2015

The Module Isomorphism Problem Reconsidered

Peter A. Brooksbank
pbrooksb@bucknell.edu

Journal of Algebra 421 (2015) 541–559

Contents lists available at ScienceDirect

Journal of Algebra

www.elsevier.com/locate/jalgebra

The module isomorphism problem reconsidered

Peter A. Brooksbank [a], James B. Wilson [b,*]

[a] *Department of Mathematics, Bucknell University, Lewisburg, PA 17837, USA*
[b] *Department of Mathematics, Colorado State University, Ft. Collins, CO 80523, USA*

ARTICLE INFO

Article history:
Received 29 April 2014
Available online 16 September 2014
Communicated by William M. Kantor and Charles Leedham-Green

Keywords:
Isomorphism
Modules
Conjugacy

ABSTRACT

Algorithms to decide isomorphism of modules have been honed continually over the last 30 years, and their range of applicability has been extended to include modules over a wide range of rings. Highly efficient computer implementations of these algorithms form the bedrock of systems such as GAP and Magma, at least in regard to computations with groups and algebras. By contrast, the fundamental problem of testing for isomorphism between other types of algebraic structures – such as groups, and almost any type of algebra – seems today as intractable as ever. What explains the vastly different complexity status of the module isomorphism problem?

This paper argues that the apparent discrepancy is explained by nomenclature. Current algorithms to solve module isomorphism, while efficient and immensely useful, are actually solving a highly constrained version of the problem. We report that module isomorphism in its general form is as hard as algebra isomorphism and graph isomorphism, both well-studied problems that are widely regarded as difficult. On a more positive note, for cyclic rings we describe a polynomial-time algorithm for the general module isomorphism problem. We also report on a Magma implementation of our algorithm.

* Corresponding author.
E-mail addresses: pbrooksb@bucknell.edu (P.A. Brooksbank), James.Wilson@ColoState.Edu (J.B. Wilson).

http://dx.doi.org/10.1016/j.jalgebra.2014.09.004
0021-8693/

Dedicated to the memory of Ákos Seress

1. Introduction

In the field of computational algebra, the problem of testing isomorphism of modules stands apart from isomorphism tests for other algebraic structures. Decades of progress has brought improvements to existing methods, and new ideas that have broadened the scope of module isomorphism tests [4,5,15,17,23,24,27]. Tools for computing with modules are now an integral part of the infrastructure of systems such as GAP [6] and MAGMA [1]. By contrast, testing isomorphism of other algebraic structures, such as finite groups, rings, and Lie and Jordan algebras, has remained extremely difficult.

In this note, we propose that the current state of play is due not to the relative ease of module isomorphism as an algorithmic problem, but rather to the fact that the problem widely referred to as "module isomorphism" is, in reality, a rather constrained version of the one your typical algebraist would likely write down. We show that, framed in a more general (and, we contend, quite natural) form, the module isomorphism problem is at least as hard as the better known *graph isomorphism problem* (Theorem 1.2). While thus suggesting that a satisfactory solution to our "general" module isomorphism problem will not soon be forthcoming, we also exhibit useful instances that do admit efficient solutions (Theorem 1.3).

It is important to stress that our intent here is not to imply that the computational algebra community has hitherto been interested in the wrong problem. On the contrary, the algorithms that underlie the accepted module isomorphism tests are among the most efficient and widely used in the entire field. It is rather that we foresee a demand for solutions to problems that are most accurately framed as module isomorphism problems of a more general flavor. In fact, as we explain briefly in the concluding section, this work grew from a particular application of such a problem to testing isomorphism of finite p-groups [3].

A motivating example. Suppose M and N are both 1-dimensional modules over a common field, say GF(9). Up to isomorphism there is only one such module, so we would expect any test of module isomorphism to confirm that $M \cong N$.

Consider the experiment in Fig. 1, conducted using the MAGMA system. The same experiment may also be carried out in GAP with the same results.

We note that in systems such as GAP and MAGMA, as well as in the literature [4,5,14,24], an A-module M is input by providing a list $(X_1, \ldots, X_\ell)$ of $(n \times n)$-matrices over a field k, where $n = \dim_k M$. These matrices correspond to the action by a fixed generating set of A on the underlying k-vector space M. Thus, the code represents the field GF(9) as a ring of (2×2)-matrices over the field $k = \mathrm{GF}(3)$, namely

$$A = B = \left\{ \begin{bmatrix} a & b \\ -b & a \end{bmatrix} : a, b \in k \right\}.$$

```
> R := MatrixAlgebra( GF(3), 2 );
> A := sub < R | [R!1, R![0,1,2,0] ] >;
> B := sub < R | [R!1, R![1,1,2,1] ] >;
> A eq B;
true
> M := RModule( A );
> N := RModule( B );
> IsIsomorphic( M , N );
false
```

Fig. 1. An illustration within MAGMA of testing module isomorphism with (possibly) confusing results.

Evidently, MAGMA confirms that A and B are the same *algebra*. The two modules M and N are determined by the action of A and B, respectively, on an identical 2-dimensional k-vector space: not only are A and B equal, so are M and N as k-spaces. So, M and N are indeed 1-dimensional (unital) vector spaces over GF(9). Why, then, does MAGMA report that M and N are nonisomorphic?

The result of the experiment will not surprise those who regularly use these tools, but some explanation is needed for those who do not. The function `IsIsomorphic` is designed to solve the following problem.

MODULEISO$_k$

Given: $(n \times n)$-matrices $X_1, \ldots, X_\ell$ and $Y_1, \ldots, Y_\ell$ over a field k.

Find: an invertible matrix Φ such that $\Phi^{-1} X_i \Phi = Y_i$ for all $1 \leqslant i \leqslant \ell$, or prove no such Φ exists.

This problem has been settled in remarkably decisive terms.

Theorem 1.1 *(Brooksbank–Luks). (See [4].) There is a deterministic algorithm to solve* MODULEISO$_k$ *that uses* $O(n^6)$ *operations in* k.

The only restriction on k in Theorem 1.1 is that one is able to carry out basic arithmetic operations; the result applies even to modules over infinite fields. Recently this result was extended to finite coefficient rings k that are not fields [27].

Now we can see why the test in Fig. 1 "failed". Though we gave the same module, we used *different* generators. Indeed,

$$A = k\left\langle \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}, \begin{bmatrix} 0 & 1 \\ 2 & 0 \end{bmatrix} \right\rangle \qquad B = k\left\langle \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}, \begin{bmatrix} 1 & 1 \\ 2 & 1 \end{bmatrix} \right\rangle.$$

The minimum polynomials of the ordered pairs of generators are $(x-1, x^2-2)$ and $(x-1, x^2-2x-1)$, respectively. So these *lists* of generators are not conjugate. In fact, we could even have used the same generating set but in a different order and the test would still have failed.

Thus, our hypothetical unwitting user receives the "wrong" answer not because of a defect in the algorithm, but because the wrong question was asked.

Module isomorphism reconsidered. Certainly every A-module can be described by one fixed generating set for A. When building A-modules, however, it may not always be possible to maintain one common generating set. Moreover, as we saw above, the algebra A is not given as part of the input of a module isomorphism test (more about this in Section 5). Thus, one cannot retroactively adjust the generators of one module to agree with those of another.

To properly solve instances of module isomorphism without carefully selected generators we focus on the following generalization.

