follows. (The motivation for these modifications will become clear during the proof of Theorem 3.1.) Recall that a BB^{o+} algorithm applied to a group G has access to an order oracle FACTOREDORDER(g) for $g \in G$. Whenever we call FACTOREDORDER(g) on any element g in either of these two procedures, we follow this call immediately by invocations of UPDATEWITNESS($g^{n/p}, w,$ DONE) for each prime p that divides the order n of g. So if any nontrivial power of g lies in a proper normal subgroup of G then, after this call to UPDATEWITNESS, the same is true of the witness w. Furthermore, if DONE is not TRUE after this call, then w remains an element of $[G, G]$. But if DONE is set to TRUE, then we immediately abort the call to NameSimple$^+$ or StandardGens$^+$ and also that to TestSimple, and return False and w.

We now give a top-level outline of TestSimple. Recall from the statement of Theorem 3.1 that its input consists of a nonabelian black-box group $G = \langle X \rangle$ and $\epsilon \in (0, 1/2)$.

(1) Initialise Done to False and w to a nontrivial commutator $[x, y]$ for some $x, y \in X$.
(2) Run NameSimple$^+(G, \epsilon/3)$. If Done is now True, or if NameSimple$^+$ returns False, then return False and the witness w.
(3) Otherwise NameSimple$^+$ returns the name S of a finite simple group. If S is one of ${}^2B_2(2^{2k+1})$, ${}^2F_4(2^{2k+1})$, or ${}^3D_4(2^k)$, then this contradicts the hypotheses of the second part of Theorem 3.1, so we return Fail, and report the reason for failure. Assume that $S \neq {}^2G_2(3^{2k+1})$: we consider that case later.
(4) Run StandardGens$^+(G, S, \epsilon/3)$. If Done is now True, or if StandardGens$^+$ returns False, then return False and w.
(5) Otherwise StandardGens$^+$ returns True, a list Y of group elements, and a constructive membership algorithm for G. Decide whether Y satisfies the known short presentation of S on its standard generators. If so, then return True, the name of S, and the constructive membership algorithm.
(6) Otherwise, one of the relators of the presentation evaluates to a nontrivial element h. Call UpdateWitness(h, w, Done) and then return False and w.

Lemma 3.2. *The number of calls to* UpdateWitness *in a run of* TestSimple *is bounded by a polynomial function* $f(N|X|, \log \epsilon^{-1})$.

Proof. Apart from the single call in the final step of TestSimple, UpdateWitness is called only from within NameSimple$^+$ and StandardGens$^+$, each of which is called at most once by TestSimple. Since $|G| \leq 2^N$, there are at most N prime divisors of $|G|$, and so UpdateWitness is called at most N times following each call to FactoredOrder from NameSimple$^+$ and StandardGens$^+$. Since NameSimple and StandardGens are Monte Carlo polynomial-time algorithms, their running times are both bounded by polynomial functions of the input length $N|X|$ and $\log \epsilon^{-1}$, and so the same applies to the total number of calls to FactoredOrder. The result now follows. $\square$

Lemma 3.3. *Let f be the polynomial function in Lemma* 3.2 *and let c be* $3f(N|X|, \log \epsilon^{-1})$ *in the call to* NormalClosure *from* UpdateWitness. *Then the probability that all calls to* NormalClosure *operate correctly (that is, they return generators of the normal closure in G of the input element g) is at least* $1 - \epsilon/3$.

Proof. Note that all calls to NormalClosure in a run of TestSimple occur from within a call to UpdateWitness. By Lemma 3.2 there are at most $c/3$ such calls, so the probability that they all operate correctly is at least $(1 - \epsilon/c)^{c/3} > 1 - \epsilon/3$. $\square$

Lemma 3.4. TestSimple *is a polynomial-time* $\mathrm{BB}^{\mathrm{od}+}$ *algorithm.*

Proof. NameSimple$^+$, StandardGens$^+$, and NormalClosure are polynomial-time BB$^{\text{od}+}$ algorithms. We saw in Lemma 3.2 that the number of calls to UpdateWitness and NormalClosure is bounded by $f(N|X|, \log \epsilon^{-1})$, a polynomial function. Furthermore, the time taken by each call to NormalClosure is a polynomial function of $N|X|$ and $\log(c/\epsilon)$. Since $\log(c/\epsilon)$ is also bounded by a polynomial function of $N|X|$ and $\log \epsilon^{-1}$, the same applies to the time taken by all calls to UpdateWitness. Finally, the presentations of the simple groups used to check correctness are short, so their length is bounded by a polynomial function of $\log |G| \le N$. □

Proof of Theorem 3.1: Suppose first that G is simple. Suppose also that all calls to NormalClosure operate correctly which, as we saw in Lemma 3.3, happens with probability at least $1 - \epsilon/3$. Since G is simple, all such calls return generators of G and, since $w \notin Z(G)$, Done does not become True during calls to UpdateWitness, which would cause TestSimple to abort. So, with probability at least $1 - 2\epsilon/3$, NameSimple$^+$ correctly identifies G, and StandardGens$^+$ correctly finds standard generators that satisfy the relators of the presentation. So TestSimple correctly returns True, the name of G, and the constructive membership algorithm for G. Hence, when G is simple, the output of TestSimple is correct with probability at least $1 - \epsilon$.

