Computing tensor decompositions of finite matrix groups

Nurullah Ankaralioglu, Akos Seress

Discrete Mathematics and Theoretical Computer Science DMTCS vol. **13:4**, 2011, 5–14

Computing tensor decompositions of finite matrix groups

Nurullah Ankaralioglu[1†] and Ákos Seress[2,3‡]

[1]*Atatürk University, Faculty of Science, Department of Mathematics, Erzurum, Turkey*
[2]*School of Mathematics and Statistics, University of Western Australia, Crawley, Australia*
[3]*Department of Mathematics, The Ohio State University, Columbus, USA*

received 8th August 2011, revised 20th September 2011, accepted 21st September 2011.

We describe an algorithm to compute tensor decompositions of central products of groups. The novelty over previous algorithms is that in the case of matrix groups that are both tensor decomposable and imprimitive, the new algorithm more often outputs the more desirable tensor decomposition.

Keywords: matrix group recognition, tensor product decomposition

1 Background

A recent active area of computational group theory is the so-called *matrix group recognition project*. Let V be a finite dimensional vector space over a finite field $GF(q)$. Given $G = \langle X \rangle \leq GL(V)$, the goal is to compute quantitative and structural information about G such as the order, a composition series, and important characteristic subgroups like the largest solvable normal subgroup of G.

There are two main approaches to matrix group recognition. The *geometric approach*, initiated by Neumann and Praeger [20] and currently led by Leedham-Green and O'Brien [18], [26], is based on Aschbacher's classification of matrix groups [1]. Aschbacher defines nine categories of matrix groups G. In seven of these categories, there is a natural normal subgroup $N \lhd G$ that can be used to divide the recognition problem into two smaller subproblems on N and G/N. Based on that result, the geometric approach tries to find a homomorphism $\varphi : G \to H$ into an appropriate permutation or matrix group H, and recursively recognize $Im(\varphi)$ and $Ker(\varphi)$. In contrast, the *black-box group approach* of Babai and Beals [2] aims for the abstract group theoretic structure of G. Babai and Beals define a series of characteristic subgroups, present in all finite groups, and initiate a program that tries to compute a composition series going through these characteristic subgroups.

The original aim of the black-box group approach was the development of an algorithm with fast asymptotic running time. This aim was recently achieved in [3]. In that paper, a Las Vegas algorithm is described

†Email: `ankarali@atauni.edu.tr` research partially supported by the TÜBİTAK
‡Email: `akos@math.ohio-state.edu` research partially supported by the NSF and by ARC Grant DP1096525

that computes $|G|$ and a composition series, and sets up a data structure for membership testing in G. The running time is polynomial in the input length, plus the cost of using number theoretic oracles for discrete logarithm computation and for the factorization of integers, with the assumption that the composition factors of G are *nice*. “Nice” for a nonabelian simple group S means that S can be constructively recognized (i.e., an isomorphism with a natural copy of S can be built) in polynomial time, using number theory oracles. Currently, the “nice” category includes all sporadic, alternating, and classical groups, and most exceptional groups defined over small fields [6], [9], [10], [11], [12], [16]. We note that the use of number theory oracles seems to be unavoidable, since the cited cryptographic problems arise already in the case of 1×1 matrix groups.

In contrast, the primary goal of the geometric approach is the development of a practical algorithm for matrix group recognition. There is great success in this direction as well: a data structure to accommodate homomorphisms, and glueing the results of the subproblems in kernels and images of homomorphisms, was designed and implemented both in Magma [7] (by O’Brien) and in *GAP* [14] (by Neunhöffer and Seress [23], [24], [25]). Many mathematicians contributed algorithms and implementations for homomorphisms and for the handling of terminal nodes of the recursion tree, to be included in these frameworks.