GenModuleIso$_k$

Given: $(n \times n)$-matrices $X_1, \ldots, X_\ell$ and $Y_1, \ldots, Y_m$ over k.
Find: an invertible matrix Φ that conjugates the subalgebra $k\langle X_1, \ldots, X_\ell\rangle$ to the subalgebra $k\langle Y_1, \ldots, Y_m\rangle$.

Note that any map returned as a solution of ModuleIso$_k$ is a legitimate solution to GenModuleIso$_k$, but an algorithm solving the latter would find that the modules constructed in Fig. 1 are, indeed, isomorphic. In this sense, we have formulated a more general module isomorphism test.

Unfortunately, our first result (proved in Section 2) rather suggests that a satisfactory solution to the new problem will not easily be found.

Theorem 1.2. GenModuleIso$_k$ *is at least as hard as isomorphism testing of finite-dimensional unital associative k-algebras, and at least as hard as graph isomorphism testing.*

Thus, freed from its traditional constraints, the module isomorphism problem seems no less difficult than its analogues for other algebraic structures. The news is less bleak, however, if we restrict to special instances of the general problem. We prove the following result in Section 3.

Theorem 1.3. *For finite fields k, there are Las Vegas polynomial-time algorithms that solve the following:*

(i) given $A \leqslant \mathrm{End}_k(V)$, where V is a finite-dimensional k-space, return an epimorphism $k[x] \to A$ or prove none exists; and

(ii) instances of GenModuleIso$_k$ *for which one of the given subalgebras, say $k\langle X_1, \ldots, X_\ell\rangle$, is an epimorphic image of $k[x]$.*

As finite fields are cyclic rings, Theorem 1.3 may be used to decide isomorphism of finite vector spaces (which of course clears up the confusion in Fig. 1).

In Section 4, we report on a prototype implementation of our algorithms in Magma. While lacking decades of performance adjustments, our prototype algorithm takes only twice as long as existing (finely-tuned) Magma functions, and at the same time admits a substantially broader class of problems.

Conventions and terminology. We assume throughout that all rings are associative and unital, and that all modules are faithful. We further stipulate that all rings and modules are vector spaces over a fixed finite field k, although this can be relaxed to arbitrary commutative rings for most of Section 2. We shall assume that k-module isomorphism is decideable in polynomial time. Thus, we ultimately depend only on the assumption that we can efficiently decide isomorphism of finitely generated $(\mathbb{Z}/p)$-modules. This permits us, in particular, to focus on the problems that emerge *after* the abelian structures have been recognized.

We assume that a finite-dimensional k-vector space, V, is encoded as a row space. We also assume that an algebra $A \leqslant \mathrm{End}_k(V)$ is specified by a finite generating set, $\mathcal{S}$, of matrices: writing $A = k\langle\mathcal{S}\rangle$, we understand that A is the image of the free algebra of rank $|\mathcal{S}|$ evaluated at $\mathcal{S}$. Note, we do not assume that $\mathcal{S}$ is a basis for A as a k-space. This is the standard model for computing with algebras of endomorphisms.

We also briefly consider a more general input method for algebras, known as the *structure constant* model. Here, one specifies A via its underlying k-module A_k and, for a fixed k-basis $\mathcal{B} = \{\beta_1, \ldots, \beta_n\}$ of A_k, one provides coefficients $c_{ij\ell} \in k$ such that $\beta_i\beta_j = \sum_\ell c_{ij\ell}\beta_\ell$. This tacitly assumes, of course, that one *is able to* list a basis for A, a constraint which is not imposed on our model for A-modules. Algebras given by structure constants need not be associative.

We will be concerned with the relative difficulty of algorithmic problems. A problem is said to be in *polynomial time* if it is solvable in $O(m^c)$ steps where m is the size of a reasonable encoding of the input. In saying that a problem $\mathcal{P}$ is *polynomial-time reducible* to a problem $\mathcal{P}'$, informally we mean that if $\mathcal{P}'$ is in polynomial time, then $\mathcal{P}$ is in polynomial time; thus, $\mathcal{P}'$ is *at least as hard* as $\mathcal{P}$. More precisely, reducibility is established by providing a polynomial-time procedure that maps any instance of $\mathcal{P}$ to an instance of $\mathcal{P}'$, whereby any solution of the latter may efficiently be adapted to produce a solution of the former. The reader is referred to [7] for a comprehensive treatment of these fundamental notions.

2. Isomorphism testing for general modules is hard

We now establish connections between our general version of the module isomorphism problem and other well-studied problems. We are interested in *constructive* versions of isomorphism problems: in the event that two structures are found to be isomorphic, we also require an explicit isomorphism from one to the other.

Evidently, the decision versions of isomorphism problems that concern us here are all in NP, since any alleged isomorphism may efficiently be verified. With the exception of ModuleIso$_k$, however, none is known to be in P. The main objective of this section is to show that isomorphism problems for standard algebraic structures (notably GenModuleIso$_k$) are at least as hard as the *graph isomorphism problem*. We remark, moreover, that the current best known algorithms to solve "algebraic" isomorphism problems have running time $O(2^{cm})$ for standard inputs of size m, whereas graph isomorphism

can be solved in $O(2^{c\sqrt{m}})$ time. While this suggests that algebraic problems may be strictly harder than graph isomorphism, none are known to be NP-complete.

In order to proceed quickly to the essential problem, we begin by phrasing GenModuleIso$_k$ in the language of algebras.

EndoConj$_k$

Given: a k-module, V, and generators for k-subalgebras A and B of $\mathrm{End}_k(V)$.

Find: $\varphi \in \mathrm{Aut}_k(V)$ conjugating A to B, or prove no such φ exists.

EndoConj$_k$ seems a natural problem to study, independent of its connection to module isomorphism.

The matrix algebras to which EndoConj$_k$ applies can usually be described succinctly using small generating sets. Often, however, one wishes to compute with algebras for which no faithful linear representation is readily available. The structure constant model is the one most commonly used in such situations, and the following isomorphism problem is therefore fundamental.

AlgebraIso$_k$

Given: structure constants for associative k-algebras A and B.

Find: a k-algebra isomorphism $A \to B$, or prove $A \ncong B$.

This is generally regarded as a hard problem; we will be more explicit about this soon. Our immediate goal is to prove the following.

Proposition 2.1. AlgebraIso$_k$ *is polynomial-time reducible to* EndoConj$_k$.

Proof. Let (A, B) be an instance of AlgebraIso$_k$. We may assume that A and B are isomorphic as k-modules (the isomorphism type is determined by the size of the input k-basis). Thus, we may assume that $A = B = V$ as k-modules.

Let $\rho\colon A \to \mathrm{End}_k(V)$ and $\sigma\colon B \to \mathrm{End}_k(V)$ be the regular representations of A and B (obtained via right multiplication). As our algebras are unital these representations are faithful. Hence, $A \cong A\rho$ and $B \cong B\sigma$, so we may consider instead the problem of testing for isomorphism between $A\rho$ and $B\sigma$.