On the other hand, since TestSimple only returns True when it finds standard generators of G and verifies that they satisfy the relations of a known presentation of a simple group S, a return value of True is guaranteed to be correct. (We still assume for now that G has no quotient isomorphic to ${}^2G_2(3^{2k+1})$ for any k.)

Hence we may assume that TestSimple returns False and that G is not simple. We need to prove that the returned element w is correct (that is, $w \neq 1$ and w lies in a proper normal subgroup of G) with probability at least $1 - \epsilon$. By Lemma 3.3 all calls to NormalClosure operate correctly with probability at least $1 - \epsilon/3$. We shall assume that this is the case and prove that, under this assumption, the returned w is correct with probability at least $1 - 2\epsilon/3$.

If Done becomes True at any point, then our assumption that all calls to NormalClosure operate correctly implies that the returned w is correct. Assume that Done never becomes True. As we observed earlier, $w \in [G, G]$ in this case, so the result returned is also correct when G is not perfect. So we may assume that G is perfect.

Hence G has a nontrivial normal subgroup K with G/K nonabelian simple. We recall that the witness w is updated during the run of TestSimple by the application of UpdateWitness to $h \in G$; each h is a power $g^{n/p}$ of g, where p is a prime divisor of the order n of g, and FactoredOrder(g) is called by either NameSimple$^+$ or StandardGens$^+$. When w is updated, it is replaced by the commutator $[w, y]$ for some element y of the normal closure of h in G (recall we assume that Done does not become True). If, at any time during the run, $w \in K$, or if any of the elements h to which UpdateWitness is applied lies in K, then the final witness w returned by TestSimple lies in K; so TestSimple operates correctly.

If neither of these conditions is satisfied, then, for each call to FactoredOrder(g) from NameSimple$^+$, the order of g must be the same as the order of gK in $G/K \cong S$, since otherwise $g^{n/p} \in K$ for some prime divisor p of the order n of g. But this means that NameSimple$^+$ operates exactly as it would if the input group were G/K with generators $\{xK : x \in X\}$. So, with probability at least $1 - \epsilon/3$, NameSimple$^+$ (incorrectly) returns the name S. This argument relies critically on our assumptions about the algorithm used to generate random elements of a black-box group, as discussed at the end of Section 2.2.

By a similar argument, if $w \notin K$ after the call to StandardGens$^+$ then, with probability at least $1 - \epsilon/3$, StandardGens$^+$ returns True and a set of elements whose images in G/K are standard generators of S. But now at least one of the relators of the known presentation of S must evaluate to a nontrivial element of K, and so $w \in K$ after the final call to UpdateWitness. So TestSimple correctly returns False and $w \in K$. (We make this final call to UpdateWitness rather than just returning h to ensure that $w \in [G, G]$, a requirement at this point in the proof.)

This completes the proof under the assumption that $S \neq {}^2G_2(3^{2k+1})$ for any k. Finally, we describe the modifications to TestSimple when NameSimple$^+$ returns $S = {}^2G_2(3^{2k+1})$ for some k in Step (3) of TestSimple. When that happens, rather than proceeding to Step (4), we use the constructive membership algorithm of [28] to write $\lceil \log_2(3\epsilon^{-1}) \rceil$ random $g \in G$ as words over X. If False is returned for any of these elements, then we return False and the witness w. Otherwise, we evaluate the word returned for g to obtain $g' \in G$. If $g = g'$ for every g, then we return True and the constructive membership algorithm. Otherwise some $g \neq g'$; now we put $h := g^{-1}g'$, call UpdateWitness(h, w, Done), and return False and w.

As before, if $G \cong {}^2G_2(3^{2k+1})$, then the correct result is returned with probability at least $1 - \epsilon$. Otherwise, the proof proceeds as before, and leads us to the situation in which there is a nontrivial normal subgroup K of G with $G/K \cong {}^2G_2(3^{2k+1})$. If $w \in K$ when TestSimple completes, then it has behaved correctly, so assume not. We claim that the constructive membership algorithm must write elements in the same coset of K to the same word over X. This is because, as we have seen earlier, $w \notin K$ implies that the black-box algorithms behave in the same way as they would if the black-box group were G/K rather than G, and hence they do not distinguish between elements in the same coset of K. Thus the proportion of elements of G for which the word returned by the constructive membership algorithm is *correct* (namely, it evaluates to that element) is at most $1/|K|$. Hence the probability that the constructive membership algorithm writes a random element of G correctly is at most $1/|K|$, and hence the probability of writing $\lceil \log_2(3\epsilon^{-1}) \rceil$ such elements correctly is at most $\epsilon/3$. So the probability of True being returned incorrectly is at most ϵ. On the other hand, by a similar argument to the main proof, if any of the elements fail to write correctly, then w lies in K with probability at least $1 - \epsilon$. □