Although the original aim of the geometric approach was the development of a practical algorithm and the guiding principle of the black-box group approach was a rigorous complexity analysis, recent developments combine the two aims very successfully. New subroutines with polynomial running time are designed and analyzed in the geometric approach, while methods and ideas from the black-box approach are utilized in practical algorithms.

The present paper is a contribution to the growing library of algorithms for the geometric approach, in the case of tensor product groups.

2 Tensor products

One of the Aschbacher classes (class $\mathbf{C4}$ in the notation of Kleidman and Liebeck [17], which is becoming the standard of the area) is the class of tensor products: the underlying vector space V of dimension mn can be written as a tensor product $V = U \otimes W$ of an n-dimensional space U and an m-dimensional space W, and $G \leq A \times B$ for some irreducible $A \leq GL(U)$ and $B \leq GL(W)$. We assume that A and B are nonabelian groups. The goal is to recognize the tensor structure of V and construct the homomorphisms $\varphi_U : G \to PGL(U)$ and $\varphi_W : G \to PGL(W)$. Note that the homomorphisms φ_U, φ_W are necessarily projective, because of the ambiguity of the tensor decomposition of scalar matrices.

A recent polynomial-time solution for this problem was announced by Ryba [27]. Given $G \leq GL(V)$, Ryba’s algorithm constructs the tensor decomposition of V if the decomposition exists, or returns `fail`. However, the algorithm works in vector spaces with dimension the square of the dimension of V, so in its current form it is not suitable for implementation.

In the special, but frequently occurring, case of $G = A \circ B$ with the central product taken over the scalar matrices, a more practical algorithm called SMASH was described by Holt et al [15]. The algorithm SMASH tries to construct some $g \in G$ which actually belongs to A or B, and computes the normal closure $C := \langle g^G \rangle$. Then a composition series for the C-module V is computed; if all composition factors are isomorphic then the tensor decomposition of V can be read from the embedding of composition factors in V and their identifications by the isomorphisms. Algorithms based on this idea are implemented both in Magma and *GAP*.

If, in addition, the group G also belongs to the Aschbacher category $\mathbf{C2}$ (case of imprimitive groups, which means that there exists a direct sum decomposition $V = V_1 \oplus V_2 \oplus \cdots \oplus V_k$ so that G permutes the subspaces V_i) then the algorithm described in the previous paragraph usually does not find the tensor decomposition of V. Rather, the composition factors fall into more than one isomorphism types, and the union of embeddings of isomorphic factors gives a subspace V_i in an imprimitivity decomposition. This outcome is sufficient in the recursive procedure of matrix recognition, because a homomorphism of G into the permutation group $Sym(k)$ can be computed. However, finding a tensor decomposition is a more valuable outcome, because it defines two homomorphisms so that the intersection of kernels fixes V pointwise (note that V may be a section of a larger vector space, occurring in the middle of a recursive procedure). In contrast, the kernel of the homomorphism into an imprimitivity decomposition acts on a sum of many invariant subspaces and finding the pointwise stabilizer of V may require much more additional work. Therefore, we set out the goal of designing and implementing an algorithm with the following properties.

(i) If the input group is nonabelian, belongs to category $\mathbf{C4}$, and $G = A \circ B$, then the algorithm constructs a tensor decomposition or imprimitivity decomposition of V.

(ii) If both tensor and imprimitivity decompositions exist then the algorithm has a good chance to find the tensor decomposition.

(iii) The algorithm succeeds and fails fast.

Goal (iii) may require some explanation. Of course, succeeding fast is the desirable outcome for algorithms. However, we should not forget that our algorithm is used in a big recursive scheme, for exploring inputs G where we have no idea whether G is in category $\mathbf{C4}$. In the case when G is not a $\mathbf{C4}$ group, the algorithm should fail fast, and not spend too much time for trying to construct the nonexisting tensor decomposition.

The novelty of our approach is the goal (ii). Although we cannot give a complexity analysis, and it is easy to construct tensor decomposable inputs on which the algorithm fails, the practical performance is quite satisfactory. We shall give some running times and frequency of success counts in Section 4.