Evidently, if $A\rho$ and $B\sigma$ are conjugate in $\mathrm{End}_k(V)$, then the two algebras are isomorphic. On the other hand, if $\varphi\colon A\rho \to B\sigma$ is an isomorphism of k-algebras, then $\varphi \in \mathrm{Aut}_k(V)$. Further, if $v, a \in V$, then $v(\rho_a\varphi) = (va)\varphi = (v\varphi)(a\varphi) = v(\varphi\sigma_{a\varphi})$. Thus, $\varphi^{-1}\rho_a\varphi = \sigma_{a\varphi}$ for all $a \in V$, so $A\rho$ is conjugate to $B\sigma$. Hence, EndoConj$_k$ applied to $(A\rho, B\sigma)$ solves AlgebraIso$_k$ for (A, B). □

Proposition 2.1 tells us that EndoConj$_k$ is at least as hard as AlgebraIso$_k$. The latter has connections to other well-studied problems. Recently, for example, Saxena established a link to the well-studied *graph isomorphism problem*:

GraphIso

Given: a finite set V, and subsets $\mathcal{E}$ and $\mathcal{F}$ of $V \times V$;
Find: a permutation of V sending $\mathcal{E}$ onto $\mathcal{F}$, or prove none exists.

Despite important breakthroughs in special cases (see, for example, [22]) the decision version of GraphIso remains in the tantalizing complexity region between P and NP-complete; see [20] for a survey. In [25, Theorem 2.2], Saxena proved the following result.

Theorem 2.2 *(Saxena).* GraphIso *is polynomial-time reducible to* AlgebraIso$_k$.

Taken together with Proposition 2.1, this result proves Theorem 1.2. Furthermore, these results immediately establish a connection with one version of the isomorphism problem for finite groups. Recall, Cayley's theorem states that every finite group G can be faithfully represented on the set G by the regular representation $h \mapsto (g \mapsto gh)$. The following version of the group isomorphism problem continues to generate interest in certain research circles.

CayleyGroupIso

Given: regular (Cayley) representations of finite groups G and H.
Find: an isomorphism $G \to H$, or prove $G \ncong H$.

Corollary 2.3. CayleyGroupIso$_k$ *is polynomial-time reducible to* GenModuleIso$_k$.

Proof. Miller showed CayleyGroupIso is polynomial-time reducible to GraphIso; see [20, p. 18]. The result now follows from Theorem 2.2 and Proposition 2.1. □

We have argued here that there are several (at least two) versions of the module isomorphism problem, and that the tractability of the problem differs strikingly with the choice of computational model. It would be remiss of us not to mention that a similar situation exists among isomorphism problems for finite groups.

It is never the case, in practical settings, that a group is specified in the redundant manner of CayleyGroupIso, namely by providing its full multiplication table. On the contrary, finite groups are usually input far more concisely via small sets of permutations or matrices, or perhaps abstractly using one of various kinds of special presentations.

It has been shown, for such concise models, that the group isomorphism problem is at least as hard as GraphIso, and is likely substantially harder. Such observations arise from the work of Heineken and Liebeck [13], and Soules [26], with the topic being addressed in complexity terms by Garzon and Zalcstein [8, p. 247].

3. Module isomorphism over cyclic algebras

Since a polynomial-time solution to GenModuleIso$_k$ is unlikely for arbitrary input, we consider the more tractable problem of isomorphism testing for $k[x]$-modules, k a finite

field, where $k[x]$ denotes the ring of polynomials. We prove this case can be handled efficiently, hence proving Theorem 1.3.

A k-algebra is *cyclic* if it is a homomorphic image of $k[x]$. The multiplicative aspect makes cyclic algebras rather distinct from their group counterparts. For instance, $\mathbb{Z}/2 \oplus \mathbb{Z}/2$ is a cyclic $(\mathbb{Z}/2)$-algebra because it is isomorphic to $(\mathbb{Z}/2)[x]/(x^2 - x)$, whereas its underlying abelian group is clearly not cyclic. More generally, if $a_1, \dots, a_n$ are distinct elements of k, then $k^n \cong k[x]/(\prod_{i=1}^{n}(x - a_i))$; in particular, for every n there is a finite field k such that k^n is cyclic.

Cyclic rings often have nontrivial radicals, and can have quotients onto multiple extension fields. Thus, while they are very far from arbitrary rings, cyclic rings exhibit many of the ring-theoretic properties that seem to influence the difficulty of fundamental algorithmic problems for rings.

As one might expect, our algorithms for cyclic algebras make essential use of canonical forms. In particular, we will use the following version of the *Rational Canonical Form (RCF)* of a linear transformation.

For $f(x) = x^n - a_{n-1}x^{n-1} - \cdots - a_0 \in k[x]$, and positive integer ℓ, the *companion matrix*, C_f, and *generalized Jordan block*, $J_\ell(f)$, are as follows:

$$C_f = \begin{bmatrix} 0 & 1 & & \\ & \ddots & \ddots & \\ & & 0 & 1 \\ a_0 & a_1 & \dots & a_{n-1} \end{bmatrix} \in \mathbb{M}_n(k), \qquad J_\ell(f) = \begin{bmatrix} C_f & I_n & & \\ & \ddots & \ddots & \\ & & \ddots & I_n \\ & & & C_f \end{bmatrix} \in \mathbb{M}_{n\ell}(k).$$

If $\lambda = [\ell_1, \dots, \ell_m]$ is a weakly decreasing list of positive integers (a partition), then $J_\lambda(f) = \operatorname{diag}(J_{\ell_1}(f), \dots, J_{\ell_m}(f))$. If X is any matrix over k, with minimal polynomial factorized over k as $f_1(x)^{e_1} \cdots f_a(x)^{e_a}$, then X is conjugate to $\operatorname{diag}(J_{\lambda_1}(f_1), \dots, J_{\lambda_a}(f_a))$ for some partitions $\lambda_1, \dots, \lambda_a$ [18, Sections 3.10–3.11]. This is what we shall refer to as the RCF of X.

Crucially, there is a Las Vegas polynomial-time algorithm that, given a matrix X over a finite field, returns an invertible matrix U such that $U^{-1}XU$ is in RCF [10, Section 5.3]; see also [11, Theorems 5.2–5.3].

3.1. Recognizing cyclic algebras

Our task, then, is to provide an efficient algorithm to solve EndoConj$_k$ whenever one of the input algebras is cyclic. Before we can tackle that problem, however, we must be sure that we can recognize when a given algebra *is* cyclic, since we do not presume that a cyclic generator is provided. We require an efficient solution to the following problem.

IsCyclic$_k$

Given: $A \leqslant \operatorname{End}_k(V)$, where V is a finite-dimensional k-module.
Find: $s \in A$ such that $A = k\langle s \rangle$, or prove that A is not cyclic.

There are effective solutions to IsCyclic$_k$ when A is known to be a field: in deterministic polynomial time one can find a *normal basis* basis for A over k, for example, and use it to find a cyclic generator [21, Theorem 3.1].

Our extension to the general problem uses a Wedderburn–Mal'cev decomposition (henceforth a *WM-decomposition* for short) of an algebra A, namely a decomposition $A = J \oplus B$, where J is the Jacobson radical of A, and B is a semisimple subring of A. We note that all such B are conjugate in A [19, Section 6.11].