4. Deciding reductions for matrix groups

Aschbacher [1] showed that maximal subgroups of classical groups over finite fields are in one of nine classes, which he called $\mathcal{C}_1 - \mathcal{C}_8$ and $\mathcal{S}$. We extend (or abuse) this notation by applying it to arbitrary subgroups of maximal subgroups in classes $\mathcal{C}_1 - \mathcal{C}_7$. For example, we view every reducible matrix group as a member of class $\mathcal{C}_1$. Viewed in this way, the main result of [1] is that every subgroup G of $\mathrm{GL}(d, \mathbb{F}_q)$ either lies in at least one of $\mathcal{C}_1 - \mathcal{C}_7$, or it lies in $\mathcal{C}_8$ or $\mathcal{S}$, in which case G has a normal absolutely irreducible subgroup that is simple modulo scalars, and G does not lie in class $\mathcal{C}_5$ (that is, it is not defined over a proper subfield of $\mathbb{F}_q$ modulo scalars). A description of these nine classes appears in [43, Theorem 5.1].

We use the Monte Carlo algorithm TestSimple of Section 3 to prove the following theorem. The operations of the final sentence are multiplication and inversion of matrices; a discrete log oracle for $\mathbb{F}_{q^i}$ for $1 \leq i \leq d$; and an oracle to factorise integers of the form $q^i - 1$ for $1 \leq i \leq d$ (which, as explained in Section 2.3, allows us to compute and factorise the orders of elements of G).

Theorem 4.1. *Let $G = \langle X \rangle \leq \mathrm{GL}(d, \mathbb{F}_q)$ for some $d > 1$. There is a randomised algorithm* SearchForDecomposition(X, ϵ) *that, provided G has no composition factor isomorphic to ${}^2B_2(2^{2k+1})$, ${}^2F_4(2^{2k+1})$, or ${}^3D_4(2^k)$, for any k, either:*

1. *finds a normal absolutely irreducible subgroup K of G such that $K/Z(K)$ is nonabelian simple, and asserts that $K/Z(K) \cong S$ for some named simple group S, and that G is in Aschbacher class $\mathcal{C}_8$ or $\mathcal{S}$; or*
2. *finds a decomposition of G in its action on the underlying vector space $V \cong \mathbb{F}_q^d$ which is dictated by its membership of one of the Aschbacher classes $\mathcal{C}_1 - \mathcal{C}_7$; or*
3. *returns* Fail *with probability less than ϵ.*

If (1) *applies, then the algorithm returns* False*, K, the name of S, and a constructive membership algorithm for K; otherwise, it returns* True *and the Aschbacher decomposition obtained for G.*

The assertion in (1) *is guaranteed to be correct except when $K/Z(K) \cong {}^2G_2(3^{2k+1})$ for some k, in which case there is an error probability of at most ϵ. If* (2) *applies, then the decomposition returned is guaranteed to be correct.*

SearchForDecomposition *is guaranteed to halt after at most*

$$f(|X|, N, \log \epsilon^{-1})$$

operations for some polynomial function f, where $N = d^2 \lceil \log_2 q \rceil$.

We comment briefly on the requirements for the discrete log oracle. As mentioned earlier, we may need this for fields $\mathbb{F}_{q^i}$, where $1 \leq i \leq d$, to process cyclic composition factors of G. By [36], nonabelian composition factors that are groups of Lie type over fields of order r coprime to q arise only when d is bounded below by r, and so discrete logs over fields of order r^k for $k \leq 3$ can be computed in polynomial time without recourse to an oracle. For Lie type composition factors in characteristic that of $\mathbb{F}_q$, we require discrete logs for fields of order at most q^d.

SearchForDecomposition uses the algorithm Smash described in [29]. This is effectively an algorithmic realisation of Clifford's theorem [21] about decompositions of V preserved by a nonscalar normal subgroup of G. That Smash runs in Monte Carlo polynomial time is established in [29, §5].

Certain types of normal subgroups of G give rise to a decomposition of V that immediately implies that G lies in one of the classes $\mathcal{C}_1 - \mathcal{C}_7$. For example, if G acts irreducibly on V, but $K \lhd G$ acts inhomogeneously on V, then G is imprimitive; we refer to this situation as a decomposition of G of type $\mathcal{C}_2$ where K lies in the kernel of the decomposition.

More precisely, Smash(X, w, ϵ) takes as input generators X of an absolutely irreducible subgroup G of $\mathrm{GL}(d, \mathbb{F}_q)$, a nonscalar $w \in G$, and a maximum error probability ϵ. It searches for an Aschbacher decomposition of G of type $\mathcal{C}_2$, $\mathcal{C}_3$, $\mathcal{C}_4$, $\mathcal{C}_6$, or $\mathcal{C}_7$ in which $\langle w^G \rangle$ lies in the kernel of the decomposition. The following proposition summarises the behaviour of Smash.