The tensor product input groups known to us, where the algorithm fails completely, are all solvable. Fortunately, failing on that class of groups is not fatal for the overall success of matrix group recognition. Solvable groups can be treated efficiently by specialized algorithms both in theory by Luks [19] and in practice. Depending on the organization of subroutines in a composition tree computation, our algorithm may never encounter such an input.

3 The algorithm

The input is $G = \langle X \rangle = GL(V) \cong GL(d, q)$, and the output is one of the three possibilities below. The first of these outcomes is the most desirable, if the input group is indeed tensor decomposable.

(a) A factorization $d = mn$ of the input dimension and a base change matrix M such that group elements written in the new basis $\mathcal{B}$ are Kronecker products of an $n \times n$ matrix and an $m \times m$ matrix. The homomorphism $\varphi_U : G \to PGL(U)$ is defined by restricting the matrix $M(g)$ in $\mathcal{B}$ of some $g \in G$ to the entries in positions $\{(k_1 n + i, k_2 n + j) \mid 1 \le i, j \le n\}$. No matter which values $0 \le k_1, k_2 \le m - 1$ are chosen, the restriction matrices are scalar multiples of each other;

we choose particular values for k_1, k_2 so that the restriction matrix $M_1(g)$ does not consist of all 0 entries. We regard $M_1(g)$ modulo scalar matrices, to define a projective transformation. Similarly, $\varphi_W : G \to PGL(W)$ is defined by restricting $M(g)$ to the entries in positions $\{(in+k_1, jn+k_2) \mid 0 \le i, j \le m-1\}$ for some fixed k_1, k_2 chosen so that $1 \le k_1, k_2 \le n$ and the restriction matrix $M_2(g)$ does not consist of all 0 entries.

(b) A decomposition $V = V_1 \oplus V_2 \oplus \cdots \oplus V_k$ such that the subspaces V_i are permuted by G.

(c) The report `fail`.

The algorithm has three major parts. In the high-level description below, we assume that $G \le A \circ B$ and G is tensor decomposable; for other inputs, the algorithm will output `fail`.

Algorithm 3.1 (TENSORDECOMPOSE)

```
Input:  $G = \langle X \rangle = GL(V) \cong GL(d, q)$.
Step 1: Construct a list L containing some elements of G that are candidates to belong either to A or B.
Step 2: Choose a sublist $L_1$ of L, consisting of candidates all belonging to A, or all belonging to B.
        If $L_1$ is empty then output fail.
        Construct the normal closure $C := \langle L_1^G \rangle$.
Step 3: Check whether C defines a tensor decomposition or imprimitivity decomposition of V.
```

Now we give details of the three main steps.

3.1 Step 1.

For this step, we utilize two standard techniques. One of them is to take (pseudo)random elements $g \in G$, compute the projective order $|g|$, and for all prime divisors p of $|g|$, add $g^{|g|/p}$ to L. The idea is that for $g = ab$, with $a \in A$ and $b \in B$, there is a hope that different powers of p divide $|a|$ and $|b|$. In this case, $g^{|g|/p} \in A$ or $g^{|g|/p} \in B$. Some justification for the hope is given by the following theorem of Babai, Pálfy, and Saxl [4].

Theorem 3.2 *[4] If a simple group S is a quotient of a group $H \le GL(n, q)$ and p is an arbitrary prime then the proportion of elements of S of order* not *divisible by p is at least* $\min\{1/31, 1/(2n)\}$.

Hence, if both A and B have simple nonabelian quotients, or cyclic quotients of different size, then we have a fair chance that some prime will divide exactly one of $|a|, |b|$ for the decomposition of a random $g \in G$, $g = ab$.

The second method to generate elements of the list L is Neunhöffer's *involution jumper* (IJ) [21]. Although the IJ is one of the most powerful techniques to construct elements in proper normal subgroups, the method may not be widely known (yet) and so we give a pseudocode description. The pseudocode (including the jesting stopping condition) was copied from [22].