Lemma 3.1. *Let A be a commutative k-algebra, and $A = J \oplus B$ a WM-decomposition of A. Then A is cyclic if, and only if, B is cyclic and J/J^2 is a cyclic B-module.*

Proof. Suppose that $A = k\langle s\rangle$ is cyclic. Let $f_1(x)^{e_1} \dots f_n(x)^{e_n} \in k[x]$ be the factorization of the minimal polynomial of s into irreducibles. A WM-decomposition, $A = J \oplus B$, with B cyclic is obtained from the RCF of s, wherein $B = B_1 \oplus \dots \oplus B_n$ and $B_i \cong k[x]/(f_i(x))$ is an extension of k. Furthermore, $J = J_1 \oplus \dots \oplus J_n$, with J_i/J_i^2 a 1-dimensional B_i-module. Hence, J/J^2 is a cyclic B-module.

Conversely, let A be a commutative k-algebra, and $A = J \oplus B$ a WM-decomposition of A such that $B = k\langle t\rangle$ is cyclic, and J/J^2 is a cyclic B-module with generator $u + J^2$, for some $u \in J$. Put $s := t + u$. We claim that $A = k\langle s\rangle$.

Let $B = B_1 \oplus \dots \oplus B_n$ be the decomposition of B into simple rings. As A is commutative, each B_i is a field extension of k. Let K be an extension of k, with $|K| = p^m$ exceeding the nilpotence degree of J, that contains all B_i as subfields. Then, $s^{p^m} = (t+u)^{p^m} = t^{p^m} + u^{p^m} = t \in k\langle s\rangle$, so $B \subseteq k\langle s\rangle$. Also, $u = s - t \in k\langle s\rangle$, so $k\langle s\rangle$ contains a generator for J/J^2. Thus, $J = Bu + J^2$ and, by Nakayama's lemma, $J = Bu \subseteq k\langle s\rangle$. Hence, $A = J \oplus B \subseteq k\langle s\rangle$, and the claim follows. □

We also require an effective way to build irreducible polynomials in $k[x]$. More precisely, we need the following result.

Lemma 3.2. *There is a polynomial-time Las Vegas algorithm that, given $\mathcal{I} \subset k[x]$, a nonempty set of monic irreducible polynomials of degree n, returns an irreducible polynomial of degree n not in $\mathcal{I}$, or reports that no such exists.*

Proof. Our approach varies according to the nature of $\mathcal{I}$. (Note that the input length is $n|\mathcal{I}| \log |k|$.)

Special case: $|\mathcal{I}| \geqslant |k|^{n/2}$ *or* $|k|^n \leqslant 16$. Form the list $\mathcal{P}_n$ of all monic polynomials in $k[x]$ of degree n. For each $f(x) \in \mathcal{P}_n \setminus \mathcal{I}$, test whether $f(x)$ is irreducible: if so, stop, and return $f(x)$. If no such $f(x)$ exists, report that $\mathcal{I}$ already contains all of the irreducible polynomials of degree n.

General case: $0 < |\mathcal{I}| < |k|^{n/2}$. Here, a classical result of Gauss ensures the existence of an irreducible polynomial of degree n not in $\mathcal{I}$, and the following is a Las Vegas procedure to find one.

Fix $f(x) \in \mathcal{I}$, and construct the field $K = k[x]/(f(x))$. For each $g(x) \in \mathcal{I}$, factor $g(x)$ completely in K (using, for example, the methods of [28, Chapter 14]). In this way, construct the set

$$R := \{\alpha \in K : \exists g(x) \in \mathcal{I} \text{ such that } \alpha \text{ is a root of } g(x)\} \subset K.$$

Repeat the following steps until an appropriate $h(x)$ is found: choose $\omega \in K \setminus R$ at random; construct the minimal polynomial, $h(x) \in k[x]$, of ω; if $\deg h = n$, stop and return $h(x)$; else continue.

We now analyze the two cases in turn. For convenience, let $\ell = |\mathcal{I}|$ and $q = |k|$.

That the special case works as advertized is obvious. We merely remark that $|\mathcal{P}_n| = O(\ell^2)$: as irreducibility testing is polynomial time, our approach yields a polynomial-time algorithm.

Next, consider the general case. As there are ℓ distinct irreducible polynomials in $|\mathcal{I}|$, there are $n\ell < q^n$, elements in R. Furthermore, there are at most $\log n$ divisors of n, so there are at most this many proper subfields L of K, each having size at most $q^{n/2}$. Hence,

$$\frac{|(K \setminus R) \setminus (\bigcup_{|K/L|>1} L)|}{|K \setminus R|} \geqslant \frac{q^n - n\ell - q^{n/2} \log n}{q^n - n\ell} = 1 - \frac{q^{n/2} \log n}{q^n - n\ell}.$$

As $1 \leqslant \ell < q^{n/2}$, we have $1 - \frac{q^{n/2} \log n}{q^n - n\ell} \geqslant 1 - \frac{\log n}{q^{n/2} - n}$, which exceeds 0.1 whenever $q^n > 16$. It follows that each independent choice of $\omega \in K \setminus R$ lies outside a proper subfield with probability at least 0.1. The minimal polynomial of any such ω is irreducible of degree n lying outside $\mathcal{I}$.

Clearly then, the algorithm for the general case exits only when it has identified an irreducible polynomial, $h(x)$, of degree n, lying outside $\mathcal{I}$, and such $h(x)$ is found, with positive probability, after a polynomial number of steps. □

Remark 3.3. There is a convenient way to produce new irreducible polynomials from old: given $f(x) \in \mathcal{I}$, choose $c \in k$, and see if the irreducible polynomial $f(x - c)$ is in $\mathcal{I}$. This will often be the most effective approach in practical settings.

We can now present our recognition algorithm for cyclic rings.

Proposition 3.4. *There is a polynomial-time Las Vegas algorithm to solve* IsCyclic$_k$.

Proof. If $A = k\langle \mathcal{S} \rangle$ is not commutative, then it is not cyclic. Commutativity of A is determined by commutativity among the elements of $\mathcal{S}$. Thus, we may now assume that $A = k\langle \mathcal{S} \rangle$ is commutative.

Use [9] to compute a WM-decomposition, $A = J \oplus B$, and to find extension fields $B_1, \ldots, B_n$ of k such that $B = B_1 \oplus \ldots \oplus B_n$. Note, if B is an epimorphic image of

$k[x]$ with kernel I, then $I = (f_1(x)\cdots f_n(x))$ with each $f_i(x)$ irreducible and $\deg f_i = [B_i : k]$.

We first proceed iteratively through the fields B_i to construct a cyclic generator for B, if such exists. Initialize $i := 1$.

Using [21, Theorem 3.1], construct $t_1 \in B_1$ with $B_1 = k\langle t_1\rangle$, and compute the minimum polynomial, $m_1(x)$, of t_1 in its restriction to the support of B_1.

Suppose $1 \leqslant i \leqslant n-1$, and that we have built t_i such that $B_1 \oplus \ldots \oplus B_i = k\langle t_i\rangle$. Suppose, further, that we have computed the minimum polynomial, $m_i(x)$, of the restriction of t_i to the support of $B_1 \oplus \cdots \oplus B_i$. (Hence, the assignment $x \mapsto t_i$ yields an isomorphism from $k[x]/(m_i(x))$ to $B_1 \oplus \cdots \oplus B_i$.)

Find $b_{i+1} \in B_{i+1}$ such that $B_{i+1} = k\langle b_{i+1}\rangle$, and compute the minimum polynomial, $f_{i+1}(x)$, for the restriction of b_{i+1} to the support of B_{i+1}.