Proposition 4.2. *Assume that G is an absolutely irreducible subgroup of* $\mathrm{GL}(d, \mathbb{F}_q)$ *and that w is a nonscalar element of G. Let $K = \langle w^G \rangle$.*

(i) *If K is not absolutely irreducible, then K lies in the kernel of an Aschbacher decomposition of type $\mathcal{C}_2$, $\mathcal{C}_3$ or $\mathcal{C}_4$, and this decomposition will be found by* Smash.
(ii) *If K is absolutely irreducible and $KZ(G)/Z(G)$ is elementary abelian, then K lies in the kernel of an Aschbacher decomposition of type $\mathcal{C}_6$, and this decomposition will be found by* Smash.
(iii) *If K is absolutely irreducible and $KZ(G)/Z(G)$ is a nonabelian minimal normal subgroup of $G/Z(G)$ that is not simple, then K lies in the kernel of an Aschbacher decomposition of type $\mathcal{C}_7$, and this decomposition will be found by* Smash *with probability at least $1 - \epsilon$.*

Proof. The claim is proved in [29, §2] apart from the error probability of (iii). In that case G is tensor induced and w lies in the kernel K of the permutation action of G on the k tensor factors for some $k > 1$, where K is isomorphic modulo scalars to the direct product of k copies of a finite nonabelian simple group S. To complete the construction of the decomposition, Smash needs to find an element of K that projects onto nonscalar elements of some but not all of these factors. Let g be a random element of K. If the prime divisors of the orders of the projections of g onto the k copies of S are not identical, then the required element is $g^{n/p}$, where p is some prime divisor of the (projective) order n of g.

It is proved in [12] that, if a nonabelian simple group is a quotient of a subgroup of $\mathrm{GL}(d, \mathbb{F}_q)$, then, for every prime r, the proportion of elements having projective order not divisible by r is at least $\rho := \min\{1/31, 1/2d\}$. Hence the probability that a random nonscalar $g \in K$ has the property described above is at least ρ^{k-1}. So, by considering powers $g^{n/p}$ of an appropriate number of random $g \in K$, we can find an element with the property required by SMASH with probability at least $1 - \epsilon$. □

Proof of Theorem 4.1: There is a Las Vegas polynomial-time algorithm [30, 31] to decide if a given subgroup G of $\mathrm{GL}(d, \mathbb{F}_q)$ acts irreducibly or absolutely irreducibly on V. If so, then this algorithm returns a new basis of V to exhibit the decomposition, G lies in one of the Aschbacher classes $\mathcal{C}_1$ or $\mathcal{C}_3$, and Case (2) of the statement of Theorem 4.1 holds. So we may assume that G acts absolutely irreducibly on V. In particular, since we assume that $d > 1$, G must be nonabelian.

The Las Vegas polynomial-time algorithm of [27] takes as input an absolutely irreducible subgroup G of $\mathrm{GL}(d, \mathbb{F}_q)$, and, provided $[G, G]$ acts absolutely irreducibly on V, decides whether there is a proper subfield $\mathbb{F}_s$ of $\mathbb{F}_q$ such that G is conjugate in $\mathrm{GL}(d, \mathbb{F}_q)$ to a subgroup of $\mathrm{GL}(d, \mathbb{F}_s)Z(\mathrm{GL}(d, \mathbb{F}_q))$. If so, then the algorithm returns a conjugating matrix; so G lies in the Aschbacher class $\mathcal{C}_5$, and Case (2) of the statement of Theorem 4.1 again holds.

Thus we may assume that either $[G, G]$ does not act absolutely irreducibly on V, or G does not lie in the class $\mathcal{C}_5$. We now present pseudocode for SEARCHFORDECOMPOSITION under this assumption. Recall that our goal is to recognise constructively a normal absolutely irreducible quasisimple subgroup of G, or to construct an Aschbacher decomposition of G. All calculations are done modulo the scalar subgroup Z of $\mathrm{GL}(d, \mathbb{F}_q)$, including the application of TESTSIMPLE. For the black-box algorithms, this is achieved by making the 'is-identity' oracle return TRUE if and only if its input matrix is scalar. The function $\lambda(d, q)$ in Line 3 bounds the length of a subnormal series of subgroups of $\mathrm{GL}(d, \mathbb{F}_q)$, which we take to be $d^2 \log q$ (although better bounds are known).

Recall that TESTSIMPLE returns at most three values FOUND, VALUE, ALG, where FOUND is TRUE, FALSE or FAIL; VALUE is the name of a simple group when FOUND is TRUE, and a witness for a proper normal subgroup when FOUND is FALSE; and ALG is a constructive membership algorithm for the simple group when FOUND is TRUE. Also SMASH returns values FOUND and DECOMP, where FOUND is TRUE if an Aschbacher decomposition is found, in which case DECOMP is that decomposition, and FOUND is FALSE otherwise.