Algorithm 3.3 (InvolutionJumper)

```
Input: $G = \langle X \rangle$ and an involution $x \in G$.
    repeat
        $y :=$ random element of $G$
        $c := x^{-1}y^{-1}xy$ and $o := |c|$
        if $o$ is even then
            return $c^{o/2}$
        else
            $z := y \cdot c^{(o-1)/2}$ and $o' := |z|$
            if $o'$ is even then
                return $z^{o'/2}$
    until patience lost
    return fail
```

The group elements c, z constructed by the IJ are the outputs of Bray's algorithm [8] to construct elements of $C_G(x)$ and, if one of them happen to be of even order, then IJ outputs the involution that is the appropriate power of c or z. Neunhöffer [21] proved the following.

Theorem 3.4 *[21] The algorithm IJ defines a Markov chain on the conjugacy classes of involutions of G. This Markov chain is irreducible and aperiodic, and so in the stationary distribution every involution class has non-zero probability.*

In particular, if at least one of the groups A, B modulo scalars has even order then repeated applications of IJ may construct an element belonging to A or B. In the implementation, we add powers of 10 random elements to L, and the results of 10 applications of the involution jumper.

3.2 Step 2.

This step is the novelty of our approach. Previous algorithms, after assembling a list of candidates such that at least one of them belongs to some proper normal subgroup, try to combine them to *one* group element x that belongs to a proper normal subgroup. To this end, the best technique is Beals and Babai's *blind descent* [5] (see also [28, Ex. 2.7]). Applying blind descent (or any variant) to our list L, the output x may fall into a proper normal subgroup of A or B; in particular, if A or B is in the Aschbacher class **C2** then x may fix pointwise an imprimitivity decomposition of U or W and the tensor decomposition algorithm also outputs a less desirable imprimitivity decomposition of V. Therefore, we proceed the following way.

We define the *non-commuting graph* of the list L. This is a graph with vertex set L, and $g, h \in L$ are connected if and only if they do not commute.

Algorithm 3.5 (ProcessList)

```
Input: A list $L$ of elements from $G$ [candidates to be in $A$ or $B$].
    Compute the non-commuting graph $\Gamma$ of $L$.
    Remove vertices of valency at least $2|L|/3$.
    if no vertices remain then
        return fail
    else
        $\Delta$ :=graph on remaining vertices
        $L_1$ :=largest connected component of $\Delta$
        $C := \langle L_1^G \rangle$
        return $C$
```

If, for some $x, y \in L$, we have $x \in A$ and $y \in B$ then of course x and y commute. On the other hand, if x, y both have nontrivial projections on A (or B) then, with positive probability, x and y do not commute. Hence, by removing the vertices of large valency from Γ, we hope that we deleted all vertices of Γ that have nontrivial projections on both A and B, and the remaining group elements are in $A \cup B$. If indeed this happens then all vertices in a connected component are in A, or they are all in B, and the group C constructed by the algorithm satisfies $C \leq A$ or $C \leq B$. Moreover, we have $C \lhd G$.

The gain compared to previous approaches is that we have better chance for achieving $C = A$ or $C = B$. If we used L to construct only one element x in $A \cup B$ then it may have happened with larger probability that $C := \langle x^G \rangle$ is a proper subgroup of A or B.

Steps 1 and 2 can be executed in a "grey-box" setting. By that, we mean that with one exception, the algorithm in Steps 1 and 2 uses only multiplication, inversion, and comparison of group elements (i.e., black-box operations); the only additional requirement is an oracle to compute element orders.

Considering Steps 1 and 2 as a grey-box algorithm, the costliest operation is the construction of the non-commuting graph of L. However, this subtask can be sped up significantly by randomization, and exploiting that L contains elements from a matrix group. To check whether $x, y \in L$ commute, instead of the matrix multiplications xy and yx we computed and compared vxy and vyx, for two randomly chosen vectors from V. The justification follows from the following simple lemma.