If $f_{i+1}(x) \nmid m_i(x)$, put $t_{i+1} := t_i + b_{i+1}$, and $m_{i+1}(x) := m_i(x)f_{i+1}(x)$. Now, increase i by 1 and iterate.

Else, use Lemma 3.2 to select $h(x) \in k[x]$ irreducible with $h(x) \nmid m_i(x)$ and $\deg h = \deg f_{i+1}$. If no such $h(x)$ exists, then exit, reporting that B (and hence A) is not cyclic. Otherwise, factor $h(x)$ in the field B_{i+1} and locate a root $w \in B_{i+1}$. Put $b_{i+1} := w$, $f_{i+1}(x) := h(x)$, $t_{i+1} := t_i + b_{i+1}$, and $m_{i+1}(x) := m_i(x)f_{i+1}(x)$. Again, increase i by 1 and iterate.

If we reach $i = n$, then we have constructed an element $t_n \in B$, which we prove below is a generator for B.

Finally, construct J/J^2 as a B-module, and use [5, Theorem 1] to construct a cyclic vector $u \in J \setminus J^2$ if such exists. If there is no such vector, report that A is not cyclic. Otherwise, return $s := t_n + u$.

The correctness of the procedure is a scholium to Lemma 3.1 provided we can show that $B = k\langle t_n\rangle$. To that end, we observe that the cyclicity of $B_1 \oplus \ldots \oplus B_i$ remains invariant under the iterative step. For, at the end of a fixed iteration i, we always have $f_{i+1}(x) \nmid m_i(x)$ (unless we exited the loop to report failure). Hence, $(m_i(x))+(f_{i+1}(x)) = k[x]$ and, by the Chinese Remainder Theorem,

$$k[x]/\big(m_i(x)f_{i+1}(x)\big) = k[x]/\big(m_i(x)\big) \oplus k[x]/\big(f_{i+1}(x)\big) \cong (B_1 \oplus \cdots \oplus B_i) \oplus B_{i+1},$$

with the isomorphism induced by the assignment $x \mapsto t_{i+1} = t_i + b_{i+1}$. □

3.2. Conjugating cyclic algebras

Now that we can recognize cyclic algebras, we return to the conjugacy problem. The obvious first case to consider is when the given cyclic algebras are fields. A deterministic, polynomial-time algorithm to solve this problem for fields described by structure constants was first given by Lenstra [21, Theorem 1.2]. For completeness, we include the following elegant solution communicated to us by W.M. Kantor.

Lemma 3.5 *(Kantor). There is a polynomial-time algorithm that, given $s, t \in \mathrm{End}_k(V)$ acting irreducibly on V, returns $c \in \mathrm{Aut}_k(V)$ with $s^c \in k\langle t\rangle$.*

Proof. Construct the minimal polynomial, $m(x)$, of s over k, and factor it in the polynomial ring $k\langle t\rangle[x]$. If $(x - z) \in k\langle t\rangle[x]$ is any one of the linear factors, then s and z are conjugate in $\mathrm{Aut}_k(V)$. One finds $c \in \mathrm{Aut}_k(V)$ conjugating s to z, and hence $k\langle s\rangle$ to $k\langle t\rangle$, by first conjugating s and z to their common RCF. □

Before presenting our extension to arbitrary cyclic algebras, we stress an important point. Suppose $\{s\}$ and $\{t\}$ are given as an instance of ModuleIso$_k$, where $s, t \in \mathrm{End}_k(V)$ are arbitrary. Then an output of "true" is expected if, and only if, s and t are conjugate *as elements* in $\mathrm{Aut}_k(V)$. The latter is determined easily from the RCFs of s and t. Deciding whether $k\langle s\rangle$ and $k\langle t\rangle$ are conjugate *algebras* is, however, a substantively different problem.

Proposition 3.6. *There is a polynomial-time algorithm that, given $s, t \in \mathrm{End}_k(V)$, returns $c \in \mathrm{Aut}_k(V)$ such that $k\langle s^c\rangle = k\langle t\rangle$, if such c exists.*

Proof. The proof uses the RCFs of s and t. Intuitively, it is clear that the canonical forms of s and t are required to be, in a certain sense, "compatible"; our proof shows that this compatibility is also a sufficient condition for conjugacy.

Suppose, first, that s is *primary*, namely s has minimal polynomial $f(x)^e \in k[x]$, where $f(x)$ is irreducible. Let $n = \deg f$, and $C_f \in \mathbb{M}_n(k)$ denote the companion matrix of $f(x)$. Then the RCF of s is $J_\lambda(f)$ for some partition $\lambda = [\ell_1, \ldots, \ell_a]$. Evidently, $d/n = \sum_{i=1}^{a} \ell_i$, where $d = \dim_k V$. For use later on, we associate to s the integer sequence

$$\mu(s) = \mu(s, f) = (n, \ell_1, \ldots, \ell_a),$$

which we regard as the *signature* of s. For $1 \leqslant i \leqslant a$, the subalgebra $k\langle J_{\ell_i}\rangle$ has a WM-decomposition $U_i \oplus S_i$, where $S_i = k\langle \mathrm{diag}(C_f, \ldots, C_f)\rangle \leqslant \mathbb{M}_{n\ell_i}$ is a field extension of k of degree n, and U_i (the Jacobson radical of $k\langle J_{\ell_i}\rangle$) is nilpotent of degree $\ell_i - 1$. Thus, A has a WM-decomposition $A = U \oplus S$, where $U = U_1 \oplus \ldots \oplus U_a$, and $S = S_1 \oplus \ldots \oplus S_a$.

Next consider t. Clearly, $k\langle s\rangle$ is conjugate to $k\langle t\rangle$ only if t is also primary, having minimal polynomial say $g(x)^e$, $g(x)$ irreducible. If $c \in \mathrm{GL}(d, k)$ conjugates $k\langle s\rangle$ to $k\langle t\rangle$, then c conjugates each WM-decomposition of $k\langle s\rangle$ to one for $k\langle t\rangle$. As WM-decompositions of $k\langle t\rangle$ are conjugate in $k\langle t\rangle$, we have $\mu(t, g) = \mu(s, f)$.

We now show that if $\mu(s, f) = \mu(t, g)$ then we can find c with $k\langle s\rangle^c = k\langle t\rangle$.

Assume s and t are written in RCF. Using Lemma 3.5, find $Y \in \mathrm{GL}(n, k)$ with $D := Y^{-1} C_f Y \in k\langle C_g\rangle$. For $1 \leqslant i \leqslant a$, put $y_i := \mathrm{diag}(Y, Y, \ldots, Y) \in \mathrm{GL}(\ell_i n, k)$, and $c := \mathrm{diag}(y_1, y_2, \ldots, y_a) \in \mathrm{GL}(d, k)$. Then,

$$c^{-1} s c = \mathrm{diag}\big(y_1^{-1} J_{\ell_1} y, y_2^{-1} J_{\ell_2} y_2, \ldots, y_a^{-1} J_{\ell_a} y_a\big),$$

and, for $1 \leqslant i \leqslant a$,

$$y_i^{-1} J_{\ell_i}(f) y_i = \begin{bmatrix} D & I_n & & \\ & \ddots & \ddots & \\ & & \ddots & I_n \\ & & & D \end{bmatrix}.$$

Write $J_{\ell_i}(f) = u_i + n_i$ and $J_{\ell_i}(g) = v_i + n_i$, where n_i is the (common) nilpotent part of $J_\ell(f)$ and $J_\ell(g)$, $u_i = \mathrm{diag}(C_f, \ldots, C_f)$, and $v_i = \mathrm{diag}(C_g, \ldots, C_g)$. Since $D \in k\langle C_g \rangle$, we have $y_i^{-1} u_i y_i \in k\langle v_i \rangle \subseteq k\langle v_i + n_i \rangle$ (for the last inclusion see Lemma 3.1). Hence, $y_i^{-1} J_{\ell_i}(f) y_i = y_i^{-1} u_i y_i + n_i \in k\langle J_{\ell_i}(g) \rangle$. Thus, $c^{-1} s c \in k\langle t \rangle$, as required.