Function SEARCHFORDECOMPOSITION(X, ϵ)

```
1  $G := \langle X \rangle$;  $K := G$;  $Y_K := X$;   ($K = \langle Y_K \rangle$ will remain a subnormal subgroup of $G$)
2  $w :=$ any nonscalar element of $X$;
3  $\epsilon' := \epsilon/(4\lambda(d, q))$;
4  for ct in $[1 \,..\, \lambda(d, q)]$ do
5      if IsAbelian($KZ/Z$) then
6          $w := w^t$ with $t \geq 0$, where $p := o(w^t Z/Z)$ is prime, and $o(w^t)$ is a power of $p$;
7          $L := \langle$NormalClosure($X, w, \epsilon'$)$\rangle$;   (so $L \leq O_p(G)$)
8          $w := z$ with $zZ$ central of order $p$ in $LZ/Z$;
9      else
```

```
10          FOUND, VALUE, ALG := TESTSIMPLE($Y_K, \epsilon'$); (applied modulo scalars)
11          if FOUND then
12              if $K \trianglelefteq G$ then return FALSE, $K$, its name VALUE, and the algorithm ALG;
13              else return FAIL;
14              end if;
15          else
16              $w$ := VALUE;
17          end if;
18      end if;
19      FOUND, DECOMP := SMASH($X, w, \epsilon'$);
20      if FOUND then return TRUE, DECOMP; end if;
21      $Y_K$ := NORMALCLOSURE($Y_K, w, \epsilon'$);
22      $K := \langle Y_K \rangle$;
23  end for;
24  return FAIL;
```

We can decide whether K is normal in G in Line 12 by using the constructive membership algorithm for K that is returned by TESTSIMPLE as ALG. Since L is a p-group in Line 7, we can easily find a nontrivial central element of LZ/Z by repeatedly taking commutators.

There are four calculations in the above function that can return FAIL or an incorrect answer: TESTSIMPLE in Line 10, SMASH in Line 19, and NORMALCLOSURE in Lines 7 and 21. Each is executed at most $\lambda(d, q)$ times; to make the total probability of failure at most ϵ, we choose our parameters so that each fails with probability at most $\epsilon' = \epsilon/(4\lambda(d, q))$. Since the decompositions returned by SMASH are guaranteed to be correct, the only way that SEARCHFORDECOMPOSITION can return a wrong answer, rather than FAIL, is for TESTSIMPLE to incorrectly identify a black-box group as ${}^2G_2(3^{2k+1})$ for some k.

Let us assume that none of the above causes of failure arises. Then we must prove that, at some stage, either the subgroup K of G is proved to be simple modulo scalars in Line 10 and normal in G in Line 12, or else SMASH finds a decomposition in Line 19. So suppose that neither of these eventualities occurs.

Note that K is redefined only in Line 22, where it is replaced by the normal closure in K of a nontrivial element; so K remains a subnormal subgroup of G throughout.

Assume first that, at some stage, ISABELIAN returns TRUE in Line 5, so K is abelian modulo scalars. The element w defined in Line 6 lies in $O_p(K)$ and hence in $O_p(G)$, and its normal closure L in G defined in Line 7 lies in $O_p(G)$. Observe that w is redefined in Line 8 as an element whose image in LZ/Z is central and of order p. So the normal closure M in G of the new w has elementary abelian image in LZ/Z. Now, by Proposition 4.2, the call to SMASH in Line 19 will find a decomposition of type $\mathcal{C}_2$, $\mathcal{C}_3$ or $\mathcal{C}_4$ if M is not absolutely irreducible, or of type $\mathcal{C}_6$ if it is.

Assume now that ISABELIAN never returns TRUE in Line 5. If we reach Line 19, then the element w is a witness from a call to TESTSIMPLE that returned FALSE at Line 10, and so it lies in a proper normal subgroup of K. So, if SMASH fails to find a decomposition, then K is redefined at Line 22 as a proper normal subgroup of itself. Since $\lambda(d, q)$ bounds the length of a subnormal series of G, and we assume that K is never abelian modulo scalars, it must eventually be simple modulo scalars. Now TESTSIMPLE returns TRUE in Line 10. It can be shown by induction on the subnormal depth that a nonabelian simple subnormal subgroup of a group lies in the socle of that group, and so KZ/Z lies in the socle of GZ/Z. Hence, if K is not normal in G, then the normal closure of KZ/Z in GZ/Z is a minimal normal subgroup of GZ/Z isomorphic to a direct product of more than one copy of KZ/Z. But then, by Proposition 4.2 (iii), SMASH should have returned a $\mathcal{C}_7$-decomposition of G in the previous iteration of the **for** loop. This justifies the return of FAIL in Line 13.

On the other hand, if $K \trianglelefteq G$ at Line 12, then K must act absolutely irreducibly, since otherwise SMASH would have found a decomposition on the previous iteration of the **for** loop by Proposition 4.2. So $[G, G]$ acts absolutely irreducibly on V and hence, by the second paragraph of this proof, G is not in the Aschbacher class $\mathcal{C}_5$. So Case (1) of the statement of Theorem 4.1 holds (the definition of the class $\mathcal{S}$ requires that $[G, G]$ acts absolutely irreducibly and that G is not in $\mathcal{C}_5$). □

4.1. **The use of** ISPERFECT**.** We have proved Theorem 4.1 as a theoretical result, but we also claim that implementations of (variants of) these methods perform well in practice. We can improve their performance by the use of an additional $\mathrm{BB}^{\mathrm{o}+}$ algorithm, which we now briefly describe.