Lemma 3.6 *Let $x, y \in GL(d, q)$. If x and y do not commute then, for a randomly chosen vector $v \in GF(q)^d$,*

$$\mathrm{Prob}(vxy \neq vyx) \geq 1 - \frac{1}{q}.$$

Proof: If x and y do not commute then the fixed point space F of $xyx^{-1}y^{-1}$ has dimension at most $d-1$. Hence $\mathrm{Prob}(vxy = vyx) = \mathrm{Prob}(v \in F) \leq q^{d-1}/q^d = 1/q$. □

3.3 Step 3.

For Step 3, we simply use the subroutine `RECOG.SortOutReducibleNormalSubgroup` from the *GAP* package `recog`, written by Max Neunhöffer, and described in [21]. That program computes a composition series of the C-module V, and checks whether the factor modules are isomorphic. If they are then we obtain the output (a) described at the beginning of Section 3; if there are more than one isomorphism classes of C-modules then we obtain output (b); finally, if C acts irreducibly on V then we report `fail`.

4 Some runtimes

We ran both the new program and `RECOG.FindHomMethodsProjective.D247` at least 100 times on each example. We report the percentage of the three possible outcomes (tensor decomposition **C4**, imprimitivity decomposition **C2**, and `fail`), and the average running time in seconds.

In general, the algorithm in the `recog` package runs faster. In the cases when the reported running times are roughly the same, the average for `recog` was increased significantly by some "unlucky" cases, or because `recog` tries harder for a positive outcome. In such cases, the new program may more readily report failure. As advertised, the advantage of the new approach is that in some cases it may construct a tensor decomposition with higher probability than the `recog` package.

Tab. 1: $G = GL(10,7) \circ GL(10,7) \leq GL(100,7)$

	new	**C4**	**C2**	`fail`	`recog`	**C4**	**C2**	`fail`
frequency		100%	0%	0%		100%	0%	0%
time		2.3s	–	–		0.4s	–	–

Tab. 2: $G = (GL(2,7) \wr C_7) \circ (GL(2,7) \wr S_5) \leq GL(140,7)$

	new	**C4**	**C2**	`fail`	`recog`	**C4**	**C2**	`fail`
frequency		57.9%	35.8%	6.3%		2.5%	97.5%	0%
time		0.4s	0.4s	02.s		0.2s	0.4s	–

Tab. 3: $G = (GL(2,7) \wr C_7) \circ (GL(2,7) \wr C_7) \leq GL(196,7)$

	new	**C4**	**C2**	`fail`	`recog`	**C4**	**C2**	`fail`
frequency		7.1%	52.1%	40.8%		0%	100%	0%
time		0.6s	0.6s	02.s		–	0.6s	–

Tab. 4: $G = GL(100,7)$

	new	**C4**	**C2**	`fail`	`recog`	**C4**	**C2**	`fail`
frequency		0%	0%	100%		0%	0%	100%
time		–	–	5.1s		–	–	5.0s

Acknowledgements

We are indebted to Max Neunhöffer both for theoretical advice and for his help with the implementation of the algorithm.

References

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

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

[3] L. Babai, R. Beals, Á. Seress, Polynomial-time theory of matrix groups, *Proc. 41st ACM Symp. on Theory of Computing*, pp. 55–64, 2009.

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

[5] R. Beals, L. Babai, Las Vegas algorithms for matrix groups, *34th Annual Symposium on Foundations of Computer Science (Palo Alto, CA, 1993)*, 427–436, *IEEE Comput. Soc. Press, Los Alamitos, CA,* 1993.

[6] R. Beals, C. Leedham-Green, A. Niemeyer, C. Praeger, Á. Seress, A black-box group algorithm for recognizing finite symmetric and alternating groups, I, *Trans. Amer. Math. Soc.*, 355:2097–2113, 2003.