We turn now to the general case. Let $s, t \in \mathrm{End}_k(V)$ be arbitrary, and assume that s and t are already in RCF. We test whether or not there exists $c \in \mathrm{GL}(d, k)$ with $s^c \in k\langle t \rangle$ as follows.

First, compute the minimal polynomial of s, and factorize as $f_1(x)^{d_1} \ldots f_{m_s}(x)^{d_{m_s}}$ with f_i irreducible. Likewise write the minimal polynomial of t as $g_1(x)^{e_1} \ldots g_{m_t}(x)^{e_{m_t}}$ with g_i irreducible. For $k\langle s \rangle$ and $k\langle t \rangle$ to be conjugate, the number of primary components in their respective $k[x]$-modules must agree, and so m_s must equal m_t. If this is not the case we return "false"; else put $m := m_s = m_t$.

Next, compute the list, $\mu(s, f_1), \mu(s, f_2), \ldots, \mu(s, f_m)$, of primary signatures of s (easily read off from the RCF of s). Let $\tau(s)$ denote the permutation that sorts the list into lexicographic order. Similarly, compute $\mu(t, g_1), \mu(t, g_2), \ldots, \mu(t, g_m)$, and corresponding permutation $\tau(t)$.

Put $\pi := \tau(s)\tau(t)^{-1}$. If $k\langle s \rangle$ is conjugate to $k\langle t \rangle$, then $\mu(s, f_i) = \mu(t, g_{i\pi})$ for all $1 \leqslant i \leqslant m$. Hence, if this is not the case, we return "false". Otherwise, use a block permutation matrix for π to rearrange the blocks in the RCF of t. Thus, we may assume that π is the identity.

Finally, for each $1 \leqslant i \leqslant m$, use the primary case to find an invertible matrix y_i that conjugates the ith primary component of s into the algebra generated by the ith primary component of t. It is now immediate that $c := \mathrm{diag}(y_1, \ldots, y_m)$ conjugates $k\langle s \rangle$ to $k\langle t \rangle$, as required. □

Proof of Theorem 1.3. Part (i) of Theorem 1.3 is just Proposition 3.4.

For part (ii), suppose $X_1, \ldots, X_\ell, Y_1, \ldots, Y_m$ is an instance of GenModuleIso$_k$. As $k\langle X_1, \ldots, X_\ell \rangle$ is presumed cyclic, using Proposition 3.4 we can now find a cyclic generator, say s, for this subalgebra. (Note, our algorithm will detect if it's not cyclic.) Next, use Proposition 3.4 again to test whether $k\langle Y_1, \ldots, Y_m \rangle$ is cyclic. If it is not, then the two algebras are clearly not conjugate. If it is cyclic, then the algorithm also returns a cyclic generator, say t, for $k\langle Y_1, \ldots, Y_m \rangle$. Now use Proposition 3.6 to test whether $k\langle s \rangle$ and $k\langle t \rangle$ are conjugate. □

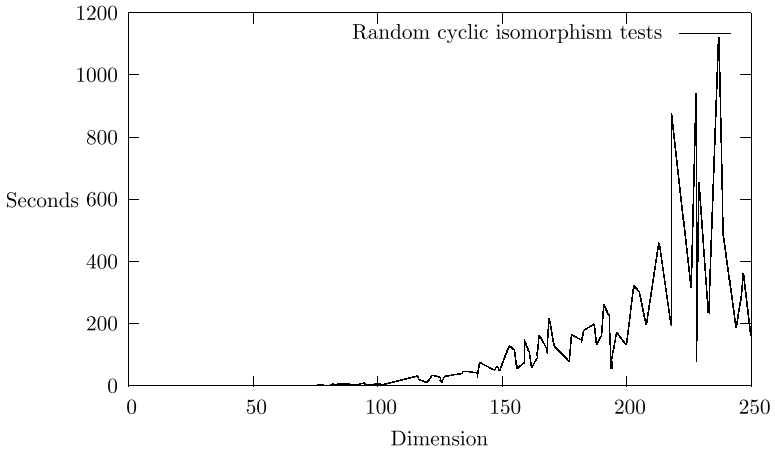

Fig. 2. 100 randomized trials of our general module isomorphism test of modules over cyclic algebras.

4. Implementation and performance

Prototypes of the algorithms presented in Section 3 have been implemented by the authors in the computer algebra system Magma. The code currently uses generic functions to handle certain computations with matrix algebras, and we expect that implementations of algorithms such as [9,16] would improve its performance significantly. Nevertheless, even on test examples that are built to allow comparisons with standard module isomorphism machinery in Magma, our functions fare reasonably well. (Remember that our algorithms are designed to solve the more general problem GenModuleIso$_k$.) We now describe some performance tests that we ran, and remark on opportunities for improvement. All tests were conducted on a computer with an Intel i5-2400 processor (four cores, 3.1 GHz) running Magma Version 2.19–10.

Our first test, summarized in Fig. 2, examines the performance of our module isomorphism test for $k[x]$-modules over $k = \mathrm{GF}(2)$. For a random d in the range $\{100, \ldots, 250\}$, we selected a random $c \in \mathbb{M}_d(k)$. We then formed a set $\mathcal{S}$, of size roughly $2\lfloor \log d \rfloor$, consisting of elements selected at random from $k\langle c\rangle$ until $A = k\langle \mathcal{S}\rangle = k\langle c\rangle$. By choosing a larger generating set for A, we are forcing the computer to forget that it is cyclic. We formed another such set, $\mathcal{T}_0$, of roughly the same size as $\mathcal{S}$, chose a random invertible matrix g, and then put $\mathcal{T} = \mathcal{T}_0^g$ and $B = k\langle \mathcal{T}\rangle$. Finally, we used our implementation to confirm, first, that A and B are cyclic and, second, that they are conjugate.