Let $G = \langle X \rangle$ be a black-box group with normal subgroup L defined by a generating set. A $\mathrm{BB}^{\mathrm{o}+}$ algorithm is given in [37, §5.3] to compute a multiplicative upper bound to the order modulo L of $g \in G$. Deciding whether this upper bound is 1 provides a membership test for g in L that is guaranteed to be correct when the answer is positive. Babai & Shalev [9, §4.4] prove that this provides a Monte Carlo polynomial-time membership test when L is a nonabelian simple group.

Consider $L = [G, G]$. We first use NORMALCLOSURE (on commutators of elements of X) to find generators of L, and then apply the membership test to all elements of X to produce an algorithm ISPERFECT to decide whether G is perfect; a positive answer is guaranteed to be correct. We enhance this to ISPERFECT$^+$ by calling UPDATEWITNESS, as we did for NAMESIMPLE and STANDARDGENS.

Now we modify SEARCHFORDECOMPOSITION as follows. We recall that this function maintains a set of group elements Y_K that generate a subnormal subgroup of the input group $G = \langle X \rangle$. We call ISPERFECT(Y_K, ϵ') for suitable ϵ' immediately before the call to TESTSIMPLE in Line 10 and, if it returns FALSE, then we proceed immediately to the call to SMASH in Line 19.

We justify these modifications as follows. If G is not perfect, then ISPERFECT returns FALSE and the witness is in the proper normal subgroup $[G, G]$ of G, so we avoid the more time-consuming call to TESTSIMPLE. If G is nonabelian simple, then ISPERFECT returns TRUE with probability at least $1 - \epsilon'$, so we proceed with the call of TESTSIMPLE. If G

is perfect but not simple, then IsPerfect may incorrectly return False, and we have no estimates for the probability of this happening. In that situation, since the algorithm has behaved differently from what it would have done if G had been simple, we use the same argument as in the proof of Theorem 3.1 to conclude that, with high probability, the witness w lies in a proper normal subgroup of G.

4.2. Constructing generators of $Z(K)$.

To compute a composition tree of a linear group, we need to find generators of $Z(K)$ when Case (1) of Theorem 4.1 occurs. If we have a short presentation of $K/Z(K)$ on its standard generators, then we can do that in polynomial time as follows.

Let Y be a set of inverse images in K of the standard generators $\overline{Y}$ of $K/Z(K)$. For a word w over $\overline{Y}$, let $\sigma(w) \in K$ be the result of evaluating w with $\bar{y} \in \overline{Y}$ replaced by the corresponding $y \in Y$. Let R be the set of defining relators of the known presentation of $K/Z(K)$, and let $A = \{\sigma(r) : r \in R\} \subseteq Z(K)$. Let ρ be the rewriting map that maps elements of $K/Z(K)$ to words over $\overline{Y}$, let X be our given generating set of K, and let $B = \{x \cdot \sigma(\rho(xZ(K)))^{-1} : x \in X\} \subseteq Z(K)$. It can be shown readily that $Z(K)$ is generated by $A \cup B$.

If $K/Z(K) \cong {}^2G_2(3^{2k+1}))$, then we have no short presentation. Instead we obtain a Monte Carlo algorithm to construct generators for $Z(K)$ by evaluating $g \cdot \sigma(\rho(gZ(K)))^{-1}$ for a collection of random $g \in K$.

5. A polynomial-time version of CompositionTree

We summarise a mildly simplified version of the CompositionTree algorithm presented in [4, §3.1]. It takes as input $G = \langle X \rangle \leq \mathrm{GL}(d, \mathbb{F}_q)$ and outputs a composition tree for G.

(1) Do one of the following:
 (i) construct an effective epimorphism $\theta : G \to G_1$, for some group G_1; or
 (ii) prove that G is simple, in which case G becomes a *leaf* in the tree.
 In Case (i), θ must be a *reduction*: namely, G_1 is "smaller" than G in some respect – for example, its degree or field of definition. Assume henceforth that Case (i) applies.
(2) Recursively construct a composition tree for $G_1 := \langle \theta(Y) \rangle$.
(3) Construct generators for $G_0 := \operatorname{Ker} \theta$.
(4) Recursively construct a composition tree for G_0.
(5) Combine the composition trees for G_1 and G_0 into a tree for G.

Theorem 5.1. *Let $G = \langle X \rangle \leq \mathrm{GL}(d, \mathbb{F}_q)$ and assume that G has no composition factor isomorphic to ${}^2B_2(2^{2k+1})$, ${}^2F_4(2^{2k+1})$, or ${}^3D_4(2^k)$, for any k. Then, subject to the existence of a discrete log oracle for $\mathbb{F}_{q^i}$ and an oracle to factorise integers of the form $q^i - 1$ for $1 \leq i \leq d$,* CompositionTree *runs in polynomial time. The resulting algorithm is Las Vegas if G has no composition factor isomorphic to ${}^2G_2(3^{2k+1})$ and Monte Carlo if there is such a factor.*

Proof. If $d = 1$, then G is cyclic: it is straightforward to use a discrete log oracle and the factorisation of $q - 1$ to find a composition series of G. So we may assume that $d > 1$ and hence G satisfies the hypotheses of Theorem 4.1.