[7] W. Bosma and J.J. Cannon, *Handbook of* MAGMA *functions*, School of Mathematics and Statistics, University of Sydney, Sydney, 1995.

[8] J. Bray, An improved method for generating the centralizer of an involution, *Arch. Math. (Basel)* 74:241–245, 2000.

[9] P. A. Brooksbank, Fast constructive recognition of black-box unitary groups, *LMS J. Comput. Math.*, 6:162–197, 2003.

[10] P. A. Brooksbank, Fast constructive recognition of black box symplectic groups, *J. Algebra*, 320:885–909, 2008.

[11] P. A. Brooksbank, W. M. Kantor, On constructive recognition of a black box $\mathrm{PSL}(d, q)$, in: Groups and Computation, III (Columbus, OH, 1999), 95–111, Ohio State Univ. Math. Res. Inst. Publ., 8, de Gruyter, Berlin, 2001.

[12] P. A. Brooksbank, W. M. Kantor, Fast constructive recognition of black box orthogonal groups, *J. Algebra*, 300:256–288, 2006.

[13] J. Conway, R. Curtis, S. Norton, R. Parker, and R. Wilson, *Atlas of Finite Groups*, Oxford University Press, 1985.

[14] The GAP Group, *GAP – Groups, Algorithms and Programming*, Version 4.4, 2004.

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

[16] W. M. Kantor, K. Magaard, Black-box exceptional groups, submitted.

[17] P. B. Kleidman and M. W. Liebeck, *The Subgroup Structure of the Finite Classical Groups*, London Math. Soc. Lecture Note Series, vol. 129, Cambridge University Press, 1990.

[18] C. R. Leedham-Green, The computational matrix group project, in: Groups and Computation, III (Columbus, OH, 1999), 229–247, Ohio State Univ. Math. Res. Inst. Publ., 8, de Gruyter, Berlin, 2001.

[19] E. M. Luks, Computing in solvable matrix groups, *Proc. 33rd IEEE Symp. Found. Comp. Sci.*, pp. 111–120, 1992.

[20] P. M. Neumann, C. E. Praeger, A recognition algorithm for special linear groups, *Proc. London Math. Soc.*, 65:555–603, 1992.

[21] M. Neunhöffer, *Constructive Recognition of Finite Groups*, Habilitationsschrift, 2009.

[22] M. Neunhöffer, Talk at the Univ. Birmingham, turnbull.mcs.st-and.ac.uk/˜neunhoef/Publications/pdf/birmingham09.pdf, 2009.

[23] M. Neunhöffer, Á. Seress, A data structure for a uniform approach to computations with finite groups, *Proc. International Symposium on Symbolic and Algebraic Computation (ISSAC '06)*, 254–261, 2006.

[24] M. Neunhöffer, Á. Seress, *GAP* Package "recogbase", http://www-groups.mcs.st-and.ac.uk/˜neunhoef/Computer/Software/Gap/recogbase.html.

[25] M. Neunhöffer, Á. Seress, *GAP* Package "recog", http://www-groups.mcs.st-and.ac.uk/˜neunhoef/Computer/Software/Gap/recog.html.

[26] E. A. O'Brien, Algorithms for matrix groups, in: Groups St. Andrews 2009 in Bath, II, 297–323, London Math. Soc. Lecture Note Ser., 260, Cambridge Univ. Press, Cambridge, 2011.

[27] A. J. E. Ryba, Personal communication at the Oberwolfach workshop on Computational Group Theory, August 2011.

[28] Á. Seress, *Permutation Group Algorithms*, Cambridge Tracts in Mathematics, vol. 152, Cambridge Univ. Press, Cambridge, 2003.

[29] R.A. Wilson et al., *A World-Wide-Web Atlas of finite group representations*, `http://brauer.maths.qmul.ac.uk/Atlas/v3/`.