The timing in the first test was dominated by the cost of verifying, constructively, that our algebras are both cyclic. The large variance that one sees in the runtimes arises from the fact that the code currently uses generic Magma functions for matrix algebras to compute a WM-decomposition $A = J \oplus S$. The efficiency of this step can vary quite dramatically depending on the dimension of the Jacobson radical, J. As alluded to earlier, however, an implementation of the methods of [9,16] would likely improve that aspect of the test significantly, and a more thoughtful exploitation of the fact that our algebras

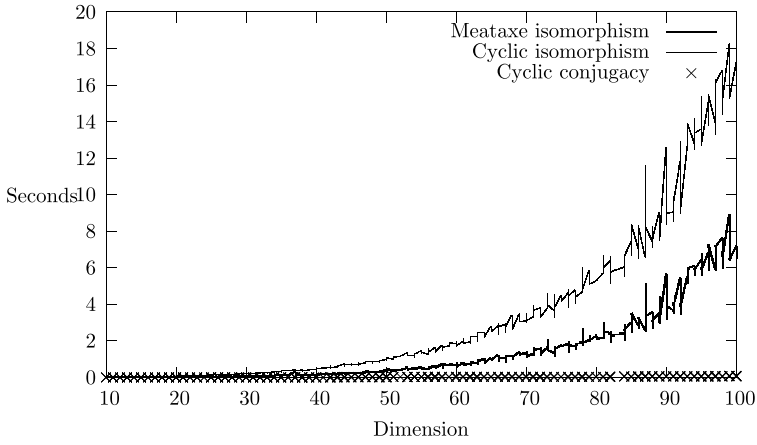

Fig. 3. Comparison, for 500 isomorphism tests of modules over cyclic rings, of our implementation with Meat-Axe methods. Note, the Meat-Axe is designed for a special computational model, so this test had to use quite restricted inputs.

may be assumed commutative would certainly improve the overall performance of our cyclicity test.

Our second test compared the performance of our implementation against standard Magma machinery for carrying out similar tasks. The comparison is somewhat artificial since we had to force the input to be compatible with both settings, but nevertheless illustrates the practical potential of our implementation.

For this test we fixed $k = \mathrm{GF}(9)$, and let d vary in the range $\{10, \ldots, 100\}$. Again, we started with a random $c \in \mathbb{M}_d(k)$. We then selected a random sequence $\mathcal{S} = [s_1, \ldots, s_n]$ from $k\langle c\rangle$ (we fixed $n = 2\lfloor \log d \rfloor$), and computed $\mathcal{T} = [t_1, \ldots, t_n]$, where $t_i = s_i^g$ for a random $g \in \mathrm{GL}(d, k)$. We then put $A = k\langle \mathcal{S}\rangle$ and $B = k\langle \mathcal{T}\rangle$, again both cyclic algebras that have forgotten they are cyclic. Hence, this represents a valid input to both $\textsc{ModuleIso}_k$ and $\textsc{GenModuleIso}_k$.

We next verified conjugacy of A and B in two ways.

First, we used the standard Magma function `IsIsomorphic` to construct an isomorphism from the module defined by A to the module defined by B. Thus, as in Fig. 1 in Section 1, we are asking Magma to solve the problem $\textsc{ModuleIso}_k$; clearly, the isomorphism it returns conjugates A to B. It does this, in practice, using a suite of functions known collectively as the *Meat-Axe*; these are founded on implementations of [15].

Second, we used our own functions to verify conjugacy. Again, the code first verifies that the given cyclic algebras are indeed cyclic.

The graphs in Fig. 3 show runtimes for the Meat-Axe functions, and for the various components of our own functions. We are, on average, only twice as slow as the Meat-Axe, which seems as good as one might reasonably expect, given that we test cyclicity twice (once for A and once for B). Note that the time needed to conjugate the cyclic algebras (once they are confirmed cyclic) is negligible. Thus, once we have found that our algebras are cyclic, our conjugacy algorithm is substantially faster than Meat-Axe methods for conjugacy.

Our final test focused just on conjugacy testing, and was designed to examine the effect of increasing block size in a semisimple cyclic algebra while decreasing the number of blocks. The experiment indicated that conjugacy of fields dominates the overall performance of the algorithm: the larger the irreducible blocks appearing in the RCFs of the cyclic generators, the worse the performance.

5. Concluding remarks

We close with some remarks on related work, and briefly mention a particular application of our results that first prompted us to consider more general forms of module isomorphism. We also comment on unfaithful modules.

5.1. Comparing with Lie module isomorphism

Recently, J. Grochow studied the problem of conjugacy of Lie subalgebras of $\mathfrak{gl}_n(k)$. He proves the following result in [12, Corollary II.3].

Theorem 5.1 *(Grochow). In an arithmetic model with an oracle for factoring polynomials, deciding conjugacy of diagonalizable Lie subalgebras of* $\mathfrak{gl}_n(k)$ *(given by generators) is as hard as graph isomorphism.*

This shows that Lie algebra isomorphism and Lie module isomorphism will likely be hard. At first glance, in view of the graph isomorphism obstruction, Grochow's result appears to accord with our own observations on testing conjugacy of associative subalgebras. A second read may, however, give some cause for alarm, since we have proved a seemingly conflicting result:

Theorem 5.2. *If* $|k| > d$*, then deciding conjugacy of diagonalizable associative algebras of* $\mathbb{M}_d(k)$ *is in polynomial time.*

Proof. If $|k| > d$ then all diagonalizable subalgebras of $\mathbb{M}_d(k)$ are cyclic. The result now follows from Proposition 3.6. □

Despite their cosmetic similarity, however, the isomorphism problems for diagonalizable Lie, and diagonalizable associative algebras are largely unrelated. Abelian Lie k-algebras are nothing more than k-vector spaces (the product is trivial), so any subspace of k^n is also a Lie subalgebra. This leads to a reduction, first to the problem of "code equivalence", and ultimately to graph isomorphism. On the other hand the unital associative diagonalizable subalgebras have nontrivial products and, up to conjugacy, their number is bounded by the number of partitions of n which is at most 2^n and thus substantially more constrained than the $2^{\Theta(n^2)}$ graphs on $\Theta(n)$ vertices or the $|k|^{\Theta(n^2)}$ subspaces of k^n.

We remark that our proof of Theorem 1.2 shows that graph isomorphism is an obstruction to isomorphism and conjugacy testing of *nilpotent* Lie algebras, just as it is for associative algebras.

5.2. Testing isomorphism of p-groups

The work in Section 3 was motivated by a particular problem that arose in connection to testing isomorphism of p-groups. In [2], we developed a strategy for constructing the automorphism group of a p-group, P, by instead using automorphisms of an associated $(\mathbb{Z}/p)$-algebra, $A(P)$. In order to adapt that strategy to a test for isomorphism between groups P and Q, we must first conjugate $A(P)$ to $A(Q)$, if this is possible [3]. This accounts for our interest in $\textsc{EndoConj}_k$. In key instances, moreover, the algebras $A(P)$ and $A(Q)$ that arise are *centralizers* of cyclic algebras. As those centralizers are conjugate if, and only if, their centers are conjugate, Theorem 1.3 now solves our problem. Hence, the results of Section 3 provide a foundation for a new approach to isomorphism testing in important classes of p-groups.

5.3. Unfaithful modules

The algorithms presented in the foregoing sections presume that the given A-modules are faithful. We caution that, when applied to unfaithful modules, our algorithms are likely to produce incorrect answers.

For example, let $A = k \oplus k$, and define two 1-dimensional representations $\rho_1, \rho_2 \colon A \to \mathrm{End}_k(k) = k$, where $\ker \rho_1 = k \oplus 0$, and $\ker \rho_2 = 0 \oplus k$. As the annihilators are unequal, the modules are certainly not isomorphic; yet, $A\rho_1 = A\rho_2$, so $\textsc{GenModuleIso}_k$ would find that they are.

If we adhere to the specifications of the standard algorithms to solve $\textsc{ModuleIso}_k$, the unfaithful modules described above may be distinguished without difficulty. For, suppose that we fix generators $s_1 = (1, 0)$ and $s_2 = (0, 1)$ for A. Then the first module is input by the list $[0], [1] \in \mathbb{M}_1(k)$, while the second is specified by $[1], [0]$. Clearly, $\textsc{ModuleIso}_k$ would correctly determine that they are nonisomorphic.