Theorem 4.1 shows that Step 1 can be carried out in polynomial time subject to the existence of the required oracles. Since the number of composition factors of G is at most $\log|G|$, the recursive Steps 2 and 4 run in polynomial time. If a decomposition of type $\mathcal{C}_6$ arises, then (as shown, for example, in [29, §2]) $d = r^k$ for some prime $r \neq p$ and $G_1 \leq \mathrm{GL}(2k, r)$. Processing G_1 requires calculating discrete logs in fields of order r^i with $1 \leq i \leq 2k$, but each order is at most d^2, so we do not need a discrete log oracle for this purpose.

Step 3 is described in [4, §5.3.1]. Once Step 2 is complete, we can evaluate images of $g \in G$ under θ, and also images of elements of G_1 under a map $\sigma : G_1 \to G$ such that $\theta \circ \sigma = 1_{G_1}$, all in polynomial time. We can now construct random elements of G_0 as $g \cdot \sigma(\theta(g))^{-1}$ for random $g \in G$. As shown in [4, §5.3], the number of random elements required to generate G_0 with high probability is $O(\log|G|)$. Step 5, described in [4, §§4.4–4.5], is straightforward once the required data structures have been set up. □

Of course, we must calculate failure (or, in the case of a composition factor ${}^2G_2(3^{2k+1})$, error) probabilities for individual steps in this process to ensure that the overall failure (or error) probability for CompositionTree is at most the chosen value of ϵ, but this is straightforward given the known bounds on the number of steps in the recursive process.

References

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

[2] Henrik Bäärnhielm. *Algorithmic problems in twisted groups of Lie type.* PhD thesis, Queen Mary, University of London, 2007.

[3] Henrik Bäärnhielm. Recognising the small Ree groups in their natural representations. *J. Algebra* **416** (2014), 139–166.

[4] Henrik Bäärnhielm, Derek Holt, C. R. Leedham-Green, and E. A. O'Brien. A practical model for computation with matrix groups. *J. Symbolic Comput.* **68** (2015), 27–60.

[5] 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.

[6] László Babai. Local expansion of vertex-transitive graphs and random generation in finite groups. *Theory of Computing*, (Los Angeles, 1991), pp. 164–174. Association for Computing Machinery, New York, 1991.

[7] László Babai, Eugene M. Luks, and Ákos Seress. Fast management of permutation groups. I. *SIAM J. Comput.* **26** (1997), 1310–1342.

[8] L. Babai and R. Beals. A polynomial-time theory of black box groups. I. In *Groups St. Andrews 1997 in Bath, I*, volume 260 of *London Math. Soc. Lecture Note Ser.*, pages 30–64. Cambridge Univ. Press, Cambridge, 1999.

[9] L. Babai and A. Shalev. Recognizing simplicity of black-box groups and the frequency of p-singular elements in affine groups. In: Groups and Computation III, Ohio, 1999, Ohio State Univ. Math. Res. Inst. Publ., de Gruyter, Berlin, pages 39–62, 2000.

[10] L. Babai, W. M. Kantor, P. P. Pálfy, and Á. Seress. Black-box recognition of finite simple groups of Lie type by statistics of element orders. *J. Group Theory* **5** (2002), 383–401.

[11] L. Babai, R. Beals, and Á. Seress. Polynomial-time Theory of Matrix Groups. In: Proceedings of the 41st Annual ACM Symposium on Theory of Computing, STOC 2009, Bethesda, MD, USA, pages 55–64, 2009.

[12] L. Babai, P. P. Pálfy, and J. Saxl. On the number of p-regular elements in finite simple groups. *LMS J. Comput. Math.* **12** (2009), 82–119.

[13] Alexandre Borovik and Şükrü Yalçınkaya. Adjoint representations of black box groups $\mathrm{PSL}_2(\mathbb{F}_q)$. *J. Algebra* **506** (2018), 540–591.

[14] W. Bosma, J. Cannon, and C. Playoust. The Magma algebra system I: The user language. *J. Symbolic Comput.* **24** (1997), 235–265.

[15] J. N. Bray, M. D. E. Conder, C. R. Leedham-Green, and E. A. O'Brien. Short presentations for alternating and symmetric groups. *Trans. Amer. Math. Soc.* **363** (2011), 3277–3285.

[16] John N. Bray and Henrik Bäärnhielm. A new method for recognising Suzuki groups. *J. Algebra* **493** (2018), 483–499.

[17] John N. Bray and Henrik Bäärnhielm. Standard generators for the Suzuki groups. Preprint, 2019.

[18] John Brillhart, D.H. Lehmer, J.L. Selfridge, Bryant Tuckerman, and S.S. Wagstaff, Jr. *Factorizations of $b^n \pm 1$*, volume 22 of *Contemporary Mathematics*. American Mathematical Society, Providence, RI, second edition, 1988. http://www.cerias.purdue.edu/homes/ssw/cun/index.html.

[19] 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. Algebra* **23** (1995), 4931–4948.

[20] Frank Celler and C.R. Leedham-Green. Calculating the order of an invertible matrix. *Groups and Computation II*, American Mathematical Society, Providence, RI (1997), 55–60.

[21] A. H. Clifford. Representations induced in an invariant subgroup. *Ann. of Math.* **38** (1937), 533–550.

[22] Arjeh M. Cohen, Scott H. Murray, and D. E. Taylor. Computing in groups of Lie type. *Math. Comp.* **73** (2004), 1477–1498.

[23] Arjeh M. Cohen and D. E. Taylor. Row reduction for twisted groups of Lie type. Preprint, 2019.

[24] M. D. E. Conder, C. R. Leedham-Green, and E. A. O'Brien. Constructive recognition of $\mathrm{PSL}(2, q)$. *Trans. Amer. Math. Soc.* **358** (2006), 1203–1221.

[25] Elliot Costi. *Constructive membership testing in classical groups.* PhD thesis, Queen Mary, University of London, 2009.

[26] H. Dietrich, C. R. Leedham-Green, and E. A. O'Brien. Effective black-box constructive recognition of classical groups, *J. Algebra* **421** (2015), 460–492.

[27] S. P. Glasby, C. R. Leedham-Green, and E. A. O'Brien. Writing projective representations over subfields. *J. Algebra* **295** (2006), 51–61.

[28] P. E. Holmes, S. A. Linton, E. A. O'Brien, A. J. E. Ryba, and R. A. Wilson. Constructive membership in black-box groups. *J. Group Theory* **11** (2008), 747–763.

[29] D. F. Holt, C. R. Leedham-Green, E. A. O'Brien, and S. Rees. Computing matrix group decompositions with respect to a normal subgroup. *J. Algebra* **184** (1996), 818–838.

[30] D. F. Holt and S. Rees. Testing modules for irreducibility. *J. Austral. Math. Soc. Ser. A* **57** (1994), 1–16.

[31] Gábor Ivanyos and Klaus Lux. Treating the exceptional cases of the MeatAxe. *Experiment. Math.* **9** (2000), 373–381.

[32] S. Jambor, M. Leuner, A. C. Niemeyer, and W. Plesken. Fast recognition of alternating groups of unknown degree. *J. Algebra* **392** (2013), 315–335.

[33] William M. Kantor. Sylow's theorem in polynomial time. *J. Comput. System Sci.* **30** (1985), 359–394.

[34] William M. Kantor and Martin Kassabov. Black box groups isomorphic to $\mathrm{PGL}(2, 2^{\mathrm{e}})$. *J. Algebra* **421** (2015), 16–26.

[35] W. M. Kantor and K. Magaard. Black box exceptional groups of Lie type II. *J. Algebra* **421** (2015), 524–540.

[36] Vicente Landazuri and Gary M. Seitz. On the minimal degrees of projective representations of the finite Chevalley groups. *J. Algebra* **32** (1974), 418–443.

[37] C. R. Leedham-Green and E. A. O'Brien. Recognising tensor-induced matrix groups. *J. Algebra* **253** (2002), 14–30.

[38] C. R. Leedham-Green and E. A. O'Brien. Presentations on standard generators for classical groups. *J. Algebra* **545** (2020), 324–389.

[39] M. W. Liebeck and E. A. O'Brien. Finding the characteristic of a group of Lie type. *J. Lond. Math. Soc.* **75** (2007), 741–754.

[40] M. W. Liebeck and E. A. O'Brien. Recognition of finite exceptional groups of Lie type. *Trans. Amer. Math. Soc.*, **368** (2016), 6189–6226.

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

[42] Eugene M. Luks. Computing in Solvable Matrix Groups. In: *Proc. 33th IEEE Sympos. Foundations Comp. Sci.*, 1992, 111–120.

[43] E. A. O'Brien. Algorithms for matrix groups. Groups St Andrews 2009 in Bath II, London Math. Soc. Lecture Note Series **388** (2011), 297–323.

[44] Igor Pak. The product replacement algorithm is polynomial. In: *41st Annual Symposium on Foundations of Computer Science (Redondo Beach, CA, 2000)*, 476–485, IEEE Comput. Soc. Press, Los Alamitos, CA, 2000.

[45] Ákos Seress. *Permutation Group Algorithms.* Cambridge Tracts in Mathematics 152. Cambridge University Press, 2003.

[46] Igor E. Shparlinski. *Finite fields: theory and computation. The meeting point of number theory, computer science, coding theory and cryptography.* Mathematics and its Applications, 477. Kluwer Academic Publishers, Dordrecht, 1999.

[47] R. A. Wilson. *The finite simple groups.* Springer Verlag, London, 2009.

[48] R. A. Wilson *et al.* Atlas of Finite Group Representations. `brauer.maths.qmul.ac.uk/Atlas`.

Mathematics Institute, University of Warwick, Coventry CV4 7AL, United Kingdom

E-mail address: `D.F.Holt@warwick.ac.uk`

School of Mathematical Sciences, Queen Mary, University of London, London E1 4NS, United Kingdom

E-mail address: `c.r.leedham-green@qmul.ac.uk`

Department of Mathematics, University of Auckland, Auckland, New Zealand

E-mail address: `e.obrien@auckland.ac.nz`