For a completely general module isomorphism test, we could insist that A, and the maps $\rho_i \colon A \to \mathrm{End}_k(V)$, be passed as input to the algorithm. The difficulty is that we often wish to compute with A-modules for which it is not easy to specify A for input, or for which the standard ring operations are prohibitively expensive. For instance, one can compute effectively with the 248-dimensional module of the group algebra kG for $G = E_8(q)$, but one would not wish to work within kG as it has dimension roughly q^{248}. For this reason one can easily see why it is desirable to ask just for the image of a representation. On the other hand that requires the unreasonable assumption that every user will agree on these same generators.

It is clear, then, that for unfaithful modules $\textsc{GenModuleIso}_k$ actually tests for a form of module equivalence that is weaker than isomorphism. We call this *semilinear*

isomorphism: given representations $\rho: A \to \mathrm{End}_k(M)$ and $\tau: B \to \mathrm{End}_k(N)$, there exists $\varphi: M \to N$ such that, for all $m \in M$, and for all $a \in A$, $\varphi(ma) = \varphi(m)a^\sigma$, where $\sigma: A\rho \to B\tau$ is an induced algebra isomorphism.

Acknowledgments

Both authors are indebted to Ákos Seress for his kindness, his counsel, and especially his friendship over many years.

We also kindly thank the referee for a number of insightful comments, references, and corrections that have contributed substantially to the quality of the paper.

This work was partially supported by a grant from the Simons Foundation (#281435 to Peter Brooksbank).

The project was also partially supported by the National Security Agency under Grant Number H98230-11-1-0146. The United States Government is authorized to reproduce and distribute reprints not-withstanding any copyright notation herein.

References

[1] Wieb Bosma, John Cannon, Catherine Playoust, The Magma algebra system. I. The user language, in: Computational Algebra and Number Theory, London, 1993, J. Symbolic Comput. 24 (3–4) (1997) 235–265, MR1484478.
[2] Peter A. Brooksbank, James B. Wilson, Groups acting on tensor products, J. Pure Appl. Algebra 218 (2014) 405–416.
[3] Peter A. Brooksbank, James B. Wilson, Reducing the futility of group isomorphism testing, preprint.
[4] Peter A. Brooksbank, Eugene M. Luks, Testing isomorphism of modules, J. Algebra 320 (11) (2008) 4020–4029, MR2464805 (2009h:16001).
[5] Alexander Chistov, Gábor Ivanyos, Marek Karpinski, Polynomial time algorithms for modules over finite dimensional algebras, in: Proceedings of the 1997 International Symposium on Symbolic and Algebraic Computation, Kihei, HI, ACM, New York, 1997, pp. 68–74.
[6] The GAP Group, GAP – Groups, Algorithms, and Programming, Version 4.7.4, http://www.gap-system.org, 2014.
[7] Michael R. Garey, David S. Johnson, Computers and Intractability: A Guide to the Theory of NP-Completeness, W.H. Freeman, 1979.
[8] Max Garzon, Yechezkel Zalcstein, On isomorphism testing of a class of 2-nilpotent groups, J. Comput. System Sci. 42 (2) (1991) 237–248, MR1103249 (92k:20063).
[9] Patrizia Gianni, Victor Miller, Barry Trager, Decomposition of algebras, in: Symbolic and Algebraic Computation, Rome, 1988, in: Lecture Notes in Comput. Sci., vol. 358, Springer, Berlin, 1989, pp. 300–308, MR1034741 (91e:12009).
[10] M. Giesbrecht, Nearly optimal algorithms for canonical matrix forms, PhD thesis and U. of Toronto Technical Report 268/93, 1993.
[11] Mark Giesbrecht, Nearly optimal algorithms for canonical matrix forms, SIAM J. Comput. 24 (5) (1995) 948–969, MR1350753 (96f:65180).
[12] J.A. Grochow, Matrix Lie algebra isomorphism, in: IEEE Conference on Computational Complexity (CCC), Porto, June 2012, 2012, pp. 203–213.
[13] Hermann Heineken, Hans Liebeck, The occurrence of finite groups in the automorphism group of nilpotent groups of class 2, Arch. Math. (Basel) 25 (1974) 8–16, MR0349844 (50 #2337).
[14] Derek F. Holt, Bettina Eick, Eamonn A. O'Brien, A Handbook of Computational Group Theory, Discrete Math. Appl., Chapman & Hall/CRC, Boca Raton, FL, 2005, MR2129747 (2006f:20001).
[15] Derek F. Holt, Sarah Rees, Testing modules for irreducibility, J. Austral. Math. Soc. Ser. A 57 (1) (1994) 1–16, MR1279282 (95e:20023).
[16] Gábor Ivanyos, Fast randomized algorithms for the structure of matrix algebras over finite fields, in: Proceedings of the 2000 International Symposium on Symbolic and Algebraic Computation, St. Andrews, ACM, New York, 2000, pp. 175–183 (electronic), MR1805121.

[17] Gábor Ivanyos, Klaus Lux, Treating the exceptional cases of the MeatAxe, Experiment. Math. 9 (3) (2000) 373–381, MR1795309 (2001j:16067).
[18] Nathan Jacobson, Basic Algebra. I, W.H. Freeman and Co., San Francisco, CA, 1974, MR0356989 (50 #9457).
[19] Nathan Jacobson, Basic Algebra. II, W.H. Freeman and Co., San Francisco, CA, 1980, MR571884 (81g:00001).
[20] Johannes Köbler, Uwe Schöning, Jacobo Torán, The Graph Isomorphism Problem: Its Structural Complexity, Progress Theoret. Comput. Sci., Birkhäuser Boston, Inc., Boston, MA, 1993, MR1232421 (95b:05154).
[21] H.W. Lenstra Jr., Finding isomorphisms between finite fields, Math. Comp. 56 (193) (1991) 329–347, MR1052099 (91d:11151).
[22] Eugene M. Luks, Isomorphism of graphs of bounded valence can be tested in polynomial time, J. Comput. System Sci. 25 (1) (1982) 42–65, MR685360 (84a:68063).
[23] Klaus M. Lux, Magdolna Szőke, Computing homomorphism spaces between modules over finite dimensional algebras, Experiment. Math. 12 (1) (2003) 91–98, MR2002676.
[24] R.A. Parker, The computer calculation of modular characters (the Meat-Axe), in: Computational Group Theory, Durham, 1982, Academic Press, London, 1984, pp. 267–274, MR760660 (85k:20041).
[25] Nitin Saxena, Morphisms of rings and applications to complexity, PhD thesis, Indian Institute of Technology, Kanpur, 2006.
[26] Panagiotis C. Soules, Construction of finite p-groups with prescribed group of noncentral automorphisms, Rend. Semin. Mat. Univ. Padova 76 (1986) 75–88, MR881561 (88k:20048).
[27] Iuliana C. Teodorescu, The module isomorphism problems for finite rings and related results, preprint.
[28] Joachim von zur Gathen, Jürgen Gerhard, Modern Computer Algebra, 2nd ed., Cambridge University Press, Cambridge, 2003, MR2001757 (2004g:68202).